เกมเอาชีวิตรอดหุ่นยนต์

The Robot Survival Game

ศรายุธ เตชะแก้ว

590510526

การค้นคว้าอิสระนี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรวิทยาศาสตรบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่

ปีการศึกษา 2562

เกมเอาชีวิตรอดหุ่นยนต์

The Robot Survival Game

ศรายุธ เตชะแก้ว

590510526

การค้นคว้าอิสระนี้ได้รับการพิจารณาอนุมัติให้นับเป็นส่วนหนึ่งของการศึกษา
ตามหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่
ปีการศึกษา 2562

คณะกรรมการสอบการค้นคว้าอิสระ

..... ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.จักริน ชวชาติ)
..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.เสมอแข สมหอม)

วันที่.....เดือน.....พ.ศ.....

กิตติกรรมประกาศ

การค้นคว้าอิสระนี้ จะสำเร็จลุล่วงไม่ได้เลยถ้าไม่ได้รับความอนุเคราะห์จาก ผู้ช่วยศาสตราจารย์ ดร. จักริน ชวชาติ ที่ได้ช่วยเสียสละเวลาอันมีค่าเพื่อให้คำปรึกษาและแนวทางต่าง ๆ ตั้งแต่การหาข้อมูล การตามความคืบหน้าของงาน ไปจนถึงการแก้ไขข้อบกพร่องของเนื้อหาและสำนวนภาษาด้วยความใส่ใจยิ่ง ผู้ค้นคว้าอิสระขอกราบขอบพระคุณเป็นอย่างสูง ณ โอกาสนี้

ขอกราบขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.เสมอแข สมหอม ที่กรุณารับเป็นกรรมการสอบการค้นคว้าอิสระนี้ ทั้งการพยายามทำความเข้าใจในชิ้นงานที่ยังไม่เป็นรูปร่าง และให้คำแนะนำเป็นอย่างดีมาโดยตลอด ขอขอบคุณคณาอาจารย์ที่ได้ให้การสนับสนุนการดำเนินการทำงานและมอบความรู้วิชาอันมีค่า เพื่อเป็นพื้นฐานในการทำการค้นคว้าอิสระในครั้งนี้ หรือกระทั่งบางความรู้ที่ได้นำมาใช้ในการค้นคว้านี้โดยตรง และขอบคุณทุกความช่วยเหลือในการทำการค้นคว้าอิสระนี้ให้สำเร็จลุล่วงไปด้วยดี

นาย ศรายุธ เตชะแก้ว

590510526

หัวข้อการค้นคว้าอิสระ เกมเอาชีวิตรอดหุ่นยนต์

ชื่อเจ้าของโครงงาน นาย ศรายุธ เตชะแก้ว **รหัสประจำตัว** 590510526

วิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์

อาจารย์ที่ปรึกษา ผู้ช่วยศาสตราจารย์ ดร.จักริน ชวชาติ

บทคัดย่อ

การค้นคว้าอิสระนี้ มีวัตถุประสงค์เพื่อพัฒนาเกมเอาชีวิตรอดหุ่นยนต์ซึ่งเป็นซอฟต์แวร์คอมพิวเตอร์บนระบบปฏิบัติการวินโดวส์ในรูปแบบเกมสองมิติจากมุมบน โดยมีเป้าหมายของเกมคือเอาชีวิตรอดจากศัตรูที่จะเข้ามาโจมตีผู้เล่นเรื่อยๆ ให้ได้นานที่สุด โดยเกมนี้มีจุดเด่นคือการสร้างและควบคุมหุ่นยนต์ของตนเองจำนวนมาก ตั้งแต่ขั้นตอนการสร้างที่ผู้เล่นสามารถเลือกชิ้นส่วนเองได้ ทั้งการเพิ่มความเร็วการเคลื่อนที่ เกราะ พลังงานแบตเตอรี่ หรือแม้แต่ป้อมปืนช่วยยิงอัตโนมัติ จนถึงการโปรแกรมคำสั่งให้หุ่นยนต์จำนวนมากที่สร้างขึ้นมาเหล่านั้นเองด้วย ซึ่งถือเป็นข้อดีหนึ่งแยกจากเกมอื่นๆ ที่ส่วนใหญ่ผู้เล่นต้องเป็นฝ่ายคอยควบคุมวัตถุจำนวนมากเหล่านั้นเองทั้งหมด ด้วยรูปแบบการโปรแกรมคำสั่งที่เข้าใจง่าย แต่ก็ใช้ได้จริง ด้วยการเขียนแบบขึ้นกับเหตุการณ์และการรับมือกับเหตุการณ์นั้น โดยผู้พัฒนามีความมุ่งหวังว่า เกมนี้จะมอบความสนุกและช่วยให้ผู้เล่นได้ฝึกความรู้ความเข้าใจการวางแผนในแนวการเขียนโปรแกรมไม่มากก็น้อย

Independent Study Title The Robot Survival Game

Author Mr. Sarayut Techakaew **Student ID** 590510526

Bachelor of Science Computer Science

Supervisor Asst. Prof. Jakarin Chawachat

Abstract

This Independent Study was created on purpose to develop the game name 'Robot Survival', which is computer software on window os with the 2D top-down perspective. The objective of this game is to survive form enemy attacks as long as possible. This game is highly focused on creating and command your robot horde from customizing robot-part step, customize each hardware part to adjust the movement speed, armor, battery or even the auto-fire turrets, to programming robot-script step, program your robot to do whatever you want autonomically unlike any other game that the player must be grinding the same thing over and over, program it with easy to understand but powerful commands with priority and condition system. More or less, with my game, I wish you the player will have fun and understand the concept of programming.

สารบัญ

หน้า

สารบัญ(ต่อ)

สารบัญรูป

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญตาราง

หน้า

บทที่ 1

บทนำ

1.1 หลักการและเหตุผล

ปัจจุบันในการศึกษาการเขียนโปรแกรมนั้น สามารถเข้าถึงและเรียนรู้ได้ง่ายกว่าในอดีตเป็นอย่างมากเนื่องจาก มีสื่อการสอนมากมาย เช่น แอนิเมชัน (Animation) หรืออินเตอร์แอคทีพเว็บไซต์ (Interactive website) ทำให้เข้าใจหลักการทำงาน การเขียนโปรแกรมแบบเป็นขั้นเป็นตอนได้ง่ายขึ้น เกมเป็นสื่อรูปแบบหนึ่งที่ถูกใช้ ในการนำเสนอ ตัวอย่างเช่นเกมปริศนา (Puzzle) เป็นเกมที่มีลักษณะเป็นปริศนาให้ผู้เล่นทำตามเงื่อนไขเพื่อ ผ่านไปในแต่ละด่านตามโจทย์ที่กำหนดไว้ เช่นเกม Light Bot[1], Human Resource Machine[2] อีกตัวอย่างได้แก่ เกมวางแผนออกแบบปัญญาประดิษฐ์ให้หุ่นยนต์ของตัวเองสู้กับหุ่นยนต์ของฝั่งตรงข้ามอย่าง เกม Gladiabot[3]

อย่างไรก็ตามเกมส่วนใหญ่จะมุ่งเน้นไปที่การแก้ปัญหาในสถานการณ์ต่างๆ ผู้พัฒนาจึงคิดว่าหากสามารถสร้างเกมที่เน้นไปในรูปแบบที่ให้อิสระกับผู้เล่นในการสร้างสรรค์ผลงานคงน่าสนใจไม่น้อย เป้าหมายของเกมคือให้ผู้เล่นอยู่รอดในเกมให้นานที่สุด สมมติให้ผู้เล่นเป็นหุ่นยนต์ที่มีช่องเก็บของ ซึ่งสามารถปรับแต่งส่วนประกอบต่างๆ ได้ โดยการเลือกชิ้นส่วนต่างๆ เข้ามาประกอบในช่องเก็บของของหุ่นยนต์ เช่น ป้อมปืน, เกราะ, แบตเตอรี่, แหล่งผลิตพลังงาน และ อุปกรณ์ขับเคลื่อน เป็นต้น โดยชิ้นส่วนแต่ละประเภทจะมีข้อมูลจำเพาะที่ทำให้รูปแบบการทำงานที่แตกต่างกันไป ทั้งราคา ประสิทธิภาพ และการกินพลังงาน ทำให้ผู้เล่นสามารถสร้างสรรค์ ปรับแต่งคุณสมบัติของหุ่นยนต์ได้หลากหลาย นอกจากนี้ผู้เล่นยังต้องเขียนโปรแกรมเพื่อควบคุมการทำงานทั้งหมดของหุ่นยนต์ด้วยตัวเองอีกด้วย ทั้งนี้การโปรแกรมการทำงานจะเป็นในรูปแบบอย่างง่าย คือ ประกอบด้วยคำสั่ง และเหตุการณ์ที่จะไปเรียกให้คำสั่งนั้นทำงาน เช่น คำสั่งโจมตี ถ้ามีศัตรูอยู่ในระยะ เป็นต้น คำสั่งสามารถมีได้หลายคำสั่งพร้อมกันแต่การเลือกทำงานนั้น จะถูกจัดเรียงตามลำดับความสำคัญ เพื่อให้สามารถโปรแกรมควบคุมการทำงานของหุ่นยนต์ที่มีความซับซ้อนขึ้นมาได้

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อพัฒนาเกมบนคอมพิวเตอร์ ในระบบปฏิบัติการวินโดวส์
2. เพื่อพัฒนาเกมที่ฝึกทักษะการคิดแบบเป็นแบบแผน โดยมุ่งเน้นไปที่การให้อิสระแก่ผู้เล่นในการสร้างสรรค์ผลงาน

1.3 ประโยชน์ที่จะได้รับจากการศึกษาเชิงประยุกต์และ/หรือ เชิงทฤษฎี

1. ได้เกมบนคอมพิวเตอร์ ในระบบปฏิบัติการวินโดวส์
2. ได้สื่อที่ช่วยในการเสริมสร้างทักษะการเขียนโปรแกรม

1.4 ขอบเขตของโครงงาน/วิธีการวิจัย

(1) ขอบเขตทางสถาปัตยกรรม Windows Application (Stand Alone)

ฮาร์ดแวร์ที่ใช้ในการพัฒนา

- พัฒนาบน Lenovo ThinkPad E585

> CPU : AMD Ryzen 7 2700U with Radeon Vega Mobile Gfx 2.20 GHz

> MEMORY : 16.0 GB

- Spec ขั้นต่ำที่ใช้ในการเล่นเกม

> CPU : 1.7 GHz

> MEMORY : 2.0 GB

> Hard Drive : 40 MB

ซอฟต์แวร์ที่ใช้ในการพัฒนา

- Windows 10 Education 64-bit

- Godot Engine (version 3.2)

(2) ขอบเขตของระบบงาน

‘Robot Survival’ เป็นเกมบนระบบปฏิบัติการวินโดวส์แบบผู้เล่นคนเดียว ในรูปแบบ 2 มิติจากมุมบน โดยใช้เม้าส์ในการควบคุม เมื่อเริ่มเกม เกมจะถูกดำเนินในโลกเปิดที่สร้างขึ้นมาแบบสุ่ม โดยมีเป้าหมายคือเอา ชีวิตรอดจากหุ่นยนต์ฝ่ายศัตรูที่จะเกิดขึ้นมาเรื่อยๆ ให้ได้นานที่สุด เกมจะสิ้นสุดเมื่อหุ่นยนต์หลักของผู้เล่น พลังชีวิตหมด(ตาย) และมีการแสดงคะแนน ผู้เล่นสามารถถอดหรือใส่ชิ้นส่วนต่างๆให้กับหุ่นยนต์ เขียนคำสั่ง ให้กับหุ่นยนต์ และใช้เม้าส์เพื่อควบคุมการเคลื่อนที่ของหุ่นยนต์ ในหุ่นยนต์ตัวหนึ่งๆ จะประกอบไปด้วย กล่องชิ้นส่วน(Case) ที่เป็นแกนหลักของหุ่น กล่องนี้มีขนาดเป็นจำนวนช่องที่สามารถบรรจุชิ้นส่วนอื่นได้ และค่าสถานะพื้นฐานต่างๆ เช่นพลังชีวิตและพลังงานแบตเตอรี่ โดยจะมีชิ้นส่วนอื่นๆที่สามารถนำมา ประกอบเพิ่มเติมได้คือ ป้อมปืนที่ใช้ในการโจมตี, เกราะ ที่เพิ่มพลังป้องกัน, แบตเตอรี่ เพื่อเพิ่มความจุพลังงานสำรองของหุ่นยนต์, แหล่งผลิตพลังงาน ที่ใช้สร้างพลังงานขึ้นมาใช้เอง หรือกระทั่งแบ่งปันแก่หุ่นตัวอื่น และ อุปกรณ์ขับเคลื่อนที่ใช้เคลื่อนที่หุ่น เป็นหลัก และอาจมีชิ้นส่วนเสริมอื่นๆ อีกตามความเหมาะสม ซึ่งชิ้นส่วน เหล่านี้เราจะหาได้จากการกำจัดศัตรูที่มีชิ้นส่วนชิ้นเดียวกัน โดยการทำงานของหุ่นยนต์ฝั่งศัตรูจะทำงาน ภายใต้การประกอบชิ้นส่วนและเขียนคำสั่งที่ออกแบบไว้ก่อนแล้วเช่นเดียวกับผู้เล่น ทำให้อาจกล่าวได้ว่าหุ่นยนต์ฝั่งศัตรูทุกตัวที่ผู้เล่นเห็นนั้น ผู้เล่นก็สามารถสร้างขึ้นมาเองได้เช่นกัน

1.5 แผนการดำเนินงานและระยะเวลาดำเนินงาน

การศึกษานี้เริ่มดำเนินงานตั้งแต่เดือนสิงหาคม พ.ศ. 2562 ถึง เดือนมีนาคม พ.ศ. 2563 แสดงรายละเอียดการดำเนินงาน ดังตารางที่ 1.1 ซึ่งมีขั้นตอนการดำเนินงานดังนี้

1. ศึกษาการใช้งานโปรแกรมที่เกี่ยวข้อง
2. ออกแบบระบบการทำงาน
3. ออกแบบการแสดงผล และรูปภาพประกอบ
4. พัฒนาระบบตามที่ได้ออกแบบไว้
5. ทดสอบการใช้งานระบบ
6. จัดทำเอกสาร

ตารางที่ 1.1 รายละเอียดขั้นตอนการดำเนินงาน

ระยะเวลา **ขั้นตอนการดำเนินงาน**	**ปี พ.ศ. 2562**					**ปี พ.ศ. 2563**		
	ส.ค.	**ก.ย.**	**ต.ค.**	**พ.ย.**	**ธ.ค.**	**ม.ค.**	**ก.พ.**	**มี.ค.**
1. ศึกษาการใช้งานโปรแกรมที่เกี่ยวข้อง								
2. ออกแบบระบบการทำงาน								
3. ออกแบบการแสดงผล และรูปภาพประกอบ								
4. พัฒนาระบบตามที่ได้ออกแบบไว้								
5. ทดสอบการใช้งานระบบ								
6. จัดทำเอกสาร								

บทที่ 2

หลักการและทฤษฎีที่เกี่ยวข้อง

2.1 องค์ประกอบของเกม

ในเกมๆนี้จะประกอบไปด้วยองค์ประกอบต่างๆคือ หน้าต่างแสดงผลรูปภาพ หน้าต่างเมนู และโค้ดคำสั่ง โดยต้องคำนึงถึงปัจจัยต่างๆเหล่านี้ เพื่อหาตัวช่วยที่เหมาะสมในการทำงาน

2.1.1 หน้าต่างแสดงผลรูปภาพ

ปกติแล้วเกมสองมิติจะทำงานคล้ายๆหลักการของการทำ Animation คือการแสดงผลรูปภาพหลายๆรูปต่อเนื่องกันอย่างรวดเร็ว และเมื่อคนดูดูรูปภาพเหล่านั้นก็จะรู้สึกเหมือนมีการเคลื่อนไหวเกิดขึ้นมา การทำเกมก็เช่นกัน เมื่อเรานำรูปของตัวละครไปวางไว้บนตำแหน่งหนึ่งของหน้าจอ และค่อยๆ ย้ายตำแหน่งของตัวละครนั้นในรูปที่จะวาดถัดๆ ไปอย่างรวดเร็ว ก็จะทำให้รู้สึกเหมือนตัวละครนั้นกำลังเคลื่อนที่อยู่ และเมื่อ เราทำสิ่งเหล่านี้กับวัตถุหลายๆ วัตถุพร้อมกันในภาพทั้ง ผู้เล่น ศัตรู ไอเทม และแผนที่ ประกอบกันก็จะกลายเป็นเกมขึ้นมาได้ หากแต่เกมนั้นไม่ใช่แค่ต้องการแค่การเคลื่อนที่อย่างเดียว แต่ต้องการการปฏิสัมพันธ์กับผู้เล่นด้วย เช่น การทำให้รูปของตัวละครผู้เล่นนั้นเคลื่อนที่ไปตามทิศทางที่ผู้เล่นกดปุ่ม ประกอบกับเหล่าศัตรูที่เคลื่อนที่ตอบสนองกับตัวละครผู้เล่น ซึ่งเกมก็คือการมองรูปภาพเหล่านี้เป็นวัตถุประกอบกับโค้ดเบื้องหลังการทำงานของวัตถุเหล่านี้

2.1.2 หน้าต่างเมนู

หน้าต่างเมนูนั้นคือส่วนที่ใช้ในการแสดงผลข้อมูลผ่าน User Interface (UI) ไปยังผู้เล่น รวมถึงการมีความสามารถที่จะปฏิสัมพันธ์กับผู้เล่นได้ เพื่อที่จะทำให้ผู้เล่นสามารถมีปฏิสัมพันธ์อื่นๆ กับเกมหรือข้อมูลในเกมได้นอกเหนือจากหน้าต่างแสดงผลรูปภาพเกมโดยปกติ ซึ่งในบางโปรแกรมก็มีการเขียนภาษาที่มาช่วยในส่วนแสดงผลนี้เฉพาะ เช่นภาษา HTML ที่ใช้ในการแสดงเว็บไซต์ การมีเฟรมเวิร์ค(Framework) ที่สามารถมาช่วยในส่วนนี้ก็จะทำให้งานง่ายขึ้น ไม่จำเป็นต้องไปเขียนคำสั่งเพื่อวาดข้อมูลลงในหน้าต่างเกมเองทั้งหมด ซึ่งเป็นสิ่งที่ถึงจะทำได้ แต่ก็ค่อนข้างลำบากในการที่จะต้องทำใหม่ขึ้นมาเองทั้งหมด

2.1.3 โค้ดคำสั่ง

เพื่อที่จะสร้างเกมที่สามารถเล่นได้แบบ Real-time ภาษาที่ใช้จะต้องรองรับการทำงานอย่างอิสระกันระหว่างการประมวลผลการทำงานและการรอรับอินพุตต่างๆ จากผู้เล่น (แบ่งเทรดการทำงานได้ multi thread[5]) หรือที่เรียกว่า event trigger[6] ที่จะรอรับการทำงานจากผู้เล่นอยู่ตลอด และจะแทรกการทำงานเข้ามาเมื่อไรก็ได้เมื่อมีเหตุการณ์ตรงตามเงื่อนไขการทริกเกอร์นั้น เพราะถ้าไม่เช่นนั้นจะกลายเป็นเกมที่ รอรับอินพุตจากผู้เล่นก่อน แล้วถึงสามารถประมวลผลคำสั่งอื่นๆ ได้ ทำแบบนี้เรื่อยๆ สลับกัน หรือที่เรียกกันว่า เกม Turn-based

เกมที่มีการทำงานซับซ้อนส่วนใหญ่จะมีการใช้การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming (OOP)) เนื่องจากเป็นการทำงานบนวัตถุจำนวนมาก ที่มีจำนวนไม่ตายตัว สามารถเพิ่มจำนวนวัตถุได้เรื่อยๆ รวมถึงการสืบทอดหรือปรับแต่งคุณสมบัติแก่วัตถุ เช่น วัตถุผู้เล่น วัตถุศัตรู วัตถุไอเทมที่มีอิสระในการเพิ่มลดจำนวนอยู่ตลอดระหว่างเกม หรือแม้กระทั่งหน้าต่างเมนูต่างๆ ทั้งหมดล้วนเหมาะสมแก่การเขียนโปรแกรมแบบ OOP ทั้งสิ้น

2.2 ซอฟต์แวร์ที่ใช้ในการพัฒนา Godot Engine[7]

เป็น game engine ที่ผู้จัดทำเลือกมาใช้ในการทำโปรเจคนี้เนื่องจากเป็นโปรแกรมฟรี มีขนาดเบา (ขนาดไฟล์เล็ก กินทรัพยากรเครื่องน้อย) และมีฟีเจอร์ที่เพียงพอต่อการสร้างเกมของผู้พัฒนา ดังนี้

2.2.1 ภาษา GDScript [8] และ C#

```
var d = {4: 5, "A key": "A value", 28: [1, 2, 3]}
d["Hi!"] = 0
d = {
    22: "value",
    "some_key": 2,
    "other_key": [2, 3, 4],
    "more_key": "Hello"
}
```

รูปที่ 2.1 ตัวอย่างการใช้ตัวแปรประเภทDictionary ในภาษา GDScript

Godot Engine มีภาษา GDScript เป็นของตัวเอง ซึ่งมีความคล้ายคลึงกับภาษาที่ใช้กันทั่วไป เช่น Javascript หรือ Python ซึ่งง่ายต่อการเรียนรู้ เขียนง่ายและมีความยืดหยุ่น อย่างเช่นการประกาศ

ตัวแปรที่เป็น dynamic ไม่ต้องระบุชนิดของตัวแปรแต่โปรแกรมรู้ได้เองจากค่าที่กำหนดเข้าไป ตัวแปรชุดข้อมูลที่สามารถปรับขนาด เองโดยอัตโนมัติได้ มีค่า index ที่สามารถติดลบได้ (เป็นการนับย้อนกลับจากลำดับหลังสุดเหมือนในภาษา Python) และยังสามารถเก็บตัวแปรหลายชนิดร่วมกันในชุดข้อมูลได้ ดังรูปตัวอย่างที่2.1 ทั้งยังรองรับ การเขียนในภาษาที่นิยมใช้ในการทำเกมอย่าง C# ถ้าต้องการ เช่นในกรณีที่มีการนำ library ภายนอกเข้ามาใช้ มี InputEvent[9]

GDScript C#

```
func _unhandled_input(event):
    if event is InputEventKey:
        if event.pressed and event.scancode == KEY_ESCAPE:
            get_tree().quit()
```

รูปที่ 2.2 ตัวอย่างการสร้างทริกเกอร์ให้ปุ่ม Esc เพื่อปิดเกม

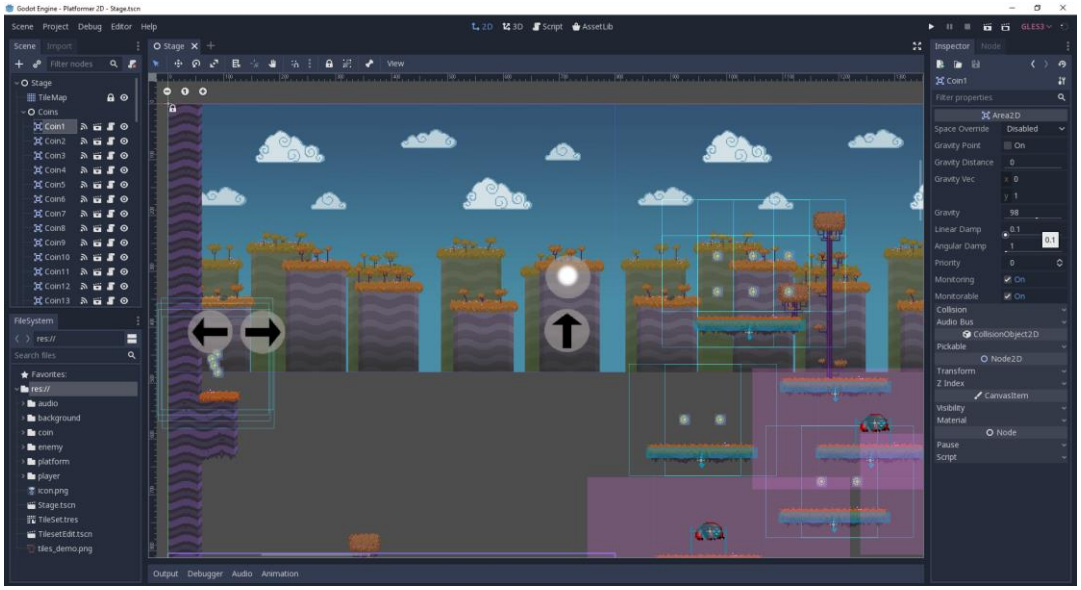
หรือที่ในบางภาษาเรียกว่า Event Trigger ทำให้เราสามารถตรวจจับการทำงานของ input ต่างๆ ไปพร้อมๆ กับการแสดงผลและทำงานประมวลผลอื่นๆ แบบ real-time ไปด้วยได้ ทั้งเหตุการณ์คลิกในเม้าส์ การสัมผัสในจอสัมผัส หรือการกดจากคีย์บอร์ดของผู้เล่น เมื่อมีเงื่อนไขที่ตรงกัน ก็จะแทรกการทำงานตามคำสั่งที่ระบุเข้ามาได้

2.2.2 รองรับโครงสร้างการทำงานแบบ OOP

ไม่ใช่เพียงแค่ตัวโปรแกรมเองที่สามารถสร้าง Class และ Object แบบ OOP ได้ แต่โครงสร้างโดยรวมของสิ่งต่างๆ ในโปรแกรมเองก็เป็นOOPเช่นกัน เราสามารถสร้างโหนดหุ่นยนต์ขึ้นมา เพิ่มโหนดลูกเป็นป้ายชื่อ หลอดเลือดและพื้นที่ที่จะตรวจสอบการสัมผัสกันของวัตถุผ่าน UI ของโปรแกรม Godot หลังจากนั้นก็กล่าวถึงได้ผ่านโค้ดคำสั่ง เพื่อสร้างวัตถุชั่วคราวของโหนดหุ่นยนต์เหล่านั้นขึ้นมาผ่านโค้ดคำสั่ง

2.2.3 เครื่องมือในการแสดงผลภาพ 2 มิติ

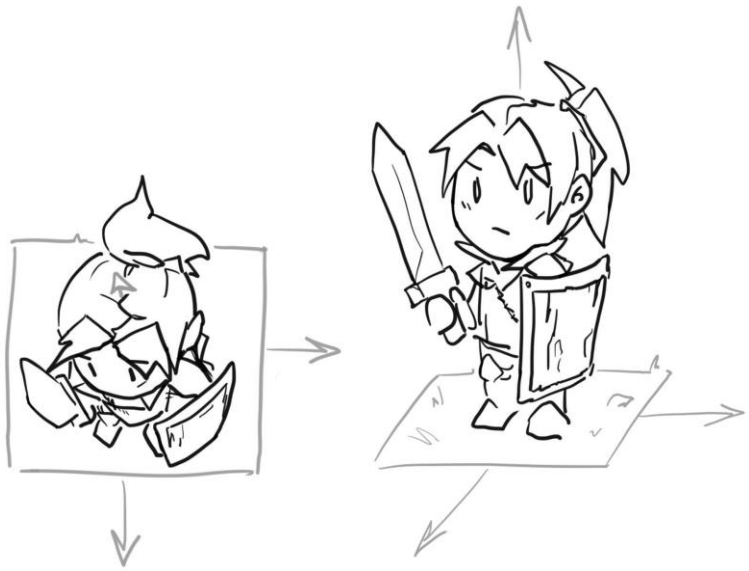
มีเครื่องมือที่สนับสนุนการทำเกมสองมิติ ทั้งการวาดรูปภาพที่กำหนด วาดรูปทรง วาดภาพเคลื่อนไหว หรือแม้กระทั่งการนำรูปภาพเหล่านั้นมาทำเป็นวัตถุหนึ่งๆ ตามหลักการของ Object-Oriented Programming รวมถึงเครื่องมือในการควบคุมตำแหน่ง และจัดการตำแหน่งกล้อง (ขอบเขตของภาพที่จะนำมาแสดงผลจริงบนจอเมื่อเริ่มเกม)



รูปที่ 2.3 ตัวอย่างเกมสองมิติที่ใช้เครื่องมือต่างๆ ของ Godot Engine ในการสร้าง

ที่มา : Godot Engine. 2D Platformer Demo. https://github.com/godotengine/godot-demo-projects/tree/3.1-c34a2b4/2d/platformer.

2.2.4 การจัดเรียงการแสดงผลโดยตัดสินใจจากลำดับความใกล้ไกลกับผู้เล่น ysort[10]



รูปที่ 2.4 ตัวอย่างภาพ2มิติที่เป็นมุมบน และแบบเยื้องรูปมาข้างหน้า ตามลำดับ

ในกรณีที่ภาพในเกมไม่ได้แสดงผลในมุมบนตรงๆ แต่มีการเยื้องมาด้านหน้าเพื่อให้เห็นส่วนลำตัวของตัววัตถุหรือตัวละครต่างๆ อย่างเกมนี้ จะทำให้มีโอกาสที่รูปภาพจะซ้อนทับกันได้เนื่องจากสามารถตีความได้ว่าเป็นการยืนอยู่ข้างหน้า ข้างหลัง ของกันและกัน ดังรูปที่2.4

รูปที่ 2.5 ตัวอย่างการวาดภาพของวัตถุที่ซ้อนทับกันลงบนจอ โดยมีลำดับการวาดที่ต่างกัน

ซึ่งเมื่อเกิดกรณีนี้ขึ้น ลำดับการวาดภาพลงบนจอจะมีผลเป็นอย่างมาก เนื่องจากรูปที่วาดทีหลังจะทับรูปที่ถูกวาดก่อนหน้าเมื่อมีส่วนของรูปที่ overlap กัน ทำให้ผู้ดูสามารถตีความได้ว่าขณะนี้ คนกำลังยืนอยู่หน้าพุ่มไม้ หรือพุ่มไม้กำลังตั้งอยู่หน้าคน ดังรูปที่ 2.5 ซึ่งเครื่องมือ YSort นี้จะช่วยตัดสินได้ว่ารูปไหนอยู่ใกล้ผู้ดูมากกว่ากันและวาดออกมาก่อน หลัง ตามค่าแกน y ที่ระบุ

2.2.5 collistion dectection[11]

การตรวจสอบการสัมผัสกันของวัตถุ เป็นเครื่องมือเกี่ยวกับ physic อย่างหนึ่งที่จำเป็นต้องมีในเกมที่วัตถุสามารถมีปฏิสัมพันธ์กันได้ เช่นการชนกัน หรือแม้แต่การตรวจจำการสัมผัสกัน เพื่อนำไปเป็นทริกเกอร์ต่อเหตุการณีต่างๆ ต่อไปได้ อย่างการตรวจสอบการชนกันของลูกกระสุนกับวัตถุเพื่อลดค่าพลังชีวิต การมีเครื่องมือนี้จะทำให้การเขียนโค้ดนั้นง่ายและมีประสิทธิภาพขึ้นมาก โดยเฉพาะกับวัตถุที่มีรูปทรงซับซ้อนและหลากหลาย

2.2.6 รองรับการบรรจุข้อมูลและเขียนไฟล์

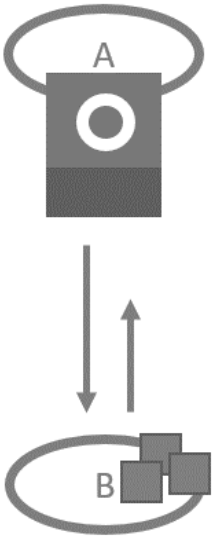
ด้วย PackedScene[12] ที่มีมาให้ ทำให้การบรรจุข้อมูลและการดึงข้อมูลกลับมาสู่สถานะเดิมใหม่เป็นเรื่องง่าย เราไม่จำเป็นต้องคอยบรรจุข้อมูลเองทั้งหมด ง่ายต่อการทำระบบโหลดเกมและเซฟเกม ทั้งยังรองรับการเขียนไฟล์อ่านไฟล์ ทำให้สามารถโหลดเกมจากจุดเดิมได้ แม้จะทำการปิดตัวโปรแกรมและเปิดขึ้นมาใหม่

2.3 การโปรแกรมคำสั่งให้หุ่นยนต์

2.3.1 การเขียนคำสั่งโดยขึ้นกับเหตุการณ์

การโปรแกรมคำสั่งหุ่นยนต์ในเกมนี้จะอาศัยรูปแบบที่ใช้เหตุการณ์ต่างๆเป็นไก (trigger) ว่าหากเกิดสถานการณ์ตามที่ระบุให้ทำอย่างไร ต่างจากการโปรแกรมชุดคำสั่งไปตรงๆ เพื่อให้สามารถสร้างเงื่อนไขและ การตัดสินใจต่างๆ ตามสถานการณ์กับหุ่นยนต์ได้

ดังเช่นสถานการณ์ต่อไปนี้ที่ต้องการให้หุ่นยนต์ เดินทางไปเก็บทรัพยากรจากจุด A ไปเก็บทรัพยากรที่จุด B แล้วนำกลับไปส่งที่จุด A จะสามารถเขียนเป็นชุดคำสั่งทั้ง 2 แบบได้ดังรูปที่ 2.6



การเขียนโปรแกรมปกติ
1. เดินไปจุด B
2. หยิบทรัพยากร
3. เดินไปจุด A
4. วางทรัพยากร

การเขียนขึ้นกับเหตุการณ์
ไม่ได้ถือทรัพยากร - เดินไปจุด B
ไม่ได้ถือทรัพยากร และ พบทรัพยากร - หยิบทรัพยากร
ถือทรัพยากร - เดินไปจุด A
ถือทรัพยากร และ อยู่จุด A - วางทรัพยากร

รูปที่ 2.6 ตัวอย่างการแก้ปัญหาหุ่นยนต์ขนส่งทรัพยากร โดยใช้วิธีโปรแกรมที่ต่างกัน

พบว่าการเขียนแบบปกติอาจจะเขียนได้สั้นกว่าก็จริง แต่สถานการณ์จริงในเกมที่ปัจจัยไม่ได้มีแค่หุ่นยนต์ของผู้เล่นเพียงเครื่องเดียวในเกมนั้น การเขียนขึ้นกับเหตุการณ์จะสามารถรองรับเหตุการณ์ที่อาจเกิดขึ้นได้มากกว่า เช่นในสถานการณ์ที่พบทรัพยากรตกอยู่ระหว่างทาง A ไป B (อาจเกิดขึ้นได้จากหุ่นยนต์ตัวอื่นถูกศัตรูทำลายระหว่างขนทรัพยากร) การเขียนแบบขึ้นกับเหตุการณ์ก็จะเข้าเงื่อนไขของเหตุการณ์ ‘ไม่ได้ถือทรัพยากร และ พบทรัพยากร’ แล้วข้ามขั้นตอนหยิบทรัพยากรนั้นแล้วนำไปส่งที่ฐานได้เลย ต่างจากการเขียนโปรแกรมธรรมดาที่จะมองข้ามและทำการไปเก็บทรัพยากรที่จุด B ตามปกติ หรือต่อให้ตั้งใจสร้างเงื่อนไขเพิ่มก็ยากต่อการใช้งานได้จริง เนื่องจากอาจไม่ใช่แค่ต้องเพิ่มคำสั่งระหว่างการเดินไปจุด B แต่อาจต้องเพิ่มคำสั่งแทรกในทุกๆ ที่เลยได้ อย่างการจะทำให้หุ่นยนต์สามารถหลบหนีได้เมื่อเจอศัตรู

2.3.2 การจัดลำดับความสำคัญของเหตุการณ์

เพื่อแก้ไขความกำกวมที่อาจเกิดขึ้นได้เมื่อสถานการณ์ที่เกิดขึ้นไปตรงเงื่อนไขของเหตุการณ์หลายๆ เหตุการณ์พร้อมกัน อย่างเช่นรูปที่ 2.6 การเขียนแบบขึ้นกับเหตุการณ์ข้างต้น จะพบว่าเงื่อนไข 'ไม่ได้ถือทรัพยากร และ พบทรัพยากร' ในเหตุการณ์ที่ 2 นั้น จะเข้าเงื่อนไขของ 'ไม่ได้ถือทรัพยากร' ในเหตุการณ์แรกด้วย ทำให้เกิดความกำกวมและหุ่นยนต์ไม่สามารถตัดสินใจได้ แก้ปัญหาได้โดยให้ความสำคัญกับเหตุการณ์จากมากไปน้อยตามลำดับ และหุ่นยนต์จะเลือกเหตุการณ์แรกสุด(สำคัญที่สุด)ที่ตรงเงื่อนไขเสมอ

หากแก้ไขความกำกวมของการเขียนขึ้นกับเหตุการณ์ในรูป 2.6 โดยเรียงตามลำดับความสำคัญได้ดังนี้

การเขียนขึ้นกับเหตุการณ์
1. ถือทรัพยากร และ อยู่จุด A - วางทรัพยากร
2. ถือทรัพยากร - เดินไปจุด A
3. ไม่ได้ถือทรัพยากร และ พบทรัพยากร - หยิบทรัพยากร
4. ไม่ได้ถือทรัพยากร - เดินไปจุด B

รูปที่ 2.7 การเขียนแบบขึ้นกับเหตุการณ์ ที่แก้ไขความกำกวมโดยใช้การให้ลำดับความสำคัญ

นอกจากนี้แล้วการลำดับความสำคัญอาจทำให้สามารถละเงื่อนไขบางเงื่อนไขไปได้ เนื่องจากถูกเงื่อนไขของเหตุการณ์ก่อนหน้ากรองไปแล้ว จึงเหลือเพียงคำสั่งดังรูปที่ 2.8

การเขียนขึ้นกับเหตุการณ์
1. ถือทรัพยากร และ อยู่จุด A - วางทรัพยากร
2. ถือทรัพยากร - เดินไปจุด A
3. พบทรัพยากร - หยิบทรัพยากร
4. - เดินไปจุด B

รูปที่ 2.8 แผนผังแสดงการทำงานจากรูป2.7ที่ผ่านการละส่วนที่ไม่จำเป็นแล้ว

2.3.3 การข้ามคำสั่งที่ไม่สามารถทำได้

เมื่อเจอเข้ากับคำสั่งบางอย่างที่ถึงแม้จะถูกตามเงื่อนไขของเหตุการณ์แล้วแต่ก็ยังไม่สามารถทำตามคำสั่งนั้นได้ เช่น หาเป้าหมายของคำสั่งไม่พบ หรือไม่มีเป้าหมายของคำสั่งที่ตรงกับฟิลเตอร์ หุ่นยนต์ก็จะทำการข้ามเหตุการณ์นั้นไปหาเหตุการณ์ต่อๆ ไปโดยอัตโนมัติ โดยถือว่าเหตุการณ์นั้นไม่ตรงเงื่อนไข

จากสคริปต์คำสั่งก่อนหน้ารูป 2.8 เราจึงสามารถละเงื่อนไขบางอย่างได้อีกเป็น

การเขียนขึ้นกับเหตุการณ์
1. อยู่จุด A - วางทรัพยากร
2. ถือทรัพยากร - เดินไปจุด A
3. - หยิบทรัพยากร
4. - เดินไปจุด B

รูปที่ 2.9 แผนผังแสดงการทำงานจากรูป2.8ที่ผ่านการละเงื่อนไขที่ไม่จำเป็นแล้ว

2.3.4 การทำคำสั่งๆ หนึ่งซ้ำไปเรื่อยๆ จนกว่าจะตรงเงื่อนไข (until)

ในบางกรณี การสร้างเงื่อนไขอย่างเดียวก็ไม่เพียงพอ เช่น การสั่งให้หุ่นยนต์คอยเดินวนไปกลับระหว่างจุด X และ Y เราจะสามารถเขียนเป็นสองคำสั่งได้ว่า เมื่ออยู่จุด X ให้เดินไปที่จุด Y และเมื่ออยู่จุด Y ให้เดินไปที่จุด X แต่เมื่อคิดดูดีๆ แล้วด้วยคำสั่งที่มีตอนนี้มันก็ยังไม่เพียงพอให้ทำอย่างที่ต้องการได้ เนื่องจาก เมื่อหุ่นยนต์เคลื่อนที่ออกห่างจากจุด X แล้ว มันจะไม่ตรงเงื่อนไขอะไรเลย หรือถึงเราจะเพิ่มเงื่อนไขเข้าไป เช่น เงื่อนไขว่าเมื่ออยู่ระหว่างจุด X และ Y ให้เดินไปจุด Y มันก็จะใช้ได้แค่ขาไปจุด Y แต่ไม่สามารถกลับมาจุด X ได้อยู่ดี เนื่องจากหุ่นยนต์ไม่สามารถแยกจากคำสั่งที่ว่าเมื่ออยู่ระหว่างจุด X และ Y มันเป็นขาไปหรือขากลับได้

เราจึงได้เพิ่มเงื่อนไขแบบ until ขึ้นมา เพื่อให้มองคำสั่งนั้นเป็นสิ่งที่ต้องทำซ้ำไปเรื่อยๆ จนกว่าจะตรงเงื่อนไขอีกเงื่อนไขที่กำหนด ทีนี้เราก็จะสามารถเขียนได้ว่า เมื่ออยู่จุด X ให้เดินไปที่จุด Y และทำสิ่งนี้เรื่อยๆ จนกว่าจะถึงจุด Y และ เมื่ออยู่จุด Y ให้เดินไปที่จุด X และทำสิ่งนี้เรื่อยๆ จนกว่าจะถึงจุด X

2.3.5 การใช้ตัวแปร และการเขียนทับเพื่อให้หุ่นยนต์มีเป้าหมายที่ต่างกันได้

เพื่อที่ผู้เล่นจะได้ไม่ต้องเขียนสคริปต์คำสั่งซ้ำๆ ให้กับหุ่นยนต์ที่มีลักษณะการทำงานเดิมๆ หลายตัวที่ต่างกันเพียงเป้าหมายการกระทำ จึงใช้ระบุถึงจุดแตกต่างเหล่านั้นด้วยตัวแปรที่สามารถกำหนดค่าใหม่ได้แทน

ยกตัวอย่างเช่นเมื่อผู้เล่นต้องการสร้างหุ่นยนตร์ขึ้นมาเพื่อคอยปกป้องพื้นที่ของผู้เล่นจากเหล่าศัตรู โดยพื้นที่ที่ต้องการปกป้องมีหลายแห่งคือ พื้นที่ A B C ผู้เล่นก็ไม่จำเป็นต้องสร้างสคริปต์ของหุ่นยนต์แตกต่างกันถึง 3 แบบ ว่า ‘สำหรับหุ่นยนต์ปกป้องพื้นที่ A’ ‘สำหรับหุ่นยนต์ปกป้องพื้นที่ B’ ‘สำหรับหุ่นยนต์ปกป้องพื้นที่ C’ แต่สามารถเขียนเพียง ‘สำหรับหุ่นยนต์ปกป้องพื้นที่’ แล้วระบุถึง ‘พื้นที่เป้าหมาย’ ในสคริปต์ด้วยตัวแปรแทน และจึงค่อยกำหนดค่าของตัวแปรในภายหลังเมื่อนำไปติดตั้งจริงให้กับหุ่นยนต์

2.3.6 การกรองเป้าหมาย (การเพิ่ม Filter)

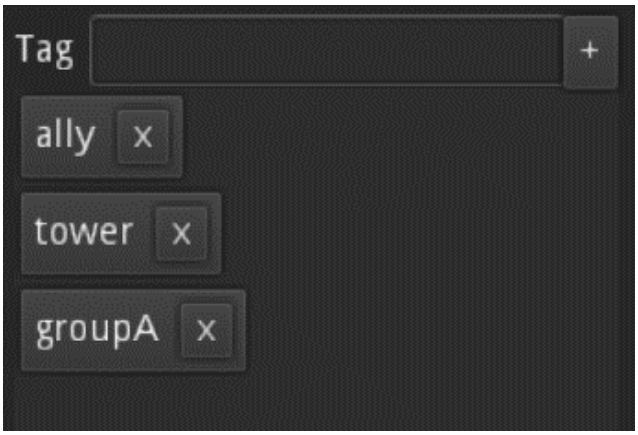
ในบางครั้งเป้าหมายของคำสั่งที่หุ่นยนต์พบอาจไม่ได้มีแค่เป้าหมายเดียว อย่างเช่นการเก็บทรัพยากรที่แสดงในรูปที่ 2.6 ที่ความจริงเราจะพบทรัพยากรหลายๆ ชิ้นพร้อมๆ กัน แต่คำสั่งหยิบทรัพยากรกระทำได้เพียงแค่ทีละเป้าหมาย เราจึงสามารสร้างฟิลเตอร์เพิ่มเติมได้ว่าจะให้หุ่นยนต์ เลือกหยิบทรัพยากรชิ้นไหน เช่น หยิบแต่ชิ้นที่อยู่ในระยะ ชิ้นที่มีมูลค่าเท่านี้ขึ้นไป หรือนำไปใช้กับอย่างอื่นก็ได้เช่นกัน เช่น ให้หุ่นยนต์ตามโจมตีเฉพาะศัตรูที่เลือดน้อยกว่าเรา ทั้งนี้ยังนำไปใช้กับส่วนที่เป็นเงื่อนไขได้อีกด้วย เช่น หนีเฉพาะเมื่อเจอศัตรูที่เลือดมากกว่าเรา

2.3.7 การปักป้ายtagให้กับพื้นที่ หรือวัตถุแวดล้อม

จากตัวอย่างที่ผ่านมา จะพบว่าคำสั่งส่วนหนึ่งมีการระบุถึงพื้นที่ต่างๆ อย่างจุด A B C ซึ่งเราสามารถกำหนดจุดเหล่านี้ขึ้นมาได้ด้วยการปักป้าย ไม่ว่าจะเป็นการปักที่พื้นที่ต่างๆ หรือปักไปกับหัวหุ่นยนต์ตัวอื่น เพื่อให้เดินตามไปที่จุดต่างๆ หรือตามโจมตีหุ่นยนต์ศัตรูที่เราปักป้ายอยู่ในขณะนั้นได้

2.3.8 การเพิ่ม hashtag ให้กับหุ่นยนต์ตัวใดๆ เพื่อให้กล่าวถึงหรือจัดกลุ่มได้

นอกจากการปักป้ายเพื่อกล่าวถึงแล้ว เรายังสามารถฝังชื่อต่างๆ ให้กับหุ่นยนต์โดยตรงได้ ซึ่งมีข้อได้เปรียบตรงที่เราสามารถฝังชื่อที่ซ้ำๆ กัน ให้กับหุ่นยนต์หลายๆตัว ได้เพื่อจัดกลุ่ม หรือแบ่งประเภทของหุ่นยนต์ เช่น การสร้างหุ่นยนต์กลุ่มหนึ่งๆมาที่มีพลังโจมตีสูงแล้วฝังชื่อไว้ว่าเป็น tower เพื่อที่จะได้โปรแกรมสคริปต์ได้ว่า ถ้าหุ่นยนต์ของเราถูกอีกฝ่ายโจมตี ให้หนีไปหา tower เพื่อให้ tower ช่วยโจมตี



รูปที่ 2.10 ตัวอย่างเมนูการฝัง Hashtag ต่างๆ ให้กับหุ่นยนต์

จะมีการฝังชื่อ ally ให้กับหุ่นยนต์ฝั่งเรา และ enemy ให้กับหุ่นยนต์ฝั่งศัตรูโดยอัตโนมัติ เพื่อกล่าวถึงได้เลย โดยไม่ต้องคอยใส่เองให้กับหุ่นยนต์ทุกๆ ตัว

เพื่อให้ได้เป้าหมายตามที่เจาะจงไปอีก การกล่าวถึงกลุ่มหุ่นยนต์เหล่านี้ยังสามารถนำมารวม หรือลบกันได้ คล้ายเรื่องเซต โดยจะมี operation เพิ่มให้ดังนี้

- **A <union> B** หมายถึง หุ่นยนต์ที่มีใน A หรือมีใน B
- **A <intersect> B** หมายถึง หุ่นยนต์ที่มีใน A และมีใน B
- **A <minus> B** หมายถึง หุ่นยนต์ที่มีใน A แต่ไม่มีใน B

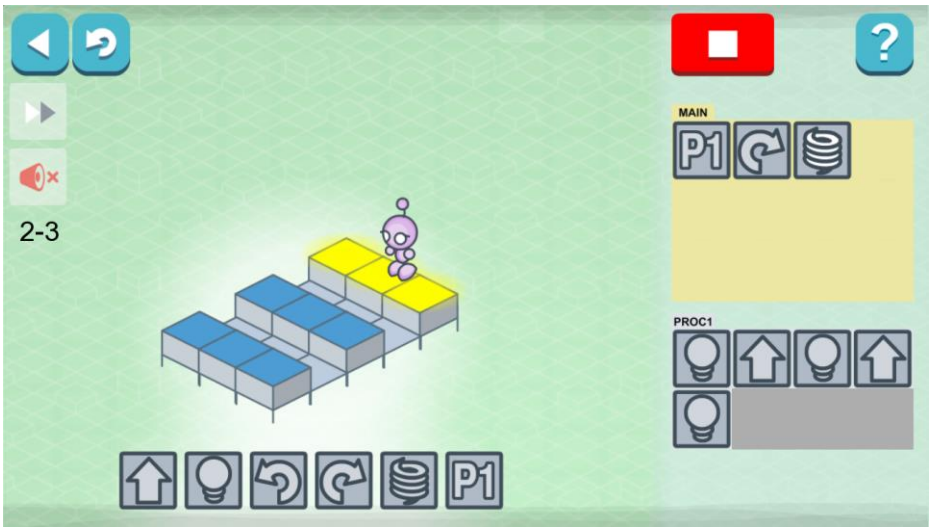
2.3.9 ประเภทของคำสั่ง (Hard และ Soft)

เนื่องจากในบางคำสั่ง ก็เป็นคำสั่งที่ไม่อยากให้หุ่นยนต์สามารถทำไปพร้อมๆกันได้ เช่นการเคลื่อนที่และชาร์จพลังงาน แต่ในบางคำสั่งก็เป็นเพียงคำสั่งเล็กๆ ที่ควรจะทำไปพร้อมๆกับคำสั่งอื่นได้ เช่นการโจมตีขณะเคลื่อนที่ เราจึงแบ่งประเภทของคำสั่งขึ้นมา โดยคำสั่ง Soft จะเป็นคำสั่งที่เมื่อทำแล้วจะสามารถเลื่อนไปหาคำสั่งอื่นที่ระดับความสำคัญต่อไปได้ แต่เมื่อเจอและทำคำสั่ง Hard จะเป็นการ Break ออกมาจากการหาคำสั่งทำทันที เพื่อเป็นการกันไม่ให้ทำพร้อมกับคำสั่งอื่นได้อีก

2.4 เกมที่เกี่ยวข้อง

เพื่อออกแบบเกมให้มีความสนุกและเข้าถึงง่ายจึงได้ทำการศึกษาเกมที่มีรูปแบบที่คล้ายคลึงกันกับสิ่งที่ผู้จัดทำต้องการ วิเคราะห์ข้อดีข้อด้อยเพื่อที่จะได้นำจุดสังเกตุที่ได้มาใช้ให้เป็นประโยชน์ในการพัฒนาเกมต่อไป

2.4.1 Light Bot [1]



รูปที่ 2.11 ตัวอย่างภายในเกม Light Bot

เป็นเกม puzzle แบบตะลุยด่านที่ช่วยฝึกความรู้ความเข้าใจในการโปรมแกรมคำสั่ง เนื่องจากตัวละครในเกมจะทำตามคำสั่งแต่ละคำสั่งของผู้เล่นตามที่ระบุ และตามลำดับเสมอ โดยมีเป้าหมายคือไปจุดไฟ ในตำแหน่งพื้นสีฟ้าให้ครบทุกตำแหน่ง จึงเป็นการฝึกให้ผู้เล่นได้แก้ปัญหาที่กำหนดจากการนำคำสั่งพื้นฐานที่ มีอยู่อย่างจำกัดมาประยุกต์ใช้ เช่น การเดินไปข้างหน้า1ช่อง หันซ้าย หันขวา กระโดด หรือจุดไฟ

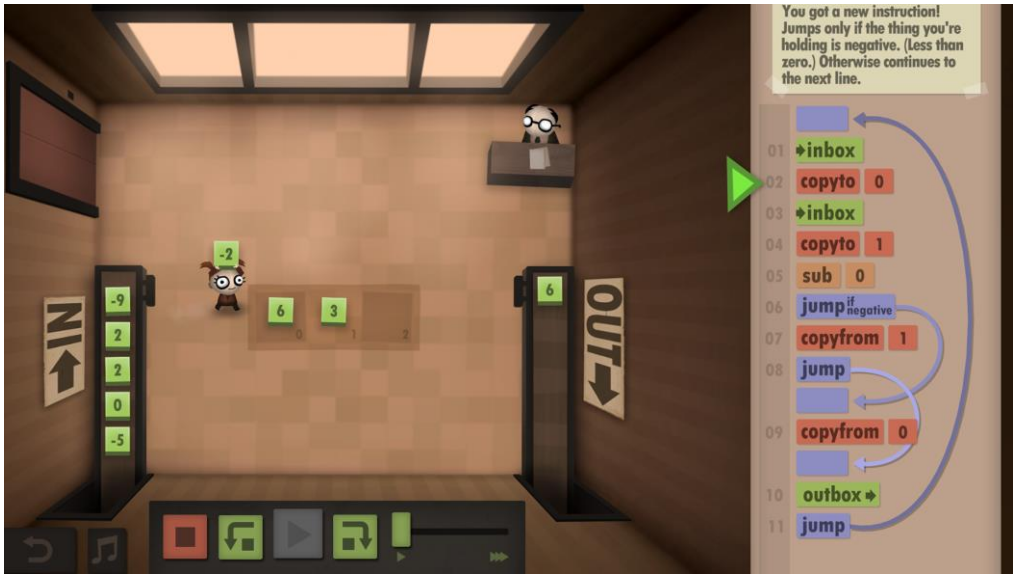
สิ่งที่ประทับใจของเกมที่อยากนำมาปรับใช้

1). รูปแบบคำสั่งควบคุมที่น้อย เข้าใจง่าย แต่กลับนำมาใช้ได้หลากหลาย

2). การรวบรวมชุดคำสั่งหนึ่งๆเป็นฟังก์ชัน เพื่อให้ง่ายต่อการเข้าใจ และนำมาใช้ใหม่

3). ผู้เล่นสามารถเร่งความเร็วเกมได้

2.4.2 Human Resource Machine [2]



รูปที่ 2.12 ตัวอย่างภายในเกม Human Resource Machine

เป็นเกม puzzle แบบตะลุยด่านที่มีความยาก และให้ความรู้สึกเหมือนการเขียนโปรแกรมขึ้นมาจริงๆ เนื่องจากเป็นการรับชุดของ input มาผ่านกระบวนการพื้นฐานเช่น การหยิบเลขจาก input ขึ้นมา คัดลอกเลขไปวางสำรองไว้ในตำแหน่งที่ระบุ นำเลขจากตำแหน่งที่ระบุขึ้นมาถือ นำเลขจากตำแหน่งที่ระบุมาเพิ่มให้กับเลขที่ถืออยู่ ลบเลขจำนวนเดียวกับตำแหน่งที่ระบุออกจากที่ถืออยู่ กระโดดไปบรรทัดต่างๆ กระโดดถ้าค่าที่ถืออยู่เป็นลบ กระโดดถ้าค่าที่ถืออยู่เป็นศูนย์ และการนำเลขที่ถืออยู่ไปส่งที่output เพื่อแก้ปัญหาตามที่โจทย์ระบุ เช่นการแสดง(OUT) เลขที่มีค่ามากกว่า ในทุกๆ สองเลขที่เข้ามา(IN) ดังรูปตัวอย่าง

ซึ่งเป็นเกมที่ให้ความรู้สึกเหมือนกับการเขียนโค้ดโปรแกรมจริงในบางภาษามากๆ (โดยเฉพาะภาษา Assembly) ทั้งการที่เราถือเลขได้เพียงทีละเลขและกระทำการต่างๆ ได้กับเพียงเลขที่ถืออยู่เท่านั้น ก็เปรียบเหมือน register หรือการคัดลอกเลขไปวางไว้ตำแหน่งต่างๆ ก็เหมือนกับการฝากเลขไว้กับตัวแปร

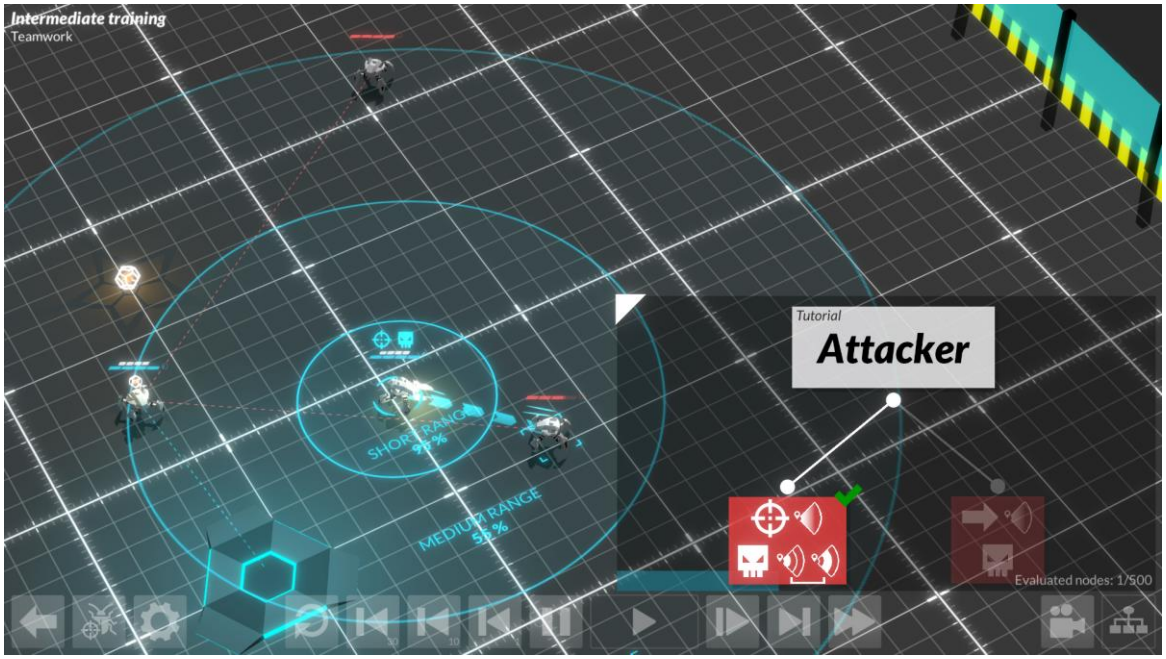
สิ่งที่ประทับใจของเกมที่อยากนำมาปรับใช้

1). ผู้เล่นสามารถเร่งความเร็วเกม รวมทั้งรันคำสั่งไปข้างหน้า หรือย้อนกลับทีละคำสั่งได้

2). มีช่องสำรองให้ผู้เล่นสามารถบันทึกโค้ดคำสั่งเก่าได้ เพื่อที่จะได้ไม่ต้องเคลียร์ใหม่ทั้งหมด เมื่อจะลองเขียนชุดคำสั่งใหม่

3). การสร้างเงื่อนไขให้ชุดคำสั่งได้อย่างอิสระ ทำให้สามารถสร้างชุดคำสั่งที่มีความซับซ้อนได้

2.4.3 Gladiabot [3]



รูปที่ 2.13 ตัวอย่างภายในเกม Gladiabot

เป็นเกมแบบตะลุยด่านที่มีเป้าหมายอยู่ที่การออกแบบคำสั่งAI ให้กับหุ่นของฝั่งตัวเอง เพื่อเอาชนะAI ของหุ่นยนต์ฝั่งตรงข้าม ในรูปแบบที่เข้าใจง่ายอย่างการใส่เงื่อนไขให้กับคำสั่งโดยเรียงลำดับความสำคัญ เช่นการเดินเข้าหาศัตรู การหนีออกจากศัตรู การโจมตีศัตรู และการเก็บทรัพยากร เป็นต้น โดยเกมนี้จะเน้นที่การวางแผนเป็นหลัก แข่งกันขนส่งทรัพยากรไปยังจุดที่กำหนดให้ได้มากกว่าอีกฝั่ง มีระบบเกมที่เยอะ

และละเอียดมาก ทั้งการทำงานเป็นทีม แบ่งตัวโจมตีและป้องกัน การtagเป้าหมายเพื่อแบ่งกันโจมตีหรือรุมทำลายทีละตัว การเรียงลำดับความสำคัญของเป้าหมายเช่นเน้นเป้ามหายที่อยู่ใกล้ที่สุดหรือเลือดน้อยที่สุด การfilter เป้าหมายเช่นสนใจเฉพาะเป้าหมายที่อยู่ในระยะโจมตีอยู่ใกล้กับเรา การใส่เงื่อนไขที่มาพร้อมกับตัวเชื่อม และ, หรือ รวมถึงรูปแบบของหุ่นยนต์ที่แตกต่างกันถึงสี่แบบ อย่างตัวที่พลังชีวิตเยอะแต่เคลื่อนที่ช้า ตัวที่โจมตีช้าแต่โจมตีได้ไกลและแรง เพื่อสนับสนุนกันและกัน และสร้างรูปแบบที่หลากหลาย ทั้งนี้ยังสามารถเข้าเล่นแบบออนไลน์เพื่อแข่งกับAI ของฝั่งตรงข้ามที่สร้างโดยผู้เล่นจริงๆ ในด่านต่างๆ ได้อีกด้วย

สิ่งที่ประทับใจของเกมที่อยากนำมาปรับใช้

1). ผู้เล่นสามารถเร่งความเร็วเกม รวมทั้งรันคำสั่งไปข้างหน้า หรือย้อนกลับทีละคำสั่งได้

2). การออกแบบแผนผังการเขียนคำสั่งที่เข้าใจได้ง่าย แต่ก็ยังสามารถเขียนให้มีความซับซ้อนสูงได้

3). การสร้างชุดคำสั่งแยกจากตัวหุ่นยนต์ โดยให้หุ่นยนต์ชี้ไปยังชุดคำสั่งร่วมกันได้ ทำให้ง่ายต่อการควบคุมเมื่อมีหุ่นยนต์จำนวนมาก เช่นเมื่อแก้ไขที่ชุดคำสั่งก็จะมีผลไปถึงหุ่นยนต์ที่ชี้มายังชุดคำสั่งนี้ทั้งหมด

4). การมีเป้าหมายอื่นให้แก่เกม(แย่งชิงทรัพยากร) ทำให้เกมมีความหลากหลายในการวางแผนมากขึ้น นอกเหนือจากการทำลายอีกฝ่ายให้หมด

2.4.4 Bad Piggies [4]

รูปที่ 2.14 ภาพตัวอย่างเกม Bad Piggies

เป็นเกม puzzle แบบตะลุยด่านที่เน้นการเอาเครื่องยนต์หรือชิ้นส่วนต่างๆ มาประกอบกัน เพื่อนำพาเหล่า piggie ไปยังเป้าหมายที่กำหนดให้ได้ โดยเน้นความคิดสร้างสรรค์และให้ความสำคัญไปที่ระบบ physic เป็นอย่างมาก เนื่องจากชิ้นส่วนที่นำมาติดก็จะมีทั้ง เครื่องยนต์ ล้อ ใบพัด บอลลูน ที่ถ่วงน้ำหนัก หรือไอพ่นต่างๆ โดยต้องคำนึงถึงทั้งความคงทน น้ำหนัก และสมดุลน้ำหนักและแรงต่างๆ เพื่อพาpiggieไปยังเป้าหมายหรือเก็บเงื่อนไขพิเศษของแต่ละด่านให้ได้ (อย่างการไม่ใช้ชิ้นส่วนที่กำหนด หรือเก็บโบนัสต่างๆ ในแผนที่)

สิ่งที่ประทับใจของเกมที่อยากนำมาปรับใช้

1). การที่สามารถปรับแต่งได้หลากหลาย ส่งผลเป็นอย่างมากต่อการเล่น และคำสั่งควบคุมที่สามารถทำได้

2). มีโหมดสร้างสรรค์ที่อนุญาตให้ปรับแต่งได้อย่างอิสระไม่จำกัดเครื่องมือใดๆ
เพื่อให้ผู้เล่นได้ลