การสร้างรูปแบบพฤติกรรมสำหรับศัตรูในเกมแอ็คชั่นสองมิติแบบเน้นตัวละครโดยอัตโนมัติด้วยเทคนิคเหมืองข้อมูล

นายพิชิต พร้อมสุทธิพงศ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2559

Automatic enemy behavior pattern generation for 2D avatar-based action game using data mining technique

Mr. Pichit Promsutipong

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2016

หัวข้อวิทยานิพนธ์	การสร้างรูปแบบพฤติกรรมสำหรับศัตรูในเกมแอ็คชั่นสองมิติแบบเน้นตัวละครโดยอัตโนมัติด้วยเทคนิคเหมืองข้อมูล
โดย	นายพิชิต พร้อมสุทธิพงศ์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร.วิษณุ โคตรจรัส

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.นัทที นิภานันท์)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(ผู้ช่วยศาสตราจารย์ ดร.วิษณุ โคตรจรัส)

.....กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.ณัฐพงศ์ ชินธเนศ)

.....กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร.จรัสศรี ฐ่งรัตนาอุบล)

พิชิต พร้อมสุทธิพงศ์ : การสร้างรูปแบบพฤติกรรมสำหรับศัตรูในเกมแอ็คชั่นสองมิติแบบเน้นตัวละครโดยอัตโนมัติด้วยเทคนิคเหมืองข้อมูล (Automatic enemy behavior pattern generation for 2D avatar-based action game using data mining technique) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร.วิษณุ โคตรจรัส, 192 หน้า.

ศัตรูเป็นหนึ่งในองค์ประกอบที่สร้างความท้าทายให้กับเกมแอ็คชั่นแบบเน้นตัวละคร โดยความท้าทายนั้นมาจากรูปแบบพฤติกรรมที่แตกต่างกันไปในศัตรูแต่ละตัวทำให้ผู้เล่นต้องเรียนรู้วิธีการรับมือกับศัตรูต่าง ๆ การออกแบบพฤติกรรมเหล่านี้ให้ออกมาดีเพียงพอที่จะนำไปใช้ในเกมต้องใช้เวลาอีกทั้งอาจมีรูปแบบพฤติกรรมที่นักออกแบบคาดไม่ถึงได้ การสร้างรูปแบบพฤติกรรมโดยอัตโนมัติจึงน่าจะช่วยบรรเทาปัญหาเหล่านี้ได้ งานวิทยานิพนธ์นี้จึงเริ่มต้นวิจัยงานด้านนี้โดยนำเสนอขั้นตอนวิธีสำหรับสร้างรูปแบบพฤติกรรมที่ยอมรับได้โดยอัตโนมัติ

งานวิทยานิพนธ์นี้พยายามหารูปแบบความสัมพันธ์ระหว่างพฤติกรรมย่อยที่น่าจะเป็นรูปแบบความสัมพันธ์ที่ทำให้ศัตรูเป็นที่ยอมรับได้ และใช้ความสัมพันธ์ที่พบเหล่านั้นในการสร้างรูปแบบพฤติกรรมของศัตรูขึ้นมา รูปแบบความสัมพันธ์ที่ใช้จะได้จากการทำเหมืองข้อมูลแบบต่าง ๆ บนชุดข้อมูลศัตรู ซึ่งเป็นข้อมูลรูปแบบพฤติกรรมของศัตรูที่อยู่ในรูปของภาษาอธิบายภาคีที่งานวิทยานิพนธ์นี้นำเสนอ พฤติกรรรมของศัตรูที่ใช้จะนำมาจากศัตรูในเกมแอ็คชั่นสองมิติแบบเน้นตัวละครที่ได้รับความนิยม รูปแบบพฤติกรรมที่สร้างขึ้นจะถูกปรับแต่งให้กลายเป็นภาคีศัตรู แล้วจึงนำไปวัดการยอมรับได้ตามเกณฑ์ที่กำหนดไว้ในงานวิทยานิพนธ์นี้โดยใช้ปัญญาประดิษฐ์วัดผลแทนผู้เล่นมนุษย์ ผลการทดลองเบื้องต้นแสดงให้เห็นว่ารูปแบบพฤติกรรมที่สร้างขึ้นยอมรับได้เป็นจำนวนไม่มากนัก แต่เมื่อทำการปรับปรุงผลลัพธ์แล้ว ผู้วิจัยพบว่ามีจำนวนรูปแบบพฤติกรรมที่ยอมรับได้เป็นอัตราส่วนที่มีประสิทธิภาพเพียงพอต่อการนำไปใช้งาน

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2559

5670308821 : MAJOR COMPUTER ENGINEERING
KEYWORDS: ENEMY MODEL / PATTERN GENERATION / ARTIFICIAL INTELLIGENT / ACTION GAME / DESCRIPTION LANGUAGE / PATTERN MINING

PICHIT PROMSUTIPONG: Automatic enemy behavior pattern generation for 2D avatar-based action game using data mining technique. ADVISOR: ASST. PROF. VISHNU KOTRAJARAS, 192 pp.

Enemy is one of the challenges in avatar-based action games. The challenges come from enemies' behavior patterns that player has to overcome. These patterns must be carefully designed so that they are acceptable to be used in games. The designing process is time consuming. Automatic generation of behavior could be used to alleviate burden on designers and help exploring emergent behavior that designers may not think of. Therefore, as a first step, this thesis proposes an algorithm for generating acceptable enemy behavior.

To generate behavior patterns, we mine behavior dataset for relation between atomic behavior defined by our agent description language and assemble found behaviors together to form behavior patterns. The patterns will then be made into agents. Their acceptability are verified by using evaluation AI. The initial results show that acceptable patterns count are not high. However, after we improve the patterns, we achieve higher acceptable patterns count. The number is considered good enough for actual usage.

Department: Computer Engineering Student's Signature .....
Field of Study: Computer Engineering Advisor's Signature .....
Academic Year: 2016

กิตติกรรมประกาศ

ขอขอบพระคุณ ผศ.ดร. วิษณุ โคตรจรัส ที่คอยให้คำปรึกษาและตรวจทานแก้ไขทั้งงานตีพิมพ์และวิทยานิพนธ์จนผลงานทั้งหมดเสร็จสิ้นและได้รับการเผยแพร่ด้วยดี

ขอขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ที่ช่วยพิจารณาวิทยานิพนธ์ชิ้นนี้และแนะนำแนวทางให้ผลงานฉบับนี้สมบูรณ์ยิ่งขึ้น

ขอขอบคุณสมาชิกแล็ปวิจัยเกมที่ช่วยเสนอแนะแนวทางที่เป็นประโยชน์ต่องานชิ้นนี้และขอขอบคุณผู้ร่วมทดลองเล่นเกมทุกท่าน

ท้ายที่สุดนี้ ขอขอบพระคุณบิดา มารดา และน้อง ๆ ที่คอยอยู่เคียงข้าง ให้ความสนับสนุน คอยเป็นกำลังใจ และให้ความช่วยเหลือในทุกด้านตลอดมา

สารบัญ

หน้า

หน้า

หน้า

หน้า

หน้า

หน้า

สารบัญตาราง

สารบัญภาพ

สารบัญขั้นตอนวิธี

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของปัญหา

อุตสาหกรรมเกมเป็นหนึ่งในอุตสาหกรรมสื่อที่มีอัตราการเติบโตสูงเป็นอันดับต้นของสื่อทั้งหมด รายงานการคาดการณ์เกี่ยวกับอุตสาหกรรมสื่อของ PwC [1] แสดงให้เห็นว่าอัตราการเติบโตของยอดขายของธุรกิจเกมสูงเป็นอันดับสาม ข้อมูลดังกล่าวนี้สะท้อนให้เห็นถึงความสำคัญในการผลักดันให้เกิดวิธีการหรือแนวคิดใหม่ที่ช่วยให้อุตสาหกรรมเกมสามารถพัฒนาต่อไปได้

ในการผลิตเกมนั้นจำเป็นต้องอาศัยเงินทุนและทรัพยากรบุคลากรจำนวนมากเพื่อที่จะผลิตเกมที่มีคุณภาพที่เป็นที่ยอมรับได้ในตลาดภายใต้กรอบเวลาที่กำหนด ประเด็นดังกล่าวนี้อาจไม่เป็นปัญหาสำหรับบริษัทเกมที่มีขนาดใหญ่ แต่ในยุคปัจจุบันนี้ ผู้พัฒนามีช่องทางการจัดจำหน่ายเกมที่เข้าถึงได้ง่ายขึ้นกว่าเดิม ทำให้เกิดผู้พัฒนารายย่อยขึ้นเป็นจำนวนมากและเกิดการแข่งขันกันภายในตลาดเกมอย่างรุนแรง เนื่องจากผู้พัฒนารายย่อยเหล่านี้มีทรัพยากรบุคคลและเงินทุนจำกัด เมื่อรวมเข้ากับสภาพการแข่งขันของตลาดเกมในยุคปัจจุบันที่ทำให้ผู้พัฒนาจำเป็นต้องผลักดันผลงานของตัวเองออกมาอย่างรวดเร็ว เทคโนโลยีหรือวิธีการที่จะช่วยลดทอนทรัพยากร เงินทุน และเวลาในการผลิตเกมจึงเป็นตัวช่วยที่สำคัญต่อกลุ่มผู้พัฒนารายย่อยเหล่านี้

การที่เกมหนึ่ง ๆ จะออกสู่ตลาดได้ จำเป็นต้องผ่านกระบวนการมากมายตั้งแต่ขั้นตอนออกแบบและพัฒนาโปรแกรม ไปจนถึงการทำตลาดและเผยแพร่ ขั้นตอนการออกแบบเกมและสร้างเนื้อหา (content) นับเป็นหนึ่งในขั้นตอนที่สำคัญที่สุดในการผลิตเกม นักออกแบบเกมจำเป็นต้องใช้ความคิดสร้างสรรค์อย่างมากในการสร้างเนื้อหาของเกม ซึ่งในบางครั้ง กว่าจินตนาการของนักออกแบบจะเข้าถึงคำตอบของการออกแบบที่เหมาะสมต้องใช้เวลานาน แนวคิดของการสร้างเนื้อหาแบบอัตโนมัติ (Procedural content generation -- PCG) จึงเกิดขึ้นเพื่อช่วยลดปัญหาดังกล่าวนี้ [2]

เกมแอ็คชั่นเป็นเกมที่ผู้เล่นจะต้องอาศัยปฏิกิริยาตอบสนองและความแม่นยำในการโต้ตอบกับสภาพแวดล้อมของตัวเกมภายใต้กฎที่เกมกำหนดไว้ และเมื่อเจาะจงลงไปเป็นเกมแอ็คชั่นแบบเน้นตัวละคร ผู้เล่นจะต้องตอบสนองกับสภาพแวดล้อมของตัวเกมผ่านการควบคุมตัวละครอวตาร และหนึ่งในสภาพแวดล้อมที่สำคัญของตัวเกมประเภทนี้คือศัตรู

ลักษณะของศัตรูในเกมประเภทนี้แบ่งเป็นศัตรูแบบฉลาดและศัตรูที่มีรูปแบบการเคลื่อนไหวตายตัว ศัตรูแบบฉลาดจะถูกควบคุมโดยปัญญาประดิษฐ์ที่มีความซับซ้อน มีความสามารถในการตัดสินใจเลือกใช้พฤติกรรมตามแต่สถานการณ์ปัจจุบัน เกมที่ใช้ศัตรูประเภทนี้จะเน้นไปที่การต่อสู้ระหว่างตัวละครจำนวนน้อย ในทางตรงกันข้าม ศัตรูที่มีการเคลื่อนไหวตายตัวจะถูกใช้ในเกมที่เน้นไปที่การต่อสู้ระหว่างอวตารผู้เล่นและศัตรูจำนวนมาก ผู้เล่นจะต้องทำความเข้าใจรูปแบบพฤติกรรมของศัตรูที่นักออกแบบกำหนดเอาไว้และเอาชนะให้ได้ ศัตรูจำพวกนี้จะต้องถูกออกแบบมาอย่างดีเพื่อให้พฤติกรรมมีความท้าทายพอที่ผู้เล่นจะยอมรับได้ กระบวนการออกแบบนี้นอกจากจะต้องใช้เวลาแล้ว อาจจะมีรูปแบบพฤติกรรมบางอย่างที่นักออกแบบอาจจะคาดไม่ถึงได้ การนำเอา PCG เข้ามาช่วยในการออกแบบพฤติกรรมของศัตรูจึงช่วยลดภาระของนักออกแบบลงได้

งานวิทยานิพนธ์นี้จึงมีวัตถุประสงค์เพื่อค้นหาวิธีการสำหรับสร้างพฤติกรรมของศัตรูที่มีรูปแบบพฤติกรรมตายตัวแบบอัตโนมัติสำหรับเกมแอ็คชั่นสองมิติแบบเน้นตัวละคร โดยมีเงื่อนไขว่า พฤติกรรมของศัตรูที่ถูกสร้างขึ้นมาจะต้องทำให้ศัตรูนั้นสามารถนำไปใช้ในเกมได้และเป็นที่ยอมรับของผู้เล่น รวมไปถึงสามารถทำความเข้าใจได้โดยมนุษย์เพื่อที่นักออกแบบจะสามารถนำผลลัพธ์ไปปรับปรุงต่อได้ในกรณีที่ต้องการ

งานวิจัยที่เกี่ยวข้องกับพฤติกรรมศัตรูเท่าที่มีอยู่ในปัจจุบันจะเน้นไปที่ศัตรูแบบฉลาด ผู้วิจัยยังไม่พบงานวิจัยที่เกี่ยวข้องกับศัตรูที่มีรูปแบบพฤติกรรมตายตัว งานวิจัยนี้จึงต้องเริ่มต้นจากการกำหนดนิยามของศัตรูและพฤติกรรมในบริบทของเกมแอ็คชั่นสองมิติแบบเน้นตัวละครโดยสร้างแบบจำลองในรูปของภาษาอธิบายภาคี แล้วจึงพัฒนาขั้นตอนวิธีสร้างรูปแบบพฤติกรรมจากแบบจำลองที่ทำให้รูปแบบพฤติกรรมดังกล่าวยอมรับได้โดยผู้เล่น ผลลัพธ์ที่ได้จากขั้นตอนวิธีจะอยู่ในรูปของภาษาอธิบายภาคีเพื่อให้มนุษย์สามารถทำความเข้าใจได้ อนึ่งการสร้างพฤติกรรมของศัตรูสำหรับงานนี้จะไม่นำลักษณะของพื้นที่ในฉากของเกมเข้ามาพิจารณาร่วม

1.2 วัตถุประสงค์ของการวิจัย

เพื่อหาแนวทางการสร้างรูปแบบพฤติกรรมสำหรับศัตรูในเกมแอ็คชั่นสองมิติเน้นตัวละครโดยอัตโนมัติที่ยอมรับได้ โดยวัดการยอมรับได้จากประสิทธิภาพการต่อสู้ระหว่างศัตรูกับผู้เล่น (ในงานวิทยานิพนธ์นี้ใช้ปัญญาประดิษฐ์ที่ผ่านการทดสอบยืนยันว่าใช้แทนผู้เล่นได้ มาทำการทดสอบเกมแทนผู้เล่นจริง)

1.3 ขอบเขตของการวิจัย

1) การสร้างรูปแบบพฤติกรรมจะไม่นำลักษณะของพื้นที่ในฉากของเกมเข้ามาพิจารณา
2) ชุดข้อมูลรูปแบบพฤติกรรมทำซ้ำรูปแบบพฤติกรรมของศัตรูบางส่วนจากเกม Metroid, Super C, Megaman, Megaman 4 และ Shovel Knight
3) เกมทดสอบที่ใช้จะเป็นเกมแนวแพลทฟอร์ม ซึ่งเป็นประเภทย่อยหนึ่งของเกมแอ็คชั่น
4) รูปร่างของศัตรูที่ปรากฏในเกมทดสอบจะถูกกำหนดโดยมนุษย์และไม่เกี่ยวข้องกับการสร้างรูปแบบพฤติกรรม

1.4 ประโยชน์ที่ได้รับ

1) ได้แบบจำลองภาคีสำหรับอธิบายภาคีในเกมแอ็คชั่นสองมิติแบบเน้นตัวละคร
2) ได้ขั้นตอนวิธีสำหรับสร้างรูปแบบพฤติกรรมศัตรูอัตโนมัติสำหรับเกมแอ็คชั่นสองมิติแบบเน้นตัวละคร

1.5 ขั้นตอนการดำเนินงาน

1) ศึกษาทฤษฎีและงานวิจัยที่เกี่ยวข้อง
2) วิเคราะห์เกมแอ็คชั่นและศัตรูภายในเกมเพื่อจัดทำแบบจำลองภาคี
3) พัฒนาตัวแปลภาษาอธิบายภาคีและเกมทดสอบ พร้อมรวมตัวแปลภาษาเข้ากับเกมทดสอบ

4) จัดเตรียมและปรับปรุงขั้นตอนวิธีสำหรับการเตรียมข้อมูลขาเข้าและขาออกของเหมืองข้อมูล รวมถึงปรับปรุงขั้นตอนวิธีสำหรับทำเหมืองข้อมูล
5) ทดสอบความถูกต้องของขั้นตอนวิธีทำเหมืองข้อมูลรวมถึงความถูกต้องของข้อมูลขาเข้าและขาออกของการทำเหมืองข้อมูล
6) สร้างชุดข้อมูลรูปแบบพฤติกรรม
7) พัฒนาขั้นตอนวิธีสำหรับสร้างรูปแบบพฤติกรรมของศัตรูโดยใช้ข้อมูลความสัมพันธ์ที่ได้จากการทำเหมืองข้อมูล
8) เก็บข้อมูลการเล่นของผู้เล่นและพัฒนาปัญญาประดิษฐ์สำหรับวัดผลแทนผู้เล่นโดยใช้ข้อมูลที่ได้
9) ทดลองสร้างรูปแบบพฤติกรรมด้วยขั้นตอนวิธีที่นำเสนอ และวัดผลด้วยปัญญาประดิษฐ์
10) วิเคราะห์ผลการทดลอง
11) สรุปผลการวิจัยและจัดทำวิทยานิพนธ์

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 เกมแอ็คชั่นแบบเน้นตัวละคร (Avatar-based action game)

Ernest Adams [3] ให้นิยามของเกมประเภทแอ็คชั่นไว้ว่าเป็นเกมที่ผู้เล่นจะต้องใช้ความสามารถทางกายภาพในการเล่น ผู้เล่นต้องมีความรวดเร็วในการตัดสินใจโต้ตอบกับตัวเกมรวมไปถึงต้องมีความแม่นยำในการโต้ตอบ นอกจากนี้ยังต้องอาศัยการประสานงานระหว่างตาและมือ (Hand-eye coordination) ในการเล่น เช่น ผู้เล่นต้องสังเกตสภาพแวดล้อมในเกมตลอดเวลาพร้อมโต้ตอบกับสภาพแวดล้อมโดยการป้อนข้อมูลใส่ตัวเกมผ่านส่วนต่อประสานต่าง ๆ เช่น แผงแป้นอักขระ ก้านควบคุม เป็นต้น หากผู้เล่นไม่สามารถโต้ตอบกับสภาพแวดล้อมของเกมได้ถูกต้อง ทันเวลาหรือแม่นยำเพียงพอ อาจจะเกิดการลงโทษ (penalty) บางอย่าง

เกมแอ็คชั่นแบ่งเป็นประเภทย่อยได้หลายประเภท การแบ่งประเภทย่อยยังไม่มีกฎเกณฑ์ที่ตายตัว อาจขึ้นอยู่กับมุมมองของผู้เล่นหรือกลุ่มสังคมผู้เล่นได้ อีกทั้งหลาย ๆ เกมในท้องตลาดก็มีการควบรวมเอารูปแบบการเล่นและกลไกของเกม (Game mechanic) หลายประเภทรวมไว้ในเกมเดียวกันได้ ทำให้เกิดความแตกต่างในการแบ่งประเภทของเกมออกไปอีก สำหรับในแวดวงการศึกษาด้านเกม ตำราที่เกี่ยวข้องกับการออกแบบเกมจะมีการกล่าวถึงประเภทย่อยของเกมแอ็คชั่นเช่นเดียวกัน ได้แก่

- เกมยิงปืน (Shooter) ซึ่งบางตำราอาจจะมีการแยกเกมยิงปืนประเภทมุมมองบุคคลที่หนึ่ง (First person shooter -- FPS) หรือเกมยานยิง (Shoot 'em up) ออกมาเป็นประเภทย่อยต่างหากดังเช่นในหนังสือของ Scott Roogers [4]
- เกมแพลทฟอร์ม (Platformer)
- เกมต่อสู้ (Fighting)
- เกมแอ็คชั่นผจญภัย (Action-Adventure)
- เกมดนตรี (Rhythm)
- เกมปริศนาแบบเร็ว (Fast Puzzle)

เกมแอ็คชั่นแบบเน้นตัวละครไม่ใช่ประเภทย่อยใด ๆ ของเกมแอ็คชั่น แต่เป็นลักษณะของเกมที่ผู้เล่นจะต้องควบคุมตัวละครอวตาร (Avatar) และตอบโต้กับสภาพแวดล้อมของเกมผ่านตัวละครอวตารดังกล่าวภายใต้กลไกที่กำหนดเพื่อนำพาตัวละครอวตารไปสู่จุดมุ่งหมาย (Goal) ของเกมให้ได้ หลายประเภทย่อยของเกมแอ็คชั่นมีลักษณะของเกมแอ็คชั่นแบบเน้นตัวละคร เช่น เกมยิงปืน เกมแพลทฟอร์ม เกมต่อสู้ และเกมแอ็คชั่นผจญภัย

Mark Davies [5] แจกแจงรายละเอียดของกลไกที่สำคัญสำหรับเกมลักษณะนี้ไว้ ซึ่งสามารถสรุปได้เป็น 3 กลไกใหญ่ ได้แก่

- กลไกการเคลื่อนไหวของตัวละครอวตาร: ระบุว่าตัวละครอวตารสามารถทำสิ่งใดได้บ้างภายในโลกของเกม
- กลไกการควบคุมกล้องติดตามตัวละคร: ระบุวิธีการแสดงภาพของโลกของเกมต่อผู้เล่น
- กลไกการต่อสู้: ระบุวิธีการต่อสู้ระหว่างตัวละครภายในเกม

กลไกการต่อสู้ประกอบด้วยกลไกค่าคุณสมบัติของตัวละคร วิธีการต่อสู้ระหว่างตัวละคร และความเกี่ยวพันระหว่างการต่อสู้และค่าพลัง ตัวละครที่มีบทบาทสำคัญในกลไกคือศัตรู ซึ่งมีหน้าที่ในการขัดขวางไม่ให้ตัวละครอวตารสามารถเดินหน้าไปต่อได้ ผู้เล่นจะต้องหลบหลีกหรือปราบศัตรูเพื่อเดินหน้าต่อไป ศัตรูจึงนับเป็นความท้าทายหลักของเกมแอ็คชั่นแบบเน้นตัวละครได้

พฤติกรรมของศัตรูในเกมแอ็คชั่นแบบเน้นตัวละครจะแตกต่างกันไปตามแต่ประเภทย่อยของเกมนั้น ในเกมต่อสู้หรือเกมยิงปืนที่เน้นการต่อสู้ระหว่างตัวละครจำนวนน้อย ตัวละครหนึ่ง ๆ จะมีความสามารถและการกระทำหลายรูปแบบ ความท้าทายของการต่อสู้กับศัตรูในบริบทนี้จะเกิดจากความสามารถของตัวศัตรูซึ่งถูกควบคุมโดยปัญญาประดิษฐ์ที่ถูกปรับแต่งให้มีความฉลาดเพียงพอที่จะรับมือผู้เล่นได้ในระดับที่กำหนด สามารถเลือกการกระทำได้ถูกต้องและเหมาะสมตามสถานการณ์ พฤติกรรมของศัตรูในบริบทเกมนี้จึงไม่มีรูปแบบที่แน่นอนหรือมีรูปแบบที่แน่นอนแต่มีความซับซ้อนมากจนไม่สามารถคาดเดาได้โดยง่าย

ในอีกประเภทย่อยหนึ่ง เช่น เกมแพลทฟอร์ม เกมแอ็คชั่นผจญภัย หรือเกมยานยิง ศัตรูที่ผู้เล่นพบจะมีรูปแบบพฤติกรรมที่ตายตัว ความท้าทายของศัตรูในเกมประเภทนี้มาจากการที่ผู้เล่นต้องต่อสู้กับศัตรูที่มีพลังเหนือกว่ามากหรือต่อสู้กับศัตรูหลายตัวพร้อมกัน ทำให้ผู้เล่นต้องเรียนรู้รูปแบบและความสามารถที่ตายตัวของศัตรูเหล่านั้นทั้งหมดให้ได้อย่างรวดเร็วเพื่อที่จะสามารถตัดสินใจว่าในสถานการณ์หนึ่ง ๆ จะทำอย่างไร ควรจะต่อสู้หรือหลีกเลี่ยงการต่อสู้ดังกล่าว หากจะต่อสู้จะต้องใช้วิธีใดจึงจะปราบศัตรูลงได้ หากจะหลีกเลี่ยง ทำอย่างไรจึงจะเสียพลังชีวิตน้อยที่สุด

2.1.2 การทำเหมืองข้อมูล (Data mining)

การทำเหมืองข้อมูล หมายถึง “การสกัดรูปแบบข้อมูล (Data pattern) ที่สนใจออกมาจากข้อมูลปริมาณมาก” [6] โดยทั่วไปจะมีขั้นตอนที่สำคัญได้แก่ การประมวลข้อมูลก่อน (Data preprocessing) การทำเหมืองข้อมูล (Data mining) การวัดผลรูปแบบผลลัพธ์ (Pattern evaluation) และการนำเสนอความรู้ (Knowledge presentation)

การประมวลข้อมูลก่อนเป็นการนำเอาข้อมูลที่อยู่ภายในฐานข้อมูลมาประมวลผลเพื่อให้พร้อมสำหรับการทำเหมืองข้อมูล ซึ่งประกอบด้วยการทำความสะอาดข้อมูล (Data cleaning) เพื่อลดทอนสัญญาณรบกวน (Noise) การเลือกข้อมูลที่เกี่ยวข้อง และการแปลงข้อมูล (Data transformation) เพื่อให้เหมาะสมกับการทำเหมืองข้อมูล

ขั้นตอนการทำเหมืองข้อมูลเป็นการนำเข้าข้อมูลที่ผ่านการประมวลข้อมูลก่อนแล้วมาสกัดหารูปแบบข้อมูลที่สนใจ ขั้นตอนวิธีที่ใช้สำหรับการสกัดข้อมูลมีมากมายหลายแบบ แต่ละขั้นตอนวิธีจะถูกออกแบบเพื่อสกัดข้อมูลจากฐานข้อมูลที่มีแบบ (Format) เฉพาะต่าง ๆ เช่น ฐานข้อมูลลำดับ (Sequence database) ฐานข้อมูลกราฟ (Graph database) ฐานข้อมูลข้อความ (Text database) เป็นต้น รูปแบบที่ได้จากการทำเหมืองข้อมูลจะถูกวัดความน่าสนใจ (Interestingness) เพื่อตัดสินว่ารูปแบบที่ได้เป็นข้อมูลที่นับว่าเป็นความรู้ (Knowledge) หรือไม่

งานวิทยานิพนธ์นี้ทำเหมืองข้อมูลบนชุดข้อมูล (Dataset) รูปแบบพฤติกรรมของศัตรู โดยมีการแปลงข้อมูลให้อยู่ในรูปของลำดับและกราฟเพื่อสกัดข้อมูลที่น่าสนใจจากหลายมุมมอง ในหัวข้อนี้จึงนำเสนอนิยามของการทำเหมืองข้อมูลสำหรับข้อมูลลำดับและกราฟ

2.1.2.1 การทำเหมืองข้อมูลสำหรับข้อมูลลำดับ (Sequential pattern mining)

การทำเหมืองข้อมูลสำหรับข้อมูลลำดับ คือ การสกัดลำดับย่อยที่พบบ่อย (Frequent subsequence) จากฐานข้อมูลลำดับ โดยมีนิยามดั้งเดิมที่ให้ไว้โดย Agrawal และ Srikant [7] ดังต่อไปนี้

"กำหนดให้ฐานข้อมูลลำดับประกอบด้วยข้อมูลลำดับ โดยแต่ละลำดับประกอบด้วยรายการของทรานแซ็คชั่น (Transaction) เรียงตามเวลา และแต่ละทรานแซ็คชั่นคือไอเท็มเซ็ต (Itemset) การทำเหมืองข้อมูลสำหรับข้อมูลลำดับ คือ การค้นหาลำดับย่อยทั้งหมดที่มีค่าซัพพอร์ท (Support) เกินค่าซัพพอร์ทขั้นต่ำ (Minimum support – min_sup) โดยค่าซัพพอร์ทของลำดับย่อย คือ จำนวนลำดับในฐานข้อมูลลำดับที่บรรจุ (Contain) ลำดับย่อยที่กำหนด"

และมีคำศัพท์และนิยามที่สำคัญดังนี้

- เซ็ตของไอเท็ม (Items) $I = \{i_1, i_{2,...}, i_p\}$; i_n คือสัญพจน์ (Literal), $1 \le n \le p$
- ไอเท็มเซ็ต (Itemset) เป็นเซ็ตของไอเท็มที่ไม่เป็นเซ็ตว่าง เขียนด้วย $(x_1 x_2 ... x_q)$; $x_n \in I, 1 \le n \le q$ ในกรณีที่ไอเท็มเซ็ตมีเพียงไอเท็มเดียว สามารถละเครื่องหมายวงเล็บได้ เช่น (x_1) เขียนด้วย x_1 แทน
- ลำดับ (Sequence) เขียนด้วย $\langle e_1, e_2, ..., e_l \rangle$; e_n คือไอเท็มเซ็ต และ e_i มาก่อน e_j เมื่อ $i < j$
- ลำดับ $s = \langle e_1, e_2, ..., e_l \rangle$ เรียกว่า ลำดับความยาว l (l-length sequence)
- $\alpha = \langle a_1, a_2, ..., a_p \rangle$ เป็นลำดับย่อย (Subsequence) ของ $\beta = \langle b_1, b_2, ..., b_q \rangle$ เมื่อมีจำนวนเต็ม $j_1, j_2, ..., j_p$; $1 \le j_1 < j_2 < ... < j_p \le q$ ที่ทำให้ $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, ..., a_p \subseteq b_{j_p}$ โดยสามารถเขียนด้วยสัญลักษณ์ $\alpha \subseteq \beta$

ข้อมูลขาเข้าในการทำเหมืองข้อมูลสำหรับข้อมูลลำดับ คือ ฐานข้อมูลลำดับ S ซึ่งเป็นเซ็ตของ 2-tuple $\langle SID, s \rangle$ เมื่อ SID คือ ID ของข้อมูลที่ไม่ซ้ำกันสำหรับทุกข้อมูลในฐานข้อมูลและ s คือลำดับ โดยการกล่าวถึงฐานข้อมูลลำดับในบางครั้งอาจตัดทอน ID ออกไป เช่น ฐานข้อมูลลำดับ S ประกอบด้วยลำดับ $s_1, s_2, ..., s_n$ ซึ่งในกรณีนี้ให้ถือว่าข้อมูลยังคงอยู่ในรูปของ 2-tuple ที่มี ID ซึ่งไม่ซ้ำกันสำหรับข้อมูลทุกตัวในฐานข้อมูลเสมอ เพียงแต่การกล่าวถึงตัวข้อมูลอาจจะกล่าวถึงเฉพาะลำดับได้

ข้อมูลขาออกในการทำเหมืองข้อมูลบนฐานข้อมูล S คือ รายการของลำดับย่อยที่พบบ่อย โดยลำดับย่อย α จะต้องเป็นไปตามเงื่อนไข $\sup(\alpha, S) \ge \min\_\sup$ เมื่อ

- $\min\_\sup$ เป็นจำนวนเต็มบวกที่กำหนดไว้สำหรับฐานข้อมูล S
- $\sup(\alpha, S) = \left| \{ \langle SID, s \rangle \in S \mid \alpha \subseteq s \} \right|$

ขั้นตอนวิธีที่ใช้ในการทำเหมืองข้อมูลสำหรับข้อมูลลำดับมีหลากหลายวิธีการ สำหรับงานวิทยานิพนธ์นี้เลือกขั้นตอนวิธี PrefixSpan [8] มาปรับปรุงให้เข้ากับลักษณะของงาน โดยรายละเอียดของขั้นตอนวิธี PrefixSpan จะกล่าวถึงในหัวข้อที่ 2.2.5 และวิธีการปรับปรุงจะกล่าวถึงในหัวข้อที่ 5.3.2

2.1.2.2 การทำเหมืองข้อมูลสำหรับข้อมูลกราฟ (Graph pattern mining)

การทำเหมืองข้อมูลสำหรับข้อมูลกราฟ คือ การสกัดกราฟย่อยที่พบบ่อย (Frequent subgraph) จากฐานข้อมูลกราฟ โดยมีคำศัพท์และนิยามที่สำคัญดังนี้

- กราฟ g คือเซ็ตของจุดยอดและเส้นเชื่อมที่เชื่อมระหว่างคู่จุดยอด แสดงได้ด้วย 2-tuple (V,E) โดยแต่ละสมาชิกมีนิยามดังนี้
  - เซ็ตจุดยอด (Vertex) v_n ; $V=\{v_1,v_{2,\ldots},v_p\}$ คือสัญพจน์, $1\le n\le p$
  - เซ็ตเส้นเชื่อม (Edge) $E\subseteq V\times V$ สมาชิกของเซ็ตถือเป็นคู่แบบไม่มีลำดับ
  - ในกรณีที่เป็นกราฟระบุทิศทาง (Directed graph) สมาชิกของเซ็ตเส้นเชื่อมจะเป็นคู่แบบมีลำดับ จุดยอดตัวแรกของคู่จะเป็นหัว และจุดยอดตัวหลังเป็นหาง เส้นเชื่อมนั้นเชื่อมจากหัวไปยังหาง
- ในกรณีที่เป็นกราฟระบุชื่อ (Labeled graph) จะสามารถแสดงด้วย 4-tuple (V,E,L,l) โดยสมาชิกที่เพิ่มขึ้นมามีนิยามดังนี้
  - เซ็ตชื่อ (Label set) $L=\{l_1,l_{2,\ldots},l_m\}$; l_n คือสัญพจน์, $1\le n\le m$
  - ฟังก์ชั่นชื่อ (Label mapping function) $l:E\cup V\to L$ สำหรับกำหนดชื่อให้กับจุดยอดและเส้นเชื่อม
- ฟังก์ชั่นสมสัณฐานกราฟ (Graph isomorphism) สำหรับกราฟระบุชื่อในกรณีของการทำเหมืองข้อมูลสำหรับข้อมูลกราฟ [9] เป็นฟังก์ชั่นหนึ่งต่อหนึ่งทั่วถึง (Bijective function) $f:V(G)\to V(G')$ ซึ่ง
  - $\forall u\in V(G), l_G(u)=l_{G'}(f(u))$
  - $\forall (u,v)\in E(G), f((u,v))\in E(G')\wedge l_G((u,v))=l_{G'}((f(u),f(v))$
  - โดย $V(g)$ แทนเซ็ตจุดยอดของกราฟ g และ $E(g)$ แทนเซ็ตเส้นเชื่อมของกราฟ g

หากมีฟังก์ชั่นดังกล่าวสำหรับกราฟระบุชื่อ g และ g' แสดงว่ากราฟทั้งสองนั้นสมสัณฐานกัน

รูปที่ *1*แสดงตัวอย่างกราฟที่สมสัณฐานกัน

- กราฟ g บรรจุกราฟ h ก็ต่อเมื่อมีกราฟย่อย (Subgraph) $g'\subseteq g$ ซึ่ง g' และ h สมสัณฐานกัน

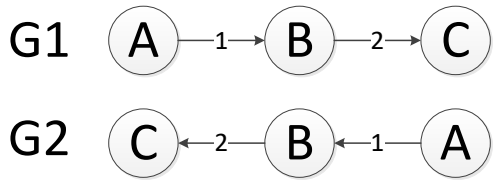
ข้อมูลขาเข้าในการทำเหมืองข้อมูลสำหรับข้อมูลกราฟ คือ ฐานข้อมูลกราฟ G ซึ่งอาจอยู่ในรูปของเซ็ตของกราฟ หรือเซ็ตของ 2-tuple $\langle ID,g\rangle$ โดย ID คือ ID ของกราฟที่ไม่ซ้ำกันสำหรับทุกข้อมูลในฐานข้อมูลและ g คือกราฟ

ข้อมูลขาออกในการทำเหมืองข้อมูลบนฐานข้อมูลกราฟ G คือ รายการของกราฟย่อยที่พบบ่อย โดยกราฟย่อย g ที่พบบ่อยมีนิยามว่า $\sup(g,G)\ge \min\_\sup$ เมื่อ

- $\min\_\sup$ เป็นจำนวนเต็มบวกที่กำหนดไว้สำหรับฐานข้อมูลกราฟ G
- $\sup(g,G)=\sum_{G_i\in G}\varsigma(g,G_i)$ โดย $\varsigma(g,G_i)$ มีค่าเป็น 1 เมื่อ G_i บรรจุ g และเป็น 0 เมื่อ G_i ไม่บรรจุ g

ปัจจุบันมีงานวิจัยจำนวนหลากหลายงานที่นำเสนอขั้นตอนวิธีสำหรับการทำเหมืองข้อมูลบนข้อมูลกราฟ ในงานวิทยานิพนธ์นี้เลือกใช้ [9] สำหรับการทำเหมืองข้อมูล เนื่องจาก gSpan รองรับกราฟแบบมีทิศทางและกราฟ

ระบุชื่อซึ่งเป็นแบบข้อมูลที่แสดงถึงรูปแบบพฤติกรรมบางประเภทที่งานวิทยานิพนธ์นี้ให้ความสนใจ gSpan ที่งานวิทยานิพนธ์นี้ใช้ถูกรวมอยู่ในชุดเครื่องมือทำเหมืองข้อมูลสำหรับข้อมูลกราฟ ParSeMiS [10]



$G1 = (V1, E1, L1, l1)$
$V1 = \{a, b, c\}$
$E1 = \{(a,b),(b,c)\}$
$L1 = \{A, B, C, 1, 2\}$
$l = \{(a, A),(b, B),(c, C),((a,b),1),((b,c),2)\}$

$G2 = (V2, E2, L2, l2)$
$V2 = \{d, e, f\}$
$E2 = \{(e,d),(f,e)\}$
$L2 = \{A, B, C, 1, 2\}$
$l = \{(d, C),(e, B),(f, A),$
$((e,d),2),((f,e),1)\}$

รูปที่ 1 $G1$ และ $G2$ สมสัณฐานกันด้วยฟังก์ชัน $f = \{(a, f),(b,e),(c,d)\}$

2.1.3 ปัญหาการตอบสนองเงื่อนไขบังคับ (Constraint satisfaction problem -- CSP)

CSP [11] เป็นปัญหาที่เกี่ยวข้องกับการกำหนดค่าให้กับตัวแปรที่สนใจภายใต้เงื่อนไขบังคับที่ผูกมัดระหว่างตัวแปรตั้งแต่ 1 ตัวขึ้นไป ซึ่งปัญหาข้อหนึ่งประกอบด้วย 3 องค์ประกอบได้แก่ X, D และ C โดยมีนิยามที่สำคัญต่อไปนี้

- $X = \{x_1, x_2, ..., x_n\}$ คือ เซ็ตของตัวแปรที่สนใจ
- $D = \{d_1, d_2, ..., d_n\}$ คือ เซ็ตของโดเมนของตัวแปรที่สนใจ ซึ่งสมาชิก d_i เป็นเซ็ตที่ระบุค่าที่สามารถกำหนดให้ตัวแปร x_i ได้ โดยเซ็ตดังกล่าวมีลักษณะใดลักษณะหนึ่งได้ต่อไปนี้
  - โดเมนจำกัดไม่ต่อเนื่อง (Discrete, finite domain) เช่น $d_i = \{0,1,2,3,4\}$
  - โดเมนไม่จำกัดไม่ต่อเนื่อง (Discrete, infinite domain) เช่น $d_i = I^+$
  - โดเมนต่อเนื่อง (Continuous domain) เช่น $d_i = [100,300]$
- $C = \{c_1, c_2, ..., c_m\}$ คือ เซ็ตของเงื่อนไขบังคับ ซึ่งสมาชิก c_i อยู่ในรูป 2-tuple $\langle scope, rel \rangle$ โดย
  - $scope = \langle s_1, s_2, ..., s_l \rangle; s_i \in X, 1 \le i \le l$ เป็น n-tuple ของตัวแปรที่เกี่ยวข้องกับเงื่อนไขจำกัด c_i
  - rel เป็นความสัมพันธ์ที่ระบุว่าตัวแปรภายใน $scope$ เป็นค่าใดได้บ้าง โดยอาจอยู่ในรูปของ l-tuple ที่แสดงทุกค่าที่เป็นไปได้ตามเงื่อนไขบังคับ หรือความสัมพันธ์ที่สามารถ
    - ใช้ตรวจสอบ l-tuple ว่าตรงตามเงื่อนไขบังคับหรือไม่
    - แจงค่าตัวแปรที่เป็นไปได้

คำตอบของ CSP ได้จากการเลือกค่าจากเซ็ต d_i ให้กับตัวแปร x_i ทุกตัวที่เป็นสมาชิกของ X โดยในหนึ่งปัญหาสามารถมีได้หลายคำตอบหรือไม่มีคำตอบก็ได้

มีหลากหลายวิธีการที่ช่วยให้การค้นหาคำตอบของ CSP ออกมาจากปริภูมิสถานะ (State space) ได้เร็วขึ้นนอกเหนือไปจากการค้นหาโดยทั่วไป ในที่นี้ขอยกตัวอย่างวิธีการส่วนหนึ่งจากหนังสือของ Stuart Russell และ Peter Norvig [11] เช่น การแผ่เงื่อนไขบังคับ (Constraint propagation) เพื่อลดขนาดของ D หรือการทำแบ็คแทรคกิ้ง (Backtracking) โดยใช้ฮิวริสติก (Heuristic) ในการเลือกลำดับของการค้นหาเพื่อเพิ่มโอกาสในการพบคำตอบที่ไม่ตรงตามเงื่อนไขบังคับรวดเร็วขึ้น ส่งผลให้ต้นไม้ค้นหาถูกตัดออก (Prune) มากขึ้น เป็นต้น ทว่า CSP โดยทั่วไปแล้วเป็นปัญหาที่จัดอยู่ในกลุ่ม NP-Complete [12] ทำให้ไม่มีขั้นตอนวิธีที่ดีที่สุดที่ใช้ในการแก้ปัญหานี้ งานวิทยานิพนธ์นี้จึงพิจารณาเลือกตัวแก้ไขปัญหาบังคับ (Constraint solver) โดยพิจารณาจาก API ว่ารองรับการสร้างแบบจำลองปัญหาในบริบทของงานวิทยานิพนธ์นี้ได้หรือไม่ ซึ่งเครื่องมือที่งานวิทยานิพนธ์นี้เลือกคือ JaCoP [13]

JaCoP รองรับการใช้งานโดเมนทั้ง 3 รูปแบบ และมีเงื่อนไขบังคับหลายแบบให้เลือกใช้งาน ในที่นี้จะแจกแจงรายละเอียดเฉพาะเงื่อนไขบังคับที่มีการใช้งานในวิทยานิพนธ์นี้

- $XlteqY(x,y)$ เป็นเงื่อนไขที่ระบุว่าค่าของตัวแปร x ต้องน้อยกว่าหรือเท่ากับตัวแปร y ซึ่งแสดงด้วยสัญลักษณ์ได้เป็น $\langle (x,y), x \le y \rangle$
- $XltC(x,c)$ เป็นเงื่อนไขที่ระบุว่าค่าของตัวแปร x ต้องน้อยกว่าค่าคงที่ c ซึ่งแสดงด้วยสัญลักษณ์ได้เป็น $\langle (x), x < c \rangle$
- $Count(X,c,y)$ เป็นเงื่อนไขที่ระบุว่าค่าของตัวแปร y จะต้องเท่ากับจำนวนตัวแปรในเซ็ต X ที่มีค่าเท่ากับค่าคงที่ c หรือแสดงด้วยสัญลักษณ์ได้เป็น $\langle X \cup \{y\}, y = |\{x' \mid x' \in X \wedge x' = c\}| \rangle$
- $Sum(X,y)$ เป็นเงื่อนไขที่ระบุว่าค่าของตัวแปร y จะต้องเท่ากับผลรวมของตัวแปรทั้งหมดในเซ็ต X หรือแสดงด้วยสัญลักษณ์ได้เป็น $\left\langle X \cup \{y\}, y = \sum_{x \in X} x \right\rangle$
- $Alldiff(X)$ เป็นเงื่อนไขที่ระบุว่าค่าของตัวแปรทุกตัวในเซ็ต X ต้องไม่เท่ากัน หรือแสดงด้วยสัญลักษณ์ได้เป็น $\langle X, \forall x_i, x_j \in X (i \ne j \rightarrow x_i \ne x_j) \rangle$
- $\mathrm{Re}gular(FSM,X)$ เป็นเงื่อนไขที่ระบุว่าค่าของตัวแปรทุกตัวในเซ็ต X ต้องมีความสัมพันธ์กันตามที่ระบุโดยเครื่องสถานะจำกัด FSM ซึ่งสามารถนิยามความสัมพันธ์ได้ดังนี้ และแสดงตัวอย่างของเงื่อนไขบังคับได้ดังรูปที่ 2

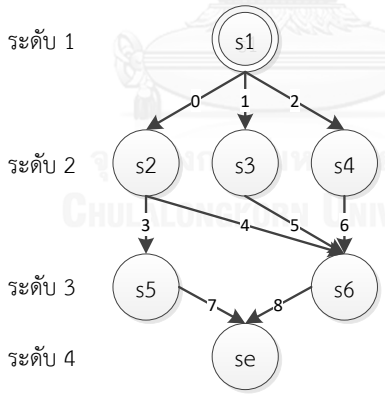
กำหนดให้ $X = \{x_1, x_2, .., x_n\}$ ซึ่งแต่ละตัวแปรมีโดเมนเป็นโดเมนจำกัดไม่ต่อเนื่องและเครื่องสถานะจำกัด FSM มีคุณลักษณะตามกฎต่อไปนี้

- มีจำนวน m สถานะ $(m \ge n+1)$ ซึ่งประกอบด้วยสถานะเริ่มต้น s_1 สถานะจบ s_e และสถานะอื่น ๆ จำนวน $m-2$ สถานะ
- แต่ละสถานะจะต้องมีค่า “ระดับ” ของตนเอง โดย
  - สถานะเริ่มต้น s_1 จะต้องมีค่าระดับเป็น 1 เสมอ
  - สถานะจบ s_e จะต้องมีค่าระดับเป็น $n+1$ เสมอ
  - สถานะอื่น ๆ มีค่าระดับเป็นค่าใดก็ได้ในช่วงค่า $[2,n]$

- เซ็ตของค่าระดับที่กำหนดให้กับทุกสถานะจะต้องเท่ากับ $[1, n+1]$
- การเปลี่ยนสถานะ (transition) จะเริ่มจากสถานะที่มีค่าระดับเป็น i ไปยัง $i+1$ เท่านั้น
- ทุกสถานะ s ใด ๆ ที่มีค่าระดับในช่วง $[2, n]$ จะต้องมีการเปลี่ยนสถานะจากสถานะใด ๆ อย่างน้อย 1 สถานะมายัง s และมีการเปลี่ยนสถานะจาก s ไปยังสถานะอื่นอย่างน้อย 1 สถานะเสมอ
- เงื่อนไขของการเปลี่ยนสถานะจากสถานะที่มีค่าระดับเป็น i ไปยังสถานะอื่นจะระบุด้วยตัวเลขที่อยู่ในโดเมนของ x_i

ค่าของตัวแปรทั้งหมดใน X จะถูกกำหนดโดยค่าระดับของสถานะและการเปลี่ยนสถานะที่เกิดขึ้นใน FSM โดย

- เริ่มต้น FSM ที่สถานะเริ่มต้น s_1 หากค่าของ x_1 เป็น c_1 ให้เปลี่ยนสถานะออกจาก s_1 โดยใช้การเปลี่ยนสถานะที่มีตัวเลข c_1 ระบุไว้
- ที่สถานะ s ใด ๆ ที่ไม่ใช่สถานะจบ s_e และมีค่าระดับเป็น k หากค่าของ x_k เป็น c ให้เปลี่ยนสถานะออกจาก s โดยใช้การเปลี่ยนสถานะที่มีตัวเลข c ระบุไว้
- หากค่าของตัวแปรทั้งหมดใน X สามารถทำให้ FSM เปลี่ยนสถานะไปจนถึงสถานะจบ s_e ได้ แสดงว่า X เป็นไปตามเงื่อนไขบังคับ $\mathrm{Re}\,gular(FSM, X)$



กำหนดให้ $X = \{x_1, x_2, x_3\}$ และ FSM เป็นดังภาพทางซ้าย ค่าของตัวแปรที่เป็นไปตามเงื่อนไขบังคับ $\mathrm{Re}\,gular(FSM, X)$ ประกอบด้วย {0,3,7}, {0,4,8}, {1,5,8}, {2,6,8}

รูปที่ 2 ตัวอย่าง FSM สำหรับเงื่อนไขบังคับ Regular

2.1.4 การหาลำดับย่อยร่วมยาวสุดทุกเอ็มเบ็ดดิ้ง (All longest common subsequence embeddings)

ปัญหาการหาลำดับย่อยร่วมยาวสุด (Longest common subsequence -- LCS) [14] เป็นการหาลำดับย่อยร่วมของลำดับ 2 ตัวที่ยาวที่สุด โดยมีนิยามของลำดับย่อยและลำดับย่อยร่วมดังนี้

- ลำดับ $Z=\{z_1,z_2,..,z_k\}$ เป็นลำดับย่อยของ $X=\{x_1,x_2,..,x_m\}$ ถ้ามีลำดับที่เพิ่มขึ้น (Increasing sequence) $\{i_1,i_2,..,i_k\}$ ที่ทำให้ $x_{i_j}=z_j$ สำหรับทุก j $1\le j\le k$
- ลำดับ Z เป็นลำดับย่อยร่วมของ X และ Y ก็ต่อเมื่อ Z เป็นลำดับย่อยของทั้ง X และ Y

ลำดับ 2 ลำดับใด ๆ อาจมีลำดับย่อยร่วมยาวสุดได้หลายลำดับ เช่น กำหนดให้ $X=balaclava$ และ $Y=bilabial$ จะได้ว่า ลำดับย่อยร่วมยาวสุดของ X และ Y ประกอบด้วย $baal$ $blal$ และ $blaa$

ในกรณีที่เป็นการหาลำดับย่อยร่วมยาวสุดทุกเอ็มเบ็ดดิ้ง จะพิจารณาทุกเอ็มเบ็ดดิ้งของ LCS ที่ได้ โดยเอ็มเบ็ดดิ้ง คือ ลำดับย่อยที่พิจารณาถึงความแตกต่างเชิงตำแหน่งของสมาชิกในลำดับ ตัวอย่างเช่น $blal$ ซึ่งเป็น LCS ของ $X=balaclava$ และ $Y=bilabial$ มี 2 เอ็มเบ็ดดิ้ง ได้แก่ $b_1^1l_3^3a_4^4l_6^8$ และ $b_1^1l_3^3a_4^7l_6^8$ เมื่อ z_j^i แสดงถึงสมาชิกร่วม z ของ X และ Y ซึ่ง z ปรากฏที่ตำแหน่ง i ใน X และปรากฏที่ตำแหน่ง j ใน Y

ขั้นตอนวิธีสำหรับหา LCS มีหลายวิธี งานวิทยานิพนธ์นี้เลือกใช้กำหนดการพลวัตแบบล่างขึ้นบน (Bottom-up dynamic programming) ซึ่งใช้วิธีการสร้างแมทริกซ์ของความยาว LCS และย้อนรอย (Backtracing) เพื่อหา LCS และปรับปรุงวิธีการย้อนรอยเพื่อหา LCS ทุกเอ็มเบ็ดดิ้ง รายละเอียดขั้นตอนวิธีสำหรับการหา LCS เป็นดังนี้

Given 2 sequences $X=\{x_1,x_2,..,x_m\}$ and $Y=\{y_1,y_2,..,y_n\}$

Construct a $(m+1)\times(n+1)$ matrix M where

$$M_{i,j}=\begin{cases}0 & i=0\vee j=0\\ M_{i-1,j-1}+1 & (i,j>0)\wedge x_i=y_j\\ \max(M_{i-1,j},M_{i,j-1}) & (i,j>0)\wedge x_i\neq y_j\end{cases}$$

denotes length of LCS of sequence $X'=\{x_1,x_2,..,x_i\}$ and $Y'=\{y_1,y_2,..,y_j\}$

Backtrace to find LCS of X and Y using:

$LCS=\phi$ $p=m,q=n$

While($p>0$ and $q>0$) do:

If(x_p == y_q) do: $LCS\leftarrow x_p$

If(M_{p-1} == M_{q-1}) do: $p=p-1,q=q-1$

Else if($M_{p-1}>M_{q-1}$) do: $p=p-1$

Else do: $q=q-1$

Return Reverse(LCS) as result

สำหรับการหาให้ครบทุกเอ็มเบ็ดดิ้ง จะต้องทำการย้อนรอยทุกเส้นทางที่เป็นไปได้ โดยใช้ฟังก์ชั่นเรียกซ้ำ (Recursive function) ซึ่งทำหน้าที่หา LCS ทุกเอ็มเบ็ดดิ้งของลำดับ X และ Y จนถึงสมาชิกที่ i ของ X และ j ของ Y ซึ่งแสดงด้วยรหัสเทียมได้ดังนี้

Function FindAllLCSEmbedding (i , j){

If(i == 0 or j == 0) do: Return ϕ

If(x_i == y_j) do:

FindAllLCSEmbedding ($i-1$, $j-1$) and store result in Tmp

For(Sequence lcs in Tmp) do: $lcs\leftarrow x_i$

Return Tmp as result

Else do:

$Tmp = \phi$

if($M_{p-1} >= M_{q-1}$) do: $Tmp \leftarrow$ FindAllLCSEmbedding ($i-1$, j)

if($M_{p-1} <= M_{q-1}$) do: $Tmp \leftarrow$ FindAllLCSEmbedding (i , $j-1$)

Return Tmp as result

}

Backtrace to find all LCS embeddings of X and Y using: FindAllLCSEmbedding(m , n)

2.1.5 การวิเคราะห์ความถดถอย (Regression analysis)

การวิเคราะห์ความถดถอย [15, 16] เป็นวิธีการทางสถิติสำหรับศึกษาความสัมพันธ์ระหว่างตัวแปร 2 กลุ่ม ได้แก่ ตัวแปรอิสระ (Independent variable หรือ Predictor) และตัวแปรตาม (Dependent variable หรือ Response) เพื่อหาความสัมพันธ์ของตัวแปรในรูปสมการคณิตศาสตร์ที่เรียกว่า สมการถดถอย ซึ่งสามารถใช้พยากรณ์ค่าของตัวแปรตามเมื่อกำหนดค่าตัวแปรต้นให้ ความสัมพันธ์ระหว่างตัวแปรมีได้ในหลายรูปแบบ เช่น เชิงเส้น หรือโพลิโนเมียล เป็นต้น ในการวิเคราะห์ความถดถอยบนข้อมูลกลุ่มตัวอย่าง ซึ่งประกอบด้วยข้อมูลย่อย (Experimental unit) หลายตัวที่ได้มาจากการสำรวจ โดยแต่ละตัวจะประกอบด้วยค่าของตัวแปรอิสระทั้งหมดและค่าของตัวแปรตามทั้งหมดที่เกิดขึ้นจากตัวแปรอิสระ จะต้องสร้างสมการถดถอยที่ทำให้ผลรวมของความคลาดเคลื่อนระหว่างข้อมูลย่อยและสมการมีค่าน้อยที่สุด

งานวิทยานิพนธ์นี้ใช้งานการวิเคราะห์ความถดถอย 2 แบบ ได้แก่ การวิเคราะห์ความถดถอยเชิงเส้น และการวิเคราะห์ความถดถอยแบบโพลิโนเมียลลำดับที่ 2

การวิเคราะห์ความถดถอยเชิงเส้น (Linear regression)

เป็นการศึกษาความสัมพันธ์แบบเชิงเส้นระหว่างตัวแปรอิสระ 1 ตัว (แทนด้วย x) และตัวแปรตาม 1 ตัว (แทนด้วย y) ข้อมูลกลุ่มตัวอย่างจะถูกใช้ในการสร้างสมการถดถอยในรูป $y = b_0 + b_1 x$

การวิเคราะห์ความถดถอยแบบโพลิโนเมียลลำดับที่ 2 (Quadratic polynomial regression)

เป็นการศึกษาความสัมพันธ์แบบโพลิโนเมียลลำดับที่ 2 ระหว่างตัวแปรอิสระ 1 ตัว (แทนด้วย x) และตัวแปรตาม 1 ตัว (แทนด้วย y) ข้อมูลกลุ่มตัวอย่างจะถูกใช้ในการสร้างสมการถดถอยในรูป $y = b_0 + b_1 x + b_2 x^2$

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 งานวิจัยที่เกี่ยวกับการสร้างสิ่งมีชีวิตประดิษฐ์ (Artificial creature)

สมบัติสำคัญประการหนึ่งของการเป็นสิ่งมีชีวิตคือ ความสามารถในการตอบสนองต่อสิ่งเร้า [17] จากมุมมองของผู้วิจัย การตอบสนองของสิ่งมีชีวิตอาจทำให้เกิดเป็นพฤติกรรมที่มีความหมายขึ้นได้ ในงานวิทยานิพนธ์นี้ต้องการสร้างพฤติกรรมที่มีรูปแบบตายตัวของศัตรู แม้ว่าพฤติกรรมที่มีรูปแบบตายตัวอาจไม่ได้เกิดจากการตอบสนองต่อสิ่งเร้าในลักษณะเดียวกับที่เกิดขึ้นในสิ่งมีชีวิต แต่พฤติกรรมที่อาจเกิดขึ้นจากการสร้างสิ่งมีชีวิตประดิษฐ์อาจจะมีความเกี่ยวข้องหรือสามารถนำมาประยุกต์ใช้กับงานวิทยานิพนธ์นี้ได้ ผู้วิจัยจึงทำการสำรวจงานวิจัยทางด้านนี้เพื่อความครอบคลุมของการทบทวนวรรณกรรม

Thomas Miconi และ Alastair Channon [18] นำเสนอระบบสำหรับการสร้างและวิวัฒน์สิ่งมีชีวิตประดิษฐ์ สิ่งมีชีวิตในงานวิจัยนี้เกิดจากการนำบล็อกสี่เหลี่ยมหลายรูปทรงมาเชื่อมต่อกันอย่างมีลำดับชั้นตามแบบจำลองของ Karl Sim โดยมีบล็อกหนึ่งเป็นบล็อกราก บล็อกลำดับชั้นที่สองจะถูกเชื่อมเข้ากับบล็อกรากด้วยข้อต่อแบบบานพับ (Hinge joint) บล็อกลำดับชั้นถัดไปจะเชื่อมต่อกับบล็อกลำดับชั้นก่อนหน้าในลักษณะเดียวกัน บล็อกแต่ละบล็อกสามารถเคลื่อนไหวได้โดยการหมุนบล็อกลำดับชั้นรองรอบข้อต่อที่เชื่อมกับบล็อกในลำดับชั้นที่เหนือกว่า การหมุนของแต่ละบล็อกจะถูกควบคุมโดยโครงข่ายประสาทเทียมซึ่งรับค่านำเข้าจากหน่วยรับความรู้สึกที่ฝังอยู่ภายในทุกบล็อกซึ่งมีหน้าที่วัดค่าจากสภาพแวดล้อม เช่น ระยะทางจากบล็อกไปยังสิ่งมีชีวิตประดิษฐ์ตัวอื่น เมื่อนำสิ่งมีชีวิตประดิษฐ์ตามนิยามดังกล่าวนี้ไปอยู่ในโลกเสมือนสามมิติที่มีการจำลองฟิสิกส์แบบสมจริงโดยกำหนดให้ทุกบล็อกของสิ่งมีชีวิตเป็นวัตถุแข็งเกร็ง สิ่งมีชีวิตประดิษฐ์ก็จะสามารถเคลื่อนไหวได้

สิ่งมีชีวิตประดิษฐ์ที่ได้จะถูกวิวัฒน์ด้วยตัวดำเนินการเชิงพันธุกรรมเพื่อผสมรวมบล็อกจากสิ่งมีชีวิตประดิษฐ์สองตัว หรือกลายพันธุ์ค่าพารามิเตอร์ของโครงข่ายประสาทเทียมของแต่ละบล็อก สิ่งมีชีวิตประดิษฐ์ที่ได้จะถูกวัดผลด้วยการแข่งขันครองกล่อง สิ่งมีชีวิตใดที่เข้าใกล้กล่องที่กำหนดในโลกเสมือนมากกว่าจะได้คะแนนมากกว่า

งานชิ้นถัดมาของ Thomas Miconi [19] นำเสนอสภาพแวดล้อมสำหรับทดลองการวิวัฒนาการของสิ่งมีชีวิตประดิษฐ์ที่อาศัยแนวคิดของการคัดเลือกโดยธรรมชาติ ซึ่งสิ่งมีชีวิตในงานนี้จะต้องต่อสู้กันเพื่อขยายเผ่าพันธุ์ของตนเอง สิ่งมีชีวิตที่สามารถฆ่าสิ่งมีชีวิตตัวอื่นได้ จะทำการเพาะพันธุ์สิ่งมีชีวิตพันธุ์เดียวกัน อีกวิธีหนึ่งของการเพิ่มจำนวนสิ่งชีวิตพันธุ์เดียวกันคือการมีอายุยืนถึงจุดที่กำหนด

งานนี้ใช้สิ่งมีชีวิตอิงแบบจำลองของ Karl Sim ซึ่งก็คือ สิ่งมีชีวิตแบบบล็อกที่เชื่อมต่อกันด้วยข้อต่อบานพับและควบคุมด้วยโครงข่ายประสาทเทียม โลกเสมือนที่สิ่งมีชีวิตอยู่จะมีการจำลองฟิสิกส์แบบสมจริงและบล็อกของสิ่งมีชีวิตจะถูกนับเป็นวัตถุแข็งเกร็ง วิวัฒนาการจะเกิดขึ้นเมื่อจะเพาะพันธุ์สิ่งมีชีวิตใหม่ โดยจะใช้วิธีการผสมรวมบล็อกหรือกลายพันธุ์เช่นเดียวกับงานก่อนหน้าของผู้วิจัย [18] จุดที่ต่างออกไปคือการให้สิ่งมีชีวิตสามารถสร้างความเสียหายให้กับสิ่งมีชีวิตตัวอื่นได้โดยการเคลื่อนบล็อกส่วนหนึ่งของตนเข้าหาบล็อกของสิ่งมีชีวิตตัวอื่น ยิ่งเคลื่อนที่เร็วก็ยิ่งสร้างความเสียหายได้มาก

จากสองงานวิจัยข้างต้น จะพบว่า พฤติกรรมของสิ่งมีชีวิตในโลกเสมือนได้จากการทำงานของโครงข่ายประสาทเทียมที่ควบคุมการเคลื่อนไหวของบล็อกร่วมกับการใช้ฟิสิกส์แบบสมจริง หากนำมาใช้งานภายในเกม ข้อดีของวิธีดังกล่าวคือศัตรูที่สร้างขึ้นด้วยวิธีแบบเดียวกันนี้จะมีรูปร่าง กล่าวคือ วิธีนี้นอกจากจะสร้างพฤติกรรมได้แล้ว

ยังสามารถสร้างรูปร่างของศัตรูขึ้นมาด้วย ทว่ารูปร่างที่เกิดขึ้นอาจไม่เหมาะสมกับการใช้งานในบริบทของเกมได้ เนื่องจากรูปร่างของสิ่งมีชีวิตและพฤติกรรมจะต้องสอดคล้องกันภายใต้กฎฟิสิกส์ ซึ่งศัตรูในเกมจำนวนมากมีพฤติกรรมที่ไม่สามารถกระทำด้วยรูปร่างที่ปรากฏในเกมเมื่ออยู่ภายใต้กฎฟิสิกส์ เช่น ศัตรูรูปร่างทรงกลมที่สามารถลอยและเปลี่ยนทิศทางในอากาศได้ นอกจากนี้ เนื่องจากรูปร่างของสิ่งมีชีวิตจากงานวิจัยทั้งสองนี้เกี่ยวพันกับพฤติกรรมที่เป็นไปได้ของสิ่งมีชีวิตโดยตรง ทำให้ไม่สามารถแยกพฤติกรรมและรูปร่างออกจากกันได้ ผลที่ตามมาคือแนวคิดดังกล่าวนี้ไม่น่าจะนำมาใช้กับการสร้างพฤติกรรมในลักษณะเดียวกับงานวิทยานิพนธ์นี้ได้

Nedjma Djezzar และคณะ [20] สร้างสิ่งมีชีวิตประดิษฐ์ด้วยการใช้ L-System ซึ่งเป็นไวยากรณ์ (Grammar) รูปแบบหนึ่งที่ใช้ในการอธิบายการเติบโตของเซลล์ร่วมกับแนวคิดเคมีของเซลล์ สิ่งมีชีวิตประดิษฐ์ในงานวิจัยนี้เป็นสิ่งมีชีวิตระดับเซลล์ที่อาศัยอยู่ในโลกเสมือนสองมิติ ภายในโลกแห่งนี้จะประกอบด้วยเซลล์และสารเคมีในธรรมชาติ พฤติกรรมของเซลล์จะถูกนิยามไว้ในระดับต่ำซึ่งเกี่ยวข้องกับสารเคมีของเซลล์และการกระทำของเซลล์ในโลกจริง เช่น การย่อยสลายตัวเองและการแบ่งตัว

วิธีการในงานวิจัยนี้น่าจะเหมาะสมกับการศึกษาเรื่องของเซลล์และสิ่งมีชีวิตในเชิงชีววิทยามากกว่า ด้วยความที่พฤติกรรมอยู่ในระดับต่ำมาก รวมทั้งไม่มีฟิสิกส์เข้ามาเกี่ยวข้อง ทำให้กิจกรรมเชิงกล เช่น การเคลื่อนที่ การผลักกันของเซลล์ไม่สามารถเกิดขึ้นได้ในโลกเสมือนของงานวิจัยนี้ งานวิจัยในลักษณะของการสร้างสิ่งมีชีวิตด้วยแนวคิดเคมีของเซลล์จึงไม่เหมาะสมที่จะนำมาใช้ร่วมกับงานวิทยานิพนธ์นี้

2.2.2 งานวิจัยที่เกี่ยวกับการสร้างเนื้อหาอัตโนมัติ (PCG)

งานวิทยานิพนธ์นี้เกี่ยวข้องกับการสร้างรูปแบบพฤติกรรมของศัตรูซึ่งนับได้ว่าเป็นเนื้อหาของเกม ในที่นี้จึงสำรวจงานวิจัยที่เกี่ยวกับการสร้างเนื้อหาของเกมอัตโนมัติ (PCG)

Mark Hendrikx และคณะ [21] แบ่งประเภทเนื้อหาของเกมออกเป็น 6 ลำดับชั้น แต่ละลำดับชั้นจะมีการแบ่งชนิดย่อยของเนื้อหาลงไป เนื้อหาที่อยู่ในลำดับชั้นด้านล่างจะสามารถนำมาประกอบรวมกันเป็นเนื้อหาในลำดับชั้นบนที่ถัดขึ้นไปได้ เนื้อหาทั้ง 6 ลำดับชั้นเรียงจากชั้นล่างสุดประกอบด้วย เกมบิท (Game bit) พื้นที่ของเกม (Game space) ระบบของเกม (Game system) เนื้อเรื่องเกม (Game scenario) การออกแบบเกม (Game design) และ เนื้อหาพลอยได้ (Derived content) เนื้อหาในชั้นเกมบิทเป็นหน่วยย่อยที่สุดของเกมที่ประกอบด้วย เท็กซ์เจอร์ เสียง พันธุ์ไม้ สิ่งปลูกสร้าง พฤติกรรม และวัตถุธรรมชาติ (ไฟ น้ำ หิน และเมฆ) ในงานวิจัยนี้ได้ให้คำจำกัดความของพฤติกรรมอันเป็นส่วนหนึ่งของเกมบิทไว้ว่า "เป็นวิธีการที่วัตถุโต้ตอบกับวัตถุอื่นหรือสิ่งแวดล้อม" ซึ่งตรงกับรูปแบบพฤติกรรมของศัตรูที่งานวิทยานิพนธ์นี้ให้ความสนใจ

ในงานวิจัยนี้ได้สำรวจขั้นตอนวิธีที่ใช้ในการสร้างเนื้อหาที่เกี่ยวข้องพฤติกรรมเอาไว้สองงาน ได้แก่ การใช้ไวยากรณ์แบบ L-System ร่วมกับเงื่อนไขเพิ่มเติมในการสร้างเส้นทางการเคลื่อนที่ของดอกไม้ไฟ [22] และการใช้ Compositional pattern producing networks (CPPNs) ในการสร้างเส้นทางการเคลื่อนที่ของกระสุนอัตโนมัติสำหรับเกมยานยิง [23]

งานวิจัยที่เกี่ยวข้องกับการสร้างพฤติกรรมทั้งสองงานดังกล่าวเน้นไปที่เส้นทางการเคลื่อนที่ของวัตถุเป็นหลัก ถึงแม้ว่าเส้นทางการเคลื่อนที่อาจจะเรียกได้ว่าเป็นส่วนสำคัญหนึ่งของรูปแบบพฤติกรรม แต่พฤติกรรมของศัตรูภายในเกมที่วางขายและเป็นที่ยอมรับของผู้เล่นไม่ได้มีเพียงเส้นทางเท่านั้น แต่ยังมีเงื่อนไขของการเปลี่ยนแปลงเส้นทางการเคลื่อนที่และการกระทำเพิ่มเติมอื่น ๆ ส่งผลให้วิธีการที่เน้นไปที่เส้นทางการเคลื่อนที่อาจนำมาใช้ร่วม

เป็นส่วนหนึ่งของงานได้ แต่ไม่สามารถทำหน้าที่ในการสร้างรูปแบบพฤติกรรมที่สมบูรณ์เพียงพอตามที่งานวิทยานิพนธ์นี้ต้องการได้

ณ จุดเวลาที่จัดทำโครงร่างวิทยานิพนธ์ฉบับนี้ เนื่องจากผู้วิจัยไม่พบงานวิจัยที่เกี่ยวข้องกับการสร้างรูปแบบพฤติกรรมของศัตรูที่ตายตัวโดยตรง จึงทำการสำรวจงานวิจัยที่เกี่ยวข้องกับการวิเคราะห์องค์ประกอบของเกมและความเกี่ยวพันกับศัตรู เพื่อหาคำค้นที่จะช่วยในการค้นหางานวิจัยที่เกี่ยวข้องต่อไป

Mark J. Nelson และ Michael Mateas [24] ได้แบ่งเกมออกเป็นออกเป็น 4 ชั้น ได้แก่ ชั้นกลไก (Abstract game mechanic) ชั้นนำเสนอ (Concrete game representation) ชั้นธีมของเนื้อหา (Thematic content) และชั้นจับคู่การควบคุม (Control mapping) ชั้นกลไกนั้นเป็นตัวระบุว่าสถานะของเกม (Game state) จะเปลี่ยนแปลงไปอย่างไรตลอดการเล่น โดยการเปลี่ยนแปลงของสถานะอาจจะเกิดขึ้นเองอัตโนมัติจากความเป็นไปของเกม หรือเกิดจากการโต้ตอบจากผู้เล่นก็ได้

ในขณะเดียวกัน Miguel Sicart [25] อธิบายกลไกของเกมด้วยแนวคิดของการเขียนโปรแกรมเชิงวัตถุโดยมองว่ากลไกของเกม “เป็นเมธอดที่ถูกเรียกโดยภาคีเพื่อโต้ตอบกับสถานะของเกม” ซึ่งอิงกับคำจำกัดความของเมธอดในการเขียนโปรแกรมเชิงวัตถุว่าเป็นเหมือนกับการกระทำหรือพฤติกรรมสำหรับคลาสและเป็นกลไกที่ใช้ในการเข้าถึงสถานะต่าง ๆ ของวัตถุอื่น

สำหรับกรณีของศัตรูในเกม พฤติกรรมของศัตรูจะทำให้สถานะของศัตรูหรือสถานะของโลกของเกมเปลี่ยนแปลงไป เช่น พฤติกรรมเดินทำให้สถานะตำแหน่งของศัตรูเปลี่ยนแปลง พฤติกรรมยิงกระสุนทำให้เกิดกระสุนใหม่ขึ้นในโลกของเกม ส่งผลให้สถานะของตัวโลกเปลี่ยนไป ซึ่งเทียบได้กับชั้นกลไกในงานของ Nelson และเมื่อใช้แนวคิดของ Sicart การกระทำย่อย ๆ ของศัตรูอาจเทียบได้กับการเรียกเมธอดของคลาสศัตรู หากพิจารณาประเด็นนี้ อาจกล่าวได้ว่า รูปแบบพฤติกรรมของศัตรูนับเป็นส่วนหนึ่งของกลไกของเกมเช่นกัน

ผู้วิจัยจึงอาศัยข้อสรุปจากการทบทวนสองงานวิจัยข้างต้น เลือกใช้ “กลไกของเกม” เป็นคำค้นเพื่อสำรวจวรรณกรรมด้าน PCG ที่เกี่ยวข้องกับกลไกของเกม

2.2.3 งานวิจัยเกี่ยวกับการสร้างกลไกของเกมอัตโนมัติ (Automatic game mechanic generation)

ก่อนจะเข้าสู่การทบทวนวรรณกรรมที่เกี่ยวข้องกับการสร้างกลไกของเกม ในที่นี้ขอกล่าวถึงคำว่า "กฎของเกม" ซึ่งเป็นคำศัพท์สำคัญอีกคำหนึ่งที่จำเป็นต้องทำให้เกิดความเข้าใจตรงกัน เนื่องจากปัจจุบันยังไม่มีการกำหนดความแตกต่างอย่างชัดเจนระหว่างคำว่ากลไกของเกมและกฎของเกมเป็นทฤษฎี ผู้วิจัยหรือบุคลากรในวงการเกมอาจนำคำศัพท์เหล่านี้ไปใช้งานภายใต้ความหมายที่แตกต่างกัน ในงานวิทยานิพนธ์นี้เลือกใช้การแยกแยะความแตกต่างของศัพท์ทั้งสองตามบทความของ Lewis Pulsipher [26] ในบทความดังกล่าวนิยามถึงกลไกของเกมไว้ว่า เป็นข้อกำหนดที่ระบุถึงสิ่งที่สามารถกระทำได้เมื่อถูกต้องตามเงื่อนไข ส่วนกฎของเกมเป็นข้อกำหนดที่ระบุถึงกลไกของเกมและอื่น ๆ นั่นคือ กลไกของเกมนับเป็นส่วนหนึ่งของกฎของเกม สำหรับงานวิจัยที่ทำการสำรวจ ในงานนี้จะเลือกงานที่มีความเกี่ยวข้องกับกลไกของเกมตามนิยามข้างต้นโดยไม่คำนึงถึงคำศัพท์ที่ผู้เขียนเลือกใช้งาน

Julian Togelius และ Jurgen Schmidhuber [27] นำเสนอการออกแบบเกม Grid-based อัตโนมัติโดยการวิวัฒน์กฎของเกมจากชุดกฎเริ่มต้นภายใต้ฟังก์ชั่นความเหมาะสมที่กำหนดขึ้นจากทฤษฎีของ Schmidhuber ซึ่ง

มีแนวคิดว่า เกมจะสนุกหากผู้เล่นสามารถเรียนรู้ที่จะเล่นได้ แต่ต้องไม่ง่ายจนเกินไปในระดับที่ไม่ต้องพึ่งการเรียนรู้หรือสามารถเอาชนะโดยการสุ่มเท่านั้น

งานวิจัยชิ้นนี้กำหนดชุดของกฎเริ่มต้นเพื่อใช้เป็นปริภูมิของกฎ (Rule space) ของเกมทั้งหมดที่เป็นไปได้ ซึ่งในกฎจะระบุถึงขนาดพื้นที่ โครงสร้างของพื้นที่ เงื่อนไขการจบเกม เงื่อนไขการชนะ รูปแบบของภาคีที่เป็นไปได้ซึ่งประกอบด้วยสีของภาคี รูปแบบการเคลื่อนที่ และผลลัพธ์จากการชนกับภาคีอื่น จากนั้นสร้างเกมเริ่มต้นโดยเลือกกฎออกมาจากปริภูมิอย่างสุ่มแล้ววิวัฒน์ด้วยการสุ่มปรับพารามิเตอร์ของกฎหนึ่งข้อ กฎที่วิวัฒน์แล้วจะถูกประเมินความยากในการเล่นโดยใช้ปัญญาประดิษฐ์ซึ่งสร้างจากโครงข่ายประสาทเทียม

ในงานนี้มีส่วนของกลไกที่เกี่ยวข้องกับพฤติกรรมของภาคี โดยรูปแบบพฤติกรรมดังกล่าวเป็นพฤติกรรมที่มีระดับสูง มีเพียง 5 รูปแบบที่เป็นไปได้ ได้แก่ อยู่นิ่ง เดินระยะสั้นแล้วสุ่มทิศ เดินระยะยาวแล้วสุ่มทิศ เดินและหันขวาเมื่อชนกำแพง เดินและหันซ้ายเมื่อชนกำแพง ซึ่งส่งผลให้ขาดความหลากหลายของพฤติกรรมหากนำเอารูปแบบพฤติกรรมนี้มาใช้ในการสร้างรูปแบบพฤติกรรมของศัตรู

Adam M. Smith และ Michael Mateas [28] นำเสนอวิธีการสำหรับสร้างกฎของเกมแบบ Grid-based โดยใช้ Answer Set Programming (ASP) ซึ่งเป็นการเขียนโปรแกรมเชิงเงื่อนไขบังคับ (Constraint programming) แบบหนึ่ง ตัวโปรแกรมจะถูกเขียนขึ้นเพื่อใช้เป็นฐานความรู้สำหรับกฎของเกมที่เป็นไปได้ทั้งหมดเรียกว่า ปริภูมิออกแบบ (Design space) ซึ่งเทียบได้กับปริภูมิของกฎในงานของ Togelius จากนั้นส่วนสร้างกฎซึ่งงานนี้เลือกใช้เครื่องมือ SMODELS จะแจงนับ (enumerate) กฎที่ถูกต้องตามเงื่อนไขบังคับที่กำหนดไว้ในโปรแกรมออกมาเพื่อใช้งานในเกมทดสอบต่อไป

ข้อมูลในฐานความรู้ของงานนี้ประกอบด้วย ขนาดและโครงสร้างของพื้นที่ เป้าหมายของเกม รูปแบบของภาคีที่เป็นไปได้ซึ่งประกอบด้วยสี รูปแบบการเคลื่อนที่ ผลลัพธ์การชนกับทั้งภาคีอื่นและพื้นที่ และจำนวนของภาคี ซึ่งโดยรวมแล้วมีความใกล้เคียงกับปริภูมิของกฎในงานของ Togelius แต่จะแตกต่างกันในรายละเอียดของค่าที่เป็นไปได้ในกฎแต่ละประเภท สำหรับในส่วนของการเคลื่อนที่ซึ่งเทียบเท่ากับพฤติกรรมของศัตรูที่งานวิทยานิพนธ์นี้สนใจ งานนี้ใช้แบบจำลองของการเคลื่อนที่ซึ่งระบุว่าภาคีมีการเคลื่อนที่แบบใดให้เลือกบ้าง แต่ไม่ได้กล่าวถึงรายละเอียดของการเคลื่อนที่ที่เลือกใช้ได้และวิธีการตัดสินใจว่าจะเคลื่อนที่อย่างไร ผลจากการใช้งานแบบจำลองการเคลื่อนที่ซึ่งมีลักษณะเป็นการเลือกการเคลื่อนที่แบบหนึ่งออกมาจากเซ็ตจำกัด ส่งผลให้ความหลากหลายของพฤติกรรมมีน้อยเช่นเดียวกับกรณีงานของ Togelius

Michael Cook, Simon Colton และ Jeremy Gow [29] นำเสนอระบบสำหรับออกแบบเกมแพลทฟอร์มอัตโนมัติโดยใช้วิธีการวิวัฒน์ร่วม (Co-operative co-evolution -- CCE) ระหว่าง 3 องค์ประกอบ ได้แก่ แผนที่ (Map) โครงสร้าง (Layout) และไอเท็มเพิ่มพลัง (Powerset) งานวิจัยนี้ทดสอบระบบโดยการสุ่มสร้างองค์ประกอบขึ้นมาองค์ประกอบละ 200 หน่วยประชากร นำมาสร้างเป็นเกมทั้งหมด 600 เกม แล้วเลือกเกมที่มีคะแนนจากการประเมินด้วยฟังก์ชั่นความเหมาะสมสูงที่สุด เป็นจำนวน 10% ของประชากรทั้งหมดเพื่อเป็นพ่อแม่ของประชากรรุ่นถัดไป การผลิตประชากรรุ่นถัดไปจะได้จากการไขว้เปลี่ยนและการกลายพันธุ์ของพ่อแม่ ซึ่งตัวดำเนินการดังกล่าวจะแตกต่างกันออกไปสำหรับแต่ละองค์ประกอบเช่นกัน สำหรับการประเมินระบบของงานนี้จะใช้ผู้เล่นในการประเมินเกมที่ได้จากการสร้างจากระบบโดยการให้คะแนนในช่วง 1-5 คะแนน

ในรายละเอียดสำหรับส่วนขององค์ประกอบโครงสร้างของงานวิจัยนี้มีการกล่าวถึงภาคีที่เป็นศัตรู โดยองค์ประกอบโครงสร้างจะอยู่ในรูปของรายการของศัตรูที่จะปรากฏในเกม ศัตรูแต่ละตัวประกอบด้วย การกระทำ การเคลื่อนที่ และตำแหน่ง

- การกระทำจะกล่าวถึงวิธีการที่ศัตรูใช้ในการโจมตีผู้เล่น ซึ่งมีทั้งหมด 3 รูปแบบ ศัตรูตัวหนึ่ง ๆ อาจมีรูปแบบการโจมตีมากกว่า 1 รูปแบบหรืออาจไม่มีรูปแบบการโจมตีก็ได้
- การเคลื่อนที่ใช้ระบุถึงวิธีที่ศัตรูจะเคลื่อนที่ไปในโลกของเกม มีทั้งหมด 3 รูปแบบเช่นกัน
- ตำแหน่งระบุตำแหน่งเริ่มต้นที่ศัตรูอยู่เมื่อเริ่มต้นเกม ซึ่งไม่เกี่ยวกับรูปแบบพฤติกรรม

จากองค์ประกอบทั้ง 3 ของศัตรูข้างต้น ส่งผลให้งานวิจัยนี้มีรูปแบบพฤติกรรมของศัตรูที่เป็นไปได้เพียง 24 รูปแบบเท่านั้นอันเป็นผลมาจากปริภูมิที่มีขนาดเล็ก

งานวิจัยทั้ง 3 งานที่ได้กล่าวไป มีจุดร่วมเดียวกัน คือ ความหลากหลายของศัตรูต่ำเนื่องจากตัวงานเน้นไปที่ภาพรวมของการออกแบบเกมอัตโนมัติมากกว่าการเจาะจงลงไปยังรายละเอียดเรื่องใดเรื่องหนึ่งของเกม ทำให้ขาดสัจพจน์ที่เหมาะสมกับการสร้างความหลากหลายให้พฤติกรรมของภาคี การปรับปรุงหรือออกแบบสัจพจน์ใหม่เพื่อสร้างปริภูมิของพฤติกรรมโดยเฉพาะสำหรับภาคีที่เป็นศัตรูจึงเป็นวิธีที่น่าจะแก้ไขข้อจำกัดนี้ได้

ในงานวิทยานิพนธ์นี้ใช้การเลือกพฤติกรรมในระดับกลางซึ่งได้จากแบบจำลองภาคีมาเรียงต่อกันภายใต้กฎและความยาวที่กำหนดแทนการเลือกกฎเพียงหนึ่งข้อจากหมวดหมู่ต่าง ๆ ทำให้รูปแบบพฤติกรรมทั้งหมดที่เป็นไปได้ซึ่งเกิดจากการเลือกและเรียงสับเปลี่ยนของพฤติกรรมมีความหลากหลายกว่างานวิจัยข้างต้น

Alexander Zook และ Mark O. Rield [30] เสนอการสร้างกลไกของเกมสำหรับเกมแบบผลัดกันเล่น (turn-based) เน้นตัวละคร โดยใช้ ASP ในการสร้างกลไกของเกมขึ้นจากโดเมนความรู้และตัวดำเนินการบนโดเมนความรู้ โดยกลไกที่ได้จะถูกตรวจสอบว่าถูกต้องตามเงื่อนไขการออกแบบ (Design requirement) จากนั้นใช้ Planner สำหรับตรวจสอบกลไกที่ได้ว่าเป็นไปตามเงื่อนไขจำกัดที่ผู้ใช้กำหนดหรือไม่

โดเมนความรู้ซึ่งในงานนี้เรียกว่าแบบจำลองสถานะ (State model) จะอยู่ในรูปของชนิดตัวละครที่ปรากฏอยู่ในเกมได้ ซึ่งแต่ละตัวละครจะมีรายการค่าพารามิเตอร์และช่วงค่าที่เป็นไปได้ เช่น ตัวละครชนิดผู้เล่น มีพารามิเตอร์พลังชีวิตที่มีค่าตั้งแต่ 0 ถึง 3 เป็นต้น โดเมนความรู้ต้องถูกกำหนดโดยผู้ใช้งานก่อนการสร้างกฎแบบอัตโนมัติ ส่งผลให้ผู้ใช้งานสามารถควบคุมลักษณะของกลไกของเกมผลลัพธ์ที่จะสร้างออกมาได้

หัวใจของงานนี้อยู่ที่แบบจำลองการเปลี่ยนสถานะ (Transition model) ซึ่งใช้อธิบายการเปลี่ยนแปลงของค่าพารามิเตอร์ของตัวละคร ภายใต้เงื่อนไขของเวลา ค่าพารามิเตอร์ของตัวละครอื่น ๆ และเงื่อนไขของการใช้งานกลไกอื่น การเปลี่ยนแปลงค่าพารามิเตอร์เป็นตัวดำเนินการระดับต่ำซึ่งประกอบด้วยการเพิ่มหรือลดค่าของพารามิเตอร์ สำหรับเงื่อนไขจะเป็นการเปรียบเทียบระหว่างพารามิเตอร์ที่สนใจโดยใช้ตัวเปรียบเทียบ เช่น เท่ากับ หรือมากกว่า เป็นต้นASP จะทำการสร้างกลไกโดยเลือกเงื่อนไขและการเปลี่ยนแปลงค่าพารามิเตอร์ที่ทำให้กลไกผลลัพธ์สามารถทำงานได้ถูกต้องตามเงื่อนไขการออกแบบ และ Planner จะช่วยตรวจสอบผลลัพธ์จาก ASP ให้เป็นไปตามเงื่อนไขที่ผู้ใช้ที่กำหนด ซึ่งในกรณีตัวอย่างที่ปรากฏในงานวิจัยนี้ได้ทดลองใช้ “เกมสามารถเล่นจนจบ” เป็นเงื่อนไข

ในทางทฤษฎี แนวคิดของงานวิจัยนี้สามารถครอบคลุมเกมแบบเน้นตัวละครได้โดยไม่ได้จำกัดว่าต้องเป็นประเภทใด ผู้ใช้งานเพียงกำหนดโดเมนความรู้ให้เหมาะสมกับประเภทเกมที่ต้องการ ก็น่าจะสร้างกลไกของเกมได้

รวมไปถึงพฤติกรรมที่ตายตัวของศัตรูด้วย แต่ในทางปฏิบัติงานนี้ยังมีข้อจำกัดหลายอย่างหากจะนำไปใช้กับการสร้างรูปแบบพฤติกรรมของศัตรูดังนี้:

- เนื่องจากตัวดำเนินการต่าง ๆ ในงานวิจัยนี้อยู่ในระดับต่ำมาก เพื่อที่จะสร้างพฤติกรรมที่มีความหมายอาจจะต้องใช้เงื่อนไขและการเปลี่ยนแปลงพารามิเตอร์จำนวนมากมาเรียงต่อกัน การค้นหารูปแบบพฤติกรรมของศัตรูที่เหมาะสมในปริภูมิน่าจะใช้เวลานาน เนื่องจากปริภูมิจะมีขนาดใหญ่ และคำตอบที่อยู่ในปริภูมิจะอยู่อย่างกระจัดกระจาย
- ตัวงานเน้นไปที่เกมแบบผลัดกันเล่น ซึ่งพฤติกรรมของตัวละครมักจะเกิดขึ้นและจบลงเป็นตา การอธิบายพฤติกรรมในเกมที่ไม่เป็นแบบผลัดกันเล่น เช่น เกมแอ็คชั่นทั่วไปที่ศัตรูสามารถเคลื่อนไหวอยู่ตลอดเวลา โดยใช้แบบจำลองการเปลี่ยนสถานะของงานนี้อาจทำได้ยาก หรือต้องใช้จำนวนของตัวดำเนินการเปลี่ยนแปลงพารามิเตอร์มาก
- ยังขาดการกระทำสำคัญที่พบในเกมแอ็คชั่นทั่วไป เช่น การเพิ่มตัวละครใหม่ลงในโลกของเกม (ในกรณีที่ศัตรูสามารถยิงกระสุนได้) หรือการลบตัวละครที่มีอยู่แล้วออกไป

Tobias Mahlmann [31] นำเสนอวิธีการสร้างเกมวางแผนการรบอัตโนมัติโดยนำเสนอแบบจำลองสำหรับอธิบายเกมวางแผนการรบแบบผลัดกันเล่นและสร้างเกมวางแผนการรบโดยการสุ่มและวิวัฒน์ยูนิต (Unit) ที่อธิบายด้วยแบบจำลองดังกล่าว

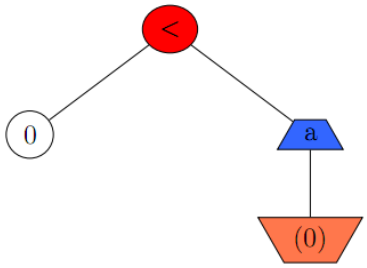
แบบจำลองของ Mahlmann อยู่ในรูปของภาษาอธิบายเกมวางแผนการรบ (Strategy game description language -- SGDL) ซึ่งอธิบายเกมวางแผนการรบโดยใช้ 4 องค์ประกอบ ได้แก่

- ยูนิต: เป็นตัวละครหน่วยหนึ่งบนสนามรบ ยูนิตมีได้หลายประเภท ยูนิตแต่ละประเภทจะมีค่าคุณสมบัติและความสามารถประจำตัว ซึ่งอยู่ในรูปของการกระทำที่สามารถทำได้
- สถานะของผู้เล่น: ประกอบด้วยค่าสถานะที่สำคัญสำหรับผู้เล่นแต่ละฝ่ายในเกมวางแผนการรบ เช่น เงิน
- สถานะของแผนที่: ประกอบด้วยค่าสถานะที่สำคัญของแผนที่ภายในเกม เช่น ยูนิตที่อยู่บนสนาม อุปสรรคบนแผนที่
- เงื่อนไขการจบเกม

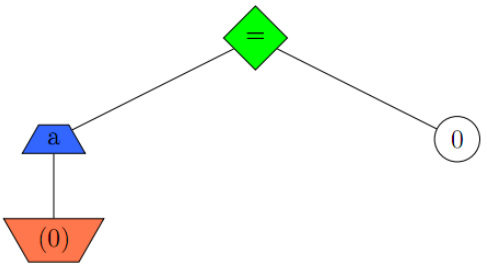
SGDL เป็นภาษาแบบ Strongly-typed และนำเสนอในรูปของต้นไม้ ปมภายในต้นไม้มีหลายชนิด แตกต่างกันไปตามข้อมูลที่อยู่ภายในปม ในการทบทวนวรรณกรรมนี้จะกล่าวถึงเฉพาะต้นไม้ย่อยในส่วนที่ใช้อธิบายยูนิตซึ่งเทียบเท่ากับศัตรูในงานวิทยานิพนธ์นี้

ต้นไม้ย่อยที่ใช้อธิบายยูนิตประกอบด้วยรายการค่าคุณสมบัติและรายการการกระทำ ผู้วิจัยระบุว่ารายการค่าคุณสมบัติเป็นรายการของคู่ระหว่างชื่อคุณสมบัติและค่าตัวเลขโดยไม่มีการแสดงลักษณะของต้นไม้ให้เห็นชัดเจน ส่วนรายการการกระทำประกอบด้วยต้นไม้ย่อยการกระทำ (Action sub-tree) หลายต้น แต่ละต้นไม้ย่อยจะมีรากเป็นปมชนิดการกระทำ (Action node) ซึ่งปมชนิดนี้จะมีต้นไม้ย่อยเป็นลูกได้เพียง 2 ชนิดได้แก่ ต้นไม้ย่อยเงื่อนไข (Condition sub-tree) และต้นไม้ย่อยผลลัพธ์ (Consequence sub-tree) ซึ่งเป็นการกระทำที่จะเกิดขึ้นได้เมื่อเงื่อนไขเป็นจริงเท่านั้น ในกรณีที่ปมการกระทำมีต้นไม้ย่อยผลลัพธ์มากกว่าหนึ่งต้น เส้นเชื่อมจากปมการกระทำไปยังต้นไม้ย่อยแต่ละต้นจะต้องมีป้ายกำกับลำดับการทำงานเพื่อกำหนดว่าจะให้ต้นไม้ย่อยผลลัพธ์ใดเริ่มทำงานก่อน

ต้นไม้ย่อยเงื่อนไขจะต้องมีรากเป็นปมชนิดตัวเปรียบเทียบ (Comparator node) เท่านั้น ซึ่งจะเปรียบเทียบระหว่างข้อมูลจากต้นไม้ย่อยลูกซ้ายและต้นไม้ย่อยลูกขวาของปม ภายในนั้น มีปมได้อีกหลายชนิด เช่น ปมค่าคงที่ (Constant node) ปมคุณสมบัติ (Property node) สำหรับเข้าถึงค่าคุณสมบัติของยูนิต และปมอ้างอิงวัตถุ (Object reference node) สำหรับเลือกยูนิตหรือวัตถุอื่น ๆ ในเกม ปมตัวดำเนินการ (Operator node) สำหรับคำนวณค่าทางคณิตศาสตร์หรือค่าตรรกะ เป็นต้น ปมต่าง ๆ ที่กล่าวถึงสามารถนำมาสร้างเป็นต้นไม้ที่คืนค่าข้อมูลสำหรับใช้ในการเปรียบเทียบ หากกล่าวโดยสรุป ต้นไม้ย่อยเงื่อนไขเทียบได้กับต้นไม้นิพจน์ที่คืนค่าจริงหรือเท็จ ตัวอย่างต้นไม้เงื่อนไขเป็นไปดังรูปที่ 3



รูปที่ 3 ตัวอย่างต้นไม้เงื่อนไข



รูปที่ 4 ตัวอย่างต้นไม้ย่อยผลลัพธ์ที่มีรากเป็นปมชนิดการกระทำ

ต้นไม้ย่อยผลลัพธ์มีรากเป็นปมชนิดตัวดำเนินการกำหนดค่าหรือปมชนิดการกระทำ ในกรณีที่เป็นปมชนิดตัวดำเนินการกำหนดค่า ปมดังกล่าวจะมีต้นไม้ย่อยลูกได้เพียง 2 ต้นเท่านั้นดังรูปที่ 4 โดยตัวดำเนินการนี้จะทำการประเมินค่าของต้นไม้ย่อยขวาและกำหนดค่าให้กับคุณสมบัติในต้นไม้ย่อยซ้าย

ในส่วนของการสร้างยูนิตแบบอัตโนมัติจะแบ่งออกเป็น 2 ตอน ได้แก่ การกำหนดค่าคุณสมบัติ และการสร้างพฤติกรรม การกำหนดค่าคุณสมบัติเป็นการสุ่มกำหนดค่าให้กับคุณสมบัติต่าง ๆ ของยูนิตแล้วใช้ขั้นตอนวิธีเชิงพันธุกรรมเพื่อวิวัฒน์ให้ค่าเหล่านั้นมีความเหมาะสม ความเหมาะสมวัดด้วยการเล่นเกมอัตโนมัติโดยใช้ปัญญาประดิษฐ์แบบ Monte-Carlo ในการควบคุมยูนิตที่ถูกสร้างขึ้นสำหรับผู้เล่นทุกฝ่าย การสร้างพฤติกรรมจะได้จากการวิวัฒน์การกระทำของยูนิตที่มีอยู่แล้วด้วยขั้นตอนวิธีเชิงพันธุกรรม ตัวดำเนินการกลายพันธุ์สำหรับการกระทำจะสุ่มลบปมและ/หรือต่อเติมต้นไม้ย่อยที่เป็นส่วนหนึ่งของต้นไม้ย่อยการกระทำ ตัวดำเนินการไขว้เปลี่ยนจะสุ่มเลือกปมจากสองต้นไม้ย่อยการกระทำแล้วสับเปลี่ยนต้นไม้ย่อยโดยมีปมที่เลือกเป็นราก

SGDL อธิบายพฤติกรรมด้วยระดับที่ต่ำกว่างานของ Togelius [27], Smith [28] และ Cook [29] ซึ่งมีข้อจำกัดในส่วนของความหลากหลายของพฤติกรรมที่เป็นไปได้ ปมตัวดำเนินการอยู่ในระดับต่ำเทียบเท่ากับงานของ Zook [30] แต่เนื่องจากการกำหนดประเภทของเกมไว้ ทำให้ค่าคุณสมบัติซึ่งเทียบได้กับแบบจำลองสถานะในงานของ Zook มีชุดกฎที่รองรับโดยเอ็นจิ้นแล้ว เช่น คุณสมบัติตำแหน่ง x และ y สามารถใช้อธิบายตำแหน่งในโลกโดยไม่ต้องมีการสร้างกฎขึ้นในภายหลัง การมีชุดกฎที่รองรับล่วงหน้าจึงส่งผลให้ขนาดของปริภูมิลดลงอย่างมาก

งานวิทยานิพนธ์นี้ใช้แนวคิดเดียวกับงานของ Mahlmann ในการสร้างรูปแบบพฤติกรรมของศัตรู นั่นคือสร้างแบบจำลองภาคีแล้วใช้แบบจำลองเป็นฐานความรู้ในการสร้างรูปแบบพฤติกรรม ทว่าเนื่องจากแบบจำลองในงานของ Mahlmann ออกแบบไว้สำหรับเกมวางแผนการรบแบบผลัดกันเล่น ซึ่งแตกต่างจากงานวิทยานิพนธ์นี้ที่เป็น

เกมแอ็คชั่นและลักษณะการเล่นแบบทันกาล (Real-time) ศัตรูในเกมแสดงรูปแบบพฤติกรรมตลอดเวลาโดยไม่มีการรอรับข้อมูลจากผู้เล่น งานวิทยานิพนธ์นี้จึงนำเสนอแบบจำลองภาคีซึ่งได้จากการวิเคราะห์เกมแอ็คชั่นที่มีอยู่

นอกจากนี้ ยูนิตที่สร้างขึ้นในงานของ Mahlmann มีจุดประสงค์เพื่อให้เกมมีความสมดุล ในขณะที่งานวิทยานิพนธ์นี้ต้องการสร้างรูปแบบพฤติกรรมของศัตรูที่ยอมรับได้โดยผู้เล่นซึ่งเป็นข้อมูลที่ขึ้นกับความชอบของบุคคล การหาฟังก์ชั่นความเหมาะสมสำหรับวัดค่าความพึงพอใจที่มีต่อรูปแบบของศัตรูโดยไม่มีข้อมูลสนับสนุนแรกเริ่ม จำเป็นจะต้องทำการทดลองกับผู้เล่นเป็นจำนวนมากคนละหลายครั้งเพื่อให้ได้ฟังก์ชั่นที่แม่นยำเพียงพอในการประเมินซึ่งระยะเวลาดังกล่าวน่าจะไม่เหมาะสมต่อกรอบเวลาของงานวิทยานิพนธ์ งานวิทยานิพนธ์นี้จึงเลือกใช้การทำเหมืองข้อมูลเข้ามาช่วยสกัดรูปแบบพฤติกรรมที่น่าสนใจและข้อมูลสนับสนุนอื่น ๆ จากชุดข้อมูล ชุดข้อมูลที่งานวิทยานิพนธ์นี้ใช้ประกอบด้วยศัตรูในเกมที่ได้รับความนิยมและเป็นที่ยอมรับของผู้เล่น ซึ่งถูกอธิบายในรูปของแบบจำลองภาคี

2.2.4 งานวิจัยเกี่ยวกับการออกแบบเกม

งานวิทยานิพนธ์นี้จำเป็นต้องสร้างแบบจำลองภาคีและการสร้างแบบจำลองจำเป็นต้องเข้าใจถึงองค์ประกอบต่าง ๆ ที่รวมกันเป็นสิ่งที่สนใจ ในที่นี้จึงสำรวจงานวิจัยทางด้านการออกแบบเกมซึ่งจะแสดงวิธีการวิเคราะห์เกมสำหรับศึกษาองค์ประกอบภายในเกม จุดประสงค์การสำรวจงานวิจัยกลุ่มนี้เป็นไปเพื่อศึกษาถึงวิธีที่ใช้วิเคราะห์เกมอย่างเป็นระบบที่มีการใช้งานอยู่แล้วในงานวิจัยอื่น และนำมาประยุกต์ใช้กับการวิเคราะห์ศัตรูของเกมแอ็คชั่นสองมิติแบบเน้นตัวละครในงานวิทยานิพนธ์นี้

Daniel Boutros [32] วิเคราะห์เกมใน 4 มุมมองได้แก่ ภาพ วิธีการควบคุม รางวัล และความท้าทาย โดยใช้การเล่นเกมที่ต้องการวิเคราะห์เป็นเวลา 10 นาที สาเหตุที่เลือกใช้ระยะเวลาสั้นเช่นนี้มาจากแนวคิดทางจิตวิทยาว่าด้วยความประทับใจแรกพบจะเกิดจากการพบเจอภายใน 5 วินาทีแรก แต่เนื่องจากระยะเวลา 5 วินาทีนั้นสั้นเกินไปจนไม่สามารถสัมผัสสิ่งที่เกมมีอย่างทั่วถึงเพียงพอ ผู้วิเคราะห์จึงเพิ่มระยะเวลาเป็น 10 นาที

Kenneth Hullett และ Jim Whitehead [33] วิเคราะห์แบบแผนการออกแบบ (Design pattern) เกมยิงปืนประเภทมุมมองบุคคลที่หนึ่งโดยการเล่นเกมแล้วบันทึกสิ่งที่พบของเกมโดยระบุว่าสิ่งที่พบดังกล่าวส่งผลต่อการกระทำของผู้เล่นอย่างไร หากสิ่งที่พบส่งผลให้การกระทำของผู้เล่นเป็นไปในทางเดียวกันจะนับเป็นแบบแผนการออกแบบเดียวกันและตั้งชื่อให้

Mark Nelson [34] และ Gillian Smith และคณะ [35] ต่างวิเคราะห์เกมด้วยวิธีการแยกย่อยองค์ประกอบโดยแบ่งเกมที่สนใจออกเป็นองค์ประกอบย่อยลงไปจนกระทั่งองค์ประกอบดังกล่าวส่งผลต่อวิธีการของผู้เล่นเพียงแบบใดแบบหนึ่งเท่านั้น

วิธีการวิเคราะห์เกมของ Hullett, Nelson และ Smith และคณะเป็นการหาองค์ประกอบที่ส่งผลต่อตัวผู้เล่นเพียงอย่างใดอย่างหนึ่งทั้งสิ้น เพียงแต่ Hullett และ Whitehead วิเคราะห์แบบรวมกลุ่มซึ่งอาศัยการเก็บรวบรวมข้อมูลจากการเล่น ในขณะที่ Nelson และ Smith และคณะใช้การวิเคราะห์แบบแยกย่อยโดยดูจากภาพรวมแล้วแบ่งองค์ประกอบย่อยลงไป งานวิทยานิพนธ์นี้อาศัยความคุ้นเคยต่อเกมประเภทแอ็คชั่นแบบเน้นตัวละครของผู้วิจัยร่วมกับการวิเคราะห์แยกย่อยในส่วนที่ไม่ใช่ศัตรูแต่เป็นองค์ประกอบที่เกี่ยวข้อง จากนั้นตั้งชื่อให้กับองค์ประกอบต่าง ๆ

2.2.5 งานวิจัยเกี่ยวกับการทำเหมืองข้อมูลสำหรับข้อมูลแบบลำดับ

ปัจจุบันมีขั้นตอนวิธีสำหรับการทำเหมืองข้อมูลสำหรับข้อมูลลำดับหลายวิธี ไลบรารี่ SPMF [36] รวบรวม 7 ขั้นตอนวิธีให้ใช้งาน โดยมีงานวิจัยของ Manika Verma และ Devarshi Mehta [37] ทดลองเปรียบเทียบประสิทธิภาพการทำงานของขั้นตอนวิธี GSP, SPADE และ PrefixSpan ภายในไลบรารี่ SPMF และพบว่า PrefixSpan เป็นขั้นตอนวิธีที่มีประสิทธิภาพสูงที่สุดทั้งในแง่ของระยะเวลาที่ใช้และปริมาณหน่วยความจำที่ต้องการ งานวิทยานิพนธ์จึงเลือกนำ PrefixSpan มาใช้สำหรับการทำเหมืองข้อมูล

PrefixSpan [8] เป็นขั้นตอนวิธีที่นำเสนอโดย Jian Pei และคณะ ขั้นตอนวิธีนี้หาลำดับย่อยที่พบบ่อยด้วยวิธีต่อเติมรูปแบบ (Pattern growth) โดยเริ่มต้นจากการนับค่าซัพพอร์ทของลำดับความยาว 1 แล้วแบ่งปริภูมิค้นหา (Divide search space) โดยการสร้างฐานข้อมูลภาพฉาย (Projected database) ของฐานข้อมูลต้นสำหรับทุกลำดับความยาว 1 ที่พบบ่อย (มีค่าซัพพอร์ทเกิน min_sup ที่กำหนด) ในขั้นตอนสุดท้ายจะเป็นการวนซ้ำเพื่อต่อเติมลำดับย่อยที่ได้จากการวนซ้ำรอบก่อนหน้าด้วยลำดับย่อยที่พบบ่อยใหม่ในการวนซ้ำรอบปัจจุบัน แล้วสร้างฐานข้อมูลภาพฉายใหม่ของฐานข้อมูลภาพฉายปัจจุบันสำหรับลำดับย่อยใหม่จนกระทั่งไม่สามารถสร้างฐานข้อมูลภาพฉายได้อีก ทุกลำดับย่อยที่พบบ่อยที่ได้จากการวนซ้ำทุกรอบเป็นคำตอบของการทำเหมืองข้อมูลนี้ รหัสเทียมของขั้นตอนวิธีเป็นไปดังขั้นตอนวิธีที่ 1

การทำเหมืองข้อมูลบนฐานข้อมูลลำดับ S ซึ่งประกอบด้วยลำดับ $s_1, s_2, ..., s_n$ และมีเซ็ตของไอเท็ม $I = \{i_1, i_2, ..., i_m\}$ กำหนดค่าซัพพอร์ทขั้นต่ำ $\min\_\sup$ ด้วยวิธี PrefixSpan เป็นดังนี้

Initialize $FrequentLength1 = \phi$, $\mathrm{Re}sult = \phi$
For (i in I) do:
if($\sup(i, S) \geq \min\_\sup$) do: $FrequentLength1, \mathrm{Re}sult \leftarrow i$
For(s in $FrequentLength1$) do:
Create s -projected database of S $proj(s, S)$
If($proj(s, S)$ is not empty) do: Recursive($s, proj(s, S)$)
Return $\mathrm{Re}sult$ as result

เมื่อฟังก์ชั่นวนซ้ำ Recursive(Sequence s , Projected database P) มีรหัสเทียมดังนี้

Initialize $\mathrm{Re}cursiveLength1 = \phi$
For(is in I) do: if($\sup(is, P) \geq \min\_\sup$) $\mathrm{Re}cursiveLength1 \leftarrow i$
For(ss in $\mathrm{Re}cursiveLength1$) do:
$\mathrm{Re}sult \leftarrow s + ss$
Create ss -projected database of P $proj(ss, P)$
If($proj(ss, P)$ is not empty) Recursive($ss, proj(ss, P)$)

และมีนิยามที่สำคัญต่อไปนี้

- $\alpha = \langle a_1, a_2, ..., a_m \rangle$ เป็นตัวเติมหน้า (Prefix) ของลำดับ $\beta = \langle b_1, b_2, ..., b_n \rangle$ กำหนดให้ $m \le n$ ก็ต่อเมื่อ
  - $a_i = b_i; i \le m-1$
  - $a_m \subseteq b_m$ ซึ่งไอเท็มทั้งหมดของ a_m จะต้องมาก่อน $b_m - a_m$ ตามลำดับอักษร เช่น $a_m = (AB)$ และ $b_m = (ABCD)$ เป็นต้น
- $\gamma = \langle e''_m, e_{m+1}, ..., e_n \rangle$ เป็นตัวเติมหลัง (Suffix) ของลำดับ $\beta = \langle e_1, e_2, ..., e_n \rangle$ เมื่อพิจารณาลำดับ $\alpha = \langle e_1, e_2, ..., e_{m-1}, e'_m \rangle$ กำหนดให้ $m \le n$ และ $e''_m = e_m - e'_m$
  - หาก a''_m ไม่เป็นไอเท็มเซ็ตว่าง ให้ใช้สัญลักษณ์ $\_$ เป็นไอเท็มแรกของไอเท็มเซ็ต เช่น $(\_CD)$
- ฐานข้อมูลภาพฉายสำหรับลำดับ α ของ D (α-projected database of D) เมื่อ D เป็นฐานข้อมูลลำดับ ซึ่งประกอบด้วยลำดับ $d_1, d_2, ..., d_n$ ประกอบด้วยลำดับย่อย $a_i; 1 \le i \le n$ ซึ่ง
  - a_i เป็นตัวเติมหลังของ d'_i เมื่อพิจารณาลำดับ α
  - d'_i เป็นลำดับย่อยของ d_i ซึ่งเลือกลำดับย่อยโดยเริ่มจาก α ที่ปรากฏครั้งแรกใน d_i จนจบลำดับ d_i
  - ตัวอย่าง เช่น $\alpha = \langle AC \rangle, d_1 = \langle A(BE)A(CD)F \rangle$ จะได้ $d_1 = \langle (\_D)F \rangle$ เป็นต้น

ขั้นตอนวิธีที่ 1 PrefixSpan

งานวิทยานิพนธ์นี้ใช้ PrefixSpan ในการหารูปแบบพฤติกรรมที่น่าสนใจจากชุดข้อมูลรูปแบบพฤติกรรมที่ถูกแปลงให้อยู่ในรูปของลำดับแล้ว นอกจากนี้ ชุดข้อมูลรูปแบบพฤติกรรมจะถูกแปลงให้อยู่ในรูปลำดับเชื่อม (Joined sequence) ซึ่งงานวิทยานิพนธ์นี้จะปรับปรุงขั้นตอนวิธี PrefixSpan ให้เหมาะสมกับลักษณะของข้อมูลดังกล่าว รายละเอียดทั้งหมดจะกล่าวถึงในหัวข้อ 5.3.2

นอกจาก PrefixSpan แล้ว ในที่นี้ได้ทำการสำรวจงานวิจัยที่เกี่ยวข้องกับการทำเหมืองข้อมูลเพื่อหารูปแบบของการเรียกใช้ฟังก์ชั่น (Function call pattern) เนื่องจากการทำเหมืองข้อมูลบนชุดข้อมูลรูปแบบพฤติกรรมที่แปลงเป็นลำดับมีจุดประสงค์เพื่อหารูปแบบของการกระทำซึ่งอยู่ในรูปของฟังก์ชั่น

Takashi Ishio และคณะ [38] ทำเหมืองข้อมูลเพื่อหารูปแบบของการเรียกใช้ฟังก์ชั่นสำหรับไลบรารี่ที่เขียนขึ้นด้วยภาษาจาวา (Java) โดยแปลงซอร์สโค้ด (Source code) ให้อยู่รูปของลำดับแล้วใช้ขั้นตอนวิธี PrefixSpan ซึ่งปรับปรุงแล้วในการทำเหมืองข้อมูล

ลำดับซอร์สโค้ดของงานวิจัยนี้ประกอบด้วยสัญพจน์สองชนิด ได้แก่ เอเลเมนท์เมธอด (Method call element) และเอเลเมนท์ควบคุม (Control element) โดยวิธีการสร้างลำดับจะได้จากการพิจารณาลำดับการทำงานของซอร์สโค้ด เมื่อพบการเรียกใช้เมธอด จะเพิ่มเอเลเมนท์เมธอดเข้าไปในลำดับ เมื่อพบคำสงวนสำหรับโครงสร้างควบคุม เช่น if หรือ for จะเพิ่มเอเลเมนท์ควบคุมเข้าไปในลำดับ (ในที่นี้ คือ IF และ FOR) โดยมีการใช้งานตัวปิดบล็อก เช่น ENDIF, END-LOOP ร่วมด้วย เอเลเมนท์ควบคุมแต่ละตัวภายในลำดับจะรู้ว่าตัวปิดบล็อกที่คู่กับตนเองคืออะไรโดยการตรวจสอบต้นไม้วากยสัมพันธ์นามธรรม (Abstract syntax tree -- AST) และเมื่อขั้นตอนวิธี PrefixSpan พบรูปแบบที่ประกอบด้วยเอเลเมนท์ควบคุม ก็จะดึงตัวปิดบล็อกที่คู่กันมาไว้ในรูปแบบด้วย

ข้อจำกัดของงานวิจัยนี้ คือ ในกรณีที่มีเอเลเมนท์เมธอดอยู่ภายใต้โครงสร้างควบคุม 2 ชั้น และโครงสร้างควบคุมในชั้นที่ 2 ไม่ปรากฏบ่อยเพียงพอ จะส่งผลให้รูปแบบที่ได้จากการทำเหมืองข้อมูลขาดข้อมูลของโครงสร้างชั้นที่ 2 และไม่มีทางรู้ได้ว่าเอเลเมนท์เมธอดเป็นเมธอดที่อยู่ในโครงสร้างข้อมูล 2 ชั้น ซึ่งการสูญเสียข้อมูลเช่นนี้จะส่งผลต่อรูปแบบพฤติกรรมหากนำวิธีเดียวกันนี้มาใช้ งานวิทยานิพนธ์นี้จึงใช้วิธีการติดป้ายข้อมูลโครงสร้างควบคุม (ซึ่งในกรณีของงานวิทยานิพนธ์นี้ มีเพียงโครงสร้างเงื่อนไข) ให้กับการกระทำ ทำให้สามารถเก็บข้อมูลไว้ได้ครบถ้วนกว่า

บทที่ 3
ภาพรวมของงานวิทยานิพนธ์

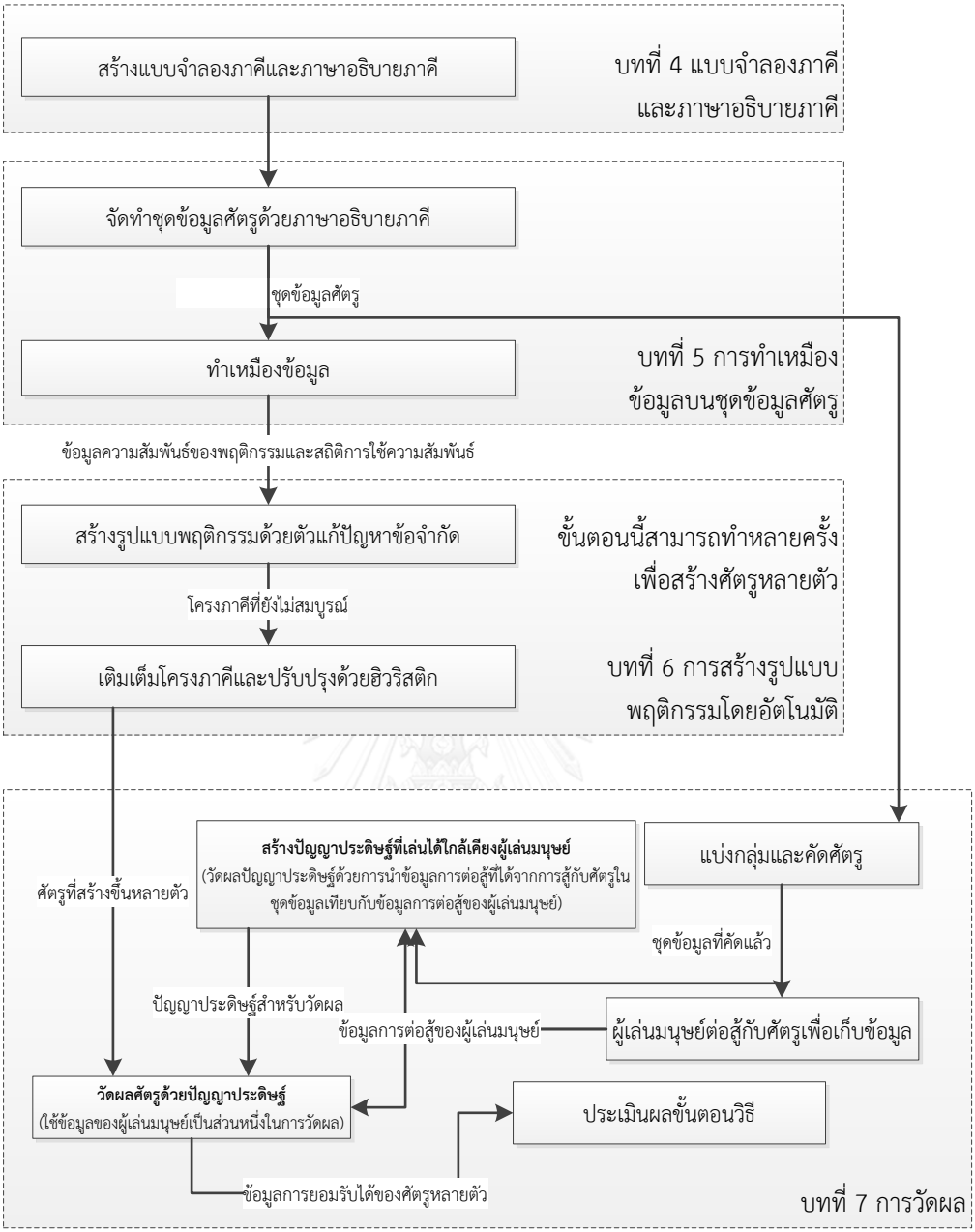
จุดประสงค์ของงานวิทยานิพนธ์นี้คือพัฒนาวิธีการสร้างรูปแบบพฤติกรรมที่ตายตัวของศัตรูแบบอัตโนมัติสำหรับเกมแอ็คชั่นสองมิติแบบเน้นตัวละคร จากการค้นคว้างานวิจัยที่เกี่ยวข้อง พบว่าไม่มีงานวิจัยที่เกี่ยวข้องกับการสร้างรูปแบบพฤติกรรมที่ตายตัวของศัตรูในเกมประเภทนี้โดยตรง งานวิทยานิพนธ์นี้จึงต้องเริ่มต้นจากการกำหนดนิยามของศัตรูและพฤติกรรมที่เป็นไปได้โดยใช้การวิเคราะห์เพื่อศึกษาองค์ประกอบของเกม แล้วจึงสร้างแบบจำลองของศัตรู (Enemy model) ซึ่งจะถูกปรับปรุงไปเป็นแบบจำลองภาคี (Agent model) เพื่อความครอบคลุมของพฤติกรรม แบบจำลองภาคีจะถูกนำเสนอในรูปของภาษาอธิบายภาคี (Agent description language) เพื่อให้ง่ายต่อการทำความเข้าใจสำหรับมนุษย์ และนำไปใช้งานได้โดยอาศัยตัวแปลภาษา รายละเอียดของแบบจำลองและภาษาอธิบายภาคีจะกล่าวถึงในบทที่ 4

เมื่อได้ภาษาอธิบายภาคีแล้ว จึงสร้างชุดข้อมูลศัตรูด้วยภาษาดังกล่าว โดยข้อมูลของศัตรูได้มาจากเกมที่ได้รับความนิยม ชุดข้อมูลดังกล่าวนี้จะถูกใช้เป็นข้อมูลขาเข้าสำหรับการทำเหมืองข้อมูลแบบต่างๆ เพื่อหาความสัมพันธ์ของพฤติกรรมที่น่าจะส่งผลให้ศัตรูเป็นที่ยอมรับของผู้เล่น รวมถึงสถิติการใช้งานความสัมพันธ์ในชุดข้อมูลขาเข้า รายละเอียดของชุดข้อมูลและการทำเหมืองข้อมูลจะกล่าวถึงในบทที่ 5

ความสัมพันธ์ของพฤติกรรมและสถิติที่ได้จากการทำเหมืองข้อมูลจะถูกใช้ในการสร้างรูปแบบพฤติกรรมใหม่ด้วยการเติมความสัมพันธ์ลงในโครงภาคี โดยวิธีการเติมความสัมพันธ์นี้จะถูกแปลงให้เป็นปัญหาการตอบสนองเงื่อนไขบังคับ และแก้ด้วยตัวแก้ปัญหาเงื่อนไขบังคับ รูปแบบพฤติกรรมที่ได้จะถูกเติมเต็มค่าคุณสมบัติที่ขาดหายไปให้กลายเป็นศัตรูที่สามารถนำไปใช้งานในเกมทดสอบได้ รวมถึงมีการปรับปรุงภาคีที่ได้ด้วยฮิวริสติกเพื่อให้ภาคีมีโอกาสเป็นภาคีที่ยอมรับได้มากขึ้น รายละเอียดการสร้างรูปแบบพฤติกรรมและศัตรูจากความสัมพันธ์จะกล่าวถึงในบทที่ 6

ศัตรูที่สร้างขึ้นจะถูกนำไปวัดการยอมรับได้ตามนิยามที่กำหนดโดยการทดลองต่อสู้กับปัญญาประดิษฐ์วัดผลในเกมทดสอบ ข้อมูลการยอมรับได้ของศัตรูหลายตัวจะถูกใช้ในการสรุปประสิทธิผลของขั้นตอนวิธี สำหรับปัญญาประดิษฐ์วัดผลที่ใช้งานในขั้นตอนนี้จะถูกออกแบบให้เล่นได้ประสิทธิภาพใกล้เคียงกับผู้เล่นเพื่อให้สามารถวัดผลแทนผู้เล่นมนุษย์ได้ การวัดผลปัญญาประดิษฐ์จะใช้วิธีเทียบข้อมูลการต่อสู้ของปัญญาประดิษฐ์กับข้อมูลการต่อสู้ของผู้เล่นมนุษย์ โดยข้อมูลทั้งสองจะได้จากการต่อสู้กับศัตรูที่ถูกคัดมาจากชุดข้อมูลศัตรู รายละเอียดของปัญญาประดิษฐ์และการวัดผลจะกล่าวถึงในบทที่ 7

ภาพรวมของขั้นตอนทั้งหมดของงานวิทยานิพนธ์เป็นไปดังรูปที่ 5



รูปที่ 5 ผังงานแสดงขั้นตอนของงานวิทยานิพนธ์

บทที่ 4
แบบจำลองภาคีและภาษาอธิบายภาคี

งานวิทยานิพนธ์นี้สร้างแบบจำลองภาคีขึ้นจากการวิเคราะห์ศัตรูและองค์ประกอบอื่น ๆ ที่เกี่ยวข้องที่ปรากฏภายในเกมแอ็คชั่นสองมิติแบบเน้นตัวละครจำนวน 5 เกม ได้แก่ Metroid, Super C, Megaman, Megaman 4 และ Shovel Knight ซึ่งแต่ละเกมเป็นเกมที่ได้รับความนิยมในยุคสมัยที่เกมนั้นได้รับการผลิตออกมา นอกจากนี้ช่วงเวลาที่แต่ละเกมถูกผลิตออกมานั้นมีตั้งแต่ปี ค.ศ. 1968 จนถึง ค.ศ. 2014 จึงน่าจะทำให้รูปแบบพฤติกรรมที่ได้จากการวิเคราะห์สามารถครอบคลุมแนวคิดของการออกแบบในต่างช่วงเวลาได้

แบบจำลองภาคีจะอยู่ในรูปของภาษาอธิบายภาคี ซึ่งจุดแข็งของการใช้ภาษาอธิบายภาคีเป็นแบบจำลองนั้นทำให้มนุษย์สามารถทำความเข้าใจแบบจำลองได้ง่าย รวมไปถึงสามารถสร้างตัวแปลภาษาขึ้นมาเพื่อแปลแบบจำลองและแสดงผลลัพธ์ในเกมได้ ในหัวข้อนี้จึงได้อธิบายขั้นตอนการสร้างแบบจำลองตั้งแต่แบบจำลองศัตรูไปจนถึงการปรับปรุงเป็นแบบจำลองภาคี รวมถึงรายละเอียดของภาษาอธิบายภาคี

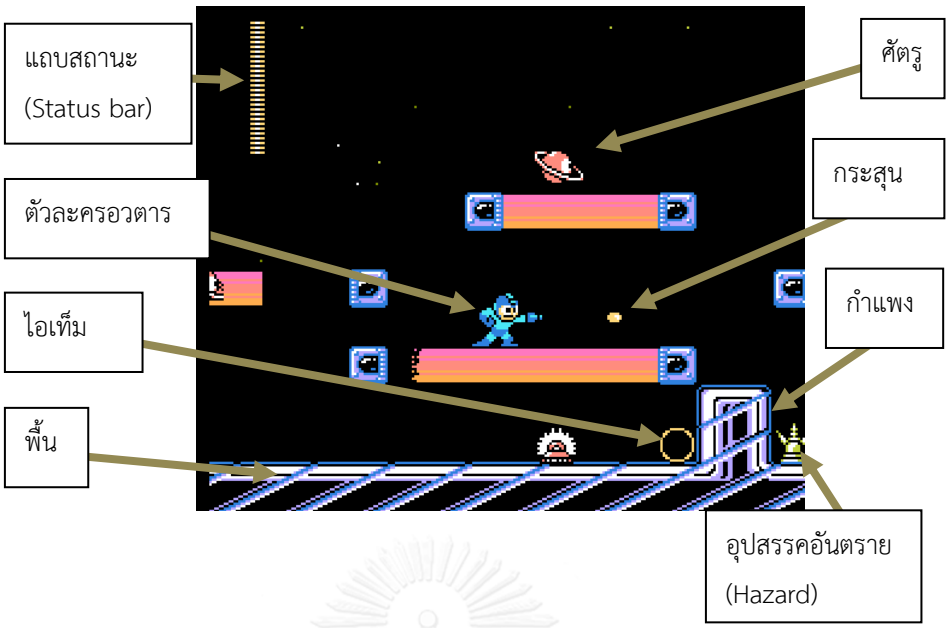
4.1 การวิเคราะห์ศัตรู

ก่อนเริ่มต้นวิเคราะห์ศัตรู ในที่นี้จำเป็นต้องให้นิยามของ “ศัตรู” ในบริบทของงานวิทยานิพนธ์นี้เพื่อแยกศัตรูออกจากองค์ประกอบอื่นของเกมเสียก่อน ในงานวิจัยเกี่ยวกับเกมแพลทฟอร์มของ Smith [35] ได้กำหนดให้ศัตรูเป็นส่วนหนึ่งของสิ่งกีดขวาง (Obstacle) โดยนิยามของสิ่งกีดขวางภายในงานวิจัยดังกล่าว คือ “องค์ประกอบที่สามารถขัดขวางการเคลื่อนไหวของตัวละครอวตารหรือสามารถสร้างความเสียหายให้กับตัวละครอวตารได้” หนังสือของ Adams [3] นิยามศัตรูในเกมยุคเก่าไว้ว่า “ศัตรูเป็นภาคีที่สามารถโจมตีตัวละครอวตารและเคลื่อนที่ด้วยรูปแบบที่ตายตัวที่ผู้เล่นสามารถเรียนรู้เพื่อหลบหลีกได้” เมื่อพิจารณานิยามของทั้งสองงานร่วมกัน งานวิทยานิพนธ์นี้จึงนิยามศัตรูว่า

“ศัตรู คือ ภาคีที่มีรูปแบบพฤติกรรมและสร้างความเสียหายให้ตัวละครอวตารได้”

นิยามส่วนที่ระบุว่าศัตรูจะต้องสร้างความเสียหายได้ มีจุดประสงค์เพื่อคัดกรองสิ่งกีดขวางอื่น ๆ ตามนิยามของ Smith เช่น กำแพง ซึ่งมีผลในการขัดขวางการเคลื่อนที่ของผู้เล่นเท่านั้น ในทางกลับกัน นิยามที่ระบุว่าศัตรูคือภาคีที่มีรูปแบบพฤติกรรมจะทำให้ครอบคลุมศัตรูที่ไม่เคลื่อนที่ เช่น ศัตรูรูปแบบป้อมปืน ซึ่งจะไม่ถูกนับเป็นศัตรูหากยึดตามนิยามของ Adams

การวิเคราะห์ศัตรูในเกมแอ็คชั่นทั้ง 5 เกมที่ถูกเลือกจะเริ่มต้นจากการค้นหาองค์ประกอบที่เป็นศัตรูตามนิยามที่กำหนด จากนั้นพิจารณาองค์ประกอบที่มีส่วนเกี่ยวข้องกับศัตรูและแยกย่อยองค์ประกอบดังกล่าวให้เหลือเพียงองค์ประกอบที่เกี่ยวข้องกับศัตรูโดยตรง รูปที่ 6 แสดงองค์ประกอบที่เป็นรูปธรรมส่วนหนึ่งของเกม Megaman 4



รูปที่ 6 องค์ประกอบส่วนหนึ่งของเกม Megaman 4

4.2 องค์ประกอบที่เกี่ยวข้องกับศัตรู

พฤติกรรมของศัตรูย่อมส่งผลหรือได้รับผลกระทบจากองค์ประกอบที่เกี่ยวข้อง เช่น กำแพงอาจจะขัดขวางการเคลื่อนที่ของศัตรูและส่งผลให้ศัตรูเปลี่ยนทิศทางการเคลื่อนที่ เป็นต้น การทำความเข้าใจถึงองค์ประกอบเหล่านี้จึงมีบทบาทสำคัญในการสร้างแบบจำลองศัตรู รวมไปถึงการสร้างเกมทดสอบสำหรับใช้วัดผลศัตรูซึ่งจำเป็นต้องควบคุมองค์ประกอบที่เกี่ยวข้องให้คงที่ตลอดทุกการทดสอบ

จากนิยาม “ศัตรู คือ ภาคีที่มีรูปแบบพฤติกรรมและสร้างความเสียหายให้ตัวละครอวตารได้” จะพบคำสำคัญได้แก่ รูปแบบพฤติกรรม, ความเสียหาย และตัวละครอวตาร ซึ่งทำให้ค้นพบองค์ประกอบที่เกี่ยวข้องได้แก่ โครงสร้างพื้นที่ (Level layout), กลไกการต่อสู้ (Battle mechanic) และตัวละครอวตาร (Avatar)

4.2.1 โครงสร้างพื้นที่

รูปแบบพฤติกรรมกล่าวถึงความสามารถในการกระทำการบางอย่างในโลกของเกม หากยกตัวอย่างเกม Megaman 4 จะมีศัตรูบางชนิดสามารถเดินไปบนพื้น หรือในเกม Super C ซึ่งกระสุนของศัตรูบางชนิดไม่สามารถเคลื่อนที่ผ่านกำแพงได้ นอกจากนี้ ในทุกเกมที่ทำการวิเคราะห์ ตัวละครอวตารจะถูกกำแพงหรือพื้นกำหนดบริเวณที่สามารถเข้าถึงได้เช่นกัน ลักษณะดังกล่าวนี้แสดงให้เห็นถึงองค์ประกอบที่มีหน้าที่ระบุถึงขอบเขตพื้นที่ที่สามารถผ่านได้หรือผ่านไม่ได้ ในที่นี้จึงกำหนดให้องค์ประกอบที่ทำหน้าที่เช่นนี้เป็น “โครงสร้างพื้นที่”

โครงสร้างพื้นที่ในเกมที่นำมาวิเคราะห์แบ่งออกได้เป็น 2 ประเภท ได้แก่ พื้นที่ที่ผ่านได้ และพื้นที่ที่ผ่านไม่ได้ ในบางเกม เช่น Super C ศัตรูบางชนิดสามารถเคลื่อนที่ผ่านบริเวณที่ตัวละครอวตารผ่านไม่ได้ ในขณะที่ศัตรูอื่นอีกหลายชนิดไม่สามารถทำเช่นเดียวกันนี้ นั่นแสดงให้เห็นว่า ศัตรูแต่ละชนิดมีรายการที่ระบุว่าตนสามารถเคลื่อนผ่านพื้นที่บริเวณประเภทใดได้บ้าง ในงานวิทยานิพนธ์นี้จะกำหนดให้เกมทดสอบมีโครงสร้างพื้นที่ 2 ประเภทดัง

ข้างต้น และเพิ่มคุณสมบัติให้กับศัตรูเพื่อแยกแยะบริเวณที่ผ่านได้หรือไม่ได้ โดยไม่ใช้ข้อมูลในรูปแบบรายการ ค่าคุณสมบัติที่เกี่ยวข้องกับโครงสร้างพื้นที่จะกล่าวถึงในหัวข้อ 4.3.1 อีกครั้งหนึ่ง

4.2.2 กลไกการต่อสู้

ความเสียหายในเกมแอ็คชั่นแบบเน้นตัวละคร คือ การสูญเสียค่าพลังชีวิต หากพลังชีวิตหมดจะทำให้ตัวละครอวตารหรือศัตรูนั้นตาย ในกรณีของตัวละครอวตาร แต่ละเกมจะมีวิธีจัดการกับตัวละครอวตารที่ตายแตกต่างกัน เช่น ในเกม Megaman ผู้เล่นจะถูกนำไปกลับไปยังจุดที่กำหนด หรือในเกม Metroid เกมจะจบลงทันทีเมื่อตาย ในกรณีของศัตรู ศัตรูจะถูกนำออกจากโลกของเกมไป ความเสียหายต่อตัวละครอวตารและศัตรูอาจเกิดจากอุปสรรคอันตรายหรือการต่อสู้ระหว่างภาคีก็ได้ แต่เนื่องจากงานวิทยานิพนธ์นี้ไม่ได้นำลักษณะของพื้นที่นอกจากคุณสมบัติการผ่านได้ของพื้นที่มาคิด ดังนั้นกรณีของอุปสรรคอันตรายจึงถูกตัดออกจากเรื่องการคิดความเสียหาย

การต่อสู้ระหว่างตัวละครอวตารและศัตรูในเกมตัวอย่าง หากมองจากมุมมองของผู้เล่น จะพบลักษณะการโจมตี 2 ประเภท ได้แก่ การโจมตีระยะใกล้ (Melee attack) และการโจมตีระยะไกล (Ranged attack) โดยตัดสินจากระยะหวังผลของการโจมตี เช่น ในเกม Shovel Knight ตัวละครอวตารจะโจมตีโดยใช้พลั่วแทงออกไปด้านหน้า ในมุมมองของผู้เล่นที่เห็นภาพเช่นนี้จะมองว่าเป็นการโจมตีระยะใกล้ ในทางตรงข้าม ตัวละครในเกมอื่นที่วิเคราะห์ซึ่งโจมตีด้วยการยิงกระสุนจะถูกมองว่าเป็นการโจมตีระยะไกลแทน การโจมตีสองประเภทในมุมมองของผู้เล่นนี้สามารถทำให้เหลือรูปแบบเดียวโดยการอธิบายด้วยตัวตรวจจับการชน (Collider) แทน เมื่อตัวตรวจจับการชนของการโจมตีกระทบกับตัวตรวจจับการชนของเป้าหมาย จะถือว่าเป้าหมายถูกโจมตีและเสียพลังชีวิต ทั้งหมดที่กล่าวมาล้วนเป็นสิ่งที่เรียกว่ากลไกการต่อสู้ รายละเอียดของกลไกจะกล่าวถึงในหัวข้อ 4.3 อีกครั้งหนึ่ง

4.2.3 ตัวละครอวตาร

ตัวละครอวตารเป็นภาคีที่ถูกควบคุมโดยผู้เล่น ในเกมแอ็คชั่นแบบเน้นตัวละคร พฤติกรรมที่สามารถกระทำได้ของตัวละครผู้เล่นจะถูกกำหนดไว้แล้ว สำหรับเกมตัวอย่างที่วิเคราะห์ พฤติกรรมที่สามารถกระทำได้ของตัวละครอวตารจะใกล้เคียงกันทั้งหมด ผู้เล่นสามารถควบคุมให้ตัวละครเดินไปทางซ้ายและขวาได้ สามารถสั่งให้ตัวละครกระโดดและโจมตีได้ นอกเหนือจากนี้ แต่ละเกมจะมีชุดพฤติกรรมเฉพาะตัวที่แตกต่างกันออกไปบ้าง เช่น ใน Super C จะมีบางฉากที่ตัวละครอวตารไม่สามารถกระโดดได้ แต่ผู้เล่นสามารถควบคุมตัวละครให้เดินไปในทิศซ้าย-ขวา-บน-ล่างแทน หรือในเกม Megaman 4 ตัวละครจะสามารถเคลื่อนที่แบบไถลได้

ความเกี่ยวพันระหว่างศัตรูและตัวละครอวตารจะอยู่ใน 2 ลักษณะ ได้แก่ ความเป็นพวกพ้องเมื่อเกิดการต่อสู้ระหว่างภาคี และพฤติกรรมของศัตรูที่ส่งผลหรือได้รับผลกระทบจากการเคลื่อนไหวของตัวละครอวตาร ในกรณีแรกจะกล่าวถึงว่าฝ่ายใดสามารถทำการโจมตีฝ่ายใดได้ ทำให้แบบจำลองภาคีต้องมีคุณสมบัติสำหรับการแบ่งฝ่าย ผลของคุณสมบัติดังกล่าวจะกล่าวถึงในหัวข้อ 4.3 สำหรับกรณีของผลกระทบต่อพฤติกรรมของศัตรูจะทำให้เกิดการกระทำและข้อมูลแบบจำเพาะบางชนิด เช่น ศัตรูที่เคลื่อนที่ตามทิศการเคลื่อนที่ของผู้เล่น เป็นต้น

4.3 องค์ประกอบของศัตรู

ในการวิเคราะห์ศัตรู ผู้วิจัยแบ่งศัตรูออกเป็นองค์ประกอบย่อย โดยเริ่มต้นจากค่าคุณสมบัติ (Property) ที่จำเป็นต่อตัวศัตรูซึ่งเกิดจากองค์ประกอบอื่น ๆ ที่เกี่ยวข้อง เช่น ค่าคุณสมบัติสำหรับการแยกแยะความสามารถในการผ่านบริเวณเกิดจากโครงสร้างพื้นที่ เป็นต้น จากนั้นจึงวิเคราะห์ว่าศัตรูมีการเปลี่ยนแปลงค่าคุณสมบัติต่าง ๆ ไปอย่างไร และรวบรวมวิธีการเปลี่ยนแปลงค่าคุณสมบัติเหล่านั้นเป็นการกระทำ (Action) โดยการกระทำของศัตรูจะถูกการจัดลำดับว่าให้ทำสิ่งใดก่อนสิ่งใดหลัง รวมถึงมีเงื่อนไขของการกระทำบางอย่าง และการเปลี่ยนชุดการกระทำทำให้เกิดเป็นองค์ประกอบที่ว่าด้วยลำดับพฤติกรรม (Behavior sequence), การทดสอบเงื่อนไข (Condition testing), การสอบถามข้อมูล (Information query) และเรื่องของสถานะ (State) และลำดับพฤติกรรมคู่ขนาน (Parallel sequence)

4.3.1 ค่าคุณสมบัติ (Property)

รูปแบบพฤติกรรมของศัตรูที่พบจากการสังเกตจะสามารถเปลี่ยนแปลงค่าคุณสมบัติบางอย่าง ค่าคุณสมบัติดังกล่าวอาจจะเป็นค่าคุณสมบัติที่ผู้เล่นสามารถรับรู้ได้อย่างชัดเจน เช่น ตำแหน่งของศัตรูในโลก หรือคุณสมบัติที่ไม่ชัดเจน เช่น ตัวนับจำนวนการกระทำสำหรับใช้ในศัตรูบางตัวที่มีรูปแบบพฤติกรรมเปลี่ยนแปลงได้หลังจากกระทำพฤติกรรมบางอย่างซ้ำกันจนครบตามจำนวนที่กำหนด คำถามสำคัญสำหรับประเด็นนี้คือ “มีสิ่งใดที่เป็นค่าคุณสมบัติของศัตรูบ้าง?”

ในที่นี้จะทำการวิเคราะห์ค่าคุณสมบัติที่จำเป็นต้องมีโดยพิจารณาจากความสัมพันธ์ระหว่างศัตรูและองค์ประกอบที่เกี่ยวข้องทั้ง 3 ได้แก่ โครงสร้างพื้นที่, กลไกการต่อสู้ และตัวละครอวตาร

ดังที่ได้กล่าวไปแล้วในหัวข้อ 4.2.1 ว่า โครงสร้างพื้นที่มีผลในการจำกัดบริเวณที่ตัวละครอวตารและศัตรูสามารถเคลื่อนผ่านไปได้ โดยพื้นที่จะถูกแบ่งเป็น 2 ประเภทได้แก่ พื้นที่ที่ผ่านได้ และพื้นที่ที่ผ่านไม่ได้ แต่เนื่องจากในเกมที่วิเคราะห์มีศัตรูหลายตัวที่สามารถเคลื่อนที่ผ่านพื้นที่ที่ผ่านไม่ได้ เช่น ศัตรู Alien Egg ใน Super C, Taketento ใน Megaman 4 และ Swarm ใน Metroid เป็นต้น ในที่นี้จึงกำหนดให้มีค่าคุณสมบัติ “ผ่านได้” (Through) ที่ทำให้ศัตรูสามารถผ่านได้ทุกพื้นที่หากมีคุณสมบัตินี้ นอกเหนือจากนี้ เนื่องจากเกมนี้เป็นเกมแอ็คชั่นสองมิติ การตรวจสอบว่าศัตรูที่กำหนดอยู่ในบริเวณพื้นที่ส่วนใดจึงต้องมีข้อมูลตำแหน่งและขนาดของศัตรู รวมไปถึงการบอกขนาดและตำแหน่งของพื้นที่ประเภทต่าง ๆ งานวิทยานิพนธ์นี้รวมคุณสมบัติตำแหน่งและขนาดเข้าด้วยกันและเรียกว่า “ตัวตรวจจับการชน” (Collider) เป็นอีกค่าคุณสมบัติของศัตรู ซึ่งตัวละครอวตารและโครงสร้างพื้นที่เองต่างก็ใช้คุณสมบัตินี้ในการกำหนดขอบเขตเช่นเดียวกัน

เกมแอ็คชั่นถูกแบ่งเป็นประเภทย่อยหลายประเภทและหนึ่งในประเภทย่อยคือเกมแพลทฟอร์ม ซึ่งเกมตัวอย่างเองก็เป็นเกมแพลทฟอร์มเป็นส่วนใหญ่ จุดที่เป็นกุญแจสำคัญอีกจุดหนึ่งของโครงสร้างพื้นที่ของเกมแพลทฟอร์มที่ทำให้เกิดลักษณะการเล่นแบบกระโดดไปตามพื้นต่างระดับ คือ แรงดึงดูด ซึ่งคอยดึงให้ตัวละครอวตารหรือศัตรูที่กระโดดกลับลงมาบนพื้น ทว่าในเกมแพลทฟอร์มมักจะมีศัตรูที่สามารถลอยได้โดยไม่สนใจแรงดึงดูด เช่น ใน Megaman 4 ศัตรู Battonton สามารถบินเข้าหาตัวละครอวตาร หรือศัตรู M-422A ลอยค้างอยู่กลางอากาศได้ เป็นต้น นั่นแสดงให้เห็นความจำเป็นของค่าคุณสมบัติ “ผลของแรงดึงดูด” (Gravity effect) สำหรับใช้ระบุว่าศัตรูที่กำหนดรับผลจากแรงดึงดูดมากเพียงใด

กลไกการต่อสู้เป็นอีกหนึ่งองค์ประกอบที่สำคัญซึ่งกล่าวถึงวิธีการต่อสู้ระหว่างตัวละครอวตารและศัตรูว่าเกิดขึ้นอย่างไรและดำเนินไปอย่างไร ในหัวข้อ 4.2.2 ได้กล่าวถึงการโจมตี 2 ประเภทในมุมมองของผู้เล่น ได้แก่ การโจมตีระยะใกล้ และการโจมตีระยะไกล การโจมตีระยะไกลมักจะอยู่ในรูปแบบของการปล่อยกระสุนออกไป ในขณะที่การโจมตีระยะใกล้มักจะเกิดขึ้นในรูปแบบของการเปลี่ยนภาพท่าทางของตัวละคร เช่น ศัตรู Specter Knight จากเกม Shovel Knight จะเปลี่ยนจากภาพอยู่นิ่งไปเป็นภาพท่าฟันเมื่อใช้เคียวโจมตี ทว่ากลไกเบื้องหลังของท่าโจมตีระยะใกล้สามารถใช้หลักการเดียวกับการโจมตีระยะไกลได้ โดยใช้การปล่อยกระสุนที่มองไม่เห็นออกไปเมื่อทำการโจมตีให้รับกับภาพท่าทางของศัตรูนั้น ผู้เล่นก็จะไม่รู้สึกว่าการโจมตีนั้นเป็นการโจมตีระยะไกล ด้วยแนวคิดของการใช้กระสุนสำหรับการโจมตีทุกประเภท กระสุนจะส่งผลในการต่อสู้เมื่อกระสุนนั้นเข้าถึงเป้าหมาย การตรวจสอบว่ากระสุนเข้าถึงเป้าหมายหรือไม่ จำเป็นต้องมีการตรวจสอบขนาดและตำแหน่งของกระสุนกับขนาดและตำแหน่งของเป้าหมาย ในที่นี้จึงต้องมีการนิยามสิ่งที่เป็นภาคีกระสุนซึ่งมีค่าคุณสมบัติตัวตรวจจับการชนเช่นเดียวกัน งานวิทยานิพนธ์นี้จะนิยามภาคีกระสุนเป็นภาคีแบบเดียวกับศัตรูในหัวข้อ 4.4.3

เมื่อกระสุนเข้าถึงเป้าหมายแล้วย่อมหมายถึงเกิดการโจมตีขึ้น ประเด็นสำคัญต่อมาจึงเกี่ยวข้องกับความเสียหายที่เกิดขึ้นจากการต่อสู้ ความเสียหายที่ได้กล่าวถึงไปแล้วในหัวข้อ 4.2.2 เป็นค่าความเสียหายที่เกิดขึ้นต่อพลังชีวิตของตัวละครอวตารหรือศัตรู ศัตรูจึงต้องมีค่าคุณสมบัติ “พลังชีวิต” (Life) นอกจากนี้ ในเกมที่วิเคราะห์ ศัตรูและกระสุนโดยส่วนใหญ่จะสามารถสร้างความเสียหายให้กับตัวละครอวตารด้วยการชนได้ ทำให้ทั้งสองจำเป็นต้องมีค่าคุณสมบัติ “พลังโจมตี” (Attack) ทว่าฝ่ายผู้เล่นเองก็สามารถโจมตีศัตรูด้วยกระสุน แต่ไม่ใช่ศัตรูทุกประเภทจะได้รับความเสียหายจากการโจมตีด้วยเช่นกัน ความสัมพันธ์ในส่วนของ “ใคร” สามารถโจมตีใส่ “ใคร” ได้ทำให้เกิดค่าคุณสมบัติสองค่าที่สำคัญได้แก่ “ฝ่ายโจมตี” (Attacker) และ “ฝ่ายตั้งรับ” (Defender) โดยเมื่อฝ่ายโจมตีชนกับฝ่ายตั้งรับ จะสร้างความเสียหายต่อพลังชีวิตของฝ่ายตั้งรับเท่ากับพลังโจมตีของตนเอง

การปะทะกันของฝ่ายโจมตีและฝ่ายตั้งรับอาจไม่ได้ทำให้เกิดความเสียหายเสมอไป เช่น ศัตรู Metall ใน Megaman หรือศัตรูที่ถือโล่ใน Shovel Knight ไม่รับความเสียหายจากการโจมตีของผู้เล่น ทำให้จำเป็นต้องมีค่าคุณสมบัติ “คงกระพัน” (Invulnerable) สำหรับศัตรูเพื่อระบุว่า ศัตรูที่มีคุณสมบัตินี้จะไม่รับความเสียหายถึงแม้ตนซึ่งเป็นฝ่ายตั้งรับจะถูกโจมตีด้วยฝ่ายโจมตี

การใช้งานคุณสมบัติฝ่ายโจมตีและฝ่ายตั้งรับยังไม่สามารถครอบคลุมในบางกรณี เนื่องจากศัตรูอาจจะโจมตีใส่ศัตรูอื่นได้หากศัตรูเป้าหมายดังกล่าวเป็นฝ่ายตั้งรับ ทั้งที่ควรจะโจมตีใส่ตัวละครอวตารซึ่งนับได้ว่าไม่ใช่พวกพ้องของตนเองดังที่ได้กล่าวถึงในหัวข้อ 4.2.3 ในขณะเดียวกันกระสุนของตัวละครอวตารก็อาจสร้างความเสียหายให้กับตนเองได้เนื่องจากกระสุนต้องมีค่าคุณสมบัติฝ่ายโจมตีเพื่อให้สร้างความเสียหายได้ ในที่นี้จึงต้องเพิ่มค่าคุณสมบัติ “กลุ่ม” (Group) เพื่อระบุว่าเป็นศัตรูหรือกระสุนหนึ่ง ๆ เป็นของฝ่ายใด กฎที่เกี่ยวข้องกับกลุ่มคือฝ่ายตั้งรับไม่รับการโจมตีจากฝ่ายโจมตีที่อยู่กลุ่มเดียวกัน

ค่าคุณสมบัติที่วิเคราะห์มาทั้งหมดข้างต้น จะแสดงสรุปอีกครั้งหนึ่งในหัวข้อ 4.4.1

4.3.2 การกระทำ (Action)

การกระทำของศัตรูมีหน้าที่เปลี่ยนแปลงค่าคุณสมบัติบางอย่างของตนเองอย่างรวมไปถึงเปลี่ยนแปลงสถานะของโลกของเกมหรือสิ่งอื่นที่อยู่ในโลก การกระทำของศัตรูที่เห็นจากมุมมองของผู้เล่นอาจส่งผลต่อค่าคุณสมบัติหรือสถานะหลายอย่างหรือได้รับผลจากค่าสถานะต่าง ๆ ในคราวเดียวกัน เพื่อให้การวิเคราะห์เป็นไปอย่าง

มีระบบ การกระทำของศัตรูที่ผู้วิเคราะห์มองเห็นในฐานะของผู้เล่นจะถูกแยกย่อยเพื่อค้นหาค่าคุณสมบัติหรือสถานะต่าง ๆ ที่การกระทำส่งผลกระทบ แล้วจดบันทึกไว้ในรูปของ 2-tuple ซึ่งประกอบด้วยการกระทำในมุมมองของผู้เล่นและรายการของค่าคุณสมบัติ/ค่าสถานะที่ได้รับผลจากการกระทำ ตารางที่ 1 แสดงตัวอย่าง 2-tuple ของการกระทำ

ตารางที่ 1 ตัวอย่าง 2-tuple ของการกระทำและผลกระทบ

การกระทำที่สังเกตได้	ค่าคุณสมบัติที่ได้รับ/ส่งผลกระทบ
เดินไปทางซ้าย (หรือขวา)	ค่าแกน x ของตัวตรวจจับการชน
กระโดด	ค่าแกน x และค่าแกน y ของตัวตรวจจับการชน, ค่าแรงดึงดูดของโลกของเกม
ยิงกระสุน	รายการวัตถุที่อยู่ในโลกของเกม

หากอธิบายกระทำของศัตรูด้วยค่าคุณสมบัติที่ได้รับผลกระทบ จะทำให้การกระทำอยู่ในระดับต่ำมากเทียบได้กับตัวดำเนินการในงานของ Zook [30] ส่งผลให้ปริภูมิค้นหามีขนาดใหญ่มาก ไม่ช่วยในการแก้ข้อจำกัดต่าง ๆ ที่พบในงานของ Zook ในที่นี้จึงมีความจำเป็นในการรวมผลกระทบต่อหลายคุณสมบัติเป็นหนึ่งการกระทำที่มีความหมายมากขึ้นโดยอิงจากเกมตัวอย่าง เช่น การกระทำกระโดดที่สังเกตได้ ไม่ควรแยกเป็นการเปลี่ยนแปลงของค่าแกน x และค่าแกน y ของตัวตรวจจับการชน แต่ควรคงไว้เป็นพฤติกรรมกระโดดซึ่งได้รับผลจากแรงดึงดูด

นอกจากนี้ การกระทำบางอย่างที่พบในเกมตัวอย่างจะมีความคล้ายคลึงกัน เช่น การเดินในทิศทางต่าง ๆ ด้วยความเร็วที่แตกต่างกัน หากนิยามการกระทำเหล่านี้แยกออกจากกันจะทำให้เกิดความซ้ำซ้อนกันได้ ตัวอย่างของความซ้ำซ้อนได้แก่ การเดินไปทางซ้ายและเดินไปทางขวา ซึ่งเป็นการเปลี่ยนแปลงค่าคุณสมบัติเดียวกัน แต่การกระทำหนึ่งลดตำแหน่งในแกน x ในขณะที่อีกการกระทำเพิ่มค่าดังกล่าว หรือแม้กระทั่งการเดินไปในทิศอื่นเองก็เป็นการเปลี่ยนแปลงที่คุณสมบัติตำแหน่งของตัวตรวจจับการชนเหมือนกัน งานวิทยานิพนธ์นี้จึงรวมเอาการกระทำที่มีการเปลี่ยนแปลงค่าคุณสมบัติเดียวกันและมีความหมายของการกระทำไปในทางเดียวกันไว้ในรูปของฟังก์ชั่นและพารามิเตอร์เพื่อลดความซ้ำซ้อน เช่น การเดินในทิศทางต่าง ๆ ตามตัวอย่างข้างต้น ถูกรวมเป็นฟังก์ชั่น RunStraight(Running direction, Running speed) ซึ่งสามารถรับทิศทางและความเร็วในการเดินได้ อนึ่ง รายการของฟังก์ชั่นการกระทำทั้งหมดได้แสดงไว้ในภาคผนวกส่วนรายการฟังก์ชั่นการกระทำ

4.3.3 การทดสอบเงื่อนไข (Condition testing) และการสอบถามข้อมูล (Information query)

การกระทำที่พบในเกมที่นำมาวิเคราะห์จำนวนหนึ่งไม่ได้เกิดขึ้นทันทีทันใด แต่เกิดขึ้นเมื่อสถานะของโลกของเกมหรือค่าคุณสมบัติบางอย่างของภาคีถูกต้องตามเงื่อนไขบางอย่าง เช่น ศัตรู Flyer ใน Metroid จะบินขึ้นในแนวดิ่งจนกระทั่งอยู่ในแนวระดับเดียวกับตัวละครอวตาร จึงเปลี่ยนทิศการเคลื่อนที่เป็นแนวราบแทน ตัวอย่างนี้แสดงให้เห็นถึงการทดสอบเงื่อนไขของคุณสมบัติตำแหน่งในแกน y ระหว่างศัตรูและตัวละครอวตาร ในบางครั้งเงื่อนไขสำหรับการกระทำอาจจะซับซ้อนกว่าการทดสอบระหว่างค่าคุณสมบัติเพียงค่าเดียว เช่น ศัตรู Bright man ใน Megaman 4 จะโจมตีด้วยท่าพิเศษเมื่อค่าพลังชีวิตของศัตรูมีค่าตรงตามที่กำหนดและตัวศัตรูยืนอยู่บนพื้น

ในที่นี้วิเคราะห์รูปแบบของเงื่อนไขที่พบและข้อมูลที่ใช้เป็นเงื่อนไขในเกมที่กำหนด พบว่าโดยส่วนใหญ่ลักษณะของเงื่อนไขจะเป็นดังต่อไปนี้

- การเปรียบเทียบระหว่างข้อมูล 2 ข้อมูล ซึ่งเป็นข้อมูลเดียวกันที่ได้จากภาคีที่แตกต่างกัน
- การเปรียบเทียบระหว่างข้อมูลและค่าคงที่
- การตรวจสอบว่าภาคีที่กำหนดอยู่ในสถานะที่สนใจหรือไม่
- อื่น ๆ ที่ไม่เข้าพวก เช่น การตรวจสอบว่าผู้เล่นกดปุ่มที่กำหนดอยู่หรือไม่ เป็นต้น

จากรูปแบบของเงื่อนไข งานวิทยานิพนธ์นี้จึงกำหนดให้มีวิธีการสำหรับการสร้างเงื่อนไขเพื่อทดสอบและกำหนดพฤติกรรมตามแต่เงื่อนไขให้กับศัตรูโดยใช้ตรรกะแบบถ้า-เงื่อนไข-แล้ว ซึ่งเทียบได้กับโครงสร้างควบคุม If-Then ในภาษาโปรแกรมมิ่งทั่วไป ด้วยลักษณะของตรรกะดังกล่าว เงื่อนไขในที่นี้จึงต้องเป็นนิพจน์แบบบูล (Boolean expression) ส่วนเงื่อนไขสามารถสร้างขึ้นจากการนำข้อมูลมาเชื่อมต่อกันด้วยตัวดำเนินการต่าง ๆ เมื่ออ้างอิงรูปแบบของเงื่อนไขข้างต้น แสดงให้เห็นความจำเป็นของตัวดำเนินการเปรียบเทียบ (Comparator) ในกรณีที่เงื่อนไขมีความซับซ้อนขึ้น ต้องมีการเชื่อมระหว่างเงื่อนไขย่อย ๆ หลายเงื่อนไข ในที่นี้จึงต้องมีตัวดำเนินการเชิงตรรกะ (Logical operator)

สิ่งสำคัญในการสร้างเงื่อนไขคือข้อมูล ข้อมูลในที่นี้อาจจะเป็นค่าคุณสมบัติของภาคีหรือสถานะของโลกของเกมก็ได้ รวมถึงข้อมูลพิเศษ เช่น การกดปุ่มของผู้เล่น เป็นต้น การเก็บข้อมูลของข้อมูลที่ถูกใช้เป็นเงื่อนไขกระทำในลักษณะเดียวกับการกระทำ นั่นคือ จดบันทึกลักษณะของข้อมูลที่ถูกใช้งานแล้วรวมข้อมูลที่มีลักษณะคล้ายคลึงกันให้อยู่ในรูปของฟังก์ชั่น เช่น ข้อมูลสถานะกำแพงอยู่ด้านหน้าหรือไม่และภาคีอยู่บนพื้นหรือไม่ จะถูกรวมเป็นฟังก์ชั่นที่มีพารามิเตอร์ SurfaceInDirection(Direction) ซึ่งคืนค่าตรรกะที่บอกว่ามีโครงสร้างพื้นที่ที่ผ่านไม่ได้อยู่ในทิศที่กำหนดหรือไม่ ฟังก์ชั่นข้อมูลเหล่านี้จะแตกต่างจากฟังก์ชั่นการกระทำที่ความสามารถในการให้ข้อมูล การสอบถามข้อมูลเพื่อนำไปใช้งานในเงื่อนไขหรือใช้งานเป็นพารามิเตอร์ของฟังก์ชั่นอื่น ๆ จึงได้จากการเรียกใช้งานฟังก์ชั่นข้อมูลดังกล่าวนี้ รายการของฟังก์ชั่นข้อมูลทั้งหมดที่พบได้แสดงไว้ในภาคผนวก

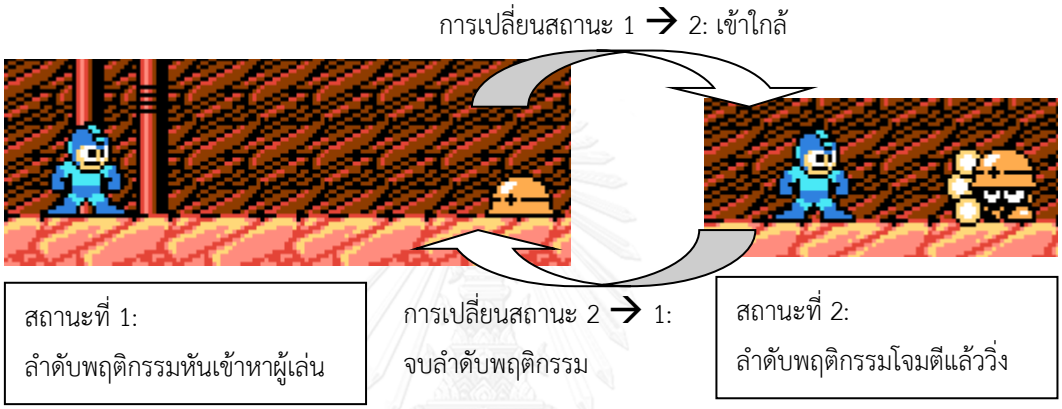
4.3.4 ลำดับพฤติกรรม (Behavior sequence)

ในเกมที่นำมาวิเคราะห์รวมถึงเกมแอ็คชั่นโดยทั่วไป ศัตรูไม่ได้มีการกระทำเพียงแบบเดียว ผู้เล่นสามารถรับรู้ได้จากความเปลี่ยนแปลงของการกระทำที่เปลี่ยนไปตามเวลา มีการเปลี่ยนแปลงจากการกระทำหนึ่งไปยังอีกแบบหนึ่ง เช่น ศัตรู Pakatto24 ซึ่งเป็นศัตรูแบบป้อมปืนใน Megaman 4 จะมีการกระทำเป็นลำดับต่อเนื่อง คือ เปิดเกราะ, ยิง, หยุดรอ, ซ่อนกลับเข้าเกราะ, หยุดรอ อีกทั้งในศัตรูบางตัวอาจจะมีเงื่อนไขของการกระทำเพื่อเลือกรูปแบบการกระทำได้ด้วย นั่นแสดงให้เห็นว่าพฤติกรรมของศัตรูมีลักษณะเป็นลำดับของการกระทำรวมถึงอาจมีการทดสอบเงื่อนไขและฟังก์ชั่นสอบถามข้อมูลเพื่อสร้างทางเลือกของการกระทำที่เป็นไปได้ ในที่นี้เรียกลำดับดังกล่าวว่าลำดับพฤติกรรม

4.3.5 สถานะของพฤติกรรม (Behavior state) และการเปลี่ยนสถานะ (State transition)

ศัตรูส่วนหนึ่งในเกมที่วิเคราะห์มีลำดับพฤติกรรมหลายแบบขึ้นอยู่กับเงื่อนไข เช่น ศัตรู Metall ใน Megaman 4 จะซ่อนอยู่ในหมวกเหล็กและคอยหันเข้าหาตัวละครอวตาร จนกระทั่งตัวละครอวตารเข้าใกล้ศัตรู

ระดับหนึ่งจึงออกมาจากหมวกเหล็ก, ยิงกระสุนโจมตี, วิ่งเข้าใส่ แล้วจึงกลับไปซ่อนในหมวกเหล็กเพื่อรอตัวละครอวตารกลับเข้ามาในระยะอีกครั้ง ในตัวอย่างนี้ แสดงให้เห็นถึงลำดับพฤติกรรม 2 ลำดับ ได้แก่ ลำดับพฤติกรรมที่ประกอบด้วยการกระทำหันหน้าเข้าหาผู้เล่นเพียงอย่างเดียว และลำดับพฤติกรรมที่ประกอบด้วยการโจมตีและการเคลื่อนที่ โดยลำดับพฤติกรรมแรกมีระยะห่างจากผู้เล่นเป็นเงื่อนไขในการเปลี่ยนไปใช้งานลำดับพฤติกรรมที่สอง ในขณะที่ลำดับพฤติกรรมที่สองมีการจบลำดับเป็นเงื่อนไขในการเปลี่ยนกลับไปยังลำดับพฤติกรรมแรก การเปลี่ยนแปลงเช่นนี้มีความคล้ายคลึงกับหลักการของเครื่องสถานะจำกัด โดยมองว่าแต่ละลำดับพฤติกรรมของศัตรูเป็นสถานะหนึ่ง และเงื่อนไขการเปลี่ยนแปลงลำดับพฤติกรรมเป็นเงื่อนไขการเปลี่ยนสถานะ รูปที่ 7 แสดงตัวอย่างการเปลี่ยนลำดับพฤติกรรมด้วยหลักการของเครื่องสถานะจำกัด



รูปที่ 7 การเปลี่ยนลำดับพฤติกรรมแสดงด้วยหลักการของเครื่องสถานะจำกัด

จากความคล้ายคลึงของหลักการข้างต้น สถานะและการเปลี่ยนสถานะจึงเป็นอีกองค์ประกอบหนึ่งของศัตรูที่ใช้ระบุความเปลี่ยนแปลงของลำดับพฤติกรรมในกรณีที่ศัตรูมีลำดับพฤติกรรมหลายลำดับ โดยศัตรูจะต้องมีการระบุตัวเลขสถานะให้กับทุกลำดับพฤติกรรม และสำหรับทุกลำดับพฤติกรรมจะต้องมีเซ็ตอธิบายการเปลี่ยนสถานะซึ่งเป็น เซ็ตของ 2-tuple $\langle condition, destination \rangle$ เมื่อ $condition$ คือเงื่อนไขการเปลี่ยนสถานะ และ $destination$ คือตัวเลขสถานะปลายทาง

4.3.6 ลำดับพฤติกรรมคู่ขนาน (Parallel sequence)

นอกจากศัตรูจะมีได้หลายลำดับพฤติกรรมแล้ว ในบางครั้งศัตรูอาจกระทำหลายลำดับพฤติกรรมในเวลาเดียวกันได้ เช่น ศัตรู Sniper ใน Super C ซึ่งสามารถยิงปืนโจมตีผู้เล่นทุกช่วงเวลาที่กำหนด ลำดับพฤติกรรมในที่นี้คือ ยิง, หยุดรอ แต่ในขณะเดียวกัน ศัตรูตัวดังกล่าวนี้ยังหันปืนเข้าหาผู้เล่นอีกด้วย แสดงให้เห็นถึงลำดับพฤติกรรมหันเข้าหาผู้เล่นที่กระทำไปพร้อมกับลำดับพฤติกรรมยิงแล้วหยุดรอ นั่นหมายถึงในหนึ่งสถานะจะไม่ได้มีเพียงลำดับพฤติกรรมเดียวที่กระทำอยู่ แต่อาจมีหลายลำดับพฤติกรรมได้ ด้วยแนวคิดดังกล่าวนี้ จึงต้องมีการปรับปรุงองค์ประกอบสถานะของศัตรู โดยให้แต่ละสถานะประกอบด้วยตัวเลขสถานะและเซ็ตของลำดับพฤติกรรม

4.4 แบบจำลองภาคี

หัวข้อนี้จะแสดงรายละเอียดของแบบจำลองศัตรูซึ่งสร้างขึ้นจากองค์ประกอบของศัตรูที่อธิบายไว้ในหัวข้อก่อนหน้า แบบจำลองดังกล่าวนี้จะถูกปรับปรุงเพื่อลดความซ้ำซ้อนลงรวมถึงเพื่อให้ครอบคลุมกระสุน แบบจำลองที่ผ่านการปรับปรุงแล้วจะถูกเรียกว่าแบบจำลองภาคี ซึ่งจะถูกนำเสนอในรูปของภาษาอธิบายภาคีที่ตัวแปลภาษาสามารถทำความเข้าใจได้

4.4.1 แบบจำลองศัตรู

ศัตรูประกอบด้วยองค์ประกอบต่าง ๆ ดังที่ได้กล่าวถึงในหัวข้อ 4.3 แบบจำลองศัตรูจะถูกสร้างขึ้นด้วยองค์ประกอบเหล่านี้ ดังนั้นแบบจำลองศัตรูจึงประกอบด้วย

ค่าคุณสมบัติ

โดยบางคุณสมบัตินั้นเป็นคุณสมบัติที่ศัตรูจะต้องมีโดยค่าจะแตกต่างกันไปในศัตรูแต่ละตัว ค่าคุณสมบัติเหล่านี้จะขาดไปไม่ได้ มิเช่นนั้นศัตรูจะไม่สามารถอยู่ในโลกของเกมได้ เช่น ค่าคุณสมบัติตัวตรวจจับการชนที่ต้องมีเพื่อระบุตำแหน่งในโลกของเกม เป็นต้น สำหรับบางคุณสมบัติจะเป็นคุณสมบัติทางเลือก อาจไม่มีก็ได้ แต่ก็อาจจำกัดความสามารถบางอย่างของศัตรูไป เช่น ศัตรูที่ไม่มีค่าคุณสมบัติฝ่ายโจมตีก็จะไม่สามารถโจมตีได้ เป็นต้น นอกเหนือจากค่าคุณสมบัติที่ผ่านการวิเคราะห์แล้ว เนื่องจากการเคลื่อนที่ของศัตรูในเกมที่วิเคราะห์มีลักษณะเป็นแบบไปข้างหน้า เพื่อเป็นการลดทอนจำนวนพารามิเตอร์ของฟังก์ชั่นการกระทำบางส่วน ในที่นี้จึงเพิ่มค่าคุณสมบัติ “ทิศทาง” (Direction) ให้กับแบบจำลองด้วย รายการค่าคุณสมบัติทั้งหมดของศัตรูจึงเป็นไปดังตารางที่ 2

ตารางที่ 2 ค่าคุณสมบัติสำหรับแบบจำลองศัตรู

ค่าคุณสมบัติจำเป็น	ค่าคุณสมบัติทางเลือก
ตัวตรวจจับการชน ซึ่งประกอบด้วยขนาดและ) (ตำแหน่ง, ผลของแรงดึงดูด, พลังชีวิต, พลังโจมตี, กลุ่ม, ทิศทาง	ผ่านได้, คงกระพัน, ฝ่ายโจมตี, ฝ่ายตั้งรับ

เซ็ตของสถานะ

โดยแต่ละสถานะจะประกอบด้วยข้อมูล 3 ชนิด ได้แก่

- หมายเลขสถานะ ซึ่งจะต้องไม่ซ้ำกันสำหรับทุกสถานะของศัตรูนั้น ๆ โดยศัตรูจะต้องมีสถานะเริ่มต้นเสมอ ในที่นี้ให้สถานะหมายเลข 0 เป็นสถานะเริ่มต้นของศัตรู
- เซ็ตของลำดับพฤติกรรมที่แสดงถึงลำดับพฤติกรรมทั้งหมดที่กระทำคู่ขนานกันในสถานะหนึ่ง
- เซ็ตของการเปลี่ยนสถานะซึ่งเป็นเซ็ตของ 2-tuple ที่ประกอบด้วยเงื่อนไขและสถานะปลายทาง

ลำดับพฤติกรรม

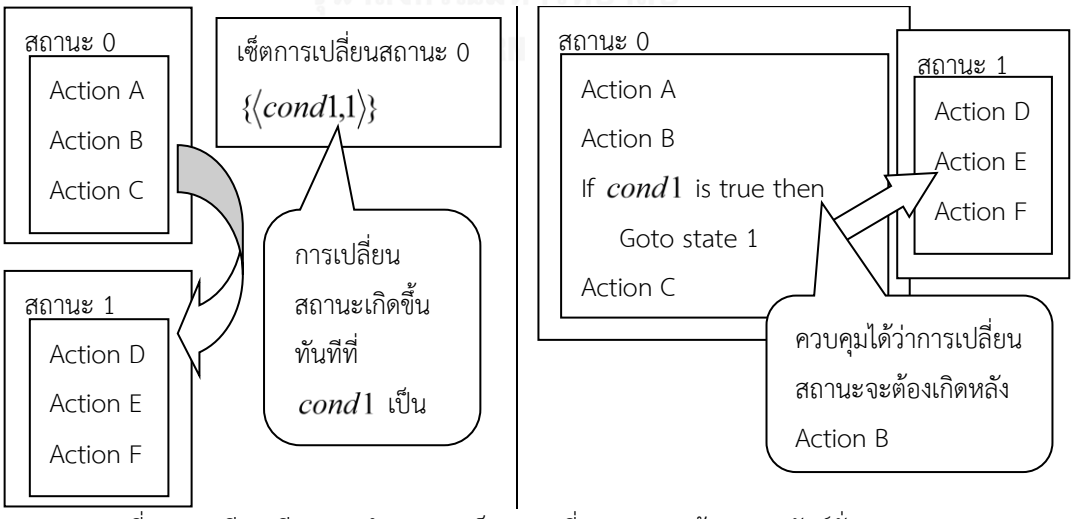
เป็นส่วนหนึ่งของสถานะ ประกอบด้วยฟังก์ชั่นการกระทำ ฟังก์ชั่นข้อมูล และโครงสร้างทดสอบเงื่อนไข

- โครงสร้างทดสอบเงื่อนไข เป็นข้อมูลในรูปแบบถ้า-เงื่อนไข-แล้ว แต่เพื่อสร้างทางเลือกเพิ่มเติมในกรณีที่เงื่อนไขไม่เป็นจริง จึงต้องปรับเปลี่ยนโครงสร้างเป็น ถ้า-เงื่อนไข-แล้ว-ถ้าไม่ใช่-แล้ว ซึ่งเทียบได้กับโครงสร้างแบบ If-Then-Else ในภาษาโปรแกรมมิ่ง
- เงื่อนไข เป็นนิพจน์แบบบูลซึ่งประกอบขึ้นจากนิพจน์แบบบูล ตัวดำเนินการเปรียบเทียบ และตัวดำเนินการตรรกะ

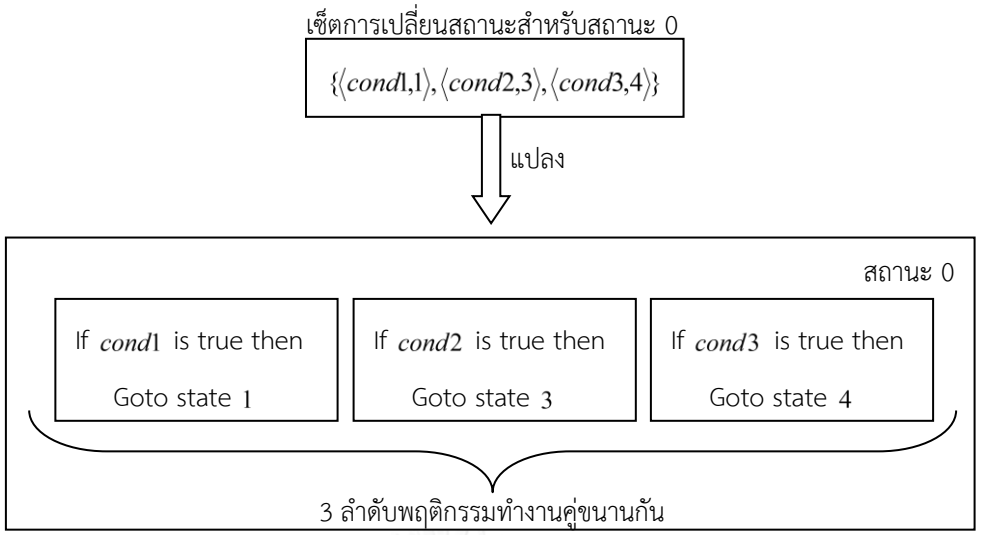
4.4.2 การละเซ็ตการเปลี่ยนสถานะ

เซ็ตของการเปลี่ยนสถานะที่กล่าวถึงในหัวข้อ 4.3.5 สามารถละได้โดยเปลี่ยนไปใช้ฟังก์ชั่นการกระทำพิเศษ Goto(Target state) เพื่อลดความซับซ้อนของแบบจำลองและทำให้ควบคุมการเปลี่ยนสถานะได้ดียิ่งขึ้น

ฟังก์ชั่น Goto เป็นฟังก์ชั่นที่สั่งให้ศัตรูเปลี่ยนสถานะไปเป็นสถานะที่กำหนดและเริ่มกระทำลำดับพฤติกรรมทั้งหมดในสถานะปลายทาง สาเหตุที่สามารถใช้ฟังก์ชั่น Goto แทนเซ็ตของการเปลี่ยนสถานะได้เนื่องมาจากวิธีการทำงานของเซ็ตของการเปลี่ยนสถานะ ซึ่งจะถูกตรวจสอบตลอดเวลาไปพร้อมกับการทำลำดับพฤติกรรมของสถานะปัจจุบัน เมื่อพบ 2-tuple ที่เงื่อนไขเป็นจริงจึงทำการเปลี่ยนไปยังสถานะปลายทางที่กำหนดใน 2-tuple นั้น ลักษณะการทำงานดังกล่าวสามารถเปลี่ยนรูปแบบไปเป็นลำดับพฤติกรรมคู่ขนานอีกหนึ่งลำดับที่ประกอบด้วยเงื่อนไขและฟังก์ชั่นการกระทำ Goto ที่มีพารามิเตอร์เป็นสถานะปลายทาง สำหรับกรณีเงื่อนไขเป็นจริง นอกจากนี้การใช้งานในรูปฟังก์ชั่นการกระทำ Goto จะทำให้ควบคุมการเปลี่ยนสถานะได้ดียิ่งขึ้น โดยสามารถแทรกฟังก์ชั่นดังกล่าวระหว่างการกระทำอื่น ๆ ทำให้สามารถยืนยันได้ว่าสถานะจะถูกเปลี่ยนโดยไม่ทำการกระทำถัดไปได้ ซึ่งในกรณีที่ใช้งานเซ็ตของการเปลี่ยนสถานะจะไม่สามารถควบคุมพฤติกรรมเช่นนี้ได้ รูปที่ 8 แสดงความแตกต่างของการเปลี่ยนสถานะระหว่างการใช้เซ็ตของการเปลี่ยนสถานะและการใช้ฟังก์ชั่น Goto และรูปที่ 9 แสดงการละเซ็ตการเปลี่ยนสถานะและไปใช้งานฟังก์ชั่น Goto แทน



รูปที่ 8 รูปเปรียบเทียบการทำงานของเซ็ตการเปลี่ยนสถานะ (ซ้าย) และฟังก์ชั่น Goto (ขวา)



รูปที่ 9 ตัวอย่างการแปลงเซ็ตการเปลี่ยนสถานะ

4.4.3 การรวมกระสุนเข้ากับแบบจำลองศัตรู

การรวมกระสุนเข้ากับแบบจำลองศัตรูจะทำได้ก็ต่อเมื่อสามารถแสดงให้เห็นว่าสามารถอธิบายกระสุนได้ครบถ้วนด้วยแบบจำลองศัตรู กระสุนอยู่ในโลกของเกมและสามารถชนกับภาคีอื่น ๆ เพื่อสร้างความเสียหายได้ จึงต้องมีตัวตรวจจับการชนซึ่งได้กล่าวถึงแล้วในหัวข้อ 4.3.1 และเมื่อกล่าวถึงความเสียหาย ย่อมเกี่ยวข้องกับกลไกการต่อสู้ซึ่งมีค่าคุณสมบัติที่สำคัญได้แก่ ฝ่ายโจมตี, พลังโจมตี และกลุ่ม นอกจากนี้ กระสุนบางชนิดของศัตรูจะสามารถทำลายได้โดยใช้การโจมตีของตัวละครอวตาร แสดงให้เห็นว่า กระสุนสามารถมีค่าคุณสมบัติฝ่ายป้องกันและพลังชีวิตได้ด้วย

อีกกลไกหนึ่งที่น่าสนใจสำหรับกระสุนคือการสลายตัว กระสุนในเกมที่วิเคราะห์บางชนิดจะสลายตัวเมื่อกระทบถูกเป้าหมายแล้ว ในขณะที่บางชนิดจะไม่สลายตัวไปหรือสลายตัวไปหลังจากกระทบเป้าหมายหลายครั้ง ในที่นี้จึงนำค่าคุณสมบัติพลังชีวิตและคงกระพันมาใช้จัดการกับกลไกเช่นนี้ โดยกำหนดว่า หากกระสุนกระทบกับเป้าหมายแล้ว จะเสียพลังชีวิตลง 1 หน่วยในกรณีที่ไม่มีค่าคุณสมบัติคงกระพัน และเมื่อพลังชีวิตหมดกระสุนจะหายไป เนื่องจากกลไกการสลายตัวของกระสุนซึ่งกล่าวถึงการสลายตัวหลังทำการโจมตีเป็นจำนวนครั้งที่กำหนดเป็นกลไกพิเศษที่ไม่พบในศัตรู จึงต้องเพิ่มค่าคุณสมบัติพิเศษ “ภาคีกระสุน” (Projectile) ให้กับแบบจำลองศัตรูหากต้องการรวมภาคีกระสุนเข้ากับแบบจำลอง และให้กลไกการสลายตัวเกิดขึ้นกับภาคีที่มีค่าคุณสมบัติกระสุนเท่านั้น

สำหรับค่าคุณสมบัติอื่น ๆ ได้แก่ ผลแรงดึงดูด, ทิศทาง และผ่านได้ กระสุนจำเป็นต้องมีค่าคุณสมบัติผลแรงดึงดูดเนื่องจากกระสุนที่พบในเกมที่วิเคราะห์มีทั้งกระสุนที่ลอยได้และกระสุนที่ตกลงตามแรงโน้มถ่วง สำหรับค่าคุณสมบัติผ่านได้ เนื่องจากกระสุนส่วนใหญ่สามารถทะลุผ่านโครงสร้างพื้นที่ที่ผ่านไม่ได้ ยกเว้นแต่กระสุนในเกม Super C ที่กระสุนทุกชนิดไม่สามารถทะลุผ่านโครงสร้างพื้นที่ที่ผ่านไม่ได้เช่นเดียวกับศัตรู ซึ่งน่าจะเป็นความตั้งใจของการออกแบบเกมมากกว่าการทำให้เกิดความแตกต่างของพฤติกรรม ในที่นี้จึงถือว่ากระสุนมีคุณสมบัติผ่านได้อยู่เสมอ ส่วนค่าคุณสมบัติทิศทางนั้นเป็นค่าคุณสมบัติที่เกี่ยวข้องกับการลดทอนความซับซ้อนของพฤติกรรม หากพฤติกรรมของกระสุนอธิบายได้ด้วยแบบจำลองศัตรู กระสุนก็ต้องมีค่าคุณสมบัติทิศทางเช่นกัน ซึ่งกระสุนส่วนใหญ่มี

พฤติกรรมที่เรียบง่าย ไม่ได้มีลำดับพฤติกรรมที่ซับซ้อนเช่นเดียวกับศัตรู ส่งผลให้แบบจำลองศัตรูที่มีความซับซ้อนมากกว่าสามารถครอบคลุมองค์ประกอบทางด้านพฤติกรรมของกระสุนได้

ด้วยเหตุผลทั้งหมดที่กล่าวมา แสดงให้เห็นว่ากระสุนเองก็มีองค์ประกอบเหมือนกับศัตรู รวมถึงมีการตอบสนองกับองค์ประกอบอื่น ๆ ในลักษณะเดียวกับศัตรู งานวิทยานิพนธ์นี้จึงรวมกระสุนซึ่งนับได้ว่าเป็นภาคีอีกประเภทหนึ่งเข้ากับแบบจำลองของศัตรู และเรียกแบบจำลองซึ่งมีค่าคุณสมบัติภาคีกระสุนเพิ่มเติมขึ้นมาว่า “แบบจำลองภาคี”

4.4.4 ภาษาอธิบายภาคี

เพื่อให้แบบจำลองภาคีมีรูปแบบที่ชัดเจน มนุษย์สามารถเข้าใจได้ง่าย และสามารถแปลด้วยตัวแปลภาษาได้ งานวิทยานิพนธ์นี้จึงทำให้แบบจำลองภาคีอยู่ในรูปของภาษาอธิบายภาคี

ภาษาอธิบายภาคีเป็นภาษาที่มีชนิดข้อมูล โครงสร้างเงื่อนไข พารามิเตอร์ทุกตัวของฟังก์ชั่นการกระทำและฟังก์ชั่นข้อมูล รวมถึงข้อมูลที่ได้จากฟังก์ชั่นข้อมูลจะถูกกำหนดชนิดของข้อมูลให้ การนำข้อมูลไปใช้งานภายในภาษาอธิบายภาคีจะต้องตรงชนิดกับชนิดข้อมูลที่ฟังก์ชั่นหรือโครงสร้างเงื่อนไขต้องการ

โครงสร้างของภาษาอธิบายภาคีจะเป็นรูปแบบบล็อก กล่าวคือ องค์ประกอบต่าง ๆ ของภาษาจะถูกห่อหุ้มด้วยสัญลักษณ์บล็อก ซึ่งในที่นี้ใช้เครื่องหมายวงเล็บปีกกาเปิด “{“ สำหรับจุดเริ่มต้นบล็อก และวงเล็บปีกกาปิด “}” สำหรับจุดสิ้นสุดบล็อก สิ่งที่อยู่ระหว่างบล็อกใด ๆ เรียกว่า เนื้อหาของบล็อก (Block content) ภาพรวมของภาษาอธิบายภาคีแสดงได้ด้วยไวยากรณ์ดังรูปที่ 10

```
Agent → IDENTIFIER { Initializer StateGroup Destructor }
Initializer → .init { Sequence } | ^
Destructor → .des { Sequence } | ^
StateGroup → StateGroup StateGroup | State
State → IDENTIFIER { Parallel }
Parallel → Parallel Parallel | SequenceBlock | ^
SequenceBlock → IDENTIFIER { Sequence }
Sequence → Sequence Sequence | CondStruct | LoopStruct | ACTION ;
CondStruct → IfBlock | IfBlock ElseBlock
IfBlock → if( BOOLEANEXP ) { Sequence }
ElseBlock → else IfBlock | else { Sequence }
LoopStruct → loop( INTEXP ) { Sequence }
```

รูปที่ 10 ไวยากรณ์ของภาษาอธิบายภาคี

รายละเอียดของภาคีทั้งหมดจะถูกห่อหุ้มอยู่ในบล็อกที่เรียกว่า “บล็อกภาคี” (Agent block) ซึ่งประกอบด้วยตัวระบุ (Identifier) ตามสัญลักษณ์บล็อก ภายในบล็อกนี้จะประกอบด้วย “บล็อกเริ่มต้น” (Initializer block), “บล็อกทำลาย” (Destructor block) และ “กลุ่มบล็อกสถานะ” (State group)

บล็อกเริ่มต้นเป็นบล็อกที่มีตัวระบุบังคับชื่อ “.init” บล็อกภาคีจะมีบล็อกเริ่มต้นหรือไม่ก็ได้ เนื้อหาของบล็อกเป็นลำดับพฤติกรรมที่ภาคีจะกระทำเมื่อภาคีเข้าสู่โลกของเกม โดยปกติลำดับพฤติกรรมภายในบล็อกนี้จะเป็นการกำหนดค่าคุณสมบัติต่าง ๆ ให้กับภาคี

บล็อกทำลายเป็นบล็อกที่มีตัวระบุบังคับชื่อ “.des” บล็อกภาคีจะมีบล็อกทำลายหรือไม่ก็ได้ เนื้อหาของบล็อกเป็นลำดับพฤติกรรมที่ภาคีจะกระทำเมื่อถูกนำออกจากโลกของเกมไปแล้ว โดยปกติจะเป็นการกระทำที่ไม่เปลี่ยนแปลงค่าคุณสมบัติของตัวเอง แต่เปลี่ยนแปลงค่าคุณสมบัติของภาคีอื่น ๆ หรือสถานะของโลกของเกม

กลุ่มบล็อกสถานะประกอบด้วย “บล็อกสถานะ” (State block) จำนวนอย่างน้อย 1 บล็อก กลุ่มบล็อกสถานะใช้อธิบายพฤติกรรมหลักของภาคี หลังจากที่ภาคีกระทำลำดับพฤติกรรมในบล็อกเริ่มต้นเสร็จสิ้น ภาคีจะเข้าสู่สถานะแรกสุดที่ปรากฏในกลุ่มสถานะ และกระทำตามพฤติกรรมที่ระบุไว้ในบล็อกสถานะนั้น โดยสถานะแรกสุดนี้จะเรียกว่า สถานะเริ่มต้น

บล็อกสถานะเป็นบล็อกที่มีตัวระบุชื่อใด ๆ ที่ไม่ซ้ำกับบล็อกสถานะอื่น ๆ ภายในบล็อกภาคีเดียวกัน เนื้อหาของบล็อกสถานะประกอบด้วย “บล็อกลำดับ” (Sequence block) จำนวนกี่บล็อกก็ได้ แต่ละบล็อกลำดับเป็นตัวแทนของ 1 ลำดับพฤติกรรม ทุกบล็อกลำดับภายในบล็อกสถานะจะอธิบายลำดับพฤติกรรมที่คู่ขนานกัน บล็อกลำดับภายในบล็อกสถานะเดียวกันจะต้องมีตัวระบุชื่อไม่ซ้ำกัน โดยเนื้อหาของแต่ละบล็อกลำดับคือลำดับพฤติกรรม

ลำดับพฤติกรรมในภาษาอธิบายภาคีนี้จะประกอบขึ้นจากการนำเอาฟังก์ชั่นการกระทำ (แสดงด้วย ACTION ในรูปที่ *10*) โครงสร้างเงื่อนไข (แสดงด้วย CondStruct ใน รูปที่ *10*) และโครงสร้างวนซ้ำ (แสดงด้วย LoopStruct ในรูปที่ *10*) มาเรียงต่อกัน โดยโครงสร้างวนซ้ำเป็นโครงสร้างเฉพาะที่เพิ่มขึ้นมานอกเหนือจากแบบจำลองภาคีเพื่อให้สามารถสื่อถึงการออกแบบพฤติกรรมแบบวนซ้ำและลดจำนวนฟังก์ชั่นการกระทำที่ต้องเขียนลง แต่ละพารามิเตอร์ของฟังก์ชั่นการกระทำและนิพจน์ที่โครงสร้างจำเป็นต้องใช้สามารถระบุได้ด้วยข้อมูลที่ได้จากฟังก์ชั่นข้อมูลหรือสัญพจน์แทนข้อมูล

ค่าคุณสมบัติจะไม่มีการระบุไว้ในภาษาอธิบายภาคี แต่เกมที่จะนำภาษาอธิบายภาคีไปใช้จะต้องมีตัวแปลภาษาที่กำหนดค่าคุณสมบัติเริ่มต้นให้กับภาคีได้ สำหรับการกำหนดค่าคุณสมบัติเริ่มต้นเฉพาะสำหรับภาคีจะกระทำในบล็อกเริ่มต้นดังที่กล่าวไว้ โดยใช้ฟังก์ชั่นการกระทำ Set

สำหรับรายละเอียดปลีกย่อยอื่น ๆ ของภาษาอธิบายภาคีเป็นดังต่อไปนี้

สัญลักษณ์สิ้นสุด (Terminal symbol) ที่ปรากฏในไวยากรณ์

- IDENTIFIER เป็นตัวระบุ ใช้เป็นชื่อของบล็อกต่าง ๆ ซึ่งมีกฎการเลือกชื่อดังที่ได้กล่าวไว้แล้ว โดยการเขียนตัวระบุจะต้องขึ้นต้นด้วยเครื่องหมายมหัพภาค “.” ตามด้วยพยัญชนะภาษาอังกฤษ ตัวเลข หรือเส้นใต้อักขระ “_”
- ACTION เป็นฟังก์ชั่นการกระทำที่ไม่มีการคืนค่าซึ่งได้จากองค์ประกอบการกระทำของศัตรู พารามิเตอร์ของฟังก์ชั่นที่ได้จากการวิเคราะห์ในหัวข้อ 4.3.2 จะถูกกำหนดชนิดข้อมูลให้และใช้เป็นฟังก์ชั่นการกระทำ ค่าพารามิเตอร์ของฟังก์ชั่นอาจเป็นสัญพจน์หรือนิพจน์ที่มีชนิดข้อมูลตรงกันหรือฟังก์ชั่นข้อมูลที่คืนค่าเป็น

ชนิดข้อมูลที่กำหนด โดยฟังก์ชั่นข้อมูลในที่นี้ได้จากการวิเคราะห์ในหัวข้อ 4.3.3 โดยมีการกำหนดชนิดข้อมูลให้กับพารามิเตอร์และค่าที่คืนกลับของฟังก์ชั่นดังกล่าว

- BOOLEANEXP เป็นนิพจน์แบบบูลซึ่งได้จากการนำนิพจน์แบบบูลหรือสัญพจน์แบบบูลมาเชื่อมกันด้วยตัวดำเนินการตรรกะ หรือได้จากการเปรียบเทียบระหว่างข้อมูลชนิดต่าง ๆ ด้วยตัวเปรียบเทียบ หรือได้จากการเรียกใช้ฟังก์ชั่นข้อมูลที่คืนค่าเป็นค่าแบบบูล
- INTEXP เป็นนิพจน์จำนวนนับซึ่งได้จากการเขียนสัญพจน์ตัวเลข ใช้งานตัวดำเนินการเลขคณิต หรือใช้งานฟังก์ชั่นข้อมูลที่คืนค่าเป็นตัวเลข ซึ่งข้อมูลจะถูกแปรให้เป็นจำนวนนับก่อนใช้งาน

ชนิดข้อมูล

ตัวแปลภาษาจะใช้ชนิดข้อมูลในการตรวจสอบความเข้ากันระหว่างฟังก์ชั่นและอากิวเมนท์ที่ส่งเข้ามา โดยชนิดข้อมูลบางชนิดจะมีสัญพจน์ของตนเองดังตัวอย่างในตารางที่ 3 ในขณะที่ชนิดข้อมูลอีกหลายชนิดจะไม่มีสัญพจน์แต่ได้จากการเรียกใช้ฟังก์ชั่นที่คืนค่าที่มีชนิดข้อมูลดังกล่าว เช่น ชนิดข้อมูลตัวอ้างอิงภาคีไม่สามารถเขียนด้วยสัญพจน์ได้ แต่สามารถใช้งานได้จากฟังก์ชั่น เช่น DynamicFilter ซึ่งเป็นฟังก์ชั่นที่คืนค่าเป็นชนิดข้อมูลตัวอ้างอิงภาคี เป็นต้น

ตารางที่ 3 ตัวอย่างสัญพจน์

ชนิดข้อมูล	สัญพจน์	ความหมาย
Identifier	.BlockName	ตัวระบุบล็อกชื่อ BlockName
Boolean	true, false	ค่าตรรกะจริง, เท็จ ตามลำดับ
Decimal	1245, 13.02	ตัวเลขจำนวนจริง
Direction	“North”	ทิศทางที่ชี้ไปทิศเหนือ หรือเวกเตอร์ 2 มิติ (0,1)
	“230”	ทิศทางที่ได้จากการหมุนเวกเตอร์ 2 มิติ (1,0) 230 องศารอบจุด (0,0) ทวนเข็มนาฬิกา
Position	“c(120,150)”	ตำแหน่งในระบบพิกัดคาร์ทีเซียนที่มีตำแหน่ง x = 120 และ y = 150
	“p(50,180)”	ตำแหน่งในระบบพิกัดเชิงขั้วที่มีรัศมีเป็น 50 พิกเซล และมุม 180 องศา
Collider	“100,130”	ตัวตรวจจับการชนขนาดกว้าง 100 พิกเซล สูง 130 พิกเซล

ตัวเปรียบเทียบ

ตัวเปรียบเทียบใช้สำหรับสร้างนิพจน์แบบบูลซึ่งเป็นข้อมูลที่มีชนิดเป็น Boolean จากข้อมูล 2 ตัวโดยใช้การเปรียบเทียบ สามารถใช้งานได้โดยการเขียนตัวเปรียบเทียบไว้ระหว่างข้อมูล โดยข้อมูลดังกล่าวต้องเป็นนิพจน์ที่นำมาเปรียบเทียบได้ ซึ่งในที่นี้ข้อมูลที่เป็นตัวเลขจะสามารถนำมาเปรียบเทียบกันด้วยตัวเปรียบเทียบทุกแบบ ได้แก่ “==”, “!=”, “>”, “<”, “>=”, “<=” ในขณะที่ข้อมูลชนิดอื่น ๆ สามารถเปรียบเทียบความเท่ากันด้วยตัวเปรียบเทียบ “==” และ “!=” เท่านั้น

ตัวดำเนินการ

ตัวดำเนินการของภาษาอธิบายภาคีแบ่งเป็น ตัวดำเนินการเชิงตรรกะ ตัวดำเนินการคำนวณ และตัวดำเนินการจัดกลุ่ม

- ตัวดำเนินการเชิงตรรกะใช้สำหรับดำเนินการกับนิพจน์แบบบูล โดยแบ่งเป็นตัวดำเนินการทวิภาคสำหรับเชื่อม 2 นิพจน์แบบบูลเข้าด้วยกัน ประกอบด้วยตัวดำเนินการ “&&” และ “||” ซึ่งแทนตัวดำเนินการ "และ" และ "หรือ" ตามลำดับ และตัวดำเนินการเอกภาค “!” สำหรับเปลี่ยนผลลัพธ์ของนิพจน์เป็นค่าตรงข้าม
- ตัวดำเนินการคำนวณประกอบด้วย “+”, “-“, “*”, “/” และ “%” ใช้เชื่อมระหว่างสัญพจน์ชนิด Decimal เข้ากันเป็นนิพจน์เชิงตัวเลข ซึ่งมีชนิดข้อมูลเป็น Decimal
- ตัวดำเนินการจัดกลุ่มใช้สัญลักษณ์ "(" และ ")" ใช้สำหรับเขียนครอบนิพจน์ใด ๆ เพื่อระบุให้ตัวแปลภาษาทำการหาค่าของนิพจน์ที่ถูกครอบไว้ก่อน

โครงสร้างเงื่อนไข

ในที่นี้ใช้โครงสร้างแบบ If-Else ดังรูปที่ 11 เหมือนในภาษาโปรแกรมมิ่งทั่วไป โดยเงื่อนไขต้องเป็นนิพจน์แบบบูลเท่านั้น และเนื้อหาภายในของบล็อก If และบล็อก Else เป็นลำดับพฤติกรรมอันประกอบด้วยฟังก์ชั่นการกระทำและโครงสร้างเงื่อนไขอื่น ๆ บล็อก Else อาจละได้ในกรณีที่ไม่ต้องการลำดับพฤติกรรมสำหรับกรณีที่เงื่อนไขไม่เป็นจริง

```
if(Random(DecimalSet(1,2,1)) == 1){
    Goto(.feather);
}else{          Goto(.tomahawk);
}
```

รูปที่ 11 ตัวอย่างโครงสร้างเงื่อนไข

โครงสร้างวนซ้ำ

ในที่นี้ใช้สัญพจน์ loop โดยกำหนดจำนวนรอบทำซ้ำด้วยนิพจน์จำนวนนับ และเนื้อหาภายในบล็อกเป็นลำดับพฤติกรรมที่ต้องการทำซ้ำ

ซิงเกิ้ลคิวรี (Single query)

เขียนด้วยสัญลักษณ์ “$” สำหรับใช้ต่อท้ายฟังก์ชั่นข้อมูลที่ใช้งานภายในฟังก์ชั่นการกระทำระยะยาว (Spanned action) เพื่อให้ข้อมูลถูกดึงมาครั้งเดียวเมื่อเริ่มการกระทำ การทำงานของฟังก์ชั่นการกระทำระยะยาวและซิงเกิ้ลคิวรีจะกล่าวถึงในหัวข้อ 4.4.5

ตัวอย่างของภาคีที่อธิบายด้วยภาษาอธิบายภาคีเป็นไปดังรูปที่ 12 ซึ่งในบล็อกเริ่มต้นของภาคีศัตรู Watton จาก Megaman 4 ที่ใช้เป็นตัวอย่างนี้จะเห็นการกำหนดค่าคุณสมบัติต่าง ๆ ด้วยฟังก์ชั่น Set และมีฟังก์ชั่นที่น่าสนใจ เช่น Spawn ซึ่งใช้สำหรับเพิ่มภาคีใหม่ลงในโลกของเกม

```
.Watton{
    .init{
        Set("texture", DynamicFilter("this"), 4);
        Set("position", DynamicFilter("this"), "c(400,200)");
        Set("direction", DynamicFilter("this"),  TurnToPlayer(DirectionSet("H")));
        Set("collider", DynamicFilter("this"),  "48,48");
        Set("gravityeff", DynamicFilter("this"),  0);
        Set("phasing", DynamicFilter("this"),  true);
        Set("hp", DynamicFilter("this"), 10);
        Set("attacker", DynamicFilter("this"), true);
        Set("atk", DynamicFilter("this"), 10);
        Set("defender", DynamicFilter("this"), true);
    }
    .state0{
        .seq0{
            RunStraight(Get("direction", DynamicFilter("this")), 2, TimePass() >= 150);
            FlipDirection("H");
        }
        .seq1{
            Wait(TimePass() >= 90);
            Spawn(.BulletPod, Get("position", DynamicFilter("this")));
        }
    }
}
.BulletPod{
    .init{
        Set("texture", DynamicFilter("this"), 3);
        Set("collider", DynamicFilter("this"),  "24,24");
        Set("gravityeff", DynamicFilter("this"),  1);
        Set("projectile", DynamicFilter("this"),  true);
        Set("invul", DynamicFilter("this"),  true);
    }
    .state0{
        .seq0{ Jump(Get("position", DynamicFilter("this")), 32, 5, false); }
        .seq1{
            if(Abs(DistanceToPlayer("Y")) <= 30){
                Despawn();
            }
        }
    }
    .des{
        Spawn(.Bullet_Watton, Get("position", DynamicFilter("this")), "0");
        Spawn(.Bullet_Watton, Get("position", DynamicFilter("this")), "-45");
        Spawn(.Bullet_Watton, Get("position", DynamicFilter("this")), "-90");
        Spawn(.Bullet_Watton, Get("position", DynamicFilter("this")), "-135");
        Spawn(.Bullet_Watton, Get("position", DynamicFilter("this")), "-180");
    }
}
.Bullet_Watton{
    .init{
        Set("texture", DynamicFilter("this"), 6);
        Set("collider", DynamicFilter("this"), "24,24");
        Set("projectile", DynamicFilter("this"), true);
        Set("attacker", DynamicFilter("this"), true);
        Set("atk", DynamicFilter("this"), 1);
    }
    .state0{
        .seq0{ RunStraight(Get("direction", DynamicFilter("this")), 3, false); }
    }
}
```

รูปที่ 12 ตัวอย่างภาคี Watton

4.4.5 ข้อกำหนดสำหรับตัวแปลภาษาและเกมเพื่อใช้งานภาษาอธิบายภาคี

หัวข้อนี้จะระบุข้อกำหนดสำหรับตัวแปลภาษาและเกมที่จะนำภาษาอธิบายภาคีพร้อมฟังก์ชั่นการกระทำและฟังก์ชั่นข้อมูลที่ระบุไว้ตามแบบจำลองไปใช้ ซึ่งเป็นข้อกำหนดที่ต้องเพิ่มเติมจากรายละเอียดของแบบจำลองภาคีที่ได้กล่าวถึงไปแล้ว เพื่อให้งานวิทยานิพนธ์นี้มีมาตรฐานชัดเจนและสามารถทำซ้ำได้ เกมทดสอบเพื่อวัดผลศัตรูที่ใช้ในงานนี้จะทำตามข้อกำหนดนี้

ข้อกำหนดสำหรับเกม

- เป็นเกมที่เล่นบนพื้นที่สองมิติเท่านั้น
- หน่วยวัดระยะทางของเกมต้องใช้หน่วยพิกเซล (Pixel) ซึ่งส่งผลให้ข้อมูลทั้งหมดที่เกี่ยวข้องกับตำแหน่ง ระยะทาง และขนาด เช่น ค่าคุณสมบัติตัวตรวจจับการชน หรือพารามิเตอร์ความสูงในการกระโดดของฟังก์ชั่นการกระทำ Jump จะต้องมีหน่วยเป็นพิกเซล
- หน่วยเวลาของเกมจะต้องใช้หน่วยเฟรม (Frame) โดยการทำงาน 1 รอบของเกมซึ่งประกอบด้วยการประมวลผลตรรกะและการสั่งวาดจะนับเป็น 1 เฟรม หน่วยเฟรมจะส่งผลต่อข้อมูลที่เกี่ยวข้องกับเวลา หรือความเร็ว เช่น ฟังก์ชั่นข้อมูล TimePass จะคืนจำนวนเฟรมที่ใช้สำหรับฟังก์ชั่นการกระทำที่เกี่ยวข้อง หรือพารามิเตอร์ความเร็วของฟังก์ชั่นการกระทำ RunStraight จะระบุว่าใน 1 เฟรม (หรือก็คือในการประมวลผลการกระทำในเฟรมดังกล่าว) ภาคีจะต้องเคลื่อนที่ไปไกลกี่พิกเซล เป็นต้น
- ระบบพิกัดใช้แบบคาร์ทีเซียน โดยค่า x เพิ่มมากขึ้นเมื่อไปทางขวา และค่า y เพิ่มมากขึ้นเมื่อขึ้นข้างบน
- ตัวตรวจจับการชนมีรูปร่างเป็นสี่เหลี่ยมและมีจุดอ้างอิงอยู่ที่ซ้ายล่างเสมอ ดังนั้น หากระบุว่า ตัวตรวจจับการชนอยู่ที่ตำแหน่ง (100,150) นั่นหมายถึงว่า จุดซ้ายล่างของตัวตรวจจับการชนจะอยู่ที่ตำแหน่งดังกล่าว

ข้อกำหนดสำหรับตัวแปลภาษา

- ลำดับความสำคัญของตัวดำเนินการ ให้แปลตัวดำเนินการที่มีความสำคัญมากกว่าก่อน โดยเรียงลำดับจากมากที่สุดไปน้อยที่สุดดังนี้: ตัวดำเนินการจัดกลุ่ม **>** สัญพจน์และฟังก์ชั่น **>** “!” และ “-“ (กรณีที่ใช้เป็นตัวดำเนินการเอกภาค) **>** “*” “/” และ “%” **>** “+” และ “-“ **>** ตัวดำเนินการเปรียบเทียบ **>** "||" **>** "&&" เมื่อต้องทำการแปลตัวดำเนินการที่มีความสำคัญเท่ากัน ให้ทำจากซ้ายไปขวา
- การแปลผลฟังก์ชั่น ให้แปลผลพารามิเตอร์ก่อน โดยเริ่มจากพารามิเตอร์ตัวซ้ายสุดเป็นตัวแรก
- การแปลผลบล็อกลำดับทั้งหมดในสถานะหนึ่งๆให้ผู้เล่นรู้สึกว่าทุกลำดับพฤติกรรมนั้นทำงานคู่ขนานกัน ให้ใช้วิธีแปลบล็อกลำดับครั้งละบล็อกจนครบทุกบล็อกที่มีอยู่ให้เสร็จสิ้นภายในเฟรมปัจจุบัน
- การแปลผลบล็อกลำดับหนึ่ง ๆ เมื่อแปลผลจนจบบล็อกให้กลับไปเริ่มแปลผลใหม่ ทำเช่นนี้ไปเรื่อย ๆ จนกว่าสถานะของตัวแปลภาษาจะตรงกับเงื่อนไขพัก จึงหยุดพักการแปลผลบล็อกปัจจุบันชั่วคราว และแปลผลบล็อกถัดไป โดยตัวแปลภาษาจะต้องจดจำฟังก์ชั่นการกระทำสุดท้ายที่แปลผลไปแล้วของบล็อกปัจจุบันได้

- เงื่อนไขพักมี 2 เงื่อนไข เมื่อเงื่อนไขใดเงื่อนไขหนึ่งเป็นจริง ตัวแปลภาษาจึงหยุดแปลผลบล็อกปัจจุบันได้ เงื่อนไขที่หนึ่งคือ ตัวแปลภาษาแปลจนจบบล็อกปัจจุบัน เงื่อนไขที่สองคือ ตัวแปลภาษาได้ทำการแปลฟังก์ชั่นการกระทำระยะยาว (Spanned action) ในเฟรมปัจจุบันไปแล้ว
- ฟังก์ชั่นการกระทำระยะยาวเป็นฟังก์ชั่นการกระทำที่ถูกระบุว่าเป็นการกระทำที่ไม่สามารถทำให้เสร็จสิ้นในเวลาอันสั้นได้ ต้องแปลผลหลายเฟรมจนกว่าเงื่อนไขที่ทำให้จบฟังก์ชั่นเป็นจริงจึงจะถือว่าจบฟังก์ชั่น หากฟังก์ชั่นการกระทำยังไม่เสร็จสิ้น ตัวแปลภาษาจะไม่แปลผลคำสั่งต่อไปแต่ยังคงแปลผลฟังก์ชั่นเดิม ฟังก์ชั่นการกระทำระยะยาวแต่ละฟังก์ชั่นจะมีเงื่อนไขจบฟังก์ชั่นแตกต่างกันออกไป เช่น อาจมีพารามิเตอร์ actionEnd ที่รับนิพจน์แบบบูลเพื่อใช้ระบุว่าการกระทำเสร็จสิ้นแล้วหรือไม่ หรือบางฟังก์ชั่นอาจถูกกำหนดให้ทำงานถาวรโดยไม่จบการกระทำก็ได้ นอกจากนี้ ทุกครั้งที่ตัวแปลภาษายังคงแปลผลฟังก์ชั่นการกระทำระยะยาวตัวเดิม พารามิเตอร์ทั้งหมดจะต้องถูกแปลใหม่เพื่อให้ข้อมูลที่ได้มีความทันสมัย สำหรับรายการของฟังก์ชั่นการกระทำที่เป็นการกระทำระยะยาวจะระบุไว้ในภาคผนวกส่วนฟังก์ชั่นการกระทำ
- กรณีที่ฟังก์ชั่นข้อมูลที่ใช้เป็นพารามิเตอร์ของฟังก์ชั่นการกระทำระยะยาวมีสัญลักษณ์ซิงเกิ้ลคิวรีต่อท้าย ฟังก์ชั่นดังกล่าวจะถูกแปลผลเพียงครั้งเดียวเมื่อการกระทำเริ่มต้นและใช้ค่าเดิมจนกว่าการกระทำจะจบลง (ตัวแปลภาษาเริ่มแปลผลคำสั่งถัดไป)
- การแปลบล็อกเริ่มต้นและบล็อกทำลาย ให้ทำการแปลจนเงื่อนไขของตัวแปลภาษาตรงกับเงื่อนไขพัก แล้วจึงทำการเปลี่ยนสถานะไปยังสถานะเริ่มต้นสำหรับกรณีของบล็อกเริ่มต้น หรือนำภาคีออกจากโลกของเกมทันทีสำหรับกรณีของบล็อกทำลาย

บทที่ 5
การทำเหมืองข้อมูลบนชุดข้อมูลศัตรู

จุดประสงค์ของงานวิทยานิพนธ์นี้คือการสร้างรูปแบบพฤติกรรมของศัตรูที่ใช้งานในเกมได้และเป็นที่ยอมรับของผู้เล่น คำถามสำคัญในการสร้างศัตรูที่เป็นที่ยอมรับของผู้เล่น คือ รูปแบบพฤติกรรมลักษณะใดทำให้ศัตรูเป็นที่ยอมรับของผู้เล่น งานวิทยานิพนธ์นี้หาคำตอบของคำถามดังกล่าวโดยใช้การทำเหมืองข้อมูลวิธีต่าง ๆ บนชุดข้อมูลศัตรูซึ่งสร้างขึ้นจากศัตรูที่ผู้เล่นยอมรับได้โดยใช้ภาษาอธิบายภาคีเพื่อหาความสัมพันธ์ของพฤติกรรมที่พบบ่อย ความสัมพันธ์ดังกล่าวจะถูกใช้ในการสร้างรูปแบบพฤติกรรมอัตโนมัติต่อไป

5.1 ชุดข้อมูลศัตรู

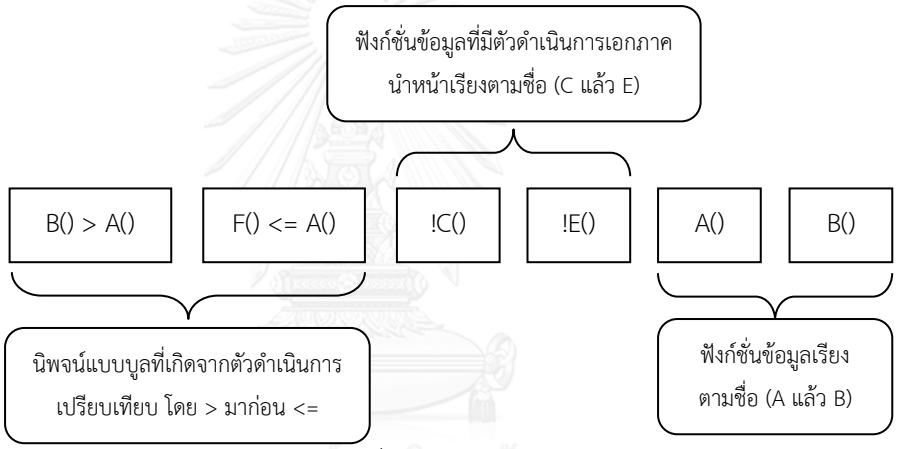
ชุดข้อมูลศัตรูที่งานวิทยานิพนธ์นี้ใช้สร้างขึ้นจากศัตรูในเกมแอ็คชั่นสองมิติเน้นตัวละครที่งานวิทยานิพนธ์นี้นำมาวิเคราะห์ซึ่งเป็นเกมที่ได้รับความนิยม และเนื่องจากศัตรูเป็นองค์ประกอบสำคัญหลักของเกมลักษณะนี้ ในที่นี้จึงถือว่าความนิยมของเกมส่วนหนึ่งมาจากการออกแบบศัตรูที่ดีที่ผู้เล่นยอมรับได้ด้วย ดังนั้นชุดข้อมูลนี้จึงนับได้ว่าเป็นชุดข้อมูลศัตรูที่ผู้เล่นยอมรับได้

งานวิทยานิพนธ์นี้จะทำซ้ำศัตรูส่วนหนึ่งจากเกมทั้ง 5 ที่นำมาวิเคราะห์ ได้แก่ Metroid, Super C, Megaman, Megaman 4 และ Shovel Knight เป็นจำนวนทั้งสิ้น 138 ตัว โดยใช้ภาษาอธิบายภาคีในการอธิบายรูปแบบพฤติกรรมเพื่อใช้เป็นชุดข้อมูลศัตรู และเนื่องจากการอธิบายพฤติกรรมหนึ่ง ๆ ด้วยภาษาอธิบายภาคีอาจสามารถทำได้หลายวิธี ซึ่งอาจส่งผลให้การทำเหมืองข้อมูลไม่มีประสิทธิภาพเพียงพอ งานวิทยานิพนธ์นี้จึงกำหนดแนวทางสำหรับการใช้ภาษาอธิบายภาคีเพื่อให้การอธิบายภาคีเป็นไปในทางเดียวกันโดยให้ปฏิบัติตามลำดับข้อ ดังนี้

1) ไฟล์สคริปต์ 1 ไฟล์ใช้แทนศัตรู 1 ตัว ซึ่งอาจมีบล็อกภาคีที่เกี่ยวข้องหลายภาคีได้ บล็อกภาคีที่ใช้อธิบายตัวศัตรู ให้ถือว่าเป็นภาคีหลัก และต้องเป็นบล็อกภาคีแรกสุดในไฟล์สคริปต์เสมอ
2) หากมีวิธีอธิบายที่ใช้จำนวนฟังก์ชั่นการกระทำเท่ากันหลายวิธี ให้เลือกวิธีที่ใช้บล็อกลำดับน้อยที่สุด
3) การอธิบายพฤติกรรมให้เลือกวิธีที่ใช้จำนวนฟังก์ชั่นการกระทำน้อยที่สุด
4) หากมีการใช้งานฟังก์ชั่นเปลี่ยนสถานะ ให้ใช้ในบล็อกลำดับแรกสุดของบล็อกสถานะก่อน แล้วจึงพิจารณาบล็อกถัดไปเมื่อต้องมีฟังก์ชั่นเปลี่ยนสถานะเพิ่มในบล็อกลำดับอื่น ๆ เช่น สถานะ .state0 มี 3 บล็อกลำดับ ได้แก่ .seq0 .seq1 และ .seq2 และมี 2 ลำดับที่ต้องใช้ฟังก์ชั่น Goto จะต้องกำหนดให้ 2 ลำดับดังกล่าว คือ .seq0 และ .seq1 เท่านั้น
5) หากมีการใช้ตัวดำเนินการเปรียบเทียบ และมีการใช้สัญพจน์ในการเปรียบเทียบ ให้ใช้สัญพจน์เป็นนิพจน์ขวาเสมอ เช่น 3 == FuncA() ให้เขียนเป็น FuncA() == 3
6) หากมีการใช้ตัวดำเนินการเปรียบเทียบกรณีมากกว่าหรือน้อยกว่า ให้เลือกใช้ตัวดำเนินการ >= หรือ > ก่อน หากไม่สามารถอธิบายพฤติกรรมได้จึงเลือกใช้ <= หรือ < เช่น FuncA() < FuncB() ให้เขียนเป็น FuncB() > FuncA() แทน

7) หลีกเลี่ยงการใช้ตัวดำเนินการคำนวณกับฟังก์ชั่นข้อมูล หากจำเป็นต้องใช้ ให้ใช้กับฟังก์ชั่นข้อมูลที่เป็นนิพจน์ขวา เช่น FuncA() + 5 > FuncB() ให้เขียนเป็น FuncA() > FuncB() – 5
8) หากมีการใช้ตัวดำเนินการทวิภาคเชิงตรรกะ ("&&" และ "||") เชื่อมหลายนิพจน์เข้าด้วยกัน ให้จัดลำดับตามลำดับข้อต่อไปนี้จากบนลงล่าง
    - นิพจน์แบบบูลที่เกิดจากตัวดำเนินการเปรียบเทียบ โดยให้ลำดับจากซ้ายไปขวาตามตัวดำเนินการ ดังนี้ >= > <= < == !=
    - ฟังก์ชั่นข้อมูลที่มีตัวดำเนินการเอกภาค ("!") นำหน้า โดยให้เรียงนิพจน์จากซ้ายไปขวาโดยเรียงชื่อฟังก์ชั่นตามลำดับอักษร
    - ฟังก์ชั่นข้อมูล โดยให้เรียงนิพจน์จากซ้ายไปขวาโดยเรียงชื่อฟังก์ชั่นตามลำดับอักษร

    ตัวอย่าง เช่น (B() > A()) && (F() <= A()) && !C() && !E() && A() && B() เป็นวิธีการเรียงที่ถูกต้อง รายละเอียดการจัดเรียงเป็นไปดังรูปที่ 13



รูปที่ 13 รูปการเรียงนิพจน์

5.2 ความสัมพันธ์ของพฤติกรรม

ในเกมแอ็คชั่นแบบเน้นตัวละคร ผู้เล่นจะต้องเรียนรู้และโต้ตอบกับศัตรูที่มีรูปแบบพฤติกรรมที่ตายตัว วิธีที่ผู้เล่นโต้ตอบกับศัตรูจะขึ้นอยู่กับรูปแบบพฤติกรรมของศัตรู หากศัตรูมีรูปแบบพฤติกรรมที่คล้ายคลึงกัน ผู้เล่นจะใช้แนวคิดเดียวกันในการโต้ตอบกับศัตรู เช่น ศัตรู Metall ใน Megaman, ศัตรู Pakatto24 ใน Megaman 4 และศัตรูป้อมปืนผนังใน Super C ต่างก็มีรูปแบบพฤติกรรมที่มีลักษณะเป็นการหลบเข้าเกราะกำบังแล้วออกมาจากเกราะเพื่อโจมตีผู้เล่นสลับกันไปมา ผู้เล่นจะใช้วิธีการหลอกล่อให้ศัตรูออกมาจากเกราะกำบังแล้วโจมตีสวนกลับเพื่อจัดการกับศัตรูทั้งสามชนิดนี้

การที่ศัตรูในเกมต่าง ๆ ซึ่งเป็นที่นิยมมีรูปแบบพฤติกรรมที่คล้ายคลึงแสดงให้เห็นว่า มีแบบแผนบางอย่างในการออกแบบรูปแบบพฤติกรรมให้ออกศัตรูมาเป็นที่ยอมรับได้โดยผู้เล่น โดยที่แบบแผนดังกล่าวไม่ได้มีเพียงหนึ่งเดียวเนื่องจากศัตรูแต่ละชนิดในเกมที่นำมาวิเคราะห์ต่างก็มีรูปแบบพฤติกรรมที่จำเป็นต้องโต้ตอบด้วยวิธีที่แตกต่างกันออกไป งานวิทยานิพนธ์นี้จะใช้วิธีการทำเหมืองข้อมูลบนชุดข้อมูลศัตรูเพื่อค้นหาแบบแผนทั้งหลายดังกล่าวบนสมมติฐานที่ว่า “แบบแผนที่พบบ่อย” น่าจะเป็นแนวทางที่เหมาะสมสำหรับใช้ออกแบบรูปแบบพฤติกรรมของศัตรู

เนื่องจากการทำเหมืองข้อมูลจำเป็นต้องรู้ว่าจุดประสงค์ของการทำเหมืองข้อมูลคืออะไร มีข้อมูลขาเข้าและขาออกเป็นอย่างไร จึงจะสามารถเลือกขั้นตอนวิธีสำหรับการทำเหมืองข้อมูลได้ ในที่นี้มีจุดประสงค์เพื่อค้นหาแบบแผนสำหรับออกแบบรูปแบบพฤติกรรม คำถามสำคัญจึงเป็น “แบบแผนซึ่งเป็นข้อมูลขาออกที่ต้องการค้นหาจากชุดข้อมูลมีลักษณะเป็นอย่างไร” เมื่อกำหนดให้ข้อมูลขาเข้าเป็นชุดข้อมูลศัตรูซึ่งอยู่ในรูปของภาษาอธิบายภาคี

ภาษาอธิบายภาคีในงานวิทยานิพนธ์นี้ใช้สำหรับอธิบายรูปแบบพฤติกรรมโดยใช้การเรียงต่อฟังก์ชั่นการกระทำและเงื่อนไขออกมาเป็นลำดับพฤติกรรม ลำดับพฤติกรรมหลายลำดับจะถูกรวมเป็นพฤติกรรมคู่ขนานของสถานะหนึ่ง ๆ สถานะหลายสถานะร่วมกับการตั้งค่าคุณสมบัติในบล็อกเริ่มต้นจะถูกรวมเป็นภาคีศัตรูหนึ่งชนิด ศัตรูบางชนิดอาจมีความสามารถในการสร้างภาคีชนิดอื่น ๆ ด้วยฟังก์ชั่นการกระทำ Spawn เพื่อให้ภาคีที่สร้างใหม่แสดงผลการกระทำไปพร้อมกับตนได้ หากพิจารณาลักษณะของภาษาอธิบายภาคีและพฤติกรรมผลลัพธ์ที่เกิดจากลักษณะของภาษา จะพบว่าความแตกต่างในเชิงความหมายของพฤติกรรมหรือพฤติกรรมผลลัพธ์ที่เกิดขึ้น เกิดจากการเลือกใช้ฟังก์ชั่นการกระทำที่แตกต่างกันและ/หรือความสัมพันธ์ระหว่างองค์ประกอบภายในภาษาอธิบายภาคี ดังนี้

ลำดับของฟังก์ชั่นการกระทำและโครงสร้างเงื่อนไข

สามารถยกตัวอย่างความแตกต่างที่เกิดจากลำดับ เช่น มีฟังก์ชั่นการกระทำวิ่ง, กระโดด และยิง หากนำมาเรียงเป็นวิ่ง, กระโดด, ยิง ศัตรูจะยิงหลังจากกระโดด ในขณะที่หากเรียงการกระทำเป็นกระโดด, วิ่ง, ยิง ซึ่งศัตรูจะยิงหลังจากวิ่ง เป็นต้น ความแตกต่างเช่นนี้ทำให้ผู้เล่นต้องโต้ตอบด้วยการรอจังหวะของพฤติกรรมที่ต่างกัน

ฟังก์ชั่นการกระทำที่คู่ขนานกัน

ส่งผลให้เกิดความแตกต่างได้ในกรณีที่ฟังก์ชั่นการกระทำที่เกิดคู่ขนานกันมีความแตกต่างกัน เช่น ฟังก์ชั่นการกระทำกระโดดคู่ขนานกับยิง และฟังก์ชั่นการกระทำวิ่งคู่ขนานกับยิง ย่อมส่งผลให้วิถีกระสุนที่ได้แตกต่างกัน ถึงแม้จะยิงกระสุนแบบเดียวกันเนื่องจากจุดยิงแตกต่างกัน

ฟังก์ชั่นข้อมูลที่เลือกใช้

แม้ฟังก์ชั่นการกระทำหรือฟังก์ชั่นข้อมูลจะเป็นฟังก์ชั่นเดียวกัน หากเลือกใช้ข้อมูลที่แตกต่างกันย่อมมีความหมายแตกต่างกันและอาจทำให้เกิดผลลัพธ์ที่แตกต่างกัน เช่น ในการกระทำระยะยาว RunStraight หากเป็น

```
RunStraight(Get(“direction”, DynamicFilter(“this”)), 5, TimePass() > 40)
```

ซึ่งสั่งให้ภาคีวิ่งตรงไปข้างหน้าเป็นระยะเวลา 40 เฟรม ย่อมได้ผลลัพธ์ต่างไปจาก

```
RunStraight(Get(“direction”, DynamicFilter(“this”)), 5, SurfaceInDir(Get(“direction”, DynamicFilter(“this”))))
```

ซึ่งสั่งให้ภาคีวิ่งตรงไปข้างหน้าเช่นเดียวกับกรณีแรก แต่ใช้เงื่อนไขจบการกระทำแตกต่างกัน โดยกรณีนี้จะหยุดวิ่งเมื่อเจอกับโครงสร้างพื้นที่ที่ผ่านไม่ได้เท่านั้น

แบบแผนของการออกแบบพฤติกรรมที่ได้ยกตัวอย่างไปก่อนหน้านี้เกิดจากการจัดเรียงองค์ประกอบภายในภาษาอธิบายภาคีอย่างเป็นระบบ (ในกรณีของตัวอย่างศัตรู Metall และศัตรู Pakatto24 เกิดจากการจัดเรียงการ

กระทำภายในลำดับพฤติกรรมแบบเดียวกัน) การมีแบบแผนได้หลายแบบหมายความว่ามีวิธีในการจัดเรียงองค์ประกอบให้เกิดเป็นรูปแบบพฤติกรรมที่แตกต่างกันได้หลายวิธี และวิธีการจัดเรียงองค์ประกอบของภาษาอธิบายภาคีให้เกิดความแตกต่างของรูปแบบพฤติกรรมสามารถทำได้โดยการเลือกใช้ฟังก์ชั่นการกระทำที่แตกต่างกันร่วมกับความสัมพันธ์ที่ทำให้เกิดพฤติกรรมที่แตกต่างทั้ง 3 รูปแบบข้างต้น

ดังนั้น แบบแผนของรูปแบบพฤติกรรมที่ต้องการจากการทำเหมืองข้อมูลในที่นี้จะอยู่ในรูปความสัมพันธ์ของฟังก์ชั่นการกระทำทั้ง 3 รูปแบบอันได้แก่ ลำดับของฟังก์ชั่นการกระทำ ฟังก์ชั่นการกระทำที่คู่ขนานกัน และการเลือกใช้ฟังก์ชั่นข้อมูล

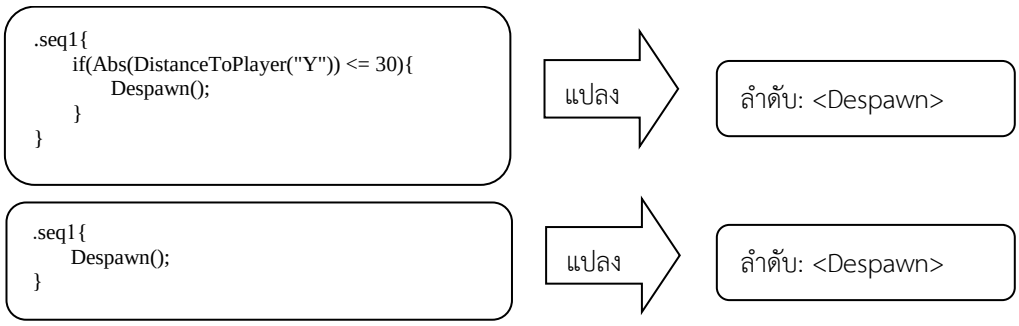
5.3 การทำเหมืองข้อมูลสำหรับลำดับของฟังก์ชั่นการกระทำ

การทำเหมืองข้อมูลสำหรับลำดับฟังก์ชั่นการกระทำมีจุดประสงค์เพื่อค้นหาว่าการจัดเรียงฟังก์ชั่นการกระทำที่พบได้บ่อยในชุดข้อมูลเป็นอย่างไร ซึ่งข้อมูลที่จะถูกนำมาใช้เป็นข้อมูลขาเข้า คือ ลำดับพฤติกรรมทั้งหมดที่พบในชุดข้อมูล ซึ่งประกอบด้วยบล็อกทำลายและบล็อกลำดับทั้งหมดที่ปรากฏในชุดข้อมูล

5.3.1 การทำเหมืองข้อมูลบนข้อมูลลำดับพฤติกรรม

เนื่องจากความสัมพันธ์ผลลัพธ์อยู่ในรูปของลำดับ งานวิทยานิพนธ์นี้จึงใช้ขั้นตอนวิธี PrefixSpan [8] ซึ่งเป็นขั้นตอนวิธีสำหรับการหารูปแบบจากฐานข้อมูลลำดับ ดังนั้นจึงต้องมีการแปลงลำดับพฤติกรรมให้อยู่ในรูปของลำดับตามนิยามของปัญหาทางด้านการทำเหมืองข้อมูลสำหรับข้อมูลแบบลำดับ

วิธีที่ง่ายที่สุดในการแปลงลำดับพฤติกรรมไปเป็นข้อมูลลำดับคือการไล่รหัสจากบรรทัดแรกสุดของลำดับจนถึงบรรทัดสุดท้ายของลำดับ หากพบฟังก์ชั่นการกระทำให้แปลงฟังก์ชั่นการกระทำเป็นไอเท็มเซ็ตที่ประกอบด้วยไอเท็มที่อยู่ในรูปของชื่อฟังก์ชั่นนั้นแล้วเพิ่มลงในข้อมูลลำดับ โดยวิธีการนี้มีข้อเสีย คือ ไม่สามารถแยกความแตกต่างระหว่างฟังก์ชั่นที่อยู่/ไม่ได้อยู่ภายใต้โครงสร้างเงื่อนไข รูปที่ 14 แสดงให้เห็นลำดับซึ่งแปลงมาจากฟังก์ชั่นเดียวกันที่อยู่และไม่อยู่ใต้โครงสร้างเงื่อนไข



รูปที่ 14 การแปลงลำดับพฤติกรรมโดยใช้เฉพาะชื่อฟังก์ชั่น

ข้อเสียนี้แก้ไขได้โดยการแปลงข้อมูลฟังก์ชั่นการกระทำและโครงสร้างที่ครอบฟังก์ชั่นให้กลายเป็นอาเรย์ของไบต์ (Array of byte) จากนั้นจึงแปลงอาเรย์ให้เป็นสตริงด้วยการเข้ารหัสแบบ ASCII และใช้สตริงดังกล่าวนี้เป็นไอเท็มในลำดับ วิธีการแปลงฟังก์ชั่นการกระทำเป็นอาเรย์ของไบต์ทำได้ ดังนี้

กำหนดให้ฟังก์ชั่นการกระทำ $Action$ ถูกครอบไว้ด้วยหลายโครงสร้าง $\{c_1, c_2, ...\}$ เมื่อ c_1 เป็นโครงสร้างชั้นนอกสุด สามารถแปลง $Action$ ให้อยู่ในรูปอาเรย์

$$[ActionID, s_1, ex_1, s_2, ex_2, ...]$$

เมื่อ

- $ActionID$ คือ ตัวเลขแทนฟังก์ชั่นการกระทำ ภาคผนวกส่วนฟังก์ชั่นการกระทำได้แสดงตัวเลขสำหรับแต่ละฟังก์ชั่นการกระทำ
- s_i คือ ตัวเลขแทนประเภทของโครงสร้าง c_i มีค่าได้ 4 ค่า ได้แก่
  - 127 เมื่อ c_i เป็นโครงสร้างเงื่อนไขที่ไม่มีบล็อก Else
  - 126 เมื่อ c_i เป็นโครงสร้างเงื่อนไขที่มีบล็อก Else และ Action อยู่ภายใต้บล็อก If ของ c_i
  - 125 เมื่อ c_i เป็นโครงสร้างเงื่อนไขที่มีบล็อก Else และ Action อยู่ภายใต้บล็อก Else ของ c_i
  - 124 เมื่อ c_i เป็นโครงสร้างวนซ้ำ
- ex_i คือ ส่วนของอาเรย์แทนนิพจน์แบบบูลอันเป็นเงื่อนไขของ c_i ซึ่งสามารถแปลงจากนิพจน์แบบบูลโดยแปลงนิพจน์ดังกล่าวให้อยู่ในรูปของนิพจน์แบบเติมหน้า (Prefix expression) ก่อน แล้วจึงแปลงนิพจน์ดังกล่าวด้วยวิธีการแปลงนิพจน์แบบเติมหน้าเป็นส่วนของอาเรย์

สำหรับการแปลงนิพจน์แบบเติมหน้าเป็นส่วนของอาเรย์ทำได้โดยวิธีต่อไปนี้

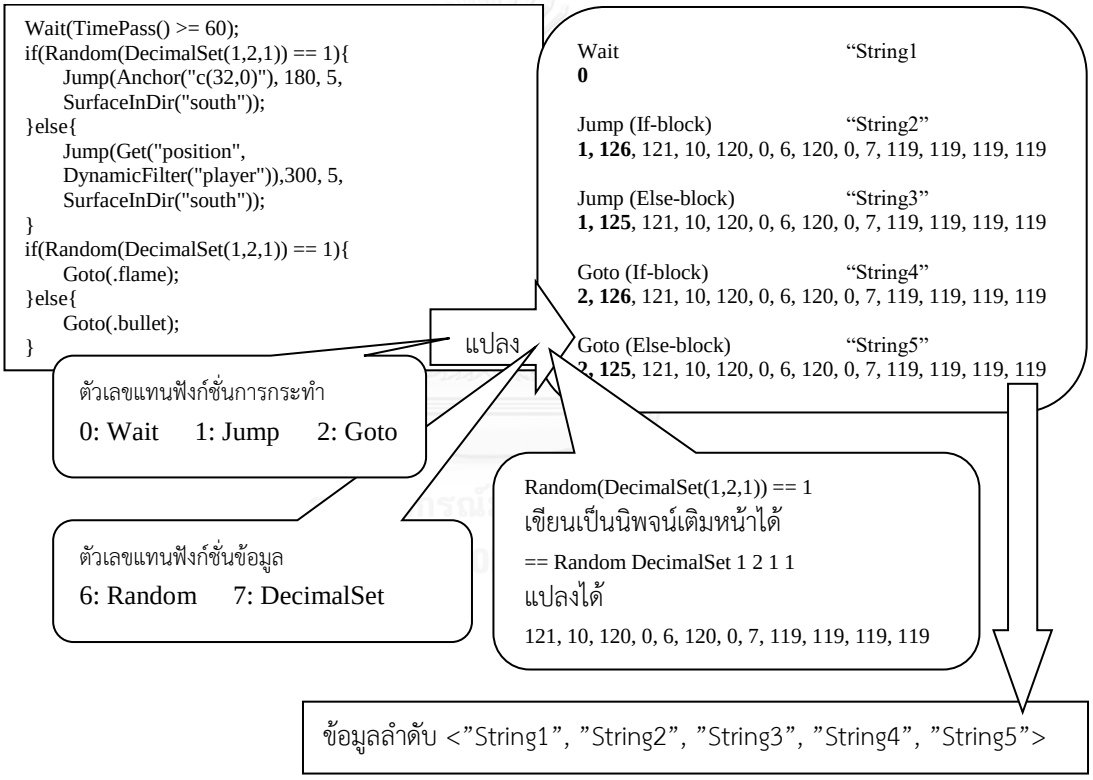
กำหนดให้นิพจน์เติมหน้า $pe = e_1 e_2 ...$ เมื่อ e_i เป็นส่วนย่อยหนึ่งของนิพจน์ สามารถแปลง pe เป็นส่วนของอาเรย์ได้โดยแปลงแต่ละส่วนย่อยเป็นตัวเลขตามแต่ประเภท ดังนี้

ประเภท	ตัวเลขที่ใช้
ตัวดำเนินการเอกภาค ! และ -	123 และ 122 ตามลำดับ
ตัวดำเนินการทวิภาค	121,x เมื่อ x คือ ตัวเลขแทนตัวดำเนินการ ดังนี้ && 0 || 1 == 10 != 11 > 12 < 13 >= 14 <= 15 + 20 - 21 * 22 / 23 % 24
ฟังก์ชั่นข้อมูล	120,y,x เมื่อ x คือ ตัวเลขแทนฟังก์ชั่นข้อมูล และ y มีค่าเป็น 0 เมื่อไม่ใช้ซิงเกิ้ลคิวรี หรือ 1 เมื่อใช้ จากนั้นให้แปลงแต่ละพารามิเตอร์เป็นตัวเลขโดยใช้วิธีแปลงนิพจน์แบบเติมหน้านี้ ภาคผนวกส่วนฟังก์ชั่นข้อมูลแสดงตัวเลขสำหรับแต่ละฟังก์ชั่น

ประเภท	ตัวเลขที่ใช้
สัญพจน์	119

ตัวอย่าง เช่น นิพจน์ > + Func()$ * 10 Func2() 8 สามารถแปลงเป็นส่วนของอาเรย์ได้ 121, 12, 121, 20, 120, 1, 1, 121, 22, 119, 120, 0, 2, 119 เมื่อกำหนดให้ฟังก์ชั่นข้อมูล Func และ Func2 มีตัวเลขแทนฟังก์ชั่นเป็น 1 และ 2 ตามลำดับ

ด้วยวิธีแปลงข้อมูลดังกล่าว ลำดับพฤติกรรมจึงสามารถแปลงให้อยู่ในรูปของข้อมูลลำดับได้โดยแต่ละไอเท็มเซ็ตในลำดับจะมีไอเท็มเพียงตัวเดียว รูปที่ 15 แสดงตัวอย่างการแปลงลำดับพฤติกรรมเป็นข้อมูลลำดับ ข้อมูลลำดับที่ได้สามารถนำเข้าสู่ขั้นตอนวิธี PrefixSpan ต่อไป



รูปที่ 15 ตัวอย่างการแปลงลำดับพฤติกรรมเป็นข้อมูลลำดับ (ตัวเลขที่ใช้และสตริงที่แสดงไม่ใช่ค่าจริง)

ผลลัพธ์จากขั้นตอนวิธี PrefixSpan จะได้ออกมาเป็นข้อมูลลำดับ ข้อมูลลำดับความยาว 1 จะถูกคัดออกเนื่องจากไม่ได้แสดงความสัมพันธ์แบบลำดับของพฤติกรรม ข้อมูลที่เหลือสามารถแปลงค่ากลับไปเป็นฟังก์ชั่นการกระทำและเงื่อนไขที่ “ไม่สมบูรณ์” และนำไปใช้งานต่อได้ แต่จะต้องมีการเติมเต็มส่วนที่ขาดหายไปซึ่งเป็นหน้าที่ของส่วนสร้างรูปแบบพฤติกรรมในบทที่ 6

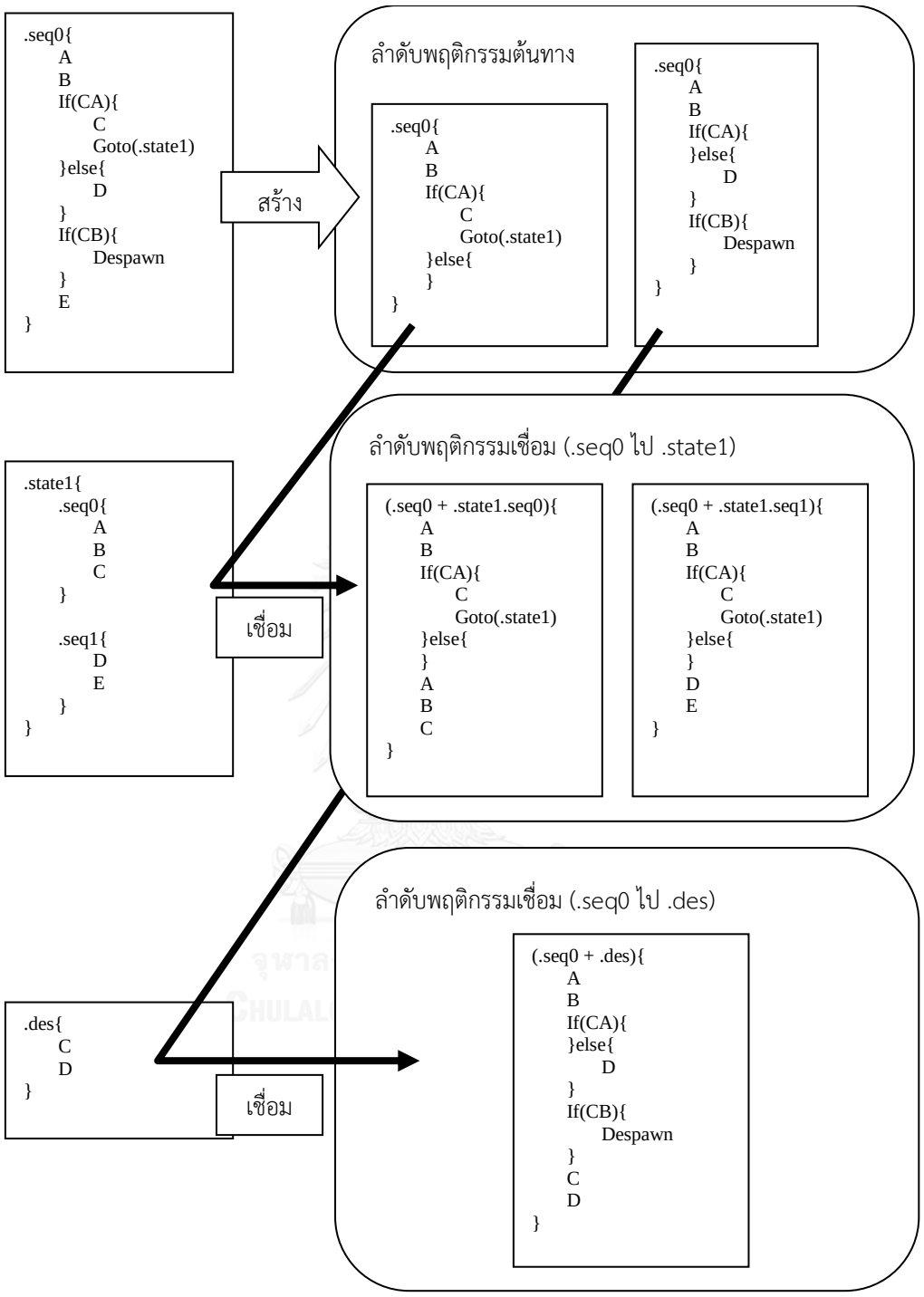
สำหรับการทำเหมืองข้อมูลบนข้อมูลลำดับพฤติกรรมในงานวิทยานิพนธ์นี้ จะใช้ข้อมูลลำดับพฤติกรรมจากทุกกลุ่มสถานะที่ปรากฏในชุดข้อมูลศัตรูเป็นข้อมูลขาเข้า และทำเหมืองข้อมูลด้วยวิธี PrefixSpan โดยใช้ค่าซัพพอร์ทเป็น 2% ของจำนวนข้อมูลขาเข้า สาเหตุที่จำเป็นต้องใช้ค่าซัพพอร์ทที่ต่ำเนื่องจากชุดข้อมูลมีจำนวนไม่มาก และมีความหลากหลาย

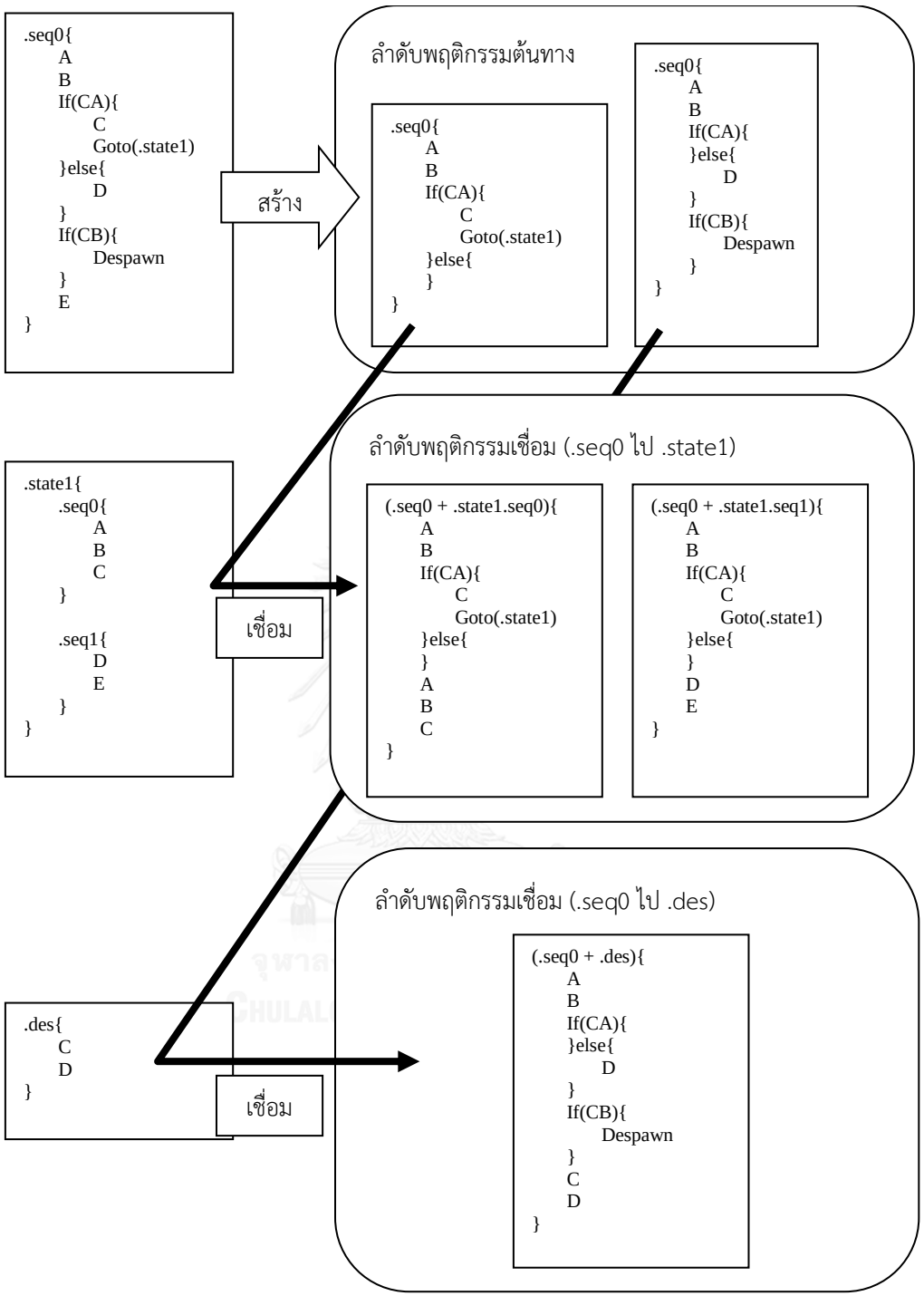
5.3.2 การทำเหมืองข้อมูลบนข้อมูลลำดับพฤติกรรมเชื่อม

นอกเหนือจากการทำเหมืองข้อมูลโดยใช้ลำดับพฤติกรรมเป็นข้อมูลนำเข้าแล้ว ในที่นี้ยังมีข้อมูลลำดับอีกลักษณะหนึ่งซึ่งเป็นลำดับพฤติกรรมที่เกิดจากฟังก์ชั่นการกระทำที่ใช้เปลี่ยนสถานะของภาคี การเปลี่ยนสถานะของภาคีทำให้พฤติกรรมผลลัพธ์ที่เกิดขึ้นเป็นเสมือนลำดับพฤติกรรมที่ประกอบด้วยลำดับพฤติกรรมที่กระทำก่อนเปลี่ยนสถานะเชื่อมต่อกับลำดับพฤติกรรมของสถานะปลายทาง ความต่อเนื่องของพฤติกรรมข้ามสถานะน่าจะมีแบบแผนเกี่ยวกับการเปลี่ยนแปลงจากชุดพฤติกรรมหนึ่งไปเป็นอีกชุดหนึ่ง ในที่นี้จึงทำเหมืองข้อมูลเพื่อค้นหาความสัมพันธ์ในลักษณะนี้เพิ่มเติมจากลำดับพฤติกรรม โดยเรียกลำดับพฤติกรรมที่เกิดจากการเชื่อมระหว่างลำดับพฤติกรรมก่อนและหลังเปลี่ยนสถานะว่า ลำดับพฤติกรรมเชื่อม (Joined behavior sequence) โดยลำดับพฤติกรรมก่อนเปลี่ยนสถานะจะเรียกว่าลำดับพฤติกรรมต้นทาง และลำดับพฤติกรรมหลังเปลี่ยนสถานะเรียกว่าลำดับพฤติกรรมปลายทาง

ลำดับพฤติกรรมเชื่อมสามารถสร้างขึ้นจากชุดข้อมูลศัตรูโดยการเลือกฟังก์ชั่นการกระทำและโครงสร้างทั้งหมดที่มีโอกาสถูกแปล (All possible program flows) ตั้งแต่เริ่มต้นลำดับจนถึงฟังก์ชั่นการกระทำที่เปลี่ยนสถานะได้ (ฟังก์ชั่น Goto และ Despawn) เป็นลำดับพฤติกรรมต้นทาง จากนั้นจึงเชื่อมต่อกับลำดับพฤติกรรมปลายทาง ซึ่งในที่นี้คือ ลำดับพฤติกรรมของสถานะปลายทาง หากสถานะปลายทางมีมากกว่า 1 ลำดับพฤติกรรมจะต้องมีการเชื่อมกับลำดับพฤติกรรมทั้งหมด ทำให้เกิดเป็นลำดับพฤติกรรมเชื่อมจำนวนเท่ากับจำนวนลำดับพฤติกรรมของสถานะปลายทาง

ในกรณีที่ลำดับพฤติกรรมต้นทางมีฟังก์ชั่นเปลี่ยนสถานะมากกว่า 1 ฟังก์ชั่น จะต้องสร้างลำดับพฤติกรรมต้นทางสำหรับทุกฟังก์ชั่น และเชื่อมทุกลำดับพฤติกรรมต้นทางกับทุกลำดับพฤติกรรมปลายทาง รูปที่ *16* แสดงตัวอย่างการสร้างลำดับพฤติกรรมเชื่อม โดยแสดงในรูปรหัสเทียม

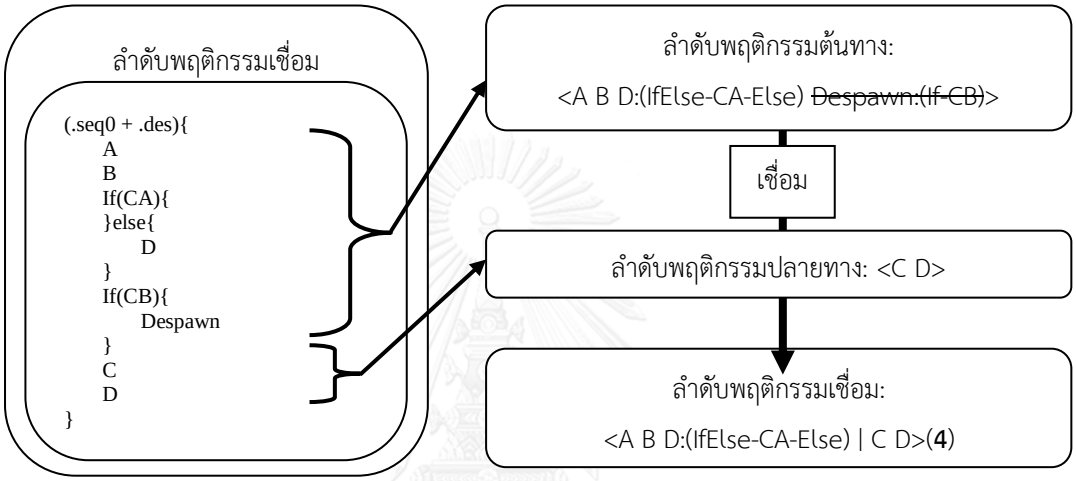

.seq0{
A
B
If(CA){
C
Goto(.state1)
}else{
D
}
If(CB){
Despawn
}
E
}
สร้าง
ลำดับพฤติกรรมต้นทาง
.seq0{
A
B
If(CA){
C
Goto(.state1)
}else{
}
}
.seq0{
A
B
If(CA){
}else{
D
}
If(CB){
Despawn
}
}
.state1{
.seq0{
A
B
C
}
.seq1{
D
E
}
}
เชื่อม
ลำดับพฤติกรรมเชื่อม (.seq0 ไป .state1)
(.seq0 + .state1.seq0){
A
B
If(CA){
C
Goto(.state1)
}else{
}
A
B
C
}
(.seq0 + .state1.seq1){
A
B
If(CA){
C
Goto(.state1)
}else{
}
D
E
}
.des{
C
D
}
เชื่อม
ลำดับพฤติกรรมเชื่อม (.seq0 ไป .des)
(.seq0 + .des){
A
B
If(CA){
}else{
D
}
If(CB){
Despawn
}
C
D
}



รูปที่ 16 ตัวอย่างการสร้างลำดับพฤติกรรมเชื่อม

เมื่อได้ลำดับพฤติกรรมเชื่อมแล้ว จึงทำการแปลงลำดับพฤติกรรมเชื่อมให้อยู่ในรูปของลำดับเชื่อม (Joined sequence) โดยงานวิทยานิพนธ์นี้กำหนดนิยามลำดับเชื่อมว่า ลำดับเชื่อม $s = \langle \alpha \mid \beta \rangle$ เมื่อ α และ β เป็นลำดับ โดย α เป็นลำดับต้นทาง และ β เป็นลำดับปลายทาง

ลำดับพฤติกรรมเชื่อมสามารถแปลงเป็นข้อมูลลำดับเชื่อมได้โดยแปลงลำดับพฤติกรรมต้นทางเป็นข้อมูลลำดับด้วยวิธีการในหัวข้อ 5.3.1 จากนั้นตัดไอเท็มเซ็ตสุดท้ายซึ่งเป็นฟังก์ชั่นเปลี่ยนสถานะออก และกำหนดให้ข้อมูลลำดับที่ได้นี้เป็นลำดับต้นทางของข้อมูลลำดับเชื่อม จากนั้นแปลงลำดับพฤติกรรมปลายทางเป็นข้อมูลลำดับและใช้เป็นลำดับปลายทางของข้อมูลลำดับเชื่อม นอกจากนี้ลำดับเชื่อมที่ได้จะถูกเสริมด้วยข้อมูลตัวเลข 1 ตัวเพื่อบอกจำนวนสถานะทั้งหมดของภาคีนี้ ดังนั้นลักษณะของลำดับเชื่อมขาเข้าสำหรับทำเหมืองข้อมูลจึงเป็น $\langle \alpha \mid \beta \rangle (i)$ โดย i เป็นข้อมูลเสริมบอกจำนวนสถานะ รูปที่ 17 แสดงตัวอย่างการแปลงลำดับพฤติกรรมเชื่อมหนึ่งของภาคีหนึ่งที่มี 4 สถานะเป็นข้อมูลลำดับเชื่อม โดยในตัวอย่างจะใช้เขียนชื่อฟังก์ชั่นลงไปแทนการแปลงข้อมูลเพื่อให้เข้าใจได้ง่าย



รูปที่ 17 ตัวอย่างการแปลงลำดับพฤติกรรมเชื่อมเป็นข้อมูลลำดับเชื่อม

การทำเหมืองข้อมูลสำหรับข้อมูลลำดับเชื่อมจะทำการหาลำดับเชื่อมย่อย γ ที่พบบ่อยจากฐานข้อมูลลำดับเชื่อม S โดยนิยามต่าง ๆ เหมือนกับการทำเหมืองข้อมูลบนข้อมูลลำดับ ยกเว้นลำดับเชื่อมย่อย (Sub-joined sequence) ซึ่งมีนิยามว่า ลำดับเชื่อม $\gamma' = \langle \alpha' \mid \beta' \rangle (j)$ เป็นลำดับเชื่อมย่อยของ $\gamma = \langle \alpha \mid \beta \rangle (i)$ เมื่อ $\alpha' \subseteq \alpha, \alpha' \neq \phi$ และ $\beta' \subseteq \beta, \beta' \neq \phi$ และ $i = j$

สำหรับขั้นตอนวิธีทำเหมืองข้อมูลจะนำเอา PrefixSpan มาแก้ไขโดยปรับปรุงการเรียกใช้ฟังก์ชั่น PrefixSpan รอบแรกสุด ให้บันทึกลำดับเชื่อม $\langle a \mid b \rangle (i)$ ที่พบบ่อยแทนการบันทึกลำดับความยาว 1 ในวิธีเดิม เพื่อเป็นการยืนยันว่าลำดับต้นทางและลำดับปลายทางของลำดับเชื่อมผลลัพธ์ไม่เป็นลำดับว่างและลำดับเชื่อมมีข้อมูลเสริม จากนั้นก่อนการเรียกใช้งาน PrefixSpan รอบถัดไป ให้สร้างฐานข้อมูลภาพฉายของฐานข้อมูลเริ่มต้นสำหรับทุกลำดับเชื่อมที่พบบ่อย โดยแบ่งเป็นฐานข้อมูลภาพฉายสำหรับลำดับต้นทางและฐานข้อมูลภาพฉายสำหรับลำดับปลายทางแยกออกจากกัน แล้วจึงใช้ขั้นตอนวิธี PrefixSpan ปกติสำหรับหาลำดับย่อยที่พบบ่อยจากฐานข้อมูลภาพฉายทั้งสอง จากนั้นทำการเชื่อมผลลัพธ์โดยใช้ผลลัพธ์จากฐานข้อมูลภาพฉายสำหรับลำดับต้นทางเป็นลำดับต้นทางของลำดับเชื่อมและผลลัพธ์จากฐานข้อมูลภาพฉายสำหรับลำดับปลายทางเป็นลำดับปลายทางของลำดับเชื่อมภายใต้เงื่อนไขว่า ลำดับเชื่อมย่อยที่ได้จะต้องเป็นลำดับเชื่อมย่อยที่พบบ่อยในฐานข้อมูลเริ่มต้น ขั้นตอนวิธีทำเหมืองข้อมูลสำหรับข้อมูลลำดับเชื่อมแสดงด้วยรหัสเทียมได้ดังขั้นตอนวิธีที่ *2* โดยในที่นี้เรียกขั้นตอนวิธีนี้ว่า JS-PrefixSpan

การทำเหมืองข้อมูลบนฐานข้อมูลลำดับเชื่อม S ซึ่งประกอบด้วยลำดับเชื่อม $s_1, s_2, ..., s_n$ และมีเซ็ตของไอเท็ม $I = \{i_1, i_2, ..., i_m\}$ และเซ็ตของตัวเลขเสริมที่เป็นไปได้ $T = \{t_1, t_2, ..., t_l\}$ กำหนดค่าซัพพอร์ทขั้นต่ำ $\min\_\sup$ เป็นดังนี้

Initialize $FreqJI1 = \phi, \mathrm{Re}sult = \phi$

For (i_a in I) do:

For (i_b in I) do:

For (t in T) do:

if($\sup(\langle i_a \mid i_b \rangle(t), S) \geq \min\_\sup$) do:

$FreqJI1 \leftarrow \langle i_a \mid i_b \rangle(t)$

$\mathrm{Re}sult \leftarrow \langle i_a \mid i_b \rangle(t)$

For($\langle i_a \mid i_b \rangle(t)$ in $FreqJI1$) do:

Create i_a -projected database of S_a $proj_a$

$\mathrm{Re}sult_a =$ PrefixSpan on $proj_a$ with itemset I and $\min\_\sup$

Create i_b -projected database of S_b $proj_b$

$\mathrm{Re}sult_b =$ PrefixSpan on $proj_b$ with itemset I and $\min\_\sup$

For(a in $\mathrm{Re}sult_a$) do:

For(b in $\mathrm{Re}sult_b$) do:

If($|a.SIDs \cap b.SIDs| \geq \min\_\sup$)

$\mathrm{Re}sult \leftarrow \langle i_a + a \mid i_b + b \rangle(t)$

Return $\mathrm{Re}sult$ as result

เมื่อ

$S_a = \{\langle SID, a \rangle \mid \forall \langle SID, s \rangle \in S, s = \langle a \mid b \rangle(i)\}$ เป็นฐานข้อมูลที่ประกอบด้วยลำดับต้นทางใน S

$S_b = \{\langle SID, b \rangle \mid \forall \langle SID, s \rangle \in S, s = \langle a \mid b \rangle(i)\}$ เป็นฐานข้อมูลที่ประกอบด้วยลำดับปลายทางใน S

และลำดับย่อยผลลัพธ์จาก PrefixSpan มีรายการ $SIDs$ ซึ่งเก็บ SID ของลำดับที่บรรจุลำดับย่อยดังกล่าว

ขั้นตอนวิธีที่ 2 JS-PrefixSpan

ผลลัพธ์ที่ได้จากขั้นตอนวิธี JS-PrefixSpan จะอยู่ในรูปลำดับเชื่อมที่มีข้อมูลเสริมซึ่งสามารถแปลงกลับเป็นลำดับพฤติกรรมต้นทางและลำดับพฤติกรรมปลายทางพร้อมจำนวนลำดับพฤติกรรมในสถานะปลายทาง โดยลำดับพฤติกรรมที่แปลงกลับมาจะไม่สมบูรณ์เช่นเดียวกับลำดับพฤติกรรมที่ได้จากขั้นตอนวิธี PrefixSpan ซึ่งต้องเติมเต็มในขั้นตอนสร้างรูปแบบพฤติกรรมอัตโนมัติเช่นเดียวกัน

สำหรับการทำเหมืองข้อมูลบนข้อมูลลำดับพฤติกรรมเชื่อมในงานวิทยานิพนธ์นี้ จะทำการตรวจข้อมูลลำดับในทุกกลุ่มสถานะ เพื่อหาลำดับต้นทางทั้งหมดที่เป็นไปได้ แล้วจึงหาลำดับปลายทางเพื่อเชื่อมกันเป็นลำดับ

พฤติกรรมเชื่อม โดยลำดับปลายทางอาจได้จากกลุ่มสถานะใด ๆ หรือบล็อกทำลายก็ได้ ขึ้นอยู่กับฟังก์ชั่นสุดท้ายของลำดับต้นทางว่าเป็น Goto หรือ Despawn

ลำดับพฤติกรรมเชื่อมที่สร้างขึ้นจะถูกรวมเป็นฐานข้อมูลลำดับเชื่อม โดยแยกเป็นฐานข้อมูลสำหรับกรณีที่ลำดับต้นทางจบด้วยฟังก์ชั่น Goto และอีกฐานข้อมูลสำหรับกรณีที่จบด้วยฟังก์ชั่น Despawn จากนั้นจึงทำเหมืองข้อมูลด้วยวิธี JS-PrefixSpan โดยใช้ค่าซัพพอร์ทเป็น 2% ของจำนวนข้อมูลขาเข้า

5.4 การทำเหมืองข้อมูลสำหรับฟังก์ชั่นการกระทำคู่ขนาน

การทำเหมืองข้อมูลสำหรับฟังก์ชั่นการกระทำคู่ขนานมีจุดประสงค์เพื่อค้นหาแบบแผนของฟังก์ชั่นที่ทำงานขนานกันที่พบได้บ่อย ฟังก์ชั่นที่ทำงานขนานกันจะไม่สนใจลำดับการทำงานสำหรับฟังก์ชั่นที่มาจากลำดับพฤติกรรมเดียวกัน สิ่งที่ต้องให้ความสนใจมีเพียงฟังก์ชั่นใดทำงานคู่ขนานไปกับฟังก์ชั่นใดที่อยู่ในลำดับพฤติกรรมอื่นบ้าง

เนื่องจากในสถานะหนึ่งสามารถมีลำดับพฤติกรรมที่ทำงานคู่ขนานกันได้หลายลำดับ อีกทั้งแต่ละลำดับพฤติกรรมจะมีฟังก์ชั่นอยู่ได้เป็นจำนวนมาก ทำให้จำนวนพฤติกรรมที่ขนานกันเพิ่มเป็นทวีคูณ ยกตัวอย่างภาคีที่ประกอบด้วยฟังก์ชั่นจำนวนน้อย เช่น สถานะของภาคีหนึ่งมี 3 ลำดับพฤติกรรมได้แก่ AB, DE และ F จะมีพฤติกรรมที่คู่ขนานกันได้ทั้งหมดถึง 24 แบบ (A/D, A/E, B/D, B/E, A/F, B/F, D/F, E/F, A/D/F, A/E/F, B/D/F, B/E/F, AB/D, AB/E, AB/F, A/DE, B/DE, DE/F, AB/D/F, AB/E/F, A/DE/F, B/DE/F, AB/DE, AB/DE/F) ดังนั้น การแจงนับพฤติกรรมคู่ขนานเพื่อสร้างฐานข้อมูลลำดับแล้วใช้ PrefixSpan ในการหาย่อมไม่มีประสิทธิภาพ และยังเกิดข้อผิดพลาดขึ้นได้จากการแจงนับข้อมูลที่ซ้ำซ้อน เช่น ลำดับพฤติกรรม AB, BA ซึ่งแจงนับได้เป็น A/B, A/A, B/B, B/A, AB/B, AB/A, A/BA, B/BA, AB/BA มีตัวอย่างข้อมูลคู่ขนานที่ซ้ำกัน เช่น A/B และ B/A, AB/B และ B/BA (เนื่องจากฟังก์ชั่นการกระทำคู่ขนานไม่สนใจลำดับของฟังก์ชั่นภายในลำดับพฤติกรรมเดียวกัน)

เพื่อป้องกันปัญหาการแจงนับซ้ำซ้อนรวมถึงปัญหาประสิทธิภาพจากการแจงนับ งานวิทยานิพนธ์นี้จึงทำการแปลงลำดับพฤติกรรมที่ขนานกันให้อยู่ในรูปของกราฟระบุชื่อแบบมีทิศทางไม่มีวัฏจักร (Directed acyclic labeled graph) การใช้กราฟมีข้อดีคือไม่ต้องมีการแจงนับพฤติกรรมคู่ขนาน แม้ว่าจะมีข้อเสียที่ต้องทดสอบความสมสัณฐานเพื่อนับค่าซัพพอร์ทซึ่งเป็นปัญหาที่มีความซับซ้อนระดับ NP-Complete แต่ก็มีงานวิจัยทางด้านการทำเหมืองข้อมูลสำหรับข้อมูลแบบกราฟเพื่อลดทอนจำนวนครั้งการทดสอบความสมสัณฐานลง เช่น ขั้นตอนวิธี gSpan [9] ที่งานวิทยานิพนธ์นี้เลือกใช้

5.4.1 การทำเหมืองข้อมูลบนข้อมูลกราฟพฤติกรรมคู่ขนาน

งานวิทยานิพนธ์นี้แปลงลำดับพฤติกรรมที่คู่ขนานกันในสถานะหนึ่ง ๆ เป็นข้อมูลกราฟพฤติกรรมคู่ขนาน โดยมีขั้นตอนการแปลงดังนี้

กำหนดให้สถานะที่สนใจประกอบด้วยลำดับพฤติกรรม $\{s_1, s_2, \ldots, s_n\}$ สามารถแปลงสถานะให้อยู่ในรูปกราฟได้โดยใช้ขั้นตอนต่อไปนี้

1) สร้างจุดยอดรากที่มีป้ายชื่อเป็น d

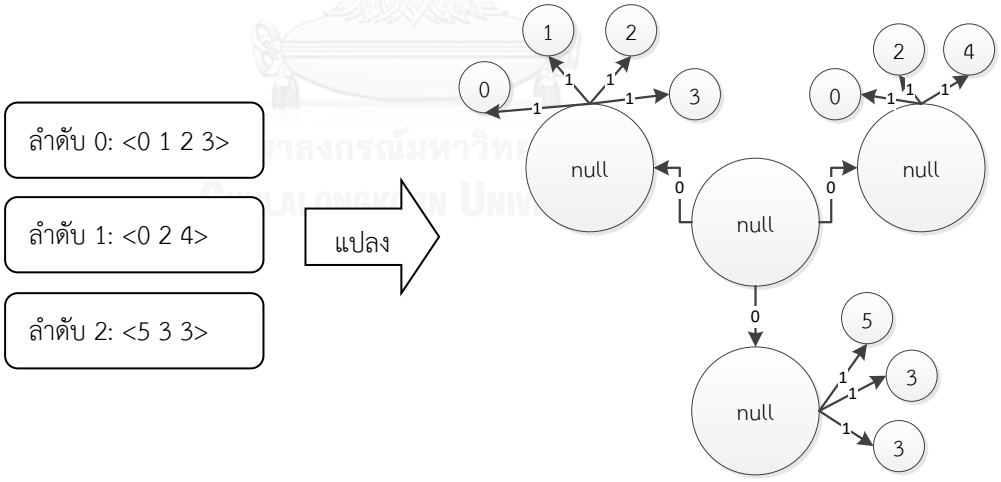
2) สร้างจุดยอดแทนลำดับพฤติกรรมจำนวน n จุด แต่ละจุดมีป้ายชื่อเป็น d และมีเส้นเชื่อมชื่อ 0 จากจุดยอดราก จุดยอดเหล่านี้จะทำหน้าที่แยกฟังก์ชั่นการกระทำที่มาจากต่างลำดับพฤติกรรมออกจากกัน

3) สำหรับแต่ละจุดยอดแทนลำดับพฤติกรรม $s_i', \ 1 \le i \le n$ ให้ทำขั้นตอนย่อยต่อไปนี้เพื่อแปลงฟังก์ชั่นการกระทำเป็นจุดยอดและนำไปเชื่อมต่อกับจุดยอดแทนลำดับพฤติกรรม

> For(Action a in s_i) do:
>
> Convert a to string a_s using method in 5.3.1
>
> Create an action vertex v labeled a_s
>
> Create an edge labeled 1 from s_i' to this v

เมื่อ d เป็นสตริงพิเศษที่ไม่มีการกระทำใดที่แปลงแล้วได้ค่านี้ โดยในที่นี้ใช้สตริงซึ่งแปลงมาจากเลข 0 ซึ่งต่อจากนี้ในภาพตัวอย่างจะใช้ null แสดงแทนสตริงดังกล่าวนี้เนื่องจากสตริงดังกล่าวไม่สามารถแสดงผลอักขระได้ การที่ต้องเลือกใช้สตริงพิเศษสำหรับจุดยอดรากและจุดยอดแทนลำดับพฤติกรรมเพื่อให้สามารถแยกแยะจุดยอดเหล่านี้ออกจากจุดยอดแทนฟังก์ชั่นการกระทำได้ ซึ่งจะมีผลในการคัดกรองผลลัพธ์จากการทำเหมืองข้อมูลเพื่อให้ได้มาซึ่งข้อมูลที่ต้องการ

รูปที่ 18 แสดงตัวอย่างการแปลงลำดับพฤติกรรมที่ขนานกันเป็นข้อมูลกราฟ โดยลำดับพฤติกรรมจะแสดงในรูปข้อมูลลำดับสมมติ



รูปที่ 18 ตัวอย่างการแปลงลำดับพฤติกรรมที่ขนานกันเป็นกราฟ

ข้อมูลพฤติกรรมคู่ขนานในรูปแบบกราฟจะสามารถใช้เป็นข้อมูลขาเข้าของขั้นตอนวิธี gSpan ซึ่งจะให้ผลลัพธ์ออกมาเป็นกราฟย่อยที่พบบ่อย กราฟย่อยที่ได้จะถูกคัดให้เหลือเฉพาะกราฟย่อยที่มีจุดยอดแทนฟังก์ชั่นการกระทำจากอย่างน้อย 2 ลำดับพฤติกรรม เพื่อให้กราฟดังกล่าวมีข้อมูลของพฤติกรรมที่ขนานกันอยู่ กราฟย่อยที่ผ่านการคัดจะถูกแปลงกลับเป็นฟังก์ชั่นการกระทำที่ขนานกันเพื่อนำไปใช้ในขั้นตอนสร้างรูปแบบพฤติกรรมต่อไป

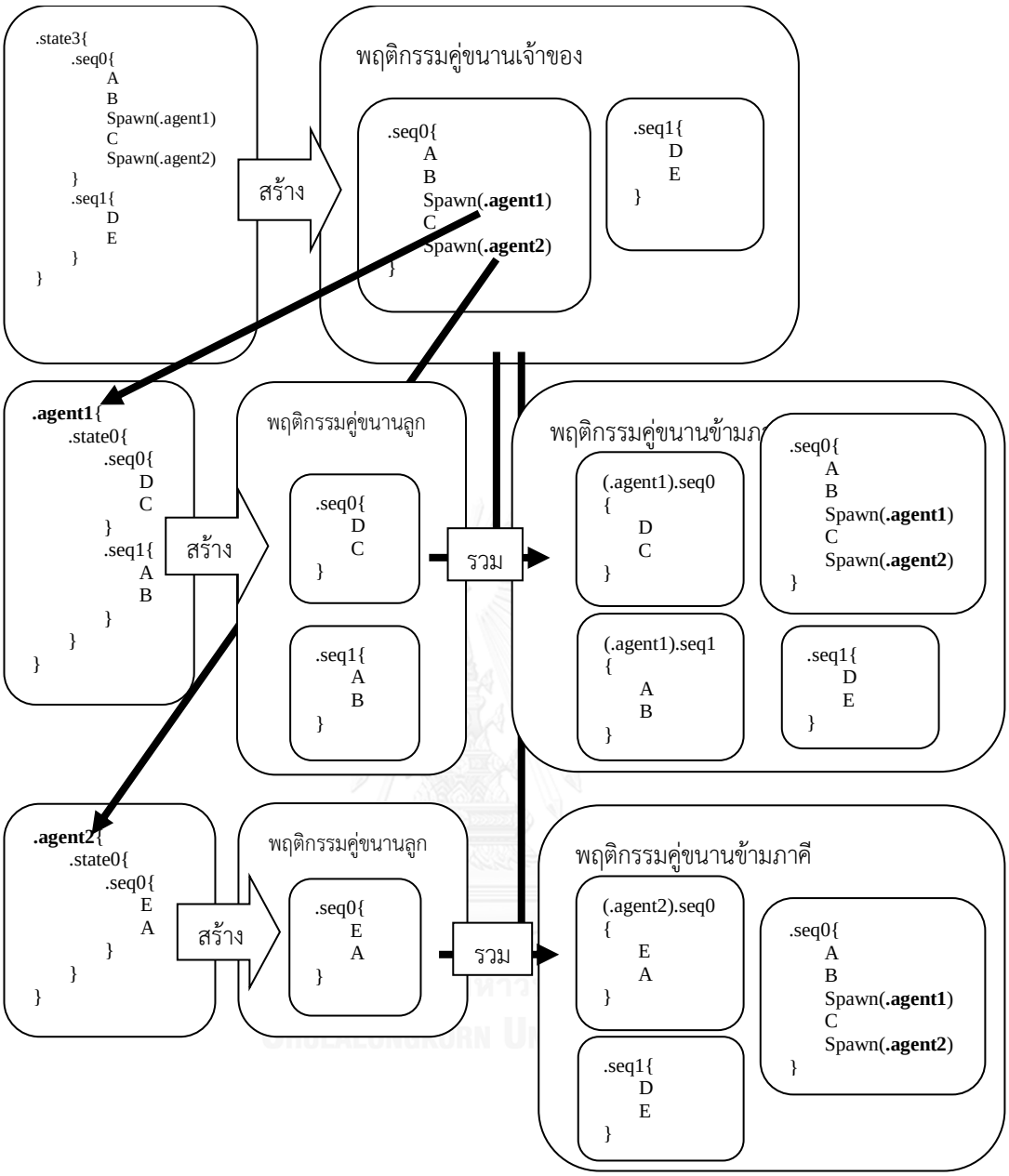
งานวิทยานิพนธ์นี้แปลงทุกสถานะในกลุ่มสถานะให้เป็นกราฟพฤติกรรมคู่ขนานเพื่อสร้างฐานข้อมูลกราฟและทำเหมืองข้อมูลด้วยขั้นตอนวิธี gSpan โดยใช้ค่าซัพพอร์ทเป็น 3%

5.4.2 การทำเหมืองข้อมูลบนข้อมูลกราฟพฤติกรรมคู่ขนานข้ามภาคี

นอกจากพฤติกรรมที่คู่ขนานกันแล้ว หากพิจารณาฟังก์ชั่นการกระทำ Spawn ซึ่งสามารถสร้างภาคีเพิ่มลงในโลกของเกมได้และแสดงพฤติกรรมไปพร้อมกับภาคีเจ้าของ พฤติกรรมลักษณะนี้อาจมองได้ว่าเป็นแบบแผนหนึ่งที่แฝงอยู่ในความสัมพันธ์ของพฤติกรรมคู่ขนานแบบข้ามภาคี ในที่นี้จึงมีการทำเหมืองข้อมูลเพื่อค้นหาความสัมพันธ์ดังกล่าวนี้ด้วยโดยเรียกข้อมูลลักษณะนี้ว่าพฤติกรรมคู่ขนานข้ามภาคี ซึ่งประกอบด้วยลำดับพฤติกรรมทั้งหมดของภาคีเจ้าของเรียกว่า พฤติกรรมคู่ขนานเจ้าของ และลำดับพฤติกรรมของภาคีที่ถูกสร้างเรียกว่า พฤติกรรมคู่ขนานลูก

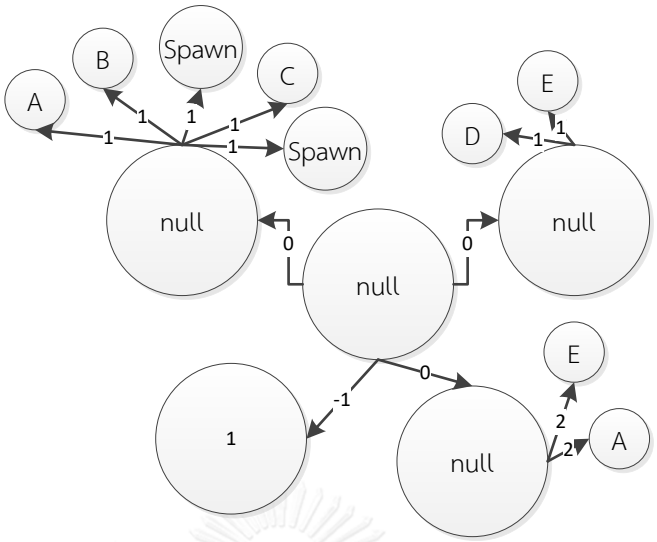
ข้อมูลพฤติกรรมคู่ขนานข้ามภาคีซึ่งเป็นข้อมูลขาเข้าในที่นี้สร้างขึ้นโดยเลือกสถานะของภาคีที่มีลำดับพฤติกรรมซึ่งมีฟังก์ชั่นการกระทำ Spawn และใช้ลำดับพฤติกรรมทั้งหมดในสถานะดังกล่าวเป็นพฤติกรรมคู่ขนานเจ้าของ จากนั้นเลือกสถานะเริ่มต้นของภาคีที่ถูกสร้างด้วยคำสั่ง Spawn และใช้ลำดับพฤติกรรมทั้งหมดในสถานะดังกล่าวเป็นพฤติกรรมคู่ขนานลูก

ในกรณีที่พฤติกรรมคู่ขนานเจ้าของมีการใช้งานฟังก์ชั่น Spawn เพื่อสร้างภาคีมากกว่า 1 ชนิด จะต้องสร้างพฤติกรรมคู่ขนานเจ้าของสำหรับภาคีทุกชนิดเพื่อจับคู่พฤติกรรมคู่ขนานลูกทั้งหมด รูปที่ 19 แสดงตัวอย่างการสร้างพฤติกรรมคู่ขนานข้ามภาคี โดยแสดงในรูปของรหัสเทียม



รูปที่ 19 ตัวอย่างการสร้างพฤติกรรมคู่ขนานข้ามภาคี

เมื่อได้พฤติกรรมคู่ขนานข้ามภาคีแล้ว จึงแปลงให้อยู่ในรูปของกราฟพฤติกรรมคู่ขนานข้ามภาคีโดยใช้วิธีตามหัวข้อ 5.4.1 เพียงแต่ต้องเปลี่ยนวิธีการตั้งชื่อเส้นเชื่อมเพื่อแยกความแตกต่างระหว่างลำดับพฤติกรรมที่มาจากพฤติกรรมคู่ขนานเจ้าของและลำดับพฤติกรรมที่มาจากพฤติกรรมคู่ขนานลูกออกจากกัน โดยให้ตั้งชื่อเส้นเชื่อมที่ออกจากจุดยอดแทนลำดับพฤติกรรมที่มาจากพฤติกรรมคู่ขนานลูกเป็น 2 แทน จากนั้นเพิ่มข้อมูลเสริมสำหรับบอกจำนวนลำดับพฤติกรรมของพฤติกรรมคู่ขนานลูก (ซึ่งอีกนัยหนึ่งก็คือจำนวนลำดับพฤติกรรมของสถานะเริ่มต้นของภาคีที่ถูกสร้าง) โดยสร้างจุดยอดที่มีป้ายชื่อเป็นจำนวนดังกล่าว โดยให้มีเส้นเชื่อมชื่อ -1 เชื่อมจากจุดยอดรากมายังจุดยอดนี้ รูปที่ 20 แสดงตัวอย่างกราฟผลลัพธ์จากการแปลงพฤติกรรมคู่ขนานข้ามภาคีสำหรับกรณี agent2 ในรูปที่ 19 อนึ่งป้ายชื่อสำหรับจุดยอดแทนฟังก์ชั่นจะแสดงโดยใช้ชื่อฟังก์ชั่นที่ปรากฏในรหัสเทียมโดยตรง



รูปที่ 20 ตัวอย่างกราฟผลลัพธ์จากการแปลงพฤติกรรมคู่ขนานข้ามภาคี

ข้อมูลกราฟพฤติกรรมคู่ขนานจะใช้เป็นข้อมูลขาเข้าของขั้นตอนวิธี gSpan กราฟย่อยที่พบบ่อยที่ได้จากขั้นตอนวิธีจะถูกคัด กราฟย่อยที่มีข้อมูลครบถ้วน (มีฟังก์ชั่นการกระทำจากทั้งพฤติกรรมเจ้าของและพฤติกรรมลูกและมีข้อมูลจำนวนลำดับ) เท่านั้นที่จะถูกนำไปใช้งานต่อในการสร้างรูปแบบพฤติกรรมอัตโนมัติ การตรวจสอบความครบถ้วนทำได้โดยตรวจสอบว่ากราฟย่อยที่ได้มีเส้นเชื่อมที่มีป้ายชื่อ 1, 2 และ -1 หรือไม่

งานวิทยานิพนธ์นี้ตรวจสอบทุกสถานะในชุดข้อมูลศัตรูเพื่อสร้างฐานข้อมูลกราฟพฤติกรรมคู่ขนานข้ามภาคี และทำเหมืองข้อมูลด้วยขั้นตอนวิธี gSpan โดยใช้ค่าซัพพอร์ทเป็น 21% สาเหตุที่ไม่สามารถใช้ค่าซัพพอร์ทที่ต่ำกว่านี้ได้เป็นข้อจำกัดของเครื่องมือที่ใช้ในการทดลอง

5.5 การทำเหมืองข้อมูลสำหรับการเลือกใช้ฟังก์ชั่นข้อมูล

ในการอธิบายภาคีด้วยภาษาอธิบายภาคีจะมีการใช้งานฟังก์ชั่นการกระทำเพื่ออธิบายการกระทำและฟังก์ชั่นข้อมูลเพื่อใช้เป็นเงื่อนไขในโครงสร้างเงื่อนไข ซึ่งฟังก์ชั่นเหล่านี้มีพารามิเตอร์อีกจำนวนหนึ่งซึ่งอาจเติมนิพจน์สัญพจน์หรือฟังก์ชั่นข้อมูลที่มีชนิดข้อมูลตรงกันก็ได้ การเติมข้อมูลที่แตกต่างกันจะส่งผลต่อพฤติกรรมผลลัพธ์ของภาคีต่างกันออกไป การทำเหมืองข้อมูลสำหรับการเลือกใช้ฟังก์ชั่นข้อมูลจึงมีจุดประสงค์เพื่อค้นหาแบบแผนของฟังก์ชั่นข้อมูลที่ถูกใช้เป็นอากิวเมนท์สำหรับพารามิเตอร์ต่าง ๆ ของฟังก์ชั่นการกระทำและฟังก์ชั่นข้อมูลหลัก ด้วยจุดประสงค์ดังกล่าว ผลลัพธ์สำหรับการทำเหมืองข้อมูลในกรณีนี้จึงต้องทำให้รู้ได้ว่าแต่ละพารามิเตอร์ของฟังก์ชั่นหลักมีการใช้งานฟังก์ชั่นข้อมูล นิพจน์ หรือสัญพจน์อะไรบ้าง

5.5.1 การทำเหมืองข้อมูลบนข้อมูลการเลือกใช้ฟังก์ชั่นข้อมูล

ในที่นี้เลือกแปลงข้อมูลขาเข้าให้อยู่ในรูปของกราฟระบุชื่อแบบมีทิศทางไม่มีวัฏจักร ข้อมูลกราฟสามารถเก็บข้อมูลทั้งชื่อฟังก์ชั่นหลัก, ฟังก์ชั่นข้อมูลที่เลือกใช้, นิพจน์, สัญพจน์ และตำแหน่งพารามิเตอร์ในฟังก์ชั่นหลักของ

ข้อมูลแต่ละตัว รวมถึงสามารถแสดงการเรียกใช้ฟังก์ชั่นซ้อนฟังก์ชั่นได้โดยไม่มีข้อมูลสูญหายอีกด้วย ทำให้กราฟย่อยจากการทำเหมืองข้อมูลเก็บข้อมูลที่ต้องการได้อย่างครบถ้วน

การแปลงฟังก์ชั่นหลักเป็นกราฟทำได้ตามขั้นตอนดังนี้

กำหนดฟังก์ชั่นที่มีพารามิเตอร์ n ตัว $func(a_0, a_1, ..., a_{n-1})$ สามารถแปลงฟังก์ชั่นดังกล่าวเป็นกราฟโดยใช้ขั้นตอนต่อไปนี้

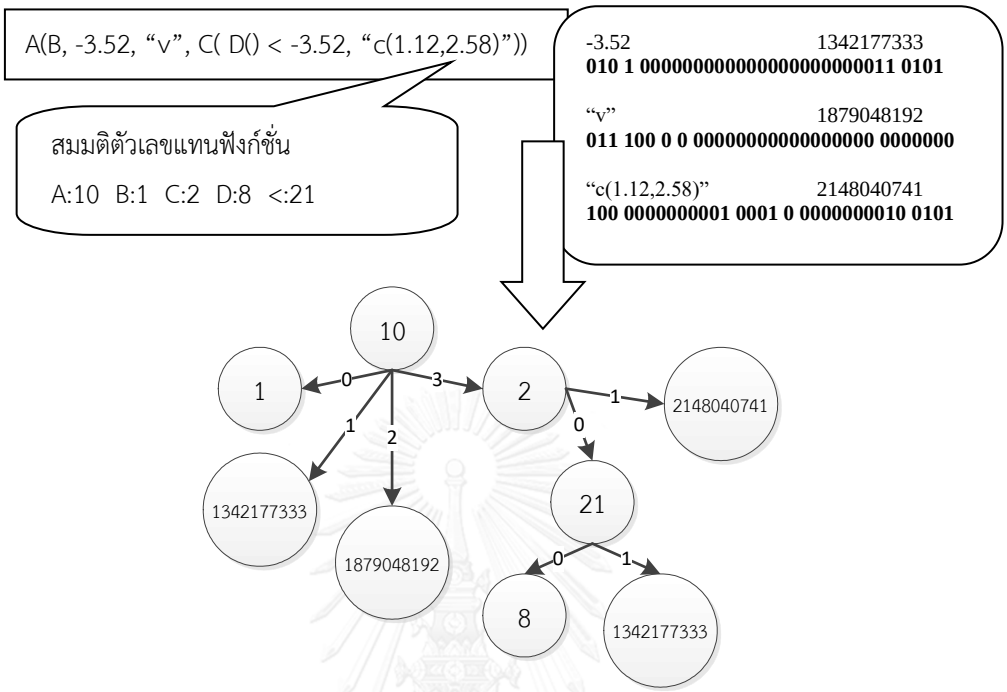
1) สร้างจุดยอด v ที่มีป้ายชื่อเป็นตัวเลขแทนฟังก์ชั่น
2) สำหรับแต่ละพารามิเตอร์ $a_i, \quad 0 \le i \le n-1$ หาก a_i ไม่ใช่สัญพจน์ที่เป็นตัวระบุ ให้สร้างจุดยอด v_i ที่มีป้ายชื่อเป็นตัวเลขแทน a_i และมีเส้นเชื่อมจาก v มายัง v_i ชื่อ i
3) สำหรับแต่ละจุดยอด $v_i, \quad 0 \le i \le n-1$ ที่มีอยู่
   - หากจุดยอดดังกล่าวเป็นจุดยอดแทนฟังก์ชั่น ให้แปลงฟังก์ชั่นเป็นกราฟย่อย
   - หากจุดยอดดังกล่าวเป็นจุดยอดแทนตัวดำเนินการเอกภาค ให้ถือว่าตัวดำเนินการเอกภาคนั้นเป็นฟังก์ชั่นที่มีพารามิเตอร์ 1 ตัว คือ นิพจน์ตามหลัง และใช้วิธีแปลงฟังก์ชั่นเพื่อสร้างกราฟย่อยต่อไป
   - หากจุดยอดดังกล่าวเป็นจุดยอดแทนตัวดำเนินการทวิภาค ให้ถือว่าตัวดำเนินการทวิภาคนั้นเป็นฟังก์ชั่นที่มีพารามิเตอร์ 2 ตัว คือ นิพจน์ซ้าย และนิพจน์ขวา และใช้วิธีแปลงฟังก์ชั่นเพื่อสร้างกราฟย่อยต่อไป

สำหรับวิธีแปลงข้อมูลใด ๆ เป็นตัวเลข จะขึ้นอยู่กับประเภทข้อมูล ดังนี้

ประเภท	วิธีแปลงเป็นตัวเลข (ขนาด 32 บิท)
ฟังก์ชั่นการกระทำ	ใช้ตัวเลขแทนฟังก์ชั่นการกระทำตามที่ระบุในภาคผนวกส่วนรายการฟังก์ชั่นการกระทำ
ฟังก์ชั่นข้อมูล	ใช้ตัวเลขแทนฟังก์ชั่นข้อมูลตามที่ระบุในภาคผนวกส่วนรายการฟังก์ชั่นข้อมูล บวกด้วย 128 (เพื่อไม่ให้ช่วงค่าซ้อนทับกับตัวเลขแทนฟังก์ชั่นการกระทำ) ในกรณีที่มีการใช้ซิงเกิ้ลคิวรี ให้คูณด้วย -1 เพื่อเปลี่ยนเครื่องหมาย
ตัวดำเนินการเอกภาค	ให้ถือว่าเป็นฟังก์ชั่นข้อมูลที่มี 1 พารามิเตอร์ คือ นิพจน์ตามหลัง และใช้ตัวเลขแทนฟังก์ชั่นตามที่ระบุในภาคผนวกส่วนรายการฟังก์ชั่นข้อมูล
ตัวดำเนินการทวิภาค	ให้ถือว่าเป็นฟังก์ชั่นข้อมูลที่มี 2 พารามิเตอร์ คือ นิพจน์ซ้าย และนิพจน์ขวา ตามลำดับ และใช้ตัวเลขแทนฟังก์ชั่นตามที่ระบุในภาคผนวกส่วนรายการฟังก์ชั่นข้อมูล
สัญพจน์บูล	ให้ใช้ตัวเลขดังที่แสดงด้วยเลขฐาน 2 ดังนี้ 000X 1000 0000 0000 0000 0000 0000 0000 โดย - X = 0 เมื่อเป็นค่า false และ X = 1 เมื่อเป็นค่า true
สัญพจน์จำนวนเต็ม	กำหนดสัญพจน์มีค่า x ให้ใช้ตัวเลขดังที่แสดงด้วยเลขฐาน 2 ดังนี้ 001X YYYY YYYY YYYY YYYY YYYY YYYY YYYY โดย - X = $\begin{cases} 0, & \text{sgn}(x) > 0 \\ 1, & \text{sgn}(x) < 0 \end{cases}$ - Y = $\lvert x \rvert$ โดยอนุญาตให้มีค่าไม่เกิน 0xFFFFFFF

<table>
<tr><th>ประเภท</th><th>วิธีแปลงเป็นตัวเลข (ขนาด 32 บิท)</th></tr>
<tr><td>สัญพจน์จำนวนจริง</td><td>กำหนดสัญพจน์มีค่า x ให้ใช้ตัวเลขดังที่แสดงด้วยเลขฐาน 2 ดังนี้
010X YYYY YYYY YYYY YYYY YYYY YYYY ZZZZ โดย
- X = $\begin{cases}0, & \operatorname{sgn}(x)>0\\1, & \operatorname{sgn}(x)<0\end{cases}$
- Y = $\lfloor\lvert x\rvert\rfloor$ โดยอนุญาตให้มีค่าไม่เกิน 0xFFFFFF
- Z = $\lfloor 10(\lvert x\rvert-\lfloor\lvert x\rvert\rfloor)\rfloor$ ซึ่งเป็นทิศนิยมตำแหน่งแรกของ x</td></tr>
<tr><td>สัญพจน์ทิศทาง</td><td>ให้ใช้ตัวเลขดังที่แสดงด้วยเลขฐาน 2 ดังนี้
011X XXYM MMMM MMMM MMMM MMMM MNNN NNNN โดย
- X ค่าขึ้นอยู่กับวิธีการเขียนสัญพจน์
north X = 0 east X = 1 v X = 4
south X = 2 west X = 3 h X = 5
เขียนสัญพจน์ด้วย “ x ” เมื่อ x เป็นจำนวนจริงบวก X = 6
- Y = $\begin{cases}0, & \operatorname{sgn}(x)>0\\1, & \operatorname{sgn}(x)<0\end{cases}$ หรือ 0 กรณีไม่มีค่า x
- M = $\lfloor\lvert x\rvert\rfloor$ โดยอนุญาตให้มีค่าไม่เกิน 0x3FFFF หรือ 0 กรณีไม่มีค่า x
- N = $\lfloor 100(\lvert x\rvert-\lfloor\lvert x\rvert\rfloor)\rfloor$ ซึ่งเป็นทิศนิยม 2 ตำแหน่งแรกของ x หรือ 0 กรณีไม่มีค่า x</td></tr>
<tr><td>สัญพจน์ตำแหน่ง</td><td>ให้ใช้ตัวเลขดังที่แสดงด้วยเลขฐาน 2 ดังนี้
TTTX XXXX XXXX XMMM MSYY YYYY YYYY NNNN โดยการแปลความหมายขึ้นอยู่กับวิธีเขียนสัญพจน์ ดังนี้
<table><tr><th>เขียนแบบคาร์ทีเซียน “c(a,b)”</th><th>เขียนแบบพิกัดเชิงขั้ว “p(a,b)”</th></tr><tr><td>a,b คือ ตำแหน่งแกน x และ y
T = $\begin{cases}4, & \operatorname{sgn}(a)>0\\5, & \operatorname{sgn}(a)<0\end{cases}$</td><td>$a$ คือ รัศมี b คือ มุม
T = 6</td></tr></table>X = $\lfloor\lvert a\rvert\rfloor$ โดยอนุญาตให้มีค่าไม่เกิน 0x3FF
M = $\lfloor 10(\lvert a\rvert-\lfloor\lvert a\rvert\rfloor)\rfloor$ ซึ่งเป็นทิศนิยมตำแหน่งแรกของ a
S = $\begin{cases}0, & \operatorname{sgn}(b)>0\\1, & \operatorname{sgn}(b)<0\end{cases}$
Y = $\lfloor\lvert b\rvert\rfloor$ โดยอนุญาตให้มีค่าไม่เกิน 0x3FF
N = $\lfloor 10(\lvert b\rvert-\lfloor\lvert b\rvert\rfloor)\rfloor$ ซึ่งเป็นทิศนิยมตำแหน่งแรกของ b</td></tr>
<tr><td>สัญพจน์ตัวระบุ</td><td>ไม่แปลงหรือบันทึกข้อมูลนี้ เนื่องจากเป็นข้อมูลที่จำเพาะกับภาคี</td></tr>
</table>

รูปที่ 21 แสดงตัวอย่างกราฟแทนความสัมพันธ์แบบการเลือกใช้ฟังก์ชั่น โดยภาษาอธิบายภาคีที่ปรากฏในตัวอย่างแสดงด้วยรหัสเทียม



รูปที่ 21 ตัวอย่างกราฟแทนความสัมพันธ์แบบการเลือกใช้ฟังก์ชั่นข้อมูล

ข้อมูลกราฟการเลือกใช้ฟังก์ชั่นจะใช้เป็นข้อมูลขาเข้าของขั้นตอนวิธี gSpan กราฟย่อยที่พบบ่อยที่ได้จากขั้นตอนวิธีจะผ่านการคัด โดยคัดกราฟย่อยที่มีจุดยอดเพียงจุดเดียวออก (กราฟดังกล่าวไม่แสดงความสัมพันธ์ใด ๆ) จากนั้นจะเข้าสู่กระบวนการเก็บอากิวเมนท์ที่เป็นไปได้ซึ่งกล่าวถึงในหัวข้อ 5.5.2 ก่อนที่จะถูกนำไปใช้ในการสร้างศัตรูอัตโนมัติ

งานวิทยานิพนธ์นี้ใช้ฟังก์ชั่นการกระทำ นิพจน์แบบบูลที่เป็นเงื่อนไขของโครงสร้างเงื่อนไข และนิพจน์จำนวนนับที่เป็นจำนวนรอบของโครงสร้างวนซ้ำ ที่ปรากฏในชุดข้อมูลศัตรูสร้างเป็นข้อมูลกราฟ และทำเหมืองข้อมูลด้วย gSpan โดยใช้ค่าซัพพอร์ทที่ 1% เพื่อให้ได้ข้อมูลที่หลากหลายที่สุดเท่าที่จะทำได้ อนึ่ง นิพจน์แบบบูลและนิพจน์จำนวนนับจะถูกแปลงให้เป็นนิพจน์แบบเติมหน้าก่อน จากนั้น หากส่วนย่อยแรกสุดของนิพจน์เป็นตัวดำเนินการแทนที่จะเป็นฟังก์ชั่น ให้เปลี่ยนตัวดำเนินการดังกล่าวเป็นฟังก์ชั่น (ตัวดำเนินการเอกภาคเป็นฟังก์ชั่น 1 พารามิเตอร์ ส่วนตัวดำเนินการทวิภาคเป็นฟังก์ชั่น 2 พารามิเตอร์) แล้วจึงแปลงข้อมูลเป็นกราฟโดยใช้วิธีที่ระบุไว้

5.5.2 กระบวนการเก็บอากิวเมนท์ที่เป็นไปได้

กราฟย่อยที่เป็นผลลัพธ์จากการทำเหมืองข้อมูลอาจมีอากิวเมนท์ของฟังก์ชั่นไม่ครบถ้วนได้ เช่น ฟังก์ชั่น A มีพารามิเตอร์ 3 ตัว แต่ผลลัพธ์จากการทำเหมืองข้อมูล อาจได้กราฟย่อยของฟังก์ชั่น A ที่มีข้อมูลเฉพาะอากิวเมนท์ที่ 1 และ 2 ได้ หากจะนำเอาผลลัพธ์นี้ไปใช้ในการสร้างศัตรู ขั้นตอนวิธีการสร้างรูปแบบพฤติกรรมจำเป็นจะต้อง

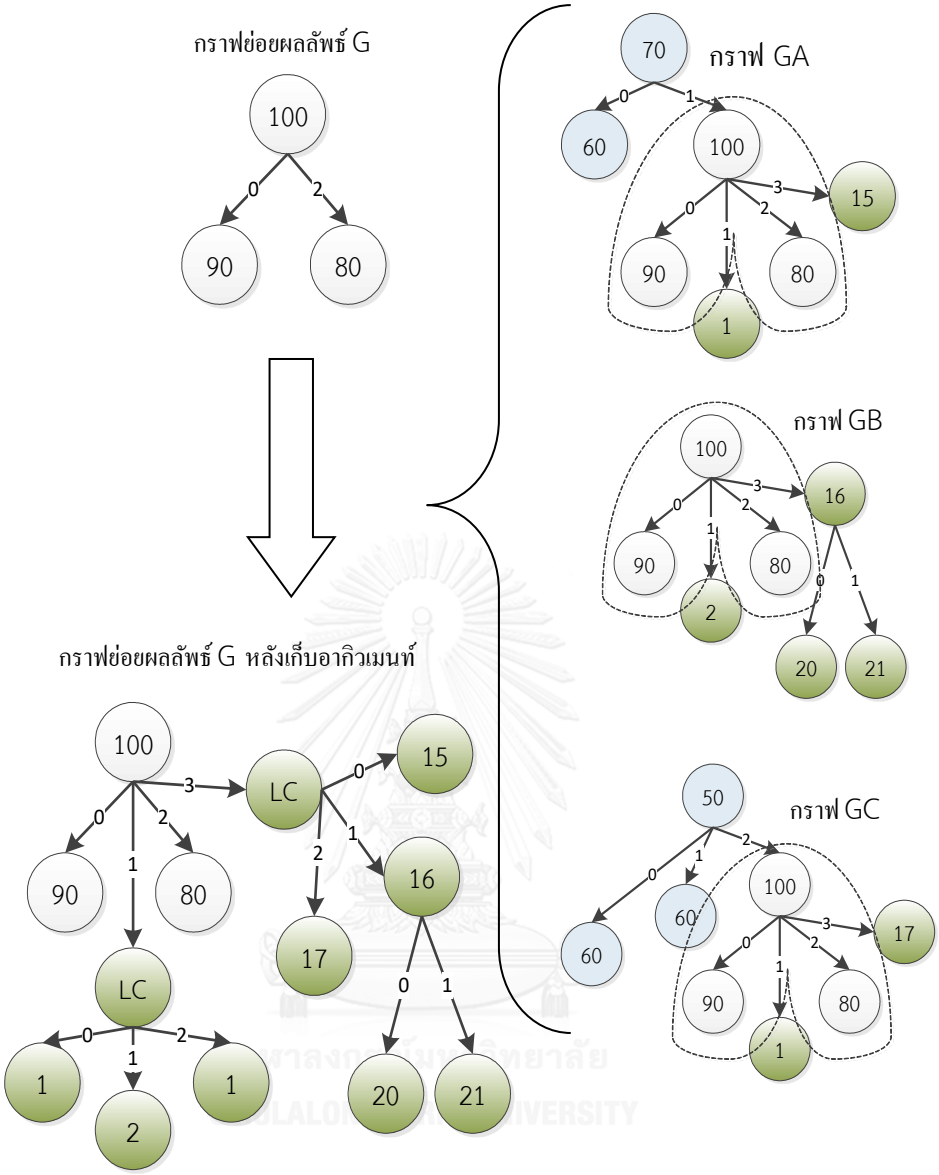
ตัดสินใจหาอากิวเมนท์ที่ 3 มาเติมเพื่อให้ฟังก์ชั่น A สมบูรณ์ จึงเกิดประเด็นปัญหาว่า ข้อมูลที่จะนำมาใช้เป็นอากิวเมนท์จะนำมาจากที่ใด

เนื่องจากแนวคิดของงานวิทยานิพนธ์นี้เป็นการสร้างศัตรูขึ้นมาโดยอิงจากข้อมูลที่มีอยู่แล้ว ในที่นี้จึงตัดสินใจแก้ปัญหาการเลือกอากิวเมนท์อิงกับข้อมูลที่มีอยู่ โดยตรวจสอบว่ากราฟย่อยผลลัพธ์ที่ปรากฏอยู่ในข้อมูลกราฟขาเข้าว่ามีการใช้ค่าใดเป็นอากิวเมนท์บ้างสำหรับอากิวเมนท์ที่ขาดหายไป และเก็บข้อมูลค่าเหล่านั้นไว้โดยสร้างจุดยอดพิเศษเชื่อมกับจุดยอดแทนฟังก์ชั่นสำหรับอากิวเมนท์ที่ขาดไป และใช้จุดยอดพิเศษนี้เชื่อมต่อกับอากิวเมนท์ทุกแบบที่มีการใช้งานในตำแหน่งนั้น ๆ ขั้นตอนทั้งหมดสามารถอธิบายได้ด้วยรหัสเทียมดังนี้

กำหนดให้กราฟย่อยผลลัพธ์ g ได้จากการทำเหมืองข้อมูลโดยมีฐานข้อมูลกราฟ $S_g = \{g_0, g_1, ...\}$ เป็นข้อมูลขาเข้า และกำหนดป้ายชื่อจุดยอดพิเศษ LC สามารถเก็บอากิวเมนท์ที่ขาดหายไปได้โดยใช้ขั้นตอนวิธีต่อไปนี้

```
For(g' in S_g) do:
    Find subgraph g'_sub of g' that is isomorphic to g ignoring vertex labeled LC
    If(g'_sub exists) do:
        For each outgoing edge e' of g'_sub 's root do:
            Find an outgoing edge e of g 's root that is isomorphic to e'
            If(e not exists) do:
                Create a vertex v labeled LC
                Create an edge e labeled the same as e' from g 's root to v
            Else do:
                If(e does not point to vertex labeled LC ) do:
                    Continue
                Let v be the vertex labeled LC pointed by e
            Copy subgraph that e' point to its root as graph g_collected
            Create an edge labeled deg+(v) from v to g_collected 's root
```

จากรหัสเทียมข้างต้น สังเกตได้ว่าการเก็บอากิวเมนท์ที่ขาดไปสำหรับแต่ละกราฟผลลัพธ์จะสำรวจจุดยอดที่ความลึก 1 จากจุดยอดรากของกราฟเท่านั้น โดยไม่มีการสำรวจจุดยอดที่อยู่ในระดับที่ลึกลงไป สาเหตุที่ไม่สำรวจเนื่องจากทุกกราฟย่อยของกราฟผลลัพธ์จะเป็นกราฟผลลัพธ์ด้วย ซึ่งจุดยอดที่อยู่ลึกลงไปจะถูกสำรวจเมื่อมีการสำรวจผลลัพธ์ที่เป็นกราฟย่อยดังกล่าว รูปที่ 22 แสดงตัวอย่างการเก็บอากิวเมนท์จากกราฟ GA, GB, และ GC ตามลำดับ สำหรับกราฟย่อยผลลัพธ์ G



รูปที่ 22 ตัวอย่างการเก็บอาก็วเมนท์

บทที่ 6
การสร้างรูปแบบพฤติกรรมโดยอัตโนมัติ

ขั้นตอนการทำเหมืองข้อมูลจะทำให้ได้ข้อมูลความสัมพันธ์รูปแบบต่าง ๆ ที่พบได้บ่อยในชุดข้อมูลศัตรู ความสัมพันธ์ที่ได้จะถูกเลือกมาใช้งานอย่างสุ่มโดยอิงกับสถิติการใช้ที่มีอยู่ ความสัมพันธ์ที่ถูกเลือกจะถูกแปลงกลับให้อยู่ในรูปของฟังก์ชั่นหรือข้อมูลอื่น ๆ แล้วเติมใส่โครงภาคี ซึ่งเป็นภาคีที่อยู่ระหว่างการสร้าง เมื่อความสัมพันธ์ที่เลือกมาถูกรวมเข้ากับโครงภาคีทั้งหมดแล้ว ขั้นตอนตกแต่งภาคีจะทำการเติมส่วนที่ยังคงขาดหายไปในโครงภาคีให้ครบถ้วนสมบูรณ์และตัดส่วนที่ไม่จำเป็นออกไปเพื่อให้กลายเป็นสคริปต์สำหรับศัตรูที่สามารถนำไปใช้งานได้

6.1 คำศัพท์ที่สำคัญ

การอธิบายวิธีสร้างศัตรูโดยอัตโนมัติจะใช้คำศัพท์ที่กำหนดขึ้นมาสำหรับอ้างอิงถึงวัตถุหรือองค์ประกอบที่เกี่ยวข้องเพื่อให้คำอธิบายมีความกระชับ หัวข้อนี้จะกล่าวถึงคำศัพท์ที่สำคัญต่าง ๆ

ข้อความสั่ง (Statement)

คือ คำที่ใช้เรียกรวมสิ่งที่สามารถใส่ลงในบล็อกลำดับได้ ซึ่งประกอบด้วย ฟังก์ชั่นการกระทำ โครงสร้างเงื่อนไข และโครงสร้างวนซ้ำ

โครงภาคี (Skeleton)

คือ สคริปต์ภาษาอธิบายภาคีสำหรับอธิบายภาคีหนึ่งที่ยังไม่สมบูรณ์ ความไม่สมบูรณ์อาจเกิดจากฟังก์ชั่นยังมีอากิวเมนท์ไม่ครบถ้วน หรือยังไม่มีการกำหนดค่าคุณสมบัติให้ตัวภาคีก็ได้ ส่วนที่ขาดหายไปในสคริปต์และจำเป็นต้องถูกเติมให้เต็ม เรียกว่า สล็อตจำเป็น (แทนด้วยสัญลักษณ์ [CRIT]) นอกเหนือจากสล็อตจำเป็น โครงภาคียังมีช่องว่างอื่น ๆ ที่อนุญาตให้เติมสคริปต์ลงไปได้อีก เช่น ช่องว่างก่อนฟังก์ชั่นแรกสุดของลำดับพฤติกรรมหรือช่องว่างระหว่างฟังก์ชั่นการกระทำ ซึ่งแทรกฟังก์ชั่นการกระทำใหม่ลงไปได้ หรือแม้กระทั่งภายในสถานะหนึ่ง ก็มีช่องว่างให้เพิ่มบล็อกลำดับใหม่ เป็นต้น ช่องว่างเหล่านี้ไม่ใช่ช่องว่างที่จำเป็นต้องเติมให้เต็ม แต่สามารถเพิ่มรหัสลงไปได้ ในที่นี้เรียกช่องว่างเหล่านี้ว่า สล็อตทางเลือก (แทนด้วยสัญลักษณ์ [OPT])

นอกจากนี้ โครงภาคีไม่จำเป็นจะต้องมีบล็อกภาคีเพียงบล็อกเดียวเสมอไป ภาคีศัตรูหนึ่งตัวสามารถมีภาคีที่เกี่ยวข้องอื่น ๆ ได้อีก เช่น ศัตรูที่สามารถยิงกระสุนได้ จะมีภาคีกระสุนมาเกี่ยวข้อง เป็นต้น ทำให้มีสล็อตทางเลือกสำหรับเพิ่มภาคีใหม่ด้วย

สล็อตจำเป็นที่ปรากฏในโครงภาคีจะมีการจำกัดประเภทของข้อมูลที่นำมาใส่ได้อยู่ โดยขึ้นอยู่กับบริเวณที่สล็อตดังกล่าวปรากฏอยู่ โดยปกติแล้วสล็อตจำเป็นจะปรากฏเป็นอากิวเมนท์ของฟังก์ชั่น ประเภทข้อมูลที่จะใส่ลงในสล็อตจำเป็นช่องดังกล่าวได้จึงต้องเป็นข้อมูลประเภทที่พารามิเตอร์ของฟังก์ชั่นต้องการ สล็อตจำเป็นอาจปรากฏในบริเวณอื่น ๆ ได้ ได้แก่ เงื่อนไขของโครงสร้างเงื่อนไข และจำนวนรอบทำซ้ำของโครงสร้างวนซ้ำ โดยในกรณีแรกสล็อตจำเป็นจะเป็นประเภทบูล [CRIT-Boolean] ส่วนกรณีหลังจะเป็นประเภทจำนวนนับ [CRIT-Int]

สล็อตตัวเลือกมีการจำกัดประเภทของข้อมูลขึ้นกับบริเวณที่สล็อตปรากฏเช่นกัน ดังนี้

- สล็อตตัวเลือกประเภทภาคี [OPT-Agent] จะอยู่ที่ต้นโครงภาคีในกรณีที่โครงภาคีไม่มีบล็อกภาคีอยู่หรือท้ายบล็อกภาคีสุดท้ายที่มีอยู่ เป็นสล็อตที่สามารถเพิ่มบล็อกภาคีใหม่ลงไปได้
- สล็อตตัวเลือกประเภทสถานะ [OPT-State] จะอยู่ก่อนเครื่องหมาย "}" ของทุกบล็อกภาคี เป็นสล็อตที่สามารถเพิ่มบล็อกสถานะใหม่ลงไปได้ จะไม่สามารถเพิ่มบล็อกเริ่มต้นหรือบล็อกทำลายได้ หากบล็อกภาคีที่สล็อตอยู่มีบล็อกดังกล่าวอยู่แล้ว
- สล็อตตัวเลือกประเภทลำดับ [OPT-Seq] จะอยู่ก่อนเครื่องหมาย "}" ของทุกบล็อกสถานะ เป็นสล็อตที่สามารถเพิ่มบล็อกลำดับใหม่ลงไปได้
- สล็อตตัวเลือกประเภทข้อความสั่ง [OPT-Stmt] จะอยู่ภายในบรรทัดใดก็ได้ภายในบล็อกลำดับ เป็นสล็อตที่สามารถเพิ่มข้อความสั่งลงไปได้

เมื่อโครงภาคีมีการเปลี่ยนแปลงเกิดขึ้น สล็อตตัวเลือกจะต้องถูกสร้างจนครบทุกบริเวณที่สล็อตจะปรากฏได้เพื่อเตรียมรองรับข้อความสั่งที่จะถูกเติมระหว่างการสร้างภาคี ตัวอย่างในรูปที่ *23* แสดงโครงภาคีและสล็อตแบบต่างๆ

```
.Sample{
    .init{
        [OPT-Stmt]
        Set("texture", DynamicFilter("this"), 4;
        [OPT-Stmt]
    }
    .state0{
        .seq0{
            [OPT-Stmt]
            RunStraight(Get("direction", DynamicFilter("this")), [CRIT-Float], [CRIT-Boolean];
            [OPT-Stmt]
            Spawn(.Bullet)
            [OPT-Stmt]
        }
        [OPT–Seq]
    }
    [OPT–State]
}
.Bullet{
    .init{
        [OPT-Stmt]
        Set("texture", DynamicFilter("this"), 3;
        [OPT-Stmt]
        Set("projectile", DynamicFilter("this"),  true);
        [OPT-Stmt]
    }
    .state0{
        .seq0{
            [OPT-Stmt]
            Jump(Get("position", DynamicFilter("this")), [CRIT-Float], [CRIT-Int], false);
            [OPT-Stmt]
        }
        [OPT–Seq]
    }
    [OPT–State]
}
[OPT–Agent]
```

รูปที่ 23 ตัวอย่างโครงภาคีและสล็อต

ฟังก์ชั่นเปลี่ยนสถานะ (Transition)

คือ ฟังก์ชั่น Goto หรือฟังก์ชั่น Despawn ซึ่งเป็นฟังก์ชั่นที่สามารถเปลี่ยนสถานะของภาคีได้ โดย Goto เปลี่ยนให้ภาคีไปทำงานในบล็อกสถานะที่เป็นอากิวเมนท์ของฟังก์ชั่น ส่วน Despawn จะนำเอาภาคีออกจากโลกของเกม และทำให้พฤติกรรมในบล็อกทำลาย (ถ้ามี) ทำงาน

ความเข้ากันของฟังก์ชั่น

- ฟังก์ชั่นเปลี่ยนสถานะ 2 ฟังก์ชั่นจะเข้ากัน ก็ต่อเมื่อเงื่อนไขใดเงื่อนไขหนึ่งต่อไปนี้เป็นจริง
    a) เป็นฟังก์ชั่น Despawn ทั้งคู่
    b) เป็นฟังก์ชั่น Goto ทั้งคู่โดยมีอากิวเมนท์เป็นตัวระบุที่มีค่าเดียวกัน
    c) เป็นฟังก์ชั่น Goto ทั้งคู่โดยมีอากิวเมนท์ของฟังก์ชั่นหนึ่งหรือทั้งคู่เป็นสล็อตจำเป็น
- ฟังก์ชั่น Spawn 2 ฟังก์ชั่นจะเข้ากัน ก็ต่อเมื่อเงื่อนไขต่อไปนี้เป็นจริงทั้งหมด
    a) ทั้งคู่มีอากิวเมนท์เป็นตัวระบุที่มีค่าเดียวกัน หรืออากิวเมนท์ของฟังก์ชั่นหนึ่งหรือทั้งคู่เป็นสล็อตจำเป็น
    b) ทั้งคู่มีจำนวนชั้นของโครงสร้างที่ครอบอยู่เท่ากัน และทุกโครงสร้างที่ชั้นเดียวกันจะต้องเหมือนกัน โดยในที่นี้นิยามความเหมือนของโครงสร้างที่ครอบฟังก์ชั่น Spawn ได้ดังนี้
        - กรณีโครงสร้างเงื่อนไข จะเหมือนกันก็ต่อเมื่อ โครงสร้างเหมือนกัน (มีบล็อก Else ทั้งคู่ หรือไม่มีทั้งคู่) และฟังก์ชั่น Spawn ต้องอยู่ภายใต้บล็อกชนิดเดียวกัน (อยู่ภายใต้บล็อก If ทั้งคู่ หรืออยู่ภายใต้บล็อก Else ทั้งคู่) และมีนิพจน์เงื่อนไขเหมือนกัน
        - กรณีโครงสร้างวนซ้ำ จะเหมือนกันก็ต่อเมื่อ มีนิพจน์กำหนดจำนวนครั้งเหมือนกัน
        - นิพจน์ จะเหมือนกันก็ต่อเมื่อ เมื่อทำให้นิพจน์อยู่ในรูปต้นไม้นิพจน์แล้ว ปมที่ความลึก 0 และความลึก 1 จะต้องเหมือนกันทั้งหมด สาเหตุที่ไม่ตรวจสอบในระดับความลึกที่มากกว่านี้ เนื่องจากจะทำให้โอกาสที่จะพบนิพจน์เหมือนกันมีน้อยมาก

บล็อกที่สามารถรองรับฟังก์ชั่นเปลี่ยนสถานะ T ได้

คือ บล็อกลำดับที่มีสล็อตตัวเลือกที่เหมาะสมกับการเติมฟังก์ชั่นเปลี่ยนสถานะ โดยแบ่งเป็น 2 ชนิด ได้แก่ บล็อกที่รองรับ T ที่ท้ายลำดับได้ และบล็อกที่รองรับ T ที่ท้ายบล็อกได้

- บล็อกที่สามารถรองรับ T ที่ท้ายลำดับได้ (End-of-sequence T-acceptable block – EOS-T block) คือ บล็อกลำดับที่เป็นไปตามเงื่อนไขข้อใดข้อหนึ่งนี้
    a) ข้อความสั่งสุดท้ายของบล็อกลำดับนี้ไม่ใช่ฟังก์ชั่นเปลี่ยนสถานะ
    b) ข้อความสั่งสุดท้ายของบล็อกลำดับนี้เป็นฟังก์ชั่นเปลี่ยนสถานะที่เข้ากับ T
- บล็อกที่สามารถรองรับ T ที่ท้ายบล็อกได้ (End-of-block T-acceptable block – EOB-T block) คือ บล็อกลำดับที่เป็นไปตามเงื่อนไขข้อใดข้อหนึ่งนี้
    c) สามารถรองรับ T ที่ท้ายลำดับได้

d) ภายในบล็อก มีบล็อกอย่างน้อย 1 บล็อกที่ข้อความสั่งสุดท้ายของบล็อกไม่ใช่ฟังก์ชั่นเปลี่ยนสถานะ

e) ภายในบล็อก มีบล็อกอย่างน้อย 1 บล็อกที่ข้อความสั่งสุดท้ายของบล็อกเป็นฟังก์ชั่นเปลี่ยนสถานะที่เข้ากับ T

6.2 ขั้นตอนการสร้างศัตรูโดยอัตโนมัติ

การสร้างศัตรูอัตโนมัติเริ่มจากการกำหนดตัวเลขต่าง ๆ สำหรับการสร้างจากค่าสถิติที่ตรวจสอบได้จากชุดข้อมูลศัตรูและผลลัพธ์จากการทำเหมืองข้อมูล จากนั้นสร้างโครงภาคีพร้อมกำหนดค่าคุณสมบัติโดยใช้แม่แบบ และสุ่มความสัมพันธ์ต่าง ๆ ตามตัวเลขที่กำหนดมารวมเข้ากับโครงภาคี แล้วจึงใช้ความสัมพันธ์รูปแบบการเลือกใช้ฟังก์ชั่นข้อมูลเพื่อเติมอากิวเมนท์ที่ขาดหายไป สุดท้ายจึงทำการตกแต่งภาคีโดยนำเอาบล็อกที่ไม่มีการใช้งานออก กำหนดค่าให้สล็อตจำเป็นประเภทตัวระบุ และกำหนดค่าคุณสมบัติให้กับภาคีที่ยังไม่มีค่าคุณสมบัติเริ่มต้นโดยใช้แม่แบบ หากสิ้นสุดกระบวนการทั้งหมดนี้แล้ว ภาคีที่ได้ออกมาเป็นภาคีที่ไม่มีพฤติกรรม ให้ยกเลิกการสร้างครั้งนี้และเริ่มสร้างภาคีตัวใหม่ ในกรณีที่ภาคีดังกล่าวมีพฤติกรรม ให้ใช้ฮิวริสติกในการปรับปรุงผลลัพธ์ก่อนจะจบกระบวนการสร้าง

ขั้นตอนการสร้างโดยละเอียดเป็นดังนี้

1) ตรวจสอบค่าสถิติการใช้งานความสัมพันธ์รูปแบบต่าง ๆ กับชุดข้อมูลศัตรูและจัดทำเป็นตารางข้อมูลเพื่อดูว่า ศัตรูแต่ละตัวที่อยู่ในชุดข้อมูลมีการใช้งานรูปแบบความสัมพันธ์ที่ได้จากการทำเหมืองข้อมูลแต่ละประเภทเป็นจำนวนเท่าใด ตารางที่ 4 แสดงตัวอย่างค่าสถิติของศัตรู Ice man จากเกม Megaman

ตารางที่ 4 ค่าสถิติของศัตรู Ice man จากเกม Megaman

Ice man			
#ลำดับฟังก์ชั่นการกระทำ	24	#ลำดับฟังก์ชั่นข้ามสถานะ (Goto)	6
#ฟังก์ชั่นการกระทำคู่ขนาน	3	#ลำดับฟังก์ชั่นข้ามสถานะ (Despawn)	0
#ฟังก์ชั่นการกระทำคู่ขนานข้ามภาคี	12	#การเลือกใช้ฟังก์ชั่นข้อมูล	143

2) กำหนดตัวเลขสำหรับการสร้างซึ่งประกอบด้วยจำนวนของความสัมพันธ์แบบต่าง ๆ ที่ภาคีศัตรูตัวที่กำลังจะใช้สร้างต้องมี (ยกเว้นการเลือกใช้ฟังก์ชั่นข้อมูล) การกำหนดตัวเลขสำหรับความสัมพันธ์แต่ละแบบจะทำโดยการสุ่มเลือกศัตรูจากชุดข้อมูลศัตรู 1 ตัว และนำเอาจำนวนความสัมพันธ์ที่สนใจที่ปรากฏในค่าสถิติของศัตรูนั้นมาใช้ในการกำหนดตัวเลข
3) สร้างโครงภาคีซึ่งประกอบด้วยภาคี 1 ตัวอันเป็นภาคีหลัก และสร้างบล็อกทำลายให้ภาคีหลักนี้
4) กำหนดค่าคุณสมบัติโดยสร้างบล็อกเริ่มต้นให้กับภาคีหลักที่ประกอบด้วยฟังก์ชั่น Set เพื่อใช้ตั้งค่าคุณสมบัติตามแม่แบบดังนี้

ตำแหน่ง	320,240	พลังชีวิต	100
ทิศทาง	หันซ้าย	พลังโจมตี	10
ขนาดตัวตรวจจับการชน	32 x 48	ฝ่ายโจมตี	ใช่
ผลของแรงดึงดูด	1	ฝ่ายตั้งรับ	ใช่

5) สุ่มเลือก 1 ความสัมพันธ์รูปแบบฟังก์ชั่นการกระทำคู่ขนานข้ามภาคีจากผลการทำเหมืองข้อมูล และเติมลงในโครงภาคี ทำซ้ำด้วยจำนวนครั้งตามตัวเลขที่ได้จากขั้นตอนที่ 2 รายละเอียดของวิธีการเติมจะกล่าวถึงในหัวข้อ 6.13 การเติมความสัมพันธ์ประเภทนี้ก่อนจะช่วยเพิ่มจำนวนภาคีในโครงภาคีได้ ซึ่งทำให้ความสัมพันธ์อื่น ๆ ที่จะเติมในขั้นตอนถัดไปมีทางเลือกมากขึ้น
6) สุ่มเลือก 1 ความสัมพันธ์รูปแบบฟังก์ชั่นการกระทำคู่ขนานจากผลการทำเหมืองข้อมูล และเติมลงในโครงภาคี ทำซ้ำด้วยจำนวนครั้งตามตัวเลขที่ได้จากขั้นตอนที่ 2 รายละเอียดของวิธีการเติมจะกล่าวถึงในหัวข้อ 6.12
7) สุ่มเลือก 1 ความสัมพันธ์รูปแบบฟังก์ชั่นการกระทำข้ามสถานะกรณีฟังก์ชั่นข้ามสถานะเป็น Goto และเติมลงในโครงภาคี ทำซ้ำด้วยจำนวนครั้งตามตัวเลขที่ได้จากขั้นตอนที่ 2 รายละเอียดของวิธีการเติมจะกล่าวถึงในหัวข้อ 6.10 ขั้นตอนนี้มีโอกาสช่วยเพิ่มจำนวนสถานะของภาคีได้
8) สุ่มเลือก 1 ความสัมพันธ์รูปแบบฟังก์ชั่นการกระทำข้ามสถานะกรณีฟังก์ชั่นข้ามสถานะเป็น Despawn และเติมลงในโครงภาคี ทำซ้ำด้วยจำนวนครั้งตามตัวเลขที่ได้จากขั้นตอนที่ 2 รายละเอียดของวิธีการเติมจะกล่าวถึงในหัวข้อ 6.11 ขั้นตอนนี้มีโอกาสช่วยเพิ่มจำนวนสถานะของภาคีได้
9) สุ่มเลือก 1 ความสัมพันธ์รูปแบบลำดับฟังก์ชั่นการกระทำ และเติมลงในโครงภาคี ทำซ้ำด้วยจำนวนครั้งตามตัวเลขที่ได้จากขั้นตอนที่ 2 รายละเอียดของวิธีการเติมจะกล่าวถึงในหัวข้อ 6.9
10) เติมอากิวเมนท์ของฟังก์ชั่นที่ขาดหายไปโดยใช้ความสัมพันธ์รูปแบบการเลือกใช้ฟังก์ชั่นข้อมูลทั้งหมดที่มี รายละเอียดวิธีการเติมจะกล่าวถึงในหัวข้อ 6.14
11) ลบบล็อกสถานะและบล็อกลำดับพฤติกรรมที่ว่างเปล่าออกจากทุกภาคี
12) สำหรับทุกบล็อกภาคีที่ไม่มีบล็อกสถานะ ให้เพิ่มบล็อกสถานะว่างลงไป
13) กำหนดตัวระบุค่าให้กับฟังก์ชั่น Goto และ Spawn ที่ยังไม่ได้ระบุสถานะปลายทางและภาคีเป้าหมาย โดยพยายามทำให้ฟังก์ชั่น Goto อ้างอิงถึงทุกบล็อกสถานะเท่าที่ทำได้ วิธีกำหนดตัวระบุค่าจะกล่าวถึงในหัวข้อ 6.15
14) สำหรับทุกภาคีที่ยังไม่มีค่าคุณสมบัติเริ่มต้น ให้สร้างบล็อกเริ่มต้นที่ประกอบด้วยฟังก์ชั่น Set เพื่อใช้ตั้งค่าคุณสมบัติตามแม่แบบดังนี้

ขนาดตัวตรวจจับการชน	16 x 16	พลังโจมตี	1
ภาคีกระสุน	ใช่	ฝ่ายโจมตี	ใช่

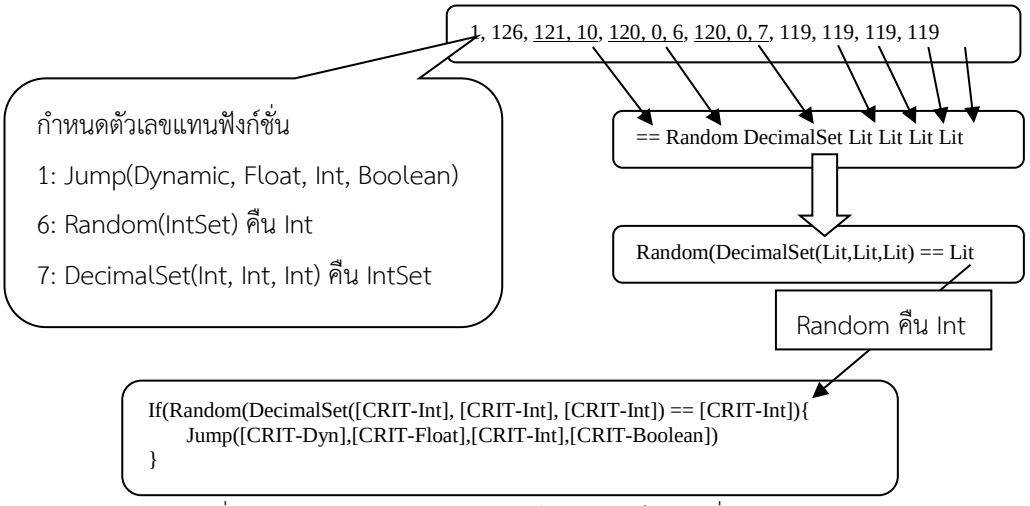
15) ในกรณีที่ภาคีที่สร้างขึ้นไม่มีพฤติกรรม ให้ยกเลิกการสร้างภาคี โดยถือว่าภาคีที่มีลักษณะดังนี้ไม่สามารถยอมรับได้ และเริ่มสร้างภาคีใหม่ตั้งแต่ขั้นตอนแรก
16) ปรับปรุงภาคีด้วยฮิวริสติกเพื่อให้ภาคีที่สร้างมีโอกาสเป็นภาคีที่ยอมรับได้สูงขึ้น โดยใช้วิธีตามหัวข้อ 6.16

ภาคีที่สร้างออกมาจะอยู่ในรูปของไฟล์สคริปต์ที่สามารถอ่านและทำความเข้าใจได้ด้วยตัวแปลภาษาที่จัดทำขึ้นตามข้อกำหนดที่ระบุไว้ในบทที่ 4

6.3 การแปลงสตริงในลำดับเป็นข้อความสั่ง

ข้อมูลผลลัพธ์จากการทำเหมืองข้อมูลสำหรับข้อมูลลำดับจะอยู่ในรูปของลำดับของสตริง จึงต้องแปลงให้อยู่ในรูปของภาษาอธิบายภาคีก่อนเพื่อให้เหมาะสมต่อการนำไปใช้สร้างภาคี ลำดับของสตริงอันเป็นผลลัพธ์ของการทำเหมืองข้อมูลจะประกอบด้วยไอเท็มเซ็ตที่เรียงต่อกันเป็นลำดับ โดยแต่ละไอเท็มเซ็ตจะประกอบด้วยไอเท็ม 1 ตัวที่เป็นสตริง ซึ่งเป็นตัวแทนของ 1 ฟังก์ชั่นการกระทำพร้อมโครงสร้างที่ครอบไว้ (ถ้ามี) ข้อมูลสตริงนี้สามารถแปลงกลับเป็นอาเรย์ของไบต์ได้โดยการถอดรหัสแบบ ASCII และอาเรย์ของไบต์ดังกล่าวสามารถแปลงกลับเป็นข้อความสั่งได้ ข้อความสั่งที่ได้อาจเป็นฟังก์ชั่นการกระทำ หรือฟังก์ชั่นการกระทำที่มีโครงภาคีที่ครอบไว้ก็ได้ วิธีการแปลงสามารถอิงตามรายละเอียดที่ระบุในหัวข้อ 5.3.1 โดยมีจุดที่แตกต่างออกไปเล็กน้อย เนื่องจากในช่วงที่แปลงข้อมูลฟังก์ชั่นการกระทำเป็นข้อมูลลำดับนั้น มีการละข้อมูลบางอย่างทิ้งไป ได้แก่ อากิวเมนท์ของฟังก์ชั่น และค่าของสัญพจน์ ดังนั้น เมื่อสตริงถูกแปลงกลับมาเป็นฟังก์ชั่นอีกครั้ง จะทำให้ไม่มีข้อมูลเหล่านี้ซึ่งเป็นข้อมูลสำคัญ ในที่นี้จึงต้องนำเอาสล็อตจำเป็นไปแทนค่าเหล่านี้ โดยประเภทของสล็อตให้เทียบกับชนิดข้อมูลของพารามิเตอร์นั้น ๆ ที่สล็อตเข้าไปแทนที่ รูปที่ 24 แสดงตัวอย่างการแปลงอาเรย์ของไบต์กลับมาเป็นฟังก์ชั่นการกระทำ

เมื่อทำการแปลงสตริงแต่ละตัวในลำดับออกมาเป็นข้อความสั่งด้วยวิธีข้างต้น ผลลัพธ์สุดท้ายจึงได้เป็นลำดับข้อความสั่ง



รูปที่ 24 ตัวอย่างการแปลงอาเรย์ของไบต์กลับเป็นฟังก์ชั่นการกระทำ

6.4 การเติมลำดับข้อความสั่งลงในโครงภาคี

ข้อมูลความสัมพันธ์แต่ละแบบที่ได้จากการทำเหมืองข้อมูล (นอกเหนือจากความสัมพันธ์รูปแบบการใช้ฟังก์ชั่นข้อมูล) เป็นความสัมพันธ์ระหว่างฟังก์ชั่นการกระทำทั้งสิ้น โดยฟังก์ชั่นการกระทำนั้นอาจมีโครงสร้างเงื่อนไขหรือโครงสร้างวนซ้ำครอบอยู่ก็ได้ การเติมความสัมพันธ์ลงในโครงภาคีทำได้โดยแปลงข้อมูลความสัมพันธ์นั้นกลับเป็นลำดับของข้อความสั่ง (ซึ่งในที่นี้ก็คือฟังก์ชั่นการกระทำที่อาจมีโครงสร้างครอบอยู่) แล้วจึงเติมลำดับดังกล่าวลงในบล็อกที่กำหนด

จุดที่แตกต่างกันในการเติมความสัมพันธ์แต่ละชนิดจะอยู่ที่การแปลงข้อมูลความสัมพันธ์กลับมาเป็นลำดับข้อความสั่ง และวิธีการเลือกบล็อกลำดับที่จะเติมลำดับข้อความสั่งลงไป ซึ่งต้องมีการตรวจสอบและคัดเลือกเฉพาะบล็อกที่เหมาะสมกับความสัมพันธ์นั้น ๆ

จุดที่เหมือนกันในการเติมความสัมพันธ์ คือ วิธีการเติมข้อความสั่งลงในบล็อกลำดับที่เลือกมาแล้ว ซึ่งใช้วิธีการสุ่มเลือกสล็อตตัวเลือกประเภทข้อความสั่งให้กับแต่ละข้อความสั่งที่ปรากฏในลำดับแล้วจึงเติมข้อความสั่งลงไปยังสล็อตที่เลือก โดยที่เมื่อเติมข้อความสั่งลงไปแล้ว ลำดับการปรากฏของข้อความสั่งภายในบล็อกลำดับ จะต้องเป็นไปตามที่ปรากฏในลำดับข้อความสั่ง นอกจากนี้ เพื่อเป็นการป้องกันไม่ให้เกิดฟังก์ชั่น Spawn มากเกินไป ขั้นตอนวิธีสำหรับการเติมความสัมพันธ์จะพยายามตรวจสอบว่าฟังก์ชั่น Spawn ที่ปรากฏอยู่ในบล็อกลำดับอยู่แล้วเข้ากันกับฟังก์ชั่น Spawn ที่ปรากฏในลำดับข้อความสั่งหรือไม่ หากเข้ากันได้ จะไม่มีการเติมฟังก์ชั่น Spawn ดังกล่าว และใช้ฟังก์ชั่น Spawn ที่มีอยู่แล้วนั้นแทน

ขั้นตอนสำหรับการเติมความสัมพันธ์ที่มีคุณสมบัติดังที่ได้กล่าวสามารถสรุปได้ดังนี้

กำหนดให้

- ลำดับข้อความสั่งที่ต้องการเติม คือ $\mathrm{Re}l = a_1, a_2, .., a_m$ เมื่อข้อความสั่ง a_i ถือว่ามาก่อน a_j หาก $i < j$ $(1 \le i < m, 1 < j \le m)$
- ลำดับของไอเท็มที่อยู่ภายในบล็อกลำดับที่เป็นเป้าหมายของการเติม คือ $Skel = s_1, s_2, .., s_n$ เมื่อ s_i คือ ไอเท็มในบล็อกลำดับที่อยู่ตำแหน่งที่ i ซึ่งอาจเป็นฟังก์ชั่นการกระทำหรือสล็อตตัวเลือกประเภทข้อความสั่งก็ได้ โดยถ้าหาก s_i ปรากฏในบล็อกก่อน s_j จะได้ว่า $i < j$ $(1 \le i < n, 1 < j \le n)$
- การสุ่มเลือกสล็อตจะต้องไม่เกินตำแหน่งที่ B ในกรณีที่ $B > n$ ให้ตรวจสอบว่าข้อความสั่งสุดท้ายของ $Skel$ เป็นฟังก์ชั่นเปลี่ยนสถานะหรือไม่ ถ้าใช่ ให้ B มีค่าเป็นตำแหน่งของข้อความสั่งดังกล่าว

ขั้นตอนสุ่มเลือกสล็อตสามารถทำได้ดังนี้

1) สร้างลำดับย่อย $\mathrm{Re}lSpawn$ ซึ่งเป็นลำดับย่อยของ $\mathrm{Re}l$ ที่ประกอบด้วยข้อความสั่งที่เป็นฟังก์ชั่น Spawn เท่านั้น
2) สร้างลำดับย่อย $SkelSpawn$ ซึ่งเป็นลำดับย่อยของ $Skel$ ที่ประกอบด้วยข้อความสั่งที่เป็นฟังก์ชั่น Spawn เท่านั้น และตรงตามเงื่อนไข $m < B$ เมื่อ s_m เป็นไอเท็มสุดท้ายของ $SkelSpawn$ และเป็นไอเท็มที่ m ของ $Skel$

3) หาลำดับย่อยร่วมยาวสุดของ $\mathrm{Rel}Spawn$ และ $SkelSpawn$ ทุกเอ็มเบ็ดดิ้ง โดยสมาชิกของลำดับจะถือว่าเหมือนกัน ถ้าฟังก์ชั่น Spawn ทั้งสองสามารถเข้ากันได้ตามนิยามในหัวข้อ 6.1 และสุ่มเลือกมา 1 เอ็มเบ็ดดิ้ง
4) กำหนดให้ลำดับย่อยที่เลือกแทนด้วย $CommonSpawn = c(i_1, j_1), c(i_2, j_2), .., c(i_k, j_k)$ เมื่อ $c(i_d, j_d)$ คือฟังก์ชั่น Spawn ซึ่งเป็นฟังก์ชั่นที่เข้ากันได้ระหว่างสมาชิกตัวที่ i_d ของ Rel และสมาชิกตัวที่ j_d ของ $Skel$
5) หากมีลำดับย่อย $CommonSpawn$ ให้ทำการรวมฟังก์ชั่นโดยใช้กฎต่อไปนี้กับทุกสมาชิก $c(i_d, j_d)$ ของลำดับ
    - ถ้าสมาชิกตัวที่ i_d ของ Rel มีอากิวเมนท์เป็นตัวระบุแต่สมาชิกตัวที่ j_d ของ $Skel$ มีอากิวเมนท์เป็นสล็อตจำเป็น ให้นำตัวระบุของสมาชิกที่ i_d ไปเติมในสล็อตจำเป็นของสมาชิกที่ j_d
    - หากเป็นกรณีอื่น ไม่จำเป็นต้องกระทำการใด ๆ เพิ่มเติม
6) กำหนดสล็อตเป้าหมายสำหรับข้อความสั่งอื่น ๆ ใน Rel ที่ไม่ได้เป็นสมาชิกของ $\mathrm{Rel}Spawn$ โดยสร้าง CSP ดังนี้
    - เซ็ตของตัวแปรที่สนใจ $X = \{x_1, x_2, .., x_m\}$ เมื่อค่าของตัวแปร x_i คือ สล็อตที่จะนำฟังก์ชั่น a_i ไปใส่ $(1 \le i \le m)$
    - โดเมนสำหรับตัวแปรทุกตัวใน X คือ เซ็ตของตำแหน่งของสล็อตทางเลือกทั้งหมดที่มีอยู่ในบล็อกที่เลือกที่อยู่ไม่เกินตำแหน่งที่ B
        - เซ็ตของตำแหน่งของสล็อตทางเลือกสร้างได้จาก $Skel$ แสดงโดยรหัสเทียมดังนี้

```
ResultSet = { }
For( s_i in Skel ) do:
    If( s_i is [OPT] slot ) then
        ResultSet ← i
Return ResultSet as result
```

        - กรณีที่มีลำดับย่อย $CommonSpawn$ ให้กำหนดโดเมนตามแต่สมาชิก $c(i_d, j_d)$ ของลำดับย่อย โดยให้โดเมนของ X_{i_d} เป็น $\{j_d\}$
    - ข้อจำกัดของปัญหา คือ $x_1 \le x_2 \le .. \le x_m < B$
7) แก้ปัญหาด้วยตัวแก้ปัญหาบังคับเพื่อหาค่าให้ทุกตัวแปรในเซ็ต X ในกรณีที่มีหลายคำตอบให้สุ่มเลือก 1 คำตอบ
8) เติมข้อความสั่งลงในสล็อตที่ระบุไว้ โดยสนใจเฉพาะข้อความสั่งที่ยังไม่ได้เป็นส่วนหนึ่งของ $CommonSpawn$ แสดงด้วยขั้นตอนย่อยดังนี้

```
Reverse Rel to a_n, a_{n-1}, .., a_1
For( a_i in Rel ) do:
```

If(a_i is referenced by *CommonSpawn*) do:

Continue

If(the slot at x_i is already occupied) do:

Fill a_i in the slot at x_i preceding all statements in the slot

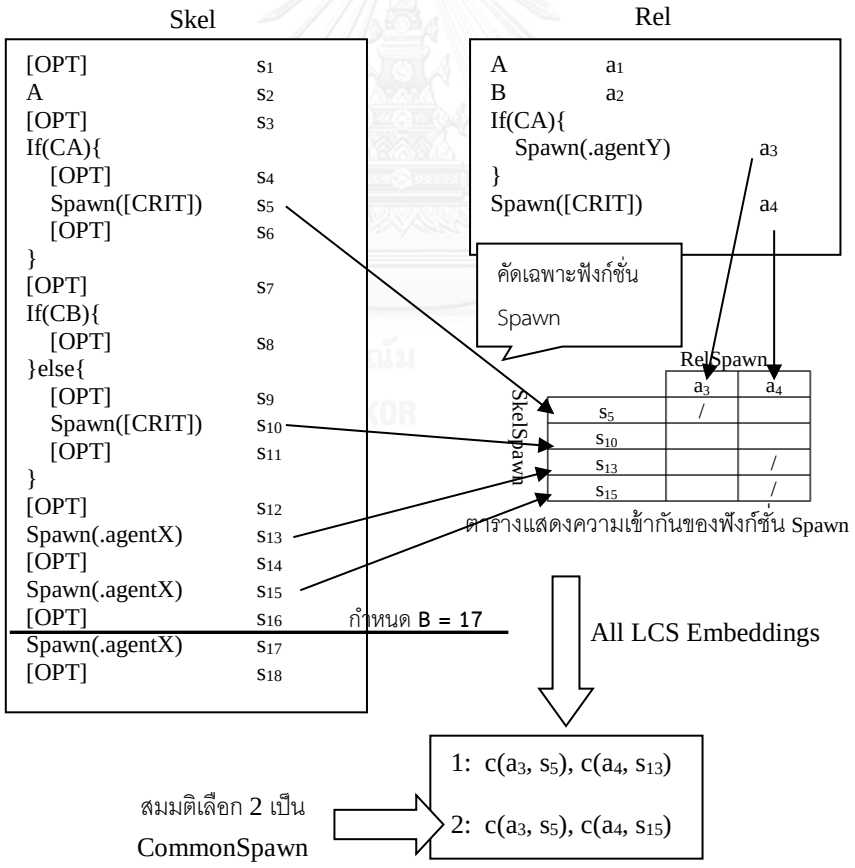
Else

Fill a_i in the slot at x_i

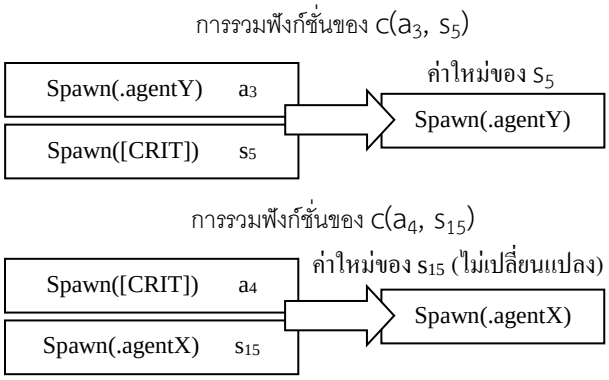
9) ทำการตรวจสอบบล็อกลำดับเป้าหมายเพื่อสร้างสล็อตทางเลือกขึ้นมาใหม่ให้ถูกต้องตามนิยามของโครงภาคีในหัวข้อ 6.1

รูปที่ *25* ถึงรูปที่ *28* แสดงตัวอย่างของขั้นตอนวิธีโดยกำหนด B เป็น 17 และสมมติ Rel และ $Skel$ ตามรูป

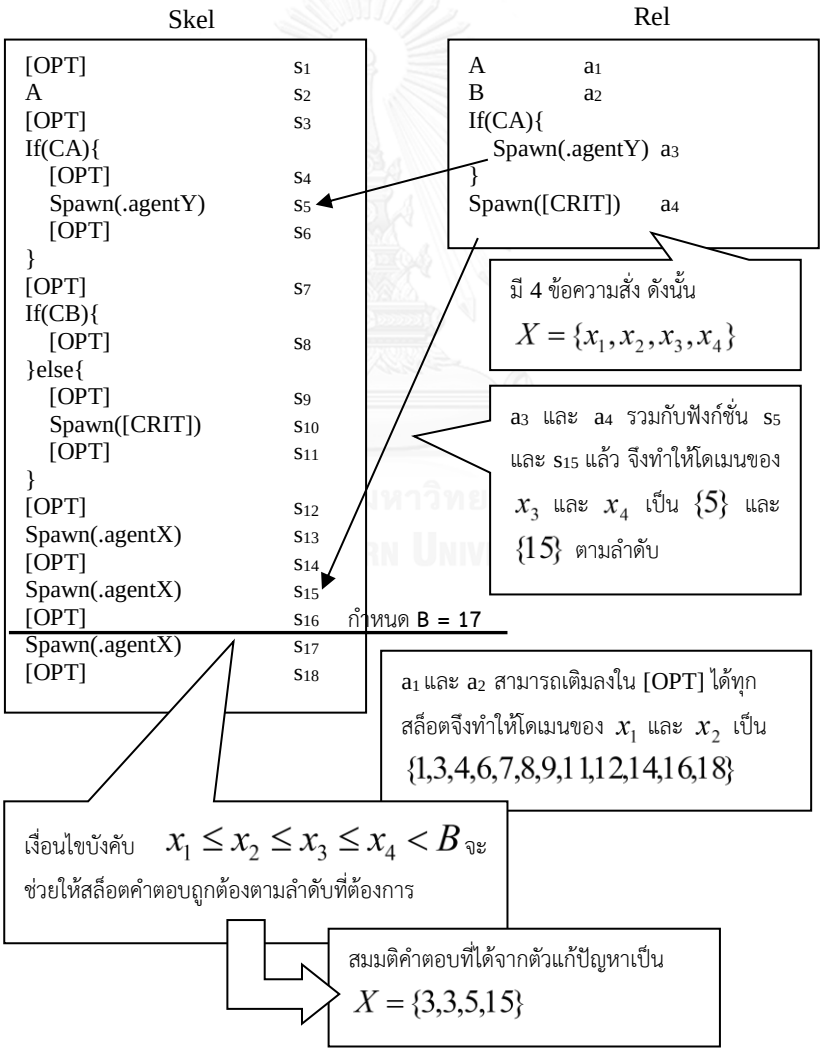
ข้อจำกัดของการเติมข้อความสั่งด้วยวิธีที่นำเสนอ คือ ไม่มีการตรวจสอบสล็อตเมื่อต้องเติมฟังก์ชั่นเปลี่ยนสถานะ ดังนั้น หากในลำดับข้อความสั่งมีฟังก์ชั่นเปลี่ยนสถานะอยู่ หากเติมด้วยวิธีการนี้ อาจทำให้ฟังก์ชั่นดังกล่าวแทรกอยู่ระหว่างฟังก์ชั่นการกระทำอื่น ๆ ส่งผลให้ฟังก์ชั่นการกระทำที่อยู่ถัดจากฟังก์ชั่นเปลี่ยนสถานะดังกล่าวไม่ถูกประมวลผล ดังนั้น ลำดับข้อความสั่งที่จะเติมลงในโครงภาคีด้วยวิธีนี้ได้ จะต้องไม่มีฟังก์ชั่นเปลี่ยนสถานะอยู่



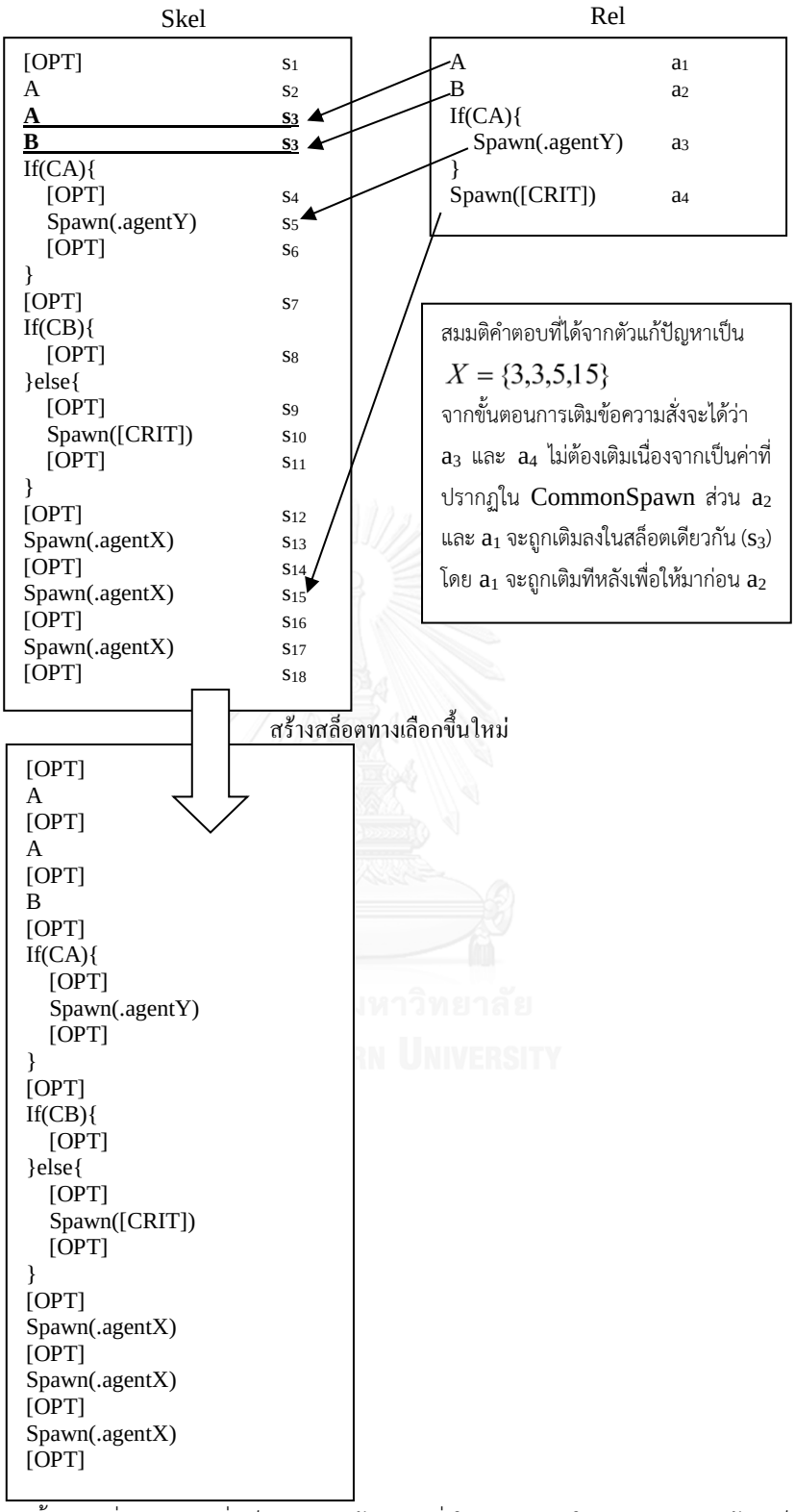
รูปที่ *25* แสดงค่าของ Rel และ $Skel$ รวมถึงขั้นตอนที่ 1 – 4 ของขั้นตอนวิธีซึ่งเป็นการหาลำดับย่อย *CommonSpawn*



รูปที่ 26 แสดงขั้นตอนที่ 5 ซึ่งเป็นการรวมฟังก์ชั่นในกรณีที่มีลำดับย่อย *CommonSpawn*



รูปที่ 27 แสดงขั้นตอนที่ 6 และ 7 ซึ่งเป็นการสร้าง CSP เพื่อกำหนดสล็อตให้ข้อความสั่งใน Re*l*



รูปที่ 28 แสดงขั้นตอนที่ 8 และ 9 ซึ่งเป็นการเติมข้อความสั่งใน Rel ลงใน *Skel* และสร้างสล็อตทางเลือกเพื่อให้โครงภาคีเป็นไปตามนิยามในหัวข้อ 6.1

6.5 การเติมฟังก์ชั่นเปลี่ยนสถานะ

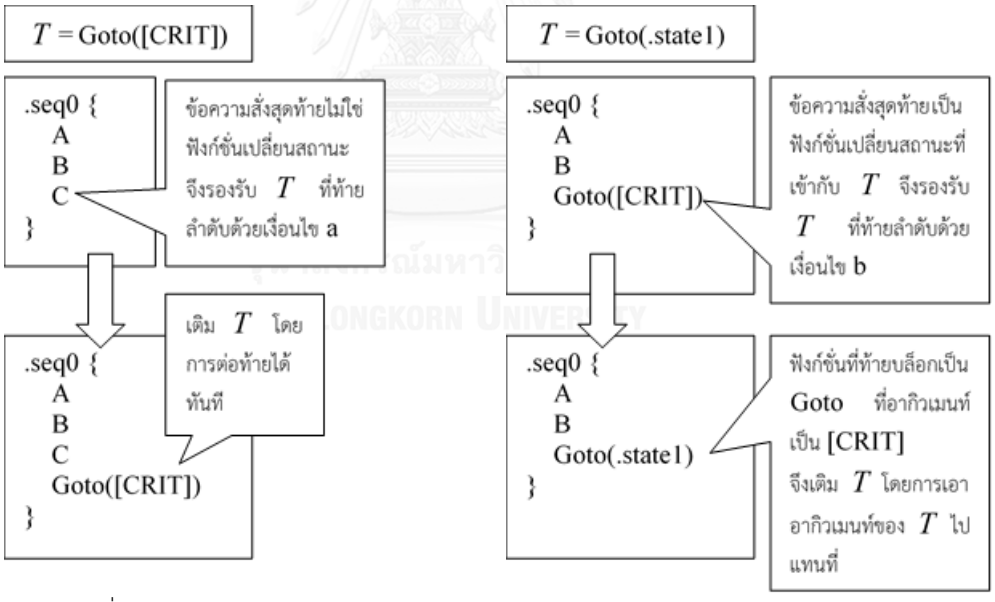
หัวข้อนี้กล่าวถึงการเติมฟังก์ชั่นเปลี่ยนสถานะลงในบล็อกที่สามารถรองรับฟังก์ชั่นดังกล่าวได้ โดยแบ่งเป็น 2 วิธี ขึ้นกับคุณลักษณะของบล็อกเป้าหมายว่าสามารถรองรับฟังก์ชั่นเปลี่ยนสถานะได้ในลักษณะใด สาเหตุที่จำเป็นต้องมีวิธีเติมฟังก์ชั่นเปลี่ยนสถานะแยกออกจากการเติมลำดับข้อความสั่ง เนื่องจากวิธีเติมลำดับข้อความสั่งที่กล่าวถึงนั้นไม่รองรับฟังก์ชั่นเปลี่ยนสถานะ ทำให้ต้องแยกฟังก์ชั่นเปลี่ยนสถานะออกมาเติมต่างหาก

วิธีการเติมฟังก์ชั่นเปลี่ยนสถานะ T แบบท้ายลำดับ

เป็นวิธีการเติม T ลงในบล็อกที่รองรับ T ที่ท้ายลำดับได้ตามเงื่อนไขที่ระบุในหัวข้อ 6.1 โดยจะแตกต่างตามเงื่อนไขที่ทำให้รองรับ T ที่ท้ายลำดับได้

1) ถ้ารองรับได้จากเงื่อนไข a ให้เติม T ลงในสล็อตตัวเลือกสุดท้ายของบล็อกลำดับดังกล่าว
2) ถ้ารองรับได้จากเงื่อนไข b จะถือว่าเติม T ลงในโครงภาคีแล้ว ในตำแหน่งเดียวกับข้อความสั่งสุดท้ายของบล็อกลำดับดังกล่าว ยกเว้น กรณีที่ข้อความสั่งนั้นเป็นฟังก์ชั่น Goto ที่มีอากิวเมนท์เป็นสล็อตจำเป็นประเภทตัวระบุ ให้นำอากิวเมนท์ของ T ไปแทนที่สล็อตดังกล่าว

รูปที่ *29* แสดงตัวอย่างการเติม T สำหรับทั้ง 2 กรณีข้างต้น



รูปที่ 29 (ซ้าย) แสดงวิธีเติม T สำหรับกรณี 1 (ขวา) แสดงวิธีเติม T สำหรับกรณี 2

วิธีการเติมฟังก์ชั่นเปลี่ยนสถานะ T แบบท้ายบล็อก

เป็นวิธีการเติม T ลงในบล็อกที่รองรับ T ที่ท้ายบล็อกได้ตามเงื่อนไขที่ระบุในหัวข้อ 6.1 โดยให้พิจารณาทุกเงื่อนไขที่ทำให้รองรับ T ได้ เพื่อหาตำแหน่งที่สามารถเติม T ได้ทั้งหมด จากนั้นจึงสุ่มเติมลงไป 1 ตำแหน่ง

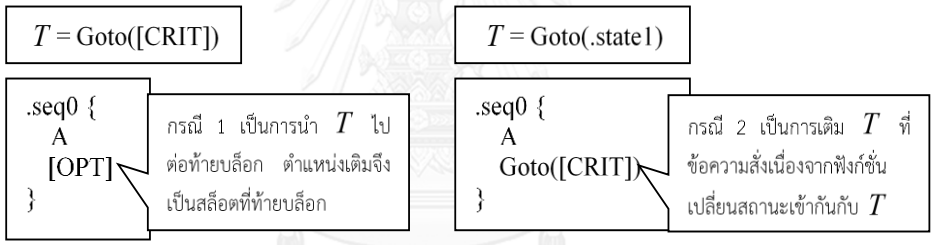
ในที่นี้สามารถหาตำแหน่งที่เติม T ได้ทั้งหมด ดังนี้

1) ถ้ารองรับ T ได้จากเงื่อนไข c โดยการรองรับที่ท้ายลำดับดังกล่าวนั้นรองรับได้จากเงื่อนไข a ตำแหน่งที่สามารถเติม T ได้ คือ สล็อตตัวเลือก
2) ถ้ารองรับ T ได้จากเงื่อนไข c โดยการรองรับที่ท้ายลำดับดังกล่าวนั้นรองรับได้จากเงื่อนไข b ตำแหน่งที่สามารถเติม T ได้ คือ ข้อความสั่งสุดท้ายของบล็อกลำดับ
3) ถ้ารองรับ T ได้จากเงื่อนไข d ตำแหน่งที่สามารถเติม T ได้ คือ สล็อตตัวเลือกทั้งหมดที่อยู่ท้ายทุกบล็อกที่เป็นไปตามเงื่อนไข
4) ถ้ารองรับ T ได้จากเงื่อนไข e ตำแหน่งที่สามารถเติม T ได้ คือ ข้อความสั่งสุดท้ายของบล็อกทั้งหมดที่เป็นไปตามเงื่อนไข

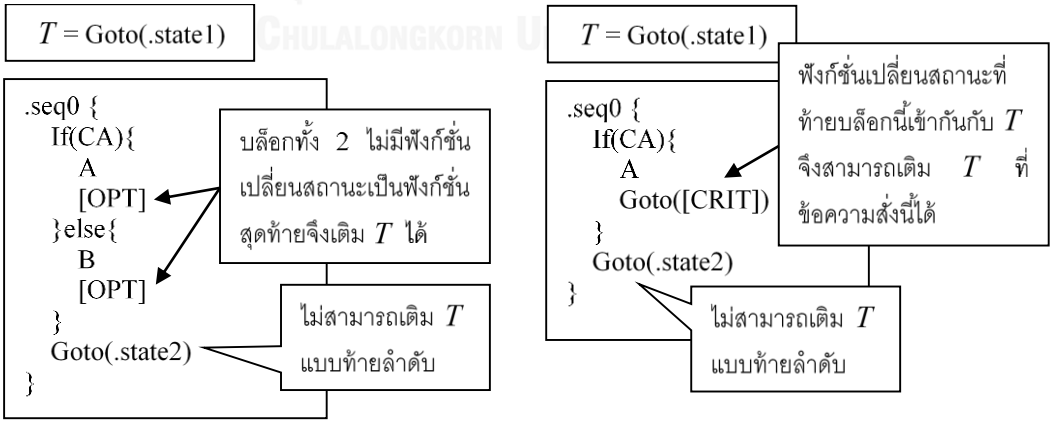
เมื่อสุ่มเลือกตำแหน่งแล้ว ให้ทำการเติม T โดย

- หากตำแหน่งที่สุ่มเลือกเป็นสล็อต ให้เติมลงในสล็อตนั้น
- หากตำแหน่งที่สุ่มเลือกเป็นข้อความสั่ง (ซึ่งเป็นฟังก์ชั่นที่เข้ากับ T) จะถือว่าเติม T แล้ว ยกเว้นข้อความสั่งนั้นเป็นฟังก์ชั่น Goto ที่มีอากิวเมนท์เป็นสล็อตจำเป็นประเภทตัวระบุ ให้นำอากิวเมนท์ของ T ไปแทนที่สล็อตดังกล่าว

รูปที่ 30 แสดงตำแหน่งที่สามารถเติม T ได้สำหรับแต่ละกรณี และรูปที่ 31 แสดงวิธีการเติม T



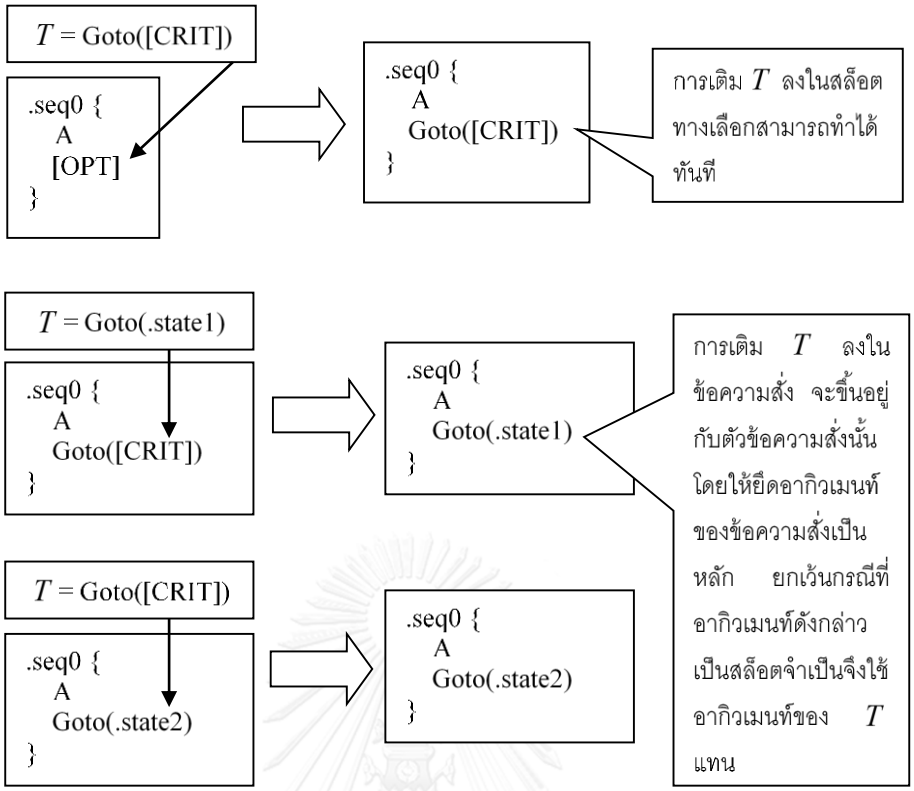
กรณี 1 และ 2 เป็นกรณีเดียวกับการเติม T แบบท้ายลำดับ



กรณี 3 เป็นการหาสล็อตที่อยู่ท้ายบล็อกที่ไม่ได้ลงท้ายด้วยฟังก์ชั่นเปลี่ยนสถานะ

กรณี 4 เป็นการหาข้อความสั่งที่ท้ายบล็อกที่เข้ากันกับ T

รูปที่ 30 แสดงตำแหน่งที่สามารถเติม T ได้ในกรณีต่าง ๆ โดยตำแหน่งดังกล่าวอาจเป็นสล็อตทางเลือก (ในภาพแสดงเฉพาะสล็อตที่สำคัญต่อการอธิบายเท่านั้น) หรืออาจเป็นข้อความสั่งที่เข้ากันกับ T ก็ได้



รูปที่ 31 แสดงตัวอย่างการเติม T ทั้งกรณีที่ตำแหน่งที่เติมเป็นตำแหน่งของสล็อตและของข้อความสั่ง

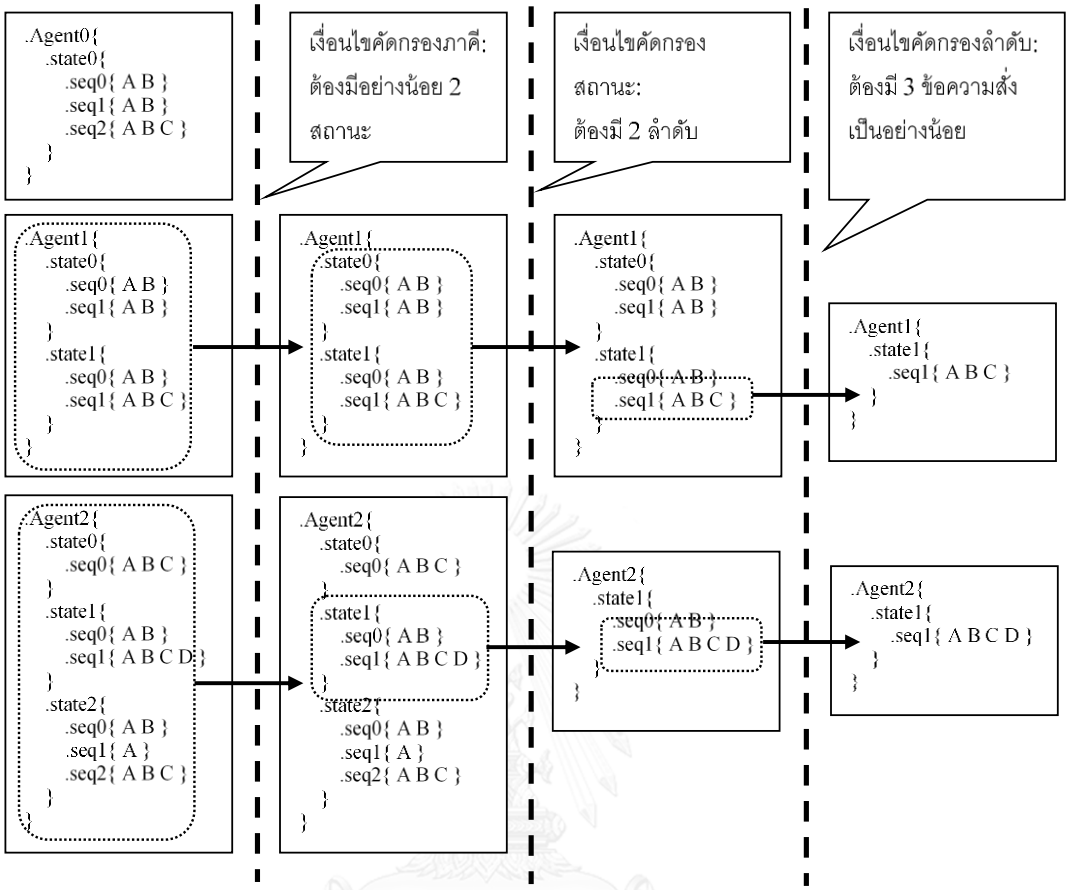
6.6 การคัดกรองบล็อกเป้าหมาย

การเติมความสัมพันธ์ลงในภาคีจะต้องมีการเลือกบล็อกเป้าหมายที่เหมาะสมกับความสัมพันธ์ดังกล่าว ในที่นี้จึงต้องมีการคัดกรองบล็อกลำดับที่ตรงตามความต้องการออกมาจากบล็อกลำดับทั้งหมดที่ปรากฏอยู่ในโครงภาคี หัวข้อนี้จะกล่าวถึงวิธีการคัดกรองบล็อกลำดับ และผลลัพธ์ของการกรองเมื่อกำหนดเงื่อนไขคัดกรอง โดยเงื่อนไขสำหรับการคัดกรองประกอบด้วย 3 เงื่อนไข ได้แก่ เงื่อนไขคัดกรองบล็อกภาคี เงื่อนไขคัดกรองบล็อกสถานะ และเงื่อนไขคัดกรองบล็อกลำดับ

การคัดกรองบล็อกลำดับจากโครงภาคี จะเริ่มจากการคัดกรองบล็อกภาคีทั้งหมดที่ปรากฏในโครงภาคีและตรงตามเงื่อนไขคัดกรองบล็อกภาคี จากนั้นจึงคัดกรองบล็อกสถานะ โดยตรวจสอบบล็อกสถานะในทุกบล็อกภาคีที่ผ่านการคัดกรอง และคัดเฉพาะบล็อกสถานะที่เป็นไปตามเงื่อนไขคัดกรองบล็อกสถานะ และในขั้นตอนสุดท้ายจึงตรวจสอบและคัดกรองบล็อกลำดับด้วยเงื่อนไขคัดกรองบล็อกลำดับ รูปที่ 32 แสดงตัวอย่างการคัดกรอง โดยใช้เงื่อนไขต่อไปนี้

- เงื่อนไขคัดกรองบล็อกภาคี : ภาคีต้องมีอย่างน้อย 2 สถานะ
- เงื่อนไขคัดกรองบล็อกสถานะ : สถานะต้องมีอย่างน้อย 2 ลำดับพฤติกรรม
- เงื่อนไขคัดกรองบล็อกลำดับ : บล็อกลำดับต้องมีอย่างน้อย 3 ข้อความสั่ง

ซึ่งได้ผลลัพธ์เป็น 2 บล็อกลำดับ ได้แก่ บล็อกลำดับ seq1 ของสถานะ state1 ในภาคี Agent1 และบล็อกลำดับ seq1 ของสถานะ state1 ในภาคี Agent2



รูปที่ 32 ตัวอย่างการคัดกรองบล็อก

6.7 การหาวิธีการเติมรายการฟังก์ชั่นเปลี่ยนสถานะ T_{list} ที่ดีที่สุด (Best T_{list} –allocation)

ในบางกรณีของการเติมความสัมพันธ์ลงในภาคี จะต้องมีการเติมฟังก์ชั่นเปลี่ยนสถานะหลายฟังก์ชั่นลงในบล็อกสถานะ โดยแต่ละฟังก์ชั่นจะถูกเติมลงในบล็อกลำดับที่แตกต่างกันที่อยู่ภายในสถานะนั้น ปัญหาที่มีโอกาสเกิดขึ้นจากสถานการณ์นี้ คือ มีจำนวนบล็อกลำดับที่รองรับฟังก์ชั่นเปลี่ยนสถานะทั้งหมดใน T_{list} ได้ไม่เพียงพอ วิธีการแก้ไขที่ดีที่สุดในสถานการณ์นี้โดยทำให้โครงภาคีมีความเปลี่ยนแปลงน้อยที่สุด คือ การพยายามเติมฟังก์ชั่นเปลี่ยนสถานะลงไปให้ได้มากที่สุดเท่าที่จะทำได้ ซึ่งมีความเป็นไปได้ที่จะมีวิธีเติมหลายวิธี

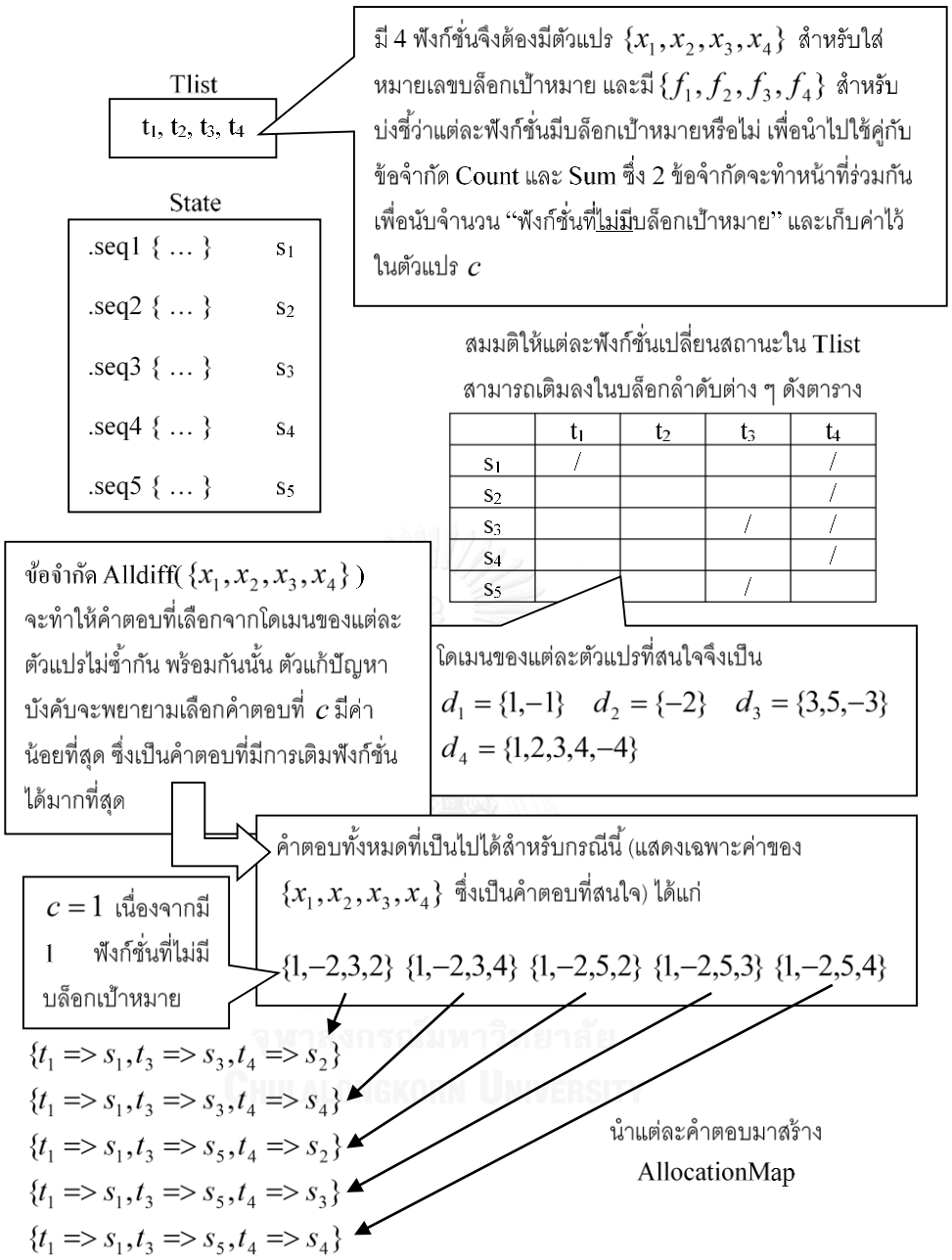
ในหัวข้อนี้จะนำเสนอขั้นตอนสำหรับหาวิธีเติมฟังก์ชั่นเปลี่ยนสถานะที่ดีที่สุดทุกวิธีที่เป็นไปได้โดยการแปลงปัญหาให้อยู่ในรูป CSP และแก้ด้วยตัวแก้ปัญหาบังคับ

กำหนดให้ $T_{list} = \{t_1, t_2, ..., t_n\}$ เป็นรายการของฟังก์ชั่นเปลี่ยนสถานะทั้งหมดที่ต้องการเติมลงในบล็อกสถานะ จำนวน n ฟังก์ชั่นและ $State = \{s_1, s_2, ..., s_m\}$ เป็นบล็อกสถานะที่สนใจซึ่งประกอบด้วยบล็อกลำดับ m บล็อก ขั้นตอนที่จะนำเสนอจะคำนวณหาวิธีเติมฟังก์ชั่นใน T_{list} ให้ได้มากที่สุด พร้อมค่าใช้จ่ายที่ใช้ (ยิ่งน้อยแสดงว่าสามารถเติมฟังก์ชั่นเปลี่ยนสถานะได้มาก) ซึ่งอาจมีได้หลายคำตอบ โดยวิธีเติมจะอยู่ในรูปของข้อมูลการส่ง (Map) $AllocationMap = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าฟังก์ชั่นเปลี่ยนสถานะ k_i ใน T_{list} จะต้อง

นำไปเติมในบล็อกลำดับ v_i ใน $State$ รายละเอียดของขั้นตอนเป็นดังด้านล่าง โดยรูปที่ 33 แสดงตัวอย่างการหาวิธีเติมฟังก์ชั่นเปลี่ยนสถานะ 4 ฟังก์ชั่นลงในบล็อกสถานะที่มีบล็อกลำดับ 5 บล็อก

1) สร้าง CSP ซึ่งมีรายละเอียดต่อไปนี้
   - เซ็ตของตัวแปรที่สนใจ $X = \{x_1, x_2, .., x_n\} \cup \{f_1, f_2, .., f_n\} \cup c$
     - ค่าของตัวแปร x_i คือ หมายเลขแทนบล็อกลำดับที่จะนำฟังก์ชั่น t_i ไปใส่ $(1 \le i \le n)$ หากค่าของตัวแปรไม่เป็นจำนวนเต็มบวก แปลว่า ไม่มีบล็อกลำดับที่จะนำ t_i ไปใส่
     - ค่าของตัวแปร f_i คือ ตัวบ่งชี้ว่ามีบล็อกลำดับที่จะนำ t_i ไปใส่หรือไม่
     - ค่าของ c คือ จำนวนของฟังก์ชั่นเปลี่ยนสถานะที่ไม่มีบล็อกลำดับให้นำไปใส่
   - โดเมนสำหรับตัวแปร $D = \{d_1, d_2, .., d_n\} \cup \{fd_1, fd_2, ..., fd_n\} \cup cd$
     - d_i $(1 \le i \le n)$ คือโดเมนของ x_i โดยเป็นเซ็ตซึ่งประกอบด้วยหมายเลขแทนบล็อกลำดับทั้งหมดที่สามารถนำฟังก์ชั่น t_i ไปใส่ได้ และค่า $-i$ ซึ่งมีไว้แสดงถึงกรณีที่ฟังก์ชั่น t_i ไม่ถูกนำไปเติม
     - fd_i $(1 \le i \le n)$ คือโดเมนของ f_i โดย $fd_i = \{0,1\}$ โดยค่า 0 แทนกรณีที่มีบล็อกให้เติม t_i
     - cd คือโดเมนของ c โดย $cd = \{0\} \cup I^+$
   - ข้อจำกัดของปัญหา คือ
     - $Count(\{x_1, ..., x_n\}, -i, f_i)$ สำหรับทุก i $(1 \le i \le n)$
     - $Sum(\{f_1, ..., f_n\}, c)$
     - $Alldiff(\{x_1, ..., x_n\})$
2) ใช้ตัวแก้ปัญหาบังคับหาคำตอบของ CSP ภายใต้เงื่อนไขให้เลือกเฉพาะคำตอบที่ทำให้ c มีค่าน้อยที่สุด ซึ่งอาจมีได้หลายคำตอบ
3) แต่ละคำตอบที่ได้จากตัวแก้ปัญหาบังคับสามารถจัดทำเป็นคำตอบที่ต้องการ (วิธีเติมฟังก์ชั่นเปลี่ยนสถานะที่ดีที่สุด) ได้โดยเติมเต็มข้อมูลการส่งด้วยวิธีด้านล่าง ทุกคำตอบที่ได้จะมีค่าใช้จ่ายเท่ากับ c ทั้งหมด

$AllocationMap = \{\}$
For(Assignment x_i in $\{x_1, x_2, .., x_n\}$) do:
  If($x_i > 0$) do:
    $AllocationMap \leftarrow (t_i \Rightarrow$ sequence block indicated by $x_i)$



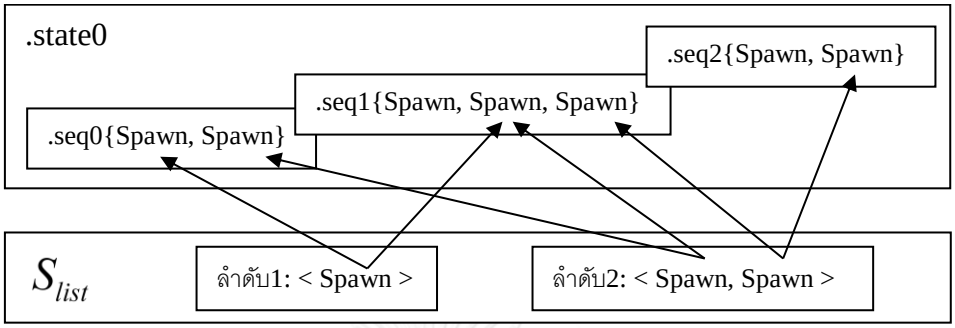
รูปที่ 33 ตัวอย่างการหาวิธีเติมฟังก์ชั่นเปลี่ยนสถานะจำนวน 4 ฟังก์ชั่นลงในสถานะที่มีบล็อกลำดับ 5 บล็อก

6.8 การหาวิธีการเติมรายการลำดับข้อความสั่ง S_{list} ที่ทำให้เกิดฟังก์ชั่น Spawn น้อยที่สุด (Best spawn-matched S_{list} -allocation)

ในกรณีของการเติมความสัมพันธ์แบบการกระทำคู่ขนานข้ามภาคีจะพยายามทำให้เกิดความเปลี่ยนแปลงกับโครงภาคีน้อยที่สุด (ในกรณีนี้ คือ เกิดฟังก์ชั่น Spawn จากการเติมน้อยที่สุด) เมื่อทำการเติม S_{list} ลงในบล็อกสถานะที่สนใจ ประเด็นที่ต้องพิจารณาในการเติมแต่ละลำดับข้อความสั่งใน S_{list} คือ จะต้องเลือกบล็อกลำดับใดให้กับลำดับข้อความสั่งใด ที่ส่งผลให้เมื่อเติมแล้วเกิดฟังก์ชั่น Spawn น้อยที่สุด ตัวอย่าง เช่น กำหนดให้ S_{list} เป็น

รายการที่มี 2 ลำดับ และบล็อกสถานะที่สนใจมี 3 บล็อกลำดับ จะสามารถพิจารณาได้ด้วยวิธีดังรูปที่ 34 (แสดงเฉพาะฟังก์ชั่น Spawn)

นอกจากนี้ การเติมลำดับข้อความสั่งอาจมีกรณีที่จำนวนลำดับมีมากกว่าจำนวนบล็อกลำดับในบล็อกสถานะที่สนใจ ทำให้มีบางลำดับข้อความสั่งที่ไม่มีบล็อกลำดับเป้าหมาย



สัญลักษณ์ลูกศรแสดงว่าฟังก์ชั่น Spawn เข้ากัน

วิธีเลือกบล็อกที่ดีที่สุดให้ S_{list} คือ

- เลือก .seq0 สำหรับลำดับที่ 1 และ .seq1 สำหรับลำดับที่ 2 (ฟังก์ชั่น Spawn เข้ากันกับที่มีอยู่ทั้งหมด ทำให้ไม่มีฟังก์ชั่น Spawn เกิดใหม่เมื่อเติมลำดับลงในบล็อกที่เลือก)

ตัวอย่างวิธีเลือกบล็อกที่ไม่ดี เช่น

- เลือก .seq1 ให้ลำดับที่ 1 และเลือก .seq2 ให้ลำดับที่ 2 (ส่งผลให้ฟังก์ชั่น Spawn ตัวแรกของลำดับที่ 2 ไม่มีฟังก์ชั่นที่เข้ากัน และต้องเพิ่มฟังก์ชั่น Spawn ใหม่ใน .seq2 เมื่อเติมลำดับ)

รูปที่ 34 ตัวอย่างวิธีเลือกบล็อกลำดับ

ในงานวิทยานิพนธ์นี้จะทำการพิจารณาเลือกบล็อกโดยแปลงปัญหาให้อยู่ในรูป CSP และแก้ด้วยตัวแก้ปัญหาบังคับโดยพยายามทำให้ค่าใช้จ่ายน้อยที่สุด ซึ่งค่าใช้จ่ายจะแตกต่างกันออกไปสำหรับการเลือกแต่ละบล็อกลำดับ

กำหนด $S_{list} = \{r_1, r_2, ..., r_n\}$ เป็นรายการของลำดับข้อความสั่งที่ต้องการเติมจำนวน n ลำดับและ $State = \{s_1, s_2, ..., s_m\}$ เป็นบล็อกสถานะที่สนใจซึ่งประกอบด้วยบล็อกลำดับ m บล็อก ขั้นตอนที่จะนำเสนอจะคำนวณหาวิธีเติมที่ทำให้เกิดฟังก์ชั่น Spawn ใหม่น้อยที่สุด พร้อมค่าใช้จ่ายที่ใช้ (ยิ่งน้อยแสดงว่ายิ่งมีฟังก์ชั่น Spawn เกิดใหม่น้อย) ซึ่งวิธีเติมอาจมีได้หลายวิธี โดยวิธีเติมจะอยู่ในรูปของข้อมูลการส่ง $AllocationMap = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าลำดับข้อความสั่ง k_i ใน S_{list} จะต้องนำไปเติมในบล็อกลำดับ v_i ใน $State$ รายละเอียดของขั้นตอนเป็นดังด้านล่าง โดยรูปที่ *35* และรูปที่ *36* แสดงตัวอย่างการหาวิธีเติมรายการลำดับข้อความสั่งที่ประกอบด้วย 2 ข้อความสั่งลงในบล็อกสถานะที่มีบล็อกลำดับ 3 บล็อก

1) สร้างตารางค่าใช้จ่าย $CostTable$ ซึ่ง $CostTable[i][j]$ แสดงค่าใช้จ่ายที่เกิดขึ้นหากเติม $r_i \quad (1 \le i \le n)$ ลงในบล็อกลำดับ $s_j \quad (1 \le j \le m)$ และ $CostTable[i][0]$ แสดงค่าใช้จ่ายจากการไม่เลือกบล็อกลำดับใดๆให้ r_i โดยการสร้างตารางสามารถทำได้ดังนี้

$CostTable = [][]$

For(Sequence r_i in S_{list}) do:

  For(Sequence s_j in $State$) do:

    Find longest common subsequence of s_j and r_i where the element of both sequences is treated as equal if both are Spawn function and are compatible with each other

    Let the subsequence found be Seq

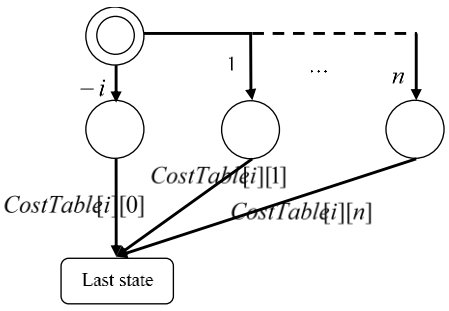
    $CostTable[i][j] = SpawnCount(r_i) - |Seq|$

  $CostTable[i][0] = SpawnCount(r_i)$

เมื่อ $SpawnCount(r)$ คืนค่าจำนวนฟังก์ชั่น Spawn ทั้งหมดที่ปรากฏในลำดับพฤติกรรม r

2) สร้าง CSP ซึ่งมีรายละเอียดต่อไปนี้

- เซ็ตของตัวแปรที่สนใจ $X = \{x_1, x_2, .., x_n\} \cup \{f_1, f_2, .., f_n\} \cup c$
  - ค่าของตัวแปร x_i คือ หมายเลขแทนบล็อกลำดับที่จะนำฟังก์ชั่น r_i ไปใส่ $(1 \le i \le n)$ หากค่าของตัวแปรไม่เป็นจำนวนเต็มบวก แปลว่า ไม่มีบล็อกลำดับที่จะนำ r_i ไปใส่
  - ค่าของตัวแปร f_i คือ ค่าใช้จ่ายที่เกิดจากการเลือกค่าของตัวแปร x_i
  1) ค่าของ c คือ ค่าใช้จ่ายรวมจากการเลือกค่าตัวแปรทั้งหมด
- โดเมนสำหรับตัวแปร $D = \{d_1, d_2, .., d_n\} \cup \{fd_1, fd_2, ..., fd_n\} \cup cd$
  1) $d_i \quad (1 \le i \le n)$ คือโดเมนของ x_i โดยเป็นเซ็ตซึ่งมีค่าเป็น $[1, m] \cup \{-i\}$ โดยค่า $-i$ มีไว้สำหรับกรณีที่ฟังก์ชั่น r_i ไม่ถูกนำไปเติม
  2) $fd_i \quad (1 \le i \le n)$ คือโดเมนของ f_i โดย $fd_i = \{0\} \cup I^+$
  3) cd คือโดเมนของ c โดย $cd = \{0\} \cup I^+$
- ข้อจำกัดของปัญหา คือ
  1) $\mathrm{Re}gular(fsm_i, \{x_i, f_i\})$ สำหรับทุก $i \quad (1 \le i \le n)$ โดยภาพด้านล่างแสดงเครื่องสถานะจำกัดสำหรับ fsm_i



2) $Alldiff(\{x_1,...,x_n\})$

3) $Sum(\{f_1,...,f_n\},c)$

3) ใช้ตัวแก้ปัญหาบังคับหาคำตอบของ CSP ภายใต้เงื่อนไขให้เลือกเฉพาะคำตอบที่ทำให้ c มีค่าน้อยที่สุด ซึ่งอาจมีได้หลายคำตอบ

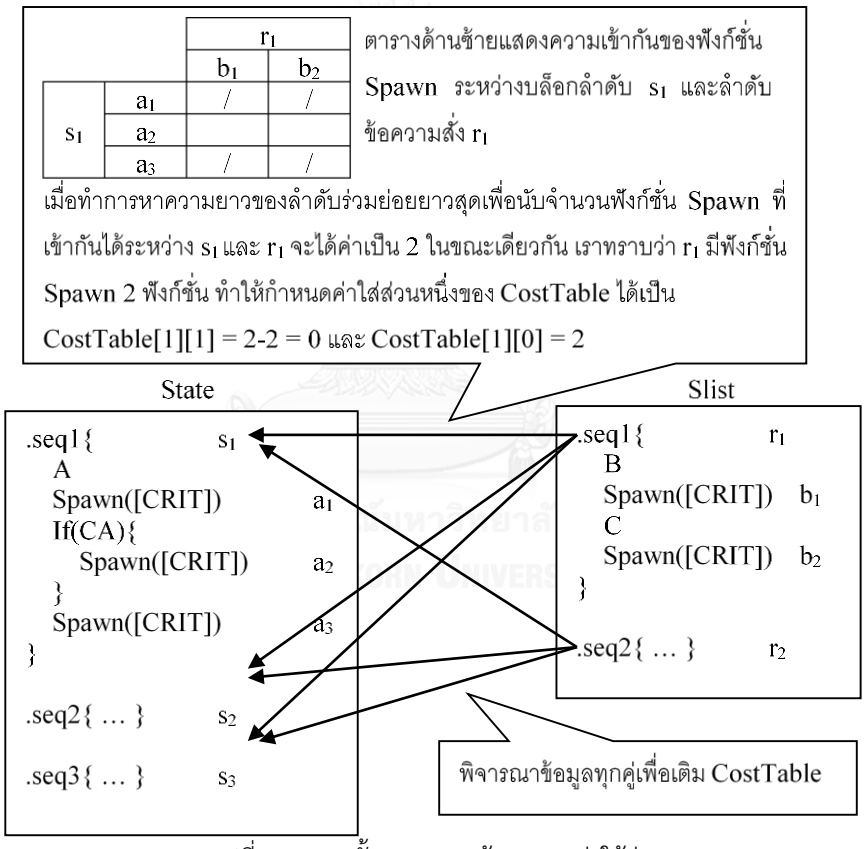
4) แต่ละคำตอบที่ได้จากตัวแก้ปัญหาบังคับสามารถจัดทำเป็นคำตอบที่ต้องการได้โดยเติมเต็มข้อมูลการส่งด้วยวิธีด้านล่าง ทุกคำตอบที่ได้จะมีค่าใช้จ่ายเท่ากับ c ทั้งหมด

$AllocationMap = \{\}$

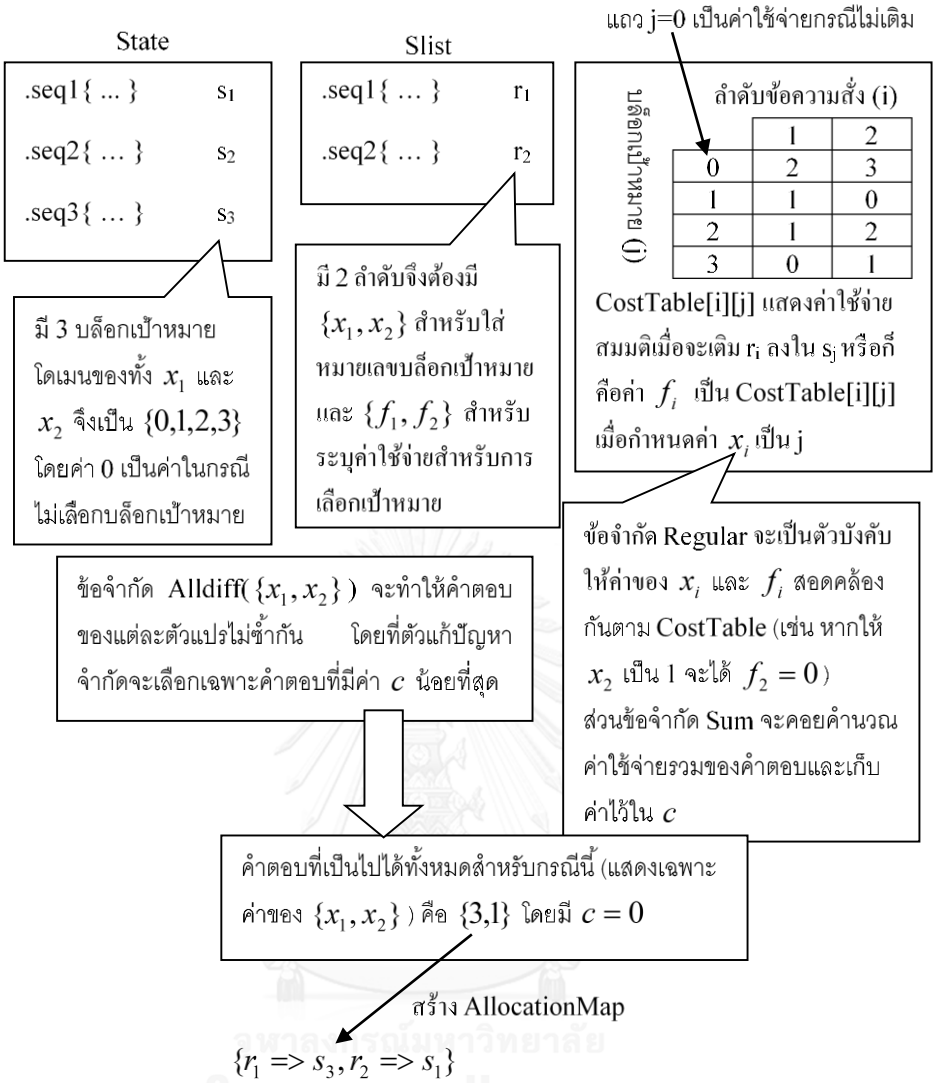
For(Assignment x_i in $\{x_1, x_2, .., x_n\}$) do:

  If($x_i > 0$) do:

    $AllocationMap \leftarrow (r_i \Rightarrow$ sequence block indicated by $x_i)$



รูปที่ 35 แสดงขั้นตอนการสร้างตารางค่าใช้จ่าย



รูปที่ 36 แสดงการหาวิธีเติมลำดับข้อความสั่งให้เกิดฟังก์ชั่น Spawn น้อยที่สุดด้วย CSP โดยใช้ข้อมูลจากตารางค่าใช้จ่าย

6.9 การเติมความสัมพันธ์รูปแบบลำดับฟังก์ชั่นการกระทำ

ความสัมพันธ์ที่ได้จากการทำเหมืองข้อมูลในหัวข้อ 5.3.1 จะอยู่ในรูปของลำดับของสตริง แต่ละสตริงจะถูกแปลงด้วยวิธีการแปลงสตริงในลำดับมาเป็นข้อความสั่งตามวิธีที่ระบุไว้ในหัวข้อ 6.3 ในที่นี้กำหนดให้ความสัมพันธ์ที่ผ่านวิธีการแปลงสตริงแล้วอยู่ในรูปของลำดับข้อความสั่งดังนี้ $Order =< a_1, a_2, .., a_n >$ เมื่อ a_i, $1 \le i \le n$ คือ ข้อความสั่ง การเติมความสัมพันธ์ $Order$ ลงในโครงภาคีทำได้ดังนี้

1) ลบทุกข้อความสั่งใน $Order$ ที่เป็นฟังก์ชั่นเปลี่ยนสถานะ และบันทึกฟังก์ชั่นสุดท้ายที่ลบไว้ (ถ้ามี) เรียกว่า T

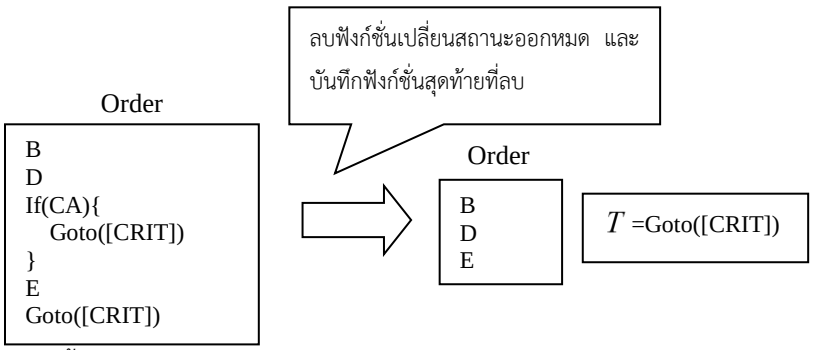
2) เลือกบล็อกลำดับที่จะเติมความสัมพันธ์โดยใช้ขั้นตอนวิธีด้านล่าง ซึ่งพยายามเลือกบล็อกลำดับที่สามารถเติม T ที่ท้ายบล็อกได้ หากเลือกไม่ได้จะทำการสร้างบล็อกลำดับใหม่ซึ่งสามารถเติม T ได้แน่นอน ในกรณีที่ไม่มี T จะทำการสุ่มเลือกบล็อกลำดับจากโครงภาคี

If(T exists) do:
  Filter skeleton using below conditions and store result in F_{result} :
    Agent condition = There exists at least 1 state
    State condition = There exists at least 1 sequence
    Sequence condition = Being EOS-T block
  If(F_{result} is not empty) do:
    Randomly select 1 sequence block from F_{result} as result
  Else do:
    Randomly select 1 state block from skeleton and create new sequence block Seq in that state block
    Select Seq as result
Else do:
  Randomly select 1 state block from skeleton and create new sequence block Seq
  Select Seq as result

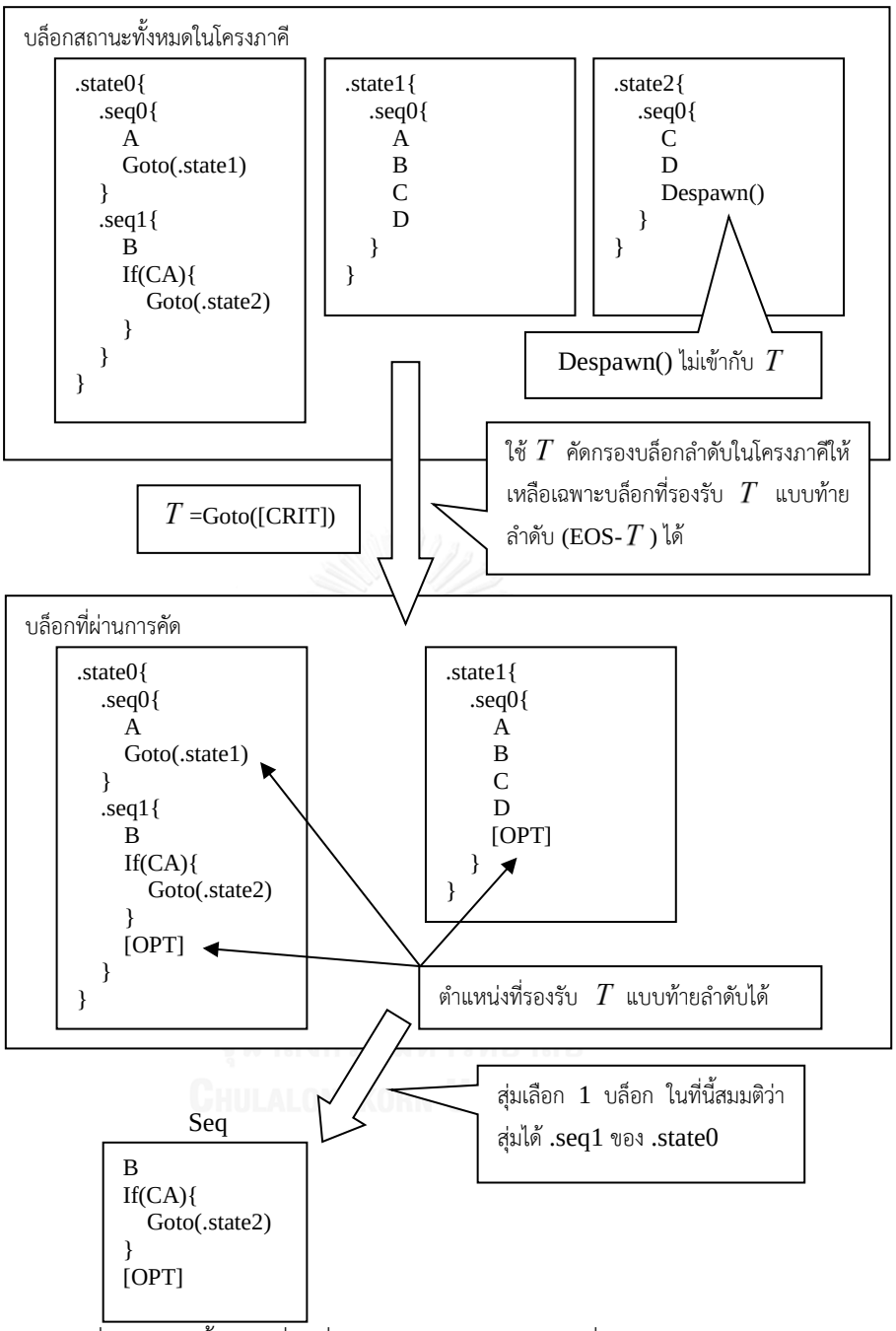
3) ถ้ามีฟังก์ชั่น T ให้เติมฟังก์ชั่น T แบบ<u>ท้ายลำดับ</u> ด้วยวิธีดังหัวข้อ 6.5 ลงในบล็อกที่เลือก และกำหนดให้ตำแหน่งที่เติมแทนด้วย T_{index}

4) เติมข้อความสั่งที่เหลือลงในบล็อกที่เลือกโดยใช้วิธีดังหัวข้อ 6.4 ถ้ามีฟังก์ชั่น T ให้ $B = T_{index}$ นอกนั้น ให้ $B = \infty$

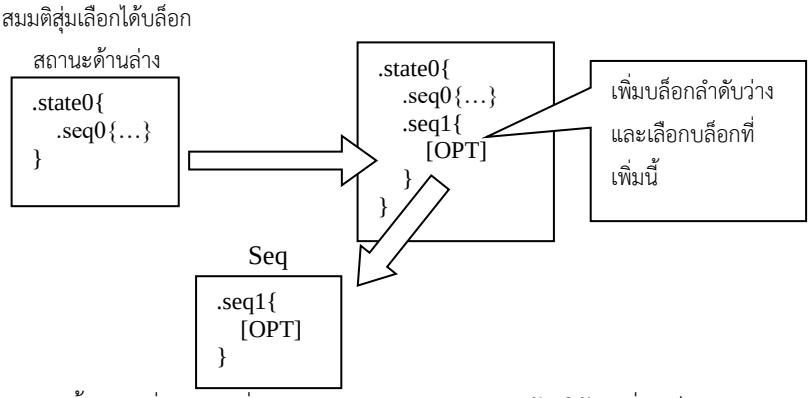
รูปที่ 37 ถึงรูปที่ 41 แสดงตัวอย่างการเติมความสัมพันธ์ซึ่งเป็นลำดับข้อความสั่งที่มี 2 ข้อความสั่งลงในภาคีที่มีบล็อกลำดับจำนวน 4 บล็อก



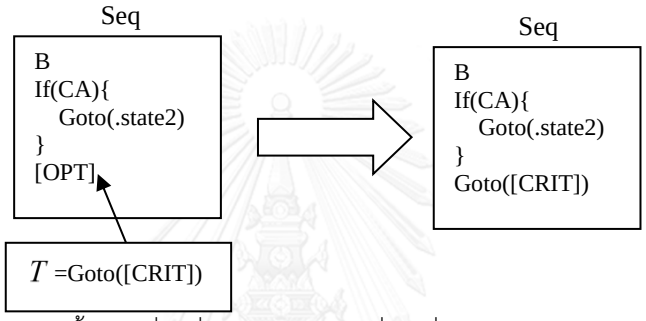
รูปที่ 37 แสดงขั้นตอนที่ 1 ซึ่งเป็นการลบข้อความสั่งที่เป็นฟังก์ชั่นเปลี่ยนสถานะออกจากความสัมพันธ์



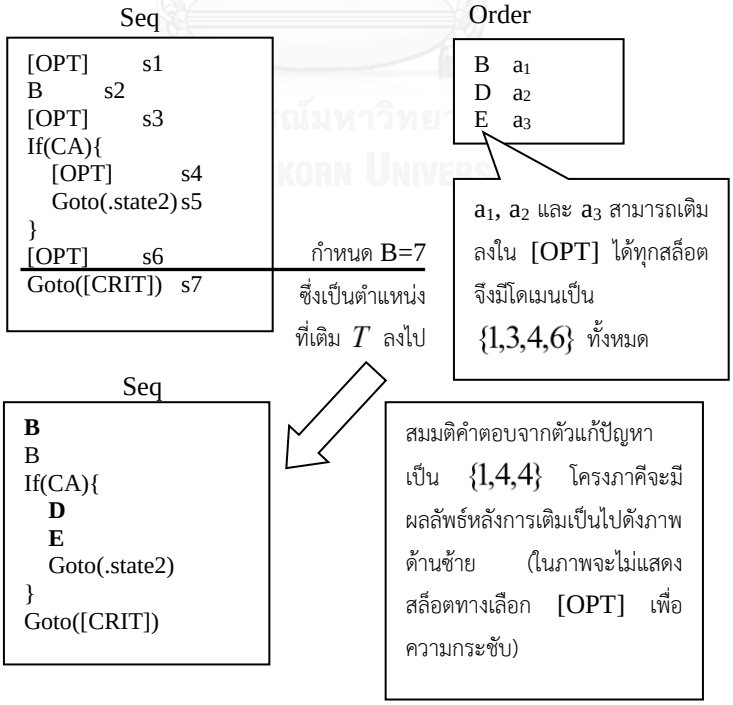
รูปที่ 38 แสดงขั้นตอนที่ 2 ซึ่งเป็นการเลือกบล็อกลำดับที่จะเติมความสัมพันธ์ลงไป



รูปที่ 39 แสดงขั้นตอนที่ 2 กรณีที่ผลการคัดกรองว่างเปล่าและต้องใช้วิธีเพิ่มบล็อกลำดับว่างแทน



รูปที่ 40 แสดงขั้นตอนที่ 3 ซึ่งเป็นการเติมฟังก์ชั่นเปลี่ยนสถานะ T ในบล็อกที่เลือก



รูปที่ 41 แสดงขั้นตอนที่ 4 ซึ่งเป็นการเติมความสัมพันธ์ด้วยขั้นตอนวิธีดังหัวข้อ 6.4

6.10 การเติมความสัมพันธ์รูปแบบฟังก์ชั่นการกระทำข้ามสถานะกรณีฟังก์ชั่นข้ามสถานะเป็น Goto

ความสัมพันธ์ที่ได้จากการทำเหมืองข้อมูลในหัวข้อ 5.3.2 จะอยู่ในรูปของลำดับพฤติกรรมเชื่อม ซึ่งประกอบด้วยลำดับพฤติกรรมก่อนเปลี่ยนสถานะ ลำดับพฤติกรรมหลังเปลี่ยนสถานะ และจำนวนสถานะของภาคี ข้อมูลลำดับทั้งหมดของความสัมพันธ์จะอยู่ในรูปของลำดับของสตริง ลำดับที่มีอยู่จึงต้องถูกแปลงด้วยวิธีการแปลงสตริงในลำดับด้วยวิธีในหัวข้อ 6.3 ก่อน ในที่นี้กำหนดให้ลำดับพฤติกรรมก่อนเปลี่ยนสถานะที่ผ่านการแปลงแล้วเป็น $StartOrder =< a_1, a_2, .., a_m >$ และลำดับพฤติกรรมหลังเปลี่ยนสถานะที่ผ่านการแปลงแล้วเป็น $DestOrder =< b_1, b_2, .., b_n >$ และมีจำนวนสถานะของภาคีเป็น N โดยในกรณีที่ $N < 2$ จะต้องกำหนดค่าให้ $N = 2$ เพื่อให้ภาคีรองรับสถานะต้นทางและปลายทางได้ การเติม $StartOrder$ และ $DestOrder$ ลงในโครงภาคีทำได้ดังนี้

1) กำหนด T เป็นฟังก์ชั่นเปลี่ยนสถานะ Goto ที่มีอากิวเมนท์เป็นสล็อตจำเป็นประเภทตัวระบุ
2) เลือกบล็อกลำดับ $StartBlock$ สำหรับเติมลำดับพฤติกรรมก่อนเปลี่ยนสถานะ โดยบล็อกดังกล่าวจะต้องอยู่ภายใต้ภาคีที่มี N สถานะเป็นอย่างน้อย หากไม่สามารถเลือกได้ จะสุ่มเลือกบล็อกลำดับและเพิ่มบล็อกสถานะลงในภาคีที่ครอบอยู่จนมีจำนวนสถานะครบ 2 สถานะ ขั้นตอนวิธีเป็นดังนี้

Filter skeleton using below conditions and store result in F_{result} :
- Agent condition = There exists at least 1 state
- State condition = There exists at least 1 sequence
- Sequence condition = Being EOB-T block

If(F_{result} is not empty) do:
- Filter F_{result} using below conditions and store result in G_{result} :
  - Agent condition = There exists at least N states
  - State condition = Any
  - Sequence condition = Any
- If(G_{result} is not empty) do:
  - Randomly select 1 sequence block from G_{result} as result
- Else do:
  - Randomly select 1 sequence block Seq from skeleton F_{result} as result
  - Create new state block in agent block that Seq belongs to until there exists 2 states in the agent

Else do:

Filter skeleton using below conditions and store result in G_{result} :

Agent condition = There exists at least N states

State condition = Any

Sequence condition = Any

If(G_{result} is not empty) do:

Randomly select 1 state block from G_{result} and create new sequence block Seq in that state block

Select Seq as result

Else do:

Randomly select 1 state block $State$ from skeleton and

Create new state block in agent block that $State$ belongs to until there exists N states in the agent

Create new sequence block in $State$ and select it as result

3) ลบทุกข้อความสั่งใน $DestOrder$ ที่เป็นฟังก์ชั่นเปลี่ยนสถานะ และบันทึกฟังก์ชั่นสุดท้ายที่ลบไว้เรียกว่า T_{Dest}

4) เลือกบล็อกลำดับ $DestBlock$ สำหรับเติมลำดับพฤติกรรมหลังเปลี่ยนสถานะ ซึ่งต้องเป็นบล็อกลำดับอยู่ภายใต้คนละบล็อกสถานะกับ $StartBlock$ ขั้นตอนวิธีเป็นดังนี้

If(T_{Dest} exists) do:

Filter skeleton using below conditions and store result in F_{result} :

Agent condition = Agent that $StartBlock$ belongs to

State condition = State that $StartBlock$ does not belong to

Sequence condition = Being EOS-T_{Dest} block

If(F_{result} is not empty) do:

Randomly select 1 sequence block from F_{result} as result

Else do:

Filter skeleton using below conditions and store result in G_{result} :

Agent condition = Agent that $StartBlock$ belongs to

State condition = State that $StartBlock$ does not belong to

Sequence condition = Any

Randomly select 1 sequence block from G_{result} as result

Else do:

Filter skeleton using below conditions and store result in G_{result} :

Agent condition = Agent that $StartBlock$ belongs to

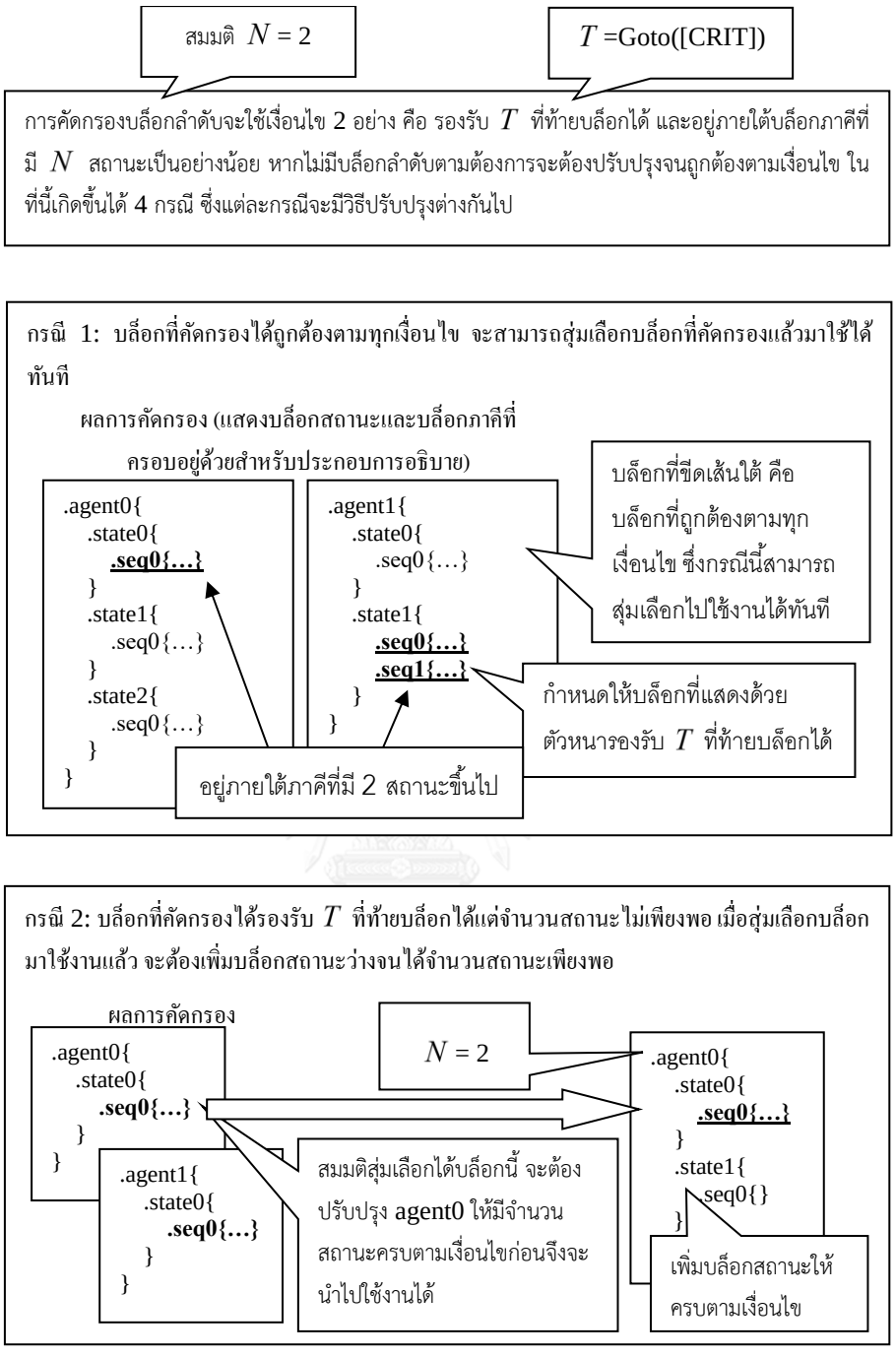
State condition = State that $StartBlock$ does not belong to

Sequence condition = Any

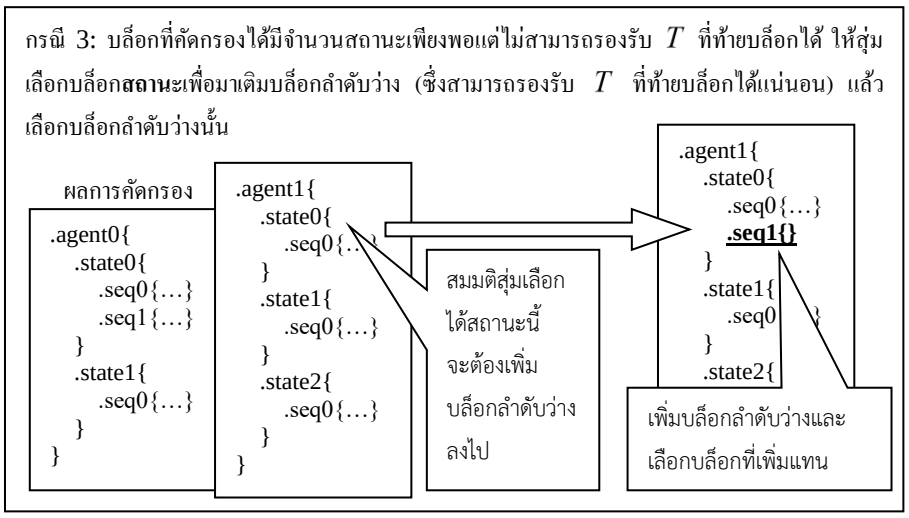
Randomly select 1 sequence block from G_{result} as result

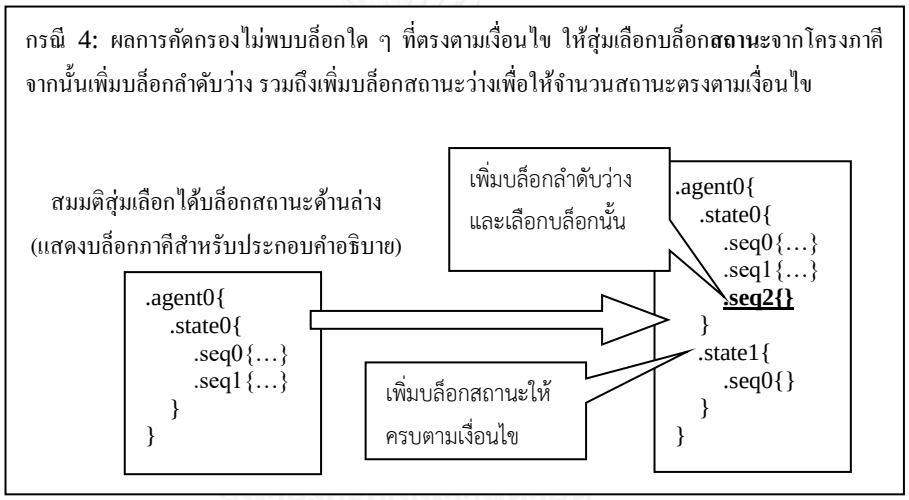
5) ถ้ามีฟังก์ชั่น T_{Dest} ให้ลองเติมฟังก์ชั่น T_{Dest} แบบท้ายลำดับ ดังวิธีในหัวข้อ 6.5 ลงใน $DestBlock$ และกำหนดให้ s_{Tdest} แทนตำแหน่งของสล็อต
6) เติมข้อความสั่งที่ยังเหลืออยู่ใน $DestOrder$ ลงใน $DestBlock$ โดยใช้วิธีดังหัวข้อ 6.4 ถ้าขั้นก่อนหน้าสามารถเติม T_{Dest} ได้ให้ $B = s_{Tdest}$ นอกนั้น ให้ $B = \infty$
7) เติมฟังก์ชั่น T แบบท้ายบล็อก ดังวิธีในหัวข้อ 6.5 ลงใน $StartBlock$ และเติมตัวระบุของบล็อกสถานะที่ครอบ $DestBlock$ อยู่ลงในสล็อตจำเป็นของ T กำหนดให้ตำแหน่งที่เติม T แทนด้วย T_{index}
8) เติม $StartOrder$ ลงใน $StartBlock$ โดยใช้วิธีดังหัวข้อ 6.4 กำหนดให้ $B = T_{index}$

รูปที่ 42 ถึงรูปที่ 45 แสดงรายละเอียดของขั้นตอนวิธีโดยเน้นไปที่การคัดกรองบล็อกเป็นหลักเนื่องจากขั้นตอนการเติมลำดับพฤติกรรมก่อนและหลังเปลี่ยนสถานะมีความคล้ายคลึงกับการเติมลำดับพฤติกรรมในหัวข้อ 6.9

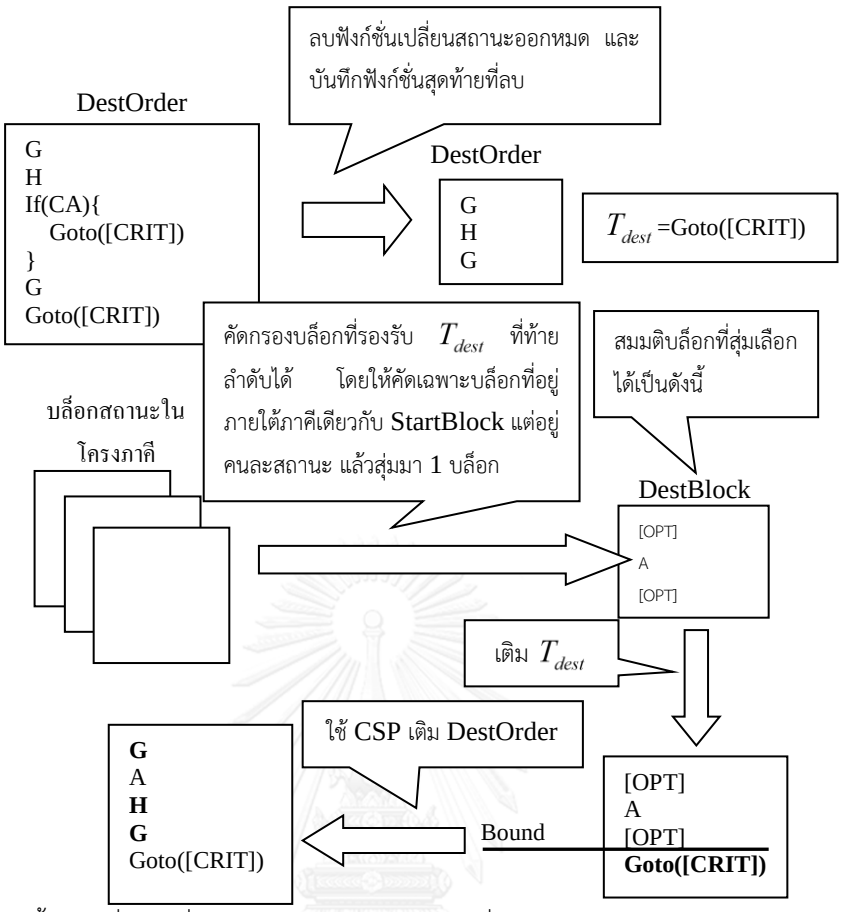


รูปที่ 42 แสดงขั้นตอนที่ 1 – 2 โดยเน้นไปที่ขั้นตอนที่ 2 ซึ่งเป็นการคัดกรองและเลือกบล็อกตามเงื่อนไข

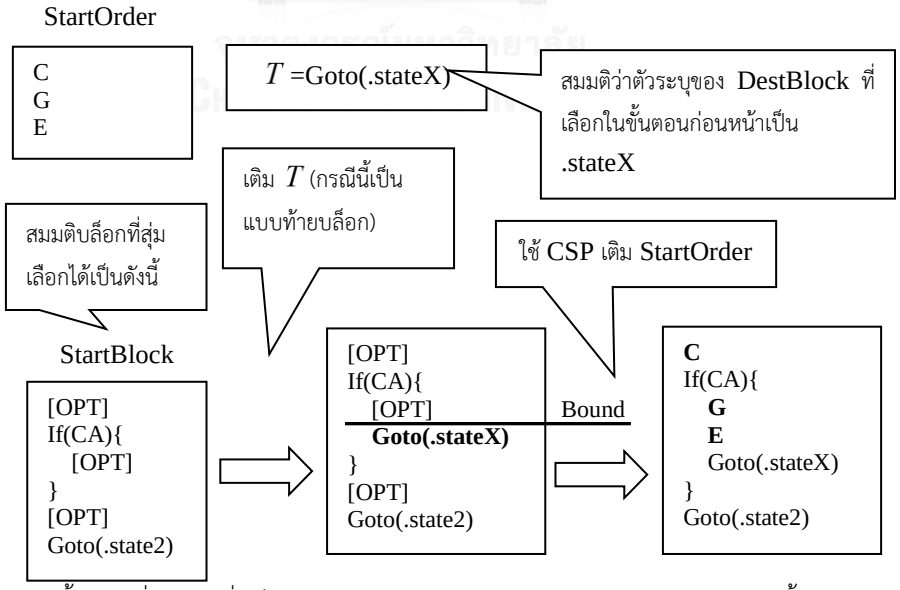




รูปที่ 43 แสดงขั้นตอนที่ 2 (ต่อ) ซึ่งเป็นการคัดกรองและเลือกบล็อกตามเงื่อนไข



รูปที่ 44 แสดงขั้นตอนที่ 3–6 ซึ่งเป็นการเติม *DestOrder* ซึ่งคล้ายคลึงกับการเติมความสัมพันธ์ในหัวข้อ 6.9



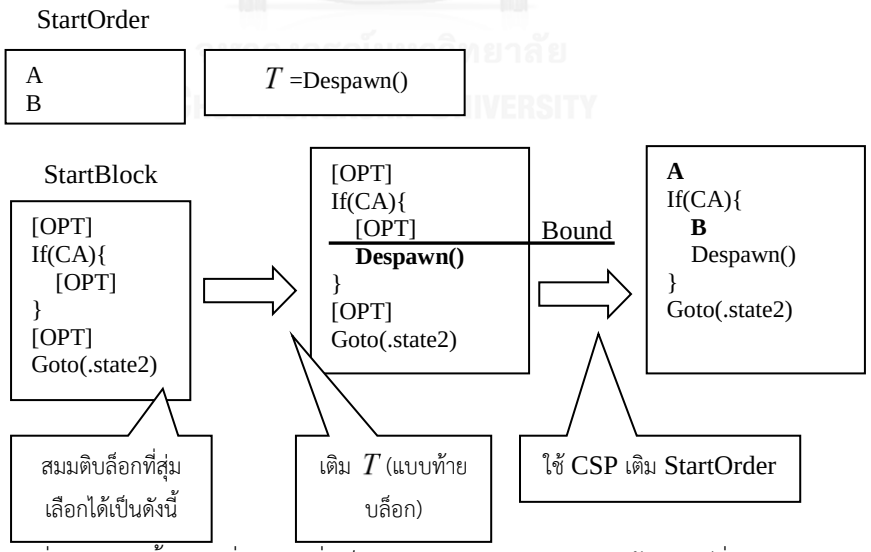
รูปที่ 45 แสดงขั้นตอนที่ 7 – 8 ซึ่งเป็นการเติม *StartOrder* โดยใช้แนวคิดแบบเดียวกับขั้นตอนก่อนหน้า คือ เติมฟังก์ชั่นเปลี่ยนสถานะก่อน แล้วจึงเติมลำดับพฤติกรรมก่อนเปลี่ยนสถานะตามลงไป

6.11 การเติมความสัมพันธ์รูปแบบฟังก์ชั่นการกระทำข้ามสถานะกรณีฟังก์ชั่นข้ามสถานะเป็น Despawn

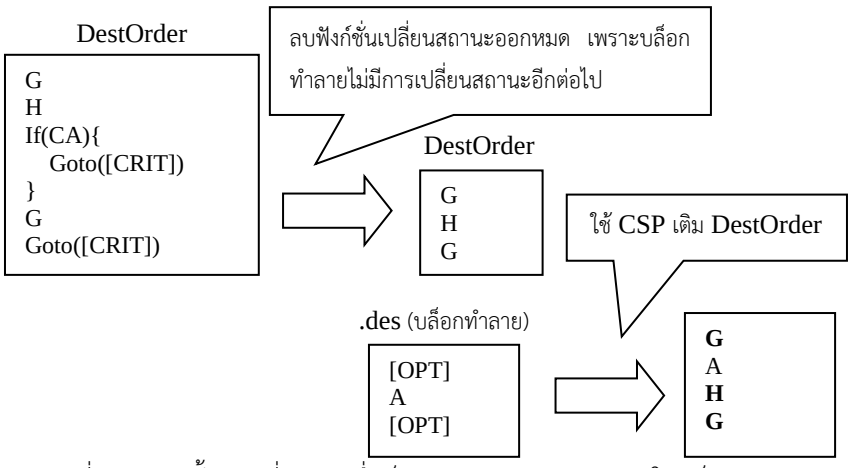
ความสัมพันธ์ที่ได้จากการทำเหมืองข้อมูลจะอยู่ในรูปของลำดับพฤติกรรมเชื่อมเช่นเดียวกับกรณีของ Goto วิธีการเติมความสัมพันธ์จึงมีความคล้ายคลึงกัน แต่เนื่องจากฟังก์ชั่น Despawn จะทำให้สถานะปลายทางเป็นบล็อกทำลายเสมอ ทำให้ไม่จำเป็นต้องสุ่มเลือกบล็อกลำดับให้กับลำดับพฤติกรรมหลังเปลี่ยนสถานะ ในที่นี้กำหนดให้ลำดับพฤติกรรมก่อนเปลี่ยนสถานะที่ผ่านการแปลงแล้วเป็น $StartOrder =< a_1, a_2, .., a_m >$ และลำดับพฤติกรรมหลังเปลี่ยนสถานะที่ผ่านการแปลงแล้วเป็น $DestOrder =< b_1, b_2, .., b_n >$ การเติม $StartOrder$ และ $DestOrder$ ลงในโครงภาคีทำได้ดังนี้

1) กำหนด T เป็นฟังก์ชั่นเปลี่ยนสถานะ Despawn
2) ใช้วิธีการเดียวกับขั้นตอนที่ 2 ของหัวข้อ 6.10 เพื่อเลือกบล็อกลำดับ $StartBlock$ สำหรับลำดับพฤติกรรมก่อนเปลี่ยนสถานะ
3) เติมฟังก์ชั่น T แบบท้ายบล็อก โดยใช้วิธีดังหัวข้อ 6.5 ลงใน $StartBlock$ กำหนดให้ตำแหน่งที่เติม T แทนด้วย T_{index}
4) เติม $StartOrder$ ลงใน $StartBlock$ โดยใช้วิธีดังหัวข้อ 6.4 กำหนดให้ $B = T_{index}$
5) ลบทุกข้อความสั่งใน $DestOrder$ ที่เป็นฟังก์ชั่นเปลี่ยนสถานะ เนื่องจากพฤติกรรมในบล็อกทำลายจะเกิดเมื่อภาคีถูกนำออกจากโลกของเกม ซึ่งไม่มีการเปลี่ยนสถานะอีกต่อไป
6) เติมข้อความสั่งใน $DestOrder$ ลงในบล็อกทำลายโดยใช้วิธีดังหัวข้อ 6.4 กำหนดให้ $B = \infty$

รูปที่ *46* และรูปที่ *47* แสดงขั้นตอนการเติมความสัมพันธ์โดยสังเขป



รูปที่ 46 แสดงขั้นตอนที่ 1 – 4 ซึ่งเป็นการเติม $StartOrder$ พร้อมฟังก์ชั่น Despawn



รูปที่ 47 แสดงขั้นตอนที่ 5 – 6 ซึ่งเป็นการเติม $DestOrder$ ในบล็อกทำลาย

6.12 การเติมความสัมพันธ์รูปแบบฟังก์ชั่นการกระทำคู่ขนาน

ความสัมพันธ์ที่ได้จากการทำเหมืองข้อมูลสำหรับฟังก์ชั่นการกระทำคู่ขนานในหัวข้อ 5.4.1 จะอยู่ในรูปของกราฟซึ่งจุดยอดมีชื่อเป็นสตริงแทนฟังก์ชั่นการกระทำ และมีจุดยอดพิเศษสำหรับแยกจุดยอดแทนฟังก์ชั่นการกระทำที่มาจากลำดับพฤติกรรมที่ต่างกันออกจากกัน ข้อมูลกราฟดังกล่าวนี้สามารถแปลงให้อยู่ในรูปของรายการลำดับข้อความสั่ง

วิธีการแปลงลำดับพฤติกรรมคู่ขนานตามหัวข้อ 5.4.1 สามารถทำย้อนกลับเพื่อแปลงกราฟให้อยู่ในรูปของรายการของพฤติกรรมไม่มีลำดับ ข้อมูลที่แปลงกลับจากกราฟจะไม่มีลำดับเนื่องจากการแปลงลำดับพฤติกรรมคู่ขนานเป็นกราฟไม่มีการเก็บรักษาความสัมพันธ์เชิงลำดับของพฤติกรรมเอาไว้ ดังนั้น สำหรับพฤติกรรมในแต่ละรายการ ลำดับของฟังก์ชั่นการกระทำจะเป็นเช่นไรก็ได้ ในที่นี้จึงทำการกำหนดลำดับอย่างสุ่มให้กับฟังก์ชั่นการกระทำภายในแต่ละพฤติกรรมไม่มีลำดับ เพื่อแปลงข้อมูลให้อยู่ในรูปของรายการของลำดับพฤติกรรม และเนื่องจากฟังก์ชั่นการกระทำอยู่ในรูปของสตริง ในที่นี้จึงสามารถแปลงลำดับพฤติกรรม (ซึ่งในที่นี้เป็นลำดับของสตริง) เป็นลำดับข้อความสั่งด้วยวิธีการแปลงสตริงในลำดับด้วยวิธีดังหัวข้อ 6.3 ผลลัพธ์สุดท้ายของการแปลงจึงได้เป็นรายการของลำดับข้อความสั่ง

ในที่นี้กำหนดให้ความสัมพันธ์ที่ผ่านการเปลี่ยนรูปแล้วเป็นรายการของลำดับข้อความสั่งดังนี้

$ParallelList = \{P_1, P_2, .., P_n\}$ เมื่อ $P_i, \quad 1 \le i \le n$ คือ ลำดับข้อความสั่ง การเติม $ParallelList$ ลงในโครงภาคีสามารถทำได้ดังนี้

1) สำหรับแต่ละลำดับข้อความสั่ง P_i ใน $ParallelList$ ให้ลบทุกข้อความสั่งที่เป็นฟังก์ชั่นเปลี่ยนสถานะ และบันทึกฟังก์ชั่นสุดท้ายที่ลบไว้ (ถ้ามี) เรียกว่า $T_i, \quad 1 \le i \le n$ ฟังก์ชั่นเปลี่ยนสถานะทั้งหมดที่บันทึกไว้ให้รวบรวมไว้ในรายการ T_{list}
2) เตรียมข้อมูลการส่งว่าง $P_{map} = \{\}$ ซึ่งใช้ระบุว่าแต่ละลำดับข้อความสั่งใน $ParallelList$ จะต้องนำไปเติมในบล็อกลำดับใด โดยอยู่ในรูป $P_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าลำดับข้อความสั่ง k_i ใน $ParallelList$ จะต้องนำไปเติมในบล็อกลำดับ v_i ในโครงภาคี

3) ถ้า T_{list} ไม่เป็นรายการว่าง ให้เลือกบล็อกสถานะสำหรับเติม $ParallelList$ โดยจะต้องรองรับฟังก์ชั่นเปลี่ยนสถานะใน T_{list} ให้ได้มากที่สุดเท่าที่เป็นไปได้ หากไม่พอจะต้องสร้างบล็อกลำดับเพิ่มจนเพียงพอกับฟังก์ชั่นเปลี่ยนสถานะที่มีทั้งหมด โดยการเลือกบล็อกจะให้ความสำคัญกับบล็อกที่มีจำนวนบล็อกลำดับพอจะรองรับลำดับทั้งหมดใน $ParallelList$ นอกจากการเลือกบล็อกแล้วขั้นตอนนี้จะต้องสร้างข้อมูลการส่งเพื่อระบุว่าแต่ละฟังก์ชั่นข้ามสถานะใน T_{list} จะต้องนำไปเติมในบล็อกลำดับใด โดยอยู่ในรูป $T_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าฟังก์ชั่นเปลี่ยนสถานะ k_i ใน T_{list} จะต้องนำไปเติมในบล็อกลำดับ v_i ในโครงภาคี ขั้นตอนทั้งหมดเป็นดังนี้

Filter skeleton using below conditions and store result in F_{result} :

Agent condition = There exists at least 1 state

State condition = There exists at least $|ParallelList|$ sequences

Sequence condition = Any

If(F_{result} is empty) do: Treat the skeleton as F_{result}

$Alloc_{result} = \phi, \min Cost = \infty$

For(State S in F_{result}) do:

Calculate T_{list} best allocation for S using algorithm described in 6.7 and store cost in $Cost$ and all allocation maps as a set $AllocationMapSet$

(Each element in the set is a map storing (transition from T_{list} => sequence from S))

If($Cost < \min Cost$) do: $Alloc_{result} = \phi, \min Cost = Cost$

If($Cost \leq \min Cost$) do:

For($AllocMap$ in $AllocationMapSet$) do:

$Alloc_{result} \leftarrow \{State = S, Map = AllocMap\}$

If($Alloc_{result}$ is not empty) do:

Randomly select 1 allocation $Alloc$ from $Alloc_{result}$

$State = Alloc.State$

$T_{map} = Alloc.Map$

Select $State$ as result

Else do:

Randomly select 1 result state $State$ from F_{result} as result

$T_{map} = \{\}$

For(Transition T_i in T_{list}) do:

If(T_i does not exist in T_{map}) do:

Create new sequence block Seq in $State$

$T_{map} \leftarrow (T_i => Seq)$

$P_{map} \leftarrow (P_i => Seq)$

Else do:

$P_{map} \leftarrow (P_i => T_{map}[T_i])$

4) ถ้า T_{list} เป็นรายการว่าง ให้สุ่มเลือกบล็อกสถานะโดยให้ความสำคัญกับบล็อกสถานะที่รองรับทุกลำดับข้อความสั่งใน $ParallelList$ ได้ก่อน ดังนี้

Filter skeleton using below conditions and store result in F_{result} :

Agent condition = There exists at least 1 state

State condition = There exists at least $|ParallelList|$ states

Sequence condition = Any

If(F_{result} is empty) do: Treat the skeleton as F_{result}

Randomly select 1 result state $State$ from F_{result} as result

5) ถ้าบล็อกสถานะที่เลือกมีจำนวนบล็อกลำดับไม่เพียงพอกับจำนวนลำดับข้อความสั่งใน $ParallelList$ ให้สร้างบล็อกลำดับใหม่และเติมลงในบล็อกสถานะดังกล่าว

6) สุ่มเลือกบล็อกลำดับที่จะเติมให้กับแต่ละลำดับข้อความสั่งใน $ParallelList$ ที่ยังไม่ได้กำหนดบล็อกเป้าหมาย ดังนี้

For(Sequence P_i in $ParallelList$) do:

If(P_i does not exist in P_{map}) do:

Randomly select 1 sequence block Seq in selected state $State$ that does not exist in P_{map}

$P_{map} \leftarrow (P_i => Seq)$

7) ถ้า T_{list} ไม่เป็นรายการว่าง ให้เติมฟังก์ชั่นเปลี่ยนสถานะตามข้อมูลการส่งใน T_{map} โดยใช้วิธีเติมฟังก์ชั่นแบบท้ายลำดับ ดังหัวข้อ 6.5 โดยต้องทำการเก็บข้อมูลตำแหน่งที่ทำการเติมไว้ในรูปของการส่ง $P_{bound} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าการเติมลำดับข้อความสั่งลงในบล็อกลำดับ k_i จะต้องไม่เกินตำแหน่ง v_i รายละเอียดของขั้นตอนเป็นดังนี้

For(Transition T_i in T_{list}) do:

Insert T_i , at the end of sequence, into sequence block $T_{map}[T_i]$ and retrieve the inserting index $Tindex_i$

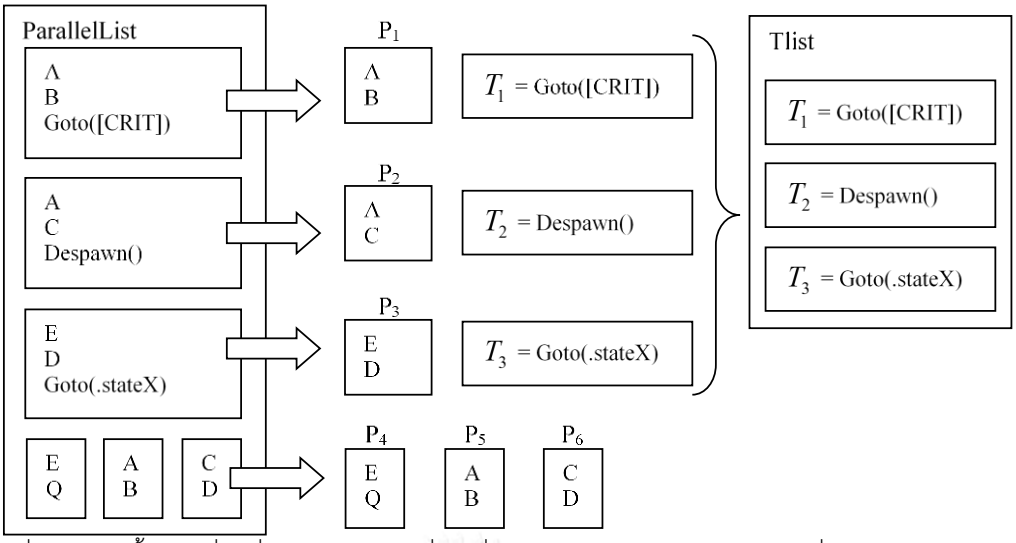
$P_{bound} \leftarrow (T_{map}[T_i] => Tindex_i)$

8) เติมข้อความสั่งทั้งหมดใน $ParallelList$ ลงในบล็อกลำดับที่กำหนดด้วย P_{map} โดยใช้วิธีดังหัวข้อ 6.4 และใช้ P_{bound} ในการกำหนดค่า B รายละเอียดของขั้นตอนเป็นดังนี้

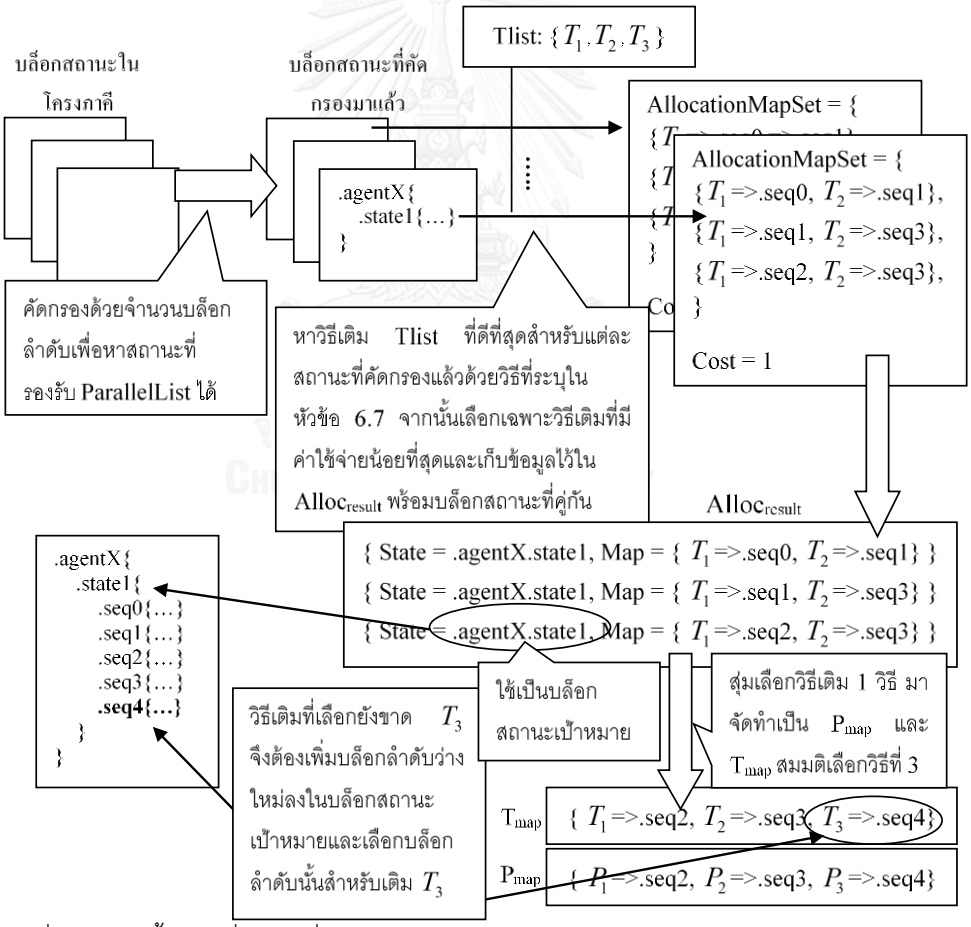
For(Sequence P_i in $ParallelList$) do:

Insert P_i into $P_{map}[P_i]$ with bound $B = P_{bound}[P_i]$

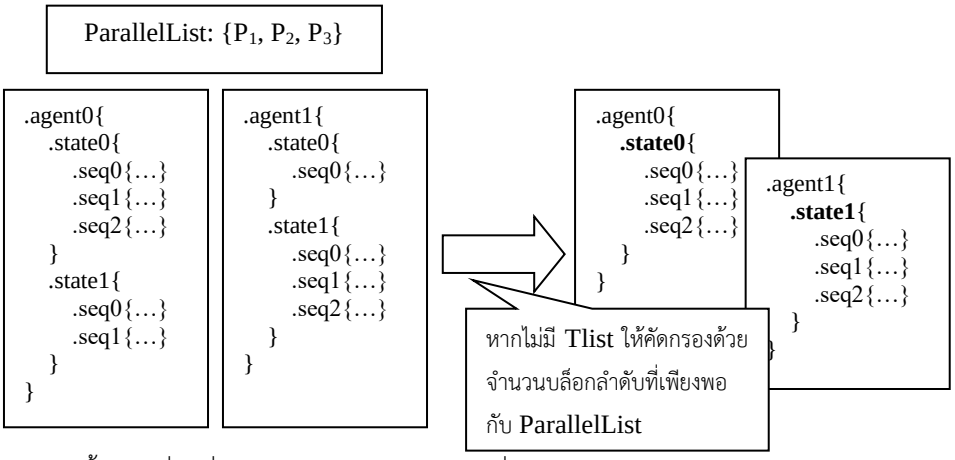
รูปที่ 48 ถึงรูปที่ 53 แสดงขั้นตอนการเติมความสัมพันธ์ที่มี 6 ลำดับข้อความสั่งทำงานคู่ขนานกันอยู่



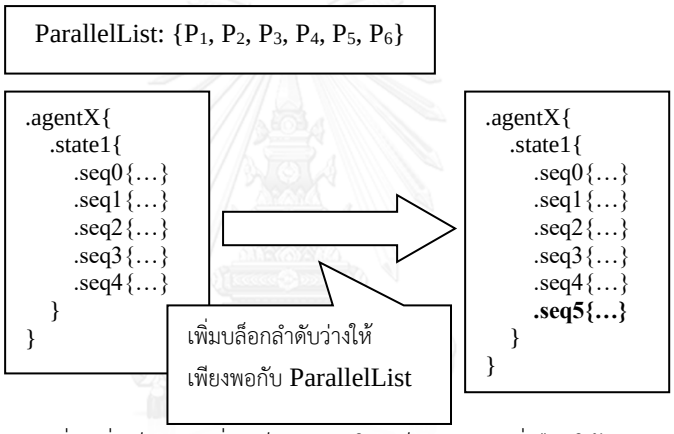
รูปที่ 48 แสดงขั้นตอนที่ 1 ซึ่งเป็นการนำฟังก์ชั่นเปลี่ยนสถานะออกจากลำดับข้อความสั่งและจัดทำเป็น T_{list}



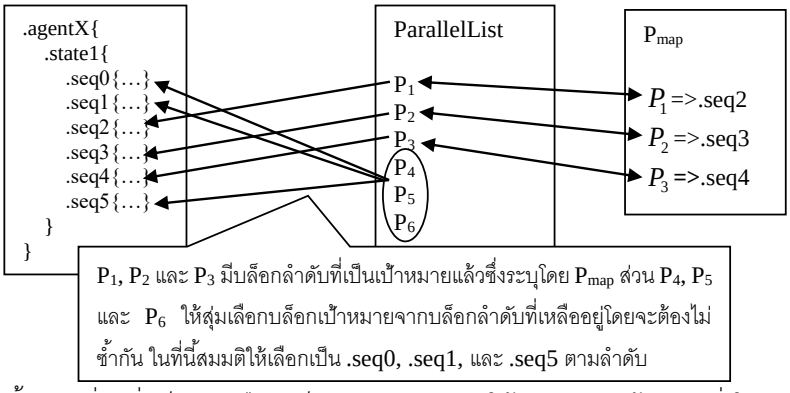
รูปที่ 49 แสดงขั้นตอนที่ 2 - 3 ซึ่งเป็นการเลือกบล็อกสถานะเป้าหมายพร้อมสร้างและเติมเต็ม P_{map}



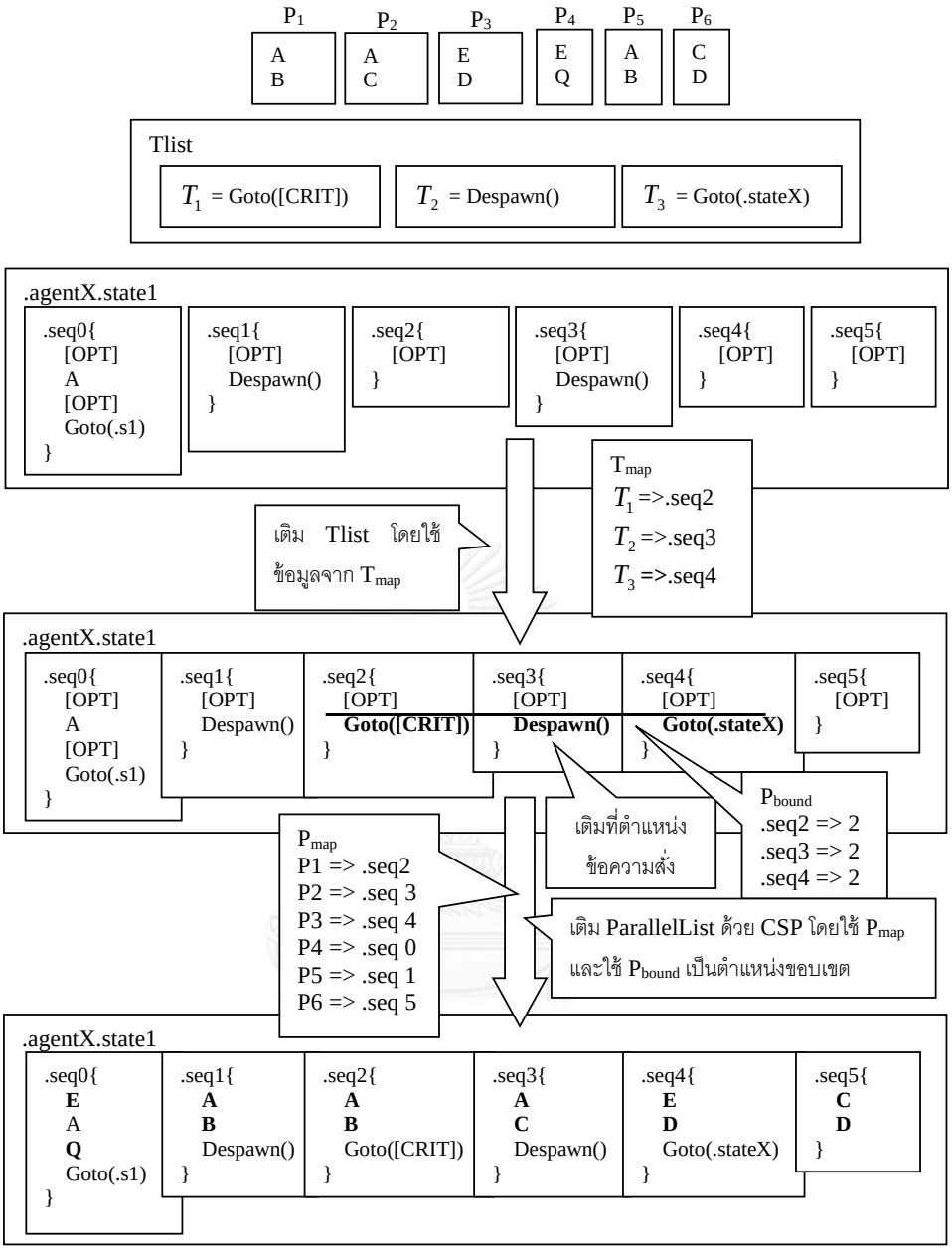
รูปที่ 50 แสดงขั้นตอนที่ 4 ซึ่งเป็นการคัดกรองบล็อกกรณีที่ T_{list} ว่างเปล่า (*ParallelList* และโครงภาคีในรูปนี้ใช้เป็นตัวอย่างสำหรับรูปนี้เท่านั้นโดยไม่เกี่ยวข้องกับรูปของขั้นตอนอื่น)



รูปที่ 51 แสดงขั้นตอนที่ 5 ซึ่งเป็นการเพิ่มบล็อกลำดับในบล็อกสถานะที่เลือกให้เพียงพอกับจำนวนลำดับข้อความสั่งใน *ParallelList*



รูปที่ 52 แสดงขั้นตอนที่ 6 ซึ่งเป็นการเลือกบล็อกลำดับเป้าหมายให้แต่ละลำดับข้อความสั่งใน *ParallelList*



รูปที่ 53 แสดงขั้นตอนที่ 7 – 8 ซึ่งเป็นการเติม T_{list} และ *ParallelList* ลงในบล็อกที่เป็นเป้าหมาย

6.13 การเติมความสัมพันธ์รูปแบบฟังก์ชั่นการกระทำคู่ขนานข้ามภาคี

ความสัมพันธ์ที่ได้จากการทำเหมืองข้อมูลสำหรับฟังก์ชั่นการกระทำคู่ขนานข้ามภาคีในหัวข้อ 5.4.2 จะอยู่ในรูปของกราฟที่คล้ายคลึงกับกรณีของฟังก์ชั่นการกระทำคู่ขนาน จุดที่แตกต่างคือ กราฟผลลัพธ์จะมีการแยกพฤติกรรมที่มาจากคนละภาคีออกจากกันด้วยป้ายชื่อของเส้นเชื่อม รวมถึงมีจุดยอดพิเศษเป็นข้อมูลเสริมเพื่อบอกจำนวนลำดับพฤติกรรมของพฤติกรรมคู่ขนานลูก ซึ่งเป็นพฤติกรรมที่ได้จากสถานะเริ่มต้นของภาคีที่ถูกสร้างด้วยภาคีเจ้าของ

การแปลงความสัมพันธ์แบบฟังก์ชั่นการกระทำคู่ขนานข้ามภาคี สามารถทำย้อนกลับวิธีการที่ระบุในหัวข้อ 5.4.2 ในลักษณะเดียวกับหัวข้อ 6.12 เพื่อจัดทำเป็นรายการของลำดับข้อความสั่งสำหรับภาคีเจ้าของและภาคีลูก รวมถึงจำนวนบล็อกลำดับขั้นต่ำในสถานะเริ่มต้นของภาคีลูก

ในที่นี้กำหนดให้รายการของลำดับของข้อความสั่งของภาคีเจ้าของ $OwnerList = \{O_1, O_2, .., O_n\}$ และรายการของลำดับของข้อความสั่งของภาคีลูก $ChildList = \{C_1, C_2, .., C_m\}$ เมื่อ $O_i,\ 1 \le i \le n$ และ $C_i,\ 1 \le i \le m$ คือ ลำดับข้อความสั่งสำหรับภาคีเจ้าของและภาคีลูก ตามลำดับ และ M คือ จำนวนบล็อกลำดับขั้นต่ำในบล็อกสถานะที่สนใจ (ตามนิยามแล้ว ค่า M คือจำนวนบล็อกลำดับขั้นต่ำในสถานะเริ่มต้นของภาคีลูก แต่เพื่อป้องกันไม่ให้ความสัมพันธ์ถูกเติมลงในสถานะเริ่มต้นทั้งหมด ในที่นี้จึงให้ความสนใจทุกบล็อกสถานะ) ในกรณีที่ $M < |ChildList|$ จะต้องกำหนดให้ $M = |ChildList|$ เพื่อให้รองรับได้ครบทุกลำดับข้อความสั่ง การเติม $OwnerList$ และ $ChildList$ ลงในโครงภาคีสามารถทำได้ด้วยวิธีที่คล้ายกัน ในที่นี้จะเริ่มอธิบายจากการเลือกบล็อกสำหรับเติม $ChildList$ แล้วจึงอธิบายวิธีเลือกบล็อกสำหรับเติม $OwnerList$ และตอนท้ายจึงเป็นการเติมรายการลำดับทั้งสองลงในบล็อกเป้าหมาย

1) เตรียมข้อมูลการส่งว่าง $C_{map} = \{\}$ ซึ่งใช้ระบุว่าแต่ละลำดับข้อความสั่งใน $ChildList$ จะต้องนำไปเติมในบล็อกลำดับใด โดยอยู่ในรูป $C_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าลำดับข้อความสั่ง k_i ใน $ChildList$ จะต้องนำไปเติมในบล็อกลำดับ v_i ในโครงภาคี
2) เตรียมข้อมูลการส่งว่าง $CT_{map} = \{\}$ ซึ่งใช้ระบุว่าแต่ละฟังก์ชั่นเปลี่ยนสถานะที่ปรากฏในลำดับข้อความสั่งใน $ChildList$ จะต้องนำไปเติมในบล็อกลำดับใด โดยอยู่ในรูป $CT_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าฟังก์ชั่นเปลี่ยนสถานะ k_i จะต้องนำไปเติมในบล็อกลำดับ v_i ในโครงภาคี
3) เติมเต็ม C_{map} และ CT_{map} โดยทำการคัดบล็อกสถานะที่มีจำนวนลำดับไม่น้อยกว่า M และหาวิธีการเติม $ChildList$ ที่ทำให้ฟังก์ชั่น Spawn และบล็อกลำดับใหม่ (ซึ่งอาจเกิดได้ในกรณีที่ลำดับข้อความสั่งที่ต้องการเติมมีฟังก์ชั่นเปลี่ยนสถานะ แต่ไม่มีบล็อกลำดับที่รองรับได้) เกิดน้อยที่สุด พร้อมกันนั้น หากลำดับข้อความสั่งใดใน $ChildList$ มีฟังก์ชั่นเปลี่ยนสถานะ จะต้องพยายามเลือกบล็อกลำดับที่รองรับฟังก์ชั่นดังกล่าวที่ท้ายบล็อกได้ด้วย และในกรณีที่ไม่มีบล็อกลำดับที่รองรับฟังก์ชั่นเปลี่ยนสถานะนี้ได้ จะต้องมีการเติมบล็อกลำดับใหม่เพิ่มลงไป การพิจารณาเรื่องของฟังก์ชั่น Spawn และฟังก์ชั่นเปลี่ยนสถานะจะทำไปพร้อมกันโดยให้ความสำคัญกับการลดจำนวนฟังก์ชั่น Spawn

มากกว่าการลดจำนวนบล็อกลำดับที่อาจเพิ่มขึ้นจากกรณีไม่มีบล็อกที่รองรับฟังก์ชั่นเปลี่ยนสถานะ ขั้นตอนทั้งหมดเป็นดังนี้

//**Substep 1**: Find all best spawn-matched allocation

Filter skeleton using below conditions and store result in F_{result} :

Agent condition = There exists at least 1 state

State condition = There exists at least M sequences

Sequence condition = Any

If(F_{result} is empty) do: Treat the skeleton as F_{result}

Remove main agent from F_{result} - - - - - (A)

If(F_{result} is empty) do: Create new agent and add it to F_{result}

$SpawnMatch_{result} = \phi, \min Cost = \infty$

For(State S in F_{result}) do:

Calculate $ChildList$ best allocation for S using algorithm described in 6.8 and store cost in $Cost$ and all allocation maps as a set $AllocationMapSet$ (Each element in the set is a map storing (sequence from $ChildList$ => sequence from S))

If($Cost < \min Cost$) do: $SpawnMatch_{result} = \phi, \min Cost = Cost$

If($Cost \leq \min Cost$) do:

For($AllocMap$ in $AllocationMapSet$) do:

$SpawnMatch_{result} \leftarrow \{State = S, Map = AllocMap\}$

If($SpawnMatch_{result}$ is empty) do:

For(State S in F_{result}) do: $SpawnMatch_{result} \leftarrow \{State = S, Map = \{\}\}$

//**Substep 2**: For each spawn-matched allocation, find all corresponding best transition allocations

$TAlloc_{result} = \phi, \min Cost = \infty$

For(Entry S_m in $SpawnMatch_{result}$) do:

Create a copy of $S_m.State$ as S' and remove all sequences in S' that exist in $S_m.Map$'s value

Create a copy of $ChildList$ as C' and remove all sequences in C' that exist in $S_m.Map$

$T_{list} = \phi$

For(Sequence C'_i in C') do:

Remove all transitions in C'_i and add the last one found to T_{list}

If(T_{list} is empty) do:

If $\min Cost > 0$ do: $TAlloc_{result} = \phi, \min Cost = 0$

Else do:

Calculate T_{list} best allocation for S' using algorithm described in 6.7 and store cost in $TCost$ and all allocation maps as a set $TAllocMap$ (Each element in the set is a map storing (transition from T_{list} => sequence from S'))

If($TCost < \min Cost$) do: $TAlloc_{result} = \phi, \min Cost = TCost$

If($TCost \leq \min Cost$) do:

For($AllocMap$ in $TAllocMap$) do:

$TAlloc_{result} \leftarrow \{State = S, Map = AllocMap\}$

//**Substep 3**: Select map and generating data for C_{map} and CT_{map}

Randomly select 1 entry from $SpawnMatch_{result}$ as S_m

Randomly select 1 entry from $TAlloc_{result}$ as T_m that satisfies $S_m.State = T_m.State$

If(there is no T_m) do: Let $T_m = \{\}$

//Handle mapping for sequence with Spawn or transition first

For(Sequence C_i in $ChildList$) do:

Remove all transitions in C_i and save the last one as t_i

If(C_i exists in $S_m.Map$) do: $C_{map} \leftarrow (C_i => S_m.Map[C_i])$

Else if(t_i exists in $T_m.Map$) do:

$C_{map} \leftarrow (C_i => T_m.Map[t_i])$

$CT_{map} \leftarrow (t_i => T_m.Map[t_i])$

Else if(t_i exists) do:

Create new sequence block Seq and add it to $S_m.State$

$C_{map} \leftarrow (C_i => Seq)$

$CT_{map} \leftarrow (t_i => Seq)$

//**Substep 4**: Fill selected state until it satisfies sequence count and complete C_{map}

Create new sequence and add it to $S_m.State$ until there exists at least M sequences

For(Sequence C_i in $ChildList$) do:

If(C_i does not exist in c) do:

Randomly select 1 sequence block from $S_m.State$ that is not exist in C_{map} as Seq

$C_{map} \leftarrow (C_i => Seq)$

4) จากขั้นตอนที่ 3 จะทำให้รู้ว่าต้องเติม $ChildList$ ไปยังบล็อกสถานะใด ให้สำรวจฟังก์ชั่น Spawn ทั้งหมดที่ปรากฏใน $OwnerList$ และแก้พารามิเตอร์เป็นตัวระบุบล็อกสถานะที่เติม $ChildList$ ลงไป

5) เตรียมข้อมูลการส่งว่าง $O_{map} = \{\}$ และ $OT_{map} = \{\}$ ในลักษณะเดียวกับ C_{map} และ CT_{map} ตามลำดับ แต่ใช้สำหรับ $OwnerList$ แทน

6) เติมเต็ม O_{map} และ OT_{map} ด้วยขั้นตอนแบบเดียวกับการเติมเต็ม C_{map} และ CT_{map} โดยให้เปลี่ยนแปลงขั้นตอนวิธีดังนี้

- โดยแทนที่ $ChildList$ ในขั้นตอนวิธีด้วย $OwnerList$
- เปลี่ยนค่า M เป็น $|OwnerList|$
- เปลี่ยนบรรทัด (A) ให้เป็น “นำภาคีลูกออกจาก F_{result} ” แทน

7) เติม $ChildList$ ลงในโครงภาคีโดยใช้ข้อมูลจาก C_{map} และ CT_{map} โดยเติมฟังก์ชั่นเปลี่ยนสถานะเพื่อหาขอบเขตการเติมสำหรับแต่ละลำดับ แล้วจึงใช้วิธีดังหัวข้อ 6.4 เพื่อเติมลำดับ ขั้นตอนเป็นดังนี้

$Bound = \{\}$

For(Entry $(t_i => Seq)$ in CT_{map}) do:

Insert t_i , at the end of sequence, into sequence block Seq and retrieve the inserting index $Tindex_i$

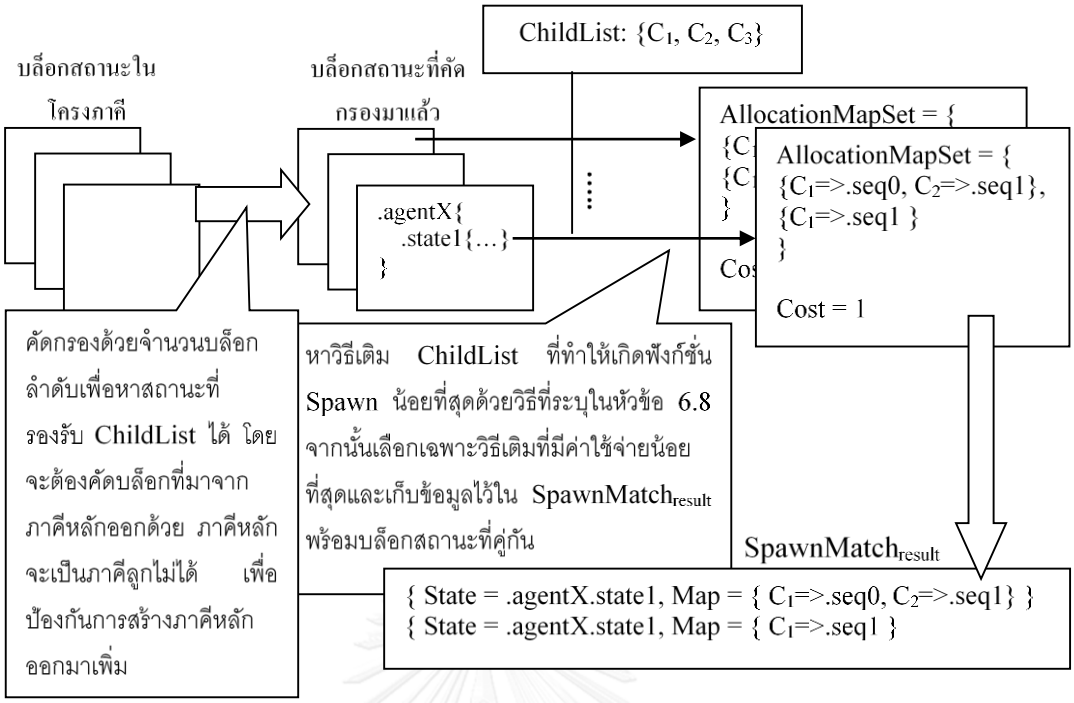
$Bound \leftarrow (Seq => Tindex_i)$

For(Sequence S_i in $ChildList$) do:

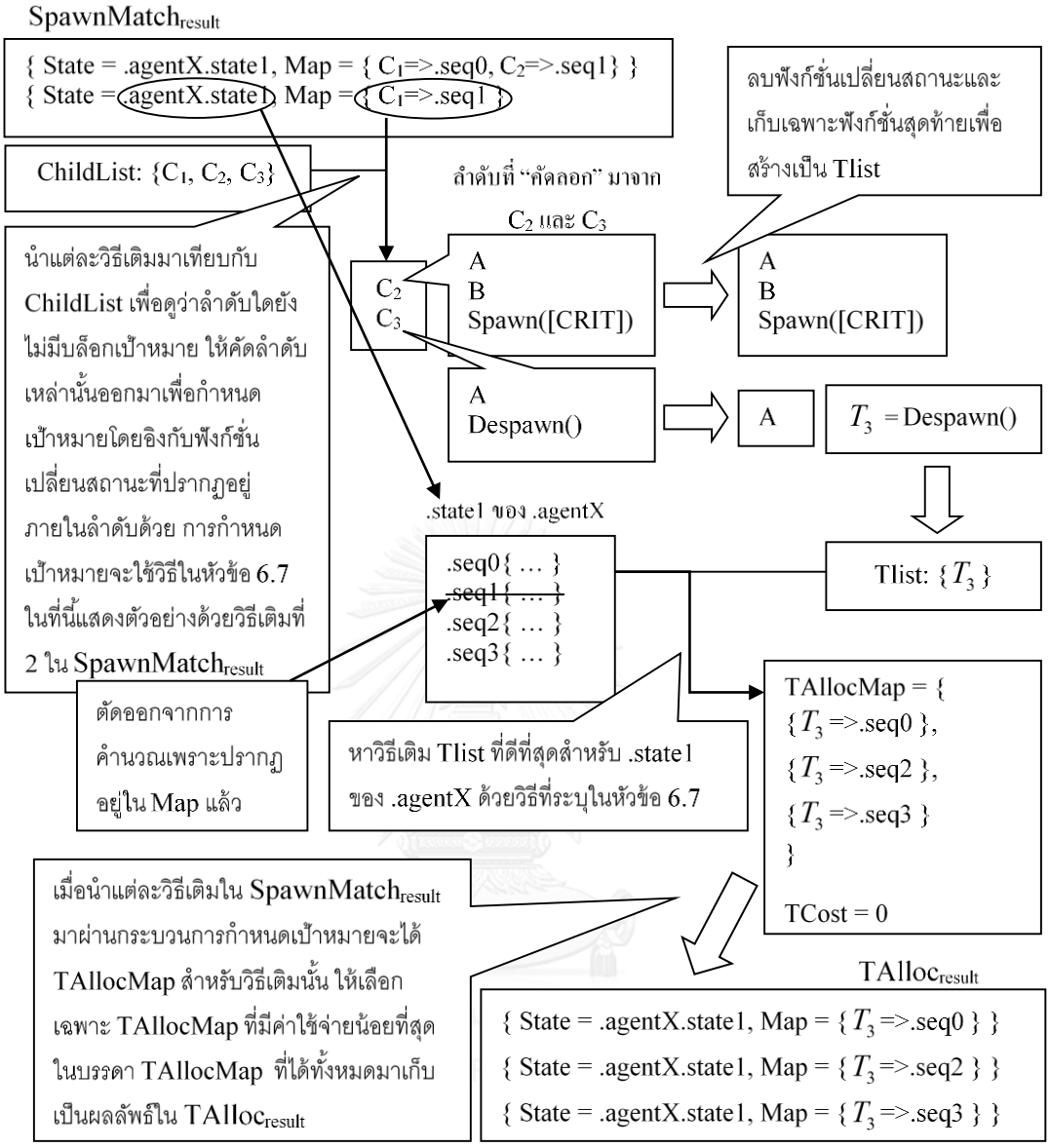
Insert S_i into $C_{map}[S_i]$ with bound $B = Bound[S_i]$

8) เติม $OwnerList$ ลงในโครงภาคีโดยใช้ข้อมูลจาก O_{map} และ OT_{map} ในลักษณะเดียวกับการเติม $ChildList$ โดยแทนที่ $ChildList$, C_{map} และ CT_{map} ในขั้นตอนวิธีด้วย $OwnerList$, O_{map} และ OT_{map} ตามลำดับ

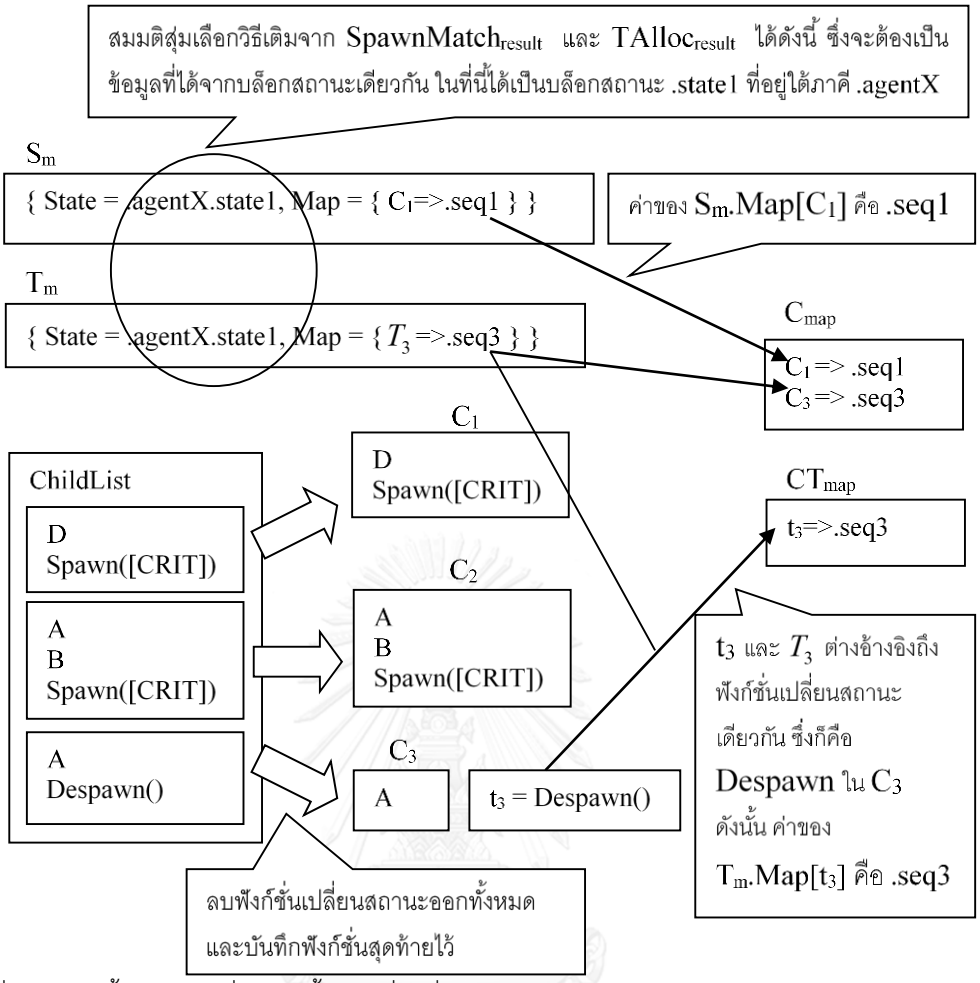
รูปที่ *54* ถึงรูปที่ *60* แสดงขั้นตอนการเติมความสัมพันธ์ที่มีลำดับข้อความสั่งของภาคีลูกจำนวน 3 ลำดับ และลำดับข้อความสั่งของภาคีเจ้าของอีก 3 ลำดับ



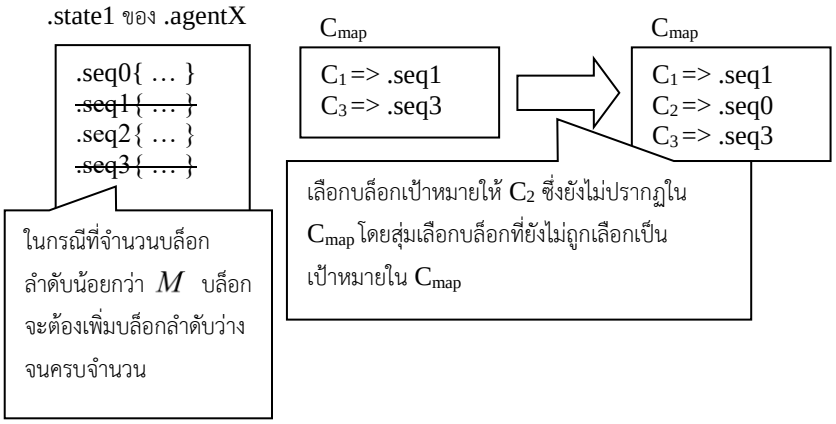
รูปที่ 54 แสดงขั้นตอนย่อย 1 ของขั้นตอนที่ 3 ซึ่งเป็นการหาวิธีเติมที่ทำให้เกิดฟังก์ชั่น Spawn น้อยที่สุด



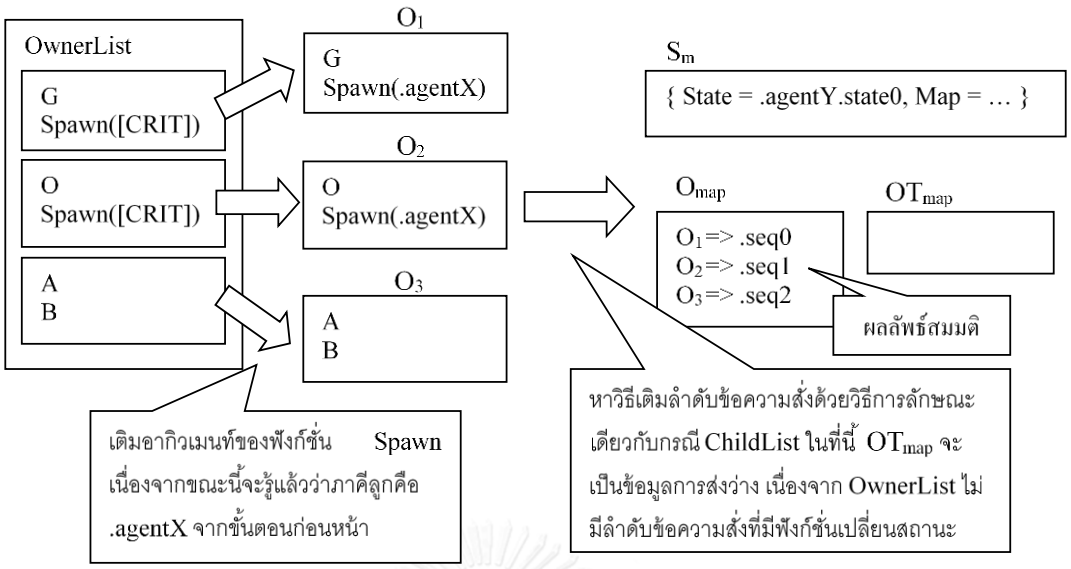
รูปที่ 55 แสดงขั้นตอนย่อยที่ 2 ของขั้นตอนที่ 3 ซึ่งเป็นการหาวิธีเติมที่ดีที่สุดสำหรับลำดับข้อความสั่งที่มีฟังก์ชั่นเปลี่ยนสถานะรวมอยู่ในลำดับ โดยจะพยายามหาวิธีเติมให้เฉพาะลำดับที่ยังไม่มีวิธีเติมจากขั้นตอนย่อยก่อนหน้า



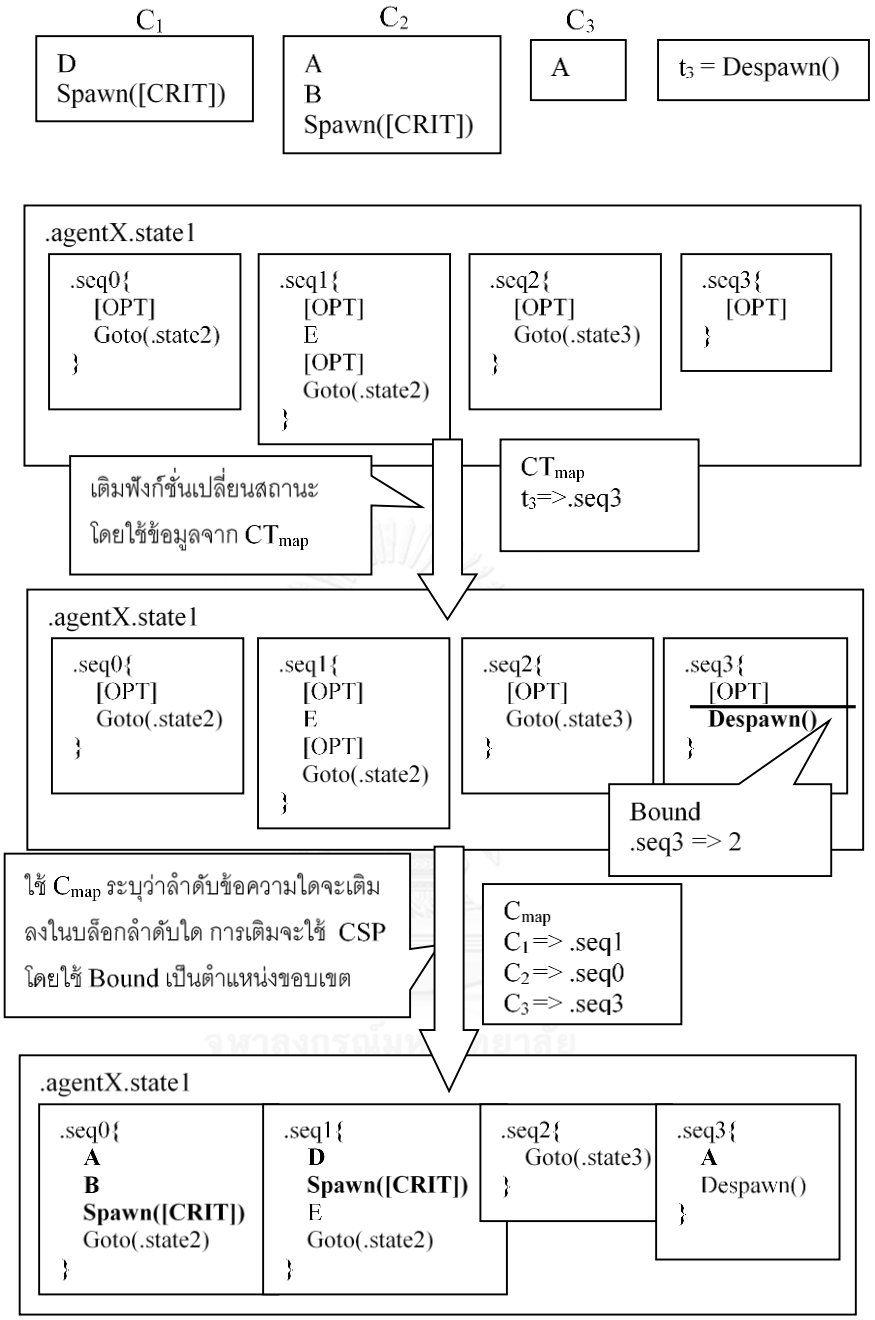
รูปที่ 56 แสดงขั้นตอนย่อยที่ 3 ของขั้นตอนที่ 3 ซึ่งเป็นการเติมเต็ม C_{map} และ CT_{map} โดยใช้วิธีเติมลำดับที่สุ่มเลือกจาก $SpawnMatch_{result}$ และ $TAlloc_{result}$ ซึ่งเป็นผลลัพธ์ของขั้นตอนย่อยก่อนหน้า



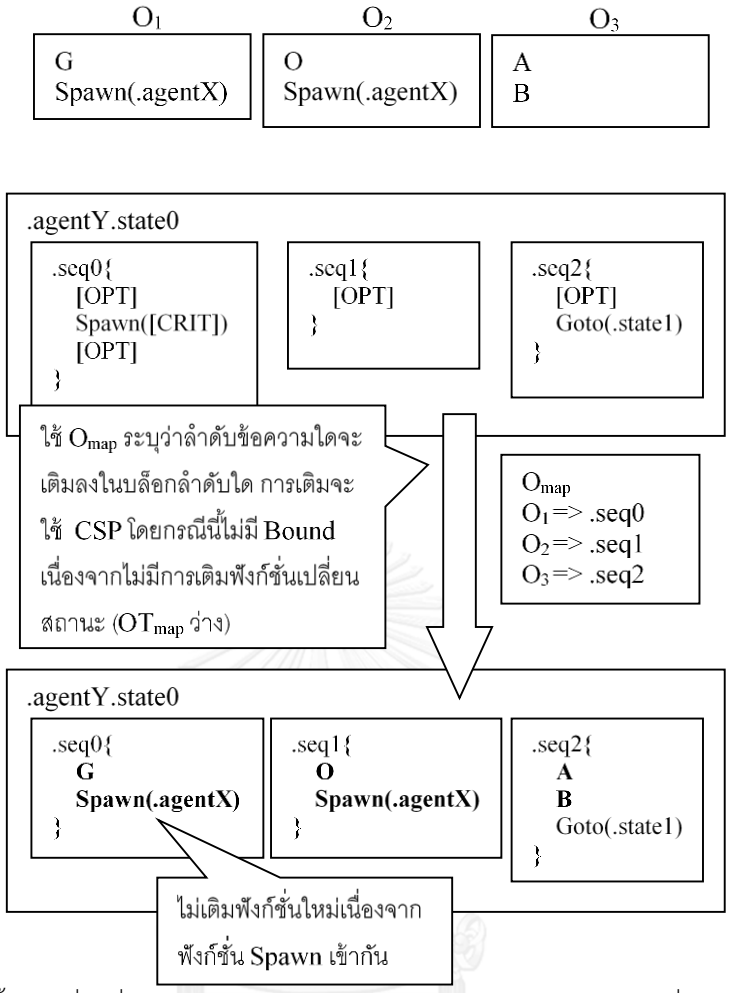
รูปที่ 57 แสดงขั้นตอนย่อยที่ 4 ของขั้นตอนที่ 3 ซึ่งเป็นการเพิ่มบล็อกลำดับให้ได้จำนวนตามเงื่อนไขแล้วจึงเติมเต็ม C_{map} เพื่อระบุบล็อกลำดับเป้าหมายให้ครบทุกลำดับข้อความสั่งใน *ChildList*



รูปที่ 58 แสดงขั้นตอนที่ 4 – 6 ซึ่งเป็นการเติมตัวระบุสำหรับฟังก์ชั่น Spawn ที่ปรากฏใน $OwnerList$ และเติมเต็ม O_{map} และ OT_{map} โดยใช้วิธีการในลักษณะเดียวกับการเติมเต็ม C_{map} และ CT_{map}



รูปที่ 59 แสดงขั้นตอนที่ 7 ซึ่งเป็นการเติม *ChildList* พร้อมฟังก์ชั่นเปลี่ยนสถานะลงในบล็อกลำดับเป้าหมาย ที่ระบุโดย C_{map} และ CT_{map}



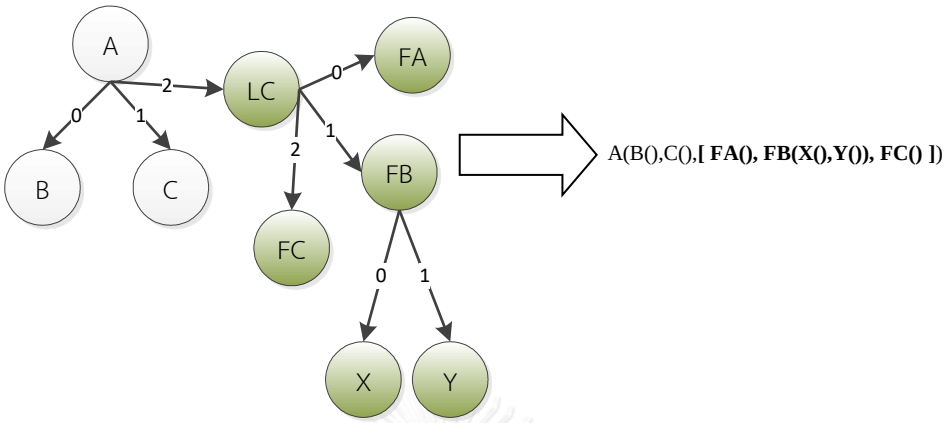
รูปที่ 60 แสดงขั้นตอนที่ 8 ซึ่งเป็นการเติม $OwnerList$ ลงในบล็อกลำดับเป้าหมายที่ระบุโดย O_{map} และ OT_{map}

6.14 การเติมความสัมพันธ์รูปแบบการเลือกใช้ฟังก์ชั่นข้อมูล

ความสัมพันธ์รูปแบบการเลือกใช้ฟังก์ชั่นข้อมูลจะใช้ในการเติมเต็มสล็อตจำเป็นต่างๆที่ยังไม่สมบูรณ์หลังจากที่เติมความสัมพันธ์รูปแบบอื่นๆแล้ว โดยความสัมพันธ์ชนิดนี้จะระบุว่า ตัวดำเนินการแต่ละตัวควรจะมีตัวถูกดำเนินการเป็นค่าใดได้บ้าง และฟังก์ชั่นที่ปรากฏอยู่ควรจะนำเอาข้อมูลใดมาเติมเป็นอากิวเมนท์ ในขั้นตอนนี้จะนำความสัมพันธ์ทั้งหมดที่ได้จากการทำเหมืองข้อมูลมาจัดทำเป็นการส่ง ซึ่งจับคู่ตัวดำเนินการและฟังก์ชั่น (ทั้งฟังก์ชั่นการกระทำและฟังก์ชั่นข้อมูล) เข้ากับรายการของตัวถูกดำเนินการหรืออากิวเมนท์ที่เป็นไปได้ จากนั้นจึงทำการสำรวจโครงภาคีเพื่อหาสล็อตจำเป็นและเติมเต็มสล็อตด้วยข้อมูลการส่งที่มี

ความสัมพันธ์ที่ได้จากการทำเหมืองข้อมูลจะอยู่ในรูปของกราฟ ซึ่งสามารถแปลงกลับเป็นตัวดำเนินการหรือฟังก์ชั่นโดยทำย้อนกลับวิธีการที่ระบุในหัวข้อ 5.5.1 สำหรับกรณีที่จุดยอดแทนอากิวเมนท์ใดไม่สามารถแปลงกลับได้เนื่องจากเป็นจุดยอดพิเศษ LC ซึ่งเกิดขึ้นจากกระบวนการเก็บอากิวเมนท์ที่เป็นไปได้ดังที่อธิบายในหัวข้อ 5.5.2 ให้แปลงแต่ละจุดยอดที่มีเส้นเชื่อมชี้เข้าหาจาก LC เพื่อจัดทำเป็นรายการของอากิวเมนท์ที่เป็นไปได้ แล้วใช้

รายการดังกล่าวนี้เป็นอากิวเมนท์ รูปที่ *61* แสดงตัวอย่างการแปลงกลับกรณีที่มีจุดยอดพิเศษ โดยป้ายชื่อของจุดยอดที่แสดงในกราฟจะแสดงด้วยฟังก์ชั่นสมมติที่แปลงกลับมาจากตัวเลขแล้วเพื่อความกระชับ



รูปที่ 61 ตัวอย่างการแปลงกราฟกลับเป็นฟังก์ชั่นที่มีรายการของอากิวเมนท์ที่เป็นไปได้

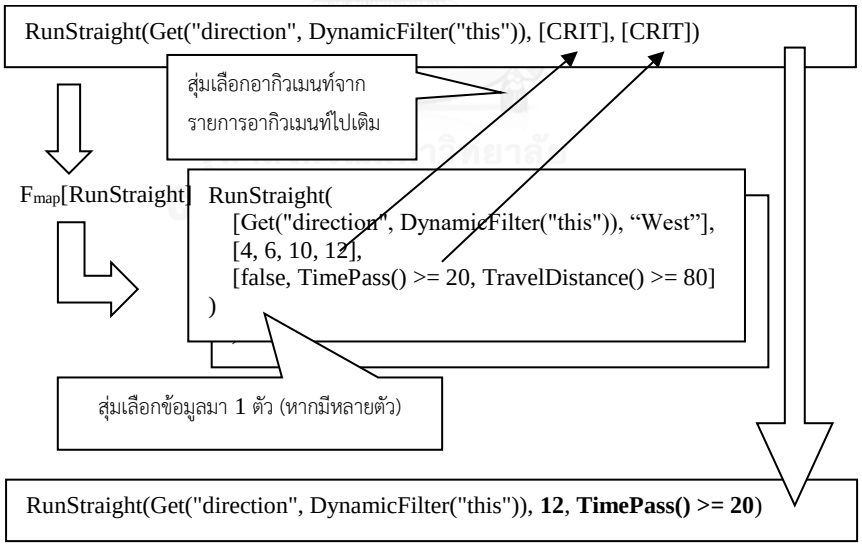
กำหนดให้ข้อมูลความสัมพันธ์ทั้งหมดที่แปลงให้อยู่ในรูปฟังก์ชั่นแล้วแทนด้วย $Nesting = \{N_1, N_2, .., N_n\}$ เมื่อ $N_i,\ \ 1 \le i \le n$ คือฟังก์ชั่นที่เป็นผลลัพธ์จากการแปลง สามารถนำไปสร้างเป็นการส่งและใช้เติมโครงภาคีด้วยขั้นตอนด้านล่าง รูปที่ *62* และรูปที่ *63* แสดงตัวอย่างการเติมความสัมพันธ์

1) เตรียมข้อมูลการส่งว่าง $F_{map} = \{\}$ ซึ่งใช้ระบุว่าฟังก์ชั่นหรือตัวดำเนินการใดๆมีอากิวเมนท์หรือตัวถูกดำเนินการเป็นค่าใดได้บ้าง โดยอยู่ในรูป $F_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าฟังก์ชั่นหรือตัวดำเนินการ k_i สามารถเติมได้ด้วยค่าต่างๆตามรายการ v_i
2) เติมเต็ม F_{map} ด้วยขั้นตอนต่อไปนี้

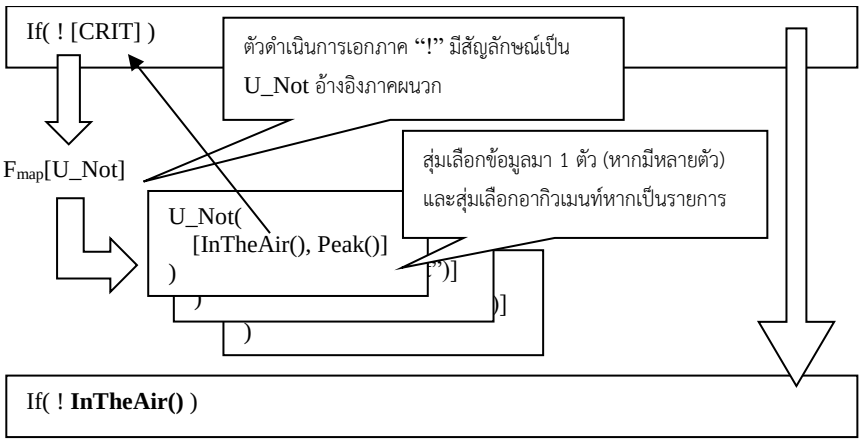
> For(N_i in $Nesting$) do:
>
> Verify N_i function name against action list (see appendix) and function list (see apeendix) to check if it is an action, function, unary operator or binary operator
>
> If(N_i is an unary operator) do: Key = operator symbol
>
> Else if(N_i is an binary operator) do: Key = operator symbol
>
> Else do: Key = function name
>
> $F_{map}[Key] \leftarrow N_i$'s signature

3) สำรวจสล็อตจำเป็นทั้งหมดที่ปรากฏในโครงภาคี สำหรับแต่ละสล็อตจำเป็นให้ตรวจสอบว่าสล็อตจำเป็นนี้อยู่ภายใต้ตัวดำเนินการหรือฟังก์ชั่นใด จากนั้นจึงดำเนินการตามแต่ประเภทข้อมูล

ประเภท	วิธีดำเนินการ
ฟังก์ชั่นการกระทำ	กำหนดให้สล็อตจำเป็นที่ต้องการเติมเป็นพารามิเตอร์ตัวที่ i ของฟังก์ชั่นชื่อ a_{name} ให้สุ่มเลือกข้อมูล 1 ตัวจาก $F_{map}[a_{name}]$ และนำอากิวเมนท์ที่ i มาแทนที่สล็อตจำเป็น ในกรณีที่อากิวเมนท์นั้นเป็นรายการของอากิวเมนท์ที่เป็นไปได้ ให้สุ่มเลือกมา 1 อากิวเมนท์จากรายการดังกล่าวเพื่อเติมลงในสล็อต
ฟังก์ชั่นข้อมูล	ใช้วิธีดำเนินการเช่นเดียวกับฟังก์ชั่นการกระทำ
ตัวดำเนินการเอกภาค เช่น ! [CRIT]	กำหนดให้ตัวดำเนินการเอกภาคนั้นมีสัญลักษณ์เป็น sym ให้สุ่มเลือกข้อมูล 1 ตัวจาก $F_{map}[sym]$ และนำอากิวเมนท์ของข้อมูลนั้นมาแทนที่สล็อตจำเป็น ในกรณีที่อากิวเมนท์นั้นเป็นรายการของอากิวเมนท์ที่เป็นไปได้ ให้สุ่มเลือกมา 1 อากิวเมนท์จากรายการดังกล่าวเพื่อเติมลงในสล็อต
ตัวดำเนินการทวิภาค เช่น [CRIT]+20	กำหนดให้ตัวดำเนินการเอกภาคนั้นมีสัญลักษณ์เป็น sym ให้สุ่มเลือกข้อมูล 1 ตัวจาก $F_{map}[sym]$ หากสล็อตจำเป็นเป็นนิพจน์ซ้ายของตัวดำเนินการ ให้นำอากิวเมนท์ที่ 1 มาแทนที่สล็อตจำเป็น หากเป็นนิพจน์ขวาให้ใช้อากิวเมนท์ที่ 2 ในกรณีที่อากิวเมนท์ที่เลือกเป็นรายการของอากิวเมนท์ที่เป็นไปได้ ให้สุ่มเลือกมา 1 อากิวเมนท์จากรายการดังกล่าวเพื่อเติมลงในสล็อต



รูปที่ 62 แสดงตัวอย่างการเติมความสัมพันธ์สำหรับฟังก์ชั่นการกระทำ



รูปที่ 63 แสดงตัวอย่างการเติมความสัมพันธ์สำหรับตัวดำเนินการเอกภาค

6.15 การกำหนดค่าให้สล็อตจำเป็นประเภทตัวระบุ

ขั้นตอนการเติมความสัมพันธ์รูปแบบการเลือกใช้ฟังก์ชั่นข้อมูลจะเติมเต็มสล็อตจำเป็นได้เกือบทั้งหมด ยกเว้นสล็อตจำเป็นประเภทตัวระบุซึ่งไม่ได้นำมาผ่านการทำเหมืองข้อมูลเพราะเป็นข้อมูลที่จำเพาะกับภาคี ขั้นตอนนี้จึงเป็นขั้นตอนสำหรับเติมเต็มข้อมูลเหล่านี้โดยพยายามทำให้ทุกสถานะเข้าถึงได้ (มีฟังก์ชั่นเปลี่ยนสถานะไปยังสถานะนั้น ๆ) และทุกภาคีถูกสร้างได้ (มีฟังก์ชั่น Spawn ที่สร้างภาคีนั้น ๆ ยกเว้นภาคีหลัก)

ขั้นตอนการเติมเต็มสล็อตประกอบด้วยการวิเคราะห์การเข้าถึงได้ เพื่อตรวจดูว่าโครงภาคีในปัจจุบัน มีสถานะใดที่ยังเข้าถึงไม่ได้ และมีภาคีใดที่ยังไม่ถูกสร้าง จากนั้นจึงกำหนดค่าให้สล็อตโดยทำให้ปัญหาอยู่ในรูปของ CSP และแก้ด้วยตัวแก้ปัญหาบังคับ ในที่นี้จะแยกอธิบายเป็นการเติมเต็มสล็อตเพื่อให้ภาคีเข้าถึงได้ และการเติมเต็มสล็อตเพื่อให้สถานะเข้าถึงได้

การเติมเต็มสล็อตเพื่อให้ภาคีเข้าถึงได้

ขั้นตอนนี้พยายามเติมเต็มสล็อตจำเป็นของฟังก์ชั่น Spawn เพื่อให้ภาคีทั้งหมดในโครงภาคีมีโอกาสถูกสร้างขึ้นได้ ในกรณีที่จำนวนสล็อตมีไม่เพียงพอกับจำนวนภาคีที่มีอยู่ ขั้นตอนนี้จะพยายามทำให้ฟังก์ชั่น Spawn ครอบคลุมภาคีให้ได้มากที่สุด รายละเอียดของขั้นตอนเป็นดังด้านล่าง โดยรูปที่ *64* และรูปที่ *65* แสดงตัวอย่างการเติมสล็อตสำหรับโครงภาคีที่ประกอบด้วย 4 ภาคีและมีสล็อตจำเป็นที่ต้องเติมตัวระบุทั้งหมด 4 สล็อต

1) วิเคราะห์การเข้าถึงได้และจัดสร้างข้อมูลการส่ง $Agent\,\mathrm{Re}ach_{map}$ สำหรับเก็บข้อมูลว่าภาคีใดถูกสร้างได้ด้วยภาคีใดบ้าง โดย $Agent\,\mathrm{Re}ach_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าภาคี k_i ถูกสร้างได้ด้วยภาคีในรายการ v_i ขั้นตอนการวิเคราะห์เป็นดังนี้

> For(Statement s in Skeleton) do:
>
> If(s is a Spawn action and its identifier I is assigned) do:
>
> $Agent\,\mathrm{Re}ach_{map}$[Agent named I] $\leftarrow$ Agent containing s

2) เตรียมข้อมูลการส่งว่าง $Spawn_{map} = \{\}$ ซึ่งใช้ระบุว่าสล็อตจำเป็นประเภทตัวระบุสำหรับฟังก์ชั่น Spawn ควรจะถูกเติมด้วยตัวระบุใด โดยอยู่ในรูป $Spawn_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าสล็อตจำเป็นที่ตำแหน่ง k_i ควรเติมด้วยตัวระบุ v_i

3) กำหนดให้

- มีสล็อตจำเป็นของฟังก์ชั่น Spawn ทั้งหมดจำนวน n ช่อง อยู่ที่ตำแหน่งต่าง ๆ ซึ่งระบุด้วยรายการ $Spawn_{index} = \{si_1,...,si_n\}$ เมื่อ si_i, $1 \le i \le n$ คือตำแหน่งของสล็อต และสัญลักษณ์ $[si_i]$ แทนสล็อตจำเป็นที่ตำแหน่ง si_i
- แต่ละภาคีที่ปรากฏในโครงภาคีมีตัวเลขแทนตัวระบุชื่อของภาคีดังกล่าว โดยตัวเลขจะกำหนดอย่างไรก็ได้แต่ต้องเป็นจำนวนเต็มบวกที่ไม่ซ้ำกันสำหรับทุกภาคี เช่น โครงภาคีที่มีภาคีชื่อ .Agent1 และ .Agent2 จะสามารถกำหนดให้เลข 1 แทนค่า .Agent1 และให้เลขจำนวนเต็มบวกใด ๆ ที่ไม่เป็น 1 แทนค่า .Agent2 เป็นต้น

สร้าง CSP ซึ่งมีรายละเอียดต่อไปนี้

- เซ็ตของตัวแปรที่สนใจ $X = \{x_1, x_2,.., x_n\} \cup \{f_1, f_2,.., f_n\} \cup c$
  - ค่าของตัวแปร x_i คือ ตัวเลขแทนตัวระบุที่ต้องการเติมลงใน $[si_i]$ $(1 \le i \le n)$ หากค่าของตัวแปรไม่เป็นจำนวนเต็มบวก แปลว่า ไม่กำหนดตัวระบุให้ $[si_i]$
  - ค่าของตัวแปร f_i คือ ตัวบ่งชี้ว่าได้กำหนดตัวระบุให้ $[si_i]$ หรือไม่
  - ค่าของ c คือ จำนวนสล็อตที่ไม่ได้กำหนดตัวระบุให้
- โดเมนสำหรับตัวแปร $D = \{d_1, d_2,.., d_n\} \cup \{fd_1, fd_2,..., fd_n\} \cup cd$
  - d_i $(1 \le i \le n)$ คือโดเมนของ x_i โดยเป็นเซ็ตซึ่งมีค่าเป็น $t\arg et_i \cup \{-i\}$ โดยค่า $-i$ มีไว้สำหรับกรณีที่ไม่กำหนดตัวระบุให้ $[si_i]$ และ $t\arg et_i$ คือเซ็ตของตัวเลขแทนตัวระบุ ซึ่งมีเฉพาะตัวระบุของภาคีที่ยังไม่ถูกสร้างด้วยภาคีใด ไม่เป็นภาคีหลัก และไม่เป็นภาคีที่ $[si_i]$ อยู่ สามารถหาสมาชิกของเซ็ต $t\arg et_i$ ได้ดังนี้

    $t\arg et_i = \{\}$
    For(Agent A in Skeleton) do:
      If($Agent\ \mathrm{Re}ach_{map}[A]$ is empty and A is not main agent
        and $[si_i]$ is not a part of A) do:
          $t\arg et_i \leftarrow$ number referring to identifier of A

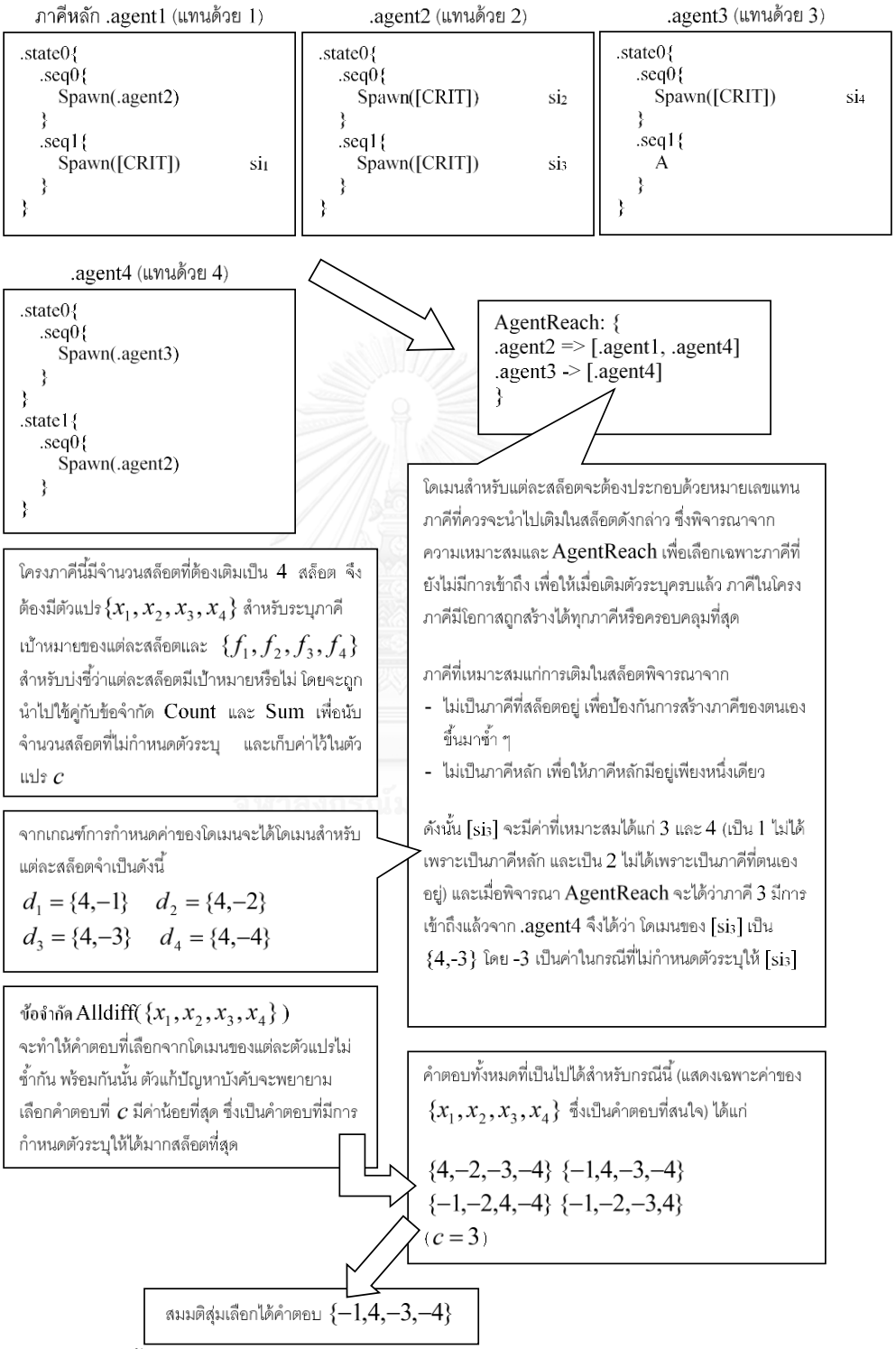
  - fd_i $(1 \le i \le n)$ คือโดเมนของ f_i โดย $fd_i = \{0,1\}$ โดยค่า 0 แทนกรณีที่กำหนดตัวระบุให้ $[si_i]$
  - cd คือโดเมนของ c โดย $cd = \{0\} \cup I^+$
- ข้อจำกัดของปัญหา คือ
  - $Count(\{x_1,...,x_n\}, -i, f_i)$ สำหรับทุก i $(1 \le i \le n)$
  - $Sum(\{f_1,..., f_n\}, c)$
  - $Alldiff(\{x_1,...,x_n\})$

4) ใช้ตัวแก้ปัญหาบังคับหาคำตอบของ CSP ภายใต้เงื่อนไขให้เลือกเฉพาะคำตอบที่ทำให้ c มีค่าน้อยที่สุด ซึ่งอาจมีได้หลายคำตอบ

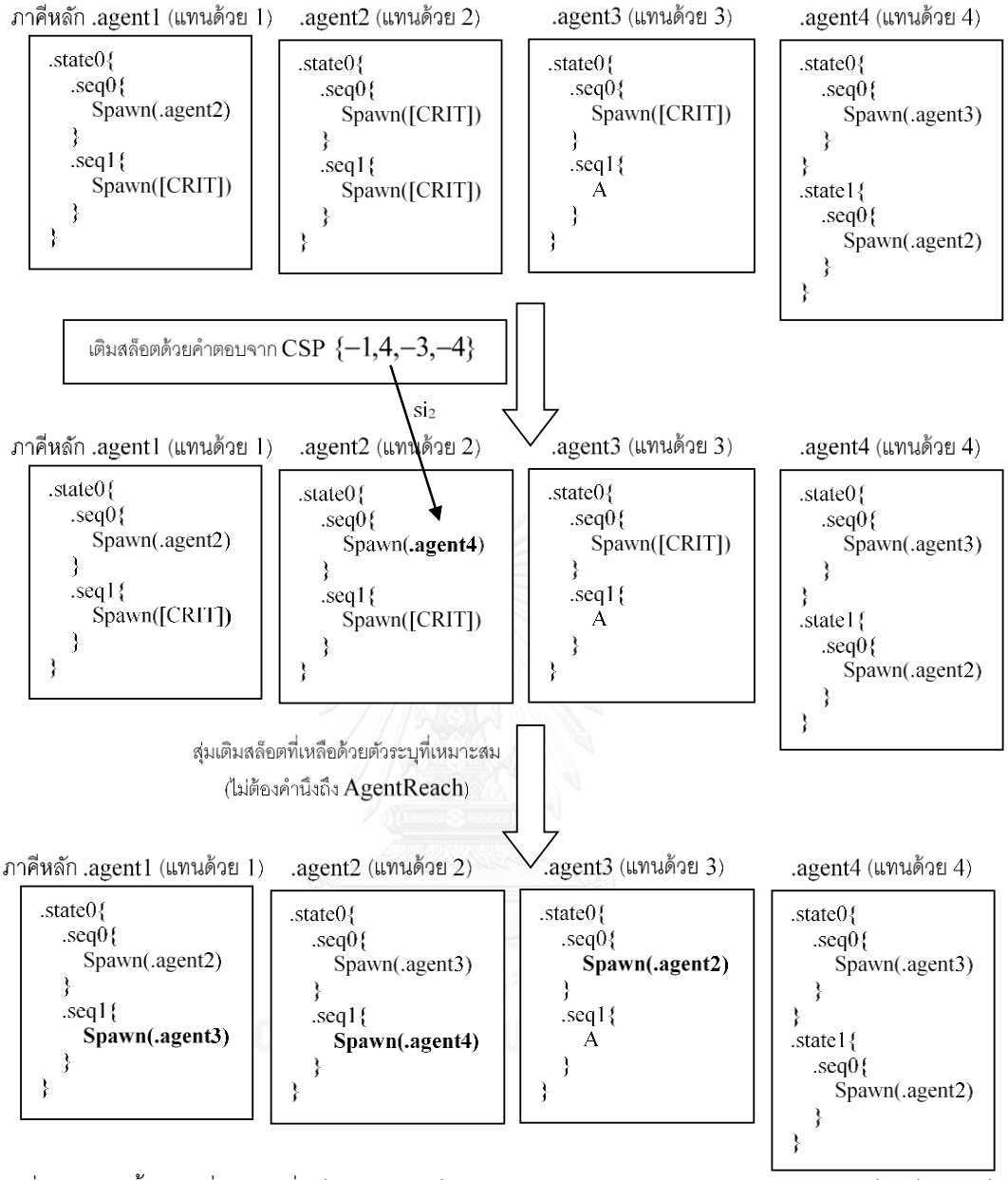
5) สุ่มเลือก 1 คำตอบและใช้คำตอบดังกล่าวเติมเต็มสล็อตจำเป็นดังนี้

For(Assignment x_i in $\{x_1, x_2, .., x_n\}$) do:
  If($x_i > 0$) do: Fill $[si_i]$ with an identifier referred by x_i

6) สุ่มเติมสล็อตจำเป็นที่เหลือด้วยตัวระบุของภาคีใดก็ได้ที่ไม่ใช่ภาคีหลักและไม่ใช่ภาคีที่สล็อตนั้นอยู่



รูปที่ 64 แสดงขั้นตอนที่ 1 – 4 ซึ่งเป็นการสร้าง CSP และแก้ปัญหาเพื่อหาวิธีกำหนดตัวระบุให้กับสล็อต



รูปที่ 65 แสดงขั้นตอนที่ 5 – 6 ซึ่งเป็นการเติมสล็อตด้วยคำตอบจากตัวแก้ปัญหาบังคับและเติมเต็มสล็อตจำเป็นที่เหลืออย่างสุ่ม

การเติมเต็มสล็อตเพื่อให้สถานะเข้าถึงได้

ขั้นตอนนี้พยายามเติมเต็มสล็อตจำเป็นของฟังก์ชั่น Goto รวมถึงพยายามเติมฟังก์ชั่น Goto เพิ่มเพื่อให้ทุกสถานะที่ปรากฏในโครงภาคีสามารถเข้าถึงได้ โดยพยายามเพิ่มฟังก์ชั่น Goto ให้น้อยที่สุด แต่ให้ครอบคลุมทุกสถานะมากที่สุด (ในกรณีที่ไม่สามารถทำให้เข้าถึงได้ทุกสถานะ) รายละเอียดของขั้นตอนเป็นดังด้านล่าง โดยรูปที่ *66* และรูปที่ *67* แสดงตัวอย่างการเติมสล็อตสำหรับกรณีที่โครงภาคีมี 5 บล็อกสถานะ และมี 2 สล็อตจำเป็นและ 3 สล็อตทางเลือกสำหรับเติมตัวระบุของสถานะปลายทาง

1) วิเคราะห์การเข้าถึงได้และจัดสร้างข้อมูลการส่ง $State\text{Re}ach_{map}$ ซึ่งระบุว่าสถานะใดเข้าถึงได้จากสถานะใดบ้าง โดย $State\text{Re}ach_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าสถานะ k_i เข้าถึงได้จากสถานะในรายการ v_i ขั้นตอนการวิเคราะห์เป็นดังนี้

> For(Statement s in Skeleton) do:
>
> If(s is a Goto action and its identifier I is assigned) do:
>
> Let Key = state named I that is a part of the same agent as s
>
> $State\text{Re}ach_{map}[Key] \leftarrow$ State containing s

2) เตรียมข้อมูลการส่งว่าง $Goto_{map} = \{\}$ ซึ่งใช้ระบุว่าสล็อตจำเป็นประเภทตัวระบุสำหรับฟังก์ชั่น Goto ควรจะถูกเติมด้วยตัวระบุใด โดยอยู่ในรูป $Goto_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าสล็อตจำเป็นที่ตำแหน่ง k_i ควรเติมด้วยตัวระบุ v_i

3) เตรียมข้อการส่งว่าง $GotoStub_{map} = \{\}$ ซึ่งใช้ระบุว่าจะต้องเติมฟังก์ชั่น Goto ใหม่ที่สล็อตตัวเลือกประเภทข้อความสั่งใด และใช้ตัวระบุใดเป็นอากิวเมนท์ โดยอยู่ในรูป $GotoStub_{map} = \{k_1 => v_1, ...\}$ เมื่อ $k_i => v_i$ ระบุว่าควรเติมฟังก์ชั่น Goto ลงในสล็อตตัวเลือกที่ตำแหน่ง k_i โดยมีอากิวเมนท์เป็น v_i

4) กำหนดให้

- มีสล็อตจำเป็นของฟังก์ชั่น Goto ทั้งหมดจำนวน m ช่อง อยู่ที่ตำแหน่งต่าง ๆ ซึ่งระบุด้วยรายการ $Goto_{index} = \{gi_1, ..., gi_m\}$ เมื่อ $gi_i, \quad 1 \le i \le m$ คือตำแหน่งของสล็อต และสัญลักษณ์ $[gi_i]$ แทนสล็อตจำเป็นที่ตำแหน่ง gi_i
- มีสล็อตตัวเลือกที่สามารถเติมฟังก์ชั่น Goto ได้จำนวน n ช่อง อยู่ที่ตำแหน่งต่าง ๆ ซึ่งระบุด้วยรายการ $GotoStub_{index} = \{gsi_1, ..., gsi_n\}$ เมื่อ $gsi_i, \quad 1 \le i \le n$ คือตำแหน่งของสล็อต และสัญลักษณ์ $[gsi_i]$ แทนสล็อตตัวเลือกประเภทข้อความสั่งที่ตำแหน่ง gsi_i
- แต่ละบล็อกสถานะที่ปรากฏในโครงภาคีมีตัวเลขแทนบล็อกสถานะดังกล่าว โดยตัวเลขจะกำหนดอย่างไรก็ได้แต่ต้องเป็นจำนวนเต็มบวกที่ไม่ซ้ำกันสำหรับทุกบล็อกสถานะ หากโครงภาคีปัจจุบันมีจำนวนบล็อกสถานะ k บล็อก จะได้ว่าเซ็ตของตัวเลขแทนบล็อกสถานะทั้งหมด คือ $S_{Iden} = \{Id_1, ..., Id_k\}$ เมื่อ $Id_i, \quad 1 \le i \le k$ คือตัวเลขแทนบล็อกสถานะ

5) สร้างตารางค่าใช้จ่าย $CostTable$ สำหรับใช้ตัดสินใจเติมสล็อตด้วย CSP ในขั้นตอนถัดไป ซึ่ง

- $CostTable[i][j] = 0 \quad (1 \le i \le m, 1 \le j \le k)$ โดยกรณีนี้เป็นค่าใช้จ่ายที่เกิดขึ้นเมื่อเติม $[gi_i]$ ด้วยตัวระบุชื่อของบล็อกสถานะที่แทนด้วยตัวเลข Id_j ซึ่งให้ค่าใช้จ่ายเป็น 0 เนื่องจากจุดประสงค์คือการเติมเต็มสล็อตจำเป็น
- $CostTable[i][0] = 100 \quad (1 \le i \le m)$ โดยกรณีนี้เป็นค่าใช้จ่ายที่เกิดขึ้นเมื่อไม่เติม $[gi_i]$ ซึ่งเป็นกรณีที่ไม่ต้องการให้เกิดขึ้น
- $CostTable[i][j] = 10 \quad (m+1 \le i \le m+n, 1 \le j \le k)$ โดยกรณีนี้เป็นค่าใช้จ่ายที่เกิดขึ้นเมื่อเติมเพิ่มฟังก์ชั่น Goto ใน $[gsi_{i-m}]$ โดยมีอากิวเมนท์เป็นตัวระบุชื่อของบล็อกสถานะที่แทนด้วยตัวเลข Id_j ซึ่งให้ค่าใช้จ่ายมากกว่ากรณีแรกสุด เนื่องจากไม่ต้องการให้เกิดฟังก์ชั่น Goto ใหม่

- $CostTable[i][0] = 0 \quad (m+1 \le i \le m+n)$ โดยกรณีนี้เป็นค่าใช้จ่ายที่เกิดขึ้นเมื่อไม่เติมฟังก์ชั่น Goto ใน $[gsi_{i-m}]$ ซึ่งให้ค่าใช้จ่ายเป็น 0 เพื่อลดการเกิดฟังก์ชั่น Goto ใหม่

6) สร้าง CSP ซึ่งมีรายละเอียดต่อไปนี้

- เซ็ตของตัวแปรที่สนใจ $X = \{x_1,..,x_m,...,x_{m+n}\} \cup \{f_1,..,f_m,..,f_{m+n}\} \cup c$
  - ค่าของตัวแปร x_i มีความหมายแตกต่างกันตามค่า i ดังนี้
    - กรณี $1 \le i \le m$ x_i คือ ตัวเลขแทนตัวระบุที่ต้องการเติมลงใน $[gi_i]$ หากค่าของตัวแปรไม่เป็นจำนวนเต็มบวก แปลว่า ไม่กำหนดตัวระบุให้สล็อตดังกล่าว
    - กรณี $m+1 \le i \le m+n$ x_i คือ ตัวเลขแทนตัวระบุที่ต้องการใช้เป็นอากิวเมนท์ของฟังก์ชั่น Goto ที่ต้องการเพิ่มลงใน $[gsi_{i-m}]$ หากค่าของตัวแปรไม่เป็นจำนวนเต็มบวก แปลว่า ไม่เพิ่มฟังก์ชั่น Goto ที่สล็อตดังกล่าว
  - ค่าของตัวแปร f_i คือ ค่าใช้จ่ายที่เกิดจากการเลือกค่าของตัวแปร x_i
  - ค่าของ c คือ ค่าใช้จ่ายรวมจากการเลือกค่าตัวแปรทั้งหมด
- โดเมนสำหรับตัวแปร $D = \{d_1,..,d_m,..,d_{m+n}\} \cup \{fd_1,...,fd_m,..,fd_{m+n}\} \cup cd$
  1) $d_i \quad (1 \le i \le m+n)$ คือโดเมนของ x_i โดยเป็นเซ็ตซึ่งมีค่าเป็น $target_i \cup \{-i\}$
     - กรณี $1 \le i \le m$ ค่า $-i$ มีไว้สำหรับกรณีที่ไม่กำหนดตัวระบุให้ $[gi_i]$ และ $target_i$ คือเซ็ตของตัวเลขแทนตัวระบุ ซึ่งมีเฉพาะตัวระบุของสถานะที่เข้าถึงไม่ได้และเป็นสถานะในภาคีเดียวกับที่ $[gi_i]$ อยู่ แต่ไม่เป็นสถานะที่ $[gi_i]$ อยู่ สามารถหาสมาชิกของเซ็ต $target_i$ ได้ดังนี้

```
target_i = {}
For( State S in Skeleton ) do:
    If( State_Reach_map[S] is empty and S is in the same agent
            as [gi_i] and [gi_i] is not a part of S ) do:
        target_i ← number referring to identifier of S
```

     - กรณี $m+1 \le i \le m+n$ ค่า $-i$ มีไว้สำหรับกรณีที่ไม่เพิ่มฟังก์ชั่น Goto ใน $[gsi_{i-m}]$ และ $target_i$ คือเซ็ตของตัวเลขแทนตัวระบุ ซึ่งมีเฉพาะตัวระบุของสถานะที่เข้าถึงไม่ได้และเป็นสถานะในภาคีเดียวกับที่ $[gsi_{i-m}]$ อยู่ แต่ไม่เป็นสถานะที่ $[gsi_{i-m}]$ อยู่ สามารถหาสมาชิกของเซ็ต $target_i$ ได้ดังนี้

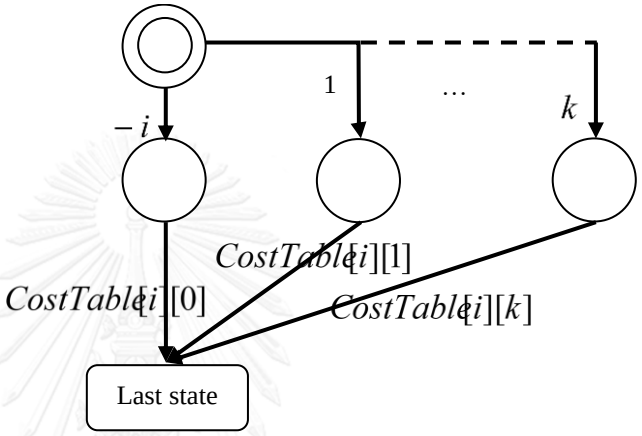
```
target_i = {}
For( State S in Skeleton ) do:
```

If($State\,\mathrm{Re}\,ach_{map}[S]$ is empty and S is in the same agent as $[gsi_{i-m}]$ and $[gsi_{i-m}]$ is not a part of S) do:
$t\,\mathrm{arg}\,et_i \leftarrow$ number referring to identifier of S

2) fd_i $(1 \le i \le m+n)$ คือโดเมนของ f_i โดย $fd_i = \{0\} \cup I^+$

3) cd คือโดเมนของ c โดย $cd = \{0\} \cup I^+$

- ข้อจำกัดของปัญหา คือ
  - $\mathrm{Re}\,gular(fsm_i, \{x_i, f_i\})$ สำหรับทุก i $(1 \le i \le m+n)$ โดยภาพด้านล่างแสดงเครื่องสถานะจำกัดสำหรับ fsm_i



  - $Alldiff(\{x_1,...,x_{m+n}\})$
  - $Sum(\{f_1,...,f_{m+n}\}, c)$

7) ใช้ตัวแก้ปัญหาบังคับหาคำตอบของ CSP หาทุกคำตอบที่เป็นไปได้ ในที่นี้กำหนดให้คำตอบที่ได้อยู่ในรูปของรายการ $\mathrm{Re}\,sult = \{asn_1, asn_2, ..\}$ เมื่อสมาชิกของรายการ คือ 1 คำตอบที่ได้จากตัวแก้ปัญหา

8) คัดคำตอบที่ได้จากขั้นตอนที่ 7 ให้ได้เฉพาะคำตอบที่ทำให้มีจำนวนภาคีที่มีการเข้าถึงสมบูรณ์เยอะที่สุด ซึ่งวัดโดยนำคำตอบมาทดลองเติมเต็มสล็อตแล้วนับจำนวนภาคีที่มีการเข้าถึงสมบูรณ์ซึ่งมีนิยามดังนี้

ภาคี A จะนับว่าเป็นภาคีที่มีการเข้าถึงสมบูรณ์ (Fully reached agent) ก็ต่อเมื่อ กราฟระบุชื่อแบบมีทิศทาง G มีทางเดินจากจุดยอด v' ไปยังจุดยอดอื่น ๆ ทุกจุด โดยกราฟ G สามารถสร้างได้ดังนี้

For(State S in A) do:
  Create a new vertex labeled as identifier of S for graph G
  If(S is initial state) do: Let $v' =$ the created vertex
For(Vertex v in G) do:
  For(Statement st in state S denoted by v) do:
    If(st is a Goto function with identifier I as an argument) do:
      Create a directed edge from v to a vertex labeled I

การเลือกคำตอบที่มีจำนวนภาคีตามเงื่อนไขดังกล่าวสูงที่สุดมีวิธีการเป็นดังนี้

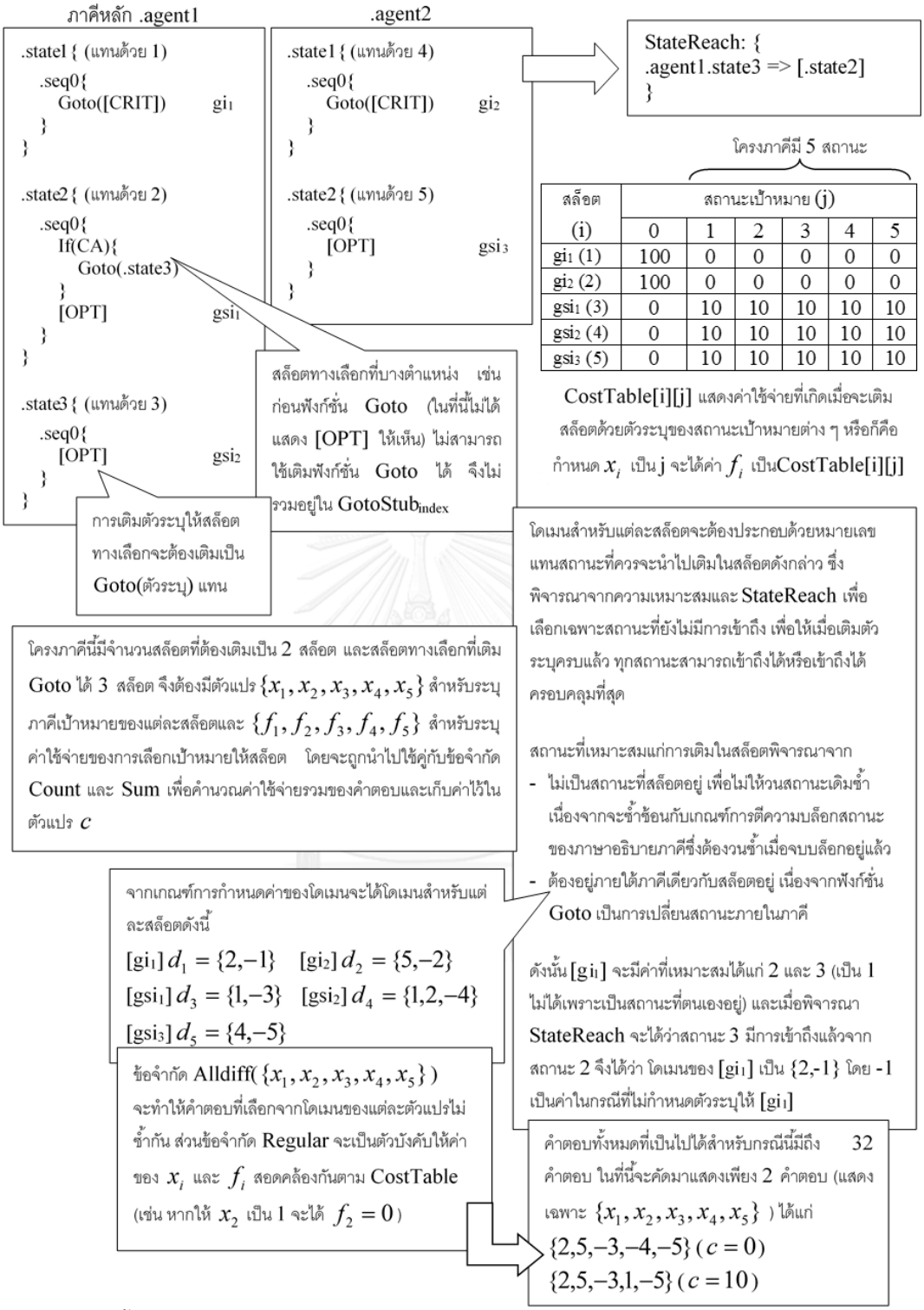
$Desired\,\mathrm{Re}sult = \phi, \max Agent = 0$
For(Result asn in $\mathrm{Re}sult$) do:
  Create a copy of skeleton as $Skel'$
  For(Assignment x_i in $\{x_1,..,x_m\}$ of asn) do:
    If($x_i > 0$) do: Fill $[gi_i]$ in $Skel'$ with an identifier referred by x_i
  For(Assignment x_i in $\{x_{m+1},..,x_{m+n}\}$ of asn) do:
    If($x_i > 0$) do:
      Fill $[gsi_{i-m}]$ in $Skel'$ with Goto(identifier referred by x_i)
  Let $agentCnt = 0$
  For(Agent A in $Skel'$) do:
    If(A is fully reached agent) do: $agentCnt++$
  If($agentCount > \max Agent$) do:
    $Desired\,\mathrm{Re}sult = \phi, \max Agent = agentCnt$
  If($agentCount \geq \max Agent$) do:
    $Desired\,\mathrm{Re}sult \leftarrow asn$
Return $Desired\,\mathrm{Re}sult$ as result

9) คัดคำตอบที่ได้จากขั้นตอนที่ 8 ให้เหลือเฉพาะคำตอบทำให้ตัวแปร c (ค่าใช้จ่ายของคำตอบ) มีค่าน้อยที่สุด ซึ่งอาจจะมีได้หลายคำตอบ

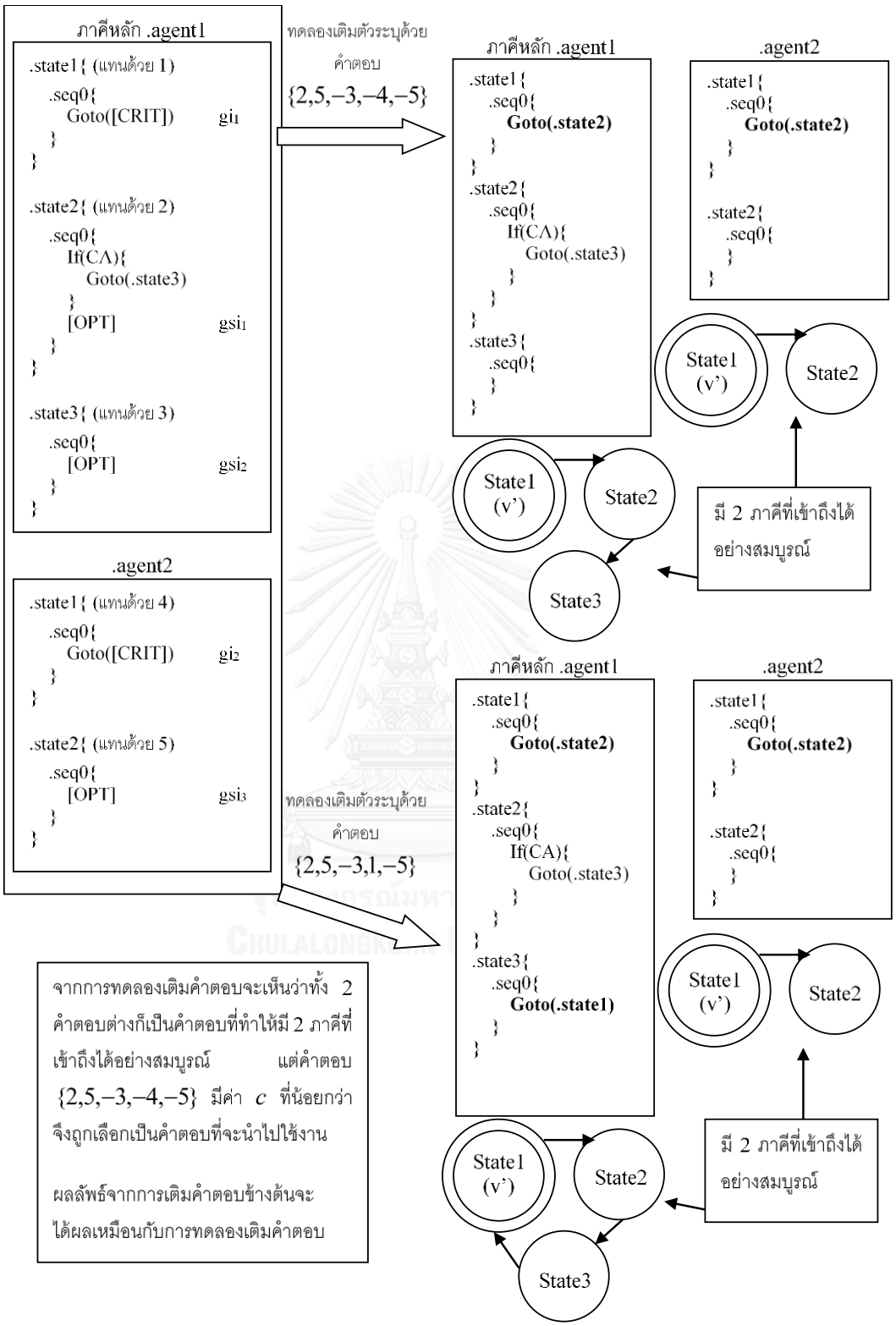
10) สุ่มเลือก 1 คำตอบจากคำตอบที่ผ่านการคัดแล้วและใช้คำตอบดังกล่าวเติมเต็มสล็อตดังนี้

For(Assignment x_i in $\{x_1,..,x_m\}$) do:
  If($x_i > 0$) do: Fill $[gi_i]$ with an identifier referred by x_i
For(Assignment x_i in $\{x_{m+1},..,x_{m+n}\}$ of asn) do:
  If($x_i > 0$) do: Fill $[gsi_{i-m}]$ with Goto(identifier referred by x_i)

11) หากมีฟังก์ชั่น Goto ที่ยังต้องการอากิวเมนท์ประเภทตัวระบุเหลืออยู่ในโครงภาคี ให้ลบฟังก์ชั่นเหล่านั้นออกจากโครงภาคี



รูปที่ 66 แสดงขั้นตอนที่ 1 – 7 ซึ่งเป็นการหาวิธีเติมตัวระบุลงในสล็อตทุกวิธีที่เป็นไปได้พร้อมค่าใช้จ่ายสำหรับแต่ละวิธี



รูปที่ 67 แสดงขั้นตอนที่ 8 – 11 ซึ่งเป็นการคัดวิธีเติมตัวระบุและทำการเติมตัวระบุด้วยวิธีที่คัด

6.16 การปรับปรุงภาคีด้วยฮิวริสติก

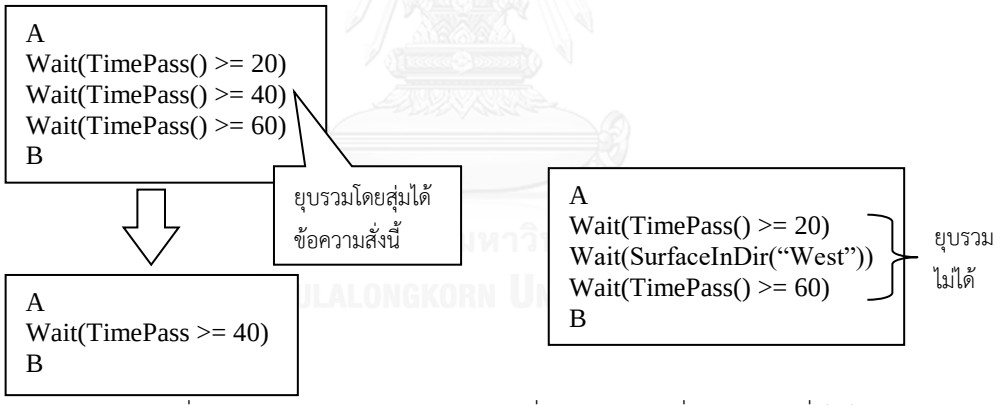
เพื่อให้ภาคีผลลัพธ์มีโอกาสเป็นภาคีที่ยอมรับได้ตามนิยามการยอมรับได้ที่จะกล่าวถึงในหัวข้อ 7.1 มากขึ้น จึงต้องมีฮิวริสติกช่วยปรับปรุงให้ภาคีเป็นไปในทิศทางดังกล่าวนั้น ในที่นี้ใช้ฮิวริสติก 2 ตัว ได้แก่

ฮิวริสติกสำหรับลดการหยุดรอของภาคี

เนื่องจากศัตรูที่อยู่นิ่งจนทำให้หลบหลีกได้ง่ายเกินไปเป็นหนึ่งในกรณีที่อาจส่งผลให้ศัตรูไม่สามารถยอมรับได้ การลดระยะเวลาการหยุดรอจึงน่าจะช่วยแก้ปัญหาจุดนี้ได้ ฮิวริสติกสำหรับลดการหยุดรอปรับปรุงภาคีโดยการค้นหาฟังก์ชั่น Wait ที่คล้ายกันจำนวนหลายฟังก์ชั่นที่เรียงติดกันในลำดับพฤติกรรม แล้วยุบรวมฟังก์ชั่นดังกล่าวให้เหลือเพียงฟังก์ชั่นเดียวโดยสุ่มเลือก 1 ฟังก์ชั่นจากฟังก์ชั่นที่เรียงติดกันดังกล่าว โดยฟังก์ชั่น Wait 2 ฟังก์ชั่นจะคล้ายกันก็ต่อเมื่ออากิวเมนท์ของทั้ง 2 ฟังก์ชั่นที่อยู่ในรูปของต้นไม้นิพจน์มีปมรากเหมือนกัน โดยมีวิธีเทียบความเหมือนของปมรากดังนี้

- กรณีที่ปมรากเป็นตัวดำเนินการเอกภาคหรือตัวดำเนินการทวิภาค ทั้งสองปมจะต้องมีตัวดำเนินการเดียวกัน
- กรณีที่ปมรากเป็นฟังก์ชั่น ทั้งสองปมจะต้องเป็นฟังก์ชั่นเดียวกัน

รูปที่ *68* แสดงตัวอย่างของฟังก์ชั่น Wait ที่ยุบรวมฟังก์ชั่นได้และกรณีที่ยุบรวมไม่ได้



รูปที่ 68 (ซ้าย) ตัวอย่างการยุบรวมฟังก์ชั่น (ขวา) กรณีที่ยุบรวมฟังก์ชั่นไม่ได้

ฮิวริสติกสำหรับให้ภาคีคงอยู่ในฉาก

เนื่องจากศัตรูที่ยอมรับได้จะต้องเป็นศัตรูที่ปราบได้ จึงต้องมีฮิวริสติกที่ป้องกันภาคีหลักออกจากฉากไป ซึ่งอาจส่งผลให้ผู้เล่นไม่สามารถปราบศัตรูได้สำเร็จ ในที่นี้ทำได้โดยลบฟังก์ชั่น Set ที่กำหนดค่าคุณสมบัติผ่านได้ให้กับศัตรูออกจากบล็อกภาคีของภาคีหลักทั้งหมด เพื่อป้องกันไม่ให้ภาคีหลักทะลุผ่านกำแพงของฉากออกไป

บทที่ 7
การวัดผล

เนื้อหาในบทนี้กล่าวถึงการวัดผลขั้นตอนวิธีสำหรับสร้างรูปแบบพฤติกรรมศัตรู ประสิทธิผลของขั้นตอนวิธีจะถูกวัดจากจำนวนของศัตรูที่ยอมรับได้เทียบกับจำนวนศัตรูทั้งหมดที่สร้างขึ้นมาด้วยขั้นตอนวิธีที่นำเสนอ ศัตรูที่ยอมรับได้ตัดสินโดยการเปรียบเทียบข้อมูลเชิงปริมาณที่ได้จากการต่อสู้ระหว่างตัวละครอวตารและศัตรูในเกมทดสอบ เทียบกับค่าบรรทัดฐานที่วัดจากชุดข้อมูลศัตรู การควบคุมตัวละครอวตารในการทดสอบจะทำโดยใช้ปัญญาประดิษฐ์ที่สามารถเล่นได้คล้ายคลึงกับผู้เล่นภายใต้นิยามที่กำหนดไว้

7.1 การประเมินการยอมรับได้

งานวิจัยนี้มีจุดประสงค์เพื่อนำเสนอขั้นตอนวิธีที่สามารถสร้างรูปแบบพฤติกรรมของศัตรูที่ยอมรับได้ ซึ่งการยอมรับได้นั้นเป็นข้อมูลเชิงคุณภาพ ในหัวข้อนี้จะทำการนิยามการยอมรับได้ และนิยามข้อมูลเชิงปริมาณที่จะถูกใช้ในการประเมินการยอมรับได้

เมื่อผู้เล่นเผชิญหน้ากับศัตรูในเกมแอ็คชั่นแบบเน้นตัวละครนั้น ผู้เล่นจะต้องตัดสินใจหลบหลีกและ/หรือปราบศัตรูเพื่อเดินหน้าต่อไป เหตุที่ต้องหลบหลีกตัวศัตรูหรือการโจมตีของศัตรูนั้น ก็เพื่อป้องกันไม่ให้ตัวละครอวตารสูญเสียพลังชีวิต ซึ่งส่งผลให้ผู้เล่นพ่ายแพ้ในเกมไป ในขณะที่การปราบศัตรูจะช่วยให้ตัวละครอวตารสามารถมุ่งหน้าต่อไปยังเส้นชัย หรือลดปริมาณของการโจมตีจากศัตรูลงเพื่อลดโอกาสการสูญเสียพลังชีวิต ศัตรูในเกมแอ็คชั่นประเภทนี้จึงถูกออกแบบให้สามารถหลบหลีกการโจมตีและ/หรือปราบได้ ดังนั้น รูปแบบพฤติกรรมของศัตรูที่ผู้เล่นยอมรับได้จึงต้องส่งผลให้ตัวศัตรูมีคุณสมบัติดังกล่าวข้างต้น งานวิจัยนี้จึงได้นิยามให้รูปแบบพฤติกรรมที่ยอมรับได้ดังนี้

“รูปแบบพฤติกรรมที่ยอมรับได้ คือ รูปแบบพฤติกรรมของศัตรูที่ให้ผู้เล่นสามารถหลบหลีกหรือปราบศัตรูได้”

ศัตรูที่ผู้เล่นปราบได้ หมายถึงศัตรูดังกล่าวสามารถสูญเสียพลังชีวิตจนเหลือ 0 ได้ ดังนั้น ข้อมูลเชิงปริมาณตัวแรกสำหรับการประเมินการยอมรับได้ คือ “พลังชีวิตที่เหลืออยู่ของศัตรู” ข้อมูลนี้จะถูกคำนวณเมื่อการต่อสู้ระหว่างตัวละครอวตารและศัตรูจบลง โดยอยู่ในรูปของอัตราส่วนพลังชีวิตที่เหลืออยู่เทียบกับพลังชีวิตเริ่มต้น สาเหตุที่งานวิจัยนี้ไม่ใช้ข้อมูลสองตัวเลือก (ปราบได้/ปราบไม่ได้) เนื่องจากในบางกรณีอาจมีการประเมินผลในขณะที่การต่อสู้ยังไม่จบลง และศัตรูยังไม่ถูกปราบ

ในกรณีของการหลบหลีก ศัตรูที่ผู้เล่นสามารถหลบหลีกได้อาจจะไม่ใช่ศัตรูที่ผู้เล่นยอมรับได้เสมอไป ยกตัวอย่างเช่น ศัตรูที่ไม่มีการเคลื่อนไหวใด ๆ หากไม่นำเอาโครงสร้างพื้นที่เข้ามาพิจารณา จะพบว่าผู้เล่นสามารถกระโดดข้ามศัตรูไปได้อย่างง่ายดาย โดยไม่จำเป็นต้องพึ่งปฏิกิริยาโต้ตอบใด ๆ อันเป็นลักษณะของเกมแอ็คชั่นแบบเน้นตัวละคร ด้วยเหตุนี้ หากจะนำเอาการหลบหลีกมาใช้ในการพิจารณาการยอมรับได้ของพฤติกรรม จะต้องมีการแบ่งระดับการหลบหลีก เพื่อให้สามารถวัดได้ว่า เมื่อผู้เล่นเผชิญหน้ากับศัตรูตัวหนึ่งๆ ผู้เล่นถูกการโจมตีของศัตรูนั้นมากน้อยเพียงใด ดังนั้น ข้อมูลเชิงปริมาณตัวที่สองที่จะใช้ในการประเมินการยอมรับได้ คือ “อัตราหลบหลีกพลาด”

โดยคำนวณจากจำนวนครั้งที่โดนการโจมตีของศัตรูเทียบกับจำนวนครั้งที่มีโอกาสโดนโจมตีทั้งหมดตลอดระยะเวลาการต่อสู้ การคำนวณ "จำนวนครั้ง" ทั้งสองค่าขึ้นอยู่กับกลไกของเกมที่เพิ่มเติมขึ้นมาในเกมทดสอบนี้ด้วย รายละเอียดของการคำนวณจึงจะกล่าวถึงอีกครั้งในหัวข้อ 7.2.2

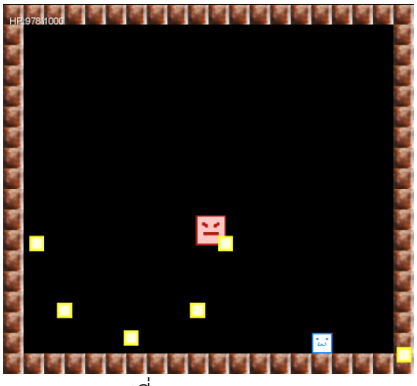
ค่าเชิงปริมาณทั้งสองข้างต้นจะแสดงถึงประสิทธิภาพของตัวผู้เล่นในการต่อสู้กับศัตรูตัวหนึ่งๆ ในการต่อสู้กับศัตรูที่มีรูปแบบพฤติกรรมเหมือนกันหรือคล้ายคลึงกัน ภายใต้ระยะเวลาที่กำหนดเท่ากัน ประสิทธิภาพก็ควรจะออกมาคล้ายคลึงกันด้วย งานวิทยานิพนธ์นี้จึงอาศัยจุดนี้ในการประเมินการยอมรับได้ โดยนำค่าเชิงปริมาณจากการต่อสู้กับศัตรูที่สร้างขึ้นไปเปรียบเทียบกับค่าบรรทัดฐาน ซึ่งเป็นค่าเชิงปริมาณที่ได้จากการต่อสู้กับศัตรูที่มีรูปแบบพฤติกรรมที่ยอมรับได้และคล้ายคลึงกับรูปแบบพฤติกรรมที่นำมาเปรียบเทียบ หากค่าทั้งสองแตกต่างกันไม่เกิน 0.1 หรืออีกนัยหนึ่งคือประสิทธิภาพแตกต่างกันไม่เกิน 10% งานวิจัยนี้จึงถือว่า รูปแบบพฤติกรรมที่สร้างขึ้นจากขั้นตอนวิธีนั้นสามารถยอมรับได้ งานวิทยานิพนธ์นี้จะหาค่าบรรทัดฐานโดยใช้รูปแบบพฤติกรรมจากชุดข้อมูลในหัวข้อ 7.3.1 ซึ่งเป็นชุดข้อมูลจากหัวข้อ 5.1 ที่ผ่านการคัดกรองมาแล้ว และวัดความคล้ายคลึงของรูปแบบพฤติกรรมด้วยการวิเคราะห์จัดกลุ่ม รายละเอียดจะกล่าวถึงในหัวข้อ 7.5.1

7.2 เกมทดสอบ

เกมทดสอบในงานวิจัยนี้จะใช้สำหรับเก็บข้อมูลการต่อสู้ระหว่างตัวละครอวตารและศัตรูที่มีรูปแบบพฤติกรรมที่สร้างขึ้นด้วยขั้นตอนวิธีที่นำเสนอ เพื่อนำข้อมูลดังกล่าวไปประเมินการยอมรับได้ของรูปแบบพฤติกรรมที่สร้าง อันจะถูกนำไปใช้ประเมินประสิทธิผลของขั้นตอนวิธี ดังนั้น เกมทดสอบที่จัดทำขึ้นจำเป็นต้องมีคุณสมบัติดังนี้

- เข้าใจแบบจำลองภาคีซึ่งอยู่ในรูปภาษาอธิบายภาคี สามารถแสดงภาคีให้ผู้เล่นมองเห็น และแสดงผลรูปแบบพฤติกรรมได้ถูกต้องตามนิยามที่กำหนดไว้ในแบบจำลอง
- เข้าใจองค์ประกอบที่เกี่ยวข้องกับศัตรู สามารถแสดงให้ผู้เล่นมองเห็น และแสดงผลกระทบต่อภาคีได้ถูกต้องตามนิยามที่กำหนดไว้
- สามารถเก็บข้อมูลการต่อสู้ได้ โดยข้อมูลการต่อสู้ระหว่างตัวละครอวตารและศัตรูจะต้องนำมาใช้คำนวณหา พลังชีวิตที่เหลืออยู่ของศัตรู และอัตราหลบหลีกพลาดได้

งานวิทยานิพนธ์นี้ใช้ libgdx ซึ่งเป็นข่ายงานสำหรับพัฒนาเกมข้ามแพลทฟอร์มด้วยภาษาจาวา สำหรับสร้างเกมทดสอบ ตัวอย่างเกมทดสอบเป็นไปดังรูปที่ 69



รูปที่ 69 เกมทดสอบ

7.2.1 แบบจำลองภาคีและองค์ประกอบที่เกี่ยวข้องในเกมทดสอบ

เพื่อให้เกมทดสอบเข้าใจแบบจำลองภาคีและองค์ประกอบที่เกี่ยวข้องกับศัตรูนั้น งานวิทยานิพนธ์สร้างเกมทดสอบขึ้นอิงกับรายละเอียดของแบบจำลอง ความสัมพันธ์ระหว่างภาคีและองค์ประกอบที่เกี่ยวข้อง รวมถึงข้อกำหนดที่ระบุไว้ในบทที่ 4 ส่วนอื่น ๆ ที่เพิ่มเติมขึ้นมามีรายละเอียดดังนี้

แบบจำลองภาคี

- เพิ่มค่าคุณสมบัติพิเศษ "texture" สำหรับระบุรูปภาพที่จะใช้แสดงภาคีเพื่อให้สามารถแสดงผลภาคีแก่ผู้เล่นได้ และเพิ่มความสามารถในการตั้งค่าคุณสมบัติดังกล่าวนี้ผ่านฟังก์ชั่นการกระทำ Set
- หากภาคีที่ไม่ใช่กระสุนมีค่าคุณสมบัติคงกระพัน รูปภาพที่แสดงผลจะเปลี่ยนไปใช้รูปพิเศษ
- หากภาคีไม่มีค่าคุณสมบัติฝ่ายป้องกัน รูปภาพของภาคีจะถูกทำให้โปร่งใสเพื่อให้ผู้เล่นมนุษย์รับรู้ได้

โครงสร้างพื้นที่

- พื้นที่ในเกมทดสอบเป็นห้องสี่เหลี่ยมขนาด 640*576 พิกเซล มีกำแพงสองด้าน พื้น และเพดานเป็นบริเวณที่ผ่านไม่ได้ หนา 32 พิกเซล
- แสดงบริเวณที่ผ่านไม่ได้ด้วยรูปภาพที่แตกต่างจากภาคีและตัวละครอวตาร
- มีแรงดึงดูดในทิศลง 10 พิกเซลต่อเฟรม คอยดึงให้ภาคีและตัวละครอวตารที่ได้รับผลของแรงดึงดูดลงทุกเฟรม

กลไกการต่อสู้

- กรณีที่ภาคีฝ่ายโจมตีเข้าปะทะกับภาคีฝ่ายป้องกันที่มีคุณสมบัติคงกระพัน ให้เล่นเสียงประกอบเพื่อแจ้งเตือนให้ผู้เล่นรู้
- ภาคีที่ถูกระบุให้เป็นศัตรูระดับ Miniboss และ Boss เมื่อถูกโจมตี จะอยู่ในสภาพคงกระพัน 30 เฟรม รายละเอียดของระดับของศัตรูจะกล่าวถึงอีกครั้งในหัวข้อ 7.3.1
- ตัวละครอวตาร เมื่อถูกโจมตีจะอยู่ในสภาพคงกระพันเป็นระยะเวลา 60 เฟรม แสดงผลด้วยการกะพริบภาพตัวละคร
- สภาพคงกระพันมีลักษณะเดียวกันกับคุณสมบัติคงกระพัน ซึ่งส่งผลให้ภาคีหรือตัวละครอวตารไม่ได้รับความเสียหายในขณะที่ยังอยู่ในสภาพดังกล่าวนี้

ตัวละครอวตาร

- แสดงด้วยรูปภาพที่แตกต่างจากโครงสร้างพื้นที่และภาคีอื่น ๆ
- พลังชีวิตเริ่มต้น 20 หน่วย
- ตัวตรวจจับการชนมีขนาด 32*48 พิกเซล
- กดปุ่มซ้ายหรือขวาเพื่อเดินไปทางซ้ายหรือขวาด้วยความเร็ว 5 พิกเซลต่อเฟรม
- กดปุ่ม Z เพื่อกระโดดด้วยความเร็ว 2.7 พิกเซลต่อเฟรม ค่าความเร็วนี้ส่งผลทุกเฟรมตราบเท่าที่ยังกดปุ่มค้างเอาไว้เป็นระยะเวลาสูงสุด 7 เฟรม

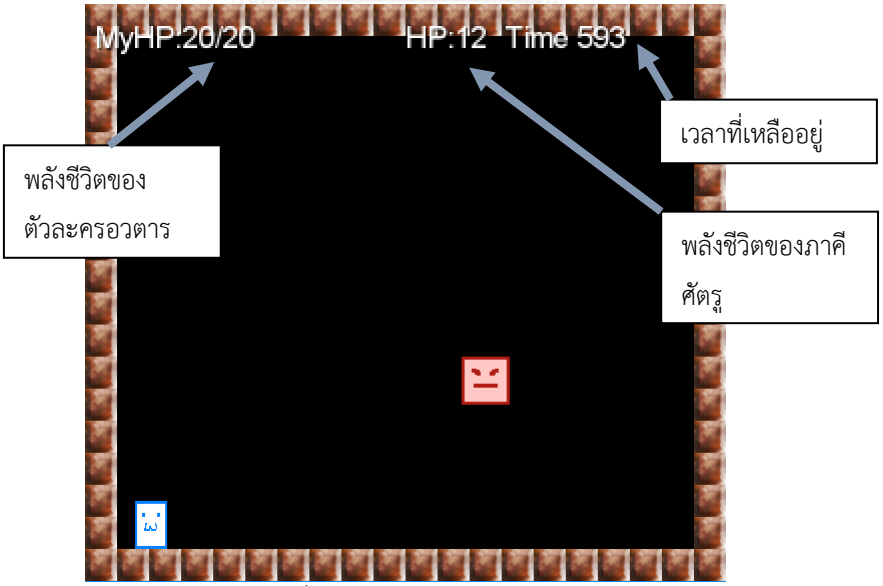
- กดปุ่ม X เพื่อยิงภาคีกระสุนในทิศที่หันหน้าอยู่ กระสุนมีขนาด 10*10 พิกเซล และเคลื่อนที่เป็นเส้นตรงด้วยความเร็ว 10 พิกเซลต่อเฟรม โดยเริ่มเคลื่อนที่จากจุดศูนย์กลางตัวละครอวตาร

7.2.2 วิธีเก็บข้อมูลการต่อสู้และคำนวณข้อมูลเพื่อประเมินการยอมรับได้

การต่อสู้เพื่อทำการเก็บข้อมูลจะแบ่งออกเป็นรอบ ในแต่ละรอบตัวละครอวตารจะต้องต่อสู้กับศัตรู 1 ตัว เมื่อการต่อสู้เริ่มขึ้น ตัวละครอวตารจะถูกสร้างไว้ที่ตำแหน่ง (50,500) เสมอ ส่วนตำแหน่งของภาคีศัตรูจะเป็นไปตามที่กำหนดไว้ในบล็อกเริ่มต้นโดยใช้ฟังก์ชั่นการกระทำ Set ตัวละครอวตารจะไม่สามารถโจมตีได้ในระยะเวลา 120 เฟรม (ประมาณ 2 วินาที) เพื่อป้องกันไม่ให้ผู้เล่นสามารถเอาชนะศัตรูได้ก่อนที่ศัตรูจะมีโอกาสจู่โจม โดยเฉพาะในกรณีที่ศัตรูมีพลังชีวิตเริ่มต้นน้อยมาก การต่อสู้จะจบลงเมื่อตัวละครอวตารหรือภาคีศัตรูสูญเสียพลังชีวิตทั้งหมด หรือหมดเวลาต่อสู้หากการต่อสู้นั้นมีการจำกัดเวลา ในกรณีที่ภาคีศัตรูสูญเสียพลังชีวิตทั้งหมด เกมจะไม่จบลงทันที แต่จะมีระยะเวลา 100 เฟรมก่อนจบการต่อสู้ เพื่อให้ผู้เล่นหลบหลีกภาคีกระสุนที่หลงเหลืออยู่ของศัตรูที่มีโอกาสเข้าถึงตัว

ในกรณีที่มีเหตุให้ศัตรูไม่สามารถปราบได้ โดยที่การต่อสู้ครั้งดังกล่าวไม่มีการจำกัดเวลาต่อสู้ เช่น ศัตรูเคลื่อนที่ออกจากบริเวณที่ใช้ต่อสู้ หรือผู้เล่นพิจารณาแล้วว่าศัตรูตัวดังกล่าวไม่สามารถปราบได้ ผู้เล่นสามารถข้ามการต่อสู้รอบดังกล่าวได้โดยการกดปุ่มสำหรับข้ามการต่อสู้

บนหน้าจอเกมจะแสดงข้อมูลที่สำคัญในการต่อสู้เพื่อเป็นข้อมูลแก่ผู้เล่น ประกอบด้วย พลังชีวิตของตัวละครอวตารและภาคีศัตรู ข้อความเตือนเมื่อตัวละครอวตารยังไม่สามารถโจมตีได้ในช่วงเริ่มการต่อสู้ และเวลาที่เหลืออยู่ (ในกรณีที่มีการจำกัดเวลาต่อสู้) รูปที่ 70 แสดงตัวอย่างข้อมูลบนหน้าจอเกม



รูปที่ 70 ข้อมูลต่าง ๆ บนหน้าจอเกม

การต่อสู้แต่ละรอบจะมีการบันทึกข้อมูลของทุกภาคีที่เกิดขึ้นระหว่างการต่อสู้ ในที่นี้เรียกว่าเซ็ตของภาคีที่เกิดขึ้นทั้งหมด E และข้อมูลเสริมอื่น ๆ ได้แก่ ตัวละครอวตารตายหรือไม่ การต่อสู้ครั้งนี้ถูกข้ามหรือไม่ สำหรับสิ่งที่ทำการบันทึกของแต่ละภาคี $e,\ \ e \in E$ มีรายละเอียดดังนี้

- ภาคีนี้เป็นภาคีศัตรูหรือไม่ เพื่อแยกความแตกต่างระหว่างศัตรูของการต่อสู้รอบดังกล่าว และภาคีกระสุนของศัตรู
- พลังชีวิตเริ่มต้นของภาคี e_{MaxHP}
- พลังชีวิตที่เหลืออยู่ของภาคี e_{HP}
- หมายเลขเฟรมที่ภาคีถูกสร้างขึ้นในโลกของเกม e_{Spawn}
- จำนวนเฟรมทั้งหมดที่ภาคีอยู่ในโลกของเกม $e_{Lifetime}$
- เซ็ตอันดับของระยะห่างจากภาคีไปยังตัวละครอวตาร $e_{Distance} = \{d_1, d_2, ..., d_i \mid 1 \le i \le e_L\}$ เมื่อ d_i คือ ค่าระยะห่างที่เฟรมที่ i
- เซ็ตอันดับของเฟรมที่ภาคีชนกับตัวละครอวตาร $e_{Collide}$ (สนใจเฉพาะกรณีที่ภาคี e มีค่าคุณสมบัติเป็นฝ่ายศัตรูและฝ่ายโจมตี)
- เซ็ตของเฟรมที่ภาคีได้รับความเสียหาย $e_{Damaged}$

จากข้อมูลข้างต้นสามารถใช้ในการคำนวณค่าต่าง ๆ ที่สนใจสำหรับการต่อสู้ครั้งใด ๆ ดังต่อไปนี้

ระยะเวลาปราบศัตรู

ซึ่งหมายถึงระยะเวลาที่ใช้จนกระทั่งปราบศัตรูได้สำเร็จ แบ่งการคำนวณออกเป็น 2 กรณี ดังนี้

กรณีปกติ

กำหนดภาคีศัตรูที่เข้าสู่โลกของเกมพร้อมกับตัวละครอวตารเมื่อการต่อสู้เริ่มขึ้น $m,\ \ m \in E$ สามารถคำนวณระยะเวลาปราบศัตรูกรณีที่ตัวละครอวตารปราบศัตรูสำเร็จ $Duration$ ด้วย

$$Duration = \max_{e \in E}(LF(e))$$

เมื่อฟังก์ชั่นคำนวณหมายเลขเฟรมแรกสุดหลังจากที่ภาคี e ถูกนำออกจากโลกของเกม $LF(e) = e_S + e_L$

ค่า $Duration$ ที่ได้นอกจากจะเป็นระยะเวลาที่ใช้ปราบศัตรูแล้ว ยังสามารถใช้เป็นระยะเวลาต่อสู้ได้ด้วย

กรณีที่ตัวละครอวตารสูญเสียพลังชีวิตทั้งหมดก่อนจะปราบศัตรูสำเร็จ

สามารถคำนวณระยะเวลาปราบศัตรู $Duration'$ โดยใช้วิธีคาดการณ์ระยะเวลาที่การต่อสู้น่าจะจบลงโดยอ้างอิงจากระยะเวลาต่อสู้ที่ใช้ ณ ปัจจุบัน ด้วย

$$Duration' = \frac{Duration \cdot m_{MaxHP}}{m_{MaxHP} - m_{HP}}$$

ในกรณีที่ศัตรูไม่สามารถปราบได้ด้วยเหตุผลอื่น ๆ นอกเหนือจากกรณีที่ตัวละครอวตารสูญเสียพลังชีวิตทั้งหมดก่อนปราบได้สำเร็จ ระยะเวลาปราบศัตรูสำหรับข้อมูลการต่อสู้ดังกล่าวจะไม่ถูกนำมาใช้งาน

อัตราหลบหลีกพลาด

กำหนดค่าคงที่ระยะที่มีความสำคัญ (Relevant range) Ran อันเป็นค่าที่ใช้กรองภาคีฝ่ายศัตรูที่ไม่เคยเข้ามาใกล้ตัวละครอวตารออกเพื่อไม่นำมาคิดอัตราหลบหลีกพลาด สามารถคำนวณอัตราหลบหลีกพลาด $MissRate$ ด้วย

$$MissRate = \begin{cases} \dfrac{\sum_{re \in RE} \dfrac{Hit(re)}{MaxHit(re)}}{|RE|} & , RE \neq \phi \\ 0 & , RE = \phi \end{cases}$$

เมื่อ

- เซ็ตของภาคีที่มีความสำคัญ $RE = \{re \in E \mid \exists d \in re_{Distance}.(d \leq Ran)\}$
- ฟังก์ชั่นคำนวณจำนวนครั้งสูงสุดที่มีโอกาสโดนโจมตีเมื่อสนใจเฉพาะภาคี e $MaxHit(e) = \dfrac{e_L}{Invul}$ ซึ่งนับจากจำนวนครั้งที่ตัวละครอวตารมีโอกาสชนกับภาคี e เมื่อกำหนดให้การชนเริ่มตั้งแต่เฟรมแรกที่ภาคี e เข้าสู่โลกของเกมและเกิดการชนทุกครั้งที่ตัวละครอวตารพ้นจากสถานะคงกระพัน
- ฟังก์ชั่นคำนวณจำนวนครั้งที่โดนโจมตีเมื่อสนใจเฉพาะภาคี e $Hit(e)$ คำนวณโดยใช้ขั้นตอนด้านล่าง ซึ่งเป็นการนับการชนที่เกิดขึ้นในขณะที่ตัวละครอวตารไม่ได้อยู่ในสถานะคงกระพัน เมื่อพิจารณาเฉพาะสถานะคงกระพันที่เกิดขึ้นจากการชนกับ e

```
Initialize HitCount = 0, LastHitFrame = −1
For(c_i in e_collide) do:
    If(c_i − LastHitFrame ≥ Invul) Then
        HitCount = HitCount + 1
        LastHitFrame = c_i
    End
End
Return HitCount as result
```

- $Invul$ เป็นระยะเวลาที่ตัวละครอวตารอยู่ในสถานะคงกระพันหลังถูกโจมตี ซึ่งเกมทดสอบกำหนดค่าไว้ที่ 60 เฟรมดังที่ระบุไว้ในหัวข้อ 7.2.1

โดยการคำนวณอัตราหลบหลีกพลาดนำเอา $Invul$ เข้ามาคำนวณเพื่อให้สนใจเฉพาะการโจมตีที่เกิดขึ้นในช่วงที่ตัวละครอวตารไม่ได้อยู่ในสถานะคงกระพัน เนื่องตัวละครอวตารจะไม่ได้รับความเสียหายขณะที่อยู่ในสถานะคงกระพัน การถูกโจมตีในช่วงเวลานี้อาจเกิดจากความตั้งใจของผู้ควบคุมตัวละครเองได้ เช่น ผู้เล่นตั้งใจวิ่งฝ่ากระสุนจำนวนมากไปยังบริเวณที่ปลอดภัยในขณะที่ยังอยู่ในสถานะคงกระพัน เป็นต้น ซึ่งการถูกโจมตีดังกล่าวนี้ไม่ควรนับเป็นการหลบหลีกพลาด

พลังชีวิตที่เหลืออยู่

กำหนดค่าคงที่เวลาต่อสู้จำกัด (Capped battle duration) $DurCap$ อันเป็นค่าที่ใช้ระบุว่าการคำนวณพลังชีวิตที่เหลืออยู่จะคำนวณตั้งแต่เริ่มการต่อสู้จนถึงระยะเวลาที่กำหนดเท่านั้น ถึงแม้การต่อสู้จริงจะยาวนานกว่านั้น และภาคีศัตรูที่เข้าสู่โลกของเกมพร้อมกับตัวละครอวตารเมื่อการต่อสู้เริ่มขึ้น $m,\ m \in E$ สามารถคำนวณพลังชีวิตที่เหลืออยู่ $\mathrm{Re}\,mainingHP$ ด้วย

$$\mathrm{Re}\,mainingHP = 1 - \frac{\left|\{h \mid h \in m_{Damaged}, h < DurCap\}\right|}{m_{MaxHP}}$$

ซึ่งการคำนวณไม่ใช้ m_{HP} แต่นับจำนวนครั้งที่ได้รับความเสียหายแทน เพื่อให้สามารถคำนวณพลังชีวิตที่เหลืออยู่ ณ เฟรมใดก็ได้ ไม่จำเป็นต้องเป็นพลังชีวิตที่เหลืออยู่เมื่อจบการต่อสู้ ซึ่งจะมีค่า 0 เสมอหากผู้เล่นปราบศัตรูได้สำเร็จ ในกรณีศัตรูไม่สามารถปราบได้ด้วยสาเหตุใด ๆ นอกเหนือจากการที่ตัวละครอวตารสูญเสียพลังชีวิตทั้งหมดก่อนปราบศัตรูสำเร็จ พลังชีวิตที่เหลืออยู่สำหรับข้อมูลการต่อสู้ดังกล่าวจะไม่ถูกนำมาใช้งาน

7.2.3 วิธีเก็บข้อมูลเพื่อหาระยะที่มีความสำคัญ

ระยะที่มีความสำคัญ คือ ระยะห่างระหว่างตัวละครอวตารและภาคีฝ่ายศัตรูที่ผู้เล่นจะเริ่มให้ความสำคัญและตัดสินใจหลบ ระยะดังกล่าวนี้มีบทบาทสำคัญในการคำนวณอัตราหลบหลีกพลาด เพื่อใช้คัดกรองภาคีฝ่ายศัตรูบางตัวที่อยู่ในระยะที่ไกลเกินกว่าที่ผู้เล่นจะสนใจ เกมทดสอบที่จัดทำขึ้นจึงมีโหมดพิเศษเพื่อหาค่าดังกล่าวนี้โดยกลไกต่าง ๆ ยังคงเป็นไปตามหัวข้อ 7.2.1

เนื่องจากความเร็วในการเคลื่อนที่ของภาคีเป็นไปได้หลายค่า และที่ความเร็วต่าง ๆ ผู้เล่นน่าจะมีระยะตัดสินใจที่แตกต่างกันไป งานวิทยานิพนธ์นี้จึงเลือกที่จะหาระยะนี้ด้วยการเฉลี่ยจากระยะตอบสนองต่อภาคีที่ความเร็วต่าง ๆ โดยมีรายละเอียดดังนี้

- ภาคีศัตรูมี 4 ชนิด ทุกชนิดมีพฤติกรรมเหมือนกัน คือ รอเวลาอย่างสุ่ม (ตั้งแต่ 80 ถึง 140 เฟรม) แล้วยิงกระสุน 1 ครั้งในทิศที่ตัวละครอวตารอยู่ โดยศัตรูแต่ละชนิดจะยิงกระสุนที่ความเร็วแตกต่างกัน ได้แก่ 2, 6, 10 และ 14 พิกเซลต่อเฟรม ตามลำดับ
- ผู้เล่นจะต้องควบคุมให้ตัวละครอวตารกระโดดข้ามกระสุนที่ยิงเข้าใส่ โดยการควบคุมทำได้เพียงเดินไปทางซ้ายและขวา และสามารถกระโดดได้ 1 ครั้ง
- หากตัวละครอวตารหลบพลาด จะถือว่าการเก็บข้อมูลครั้งนั้นผิดพลาด และถือว่า “ไม่ผ่าน” การต่อสู้นั้น
- หากหลบสำเร็จ ข้อมูลระยะห่างระหว่างตัวละครอวตารและกระสุนขณะที่ตัวละครอวตารเริ่มกระโดดจะถูกบันทึกไว้ และถือว่า “ผ่าน” การต่อสู้นั้น
- ผู้เล่นจะต้องเผชิญหน้ากับศัตรูทั้ง 4 ชนิด ชนิดละ 6 ครั้ง โดยลำดับการเผชิญหน้าจะเป็นแบบสุ่ม เมื่อผู้เล่นผ่านการต่อสู้ทั้งหมด การเก็บข้อมูลจะจบลง
- ระยะที่มีความสำคัญสามารถคำนวณได้จากข้อมูลของผู้เล่นทุกคนที่เก็บดังนี้

กำหนดเซ็ตของข้อมูลที่เก็บจากผู้เล่น n คน $Data = \{p_1, p_2, ..., p_n\}$ และข้อมูลที่เก็บจากผู้เล่นคนหนึ่ง $P = \{r_1, r_2, ..., r_{24}\}$, $P \in Data$ ซึ่งประกอบด้วยระยะห่าง r, $r \in P$ ที่บันทึกไว้ทั้งหมด 24 ครั้ง สามารถคำนวณระยะที่มีความสำคัญ Ran ด้วย

$$Ran = \frac{\sum_{P \in Data} \sum_{r \in P} r}{24|Data|}$$

7.3 การเก็บข้อมูลผู้เล่น

ข้อมูลของผู้เล่นจะถูกนำมาใช้สำหรับวัดประสิทธิภาพของปัญญาประดิษฐ์ รวมถึงใช้เป็นค่าบรรทัดฐานเพื่อวัดการยอมรับได้ของศัตรูที่สร้างขึ้นด้วยขั้นตอนวิธีที่นำเสนอ การเก็บข้อมูลผู้เล่นจะแบ่งออกเป็น 2 ส่วน ได้แก่ การเก็บข้อมูลการต่อสู้ และการหาระยะที่มีความสำคัญ การเก็บข้อมูลการต่อสู้จะใช้ศัตรูจากชุดข้อมูลศัตรูเดียวกับที่ใช้ในการทำเหมืองข้อมูลตามหัวข้อ 5.1 โดยมีการจัดหมวดหมู่ศัตรูและคัดศัตรูบางส่วนออกเนื่องจากกลไกของเกมทดสอบไม่เอื้ออำนวยต่อการใช้งานศัตรูดังกล่าว

7.3.1 การเตรียมชุดข้อมูลศัตรู

ศัตรูในแต่ละเกมที่นำมาใช้เป็นชุดข้อมูลมีความยากง่ายในการปราบแตกต่างกัน ส่งผลให้ประสิทธิภาพที่คำนวณได้จากข้อมูลการต่อสู้มีแนวโน้มจะแตกต่างกันไปด้วยสำหรับศัตรูที่มีระดับความยากแตกต่างกัน เนื่องจากวิธีการวัดผลจะนำเอาการเปรียบเทียบประสิทธิภาพมาใช้ การเปรียบเทียบผลการต่อสู้ที่ได้จากศัตรูหลายระดับความยากปะปนกันจึงอาจเกิดความโน้มเอียงได้ ในที่นี้จึงจำเป็นต้องแบ่งกลุ่มศัตรูตามระดับความยากก่อน

ความยากของศัตรูอาจเกิดจากปริมาณพลังชีวิตหรือความซับซ้อนของรูปแบบพฤติกรรมก็ได้ จากการวิเคราะห์ศัตรูในเกมแอ็คชั่นทั้ง 5 เกม พบว่า ในแต่ละเกม ปริมาณพลังชีวิตและความซับซ้อนของศัตรูมีความแปรผันตามกัน ศัตรูที่มีพลังชีวิตมาก ก็มีแนวโน้มที่รูปแบบพฤติกรรมจะมีความซับซ้อน ใช้เวลาในการแสดงพฤติกรรมเป็นระยะเวลานานกว่าที่รูปแบบพฤติกรรมจะกลับมาซ้ำเดิม และยังสัมพันธ์กับการปรากฏตัวภายในเกมอีกด้วย การแบ่งกลุ่มศัตรูจึงใช้หลักเกณฑ์ดังกล่าว แบ่งศัตรูออกเป็น 4 ระดับที่มีรายละเอียดดังนี้

1) ระดับ Enemy เป็นศัตรูที่พบได้ทั่วไปในฉาก มีพลังชีวิตน้อย พฤติกรรมไม่ซับซ้อน แต่ก็มีศัตรูบางตัวที่มีพฤติกรรมไม่ซับซ้อน แต่ถูกออกแบบให้ปราบไม่ได้และต้องหลบหลีกอย่างเดียว เช่น Garyoby จาก Megaman 4 เป็นต้น
2) ระดับ Elite เป็นศัตรูที่พบได้ในฉาก มีพลังชีวิตมากกว่าศัตรูระดับ Enemy และมีความถี่ในการปรากฏตัวน้อยกว่า Enemy อาจถูกออกแบบให้ปราบไม่ได้และต้องหลบหลีกเท่านั้น
3) ระดับ Miniboss เป็นศัตรูที่พบได้ในบางฉากเท่านั้น ในฉากหนึ่งพบได้ 1 – 2 ตัว มีพลังชีวิตมาก และมีพฤติกรรมซับซ้อนสูงกว่าศัตรู 2 ระดับแรก เป็นศัตรูที่ต้องปราบให้ได้เพื่อเดินหน้าไปต่อ
4) ระดับ Boss เป็นศัตรูที่พบเมื่อจบฉาก โดยส่วนใหญ่จะมีเพียง 1 ตัว และส่วนใหญ่มีพลังชีวิตที่มากกว่าศัตรูกลุ่ม Miniboss ภายในเกมเดียวกัน อาจมีความซับซ้อนของพฤติกรรมใกล้เคียงกับ Miniboss หรือสูงกว่า เป็นศัตรูที่ต้องปราบให้ได้เพื่อผ่านฉาก

ชุดข้อมูลศัตรูจะถูกแบ่งออกเป็น 4 ระดับข้างต้นโดยใช้เกณฑ์ข้างต้น โดยใช้ค่าพลังชีวิตของศัตรูในการแบ่งกลุ่มเป็นหลัก โดยอิงมาตรฐานค่าพลังชีวิตของชนิดศัตรูตามเกม Megaman และ Megaman 4 ดังนี้

- ระดับ Enemy คือศัตรูที่มีพลังชีวิต 1 – 4
- ระดับ Elite คือศัตรูที่มีพลังชีวิต 5 - 10
- ระดับ Miniboss คือศัตรูที่มีพลังชีวิต 11 – 27 ศัตรูบางตัวอาจมีพลังชีวิตนอกช่วงค่าได้ เนื่องจากความแตกต่างของมาตรฐานพลังชีวิต โดยจะคำนึงถึงคุณลักษณะอื่นของความเป็น Miniboss ด้วย เช่น ป้อมปืนที่ปรากฏกลางฉากที่ 3 ของ Super C มีพลังชีวิตมากถึง 32 แต่ถูกนับเป็น Miniboss เนื่องจากความซับซ้อนไม่สูง (ทำเพียงยิงกระสุนรูปแบบเดียวใส่ผู้เล่น) และปรากฏระหว่างฉาก
- ระดับ Boss คือศัตรูที่มีพลังชีวิตตั้งแต่ 28 ขึ้นไป ศัตรูบางตัวอาจมีพลังชีวิตนอกช่วงค่า เนื่องจากความแตกต่างของมาตรฐานพลังชีวิตในแต่ละเกม โดยจะคำนึงถึงบทบาทและคุณลักษณะอื่นร่วม เช่น ศัตรูระดับ Boss ในเกม Shovel knight ที่มีพลังชีวิตในช่วง 12 – 20 แต่สามารถนับเป็น Boss เนื่องจากบทบาทในเนื้อเรื่อง การปรากฏตัว และความซับซ้อนของพฤติกรรม

การแบ่งกลุ่มชุดข้อมูลศัตรูจะใช้วิธีแยกไฟล์สคริปต์ออกไปไว้ในไดเร็คทอรี่ที่แตกต่างกันตามกลุ่ม และปรับปรุงให้เกมทดสอบสามารถรู้ว่าไฟล์สคริปต์ศัตรูที่ดึงมาเป็นศัตรูระดับใดโดยดูจากชื่อไดเร็คทอรี่ โดยไม่มีการยุ่งเกี่ยวกับสคริปต์แต่อย่างใด

หลังจากแบ่งกลุ่มชุดข้อมูลแล้ว ผู้วิจัยได้เลือกเอาศัตรูบางส่วนออก โดยมีสาเหตุการคัดออกและศัตรูที่ถูกคัดออก ดังนี้

- ศัตรูบางส่วนถูกคัดออกเนื่องจากเป็นศัตรูที่ถูกออกแบบมาให้เล่นภายใต้กลไกของเกมที่มีความแตกต่างไปจากเกมทดสอบ ซึ่งความแตกต่างดังกล่าวส่งผลให้ผู้เล่นไม่สามารถปราบศัตรูดังกล่าวได้ทั้งที่ไม่ใช่ความตั้งใจของการออกแบบนั้น หรืออาจส่งผลให้เกิดความโน้มเอียงของประสิทธิภาพที่คำนวณได้จากข้อมูลการต่อสู้ หากนำศัตรูกลุ่มนี้มาใช้สำหรับการวัดผลด้วย ศัตรูในกลุ่มนี้ประกอบด้วย
  - ศัตรูจากทุกฉากของเกม Super C ที่เป็นการเล่นแบบมุมมองด้านบน ซึ่งตัวละครอวตารสามารถเดินได้ 4 ทิศทาง (ขึ้น ลง ซ้าย ขวา) โดยไม่มีแรงดึงดูดมาเกี่ยวข้อง แตกต่างจากกลไกที่กำหนดไว้ในเกมทดสอบ ที่ตัวละครอวตารสามารถเคลื่อนที่ได้ 2 ทิศทาง (ซ้าย ขวา) และสามารถกระโดดเพื่อเคลื่อนที่ในแนวดิ่ง
  - ศัตรูบางส่วนของเกม Super C และ Metroid ที่ถูกออกแบบให้ตัวละครต้องโจมตีขึ้นด้านบนเพื่อปราบศัตรูดังกล่าว ซึ่งไม่ใช่พฤติกรรมที่ตัวละครอวตารสามารถทำได้ในเกมทดสอบ
- ศัตรูที่ออกแบบไว้อิงกับโครงสร้างพื้นที่เฉพาะที่แตกต่างไปจากเกมทดสอบ พบในศัตรู Lava dragon ของเกม Metroid
- ศัตรูที่ต้องใช้วิธีรับมือเฉพาะ ซึ่งน่าจะส่งผลให้เกิดความแตกต่างของประสิทธิภาพในการต่อสู้ระหว่างผู้เล่นที่จับจุดได้และผู้เล่นที่จับจุดไม่ได้มากเกินไป พบในศัตรู Toad man ของเกม Megaman 4
- ศัตรูที่ออกแบบไว้ให้เอาชนะได้ง่ายเกินไปเนื่องจากในเกมต้นฉบับใช้จำนวนของศัตรูในการสร้างความท้าทาย พบในศัตรูส่วนหนึ่งของเกม Super C

เมื่อทำการคัดศัตรูบางส่วนออกแล้ว จึงทำการทดสอบเบื้องต้นกับชุดข้อมูล โดยผู้วิจัยทำการทดลองเก็บข้อมูลเบื้องต้นกับผู้เล่น 2 คน แต่ละคนจะต้องต่อสู้กับศัตรูจากทั้ง 4 ระดับอย่างสุ่ม เป็นศัตรูระดับ Enemy 5 ตัว, ระดับ Elite 4 ตัว, ระดับ Miniboss 3 ตัว และระดับ Boss 3 ตัว หากผู้เล่นข้ามการต่อสู้ใด จะไม่นับว่าได้ต่อสู้กับศัตรูตัวดังกล่าว

จากการทดลองพบว่า กรณีที่ผู้เล่นต้องต่อสู้กับศัตรูที่ถูกออกแบบให้ปราบไม่ได้ ผู้เล่นอาจใช้เวลาไม่เท่ากันในการตัดสินใจข้ามการต่อสู้ ซึ่งอาจส่งผลให้เกิดความโน้มเอียงของประสิทธิภาพที่คำนวณจากข้อมูลการต่อสู้ได้ เนื่องจากระยะเวลาต่อสู้ที่ยาวนานกว่าย่อมส่งผลให้ผู้เล่นมีโอกาสหลบหลีกการโจมตีพลาดได้มากกว่า ผู้วิจัยจึงตัดสินใจเปลี่ยนแปลงการทดสอบกับศัตรูประเภทดังกล่าว โดยให้การต่อสู้กับศัตรูประเภทที่ปราบไม่ได้นี้มีระยะเวลาต่อสู้จำกัด ประเด็นสำคัญต่อมาจึงเกี่ยวข้องกับการเลือกระยะเวลาต่อสู้ดังกล่าวนี้

จากการทดลองเดียวกัน พบว่าผู้ทดสอบใช้เวลาเฉลี่ยประมาณ 1800 เฟรม (ประมาณ 30 วินาที) ในการตัดสินใจข้ามการต่อสู้ เมื่อสอบถามความคิดเห็นว่าควรจะใช้ค่าดังกล่าวเป็นระยะเวลาการต่อสู้หรือไม่ ผู้ทดสอบเห็นพ้องว่าน่าจะปรับลดระยะเวลา ในที่นี้จึงทำการทดลองเดียวกันเพิ่มเติมกับผู้ทดสอบใหม่อีก 2 คน และสอบถามความคิดเห็นเกี่ยวกับระยะเวลาที่น่าจะใช้ และได้ข้อสรุปว่าประมาณครึ่งหนึ่งของเวลาเฉลี่ยเดิมน่าจะเหมาะสมกว่า ซึ่งมีค่าเท่ากับ 900 เฟรม (ประมาณ 15 วินาที)

โดยสรุปแล้ว ในการใช้งานชุดข้อมูลศัตรูต่อจากนี้ไป จะใช้ชุดข้อมูลศัตรูที่ผ่านการคัดและแบ่งกลุ่มแล้วในหัวข้อนี้ หากศัตรูในการต่อสู้รอบใดเป็นศัตรูที่ถูกออกแบบให้ปราบไม่ได้ จะใช้ระยะเวลาต่อสู้จำกัดที่ 900 เฟรม

7.3.2 รายละเอียดการเก็บข้อมูล

งานวิทยานิพนธ์นี้ทำการเก็บข้อมูลผู้เล่นเพื่อนำไปใช้สร้างปัญญาประดิษฐ์และวัดผลขั้นตอนวิธี ในขั้นตอนนี้ได้ทำการเก็บข้อมูลกับผู้เล่นทั้งหมดจำนวน 14 คน โดยแบ่งเป็นการเก็บข้อมูลเพื่อหาระยะที่มีความสำคัญตามหัวข้อ 7.2.3 และการเก็บข้อมูลการต่อสู้ โดยผู้เล่นแต่ละคนจะต้องต่อสู้กับศัตรูที่สุ่มมาจากชุดข้อมูลศัตรูที่เตรียมไว้ตามหัวข้อ 7.3.1 ซึ่งประกอบไปด้วยศัตรูระดับ Enemy 5 ตัว, ระดับ Elite 4 ตัว, ระดับ Miniboss 3 ตัว และระดับ Boss 3 ตัว ในกรณีที่ศัตรูถูกออกแบบให้ปราบไม่ได้ จะมีระยะเวลาต่อสู้จำกัดที่ 900 เฟรม และถ้าหากผู้เล่นพิจารณาแล้วว่าไม่สามารถปราบศัตรูได้ ผู้เล่นสามารถข้ามการต่อสู้ได้ โดยการต่อสู้รอบดังกล่าวจะไม่นับรวมกับจำนวนศัตรูทั้งหมดที่ผู้เล่นต้องต่อสู้ด้วย

ขั้นตอนการเก็บข้อมูลสำหรับผู้เล่น 1 คนเป็นไปดังนี้

1) อธิบายวิธีการควบคุมตัวละครอวตารและกฎของเกม
2) ผู้เล่นทดลองต่อสู้กับศัตรู Bright man ซึ่งเป็นศัตรูระดับ Boss จากชุดข้อมูลศัตรูที่ผู้วิจัยเลือก เนื่องจากรูปแบบพฤติกรรมไม่ซับซ้อน แต่มีการโจมตีที่ทำให้ผู้เล่นได้กดทุกปุ่มเพื่อควบคุมตัวละครอวตาร การต่อสู้ในรอบนี้ไม่มีการเก็บข้อมูลแต่อย่างใด แต่มีจุดประสงค์ให้ผู้เล่นคุ้นเคยกับวิธีควบคุมและกลไกของเกมเท่านั้น
3) อธิบายวิธีการเก็บข้อมูลเพื่อหาระยะที่มีความสำคัญ
4) เก็บข้อมูลเพื่อหาระยะที่มีความสำคัญ
5) อธิบายวิธีการเก็บข้อมูลการต่อสู้
6) เก็บข้อมูลการต่อสู้

7.4 ปัญญาประดิษฐ์สำหรับวัดผล

งานวิทยานิพนธ์นี้จำเป็นต้องทดลองต่อสู้กับศัตรูที่สร้างขึ้นจำนวนมาก การนำเอาผู้เล่นจริงมาทดสอบต่อสู้กับศัตรูครั้งละตัวจึงไม่เหมาะสมกับกรอบเวลาในการทำงาน งานวิทยานิพนธ์นี้จึงนำเอาปัญญาประดิษฐ์มาควบคุมตัวละครอวตารแทนผู้เล่นจริง โดยปัญญาประดิษฐ์ดังกล่าวนี้จะต้องเล่นได้มีประสิทธิภาพใกล้เคียงกับผู้เล่นจริง การสร้างปัญญาประดิษฐ์จึงต้องคำนึงถึงลักษณะเฉพาะในวิธีการเล่นและข้อจำกัดของผู้เล่นที่เป็นมนุษย์ด้วย งานวิทยานิพนธ์นี้สร้างปัญญาประดิษฐ์ 2 ตัว ได้แก่ ปัญญาประดิษฐ์ A และปัญญาประดิษฐ์ B โดยปัญญาประดิษฐ์ B แตกต่างจากปัญญาประดิษฐ์ A ในส่วนของฟังก์ชั่นวัตถุประสงค์ จากนั้นทำการวัดประสิทธิภาพของปัญญาประดิษฐ์โดยเทียบกับประสิทธิภาพของผู้เล่นจริง และเลือกปัญญาประดิษฐ์ที่มีประสิทธิภาพใกล้เคียงผู้เล่นจริงที่สุด

7.4.1 การออกแบบปัญญาประดิษฐ์ที่อิงกับพฤติกรรมและข้อจำกัดของผู้เล่นมนุษย์

งานวิทยานิพนธ์นี้ได้ให้ผู้เล่นจริงจำนวนหนึ่งต่อสู้กับศัตรูจากชุดข้อมูลศัตรูเพื่อเก็บข้อมูลประสิทธิภาพสำหรับวัดผลปัญญาประดิษฐ์ จากการสังเกตพฤติกรรมของผู้เล่นแล้ว พบว่ามีส่วนที่คล้ายคลึงกัน นั่นคือ ผู้เล่นจะพยายามหลบหลีกการโจมตีของศัตรู ไปพร้อมกับโจมตีใส่ด้วยหากเป็นไปได้ ในกรณีที่ผู้เล่นไม่สามารถลดพลังชีวิตศัตรูได้ ผู้เล่นจะทดลองโจมตีใส่เรื่อย ๆ เป็นระยะเวลาหนึ่งก่อนล้มเลิก และเปลี่ยนไปเคลื่อนที่อย่างสุ่มในฉากพร้อมกับหลบศัตรูไปด้วย อีกกรณีที่น่าสนใจ คือ กรณีที่ศัตรูซ่อนตัวอยู่ตั้งแต่เริ่มการต่อสู้ ทำให้ผู้เล่นมองไม่เห็นศัตรูเมื่อการต่อสู้เริ่มต้น ผู้เล่นส่วนใหญ่จะยืนนิ่งไม่กระทำการใดครู่หนึ่ง ก่อนเริ่มเดินสุ่มในฉากจนกระทั่งศัตรูปรากฏตัวให้เห็น จึงเริ่มการต่อสู้ พฤติกรรมเหล่านี้ส่งผลให้งานวิทยานิพนธ์นี้แบ่งปัญญาประดิษฐ์ออกเป็น 2 สถานะ โดยในสถานะเดินจะใช้สำหรับกรณีที่ปัญญาประดิษฐ์ไม่เห็นศัตรู เพื่อสุ่มเดินแบบเดียวกับผู้เล่น ส่วนสถานะต่อสู้จะถูกออกแบบให้พยายามหลบหลีกการโจมตีและโจมตีใส่ศัตรูไปด้วย

ในส่วนของข้อจำกัดของผู้เล่นมนุษย์นั้นจะเกี่ยวข้องกับความเร็วของการตอบโต้กับสภาพแวดล้อม ซึ่งเป็นปัจจัยหลักของเกมแอ็คชั่น มนุษย์จะไม่สามารถโต้ตอบกับสิ่งที่มองเห็นได้ในทันที แต่มีระยะเวลาสั้น ๆ ในการประมวลผลภาพเหตุการณ์ที่เห็นก่อน [39] แล้วจึงทำการโต้ตอบหลังจากนั้น ในกรณีของการโจมตีของศัตรูในเกมหากเป็นการโจมตีในระยะประชิด ผู้เล่นมักจะตอบโต้ไม่ทันเนื่องจากขีดจำกัดของตัวมนุษย์ ส่งผลให้ผู้เล่นส่วนใหญ่พยายามทิ้งระยะห่างจากศัตรูให้ไกลระดับหนึ่งเพื่อที่จะได้มีเวลาประมวลผลการโจมตีของศัตรู วิธีการประมวลผลที่ผู้เล่นใช้จะเป็นการคาดเดาทิศทางของการโจมตีโดยใช้ทิศทางการเคลื่อนที่ที่มองเห็น ร่วมกับฐานความรู้เดิมในกรณีที่เคยเห็นการโจมตีดังกล่าวมาก่อนแล้ว ปัญญาประดิษฐ์ในงานวิทยานิพนธ์นี้จึงนำเอาการคาดเดาทิศทางการโจมตีนี้มาใช้ในการตัดสินใจด้วย นอกจากนี้ ปัญญาประดิษฐ์ยังถูกจำกัดความถี่ในการตัดสินใจเปลี่ยนปุ่มที่กดอยู่ ให้เปลี่ยนได้ทุก ๆ 4 เฟรม เพื่อป้องกันไม่ให้ปุ่มกดเปลี่ยนถี่เกินไป ซึ่งมนุษย์ไม่สามารถทำได้

โดยสรุป ปัญญาประดิษฐ์ในงานวิทยานิพนธ์นี้ถูกออกแบบให้มีวิธีเล่นและมีข้อจำกัดคล้ายคลึงกับผู้เล่นมนุษย์ดังนี้

- ต่อสู้เมื่อสามารถทำได้ และเดินสุ่มเมื่อไม่เห็นศัตรู
- ตัดสินใจหลบหลีกโดยใช้วิธีคาดเดาทิศทางการโจมตี
- เปลี่ยนปุ่มกดได้ทุก 4 เฟรม

7.4.2 ปัญญาประดิษฐ์แบบเครื่องสถานะจำกัด

พฤติกรรมต่อสู้และเดินสุ่มเป็นพฤติกรรมที่แยกออกจากกันโดยสิ้นเชิง ในที่นี้จึงสร้างปัญญาประดิษฐ์โดยใช้เครื่องสถานะจำกัดที่มี 2 สถานะแยกตามพฤติกรรมทั้งสอง ได้แก่

- สถานะต่อสู้

  ใช้สำหรับต่อสู้และหลบหลีกศัตรู ปัญญาประดิษฐ์จะเลือกวิธีกดปุ่มที่ดีที่สุดสำหรับสถานการณ์ปัจจุบันโดยใช้ฟังก์ชั่นวัตถุประสงค์ ซึ่งประเมินคะแนนของการกดปุ่มโดยนำเอาการคาดเดาการโจมตีมาใช้เลือกวิธีกดปุ่มที่ทำให้ตัวละครอวตารปลอดภัยที่สุดและสามารถตอบโต้ศัตรูได้ ในกรณีที่ฟังก์ชั่นวัตถุประสงค์ไม่สามารถให้คำตอบที่ดีที่สุดได้ ปัญญาประดิษฐ์จะใช้กฎเพิ่มเติมในการตัดสินใจ

- สถานะเดิน

  ใช้ในกรณีที่ปัญญาประดิษฐ์ไม่เห็นศัตรู เป็นการบังคับให้ตัวละครอวตารเดินไปข้างหน้าจนสุดฉากแล้วเดินกลับ ทำเช่นนี้วนซ้ำไปมาเรื่อย ๆ

ปัญญาประดิษฐ์จะเริ่มในสถานะต่อสู้เสมอ และเมื่อเงื่อนไขเป็นจริง จะเปลี่ยนเป็นสถานะเดินได้ ซึ่งจะเปลี่ยนกลับเป็นสถานะต่อสู้อีกครั้งหนึ่งเมื่อเงื่อนไขการเปลี่ยนสถานะเป็นจริง สถานะของปัญญาประดิษฐ์จะไม่สามารถเปลี่ยนไปมาได้ทันที แต่ต้องอยู่ในสถานะปัจจุบันเป็นระยะเวลา 30 เฟรมก่อน จึงจะเปลี่ยนสถานะได้ การหน่วงเวลาเช่นนี้มีไว้เพื่อป้องกันไม่ให้สถานะเปลี่ยนเร็วเกินไป

เงื่อนไขเปลี่ยนสถานะจากสถานะต่อสู้เป็นสถานะเดิน

ขณะปัญญาประดิษฐ์อยู่ในสถานะต่อสู้ หากศัตรูยังมีพลังชีวิตเหลืออยู่ การตรวจสอบเงื่อนไขเปลี่ยนสถานะจะเริ่มเมื่อปัญญาประดิษฐ์ไม่สามารถสร้างความเสียหายให้กับศัตรูเป็นระยะเวลาติดต่อกันอย่างน้อย 240 เฟรม เงื่อนไขที่ใช้จะแบ่งเป็นกรณีที่ศัตรูยังอยู่ในหน้าจอและไม่อยู่ในในหน้าจอแล้ว

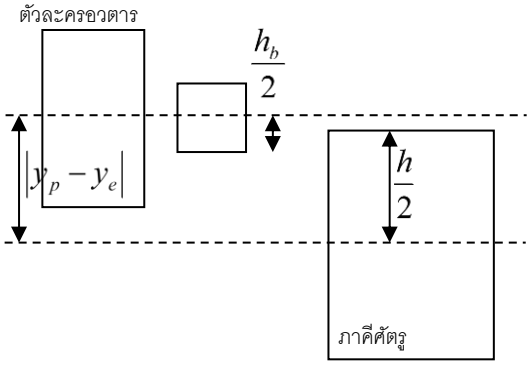
- กรณีศัตรูยังอยู่ในหน้าจอ เงื่อนไขการเปลี่ยนสถานะคือ ศัตรูอยู่ในสถานะคงกระพันเกิน 180 เฟรม หรืออยู่นอกระยะยิงเกิน 180 เฟรม โดยระยะยิงนิยามดังนี้

กำหนดให้ตัวละครอวตารอยู่ที่ตำแหน่ง (x_p, y_p) และภาคีศัตรูอยู่ที่ตำแหน่ง (x_e, y_e) และมีขนาด (w, h) ศัตรูจะถือว่าอยู่ในระยะยิงก็ต่อเมื่อนิพจน์แบบบูลต่อไปนี้เป็นจริงซึ่งเป็นการตรวจสอบว่ากระสุนที่ตัวละครอวตารยิงออกมาซ้อนทับกับภาคีศัตรูในแนวแกน y ดังรูปที่ *71*

$$\left| y_p - y_e \right| \leq \frac{(h + h_b)}{2}$$

โดย h_b เป็นค่าคงที่ความสูงของกระสุนที่ตัวละครอวตารใช้ มีค่าเท่ากับ 10

- กรณีศัตรูไม่อยู่ในหน้าจอแล้ว ให้ปัญญาประดิษฐ์เปลี่ยนสถานะทันที



รูปที่ 71 แสดงวิธีตรวจสอบว่าภาคีศัตรูอยู่ในระยะยิงของตัวละครอวตารหรือไม่

เงื่อนไขเปลี่ยนสถานะจากสถานะเดินเป็นสถานะต่อสู้

ขณะปัญญาประดิษฐ์อยู่ในสถานะเดิน แบ่งพิจารณาเป็น 3 กรณี

- กรณีศัตรูถูกนำออกจากเกมไปแล้ว ให้เปลี่ยนสถานะทันทีเนื่องจากไม่มีความจำเป็นที่จะอยู่ในสถานะเดินอีกต่อไป รวมถึงเป็นการเตรียมรับมือกับภาคีกระสุนฝ่ายศัตรูที่อาจจะหลงเหลืออยู่
- กรณีที่มีภาคีกระสุนฝ่ายศัตรูเข้ามาใกล้ตัวละครอวตารในระยะอันตราย ให้เปลี่ยนสถานะทันทีเพื่อพยายามหลบการโจมตี โดยนิยามของระยะอันตรายเป็นดังนี้

กำหนดให้ตัวละครอวตารอยู่ที่ตำแหน่ง (x_p, y_p) และภาคีกระสุนอยู่ที่ตำแหน่ง (x_b, y_b) และมีขนาด (w, h) จะถือว่าภาคีกระสุนอยู่ในระยะอันตรายก็ต่อเมื่อนิพจน์แบบบูลต่อไปนี้เป็นจริง

$$(|x_p - x_b| \le \frac{w + w_p}{2} + TP) \wedge (|y_p - y_b| \le \frac{h + h_p}{2} + TP)$$

โดย w_p และ h_p เป็นค่าคงที่ความกว้างและความสูงของตัวละครอวตารใช้ ซึ่งมีค่า 32 และ 48 ตามลำดับ และ TP เป็นค่าคงที่ระยะอันตรายซึ่งกำหนดค่าไว้เท่ากับ 180

- กรณีศัตรูอยู่ในระยะหน้าจอ หากเงื่อนไขใดเงื่อนไขหนึ่งเป็นจริงให้เปลี่ยนสถานะ
  - น่าจะสร้างความเสียหายให้ได้ ซึ่งใช้เงื่อนไข ต้องไม่อยู่ในสถานะคงกระพันเกิน 180 เฟรม และไม่อยู่นอกระยะยิงเกิน 180 เฟรม
  - ภาคีศัตรูเข้ามาใกล้ตัวละครอวตารในระยะอันตราย

7.4.3 การคาดเดาการโจมตี

การโจมตีภายในเกมแอ็คชั่นจะเกิดจากการเคลื่อนที่ของภาคีฝ่ายศัตรู การคาดเดาการโจมตีในที่นี้จึงเป็นการคาดเดาเส้นทางการเคลื่อนที่ของภาคีแต่ละตัวในโลกของเกม เมื่อพิจารณาฟังก์ชั่นการกระทำในแบบจำลอง จะพบว่าการเคลื่อนที่ที่เป็นไปได้ในเกมนี้เป็นการเคลื่อนที่ในแนวตรงทั้งหมด สำหรับกรณีของฟังก์ชั่น Jump ซึ่งเป็นการเคลื่อนที่แบบกระโดด ก็สามารถพิจารณาได้ว่า เป็นการเคลื่อนที่แนวตรงตามแนวแกน x และแกน y

ปัญญาประดิษฐ์จึงใช้การวิเคราะห์ความถดถอยและข้อมูลตำแหน่งของภาคีในอดีตรวมกับเฟรมปัจจุบันจำนวน 50 เฟรมล่าสุดเพื่อคาดเดาตำแหน่งของภาคีในอนาคตที่เวลาต่าง ๆ โดยแยกวิเคราะห์ตามแนวแกน ดังนี้

- ตำแหน่งแกน x

  ใช้การวิเคราะห์ความถดถอยเชิงเส้น โดยมีจุดเวลาเป็นตัวแปรอิสระ และค่าตำแหน่ง x เป็นตัวแปรตาม ข้อมูลที่เก่าที่สุดจะใช้ค่าจุดเวลาเป็น 0

- ตำแหน่งแกน y

  เนื่องจากในแกน y มีแรงดึงดูดมากระทำ ทำให้การเลื่อนที่ในแกน y มีความเร่ง ความสัมพันธ์ระหว่างตำแหน่งและเวลาจึงไม่ได้อยู่ในรูปของเส้นตรง แต่เป็นโพลีโนเมียลดีกรี 2 ในกรณีจึงต้องใช้การวิเคราะห์ความถดถอยแบบโพลีโนเมียลลำดับที่ 2 ใช้จุดเวลาเป็นตัวแปรอิสระ และค่าตำแหน่ง y เป็นตัวแปรตาม โดยข้อมูลที่เก่าที่สุดจะใช้ค่าจุดเวลาเป็น 0

ในกรณีที่ข้อมูลตำแหน่งมีไม่เพียงพอสำหรับการวิเคราะห์ความถดถอย (ไม่ถึง 3 เฟรม) ทำให้ไม่สามารถใช้หาสมการถดถอยสำหรับตำแหน่งในแกน y ได้ (จำนวนข้อมูลน้อยกว่าจำนวนพารามิเตอร์ไม่ทราบค่า) ปัญญาประดิษฐ์จำเป็นต้องใช้ข้อมูลเท่าที่มีในการคาดเดาตำแหน่งในอนาคตโดยไม่ใช้สมการถดถอย วิธีการคาดเดาตามจำนวนข้อมูลที่มีจึงเป็นไปดังนี้

จำนวนข้อมูล	วิธีคาดเดา
0 หรือ 1	ไม่สามารถคาดเดาตำแหน่งได้ ให้ใช้ตำแหน่งปัจจุบันเป็นตำแหน่งสำหรับทุกเฟรมในอนาคต
2	ใช้วิธีคาดการณ์โดยสมมติให้ทิศทางและความเร็วของการเคลื่อนที่ของภาคีคงที่ ดังนี้ กำหนดให้ตำแหน่งในเฟรมแรกคือ (x', y') และตำแหน่งในเฟรมที่สองคือ (x, y) สามารถคำนวณตำแหน่ง (x_f, y_f) ที่เฟรมอนาคต t_{future} โดยใช้ $$(x_f, y_f) = (x, y) + t_{future}(x - x', y - y')$$ เมื่อ t_{future} คือเฟรมในอนาคต โดย $t_{future} \geq 1$ หากค่าเป็น 1 จะหมายถึงเฟรมถัดไปจากเฟรมปัจจุบัน
ตั้งแต่ 3 ขึ้นไป	ใช้การวิเคราะห์ความถดถอย ดังนี้ กำหนดให้สมการถดถอยสำหรับตำแหน่งแกน x และแกน y ที่วิเคราะห์โดยใช้ข้อมูลตำแหน่งจำนวน T เฟรม อยู่ในรูปของฟังก์ชั่น $f_x(t)$ และ $f_y(t)$ ตามลำดับ สามารถคำนวณตำแหน่ง (x_f, y_f) ที่เฟรมอนาคต t_{future} โดยใช้ $$(x_f, y_f) = (f_x(T + t_{future} - 1), f_y(T + t_{future} - 1))$$ เมื่อ t_{future} คือเฟรมในอนาคต โดย $t_{future} \geq 1$

การเดาโดยใช้ข้อมูลในอดีตจะแม่นยำก็ต่อเมื่อข้อมูลตำแหน่งทั้งหมดในอดีต เป็นตำแหน่งบนเส้นทางการเคลื่อนที่เดียวกัน หากภาคีมีการเปลี่ยนแปลงความเร็วหรือเส้นทางการเคลื่อนที่ไป ก็อาจทำให้ข้อมูลเกิดความคลาดเคลื่อนได้ เช่น ในเฟรมอดีตที่ 1 – 20 ภาคีเคลื่อนที่ไปทางขวา ต่อมาเฟรมที่ 21 จนถึงปัจจุบัน ภาคีเคลื่อนที่ขึ้นแทน หากใช้ข้อมูลตำแหน่งทั้งหมดในการหาสมการถดถอย ย่อมทำให้ความแม่นยำลดลงเนื่องจากไม่สามารถทำการปรับเส้นโค้งให้เข้ากับเส้นทางที่แตกต่างกันของภาคีได้ ดังนั้น ก่อนการวิเคราะห์ความถดถอยจึงต้องตรวจสอบความเปลี่ยนแปลงของการเคลื่อนที่ก่อนเพื่อหาเส้นทางการเคลื่อนที่ล่าสุด หากการเคลื่อนที่ในอดีตมีความเปลี่ยนแปลงมากเกินไป ให้ละข้อมูลเหล่านั้นทิ้ง ขั้นตอนการตรวจสอบและละข้อมูลเป็นดังนี้

กำหนดความเปลี่ยนแปลงของทิศทางสูงสุดที่ยอมรับได้ $DirThreshold$ องศา และความเปลี่ยนแปลงของตำแหน่งสูงสุดที่ยอมรับได้ $DspThreshold$ พิกเซล การตรวจสอบข้อมูลตำแหน่งจำนวน n เฟรม $History = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}, \quad n \geq 3$ ทำได้โดย

Initialize $Frame = 1$

For (Pos in $History$) do:

If($Frame > 1$) do:

$CurrentDsp = Pos - LastPos$

If($Frame > 2$) do:

If($\left\|\|CurrentDsp\right\| - \left\|LastDsp\right\|\| > DspThreshold$ or

$\left|Angle(CurrentDsp, LastDsp)\right| > DirThreshold$) do:

Remove all data up to $\#(Frame-1)$ frame

$LastDsp = CurrentDsp$

$Frame = Frame + 1$

$LastPos = Pos$

งานวิทยานิพนธ์นี้ใช้ค่า $DirThreshold$ และ $DspThreshold$ เป็น 20 องศาและ 15 พิกเซลตามลำดับ

7.4.4 ฟังก์ชั่นวัตถุประสงค์

ฟังก์ชั่นวัตถุประสงค์ใช้ประเมินอนาคตจากการกดปุ่มเพื่อเคลื่อนตัวละครอวตารแต่ละแบบ โดยจำลองเส้นทางเดินในอนาคตของผู้เล่นจากการกดปุ่มและนำไปเทียบกับเส้นทางการโจมตีของภาคีฝ่ายศัตรูที่คาดเดาเอาไว้เพื่อค้นหาวิธีกดปุ่มที่ดีที่สุดสำหรับสถานการณ์ปัจจุบัน หากฟังก์ชั่นนี้มีค่าน้อยจะนับว่าเป็นค่าที่ดี ฟังก์ชั่นวัตถุประสงค์มีความแตกต่างกันเล็กน้อยสำหรับปัญญาประดิษฐ์ทั้ง 2 ตัว โดยฟังก์ชั่นวัตถุประสงค์สำหรับปัญญาประดิษฐ์ A คำนวณได้ดังนี้

กำหนดเซ็ตของวิธีกดปุ่มเพื่อเคลื่อนที่ที่เป็นไปได้ $C = \{\phi, \{\leftarrow\}, \{\rightarrow\}, \{Z\}, \{\leftarrow, Z\}, \{\rightarrow, Z\}\}$ ซึ่งประกอบด้วยกรณีไม่กดปุ่มเคลื่อนที่ใด ๆ , กดปุ่มซ้ายเท่านั้น, กดปุ่มขวาเท่านั้น, กดปุ่มกระโดดเท่านั้น, กดปุ่มซ้ายพร้อมกระโดด และกดปุ่มขวาพร้อมกระโดด ตามลำดับ ฟังก์ชั่นวัตถุประสงค์เมื่อกดปุ่ม $c, \quad c \in C$ คำนวณด้วย

$$f(c) = EC_{Cost}(Future(c)) + WC_{Cost}(Future(c)) - S_{\text{Re}ward}(Future(c))$$

เมื่อ

- $Future(c)$ เป็นฟังก์ชั่นคาดการณ์อนาคตเมื่อเลือกวิธีกดปุ่ม c โดยไม่เปลี่ยนแปลงเป็นระยะเวลา 50 เฟรม โดยคืนค่า
  - ตำแหน่งของตัวละครอวตารที่เฟรมอนาคตต่าง ๆ จำนวน 50 ตำแหน่ง ซึ่งคำนวณโดยให้ตัวละครอวตารทดลองเดินภายใต้กลไกของเกมที่ทำงานอยู่จริง ๆ โดยสนใจเฉพาะกลไกที่เกี่ยวข้องกับการเคลื่อนที่ เช่น โครงสร้างพื้นที่ เป็นต้น

- ตำแหน่งของภาคีฝ่ายศัตรูทุกตัวที่เฟรมอนาคตต่าง ๆ ภาคีละ 50 ตำแหน่ง ซึ่งคาดการณ์โดยใช้วิธีตามหัวข้อ 7.4.3

- $EC_{Cost}(Future)$ เป็นฟังก์ชั่นคำนวณค่าใช้จ่ายจากการที่ตัวละครอวตารชนกับภาคีศัตรูในอนาคต ยิ่งชนกับภาคีศัตรูในอนาคตอันใกล้ ยิ่งเกิดค่าใช้จ่ายสูง แต่ถ้าชนในอนาคตที่ไกลออกไปมากจะไม่สนใจ โดยฟังก์ชั่นนี้เทียบตำแหน่งตัวละครอวตารและภาคีศัตรูทุกตัวที่ได้จาก $Future(c)$ และคำนวณค่าที่คืนเป็น 2 กรณี
  - กรณีชนกับศัตรูครั้งแรกตั้งแต่เฟรมที่ 30 ขึ้นไป คืนค่า 0
  - กรณีที่ชนก่อนหน้านี้ คืนค่า 50 – หมายเลขเฟรมที่ชน
- $WC_{Cost}(Future)$ เป็นฟังก์ชั่นคำนวณค่าใช้จ่ายจากการที่ตัวละครอวตารวิ่งชนกำแพงในอนาคตอันใกล้ มีไว้เพื่อให้ปัญญาประดิษฐ์เลี่ยงการเดินเข้าหากำแพง ซึ่งมีความสำคัญกว่าการหลบหลีกศัตรู เพราะการเดินเข้าหากำแพงจะทำให้การเคลื่อนที่ในอนาคตข้างหน้าถูกจำกัดมากขึ้น ฟังก์ชั่นนี้คำนวณค่าที่คืนเป็น 2 กรณี
  - กรณีชนกำแพงตั้งแต่เฟรมที่ 10 ขึ้นไป คืนค่า 0
  - กรณีที่ชนก่อนหน้านี้ คืนค่า 100 ซึ่งเป็นค่าใช้จ่ายที่มากกว่าการชนกับศัตรูทุกกรณี
- $S_{\mathrm{Re}ward}(Future)$ เป็นฟังก์ชั่นคำนวณรางวัลจากการที่ตัวละครอวตารสามารถจัดวางตำแหน่งและทิศทางการหันหน้าให้สามารถโจมตีใส่ภาคีศัตรูได้ โดยจะให้รางวัลทุกเฟรมอนาคตที่ศัตรูอยู่ในระยะยิง, ตัวละครหันหน้าเข้าหาภาคีศัตรู และสามารถยิงได้ (การยิงครั้งหนึ่ง ๆ จะต้องเสียเวลาอย่างน้อย 1 เฟรม เพื่อปล่อยปุ่มยิง ซึ่งเป็นจังหวะที่ไม่สามารถยิงได้) รางวัลนี้จะสนใจในขณะที่ตัวละครอวตารยังมีพลังชีวิตเพียงพอเท่านั้น เนื่องจากในขณะที่ตัวละครอวตารเหลือพลังชีวิตน้อย ปัญญาประดิษฐ์ควรเน้นหลีกเลี่ยงการโจมตีมากกว่าการโจมตี ฟังก์ชั่นนี้คำนวณค่าที่คืนเป็น 2 กรณี
  - กรณีพลังชีวิตตัวละครอวตารเหลือเกิน 5 คำนวณรางวัลดังนี้

กำหนดให้ m คือภาคีศัตรูที่เข้าสู่โลกของเกมพร้อมกับตัวละครอวตารเมื่อการต่อสู้เริ่มขึ้น และ p คือตัวละครอวตาร สามารถคำนวณรางวัลโดย

Initialize $ShootingDelay = 0, \mathrm{Re}ward = 0$

For ($Frame$ in $Future$) do:

If($ShootingDelay = 0$) //The avatar can shoot during this frame

If(p faces toward m and $Frame.m$ is in $Frame.p$ range) do:

$\mathrm{Re}ward = \mathrm{Re}ward + 1$

$ShootingDelay = 2$

Else

$ShootingDelay = ShootingDelay - 1$

Return $\mathrm{Re}ward$ as result

เมื่อการตรวจสอบระยะยิงใช้นิยามตามที่ระบุในหัวข้อ 7.4.2

○ กรณีพลังชีวิตตัวละครอวตารมีไม่ถึง 5 ให้คืนค่า 0 เพื่อเน้นให้ปัญญาประดิษฐ์เล่นแบบเน้นความปลอดภัยของตัวละครอวตารมากกว่าการพยายามเอาชนะศัตรู

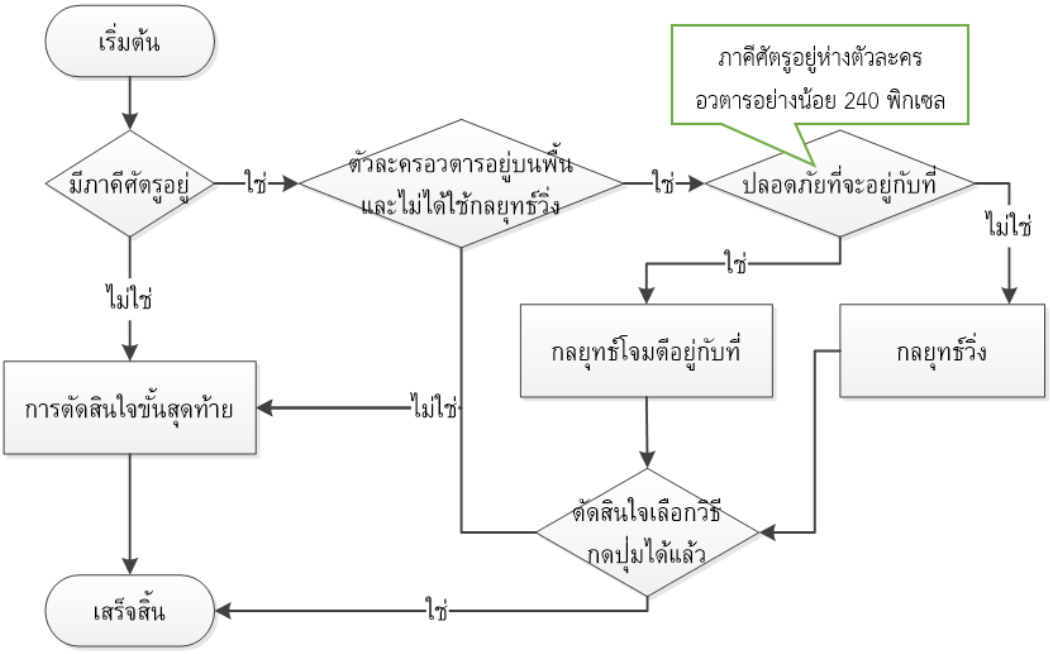
สำหรับปัญญาประดิษฐ์ B จะใช้งานฟังก์ชั่นวัตถุประสงค์เดียวกัน แต่มีการปรับปรุงฟังก์ชั่น $EC_{Cost}(Future)$ ให้เพิกเฉยต่อการชนที่เกิดขึ้นใน 10 เฟรมแรก เพื่อป้องกันค่าใช้จ่ายที่อาจเกิดขึ้นจากกรณีที่ตัวละครอวตารชนภาคีฝ่ายศัตรูไปแล้ว และกำลังต้องตัดสินใจกดปุ่มต่อ หากไม่เพิกเฉยจะทำให้ฟังก์ชั่นตรวจพบการชนกันตั้งแต่เฟรมอนาคตแรก และเกิดค่าใช้จ่ายที่สูงมากสำหรับทุกวิธีกดปุ่มที่เป็นไปได้ ส่งผลให้ไม่มีโอกาสได้ตรวจสอบอนาคตที่ไกลออกไป ซึ่งอาจจะมีช่องทางการหลบหลีกที่น่าสนใจอยู่

7.4.5 การตัดสินใจเพื่อควบคุมตัวละครอวตาร

การตัดสินใจของปัญญาประดิษฐ์แบ่งเป็นการตัดสินใจเพื่อควบคุมทิศทางการเคลื่อนที่ของตัวละครและการตัดสินใจกดปุ่มโจมตี

ปัญญาประดิษฐ์จะโจมตีใส่เป้าหมายที่สามารถสร้างความเสียหายให้ได้เท่านั้น (มีค่าคุณสมบัติฝ่ายศัตรูเป็นฝ่ายตั้งรับ และไม่มีคุณสมบัติคงกระพัน) โดยจะเลือกกดปุ่มโจมตีเมื่อภาคีเป้าหมายอยู่ในระยะยิงตามนิยามในหัวข้อ 7.4.2 และตัวละครอวตารหันหน้าเข้าภาคีดังกล่าวเท่านั้น

สำหรับการตัดสินใจควบคุมทิศทาง หากปัญญาประดิษฐ์อยู่ในสถานะเดิน ตัวละครอวตารจะถูกควบคุมให้เดินไปในทิศทางหนึ่งจนสุดฉากแล้วหันหลังกลับจากนั้นจึงวนซ้ำพฤติกรรม หากปัญญาประดิษฐ์อยู่ในสถานะต่อสู้ ปัญญาประดิษฐ์จะเลือกวิธีกดปุ่มที่ทำให้ฟังก์ชั่นวัตถุประสงค์ในหัวข้อ 7.4.4 มีค่าน้อยที่สุด หากมีวิธีกดปุ่มหลายแบบที่ทำให้ผลลัพธ์ฟังก์ชั่นวัตถุประสงค์มีค่าน้อยที่สุดได้ ปัญญาประดิษฐ์จะใช้ขั้นตอนตามผังงานในรูปที่ *72* ในการพิจารณาเลือกวิธีกดปุ่มที่ดีที่สุด ซึ่งสามารถสรุปโดยคร่าว ๆ คือ พยายามเลือกวิธีกดปุ่มที่ช่วยให้ยืนโจมตีศัตรูอยู่กับที่ได้ก่อน หากไม่สามารถทำได้จะพยายามสร้างระยะห่างจากศัตรู แต่ถ้าหากศัตรูอยู่ใกล้ตัวละครอวตารมากแล้ว จะใช้วิธีวิ่งเข้าหาศัตรูเพื่อกระโดดหรือลอดใต้ไปยังอีกฝั่งหนึ่งแทน หากยังไม่สามารถตัดสินใจเลือกวิธีกดปุ่มได้ ให้พยายามเลือกวิธีกดปุ่มที่เหมือนหรือไปในทิศทางเดียวกับการตัดสินใจครั้งก่อนหน้า หากยังไม่สามารถเลือกได้ ให้เลือกวิธีกดปุ่มแบบสุ่ม รายละเอียดของแต่ละส่วนในผังงานจะอธิบายในส่วนถัดไป โดยผังงานจะมีการใช้สัญลักษณ์พิเศษเพื่อลดทอนการใช้ข้อความดังแสดงไว้ในรูปที่ *73*



รูปที่ 72 ผังงานของการตัดสินใจของปัญญาประดิษฐ์

สัญลักษณ์แทนขั้นตอนการตัดสินใจเลือกวิธีกดปุ่มจากวิธีทั้งหมดที่ได้จากการใช้ฟังก์ชั่นวัตถุประสงค์เลือกเป็นดังภาพด้านล่าง

เลือก : เงื่อนไข A
ลำดับ : เซ็ตเงื่อนไข B

สัญลักษณ์ข้างต้นระบุว่า ปัญญาประดิษฐ์จะต้องเลือกวิธีกดปุ่มที่เป็นไปตามเงื่อนไข A หากมีวิธีกดปุ่มหลายวิธีที่เป็นไปตามเงื่อนไข A ให้เลือกวิธีกดปุ่มที่ตรงตามเงื่อนไขในเซ็ตอันดับ B โดยพิจารณาจากเงื่อนไขแรกสุดก่อน หากไม่ตรงตามเงื่อนไขแรกจึงพิจารณาเงื่อนไขถัดไป ในกรณีที่ไม่มีวิธีกดปุ่มที่ตรงตามเงื่อนไข A ให้ถือว่าปัญญาประดิษฐ์ไม่สามารถตัดสินใจวิธีกดปุ่มได้จากขั้นตอนนี้

ตัวอย่างการอ่านสัญลักษณ์

เลือก : ขวา
ลำดับ : {ไม่มีกระโดด, กระโดด}

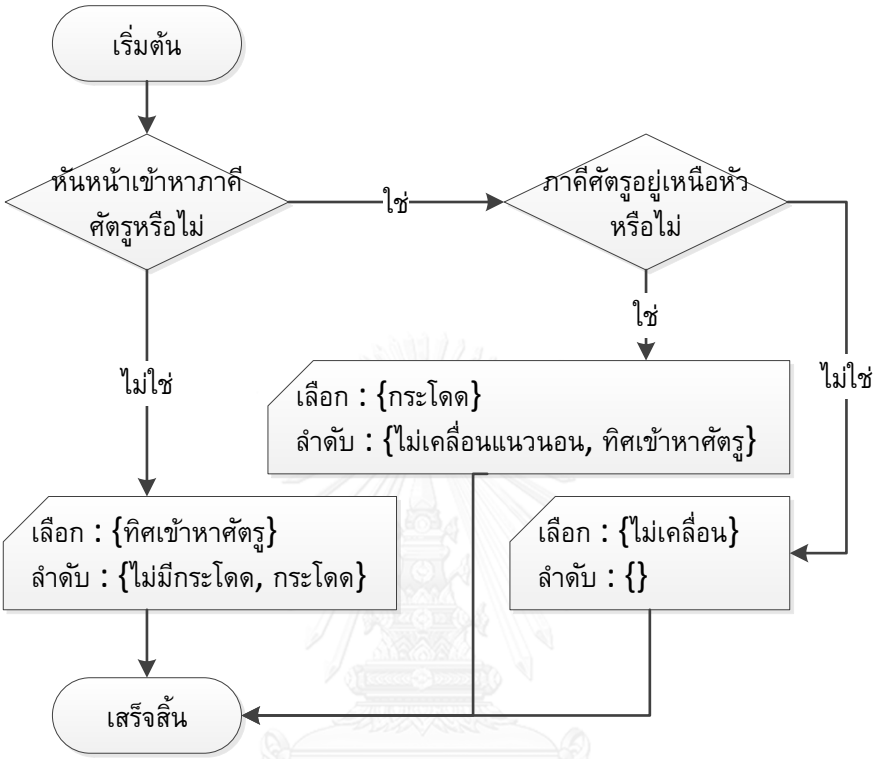
เมื่อกำหนดให้วิธีกดปุ่มที่ได้จากการตัดสินด้วยฟังก์ชั่นวัตถุประสงค์ ประกอบด้วย ไม่กด, กดขวา และกดขวาพร้อมกระโดด การตัดสินใจของปัญญาประดิษฐ์ตามสัญลักษณ์ตัวอย่าง เป็นดังนี้

- ขั้นตอนการเลือกจะทำให้เหลือตัวเลือกคือ กดขวา และกดขวาพร้อมกระโดด
- เมื่อพิจารณาโดยใช้เซ็ตเงื่อนไข จะทำให้ตัดสินใจได้ว่า ต้องเลือกวิธีกดปุ่ม “กดขวา”

รูปที่ 73 สัญลักษณ์พิเศษที่ใช้ในผังงานการตัดสินใจของปัญญาประดิษฐ์

กลยุทธ์โจมตีอยู่กับที่

ในกลยุทธ์นี้ ปัญญาประดิษฐ์จะพยายามเลือกวิธีกดปุ่มที่ทำให้ภาคีศัตรูอยู่ในระยะยิง หากยังไม่ได้หันหน้าเข้าหาเป้าหมาย จะเลือกวิธีกดปุ่มที่ทำให้หันเข้าใส่ หากเป้าหมายอยู่เหนือหัวขึ้นไป จะเลือกวิธีกดปุ่มที่ทำให้กระโดดขึ้นไปหาได้ โดยทุกวิธีจะพยายามเลือกวิธีกดปุ่มที่เคลื่อนที่น้อยที่สุด แสดงวิธีตัดสินใจได้ด้วยผังงานดังนี้



รูปที่ 74 ผังงานของกลยุทธ์โจมตีอยู่กับที่

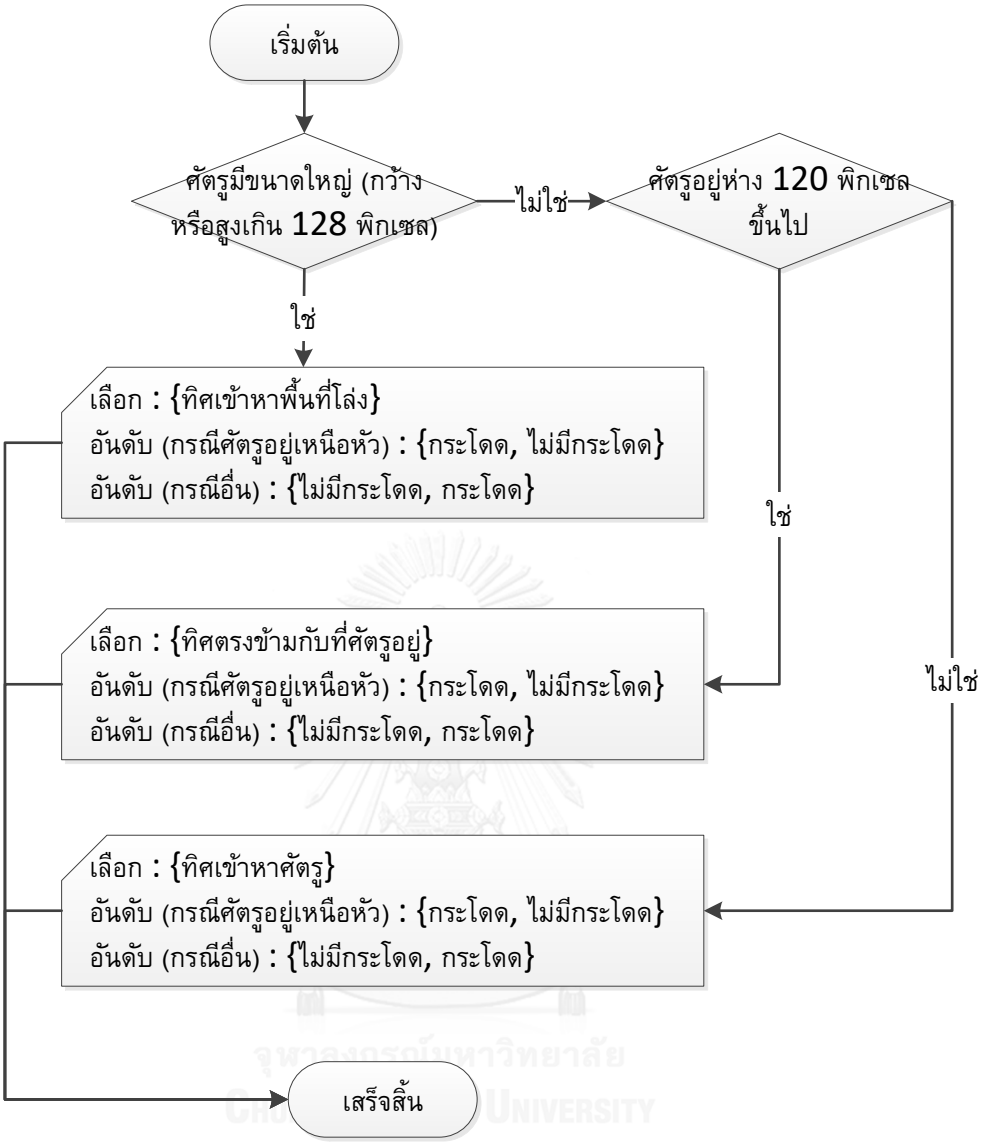
กลยุทธ์วิ่ง

ในกลยุทธ์นี้ ปัญญาประดิษฐ์จะพยายามเลือกวิธีกดปุ่มที่ทำให้ตัวละครอวตารวิ่งออกไปอยู่ในบริเวณที่ปลอดภัย หากมีการเลือกวิธีกดปุ่มภายในกลยุทธ์นี้ จะถือว่าตัวละครใช้กลยุทธ์วิ่งเป็นระยะเวลา 30 เฟรม ซึ่งจะส่งผลต่อการตัดสินใจในช่วงแรก (กล่องตัดสินใจที่ 2 ในรูปที่ *72*)

กลยุทธ์นี้เริ่มจากการตรวจสอบว่าภาคีศัตรูมีขนาดใหญ่หรือไม่ (กว้างหรือสูงตั้งแต่ 128 พิกเซลขึ้นไป) หากเป็นขนาดใหญ่ ให้พยายามเลือกวิธีกดปุ่มที่ทำให้ตัวละครอวตารวิ่งออกไปในทิศที่มีพื้นที่โล่ง ซึ่งนิยามดังนี้

> กำหนดให้พื้นที่ข้างซ้ายคือระยะทางแนวนอนจากภาคีศัตรูไปยังกำแพงฝั่งซ้ายของฉากในเกมทดสอบ และพื้นที่ข้างขวาคือระยะทางแนวนอนจากภาคีศัตรูไปยังกำแพงฝั่งขวา หากพื้นข้างค่าซ้ายมีค่ามากกว่า ทิศที่มีพื้นที่โล่งคือซ้าย ในทางกลับกันหากพื้นที่ข้างขวามีค่าเท่ากับหรือมากกว่า ทิศที่มีพื้นที่โล่งคือขวา

กรณีที่ไม่ใช่ศัตรูขนาดใหญ่ ให้พยายามวิ่งหนีหากอยู่ไกลจากศัตรูเกินกว่า 120 พิกเซล หากอยู่ใกล้กว่านั้นให้ใช้วิธีวิ่งเข้าหาศัตรูเพื่อกระโดดข้ามไปอีกฝั่ง วิธีตัดสินใจทั้งหมดแสดงได้ด้วยผังงานดังนี้

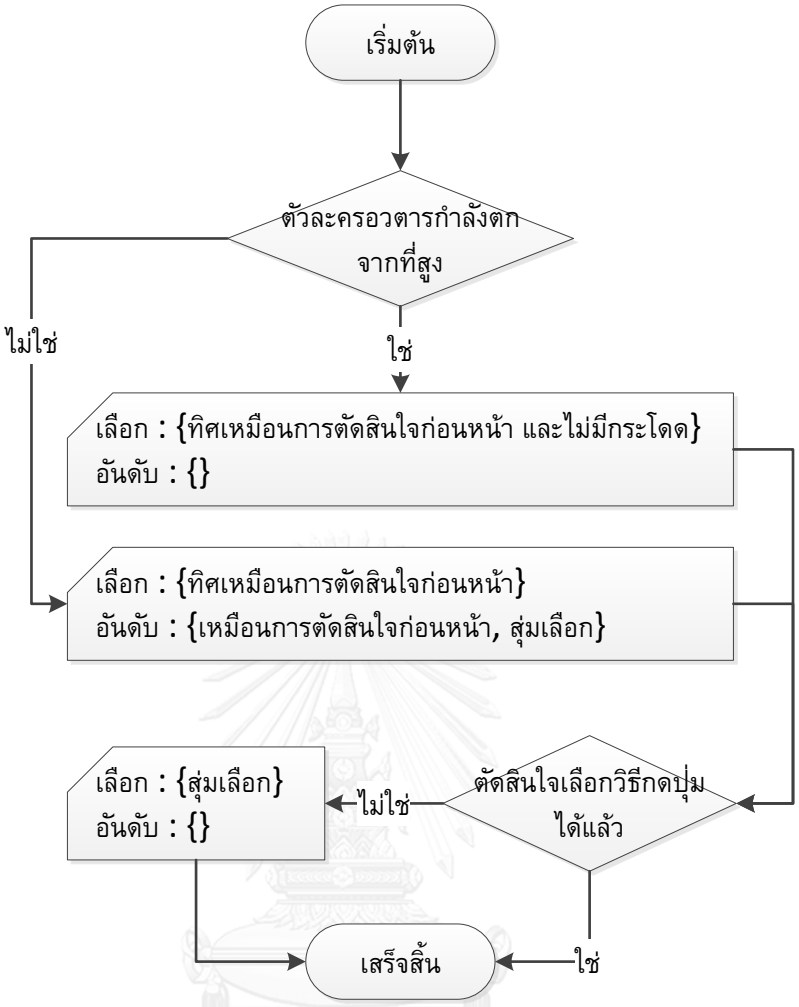


รูปที่ 75 ผังงานของกลยุทธ์วิ่ง

การตัดสินใจขั้นสุดท้าย

ในกรณีที่ปัญญาประดิษฐ์ยังไม่สามารถตัดสินใจได้ ปัญญาประดิษฐ์จะพยายามเลือกวิธีกดปุ่มที่มีทิศการเคลื่อนที่เหมือนกับการตัดสินใจก่อนหน้า โดยแบ่งเป็นกรณีที่ตัวละครอวตารกำลังตกลงจากที่สูง (หลังจากกระโดดจนสุดแล้ว) และกรณีอื่น ๆ ในกรณีแรก ให้พยายามเลือกวิธีกดปุ่มที่มีทิศแนวนอนเหมือนการตัดสินใจก่อนหน้าและไม่มีปุ่มกระโดด สำหรับกรณีหลังให้เลือกวิธีกดปุ่มที่มีทิศแนวนอนเหมือนการตัดสินใจก่อนหน้า หากมีหลายตัวเลือกให้เลือกวิธีกดปุ่มที่เหมือนการตัดสินใจก่อนหน้า หากทำไม่ได้ ให้เลือกอย่างสุ่ม

ในท้ายที่สุดหากยังตัดสินใจไม่ได้ ให้เลือกอย่างสุ่ม ผังงานสำหรับการตัดสินใจเป็นไปดังผังงานด้านล่าง



รูปที่ 76 ผังงานการตัดสินใจขั้นสุดท้าย

7.4.6 การวัดผลปัญญาประดิษฐ์สำหรับวัดผล

ปัญญาประดิษฐ์ที่จัดทำขึ้นมีจุดประสงค์เพื่อใช้วัดผลขั้นตอนวิธีแทนผู้เล่นมนุษย์ และเนื่องจากการวัดผลขั้นตอนวิธีจะใช้การวัดประสิทธิภาพของการต่อสู้ดังที่ได้กล่าวไว้ในหัวข้อ 7.1 ดังนั้นปัญญาประดิษฐ์ที่จะใช้งานจึงจำเป็นจะต้องเล่นได้ประสิทธิภาพใกล้เคียงกับผู้เล่นมนุษย์ ไม่เก่งเกินไปและไม่แย่เกินไป

การวัดผลปัญญาประดิษฐ์จึงใช้การเปรียบเทียบประสิทธิภาพการต่อสู้เทียบกับค่าเฉลี่ยของผู้เล่นมนุษย์ โดยจะคิดแยกตามศัตรูแต่ละระดับที่แบ่งไว้ในหัวข้อ 7.3.1 เนื่องจากศัตรูแต่ละระดับมีความซับซ้อนแตกต่างกัน ซึ่งอาจส่งผลให้เกิดความแตกต่างในค่าประสิทธิภาพได้ หากค่าประสิทธิภาพแตกต่างกันไม่เกิน 0.1 จึงจะถือว่าประสิทธิภาพใกล้เคียงกันเพียงพอ ซึ่งเกณฑ์ความแตกต่างไม่เกิน 0.1 นั้น เป็นค่าเดียวกับที่จะใช้ในการวัดการยอมรับได้ของศัตรูที่สร้างขึ้นโดยขั้นตอนวิธีเช่นกัน

วิธีการคำนวณค่าประสิทธิภาพสำหรับปัญญาประดิษฐ์เป็นไปตามหัวข้อ 7.2.2 โดยค่าคงที่ระยะที่มีความสำคัญและเวลาต่อสู้จำกัดที่ใช้ในการคำนวณจะเป็นค่าได้จากการคำนวณประสิทธิภาพเฉลี่ยของผู้เล่นมนุษย์ เพื่อให้ข้อมูลที่คำนวณได้อยู่บนบรรทัดฐานของผู้เล่นมนุษย์

การเก็บข้อมูลการต่อสู้สำหรับปัญญาประดิษฐ์

งานวิทยานิพนธ์นี้ได้ให้ปัญญาประดิษฐ์แต่ละตัวทำการเก็บข้อมูลตัวละ 3 รอบเพื่อป้องกันข้อมูลผิดปกติที่อาจส่งผลให้ค่าประสิทธิภาพที่คำนวณได้ผิดเพี้ยนไป โดยในการเก็บข้อมูลแต่ละรอบ ปัญญาประดิษฐ์จะต้องต่อสู้กับศัตรูทุกตัวที่ปรากฏในชุดข้อมูลศัตรู ในกรณีที่ศัตรูดังกล่าวไม่สามารถปราบได้ และไม่ใช่ศัตรูที่ออกแบบให้ปราบไม่ได้ ปัญญาประดิษฐ์สามารถตัดสินใจข้ามการต่อสู้ได้เช่นกัน โดยใช้เกณฑ์ว่า หากไม่สามารถสร้างความเสียหายให้ภาคีศัตรูได้นานเกินกว่า 3600 เฟรม (ประมาณ 1 นาที) ให้ข้ามการต่อสู้ สาเหตุที่ผู้วิจัยเพิ่มระยะเวลาตัดสินใจให้ปัญญาประดิษฐ์นานกว่าที่ผู้เล่นจริงใช้ในการตัดสินใจ (ประมาณ 1800 เฟรม) เนื่องจากอาจมีความเป็นไปได้ที่ปัญญาประดิษฐ์ตัดสินใจเลือกวิธีกดปุ่มไม่ดีเพียงพอที่จะโจมตีใส่ภาคีศัตรูทั้งที่ศัตรูดังกล่าวยังมีโอกาสรับความเสียหายได้

ขั้นตอนวัดผล

การคำนวณค่าประสิทธิภาพจากข้อมูลของผู้เล่นและข้อมูลการต่อสู้ของปัญญาประดิษฐ์ ประกอบด้วย การคัดกรองข้อมูลเพื่อเลือกเฉพาะข้อมูลจากการต่อสู้กับศัตรูที่ทั้งผู้เล่นและปัญญาประดิษฐ์ได้ต่อสู้ด้วย เพื่อป้องกันความโน้มเอียงของข้อมูลที่อาจเกิดจากศัตรูที่ผู้เล่นไม่ได้ต่อสู้ด้วยแต่ปัญญาประดิษฐ์มีโอกาสต่อสู้ด้วย จากนั้นคำนวณค่าประสิทธิภาพของผู้เล่น แล้วจึงคำนวณค่าประสิทธิภาพของปัญญาประดิษฐ์โดยใช้ระยะเวลาปราบศัตรูเฉลี่ยและระยะที่มีความสำคัญของผู้เล่นเป็นค่าคงที่ในการคำนวณเพื่อให้ผลลัพธ์อยู่บนบรรทัดฐานเดียวกัน รายละเอียดขั้นตอนการวัดผลสำหรับปัญญาประดิษฐ์หนึ่งตัวใด ๆ สำหรับข้อมูลศัตรูระดับใด ๆ เป็นดังนี้

1) คำนวณหาระยะที่มีความสำคัญ Ran จากข้อมูลของผู้เล่นตามหัวข้อ 7.2.3
2) คัดกรองรายชื่อศัตรูสำหรับการคำนวณระยะเวลาปราบศัตรูและพลังชีวิตที่เหลืออยู่ เพื่อจัดทำเซ็ตรายชื่อ A โดยเลือกเฉพาะข้อมูลการต่อสู้กับศัตรูที่ตรงตามเงื่อนไขดังนี้
    - ปัญญาประดิษฐ์ได้ต่อสู้ด้วยครบทั้ง 3 รอบ
    - ผู้เล่นอย่างน้อย 1 คนได้ต่อสู้ด้วย
    - ในทุกการต่อสู้ของทั้งปัญญาประดิษฐ์และผู้เล่น จะต้องปราบศัตรูได้สำเร็จ หรือสร้างความเสียหายให้ศัตรูได้ในกรณีที่ปราบไม่สำเร็จ เงื่อนไขนี้มีไว้เพื่อคัดกรองศัตรูที่ถูกออกแบบให้ปราบไม่ได้และศัตรูที่มีการข้ามการต่อสู้ออก เพื่อให้สามารถคำนวณระยะเวลาปราบศัตรูได้
3) คัดกรองรายชื่อศัตรูสำหรับการคำนวณอัตราหลบหลีกพลาด เพื่อจัดทำเซ็ตรายชื่อ B โดยเลือกเฉพาะข้อมูลการต่อสู้กับศัตรูที่ตรงตามเงื่อนไขดังนี้
    - ปัญญาประดิษฐ์ได้ต่อสู้ด้วยครบทั้ง 3 รอบ
    - ผู้เล่นอย่างน้อย 1 คนได้ต่อสู้ด้วย
    - ในทุกการต่อสู้ของทั้งปัญญาประดิษฐ์และผู้เล่น ภาคีฝ่ายศัตรูอย่างน้อย 1 ตัวจะต้องเคยเข้ามาในระยะที่มีความสำคัญ หรือก็คือเซ็ตของภาคีที่มีความสำคัญตามที่ระบุไว้ในหัวข้อ 7.2.2 ภายใต้วิธีคำนวณอัตราหลบหลีกพลาด ต้องไม่เป็นเซ็ตว่าง เงื่อนไขนี้มีไว้เพื่อคัดกรองศัตรูนอกระยะซึ่งตัวละครอวตารไม่จำเป็นต้องพยายามหลบการโจมตีก็ได้

4) คำนวณระยะเวลาปราบศัตรูเฉลี่ยของผู้เล่น $DurAvg_{Player} = \frac{\sum_{e \in E} \sum_{p \in P(e)} Dur(p)}{\sum_{e \in E} |P(e)|}$ เมื่อ

- E คือ เซ็ตของรายชื่อศัตรูที่สนใจจากขั้นตอน 2
- $P(e)$ คือเซ็ตของข้อมูลการต่อสู้กับศัตรูชื่อ e ที่เก็บจากผู้เล่นทุกคน
- $Dur(p)$ คือระยะเวลาต่อสู้ที่คำนวณจากข้อมูลการต่อสู้ p โดยใช้วิธีที่ระบุในหัวข้อ 7.2.2 ซึ่งระยะเวลาดังกล่าวนี้เป็นเวลาที่ใช้หรือน่าจะใช้ (ในกรณีที่ผู้เล่นปราบศัตรูไม่สำเร็จ) จนกระทั่งปราบศัตรูได้สำเร็จ

5) คำนวณอัตราหลบหลีกพลาดเฉลี่ยของผู้เล่น $MRAvg_{Player} = \frac{\sum_{e \in E} \sum_{p \in P(e)} MR(p)}{\sum_{e \in E} |P(e)|}$ เมื่อ

- $MR(p)$ คืนอัตราหลบหลีกพลาดที่คำนวณจากข้อมูลการต่อสู้ p โดยใช้วิธีที่ระบุในหัวข้อ 7.2.2 กำหนดค่าคงที่ระยะที่มีความสำคัญเท่ากับ Ran

6) คำนวณอัตราหลบหลีกพลาดเฉลี่ยของปัญญาประดิษฐ์ $MRAvg_{AI} = \frac{\sum_{e \in E} \sum_{p \in A(e)} MR(p)}{\sum_{e \in E} |A(e)|}$ เมื่อ

- $A(e)$ คือเซ็ตของข้อมูลการต่อสู้กับศัตรูชื่อ e ที่เก็บจากปัญญาประดิษฐ์ 3 รอบ
- $MR(p)$ ใช้ค่าคงที่ระยะที่มีความสำคัญเท่ากับ Ran เพื่อให้การประเมินอยู่บนบรรทัดฐานเดียวกับผู้เล่นมนุษย์

7) คำนวณพลังชีวิตที่เหลืออยู่เฉลี่ยของศัตรูที่ปัญญาประดิษฐ์ต่อสู้ด้วย

$HPAvg_{AI} = \frac{\sum_{e \in E} \sum_{p \in A(e)} HP(p)}{\sum_{e \in E} |A(e)|}$ เมื่อ

- $HP(p)$ คือพลังชีวิตที่เหลืออยู่ของศัตรูที่คำนวณจากข้อมูลการต่อสู้ p โดยใช้วิธีที่ระบุในหัวข้อ 7.2.2 กำหนดค่าคงที่เวลาต่อสู้จำกัดเท่ากับ $DurAvg_{Player}$ เพื่อให้ค่าที่ได้จากการประเมินสื่อถึงพลังชีวิตของศัตรูที่เหลืออยู่เมื่อปัญญาประดิษฐ์ต่อสู้ด้วยระยะเวลาที่เท่ากับที่ผู้เล่นใช้หรือน่าจะใช้ในการปราบศัตรูกลุ่มดังกล่าวโดยเฉลี่ย

8) เปรียบเทียบค่าประสิทธิภาพ โดยปัญญาประดิษฐ์จะถือว่ามีประสิทธิภาพใกล้เคียงมนุษย์ สามารถนำมาใช้วัดผลได้ก็ต่อเมื่อ 2 นิพจน์ข้างล่างเป็นจริง

- $|MRAvg_{Player} - MRAvg_{AI}| \leq 0.1$ เป็นเงื่อนไขสำหรับเปรียบเทียบอัตราหลบหลีกเฉลี่ยของผู้เล่นและปัญญาประดิษฐ์ที่จะต้องแตกต่างกันไม่เกิน 0.1
- $HPAvg_{AI} \leq 0.1$ เป็นเงื่อนไขสำหรับเปรียบเทียบพลังชีวิตที่เหลืออยู่เฉลี่ย ซึ่งค่าเฉลี่ยจากข้อมูลต่อสู้ของปัญญาประดิษฐ์จะถูกนำมาเทียบกับค่า 0 เนื่องจากการคำนวณตั้งอยู่บนฐานที่ว่าผู้เล่นสามารถปราบศัตรูได้ (พลังชีวิตศัตรูเหลือ 0 เสมอ)

7.4.7 ปัญญาประดิษฐ์ที่เลือกใช้

เมื่อทำการเก็บข้อมูลและคำนวณค่าประสิทธิภาพโดยใช้ขั้นตอนตาม 7.4.6 สำหรับปัญญาประดิษฐ์ทั้ง 2 ตัว และสำหรับศัตรูทั้ง 4 ระดับแล้ว ผลลัพธ์ที่ได้เป็นดังนี้

ตารางที่ 5 ค่าประสิทธิภาพของผู้เล่นมนุษย์และปัญญาประดิษฐ์ที่คำนวณจากการต่อสู้กับศัตรูในชุดข้อมูล

ข้อมูลประสิทธิภาพ	ระดับศัตรู			
	Enemy	Elite	Miniboss	Boss
ผู้เล่นมนุษย์				
$MRAvg_{Player}$	0.09356051	0.110458866	0.1599621	0.20182736
ปัญญาประดิษฐ์ A				
$MRAvg_A$	0.0787753	0.078663	0.11227553	0.24946557
$\lvert MRAvg_{Player} - MRAvg_A \rvert$	**0.01478521**	**0.031795866**	**0.04768657**	**0.04763821**
$HPAvg_A$	**0.0546875**	**0**	**0**	**0.004520796**
ปัญญาประดิษฐ์ B				
$MRAvg_B$	0.07244554	0.066702284	0.12457907	0.18737853
$\lvert MRAvg_{Player} - MRAvg_B \rvert$	**0.02111497**	**0.043756582**	**0.03538303**	**0.01444883**
$HPAvg_B$	**0.057894737**	**0**	**0**	**0.0013227513**

จากตารางข้างต้น พบว่าทั้งปัญญาประดิษฐ์ A และปัญญาประดิษฐ์ B ต่างก็มีประสิทธิภาพใกล้เคียงกับผู้เล่นเพียงพอที่จะนำมาใช้วัดผล ในที่นี้จึงตัดสินใจเลือกปัญญาประดิษฐ์ที่มีความใกล้เคียงผู้เล่นมากกว่า โดยพิจารณาเลือกปัญญาประดิษฐ์ที่มีค่าเฉลี่ยของผลต่างประสิทธิภาพน้อยกว่า ค่าเฉลี่ยดังกล่าวคำนวณดังนี้

$$AvgPerformanceDiff = \frac{\sum \left| MRAvg_{Player} - MRAvg_{AI} \right| + \sum HPAvg_{AI}}{8}$$

ผลลัพธ์ที่ได้เป็นดังนี้

	ปัญญาประดิษฐ์ A	ปัญญาประดิษฐ์ B
$AvgPerformanceDiff$	0.025139269	0.02174011254

งานวิทยานิพนธ์นี้จึงเลือกใช้ปัญญาประดิษฐ์ B ในการวัดผลขั้นตอนวิธี

7.5 การวัดผลขั้นตอนวิธีด้วยปัญญาประดิษฐ์

จากนิยามการยอมรับได้ในหัวข้อ 7.1 ศัตรูที่สร้างขึ้นจากขั้นตอนวิธีจะถือว่ายอมรับได้หากประสิทธิภาพการต่อสู้กับศัตรูดังกล่าว ใกล้เคียงกับประสิทธิภาพการต่อสู้กับศัตรูที่มีคุณลักษณะใกล้เคียงกัน งานวิทยานิพนธ์นี้จึง

วัดผลโดยนำเอาข้อมูลการต่อสู้กับศัตรูที่สร้างขึ้น ไปเทียบกับค่าบรรทัดฐานซึ่งเป็นค่าประสิทธิภาพที่คำนวณจากข้อมูลการต่อสู้กับศัตรูในชุดข้อมูลศัตรูที่มีคุณลักษณะใกล้เคียงกับศัตรูที่สร้างขึ้น โดยข้อมูลการต่อสู้ที่นำมาใช้จะเลือกเฉพาะข้อมูลที่เก็บโดยใช้ปัญญาประดิษฐ์ในการควบคุมตัวละครอวตารเท่านั้น

7.5.1 การหาค่าบรรทัดฐาน

หากอ้างอิงตามนิยามของการยอมรับได้ในหัวข้อ 7.1 ค่าบรรทัดฐานสำหรับการเปรียบเทียบค่าประสิทธิภาพจะต้องประกอบด้วย ค่าบรรทัดฐานของพลังชีวิตที่เหลืออยู่ และค่าบรรทัดฐานของอัตราหลบหลีกพลาด ในที่นี้เลือกใช้ค่า 0 เป็นค่าบรรทัดฐานของพลังชีวิตที่เหลืออยู่สำหรับศัตรูทุกตัว โดยไม่สนใจรูปแบบพฤติกรรม เนื่องจากหนึ่งในเงื่อนไขของการเป็นศัตรูที่ยอมรับได้ตามนิยาม คือ ศัตรูจะต้องถูกปราบได้

สำหรับเงื่อนไขที่ว่ารูปแบบพฤติกรรมจะต้องทำให้ผู้เล่นสามารถหลบหลีกศัตรูได้นั้น จะต้องใช้ค่าบรรทัดฐานของอัตราหลบหลีกพลาดที่แตกต่างกันไปสำหรับวัดผลศัตรูที่สร้างขึ้นแต่ละตัว ด้วยแนวคิดที่ว่าศัตรูที่มีคุณลักษณะคล้ายกัน ควรจะส่งผลให้ประสิทธิภาพการต่อสู้ที่คำนวณได้ใกล้เคียงกัน การเลือกค่าบรรทัดฐานในกรณีนี้จึงเป็นการเลือกศัตรูที่มีคุณลักษณะใกล้เคียงกับศัตรูที่ต้องการวัดผลจากชุดข้อมูลศัตรู เพื่อคำนวณหาอัตราหลบหลีกพลาดและใช้ค่านั้นเป็นค่าบรรทัดฐานของอัตราหลบหลีกสำหรับวัดผลศัตรูที่สร้างขึ้น หัวใจสำคัญสำหรับการหาค่าบรรทัดฐานของอัตราหลบหลีกพลาดจึงอยู่ที่การเลือกศัตรูที่เหมาะสมจากชุดข้อมูลศัตรู

การเลือกศัตรูเพียงหนึ่งตัวจากชุดข้อมูลศัตรูเพื่อนำมาเทียบกับศัตรูที่สร้างขึ้นอาจไม่เหมาะสม เนื่องจากศัตรูที่สร้างขึ้นอาจมีความคล้ายคลึงกับศัตรูได้หลายตัว ในที่นี้จึงใช้วิธีแบ่งศัตรูในชุดข้อมูลศัตรูออกเป็นกลุ่มที่มีคุณลักษณะบางอย่างร่วมกัน โดยคุณสมบัติที่สนใจจะเป็นคุณสมบัติที่น่าจะส่งผลต่ออัตราหลบหลีกพลาด การเลือกศัตรูเพื่อหาค่าบรรทัดฐานจึงเปลี่ยนเป็นการเลือกกลุ่มของศัตรูที่คล้ายกับศัตรูที่ต้องการวัดผลแทน แล้วจึงคำนวณค่าบรรทัดฐานของอัตราหลบหลีกพลาดของกลุ่มโดยใช้ค่าเฉลี่ยของอัตราหลบหลีกพลาดของศัตรูภายในกลุ่มดังกล่าว

ในหัวข้อ 7.3.1 ได้ทำการแบ่งกลุ่มศัตรูตามระดับศัตรูไว้แล้ว ภายใต้แนวคิดที่ว่าความยากและความซับซ้อนที่แตกตางกันจะส่งผลให้ค่าประสิทธิภาพของกลุ่มแตกต่างกัน หนึ่งในเกณฑ์ที่ใช้แบ่งกลุ่มตามระดับนั้น คือ ค่าพลังชีวิตของศัตรู ซึ่งเป็นค่าคุณสมบัติที่ระยะเวลาต่อสู้แปรผันตาม และหากระยะเวลาต่อสู้ยาวนานขึ้น โอกาสที่จะโดนโจมตีก็น่าจะมีมากขึ้นด้วย หรืออีกนัยหนึ่งคือ หากศัตรูมีพลังชีวิตมาก อัตราหลบหลีกพลาดก็น่าจะสูงขึ้น ในที่นี้จึงเลือกใช้การแบ่งกลุ่มตามระดับศัตรูในการหาค่าบรรทัดฐาน

ปัญญาประดิษฐ์ B จะถูกใช้ต่อสู้กับศัตรูทั้งหมดในชุดข้อมูลศัตรูตัวละ 3 รอบเพื่อเก็บข้อมูลและคำนวณหาค่าเฉลี่ยของอัตราหลบหลีกพลาดสำหรับศัตรูทั้ง 4 กลุ่ม (Enemy, Elite, Miniboss และ Boss) เพื่อใช้เป็นค่าบรรทัดฐานของอัตราหลบหลีกพลาด

ตารางที่ *6* แสดงค่าบรรทัดฐานที่ใช้ในงานวิทยานิพนธ์นี้ แยกตามระดับของศัตรู

ตารางที่ 6 ค่าบรรทัดฐานสำหรับศัตรูแต่ละระดับ

ค่าบรรทัดฐาน	ระดับของศัตรู			
	Enemy	Elite	Miniboss	Boss
อัตราหลบหลีกพลาด	0.06520099891	0.06670228154	0.1245790665	0.1667809678
พลังชีวิตที่เหลืออยู่	0	0	0	0

7.5.2 การวัดผล

ศัตรูที่ถูกสร้างขึ้นด้วยขั้นตอนวิธีในบทที่ 6 จะถูกกำหนดค่าคุณสมบัติที่ตายตัวไว้โดยใช้ค่าคุณสมบัติแม่แบบ เนื่องจากขั้นตอนวิธีที่นำเสนอสร้างรูปแบบพฤติกรรมโดยไม่ได้นำเอาค่าคุณสมบัติเริ่มต้น (ค่าที่ถูกกำหนดในบล็อกเริ่มต้นของภาคี) เข้ามาพิจารณาด้วย ดังนั้น ในงานวิทยานิพนธ์นี้จะถือว่ารูปแบบพฤติกรรมของศัตรูที่สร้างขึ้นมาสามารถนำไปใช้กับศัตรูระดับใดก็ได้ ในขั้นตอนวัดผลนี้จึงจะทำการปรับปรุงศัตรูที่สร้างขึ้นให้กลายเป็นศัตรูที่มีระดับต่างกัน แต่มีพฤติกรรมเหมือนกัน โดยการคัดลอกสคริปต์ศัตรูที่สร้างขึ้นมา 4 ชุด เพื่อใช้เป็นศัตรูสำหรับระดับ Enemy, Elite, Miniboss และ Boss และปรับค่าพลังชีวิตเริ่มต้นของศัตรูตามระดับ โดยใช้ค่าเฉลี่ยของพลังชีวิตของศัตรูแต่ละระดับในชุดข้อมูลศัตรู (นับเฉพาะศัตรูที่ออกแบบให้ปราบได้) ซึ่งมีค่าดังนี้

Enemy	Elite	Miniboss	Boss
2	12	18	26

ศัตรูทั้ง 4 ระดับที่เกิดขึ้นนี้ ในการวัดผลจะถือว่าเป็นศัตรูคนละตัวกัน สำหรับขั้นตอนการวัดผลศัตรู e ที่สร้างขึ้นใด ๆ โดยใช้ปัญญาประดิษฐ์ B มีขั้นตอนดังนี้

1) กำหนดให้ MR_{base} และ HP_{base} คือ ค่าบรรทัดฐานของอัตราหลบหลีกพลาดและค่าบรรทัดฐานของพลังชีวิตที่เหลืออยู่ตามลำดับ โดยได้จากตารางที่ 6 ตามแต่ระดับของศัตรู e
2) ให้ปัญญาประดิษฐ์ B ต่อสู้กับศัตรู e เป็นจำนวน 5 รอบ
3) คำนวณหาระยะที่มีความสำคัญ Ran จากข้อมูลของผู้เล่นตามหัวข้อ 7.2.3
4) คำนวณอัตราหลบหลีกพลาดเฉลี่ยของปัญญาประดิษฐ์ $MRAvg_{AI} = \frac{\sum_{e\in E}\sum_{p\in A(e)} MR(p)}{\sum_{e\in E}|A(e)|}$ เมื่อ
   - $A(e)$ คือเซ็ตของข้อมูลการต่อสู้กับศัตรู e 5 รอบ
   - $MR(p)$ คืออัตราหลบหลีกพลาดที่คำนวณจากข้อมูลการต่อสู้ p โดยใช้วิธีที่ระบุในหัวข้อ 7.2.2 กำหนดค่าคงที่ระยะที่มีความสำคัญเท่ากับ Ran
5) คำนวณพลังชีวิตที่เหลืออยู่เฉลี่ยของศัตรูด้วย $HPAvg_{AI} = \frac{\sum_{e\in E}\sum_{p\in A(e)} HP(p)}{\sum_{e\in E}|A(e)|}$ เมื่อ
   - $HP(p)$ คือพลังชีวิตที่เหลืออยู่ของศัตรูที่คำนวณจากข้อมูลการต่อสู้ p โดยใช้วิธีที่ระบุในหัวข้อ 7.2.2 กำหนดค่าคงที่เวลาต่อสู้จำกัดเท่ากับ ∞ สาเหตุที่ไม่จำกัดระยะเวลาต่อสู้

ในการคำนวณเพราะศัตรูที่ยอมรับได้ตามนิยามที่กำหนดไม่ได้ระบุว่าจำเป็นต้องปราบศัตรูได้ภายใต้ระยะเวลาที่จำกัด

6) เปรียบเทียบค่าประสิทธิภาพ โดยศัตรู e ที่สร้างขึ้นด้วยขั้นตอนวิธีจะถือว่ายอมรับได้ก็ต่อเมื่อ 2 นิพจน์ข้างล่างเป็นจริง
   - $\left|MRAvg_{AI} - MR_{base}\right| \leq 0.1$ เป็นเงื่อนไขสำหรับเปรียบเทียบอัตราหลบหลีกเฉลี่ย ซึ่งต้องแตกต่างจากค่าบรรทัดฐานไม่เกิน 0.1
   - $\left|HPAvg_{AI} - HP_{base}\right| \leq 0.1$ เป็นเงื่อนไขสำหรับเปรียบเทียบพลังชีวิตที่เหลืออยู่เฉลี่ย ซึ่งต้องแตกต่างจากค่าบรรทัดฐานไม่เกิน 0.1

7.6 การทดลอง ผลการทดลองและการวิเคราะห์ผล

ผู้วิจัยทดลองสร้างศัตรูด้วยขั้นตอนวิธีที่นำเสนอจำนวน 300 ตัว ซึ่งได้ผลลัพธ์เป็นศัตรูที่ถูกยกเลิกการสร้างเนื่องจากไม่มีพฤติกรรมจำนวน 33 ตัว และศัตรูที่สามารถสร้างจนจบกระบวนการได้จำนวน 267 ตัว ศัตรูที่สร้างจนสำเร็จจะถูกคัดลอกจัดทำเป็นศัตรู 4 ระดับตามวิธีวัดผลที่ระบุไว้ใน 7.5.2 ดังนั้นจึงมีศัตรูทั้งสิ้นจำนวน 1068 ตัว ระดับละ 267 ตัว เมื่อทำการวัดผลโดยใช้ปัญญาประดิษฐ์แล้ว ได้ผลลัพธ์ดังตารางที่ *7*

ตารางที่ 7 ผลการวัดผลศัตรูที่สร้างแยกตามระดับ

การวัดผล	จำนวนศัตรูแยกตามระดับ				รวม
	Enemy	Elite	Miniboss	Boss	
ผ่าน (ยอมรับได้)	161	168	81	56	466
ไม่ผ่าน	106	99	186	211	602
% ที่ผ่าน	**60.3**	**62.92**	**30.34**	**20.97**	**43.63**

จากผลการทดลองจะเห็นได้ว่าผลลัพธ์ถูกแบ่งออกเป็น 2 กลุ่มชัดเจน โดยกลุ่ม Enemy และ Elite ผ่านการวัดผลด้วยอัตราที่ใกล้เคียงกันที่ประมาณ 60% และกลุ่ม Miniboss และ Boss ที่ผ่านการวัดผลในช่วง 20-30% เมื่อผู้วิจัยทำการตรวจสอบสาเหตุที่ทำให้ศัตรูแต่ละตัวไม่ผ่านการวัดผล พบว่า ศัตรูเกือบทั้งหมดไม่ว่าจะเป็นศัตรูระดับใดก็ตาม ไม่ผ่านการวัดผลเพราะค่าอัตราหลบหลีกพลาด ($MRAvg_{AI}$) ที่มากกว่าหรือน้อยกว่าค่าบรรทัดฐานเกินกว่า 10% และอีกจำนวนน้อยที่ไม่ผ่านเพราะไม่สามารถทำให้พลังชีวิตของศัตรูเหลือ 0 ได้ ประเด็นน่าสนใจที่ได้จากการทดลองแบ่งเป็น 3 ข้อ ดังนี้

1) ปัจจัยใดที่ทำให้ค่าอัตราหลบหลีกพลาดเกินค่าบรรทัดฐาน
2) ปัจจัยใดที่ทำให้ค่าอัตราหลบหลีกพลาดต่ำกว่าค่าบรรทัดฐาน
3) ปัจจัยใดที่ทำให้ศัตรูไม่สามารถปราบได้

ปัจจัยที่ทำให้อัตราหลบหลีกพลาดเกินค่าบรรทัดฐาน

เมื่อผู้วิจัยดูการต่อสู้ระหว่างปัญญาประดิษฐ์และศัตรูที่ทำให้อัตราหลบหลีกพลาดเกินค่าบรรทัดฐาน พบว่าปัญหาที่ทำให้อัตราหลบพลาดสูงนั้น เกิดจากสาเหตุหลัก ๆ ทั้งหมด 3 ประการ ได้แก่

1) ศัตรูโจมตีด้วยกระสุนเยอะเกินไปจนทำให้ไม่สามารถหลบได้ หรือเคลื่อนที่ต่อเนื่องมากเกินไปจนเคลื่อนหลบไม่ทัน
2) ศัตรูเคลื่อนที่เร็วมากโดยไม่มีการหยุดการกระทำชั่วคราวและมีทิศเข้าหาตัวละครอวตารตลอดทำให้ไม่สามารถกระโดดหลบได้พ้นแน่นอน
3) ปัญญาประดิษฐ์ไม่สามารถควบคุมให้ตัวละครอวตารหลบได้ทัน ซึ่งเกิดจากการที่ปัญญาประดิษฐ์ที่จัดทำขึ้นไม่มีความสามารถในการจดจำรูปแบบการโจมตีของศัตรูและอาศัยปฏิกิริยาโต้ตอบในการหลบหลีกเท่านั้น ทำให้ไม่สามารถเตรียมรับมือการโจมตีล่วงหน้าได้

ปัจจัยที่ทำให้อัตราหลบหลีกพลาดต่ำกว่าค่าบรรทัดฐาน

ผลลัพธ์ในกรณีนี้เกิดจากการที่ศัตรูไม่ทำการโจมตีเข้าหาตัวละครอวตาร จากการตรวจสอบการต่อสู้ของปัญญาประดิษฐ์ พบหลายสาเหตุที่ทำให้เกิดกรณีที่ศัตรูไม่โจมตีใส่ดังนี้

1) รูปแบบพฤติกรรมของศัตรูไม่ทำให้ศัตรูเคลื่อนไหวหรือจู่โจมตัวละครอวตาร ทำให้ศัตรูเป็นเสมือนกับกรณีที่ไม่มีพฤติกรรม เช่น ศัตรูที่มีเพียงฟังก์ชั่น Wait ในบล็อกลำดับ
2) ความถี่ในการโจมตีด้วยกระสุนต่ำเกินไป
3) ศัตรูโจมตีไปในทิศอื่นที่ตัวละครอวตารไม่อยู่หรือไม่สามารถอยู่ในจุดนั้น เช่น ศัตรูไม่เคลื่อนไหว แต่สามารถยิงกระสุนได้ ทว่ากระสุนที่ยิงนั้นไปในทิศลงล่างซึ่งไม่มีโอกาสโดนตัวละครอวตาร
4) ศัตรูสามารถเคลื่อนไหวได้ และโจมตีด้วยกระสุนไปในทิศที่หันหน้าอยู่ได้ แต่กลับวิ่งเข้าหากำแพงโดยไม่มีการกลับทิศออกมาจากกำแพง ส่งผลให้ยิงไม่โดนผู้เล่นและทำให้ศัตรูหยุดเคลื่อนไหวไปโดยปริยาย
5) ปัญญาประดิษฐ์มีความแม่นยำในการควบคุมสูงมากเกินกว่าที่ผู้เล่นมนุษย์จะทำได้ ทำให้บางสถานการณ์ตัวละครอวตารสามารถหลบได้ดีเกินจริงเมื่อเทียบกับผู้เล่นมนุษย์ที่มีฝีมือการเล่นระดับปานกลาง

ปัจจัยที่ทำให้ศัตรูปราบไม่ได้

ในที่นี้พบเพียงปัญหาเดียวที่ทำให้ปราบศัตรูไม่ได้ คือ ศัตรูค้างอยู่กลางอากาศในจุดที่ผู้เล่นกระโดดไม่ถึงทำให้ไม่สามารถสร้างความเสียหายให้ได้ ซึ่งเป็นผลมาจากฟังก์ชั่น Set ที่ตั้งค่าผลของแรงดึงดูดให้เป็น 0 ทำให้ศัตรูไม่ลงมาถึงพื้น โดยที่ไม่มีฟังก์ชั่นสำหรับการเคลื่อนที่ในแนวตั้งเพื่อเลื่อนให้ศัตรูอยู่ต่ำลง

ปัจจัยที่ส่งผลให้รูปแบบพฤติกรรมที่สร้างไม่ผ่านการวัดผลสามารถแบ่งได้เป็น 2 กลุ่ม ได้แก่ ปัจจัยที่เกิดจากรูปแบบพฤติกรรม และปัจจัยที่เกิดจากปัญญาประดิษฐ์ สำหรับปัจจัยกลุ่มหลัง จะเห็นได้ว่าเกิดจากตัวปัญญาประดิษฐ์ที่ยังไม่สามารถเล่นได้ใกล้เคียงกับมนุษย์อย่างแท้จริง ถึงแม้ว่าค่าประสิทธิภาพจะออกมาใกล้เคียงกับผู้เล่นมนุษย์ก็ตาม

สำหรับปัจจัยกลุ่มแรกซึ่งเป็นรูปแบบพฤติกรรมไม่พึงประสงค์ ต้นเหตุของการเกิดพฤติกรรมเช่นนี้ย่อมแฝงอยู่ในขั้นตอนวิธีที่ใช้สร้างรูปแบบพฤติกรรม ซึ่งอาจจะเป็นส่วนของการทำเหมืองข้อมูล หรือส่วนสร้างรูปแบบพฤติกรรมอัตโนมัติที่นำเอาข้อมูลมาใช้ก็ได้ ในที่นี้จะทำการวิเคราะห์แต่ละปัจจัยและอธิบายถึงสิ่งที่น่าจะเป็นต้นตอของรูปแบบพฤติกรรมไม่พึงประสงค์เหล่านี้

กรณีที่การโจมตีด้วยกระสุนมีมากหรือน้อยเกินไปนั้น เกิดจากฟังก์ชั่น Spawn ถูกเรียกใช้ด้วยความถี่ที่ไม่เหมาะสม ซึ่งเกิดจากหลายสาเหตุ ได้แก่

- จำนวนฟังก์ชั่น Spawn ในภาคีมีมากเกินไปหรือน้อยเกินไป ปัญหานี้น่าจะเกิดจากการที่ขั้นตอนวิธีในการสร้างรูปแบบพฤติกรรมไม่ได้กำหนดจำนวนของฟังก์ชั่นแต่ละชนิดที่ควรใช้งานได้ในภาคีหนึ่ง ๆ
- ฟังก์ชั่นการกระทำระยะยาวที่คั่นระหว่างฟังก์ชั่น Spawn ใช้เวลามากหรือน้อยเกินไปในการทำงานจนเสร็จสิ้น ปัญหานี้น่าจะเกิดจากการที่ข้อมูลความสัมพันธ์ที่ได้จากการทำเหมืองข้อมูลให้ความสนใจแค่ลำดับของฟังก์ชั่นการกระทำ โดยไม่ได้คำนึงถึงอากิวเมนท์ของฟังก์ชั่นนั้น ๆ ไปพร้อมกับการพิจารณาความสัมพันธ์เชิงลำดับ
- รูปแบบพฤติกรรมของกระสุนที่ถูกสร้างด้วยฟังก์ชั่น Spawn สามารถสร้างกระสุนซ้ำได้ ซึ่งกรณีนี้ไม่ใช่รูปแบบพฤติกรรมไม่พึงประสงค์แต่อย่างใด ทว่าเมื่อรวมเข้ากับปัญหาด้านบน จะทำให้มีโอกาสที่กระสุนที่สร้างสามารถสร้างกระสุนซ้ำได้ถี่เกินไป เป็นการเพิ่มจำนวนกระสุนในฉากแบบทวีคูณ

กรณีที่ศัตรูเคลื่อนที่ต่อเนื่องมากเกินไปจะคล้ายคลึงกับกรณีของกระสุนที่มากเกินไป แต่เกิดจากฟังก์ชั่นที่ทำให้เคลื่อนไหวทำงานต่อเนื่องเกินไปโดยไม่มีหยุดพัก ซึ่งอาจเกิดพร้อมกับความเร็วในการเคลื่อนที่ที่สูง หรือระยะเวลาของการกระทำที่สั้นส่งผลให้ทิศทางการเคลื่อนที่เปลี่ยนแปลงบ่อยและไม่สามารถเตรียมรับมือได้ทัน ปัญหาในกรณีนี้เกิดจากการทำเหมืองข้อมูลที่ไม่ได้พิจารณาความสัมพันธ์ระหว่างฟังก์ชั่นแบบการไม่เป็นลำดับของกันและกัน ซึ่งระบุความสัมพันธ์ในลักษณะที่ว่าฟังก์ชั่นหนึ่ง ๆ จะต้องไม่อยู่ต่อหลังฟังก์ชั่นอีกตัวหนึ่งทันทีทันใด ซึ่งอาจต้องนำเอาอากิวเมนท์มาพิจารณาร่วมด้วย

กรณีที่ศัตรูเคลื่อนที่เข้าหาตัวละครอวตารอย่างรวดเร็วจนไม่สามารถหลบได้พ้น เป็นกรณีของพฤติกรรมที่ค่อนข้างเฉพาะเจาะจงของกรณีก่อนหน้า ซึ่งเกิดจากฟังก์ชั่น RunStraight ที่มีอากิวเมนท์ของทิศทางเป็น TurnToPlayer ซึ่งทำให้ศัตรูวิ่งเข้าหาผู้เล่น มีค่าความเร็วที่มากเกินกว่าที่ผู้เล่นจะหลบได้ และไม่มีเงื่อนไขจบ (พารามิเตอร์ actionEnd เป็น false) โดยจะต้องเกิดขึ้นในบล็อกสถานะที่ไม่มีบล็อกลำดับอื่นที่สามารถปรับเปลี่ยนทิศทางการเคลื่อนที่ของศัตรู หรือเปลี่ยนสถานะของศัตรูได้ สาเหตุหลักของปัญหานี้น่าจะเกิดจากการที่ไม่มีการเรียนรู้ความสัมพันธ์ระหว่างฟังก์ชั่นการกระทำไปพร้อมกับอากิวเมนท์ ทำให้ไม่รู้ว่า หากฟังก์ชั่น RunStraight มีอากิวเมนท์ทิศทางเป็น TurnToPlayer ควรจะต้องมีพฤติกรรมคู่ขนานอะไรจึงจะเหมาะสม

กรณีที่ศัตรูไม่เคลื่อนไหวหรือจู่โจมนั้นเกิดจากวิธีเลือกความสัมพันธ์มาเติมในโครงภาคี ซึ่งขั้นตอนวิธีใช้การเลือกแบบสุ่มทำให้มีโอกาสที่จะสุ่มได้เฉพาะความสัมพันธ์ที่ไม่มีฟังก์ชั่นการกระทำที่ทำให้เคลื่อนไหว (เช่น RunStraight หรือ Spawn) หรือสุ่มเลือกได้ฟังก์ชั่นที่ไม่เข้ากัน เช่น RunStraight ในทิศทางลงเกิดขึ้นในศัตรูที่มีค่าคุณสมบัติผลของแรงดึงดูดมากกว่า 0 (ได้รับผลของแรงดึงดูดในทิศทางลง) หรือมีฟังก์ชั่น Set ที่กำหนดให้ผลของแรงดึงดูดเป็นค่ามากกว่า 0 เป็นต้น ปัญหากรณีที่สุ่มไม่เจอความสัมพันธ์ที่มีฟังก์ชั่นการกระทำที่ทำให้เคลื่อนไหวเกิดจากการที่ขั้นตอนวิธีไม่ได้นำเอาสถิติการใช้งานฟังก์ชั่นแต่ละชนิดมาใช้กำหนดจำนวนฟังก์ชั่นที่ควรปรากฏในรูปแบบพฤติกรรมที่สร้างขึ้น สำหรับปัญหากรณีที่สุ่มเลือกได้ฟังก์ชั่นที่ไม่เข้ากันนั้นเกิดจากการทำเหมืองข้อมูลที่ไม่ได้ให้

ความสนใจความสัมพันธ์ระหว่างการใช้ฟังก์ชั่นการกระทำแบบต่าง ๆ และค่าคุณสมบัติ รวมถึงความสัมพันธ์แบบการไม่เกิดร่วมกันของฟังก์ชั่น ซึ่งระบุความสัมพันธ์ในลักษณะที่ว่าหากมีฟังก์ชั่นใดอยู่แล้วต้องไม่มีฟังก์ชั่นใดปรากฏในภาคี

กรณีที่ศัตรูโจมตีผิดทิศทาง เคลื่อนที่จนไปติดอยู่กับกำแพงและไม่เคลื่อนห่างออกมา หรือไม่สามารถปราบได้เนื่องจากอยู่ในตำแหน่งที่ตัวละครอวตารโจมตีไม่ถึงเป็นปัญหาที่เกิดจาก 3 องค์ประกอบ ได้แก่ ตำแหน่งปัจจุบันของตัวละครอวตารหรือศัตรู โครงสร้างพื้นที่ และฟังก์ชั่นการกระทำ มีความไม่เข้ากัน เช่น ศัตรูเป็นภาคีที่อยู่บนพื้นเป็นหลักแต่กลับมีกระสุนที่วิ่งลงด้านล่าง หรือศัตรูหยุดค้างอยู่กลางอากาศในจุดที่สูงเกินไป เป็นต้น ซึ่งขั้นตอนวิธีไม่สามารถทำความเข้าใจความสัมพันธ์นี้ได้เนื่องจากข้อมูลตำแหน่งเป็นข้อมูลที่รู้ได้ก็ต่อเมื่อตัวศัตรูแสดงพฤติกรรมอยู่ในโลกของเกมแล้ว หรือก็คือเป็นศัตรูที่สร้างจนเสร็จตามกระบวนการแล้วนั่นเอง

7.7 การปรับปรุงผลลัพธ์ของขั้นตอนวิธี และผลการทดลองหลังปรับปรุง

จากตารางที่ *7* จะเห็นได้ว่าศัตรูที่สร้างขึ้นมาทั้งหมดด้วยขั้นตอนวิธีที่นำเสนอ มีศัตรูที่ยอมรับได้เพียง 41.38% ของจำนวนศัตรูทั้งหมด ผู้วิจัยจึงเล็งเห็นว่าควรมีวิธีการปรับปรุงศัตรูที่สร้างจนเสร็จเหล่านี้เพื่อให้ศัตรูที่สร้างขึ้นสามารถนำไปใช้งานได้จำนวนหลายตัวขึ้น ในหัวข้อนี้จะกล่าวถึงกระบวนการปรับปรุงสคริปต์ศัตรูเพื่อแก้ไขปัญหา 3 ประการที่พบจากการทดลองใน 7.6 ได้แก่ ศัตรูโจมตีถี่เกินไป ศัตรูโจมตีด้วยความถี่ที่ต่ำเกิน และศัตรูอยู่นิ่งเสมือนไม่มีรูปแบบพฤติกรรม (เฉพาะกรณีที่ไม่มีฟังก์ชั่นการกระทำที่ทำให้เคลื่อนไหวได้) โดยจะทำการปรับปรุงศัตรูกลุ่มที่อยู่นิ่งให้มีพฤติกรรมที่ทำให้เคลื่อนไหวก่อน จากนั้นจึงปรับปรุงความถี่ในการโจมตี วิธีการปรับปรุงเป็นดังนี้

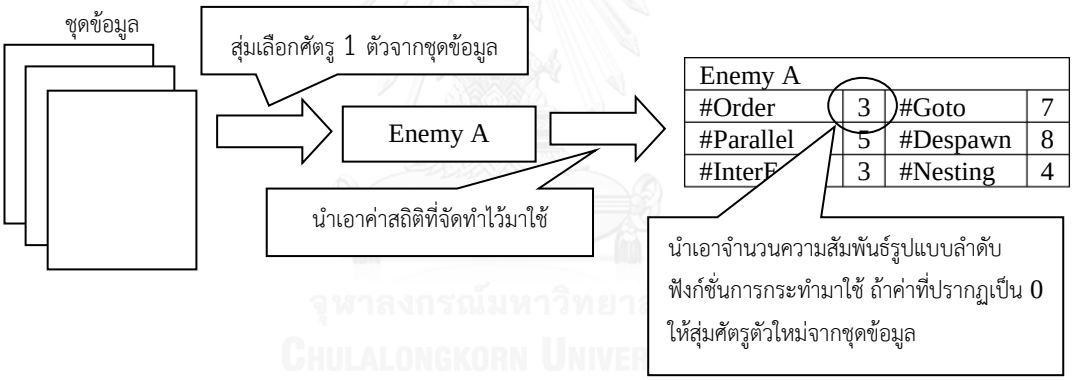
การปรับปรุงศัตรูที่อยู่นิ่ง

ขั้นตอนนี้จะใช้ปรับปรุงศัตรูที่อยู่นิ่งเท่านั้น ในที่นี้สามารถตรวจสอบภาคีที่อยู่นิ่งได้โดยตรวจว่ามีฟังก์ชั่นการกระทำที่ทำให้เคลื่อนไหวได้ปรากฏอยู่ในลำดับพฤติกรรมหรือไม่ ฟังก์ชั่นการกระทำที่งานวิทยานิพนธ์นี้ถือว่าเป็นฟังก์ชั่นการกระทำที่ทำให้เคลื่อนไหวแบ่งเป็น 2 กลุ่ม ได้แก่

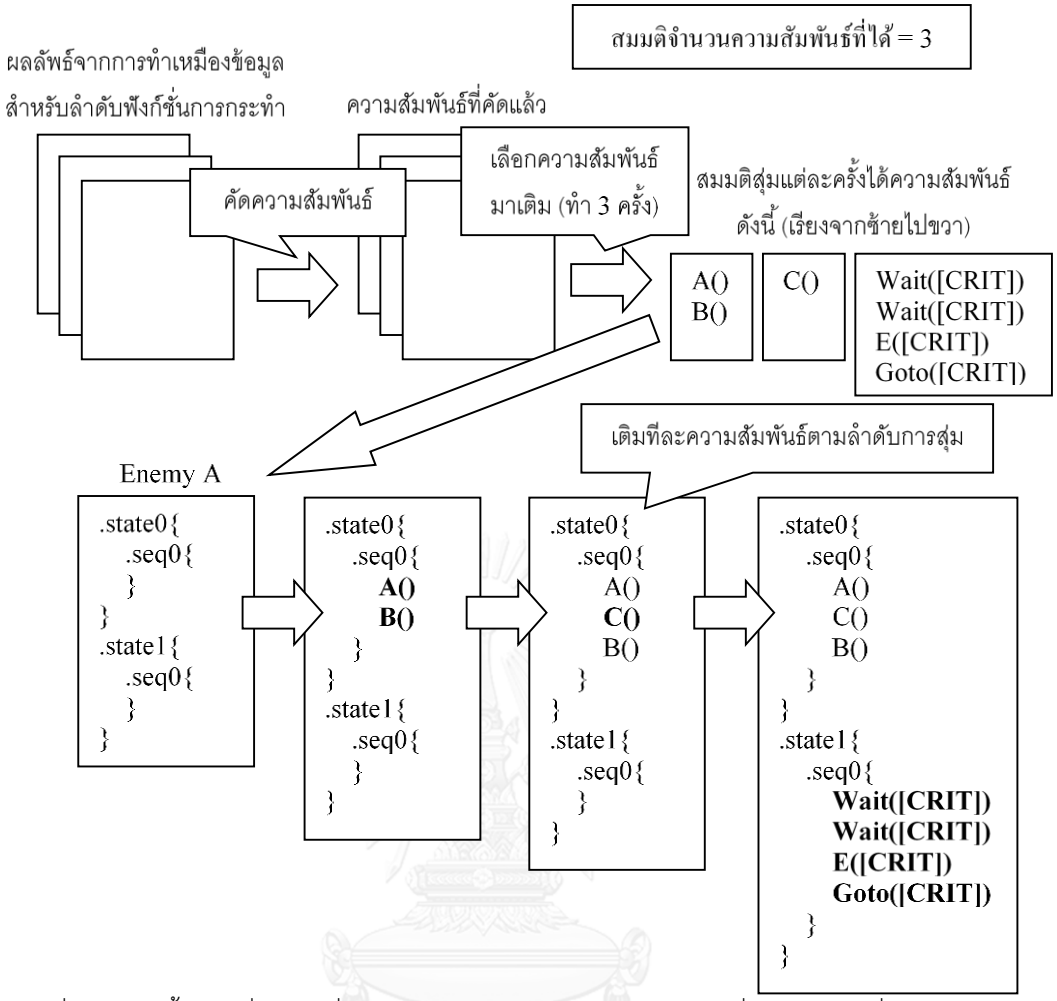
- การกระทำที่ทำให้ภาคีเปลี่ยนตำแหน่งได้ ประกอบด้วยฟังก์ชั่น Jump, RunHarmonic, RunStraight และ RunTo รวมไปถึงฟังก์ชั่น Set position ที่ทำการเปลี่ยนตำแหน่งของภาคี (มีอากิวเมนท์ของภาคีเป้าหมายเป็น DynamicFilter(“this”))
- การกระทำที่ส่งผลกับตัวละครอวตาร ประกอบด้วย
    - AddExtraVelocityToPlayer ซึ่งทำให้ตัวละครอวตารเลื่อนที่
    - Spawn ซึ่งสามารถสร้างกระสุน ส่งผลต่อการเคลื่อนที่ของตัวละครอวตาร
    - ฟังก์ชั่น Set ทุกแบบที่มีอากิวเมนท์ของภาคีเป้าหมายเป็น DynamicFilter(“player”)

การปรับปรุงสคริปต์สามารถทำได้โดยเติมความสัมพันธ์รูปแบบลำดับฟังก์ชั่นการกระทำลงในภาคีหลักโดยที่ความสัมพันธ์เหล่านั้นจะต้องมีฟังก์ชั่นการกระทำที่ทำให้เคลื่อนไหวรวมอยู่ ขั้นตอนการปรับปรุงเป็นดังด้านล่าง โดยรูปที่ *77* ถึงรูปที่ *79* แสดงตัวอย่างการปรับปรุงสคริปต์

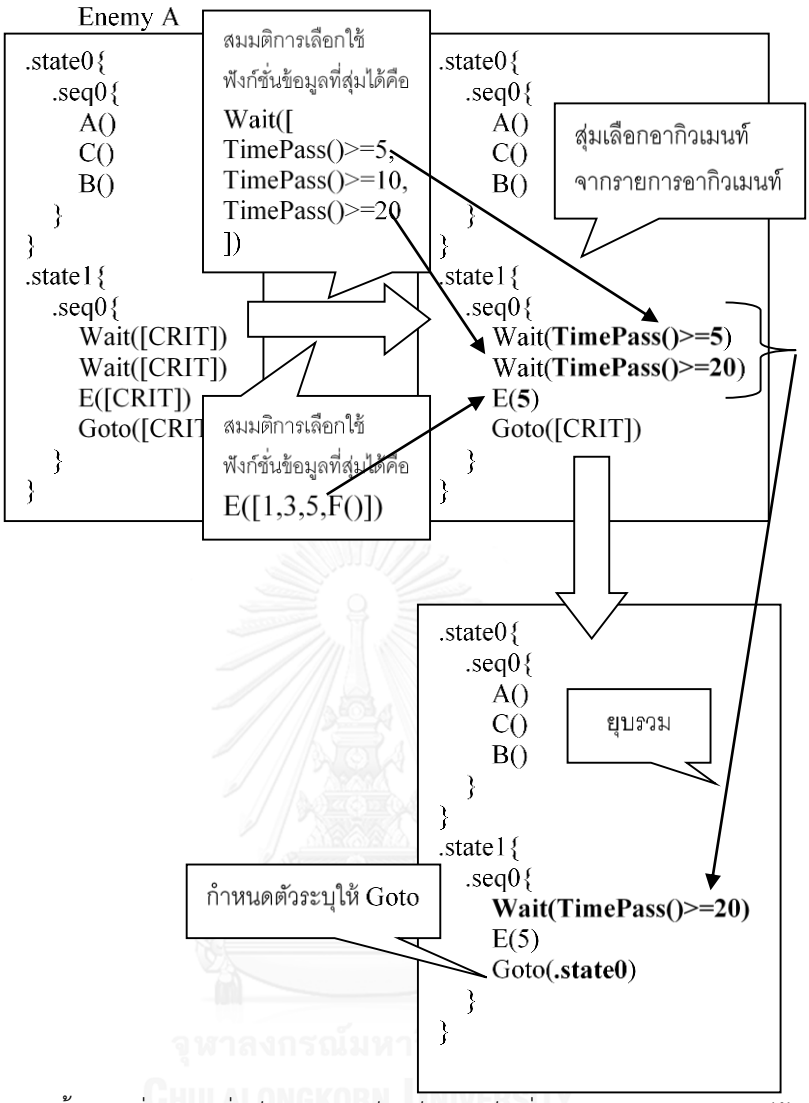
1) กำหนดตัวเลขของจำนวนความสัมพันธ์ที่จะเติมโดยสุ่มเลือกศัตรู 1 ตัวจากชุดข้อมูลใน 5.1 และนำเอาจำนวนความสัมพันธ์รูปแบบลำดับฟังก์ชั่นที่ปรากฏในค่าสถิติของศัตรูมาใช้ โดยค่าสถิติจะใช้ข้อมูลเดียวกับขั้นตอนที่ 1 ของขั้นตอนการสร้างศัตรูโดยอัตโนมัติ 6.2 ในกรณีที่ตัวเลขที่ได้เป็น 0 ให้สุ่มใหม่จนกว่าจะได้ค่าที่ไม่ใช่ 0
2) คัดกรองผลจากการทำเหมืองข้อมูลสำหรับลำดับฟังก์ชั่นการกระทำให้เหลือเฉพาะผลลัพธ์ที่มีฟังก์ชั่นการกระทำที่ทำให้เคลื่อนไหว ในกรณีที่สคริปต์ที่ต้องการปรับปรุงมีภาคีเพียงภาคีเดียว ให้คัดความสัมพันธ์ที่มีฟังก์ชั่น Spawn ปรากฏอยู่ออกด้วย เพื่อป้องกันการเพิ่มขึ้นของภาคีจากการเติมคำสั่ง Spawn
3) สุ่มเลือก 1 ความสัมพันธ์รูปแบบลำดับฟังก์ชั่นการกระทำ และเติมลงในภาคีโดยใช้วิธีการตามที่ระบุในหัวข้อ 6.9 ทำซ้ำด้วยจำนวนครั้งตามตัวเลขที่ได้จากค่าสถิติ
4) เติมอากิวเมนท์ที่ขาดหายไปด้วยความสัมพันธ์แบบการเลือกใช้ฟังก์ชั่นข้อมูลด้วยวิธีที่ระบุในหัวข้อ 6.14
5) ลบบล็อกว่าง จากนั้นกำหนดตัวระบุค่าให้ Goto และ Spawn (ถ้ามี) ด้วยวิธีที่ระบุในหัวข้อ 6.15
6) ปรับปรุงภาคีด้วยฮิวริสติกโดยใช้วิธีตามหัวข้อ 6.16



รูปที่ 77 แสดงขั้นตอนที่ 1 ซึ่งเป็นการเลือกจำนวนความสัมพันธ์รูปแบบลำดับฟังก์ชั่นการกระทำที่จะใช้



รูปที่ 78 แสดงขั้นตอนที่ 2 – 3 ซึ่งเป็นการสุ่มความสัมพันธ์แบบลำดับฟังก์ชั่นการกระทำที่คัดแล้วมาเติมในสคริปต์ครั้งละ 1 ความสัมพันธ์



รูปที่ 79 แสดงขั้นตอนที่ 4 – 6 ซึ่งเป็นการเติมเต็มสล็อตจำเป็นที่มีอยู่และปรับปรุงสคริปต์ด้วยฮิวริสติก

การปรับปรุงความถี่ในการโจมตี

หนึ่งในสาเหตุที่ทำให้ศัตรูโจมตีถี่มากเกินหรือน้อยเกินไป คือ การที่มีฟังก์ชั่นการกระทำระยะยาวที่ใช้เวลาในการทำงานน้อยหรือมากเกินไปคั่นระหว่างฟังก์ชั่น Spawn ขั้นตอนนี้พยายามความปรับปรุงความถี่โดยการแก้ไขระยะเวลาทำงานของฟังก์ชั่น Wait ซึ่งเป็นหนึ่งในฟังก์ชั่นการกระทำระยะยาว สาเหตุที่เลือกปรับปรุงเฉพาะฟังก์ชั่น Wait เนื่องจากเป็นฟังก์ชั่นที่ส่งผลกระทบต่อพฤติกรรมโดยรวมน้อยที่สุด เนื่องจากฟังก์ชั่นการกระทำระยะยาวอื่น ๆ สามารถเปลี่ยนแปลงตำแหน่งของภาคีได้ นอกจากนี้การปรับปรุงระยะเวลาของฟังก์ชั่น Wait อาจช่วยในกรณีที่ศัตรูเคลื่อนที่ต่อเนื่องเกินไปจนหลบได้ยากด้วย

ประเด็นสำคัญในการปรับปรุงระยะเวลาหยุดรอของ Wait อยู่ที่การเลือกค่าที่เหมาะสม ในที่นี้เลือกใช้วิธีการเพิ่มค่าเป็น 2 เท่าในกรณีที่ศัตรูโจมตีถี่เกินไป และลดระยะเวลาเหลือ 75% ในกรณีที่ศัตรูโจมตีถี่น้อยเกินไป โดยสามารถพิจารณาว่าศัตรูโจมตีถี่มากหรือน้อยเกินไปได้จากอัตราหลบหลีกพลาดที่คำนวณได้ดังที่วิเคราะห์ในหัวข้อ 7.6 หากอัตราหลบหลีกพลาดสูงเกินค่าบรรทัดฐาน จะถือว่าศัตรูโจมตีถี่มากเกินไป และหากต่ำเกินค่าบรรทัด

ฐาน จะถือว่าศัตรูโจมตีถี่น้อยเกินไป การปรับระยะเวลาหยุดรอจะทำหลาย ๆ รอบเพื่อปรับแต่งจนได้ตัวเลขที่เหมาะสม และทำให้สคริปต์ผลลัพธ์สามารถยอมรับได้ตามเกณฑ์ ในงานวิทยานิพนธ์นี้จะทำสูงสุด 5 รอบ

วิธีการปรับระยะเวลาหยุดรอเป็น 2 เท่านี้ได้จากแนวคิดของการปรับสมดุลของเกมในกรณีที่ผู้ออกแบบไม่ทราบถึงค่าที่น่าจะเหมาะสม ซึ่ง Brenda [40] ได้แนะนำให้ใช้วิธีการเพิ่มค่าเป็นเท่าตัว สำหรับกรณีการปรับค่าลง Brenda ได้แนะนำให้ใช้กฎเดียวกันโดยการหารครึ่ง แต่ผู้วิจัยเล็งเห็นว่าการปรับแบบหารครึ่งอาจทำให้ค่าที่ปรับกลับไปเป็นค่าเดิมได้ จึงใช้การปรับลงมาที่ 75% แทน ยกตัวอย่างปัญหา เช่น ในกรณีที่รอบก่อนหน้าปรับด้วยวิธีการเพิ่ม 2 เท่า แล้วส่งผลให้อัตราหลบหลีกพลาดน้อยเกินไปจนต้องปรับค่าลง หากใช้วิธีหารครึ่ง จะทำให้ค่าที่ปรับกลับไปเป็นค่าเดิม ซึ่งไม่ผ่านการวัดผลเช่นกัน

หลังจากการปรับปรุงศัตรูที่อยู่นิ่ง และปรับปรุงความถี่ 5 รอบแล้ว ยังคงเหลือศัตรูบางส่วนที่ไม่ผ่านการวัดผลดังตารางที่ *8* และผลการทดลองหลังการปรับปรุงเทียบกับก่อนปรับปรุงเป็นไปดังตารางที่ *9*

ตารางที่ 8 จำนวนศัตรูที่ไม่ผ่านการวัดผลในแต่ละขั้นของการปรับปรุงแยกตามระดับ

การปรับปรุง	จำนวนศัตรูที่**ไม่ผ่าน**การวัดผลแยกตามระดับ				รวม
	Enemy	Elite	Miniboss	Boss	
ก่อนปรับปรุง	106	99	186	211	602
แก้ศัตรูที่อยู่นิ่ง	105	98	163	203	569
แก้ความถี่ (1)	78	77	139	187	481
แก้ความถี่ (2)	62	69	129	178	438
แก้ความถี่ (3)	55	62	124	169	410
แก้ความถี่ (4)	51	55	118	161	385
แก้ความถี่ (5)	47	51	110	159	367

ตารางที่ 9 จำนวนศัตรูที่ผ่านการวัดผลก่อนและหลังปรับปรุงแยกตามระดับ

การปรับปรุง	จำนวนศัตรูที่**ผ่าน**การวัดผลแยกตามระดับ				รวม
	Enemy	Elite	Miniboss	Boss	
ก่อนปรับปรุง	161	168	81	56	466
คิดเป็น %	**60.3**	**62.92**	**30.34**	**20.97**	**43.63**
หลังปรับปรุง	220	216	157	108	701
คิดเป็น %	**82.4**	**80.9**	**58.8**	**40.45**	**65.64**

จากผลการปรับปรุงจะเห็นว่าการปรับปรุงรอบหลัง ๆ มีแนวโน้มที่จะได้ศัตรูที่ผ่านการวัดผลจำนวนน้อยลง โดยเฉพาะศัตรูระดับ Boss ที่การปรับปรุงรอบที่ 5 ทำให้ศัตรูผ่านการวัดผลมากขึ้นเพียง 2 ตัวเท่านั้น ผู้วิจัยคาดว่าหากเพิ่มจำนวนรอบการปรับปรุงขึ้นอีกมากกว่า 1 รอบ ศัตรูระดับ Boss น่าจะไม่ได้ผลลัพธ์ที่ดีไปกว่าการปรับปรุงเพิ่ม 1 รอบแล้ว ในขณะที่ศัตรูระดับอื่น ๆ ก็น่าจะได้ศัตรูที่ผ่านการวัดผลจำนวนน้อยลงไปอีกเช่นกัน การที่แนวโน้มของจำนวนศัตรูที่ผ่านการวัดผลน้อยลงทุกรอบการปรับปรุงน่าจะเกิดจากการที่ศัตรูที่เหลืออยู่ไม่ผ่านการวัดผลจากสาเหตุอื่นซึ่งน่าจะต้องปรับแก้ในระดับขั้นตอนวิธี

7.8 ความถูกต้องของขั้นตอนวิธี

เพื่อเป็นการทดสอบว่าขั้นตอนวิธีที่นำเสนอในงานวิทยานิพนธ์นี้เป็นปัจจัยที่ทำให้รูปแบบพฤติกรรมที่สร้างขึ้นสามารถยอมรับได้จริงหรือไม่ ในที่นี้จึงจะทำการเปรียบเทียบผลลัพธ์ของขั้นตอนวิธีกับสัดส่วนของจำนวนศัตรูที่ยอมรับได้ในกรณีที่สร้างรูปแบบพฤติกรรมแบบสุ่มโดยไม่มีการพึ่งพาข้อมูลจากฐานข้อมูล

7.8.1 ขั้นตอนการสร้างรูปแบบพฤติกรรมแบบสุ่ม

การสร้างรูปแบบพฤติกรรมแบบสุ่มจะเริ่มจากการกำหนดตัวเลขต่าง ๆ ที่เกี่ยวข้องกับการสร้างภาคีก่อน และสร้างโครงภาคีเริ่มต้นโดยใช้ตัวเลขที่กำหนดเป็นตัวกำหนดจำนวนบล็อกชนิดต่าง ๆ ที่ปรากฏในภาคี จากนั้นจึงเติมข้อความสั่ง (ฟังก์ชั่นการกระทำรวมถึงโครงสร้างเงื่อนไขและโครงสร้างวนซ้ำ) ลงในภาคีอย่างสุ่มโดยอิงกับตัวเลขที่กำหนด แล้วจึงเติมเต็มสล็อตจำเป็น (อากิวเมนท์ของฟังก์ชั่น เงื่อนไขของโครงสร้างเงื่อนไข และจำนวนรอบทำซ้ำ) ด้วยฟังก์ชั่นข้อมูล ตัวดำเนินการ หรือสัญพจน์ที่สุ่มเลือกมา ขั้นตอนสุดท้ายเป็นการคัดบล็อกที่ไม่เหมาะสมออก

ตัวเลขที่เกี่ยวข้องกับการสร้างภาคีและการเตรียมตัวเลข

ตัวเลขจะอยู่ในรูปของช่วงค่าเพื่อให้ขั้นตอนสร้างภาคีสุ่มเลือกตัวเลขในช่วงค่าไปใช้งาน ตัวเลขเหล่านี้จะระบุถึงโครงสร้างของภาคี ซึ่งประกอบด้วย

- จำนวนบล็อกภาคีในโครงภาคี
- จำนวนบล็อกสถานะในแต่ละบล็อกภาคี
- จำนวนบล็อกลำดับในแต่ละบล็อกสถานะ
- จำนวนฟังก์ชั่นการกระทำในแต่ละบล็อกลำดับ
- จำนวนโครงสร้างเงื่อนไขในแต่ละบล็อกลำดับ
- จำนวนโครงสร้างวนซ้ำในแต่ละบล็อกลำดับ

ตัวเลขเหล่านี้จะได้จากการสำรวจสคริปต์ทั้งหมดในชุดข้อมูลศัตรูและทำการเก็บจำนวนต่ำสุดและสูงสุดของค่าเหล่านี้ เพื่อจัดทำเป็นช่วงค่าสำหรับใช้งานในขั้นตอนต่อไป สาเหตุที่การสร้างรูปแบบพฤติกรรมอย่างสุ่มจำเป็นต้องพึ่งพาค่าจากชุดข้อมูลเนื่องจากการสุ่มตัวเลขใด ๆ มาใช้งานอาจทำให้ภาคีที่ต้องการสุ่มสร้างมีขนาดใหญ่หรือมีความซับซ้อนเกินกว่าที่เครื่องจะสามารถประมวลผลได้ เช่น การสุ่มตัวเลขใด ๆ อาจทำให้ได้โครงภาคีที่มีจำนวน 1,000 ภาคี

การสร้างโครงภาคีเริ่มต้น

ขั้นตอนนี้เป็นการสร้างโครงภาคีโดยการเติมบล็อกว่างลงในโครงภาคีเพื่อให้โครงภาคีพร้อมรับการเติมฟังก์ชั่นการกระทำและข้อความสั่งอื่น ๆ ในขั้นตอนต่อไป รายละเอียดขั้นตอนเป็นดังนี้

1) สุ่มจำนวนบล็อกภาคีที่จะใช้ และสร้างบล็อกภาคีว่างตามจำนวน แต่ละบล็อกภาคีว่างจะต้องมีบล็อกทำลาย และบล็อกเริ่มต้นที่ทำการกำหนดค่าคุณสมบัติของภาคีด้วยฟังก์ชั่น Set โดยในที่นี้จะใช้ค่าคุณสมบัติแม่แบบเดียวกับการสร้างรูปแบบพฤติกรรมในบทที่ 6
    - ภาคีแรกสุดที่ปรากฏในโครงภาคี จะถือว่าเป็นภาคีหลักและใช้ค่าคุณสมบัติแม่แบบดังนี้

ตำแหน่ง	320,240	พลังชีวิต	100
ทิศทาง	หันซ้าย	พลังโจมตี	10
ขนาดตัวตรวจจับการชน	32 x 48	ฝ่ายโจมตี	ใช่
ผลของแรงดึงดูด	1	ฝ่ายตั้งรับ	ใช่

- สำหรับภาคีที่เหลือจะถือว่าเป็นภาคีกระสุนและใช้ค่าคุณสมบัติแม่แบบดังนี้

ขนาดตัวตรวจจับการชน	16 x 16	พลังโจมตี	1
ภาคีกระสุน	ใช่	ฝ่ายโจมตี	ใช่

2) สำหรับแต่ละบล็อกภาคีในโครงภาคี ให้สุ่มจำนวนบล็อกสถานะที่จะใช้ แล้วเติมบล็อกสถานะว่างลงไปตามจำนวนที่สุ่มได้
3) สำหรับแต่ละบล็อกสถานะในโครงภาคี ให้สุ่มจำนวนบล็อกลำดับที่จะใช้ แล้วเติมบล็อกลำดับว่างลงไปตามจำนวนที่สุ่มได้

การเติมข้อความสั่งลงในโครงภาคี

ขั้นตอนนี้เป็นการเติมข้อความสั่งลงในทุกบล็อกลำดับของโครงภาคี โดยจำนวนข้อความสั่งแต่ละชนิดจะถูกสุ่มเลือกจากช่วงค่า และการเติมข้อความสั่งจะเติมแบบสุ่มลำดับ ขั้นตอนเป็นดังนี้

```
Let actionCount = Randomly picked value from corresponding usage interval
Let conditionCount = Randomly picked value from corresponding usage interval
Let loopCount = Randomly picked value from corresponding usage interval
Let array = []
For(i=1; i<=actionCount; i++) do: array ← 1
For(i=1; i<=conditionCount; i++) do: array ← 2
For(i=1; i<=loopCount; i++) do: array ← 3
Shuffle(array)
For( item in array ) do:
    If(item == 1) do: Add random action to current skeleton
    If(item == 2) do: Add condition block to current skeleton
    If(item == 3) do: Add loop block to current skeleton
```

สำหรับวิธีการเติมข้อความสั่งแต่ละชนิดลงในโครงภาคีจะเป็นดังนี้

- กรณีฟังก์ชั่นการกระทำ: สุ่มเลือกฟังก์ชั่นการกระทำใด ๆ จากฟังก์ชั่นการกระทำทั้งหมดที่มีอยู่ (ดังที่แสดงในภาคผนวก) แล้วเติมลงในโครงภาคี โดยแต่ละพารามิเตอร์ของฟังก์ชั่นให้ใช้สล็อตจำเป็นที่ระบุชนิดข้อมูล

ที่ตรงกันเป็นอากิวเมนท์ เช่น RunStraight(Direction dir, Decimal spd, Boolean actionEnd) จะถูกเติมเป็น RunStraight([CRIT-Direction], [CRIT-Decimal], [CRIT-Boolean]); เป็นต้น

- กรณีโครงสร้างเงื่อนไข: สุ่มเลือกชนิดของโครงสร้างเงื่อนไขก่อนว่าเป็นโครงสร้างที่มีบล็อก Else หรือไม่ จากนั้นจึงเติมโครงสร้างลงไป โดยเงื่อนไขให้แทนที่ด้วยสล็อตจำเป็น [CRIT-Boolean]
- กรณีโครงสร้างวนซ้ำ: เติมโครงสร้างลงไปโดยจำนวนรอบทำซ้ำให้แทนที่ด้วยสล็อตจำเป็น [CRIT-Int]

การเติมเต็มสล็อตจำเป็น

การเติมเต็มจะเติมโดยให้ชนิดของข้อมูลถูกต้องตามชนิดข้อมูลที่สล็อตต้องการ วิธีเติมสล็อตจำเป็นแบ่งออกเป็น 3 กรณี ได้แก่ การเติมสล็อตจำเป็นของฟังก์ชั่น Goto, การเติมสล็อตจำเป็นของฟังก์ชั่น Spawn และการเติมสล็อตจำเป็นอื่น ๆ

- กรณีฟังก์ชั่น Goto: ให้สุ่มเลือกตัวระบุของบล็อกสถานะที่อยู่ในภาคีเดียวกับฟังก์ชั่น Goto เท่านั้น
- กรณีฟังก์ชั่น Spawn: ให้สุ่มเลือกตัวระบุของบล็อกภาคีที่อยู่ในโครงภาคีเท่านั้น
- กรณีอื่น ๆ : แบ่งเป็น 2 กรณีย่อยขึ้นกับชนิดข้อมูลที่สล็อตต้องการ หากชนิดข้อมูลดังกล่าวมีสัญพจน์ให้ใช้งานได้ จะทำการสุ่มว่าจะเติมสล็อตด้วยสัญพจน์หรือไม่ก่อน หากต้องการเติมด้วยสัญพจน์จะใช้วิธีสุ่มดังนี้

ชนิดข้อมูล	วิธีเติม
Boolean	สุ่มค่าเป็น true หรือ false
Decimal	สุ่มค่าในช่วง $[2^{-126}, (2-2^{-23})\cdot 2^{127}]$ ซึ่งเป็นช่วงค่าต่ำสุดถึงสูงสุดของข้อมูลชนิด Float ในภาษา Java
Int	สุ่มค่าเป็นจำนวนเต็มในช่วง $[-2^{31}, 2^{31}-1]$
Direction	สุ่มค่ามุมในช่วง [0, 360)
Position	สุ่มตำแหน่งแบบคาร์ทีเซียนโดยค่าแกน x และ y จะสุ่มจากช่วง $[2^{-126}, (2-2^{-23})\cdot 2^{127}]$
Collider	สุ่มความกว้างและความสูงของตัวตรวจจับการชนเป็นจำนวนเต็มบวกในช่วง $[1, 2^{31}-1]$

หากไม่ใช้สัญพจน์ จะทำการสุ่มเลือกฟังก์ชั่นข้อมูลหรือตัวดำเนินการที่คืนค่าเป็นชนิดข้อมูลที่เข้ากันได้มาเติม โดยชนิดข้อมูลจะเข้ากันได้ก็ต่อเมื่อเงื่อนไขใดเงื่อนไขหนึ่งต่อไปนี้เป็นจริง

- ชนิดข้อมูลทั้งสองเป็นชนิดข้อมูลเดียวกัน
- สล็อตจำเป็นต้องการข้อมูลประเภท Decimal และชนิดข้อมูลที่จะนำมาเติมเป็น Int

นอกจากนี้จะมีกฎการเติมที่เฉพาะเจาะจงกับบางฟังก์ชั่น ได้แก่

- อนุญาตให้เติมฟังก์ชั่น Random ในสล็อตจำเป็นประเภท Decimal และ Direction เท่านั้น โดยเมื่อเติมแล้ว จะถือว่าสล็อตข้อมูลชนิด Set<Any> ของ Random เป็น Set<Decimal> หรือ Set<Direction> ขึ้นอยู่กับชนิดข้อมูลของสล็อตที่นำ Random ไปเติม เช่น เติม Random ใน FlipDirection([CRIT-Direction]) จะได้เป็น FlipDirection(Random([CRIT-Set<Direction>])) สาเหตุที่ต้องมีข้อจำกัดนี้เนื่องจากฟังก์ชั่นข้อมูลที่ใช้ในงานวิทยานิพนธ์ที่คืนค่าชนิด Set สามารถคืนค่าได้เพียง Set<Decimal> หรือ Set<Direction> เท่านั้น

- ฟังก์ชั่น B_Equal และ B_NotEqual (ตัวดำเนินการ == และ !=) เมื่อเติมแล้วจะต้องสุ่มเปลี่ยนสล็อตชนิด Any เป็นชนิด Boolean หรือ Decimal โดยทั้ง 2 สล็อตจะต้องเปลี่ยนเป็นชนิดข้อมูลเดียวกัน เนื่องจากการเปรียบเทียบความเท่ากันของข้อมูลที่มีความหมายมีเพียงการเปรียบเทียบข้อมูล 2 ชนิดนี้เท่านั้น

การคัดบล็อกที่ไม่เหมาะสมออก

โครงสร้างวนซ้ำที่มีจำนวนรอบการวนซ้ำสูงมากโดยที่บล็อกลำดับของโครงสร้างไม่มีฟังก์ชั่นการกระทำระยะยาว จะทำให้การทำงานวนซ้ำของบล็อกไม่พบเงื่อนไขพักและต้องวนซ้ำจนครบจำนวนรอบ ซึ่งอาจใช้ระยะเวลายาวนานมากจนตัวเกมเสมือนหยุดการทำงานไป ในที่นี้จึงต้องคัดโครงสร้างวนซ้ำที่มีลักษณะดังกล่าวออก

7.8.2 การทดลองและเปรียบเทียบผลลัพธ์

งานวิทยานิพนธ์นี้ทำการสร้างรูปแบบพฤติกรรมแบบสุ่มเป็นจำนวน 267 ไฟล์ จากนั้นคัดลอกสคริปต์ที่สร้างเป็น 4 ไฟล์เพื่อใช้เป็นศัตรู 4 ระดับ และปรับค่าพลังชีวิตเริ่มต้นตามระดับด้วยค่าเดียวกับที่ใช้กับการวัดผลรูปแบบพฤติกรรมที่สร้างขึ้นในหัวข้อ 7.5.2 แล้วจึงใช้ปัญญาประดิษฐ์วัดผลศัตรูทั้ง 1068 ตัวที่ได้นี้ด้วยวิธีเดียวกับหัวข้อ 7.5.2 แต่ทำการต่อสู้เพียงตัวละ 1 รอบแทน โดยผลลัพธ์เป็นดังตารางที่ *10*

ตารางที่ 10 ผลการวัดผลศัตรูที่สร้างขึ้นด้วยวิธีสุ่มแยกตามระดับ

การวัดผล	จำนวนศัตรูแยกตามระดับ				รวม
	Enemy	Elite	Miniboss	Boss	
ผ่าน (ยอมรับได้)	66	65	15	1	147
ไม่ผ่าน	201	202	252	266	921
% ที่ผ่าน	**24.72**	**24.34**	**5.62**	**0.37**	**13.76**

เมื่อเปรียบเทียบกับผลที่ได้จากขั้นตอนวิธีที่นำเสนอ (43.63% ก่อนปรับปรุงและ 65.64% หลังปรับปรุง) จะพบว่าจำนวนศัตรูที่ถือว่ายอมรับได้จากกรณีสร้างแบบสุ่มมีน้อยกว่าพอสมควร โดยค่าประสิทธิภาพที่ทำให้ศัตรูกรณีสุ่มไม่ผ่านการวัดผลจะแตกต่างกันไปตามระดับของศัตรู ศัตรูระดับ Enemy และ Elite จะไม่ผ่านการวัดผลเพราะปราบศัตรูไม่สำเร็จเป็นส่วนใหญ่ ในขณะที่กรณีของศัตรูระดับ Miniboss และ Boss ไม่ผ่านการวัดผลจากทั้งค่าอัตราหลบหลีกพลาดและปราบศัตรูไม่สำเร็จ โดยค่าอัตราหลบหลีกพลาดเป็นตัวแปรหลักที่ทำให้ไม่ผ่าน

กรณีที่ปราบศัตรูไม่สำเร็จสำหรับศัตรูทุกระดับนั้น ส่วนใหญ่จะเกิดจากการที่ศัตรูไม่อยู่ในจุดที่ผู้เล่นโจมตีใส่ได้ โดยตัวศัตรูจะเคลื่อนที่ขึ้นไปติดอยู่ด้านบนของฉาก โดยส่วนใหญ่จะเป็นกรณีที่ศัตรูมีรูปแบบพฤติกรรมที่ทำให้เคลื่อนที่แต่อากิวเมนท์ที่เกี่ยวข้องไม่เหมาะสม เช่น ฟังก์ชั่น RunStraight มีทิศทางที่สุ่มได้เป็นทิศขึ้นบน (ทิศที่มีค่าองศาในช่วงค่า (0,180)) เกิดร่วมกับค่าความเร็วที่สูงเกินไปจนสามารถเอาชนะแรงดึงดูดของฉากได้ เป็นต้น ความ

ไม่เข้ากันเช่นนี้เกิดจากวิธีสร้างแบบสุ่มที่ไม่มีการเรียนรู้ข้อมูลที่ควรนำไปใช้เป็นอากิวเมนท์ ทำให้อากิวเมนท์ที่นำมาใช้ไม่เหมาะสมกับฟังก์ชั่นการกระทำ

สำหรับกรณีที่ไม่ผ่านการวัดผลจากค่าอัตราหลบหลีกพลาด ส่วนใหญ่เป็นกรณีที่อัตราหลบหลีกพลาดต่ำเกินไป โดยมีต้นเหตุหลัก ๆ 2 ประการ ได้แก่

- ศัตรูอยู่นิ่งเพราะสุ่มได้ฟังก์ชั่นการกระทำที่ไม่มีการเคลื่อนไหวหรือในบางครั้งก็ได้เป็นการกระทำที่มีการเคลื่อนไหวแต่มีอากิวเมนท์ไม่เหมาะสม เช่น ฟังก์ชั่น RunStraight ที่วิ่งในทิศลง ซึ่งการกระทำระยะยาวที่ไม่ดีเช่นนี้ บางครั้งก็เกิดร่วมกับการสุ่มได้ค่า false สำหรับพารามิเตอร์ actionEnd ทำให้ฟังก์ชั่นการกระทำนั้นทำงานตลอดไปทั้งที่เป็นฟังก์ชั่นที่ให้ผลลัพธ์ไม่ดี
- มีคำสั่ง Despawn ตั้งแต่ช่วงแรกเริ่มของการต่อสู้ ทำให้การต่อสู้จบอย่างรวดเร็วโดยไม่มีโอกาสให้ศัตรูได้โจมตีใส่ตัวละครอวตาร กรณีนี้จะเกิดขึ้นได้ง่ายมากเมื่อเทียบกับการสร้างรูปแบบพฤติกรรมจากข้อมูล เนื่องจากกรณีที่มีการใช้ข้อมูลมาคำนวณร่วม ขั้นตอนวิธีจะรู้ว่าพฤติกรรมเช่นนี้ไม่ใช่พฤติกรรมที่เกิดได้บ่อย โดยสังเกตได้จากจำนวนการใช้งานความสัมพันธ์ลำดับข้ามสถานะกรณี Despawn ที่ปรากฏในตารางค่าสถิติ ซึ่งต่างกับการสร้างรูปแบบพฤติกรรมแบบสุ่มที่อาจสุ่มได้คำสั่ง Despawn เมื่อไรก็ได้ ในบล็อกลำดับใดก็ได้

ด้วยเหตุนี้ จึงอาจสรุปได้ว่า การสร้างรูปแบบพฤติกรรมโดยมีการใช้ข้อมูลต่าง ๆ มาสนับสนุนด้วยการทำเหมืองข้อมูลด้วยวิธีการที่นำเสนอในงานวิทยานิพนธ์นี้ ช่วยให้สามารถสร้างศัตรูที่ยอมรับได้เป็นอัตราส่วนที่มากขึ้นอย่างเห็นได้ชัด ขั้นตอนวิธีที่นำเสนอจะประกอบฟังก์ชั่นต่าง ๆ เข้าด้วยกันด้วยอัตราส่วนและค่าซึ่งเคยถูกใช้งานสำเร็จมาแล้ว จึงลดรูปแบบพฤติกรรมที่ไม่พึงประสงค์ได้มากขึ้น

บทที่ 8
สรุปผลการวิจัยและข้อเสนอแนะ

8.1 สรุปผลการวิจัย

งานวิทยานิพนธ์นี้นำเสนอขั้นตอนวิธีสำหรับสร้างรูปแบบพฤติกรรมของศัตรูในเกมแอ็คชั่นสองมิติแบบเน้นตัวละครโดยอัตโนมัติ ซึ่งเป็นการสร้างโดยอาศัยข้อมูลความสัมพันธ์ของพฤติกรรมต่าง ๆ ที่ได้จากการทำเหมืองข้อมูล โดยมีข้อมูลขาเข้าเป็นรูปแบบพฤติกรรมของศัตรูในเกมแอ็คชั่นต่าง ๆ ที่อยู่ในรูปของภาษาอธิบายภาคี ซึ่งเป็นภาษาที่งานวิทยานิพนธ์นี้นำเสนอเพื่อใช้ในการอธิบายรูปแบบพฤติกรรมของภาคีในเกมแอ็คชั่น รูปแบบพฤติกรรมผลลัพธ์ที่สร้างขึ้นจะอยู่ในรูปภาษาอธิบายภาคี และเก็บบันทึกไว้เป็นไฟล์สคริปต์ศัตรูที่เสร็จสมบูรณ์ ซึ่งสามารถนำไปวัดผลได้โดยให้ทำการต่อสู้กับผู้เล่นเพื่อเก็บข้อมูลการต่อสู้ และใช้ข้อมูลการต่อสู้นี้มาคำนวณเป็นค่าประสิทธิภาพการต่อสู้ที่ใช้สำหรับวัดการยอมรับได้ตามนิยามที่กำหนดไว้ในงานวิทยานิพนธ์นี้ ผู้เล่นที่ทำการต่อสู้กับศัตรูเพื่อวัดผลรูปแบบพฤติกรรมที่สร้างไม่ใช่ผู้เล่นมนุษย์แต่ใช้ปัญญาประดิษฐ์เล่นแทน เนื่องจากจำนวนรอบการต่อสู้ที่สูงมากจนไม่เหมาะสมที่จะให้ผู้เล่นมนุษย์เป็นผู้ควบคุมตัวละครอวตาร ปัญญาประดิษฐ์ที่จัดทำขึ้นเป็นปัญญาประดิษฐ์ที่ผู้วิจัยพยายามทำให้เล่นเหมือนผู้เล่นมนุษย์ โดยใช้ความคล้ายคลึงของค่าประสิทธิภาพการต่อสู้เทียบกับผู้เล่นมนุษย์เป็นเกณฑ์ในการตัดสิน

จากการทดลองสร้างศัตรูด้วยขั้นตอนวิธีที่นำเสนอ 300 ตัว พบว่าเป็นศัตรูที่ถูกขั้นตอนวิธียกเลิกการสร้างไป 33 ตัว คงเหลือศัตรูที่สามารถสร้างจนเสร็จจำนวน 267 ตัว หลังจากถูกจัดทำเป็นศัตรู 4 ระดับ และวัดผลแล้วพบว่ามีเพียงศัตรูระดับ Enemy และ Elite ที่ผ่านการวัดผล ถือว่าเป็นศัตรูที่ยอมรับได้เกินกว่า 50% ในขณะที่ศัตรูกลุ่ม Miniboss และ Boss เป็นศัตรูที่ยอมรับได้เพียง 20-30% เท่านั้น และมีจำนวนศัตรูที่ยอมรับได้รวม 43.63% เท่านั้น ซึ่งผู้วิจัยเห็นว่าเป็นจำนวนที่น้อยเกินไป จึงได้ตรวจสอบหาปัญหาที่ทำให้ผลลัพธ์ออกมาไม่ดี และเสนอวิธีปรับปรุงสคริปต์ผลลัพธ์ ซึ่งในท้ายที่สุด ทำให้จำนวนศัตรูที่ยอมรับได้คิดเป็น 65.64% ซึ่งนับได้ว่ามีศัตรูที่สามารถนำไปใช้ได้ในอัตราส่วนที่มีประสิทธิภาพ แต่วิธีการปรับปรุงสคริปต์ที่นำเสนอมีข้อเสียตรงที่ต้องใช้เวลาในการปรับปรุงแต่ละรอบนาน เนื่องจากต้องทดลองต่อสู้จริงหลายครั้งเพื่อตรวจสอบว่าผ่านการวัดผลหรือไม่

นอกจากนี้ งานวิทยานิพนธ์ได้เปรียบเทียบผลลัพธ์ที่ได้กับผลลัพธ์จากกรณีที่รูปแบบพฤติกรรมถูกสร้างด้วยวิธีสุ่ม พบว่าผลลัพธ์จากวิธีสร้างรูปแบบพฤติกรรมแบบสุ่มมีศัตรูที่ยอมรับได้เพียง 13.76% ซึ่งแสดงให้เห็นว่าการใช้ข้อมูลสนับสนุนจากการทำเหมืองข้อมูลในการสร้างรูปแบบพฤติกรรม ช่วยเพิ่มอัตราส่วนของศัตรูที่ยอมรับได้มากขึ้นอย่างเห็นได้ชัด

8.2 ข้อเสนอแนะ

จากการวิเคราะห์ผลการทดลองที่ได้ในบทที่ 7 ผู้วิจัยได้แสดงให้เห็นถึงปัจจัยต่าง ๆ ที่น่าจะเป็นต้นตอของปัญหาที่ทำให้รูปแบบพฤติกรรมที่สร้างขึ้นไม่สามารถยอมรับได้ตามเกณฑ์ที่กำหนด หัวข้อนี้จะกล่าวถึงแนวทางที่ผู้วิจัยคาดว่าน่าจะช่วยแก้ไขปัญหาที่กล่าวถึงได้ โดยแบ่งเป็นกรณีต่าง ๆ

ปัญหาที่เกิดจากปัญญาประดิษฐ์มาจากการที่ปัญญาประดิษฐ์ที่นำมาใช้มีทั้งส่วนที่เก่งเกินมนุษย์ผู้เล่น คือ การควบคุมที่มีความแม่นยำเกินไป และส่วนที่สู้มนุษย์ผู้เล่นไม่ได้ คือ การที่ไม่มีฐานความรู้ ไม่สามารถจดจำรูปแบบการต่อสู้ของศัตรูที่เกิดขึ้นไปแล้วได้ แต่ถึงแม้จะมีปัญหาที่กล่าวไป ปัญญาประดิษฐ์เองก็ยังถือว่าใช้งานได้ตามเกณฑ์ที่กำหนด แสดงให้เห็นว่า อาจมีความจำเป็นในการปรับปรุงเกณฑ์สำหรับวัดผลปัญญาประดิษฐ์ด้วย

- กรณีของการควบคุมที่แม่นยำเกินไป อาจจำลองความไม่แม่นยำของมนุษย์โดยการใส่สัญญาณรบกวนลงในการกดปุ่มควบคุมปัญญาประดิษฐ์ หรือทำให้การคาดการณ์ที่ได้จากการวิเคราะห์ความถดถอยมีความคลาดเคลื่อนไป
- กรณีของฐานความรู้ วิธีที่ง่ายที่สุดน่าจะเป็นการอนุญาตให้ปัญญาประดิษฐ์สามารถจำลองอนาคตที่จะเกิดขึ้นได้หลังจากระยะเวลาการต่อสู้ผ่านไประยะหนึ่ง เสมือนว่า ปัญญาประดิษฐ์ได้ใช้เวลาก่อนหน้านี้ในการเรียนรู้รูปแบบการโจมตีแล้ว แต่วิธีนี้อาจมองได้ว่าทำให้ปัญญาประดิษฐ์เก่งเกินผู้เล่นมนุษย์ไปได้เช่นกัน เพราะการโจมตีของศัตรูมีโอกาสเกิดจากรูปแบบพฤติกรรมหลายรูปแบบที่ทำงานพร้อมกัน ซึ่งผู้เล่นมนุษย์อาจแยกได้ยากว่า การโจมตีใดเป็นของรูปแบบพฤติกรรมชุดไหน

ปัญหาที่เกิดจากรูปแบบพฤติกรรมไม่พึงประสงค์โดยส่วนใหญ่มีต้นตอมาจากขั้นตอนการทำเหมืองข้อมูลที่สำรวจความสัมพันธ์ได้ไม่หลากหลายเพียงพอต่อการนำมาใช้งาน เช่น กรณีของการโจมตีที่ถี่เกินไปเกิดจากการที่ไม่ได้ทดลองทำเหมืองข้อมูลเพื่อหารูปแบบความสัมพันธ์ระหว่างลำดับพฤติกรรมและอากิวเมนท์ที่ปรากฏภายในฟังก์ชั่นการกระทำ เพื่อดูว่าหากพฤติกรรมมีลำดับเป็นเช่นนี้แล้ว ค่าอากิวเมนท์ใดจะเป็นค่าที่เหมาะสม เป็นต้น ในที่นี้สามารถสรุปปัญหาเป็นกรณีย่อยแยกตามต้นตอของปัญหาพร้อมแนวทางการแก้ไขได้ดังนี้

- กรณีที่ไม่ได้จำกัดจำนวนครั้งของการใช้งานฟังก์ชั่น จนส่งผลให้การโจมตีที่เกิดขึ้นถี่เกินไป แนวทางการแก้ไขที่ผู้วิจัยคาดว่าช่วยบรรเทาปัญหาลงได้ คือ การนับจำนวนฟังก์ชั่นที่สามารถใช้งานได้ต่อภาคี โดยนับแยกกันระหว่างภาคีแต่ละระดับ เนื่องจากตามเกณฑ์การแบ่งระดับของศัตรูแล้ว ศัตรูแต่ละระดับจะมีความซับซ้อนที่ต่างกัน ซึ่งส่งผลให้จำนวนฟังก์ชั่นที่นำมาใช้แตกต่างกันออกไปได้ด้วย เมื่อนับแล้วจึงนำข้อมูลนี้มาใช้ร่วมกับจำนวนความสัมพันธ์ในขั้นตอนการสร้างรูปแบบพฤติกรรมในบทที่ 6
- กรณีที่ไม่มีการทำเหมืองข้อมูลเพื่อหาความสัมพันธ์ระหว่างลำดับพฤติกรรมและอากิวเมนท์ที่ใช้ ซึ่งน่าจะเป็นปัจจัยที่ทำให้ลำดับพฤติกรรมที่สร้างขึ้นมีอากิวเมนท์บางตัวที่มีค่าไม่เหมาะสม จนเกิดกรณีอย่างศัตรูโจมตีถี่มากหรือน้อยเกินไปขึ้น แนวทางการแก้ไขจึงเป็นการทดลองทำเหมืองข้อมูลโดยนำเอาอากิวเมนท์มาวิเคราะห์ร่วมด้วย แต่เนื่องจากอากิวเมนท์นั้นมีโอกาสเป็นได้หลายค่ามาก จึงอาจจะต้องมีวิธีการควอนไทซ์ (Quantization) ข้อมูลออกเป็นกลุ่มเพื่อให้ข้อมูลซ้ำกันได้ และถูกนับเป็นรูปแบบเดียวกัน
- กรณีที่ไม่มีการทำเหมืองข้อมูลเพื่อหาความสัมพันธ์แบบการไม่ปรากฏขึ้นพร้อมกัน ซึ่งเป็นความสัมพันธ์ที่จะช่วยคัดกรองลำดับพฤติกรรมบางอย่างที่ไม่ควรปรากฏพร้อมกันออกไป แนวทางแก้ไขจึงเป็นการทดลองทำเหมืองข้อมูล แต่เนื่องจากกรณีนี้เป็นการหาว่าสิ่งใดไม่ควรปรากฏพร้อมกัน ผู้วิจัยจึงคิดว่าข้อมูลขาเข้าควรจะต้องเป็นข้อมูลที่ไม่ดี เพื่อหารูปแบบพฤติกรรมที่ไม่ดี ดังนั้น การทำเหมืองข้อมูลกรณีนี้ น่าจะต้องมีการสร้างชุดข้อมูลศัตรูอีกชุดหนึ่งซึ่งคัดเฉพาะศัตรูที่ไม่ดี เช่น ศัตรูที่สร้างจากขั้นตอนวิธีแต่ไม่ผ่านการวัดผล แล้วจึงทำเหมืองข้อมูลบนชุดข้อมูลใหม่นี้

- กรณีที่ไม่มีการทำเหมืองข้อมูลเพื่อหาความสัมพันธ์แบบการไม่เป็นลำดับของกันและกัน ซึ่งนับได้ว่าเป็นกรณีเฉพาะของกรณีก่อนหน้า วิธีการเตรียมชุดข้อมูลจะเป็นไปในทางเดียวกัน แต่แตกต่างกันที่วิธีการแปลงข้อมูลเพื่อให้สามารถทำเหมืองข้อมูลเพื่อดึงเอารูปแบบความสัมพันธ์ที่ต้องการออกมา

การทำเหมืองข้อมูลแบบต่าง ๆ ที่ได้กล่าวถึงข้างต้นเป็นการหารูปแบบความสัมพันธ์ที่มีความซับซ้อนสูง และต้องการชุดข้อมูลที่มีขนาดใหญ่กว่าที่ใช้ในงานวิทยานิพนธ์นี้เพื่อให้มีโอกาสพบเจอรูปแบบความสัมพันธ์ที่ถี่มากเพียงพอ

นอกเหนือจากแนวทางปรับปรุงขั้นตอนวิธีตามปัญหาที่พบ ยังมีประเด็นที่ผู้วิจัยคิดว่าน่าสนใจ สามารถทดลองต่อยอดได้อีก เช่น การนำเอาค่าคุณสมบัติเริ่มต้นที่ถูกกำหนดในบล็อกเริ่มต้นมาคิดร่วมในการสร้างรูปแบบพฤติกรรม เพื่อให้สามารถสร้างค่าคุณสมบัติโดยอัตโนมัติ แทนการใช้ค่าคุณสมบัติแม่แบบเหมือนในงานวิทยานิพนธ์นี้

ท้ายที่สุดนี้ วิธีการที่นำเสนอในงานวิทยานิพนธ์ มีแนวคิดหลักคือการสร้างสคริปต์ขึ้นจากชุดข้อมูล ซึ่งผู้วิจัยคิดว่าน่าจะสามารถใช้กับงานลักษณะอื่นได้ เช่น การสร้างโปรแกรมโดยอัตโนมัติขึ้นจากโปรแกรมที่มีอยู่แล้ว เป็นต้น

รายการอ้างอิง

1 http://thaipublica.org/wp-content/uploads/2014/06/EM-Outlook-2014-2018_Thailand_Findings-1.pdf, accessed 26/06/2017 2017

2 Togelius, J., Yannakakis, G.N., Stanley, K.O., and Browne, C.: 'Search-based procedural content generation: A taxonomy and survey', IEEE Transactions on Computational Intelligence and AI in Games, 2011, 3, (3), pp. 172-186

3 Adams, E.: 'Fundamentals of Game Design' (Pearson Education Inc, 2010, 2nd edn. 2010)

4 Scott Rogers: 'LEVEL UP! The Guide to Great Video Game Design' (John Wiley & Sons, Ltd, 2010. 2010)

5 Mark Davies: 'Designing Character-based Console Games' (Thompson Learning Inc, 2007, 1st edn. 2007)

6 Jiawei Han, and Micheline Kamber: 'Data Mining: Concepts and Techniques' (Morgan Kaufmann, 2006, 2nd edn. 2006)

7 Agrawal, R., and Srikant, R.: 'Mining sequential patterns'. Proc. Data Engineering, 1995. Proceedings of the Eleventh International Conference on1995 pp. Pages

8 Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C.: 'Mining sequential patterns by pattern-growth: The prefixspan approach', IEEE Transactions on knowledge and data engineering, 2004, 16, (11), pp. 1424-1440

9 Yan, X., and Han, J.: 'gspan: Graph-based substructure pattern mining'. Proc. Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on pp. 721-724

10 https://www2.informatik.uni-erlangen.de/EN/research/zold/ParSeMiS/, accessed 11/06/2015 2015

11 Stuart Russell, and Peter Norvig: 'Artificial Intelligence A Modern Approach' (Pearson Education Inc, 2009, 3rd edn. 2009)

12 Bulatov, A.A.: 'Complexity of conservative constraint satisfaction problems', ACM Transactions on Computational Logic (TOCL), 2011, 12, (4), pp. 24

13 https://osolpro.atlassian.net/wiki/display/JACOP/JaCoP+-+Java+Constraint+Programming+solver, accessed 26/06/2017 2017

14 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein: 'Introduction to Algorithms' (The MIT Press, 2009, 3rd edn. 2009)

15 https://onlinecourses.science.psu.edu/stat501/node/250, accessed 27/06/2017 2017

16 http://www.public.iastate.edu/~maitra/stat501/lectures/MultivariateRegression.pdf, accessed 27/06/2017 2017

17 McKay, C.P.: 'What is life—and how do we search for it in other worlds?', PLoS biology, 2004, 2, (9), pp. e302

18 Miconi, T., and Channon, A.: 'An improved system for artificial creatures evolution', Proceedings of Artificial Life X, 2006, pp. 255-261

19 Miconi, T.: 'Evosphere: evolutionary dynamics in a population of fighting virtual creatures'. Proc. Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on pp. 3066-3073

20 Djezzar, N., Djedi, N., Cussat-Blanc, S., Luga, H., and Duthen, Y.: 'L-systems and artificial chemistry to develop digital organisms'. Proc. Artificial Life (ALIFE), 2011 IEEE Symposium on2011 pp. Pages

21 Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A.: 'Procedural content generation for games: A survey', ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 2013, 9, (1), pp. 1

22 Hidalgo, J.L., Camahort, E., Abad, F., and Vicent, M.J.: 'Procedural graphics model and behavior generation'. Proc. International Conference on Computational Science2008 pp. Pages

23 Hastings, E.J., Guha, R.K., and Stanley, K.O.: 'Automatic content generation in the galactic arms race video game', IEEE Transactions on Computational Intelligence and AI in Games, 2009, 1, (4), pp. 245-263

24 Nelson, M., and Mateas, M.: 'Towards automated game design', AI* IA 2007: Artificial Intelligence and Human-Oriented Computing, 2007, pp. 626-637

25 Sicart, M.: 'Defining game mechanics', Game Studies, 2008, 8, (2), pp. 1-14

26 http://www.gamasutra.com/blogs/LewisPulsipher/20120109/90875/Game_descriptions_rules_and_mechanics_what_are_the_differences_and_similarities.php, accessed 26/06/2017 2017

27 Togelius, J., and Schmidhuber, J.: 'An experiment in automatic game design'. Proc. Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On pp. 111-118

28 Smith, A.M., and Mateas, M.: 'Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games'. Proc. Computational Intelligence and Games (CIG), 2010 IEEE Symposium on pp. 273-280

29 Cook, M., Colton, S., and Gow, J.: 'Initial Results from Co-operative Co-evolution for Automated Platformer Design'. Proc. EvoApplications2012 pp. Pages

30 Zook, A., and Riedl, M.O.: 'Generating and adapting game mechanics'. Proc. Proceedings of the 2014 Foundations of Digital Games Workshop on Procedural Content Generation in Games, Ft. Lauderdale, Florida

31 Tobias Mahlmann: 'Modeling and Generating Strategy Games Mechanics', IT University of Copenhagen, 2013

32 http://www.gamasutra.com/view/feature/130268/a_detailed_crossexamination_of_.php, accessed 26/06/2017 2017

33 Hullett, K., and Whitehead, J.: 'Design patterns in FPS levels'. Proc. proceedings of the Fifth International Conference on the Foundations of Digital Games pp. 78-85

34 http://www.gamasutra.com/view/feature/1630/breaking_down_breakout_system_and_.php, accessed 26/06/2017 2017

35 Smith, G., Whitehead, J., Mateas, M., Treanor, M., March, J., and Cha, M.: 'Launchpad: A rhythm-based level generator for 2-d platformers', IEEE Transactions on Computational Intelligence and AI in Games, 2011, 3, (1), pp. 1-16

36 Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.-W., and Tseng, V.S.: 'SPMF: a Java open-source pattern mining library', The Journal of Machine Learning Research, 2014, 15, (1), pp. 3389-3393

37 Manika Verma, and Devarshi Mehta: ‘Sequential Pattern Mining: A Comparison between GSP, SPADE and Prefix SPAN’, International Journal of Engineering Development and Research, 2014, 2, (3), pp. 3016-3036

38 Ishio, T., Date, H., Miyake, T., and Inoue, K.: ‘Mining coding patterns to detect crosscutting concerns in java programs’. Proc. Reverse Engineering, 2008. WCRE'08. 15th Working Conference on pp. 123-132

39 https://www.humanbenchmark.com/tests/reactiontime, accessed 27/06/2017 2017

40 Brenda Brathwaite, and Ian Schreiber: ‘Challenges for Game Designers’ (Course Technology, 2009. 2009)

ภาคผนวก

รายการชนิดข้อมูลที่ไม่มีสัญพจน์

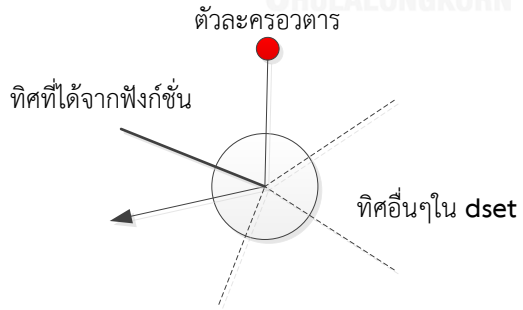
หัวข้อนี้แสดงรายการชนิดข้อมูลที่ไม่มีสัญพจน์ แต่ใช้งานได้โดยอาศัยฟังก์ชั่นข้อมูลที่คืนค่าเป็นข้อมูลชนิดที่ต้องการแทน

ชนิดข้อมูล	ความหมาย
Any	ข้อมูลชนิดใดก็ได้ หากมี Any มากกว่า 1 ตัวปรากฏในรายการพารามิเตอร์ ชนิดข้อมูลของ Any ทุกตัวจะต้องเหมือนกันทั้งหมด
Dynamic	ตัวอ้างอิงไปยังภาคีที่ปรากฏในโลกของเกม
Int	ข้อมูลจำนวนเต็ม ใช้สัญพจน์แบบเดียวกับข้อมูลชนิด Decimal แต่เมื่อนำมาใช้งานจะถูกปัดทศนิยมออกด้วยฟังก์ชั่น Floor
Set<Any>	รายการของข้อมูลชนิดใดก็ได้ อาจมีการเจาะจงชนิดข้อมูลได้ เช่น Set<Int> เป็นรายการของจำนวนเต็ม

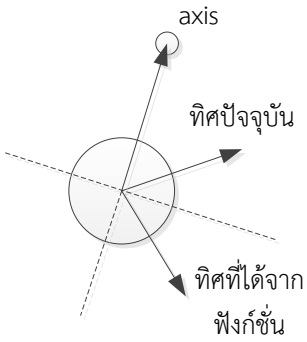
รายการฟังก์ชั่นการกระทำ

หัวข้อนี้แสดงรายการฟังก์ชั่นการกระทำที่ใช้ในงานวิทยานิพนธ์นี้ รวมถึงรายละเอียดของแต่ละฟังก์ชั่นโดย ID คือ หมายเลขแทนฟังก์ชั่น หากฟังก์ชั่นใดมีการทำเครื่องหมาย [Span] ไว้ แสดงว่าฟังก์ชั่นดังกล่าวเป็นฟังก์ชั่นการกระทำระยะยาว และถ้าหากฟังก์ชั่นใดแสดงชื่อฟังก์ชั่นในรูป FunctionName#Choice เช่น Set#atk แสดงว่าฟังก์ชั่นดังกล่าวเป็นฟังก์ชั่นที่มีพารามิเตอร์ตัวแรกเป็นตัวเลือก และให้ใช้รายการพารามิเตอร์ที่ปรากฏในตารางนี้ สำหรับวิธีการแปลงข้อมูลในบทที่ 5 เมื่อจะนำไปใช้งานในภาษาอธิบายภาคีจะต้องใช้ส่วน Choice ของชื่อฟังก์ชั่นเป็นพารามิเตอร์ตัวแรกแทน ตัวอย่างเช่น Set#atk จะแสดงรายการพารามิเตอร์เป็น Set#atk(Dynamic agentRef, Int value) เมื่อใช้งานในภาษาอธิบายภาคีจะต้องอยู่ในรูป Set(“atk”, ข้อมูลประเภท Dynamic, ข้อมูลประเภท Int) ในขณะเดียวกันฟังก์ชั่นที่ปรากฏอยู่ในสคริปต์ในรูป Set(“atk”, DynamicFilter(“this”), 5) เมื่อทำการแปลงข้อมูลจะต้องถือว่าฟังก์ชั่นดังกล่าวเป็นฟังก์ชั่น Set#atk โดยมี DynamicFilter(“this”) เป็นพารามิเตอร์ตัวแรก และตัวเลข 5 เป็นพารามิเตอร์ตัวที่สอง

ID: 1	AddExtraVelocityToPlayer(Direction **dir**, Decimal **spd**, Boolean **actionEnd**) [Span]
ขยับตัวละครอวตารไปในทิศ **dir** ด้วยความเร็ว **spd** พิกเซลต่อเฟรม ฟังก์ชั่นจะเสร็จสิ้นก็ต่อเมื่อ **actionEnd** คืนค่า true	

ID: 2	ChangeDirectionToPlayerByStep(Set<Direction> **dset**, int **delay**) [Span]
ทุก **delay** เฟรม จะเลือก 2 ทิศทางที่ใกล้กับทิศทางของตนเองที่สุดในทิศตามเข็มนาฬิกาและทวนเข็มนาฬิกาตามลำดับจาก **dset** จากนั้นเลือก 1 ทิศทางที่หันเข้าหาตัวละครอวตารมากที่สุดและเปลี่ยนทิศทางตนเองเป็นทิศที่เลือก เงื่อนไขเพิ่มเติม คือ **dset** ต้องเรียงทิศทางแบบทวนเข็มนาฬิกา ฟังก์ชั่นนี้ทำงานถาวร  	

ID: 3	Despawn() [Span]
ลบตนเองออกจากโลกของเกม ฟังก์ชั่นนี้เสร็จสิ้นทันทีที่ใช้งาน	

ID: 4	FlipDirection(Direction **axis**)
แปลงทิศทาง **axis** ให้เป็นแกน จากนั้นพลิกทิศทางปัจจุบันไปตามแกนที่ได้นี้ 	

ID: 5	Goto(Identifier **state**) [Span]
เปลี่ยนสถานะของตนเองไปยังสถานะ **state** ฟังก์ชั่นนี้เสร็จสิ้นทันทีที่ใช้งาน	

ID: 6	Jump(Position **point**, Decimal **maxHeight**, Int **spd**, Boolean **actionEnd**) [Span]
กระโดดโดยให้ผ่านจุด **point** และมีจุดสูงสุดของการกระโดดอยู่ที่ **maxHeight** พิกเซลวัดจากจุดเริ่มกระโดด โดยค่า **maxHeight** ต้องเป็นจำนวนบวก ฟังก์ชั่นนี้จะสิ้นสุดเมื่อ **actionEnd** คืนค่า true	

ID: 7	Notify(Dynamic **agentRef**, Int **message**)
ส่งข้อความ **message** ไปให้ภาคี **agentRef**	

ID: 8	RunHarmonic(Direction **startDirection**, Decimal **spd**, bool **shouldFlip**) [Span]
วิ่งไปในทิศ **startDirection** ด้วยความเร็ว **spd** พิกเซลต่อเฟรม เมื่อ **shouldFlip** คืนค่า true ให้กลับทิศทาง 180 องศา ฟังก์ชั่นนี้ทำงานถาวร	

ID: 9	RunStraight(Direction **dir**, Decimal **spd**, bool **actionEnd**) [Span]
วิ่งไปในทิศ **dir** ด้วยความเร็ว **spd** พิกเซลต่อเฟรม ฟังก์ชั่นนี้จะสิ้นสุดเมื่อ **actionEnd** คืนค่า true	

ID: 10	RunTo(Position **pos**, Decimal **spd**) [Span]
วิ่งไปยังตำแหน่ง **pos** ด้วยความเร็ว **spd** พิกเซลต่อเฟรม ฟังก์ชั่นนี้จะสิ้นสุดเมื่อวิ่งถึงตำแหน่งที่กำหนด	

ID: 11	Set#atk(Dynamic **agentRef,** Int **value**)
กำหนดค่าคุณสมบัติพลังโจมตีของภาคี **agentRef** เป็น **value**	

ID: 12	Set#group(Dynamic **agentRef,** Int **value**)
กำหนดค่าคุณสมบัติกลุ่มของภาคี **agentRef** เป็น **value** โดยแต่ละค่าที่ใช้ได้มีความหมายดังนี้ 0: กลุ่มเดียวกับตัวละครอวตาร 1: กลุ่มเดียวกับศัตรู 2: ไม่มีกลุ่ม ถือว่าภาคีอื่น ๆ เป็นคนละกลุ่มทั้งหมด	

ID: 13	Set#direction(Dynamic **agentRef,** Direction **value**)
กำหนดค่าคุณสมบัติทิศทางของภาคี **agentRef** เป็น **value**	

ID: 14	Set#position(Dynamic **agentRef,** Position **value**)
กำหนดค่าคุณสมบัติตำแหน่งของภาคี **agentRef** เป็น **value**	

ID: 15	Set#gravityEff(Dynamic **agentRef,** Decimal **value**)
กำหนดค่าคุณสมบัติผลของแรงดึงดูดของภาคี **agentRef** เป็น **value**	

ID: 16	Set#collider(Dynamic **agentRef,** Collider **value**)
กำหนดขนาดของตัวตรวจจับการชนของภาคี **agentRef** เป็น **value**	

ID: 17	Set#attacker(Dynamic **agentRef,** Boolean **value**)
กำหนดค่าคุณสมบัติฝ่ายโจมตีของภาคี **agentRef** เป็น **value**	

ID: 18	Set#defender(Dynamic **agentRef,** Boolean **value**)
กำหนดค่าคุณสมบัติฝ่ายตั้งรับของภาคี **agentRef** เป็น **value**	

ID: 19	Set#invul(Dynamic **agentRef,** Boolean **value**)
กำหนดค่าคุณสมบัติคงกระพันของภาคี **agentRef** เป็น **value**	

ID: 20	Set#projectile(Dynamic **agentRef,** Boolean **value**)
กำหนดค่าคุณสมบัติภาคีกระสุนของภาคี **agentRef** เป็น **value** เพื่อระบุว่าภาคีดังกล่าวถือว่าเป็นภาคีกระสุนหรือไม่	

ID: 21	Set#phasing(Dynamic **agentRef,** Boolean **value**)
กำหนดค่าคุณสมบัติผ่านได้ของภาคี **agentRef** เป็น **value**	

ID: 22	Set#texture(Dynamic **agentRef,** Boolean **value**)
กำหนดภาพของภาคี **agentRef** เป็น **value**	

ID: 23	Set#hp(Dynamic **agentRef,** Int **value**)
กำหนดค่าคุณสมบัติพลังชีวิตของภาคี **agentRef** เป็น **value**	

ID: 24	Spawn(Identifier **agent**, Position **pos**, Direction **dir**)
เพิ่มภาคี **agent** ลงในโลกของเกมที่ตำแหน่ง **pos** และมีทิศทางเริ่มต้นเป็น **dir**	

ID: 25	Var(Int **name**)
ประกาศใช้งานตัวแปร **name** โดยกำหนดค่าเริ่มต้นให้ตัวแปรเป็น 0	

ID: 26	VarDec(Int **name**)
ลดค่าของตัวแปร **name** ลง 1 โดยตัวแปร **name** จะต้องถูกประกาศใช้งานไว้แล้ว	

ID: 27	VarInc(Int **name**)
เพิ่มค่าของตัวแปร **name** ขึ้น 1 โดยตัวแปร **name** จะต้องถูกประกาศใช้งานไว้แล้ว	

ID: 28	VarSet(Int **name**, Int **value**)
กำหนดค่าของตัวแปร **name** เป็น **value** โดยตัวแปร **name** จะต้องถูกประกาศใช้งานไว้แล้ว	

ID: 29	Wait(Boolean **actionEnd**)
หยุดรอโดยไม่กระทำการใด ๆ ฟังก์ชั่นนี้จะสิ้นสุดเมื่อ **actionEnd** คืนค่า true	

รายการฟังก์ชั่นข้อมูล

หัวข้อนี้แสดงรายการฟังก์ชั่นข้อมูลที่ใช้ในงานวิทยานิพนธ์นี้ รวมถึงรายละเอียดของแต่ละฟังก์ชั่นโดย ID คือ หมายเลขแทนฟังก์ชั่น ฟังก์ชั่นข้อมูลมีฟังก์ชั่นที่มีตัวเลือกเช่นเดียวกับการกระทำ และทำงานภายใต้เกณฑ์เดียวกับฟังก์ชั่นการกระทำที่มีตัวเลือก

ID: 1	Decimal Abs(Decimal **a**)
คืนค่า $\|a\|$	

ID: 2	Position Anchor(Position **pos**)
คืนตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบันโดยนำเอาทิศการหันหน้า (ซ้ายหรือขวา) เข้ามาคิดร่วม หากหันซ้าย คืนตำแหน่ง (currentX - pos.x, currentY + pos.y) หากหันขวา คืนตำแหน่ง (currentX + pos.x, currentY + pos.y)	

ID: 3	Position AnchorPlayer(Position **pos**)
คืนตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบันของตัวละครอวตารโดยนำเอาทิศการหันหน้า (ซ้ายหรือขวา) เข้ามาคิดร่วม หากหันซ้าย คืนตำแหน่ง (avatarX - pos.x, avatarY + pos.y) หากหันขวา คืนตำแหน่ง (avatarX + pos.x, avatarY + pos.y)	

ID: 4	Boolean Attack()
คืนค่า true ถ้าภาคีนี้ได้โจมตีโดนเป้าหมายในเฟรมปัจจุบัน นอกนั้นคืนค่า false	

ID: 5	Boolean Attacked()
คืนค่า true ถ้าภาคีนี้ได้โดนโจมตีในเฟรมปัจจุบัน นอกนั้นคืนค่า false	

ID: 6	Boolean ButtonPress#up()
คืนค่า true ถ้าผู้เล่นกดปุ่มขึ้นในเฟรมปัจจุบัน	

ID: 7	Boolean ButtonPress#down()
คืนค่า true ถ้าผู้เล่นกดปุ่มลงในเฟรมปัจจุบัน	

ID: 8	Boolean ButtonPress#left()
คืนค่า true ถ้าผู้เล่นกดปุ่มซ้ายในเฟรมปัจจุบัน	

ID: 9	Boolean ButtonPress#right()
คืนค่า true ถ้าผู้เล่นกดปุ่มขวาในเฟรมปัจจุบัน	

ID: 10	Boolean ButtonPress#attack()
คืนค่า true ถ้าผู้เล่นกดปุ่มโจมตีในเฟรมปัจจุบัน	

ID: 11	Boolean ButtonPress#jump()
คืนค่า true ถ้าผู้เล่นกดปุ่มกระโดดในเฟรมปัจจุบัน	

ID: 12	Boolean CollideWithDynamic()
คืนค่า true ถ้าภาคีนี้ชนกับภาคีอื่นในเฟรมปัจจุบัน	

ID: 13	Boolean Damage()
คืนค่า true ถ้าภาคีนี้สร้างความเสียหายให้ภาคีอื่นในเฟรมปัจจุบัน	

ID: 14	Boolean Damaged()
คืนค่า true ถ้าภาคีนี้ได้รับความเสียหายในเฟรมปัจจุบัน	

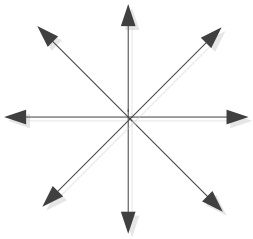
ID: 15	Set<Decimal> DecimalSet(Decimal **a**, Decimal **b**, Decimal **c**)
คืนรายการที่ประกอบด้วย {**a**, **a**+**c**, **a**+2**c**, ...} โดยค่าสูงสุดต้องไม่เกิน **b**	

ID: 16	Set<Decimal> DecimalSetSymmertry(Decimal **a**, Decimal **b**, Decimal **c**)
คืนรายการที่ประกอบด้วย {..., **a**-2**c**, **a**-**c**, **a**, **a**+**c**, **a**+2**c**, ...} โดยสมาชิกตัวที่มากสุดและน้อยสุดจะต้องมีค่าต่างจาก **a** ไม่เกิน **b**	

ID: 17	Direction DirectionComponent#X(Direction **dir**)
คืนค่าทิศทางแกน x ของ **dir** แสดงได้ด้วยเวกเตอร์ [**dir**.x, 0]	

ID: 18	Direction DirectionComponent#Y(Direction **dir**)
คืนค่าทิศทางแกน y ของ **dir** แสดงได้ด้วยเวกเตอร์ [0, **dir**.y]	

ID: 19	Set<Direction> DirectionSet(Direction **dir**)
คืนค่ารายการของทิศทางที่ประกอบด้วย **dir** และ **dir** ที่หมุนไป 180 องศา	

ID: 20	Set<Direction> DirectionSetDivide(Int **slice**)
คืนรายการของทิศทางที่ได้จากการแบ่งมุม 360 องศาออกเป็น **slice** ส่วน ภาพด้านล่างแสดงตัวอย่างกรณี **slice** = 8 	

ID: 21	Set<Direction> DirectionSetRange(Direction **start**, Direction **end**, Decimal **step**)
คืนรายการของทิศทางที่ประกอบด้วย {**start**, rot(**step**), rot(**step***2), ...} เมื่อ rot(X) แทนการหมุน **start** ไป X องศา โดยทิศทางจะต้องไม่เกิน **end**	

ID: 22	Decimal DistanceTo#X(Position **pos**)
คืนระยะห่างแกน x จากภาคีนี้ไปยัง **pos** คำนวณได้โดย currentX – **pos**.x	

ID: 23	Decimal DistanceTo#Y(Position **pos**)
คืนระยะห่างแกน y จากภาคีนี้ไปยัง **pos** คำนวณได้โดย currentY – **pos**.y	

ID: 24	Decimal DistanceToPlayer#X()
คืนระยะห่างแกน x จากภาคีนี้ไปยังตัวละรอวตารคำนวณได้โดย currentX – avatarX	

ID: 25	Decimal DistanceToPlayer#Y()
คืนระยะห่างแกน y จากภาคีนี้ไปยังตัวละรอวตารคำนวณได้โดย currentY – avatarY	

ID: 26	Int DynamicCount ()
คืนจำนวนภาคีทั้งหมดที่อยู่ในโลกของเกม ณ เฟรมปัจจุบัน	

ID: 27	Dynamic DynamicFilter#this()
คืนตัวอ้างอิงมายังภาคีนี้	

ID: 28	Dynamic DynamicFilter#player()
คืนตัวอ้างอิงไปยังภาคีที่เป็นตัวละครอวตาร	

ID: 29	Int Get#atk(Dynamic **agentRef**)
คืนค่าคุณสมบัติพลังโจมตีของภาคี **agentRef**	

ID: 30	Int Get#group(Dynamic **agentRef**)
คืนค่าคุณสมบัติกลุ่มของภาคี **agentRef** ความหมายของค่าที่ได้เหมือนกับฟังก์ชั่นการกระทำ Set#group	

ID: 31	Direction Get#direction(Dynamic **agentRef**)
คืนค่าคุณสมบัติทิศทางของภาคี **agentRef**	

ID: 32	Position Get#position(Dynamic **agentRef**)
คืนค่าคุณสมบัติตำแหน่งของภาคี **agentRef**	

ID: 33	Decimal Get#gravityEff(Dynamic **agentRef**)
คืนค่าคุณสมบัติผลของแรงดึงดูดของภาคี **agentRef**	

ID: 34	Collider Get#collider(Dynamic **agentRef**)
คืนขนาดตัวตรวจจับการชนของภาคี **agentRef**	

ID: 35	Boolean Get#attacker(Dynamic **agentRef**)
คืนค่า true เมื่อภาคี **agentRef** มีค่าคุณสมบัติฝ่ายโจมตี คืนค่า false ในกรณีอื่น	

ID: 36	Boolean Get#defender(Dynamic **agentRef**)
คืนค่า true เมื่อภาคี **agentRef** มีค่าคุณสมบัติฝ่ายตั้งรับ คืนค่า false ในกรณีอื่น	

ID: 37	Boolean Get#invul(Dynamic **agentRef**)
คืนค่า true เมื่อภาคี **agentRef** มีค่าคุณสมบัติคงกระพัน คืนค่า false ในกรณีอื่น	

ID: 38	Boolean Get#projectile(Dynamic **agentRef**)
คืนค่า true เมื่อภาคี **agentRef** เป็นภาคีกระสุน คืนค่า false ในกรณีอื่น	

ID: 39	Boolean Get#phasing(Dynamic **agentRef**)
คืนค่า true เมื่อภาคี **agentRef** มีค่าคุณสมบัติผ่านได้ คืนค่า false ในกรณีอื่น	

ID: 40	Int Get#hp(Dynamic **agentRef**)
คืนค่าคุณสมบัติพลังชีวิตของภาคี **agentRef**	

ID: 41	Boolean InTheAir()
คืนค่า true เมื่อภาคีนี้อยู่กลางอากาศ โดยใช้วิธีพิจารณาว่าภาคียืนอยู่บนพื้นหรือไม่ (หรือเพดาน ในกรณีที่ค่าคุณสมบัติผลของแรงดึงดูดมีค่ามากกว่า 0) คืนค่า false ในกรณีอื่น	

ID: 42	Boolean Notified(Int **message**)
คืนค่า true เมื่อภาคีนี้ได้รับข้อความ **message** จากฟังก์ชั่นการกระทำ Notify ในเฟรมปัจจุบัน	

ID: 43	Boolean Peak()
คืนค่า true เมื่อภาคีนี้กระโดดถึงจุดสูงสุดของการกระโดดที่เกิดจากฟังก์ชั่นการกระทำ Jump	

ID: 44	Direction Perpendicular(Direction **dir**)
คืนทิศทางที่ตั้งฉากกับ **dir** ซึ่งคำนวณได้จากการหมุน **dir** ในทิศทวนเข็มนาฬิกา 90 องศา	

ID: 45	Any Random(Set<Any> **set**)
สุ่มเลือกค่าจากรายการ **set** และคืนค่าดังกล่าว **set** จะต้องมีอย่างน้อย 1 ค่าเพื่อใช้ฟังก์ชั่นข้อมูลนี้	

ID: 46	Position RandomPositionInRadius(Position **pos**, Decimal **radius**)
สุ่มตำแหน่งที่อยู่ในระยะ **radius** รอบจุด **pos** และคืนจุดนั้น	

ID: 47	Position RandomPositionInRange(Position **posA**, Position **posB**)
สร้างขอบเขตเล็กสุด (Minimum bounding box) สำหรับจุด **posA** และ **posB** จากนั้นสุ่มเลือก 1 จุดที่อยู่ภายในขอบเขตและคืนจุดนั้น	

ID: 48	Decimal RangeCap(Decimal **a**, Decimal **b**, Decimal **c**)
จำกัด (clamp) ค่า **a** ให้อยู่ในช่วง [**b**,**c**] และคืนค่าดังกล่าว คำนวณได้ด้วย $\min(\max(a,b),c)$	

ID: 49	Decimal RangeCapCircular(Decimal **a**, Decimal **b**, Decimal **c**)

จำกัดค่า **a** ให้อยู่ในช่วง [**b**,**c**] โดยใช้วิธีวนค่าและคืนค่าดังกล่าว คำนวณได้โดยขั้นตอนวิธีต่อไปนี้
Let $range = c - b + 1$
While($a < b$) do: $a = a + range$
While($a > c$) do: $a = a - range$	

ID: 50	Position Rel(Position **pos**)
คืนตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบันของภาคีนี้ โดยคำนวณจาก (currentX + pos.x, currentY + pos.y)	

ID: 51	Position RelPlayer(Position **pos**)
คืนตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบันของตัวละครอวตาร โดยคำนวณจาก (avatarX + pos.x, avatarY + pos.y)	

ID: 52	Direction RelDirection(Direction **dir**)
คืนมุมเงยขนาด X องศาเทียบกับทิศทางปัจจุบันของภาคีนี้โดยนำเอาทิศการหันหน้ามาคิดร่วม เมื่อ X = มุมวัดจากเวกเตอร์ [1, 0] ไปยังเวกเตอร์ทิศทางที่ระบุด้วย **dir** หากหันซ้าย ให้คืนทิศที่ได้จากการหมุนทิศทางปัจจุบันของภาคีนี้ตามเข็มนาฬิกาไป X องศา หากหันขวา ให้คืนทิศที่ได้จากการหมุนทิศทางปัจจุบันของภาคีนี้ทวนเข็มนาฬิกาไป X องศา	

ID: 53	Direction Rel2Direction(Direction **dir**)
คืนทิศทางที่ได้จากการหมุนทิศทางปัจจุบันของภาคีนี้ไป X องศาในทิศทวนเข็มนาฬิกา เมื่อ X = มุมวัดจากเวกเตอร์ [1, 0] ไปยังเวกเตอร์ทิศทางที่ระบุด้วย **dir**	

ID: 54	Boolean SurfaceInDir(Direction **dir**)
คืนค่า true เมื่อมีพื้นที่ที่ผ่านไม่ได้ในทิศ **dir** ของภาคีนี้ โดยให้แยกคิดเป็นแกน x (ทิศซ้ายหรือขวา) และแกน y (ทิศบนหรือล่าง) เนื่องจากเกมทดสอบไม่มีพื้นเอียง	

ID: 55	Int TimePass()
ใช้งานคู่กับฟังก์ชั่นการกระทำระยะยาว เมื่อเรียกใช้ในลำดับพฤติกรรมที่มีฟังก์ชั่นการกระทำยังไม่เสร็จสิ้น ให้คืนค่าจำนวนเฟรมที่กระทำฟังก์ชั่นดังกล่าวตั้งแต่เริ่มฟังก์ชั่น คืนค่า 0 ในกรณีอื่น	

ID: 56	Int TravelDistance()
ใช้งานคู่กับฟังก์ชั่นการกระทำระยะยาวจำพวก Run เมื่อเรียกใช้ในลำดับพฤติกรรมที่มีฟังก์ชั่นการกระทำยังไม่เสร็จสิ้น ให้คืนค่าระยะทางหน่วยพิกเซลที่เคลื่อนที่ได้จากการใช้ฟังก์ชั่นดังกล่าวตั้งแต่เริ่มฟังก์ชั่น คืนค่า 0 ในกรณีอื่น	

ID: 57	Direction TurnTo(Set<Direction> **dset**, Position **pos**)
เลือกทิศทางจาก **dset** ที่จะทำให้ภาคีนี้หันเข้าหาตำแหน่ง **pos** มากที่สุด (มีผลต่างมุมระหว่างทิศที่เลือกและเวกเตอร์ที่สร้างจากภาคีนี้ไปยัง **pos** น้อยที่สุด) และคืนทิศทางดังกล่าว	

ID: 58	Direction TurnToPlayer(Set<Direction> **dset**)
เลือกทิศทางจาก **dset** ที่จะทำให้ภาคีนี้หันเข้าหาตัวละครอวตารมากที่สุด และคืนทิศทางดังกล่าว	

ID: 59	Int VarGet(Int **name**)
คืนค่าของตัวแปร **name** โดยตัวแปร **name** จะต้องถูกประกาศใช้งานไว้แล้ว	

ID: 60	Decimal U_Neg(Decimal **exp**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการเอกภาค “-“	

ID: 61	Boolean U_Not(Boolean **exp**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการเอกภาค “!“	

ID: 62	Boolean B_And(Boolean **left**, Boolean **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “&&“ โดยในที่นี้แทนนิพจน์ **left** && **right**	

ID: 63	Boolean B_Or(Boolean **left**, Boolean **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “||“ โดยในที่นี้แทนนิพจน์ **left** || **right**	

ID: 64	Boolean B_Equal(Any **left**, Any **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “==“ โดยในที่นี้แทนนิพจน์ **left** == **right**	

ID: 65	Boolean B_NotEqual(Any **left**, Any **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “!=“ โดยในที่นี้แทนนิพจน์ **left** != **right**	

ID: 66	Boolean B_GreaterThan(Decimal **left**, Decimal **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “>“ โดยในที่นี้แทนนิพจน์ **left** > **right**	

ID: 67	Boolean B_LessThan(Decimal **left**, Decimal **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “<“ โดยในที่นี้แทนนิพจน์ **left** < **right**	

ID: 68	Boolean B_GreaterThanOrEqual(Decimal **left**, Decimal **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “>=“ โดยในที่นี้แทนนิพจน์ **left** >= **right**	

ID: 69	Boolean B_LessThanOrEqual(Decimal **left**, Decimal **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “<=“ โดยในที่นี้แทนนิพจน์ **left** <= **right**	

ID: 70	Decimal B_Add(Decimal **left**, Decimal **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “+“ โดยในที่นี้แทนนิพจน์ **left** + **right**	

ID: 71	Decimal B_Sub(Decimal **left**, Decimal **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “-“ โดยในที่นี้แทนนิพจน์ **left** - **right**	

ID: 72	Decimal B_Mul(Decimal **left**, Decimal **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “*“ โดยในที่นี้แทนนิพจน์ **left** * **right**	

ID: 73	Decimal B_Div(Decimal **left**, Decimal **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “/“ โดยในที่นี้แทนนิพจน์ **left** / **right**	

ID: 74	Decimal B_Mod(Decimal **left**, Decimal **right**)
ฟังก์ชั่นข้อมูลแทนตัวดำเนินการทวิภาค “%“ โดยในที่นี้แทนนิพจน์ **left** % **right**	

ประวัติผู้เขียนวิทยานิพนธ์

ผู้จัดทำวิทยานิพนธ์ นายพิชิต พร้อมสุทธิพงศ์ เกิดเมื่อปี พ.ศ. 2532 สำเร็จการศึกษาระดับมัธยมศึกษาจากโรงเรียนทวีธาภิเศก และระดับปริญญาบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2555 จากนั้นในปีการศึกษา 2556 ได้เข้าศึกษาต่อในระดับปริญญามหาบัณฑิตที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย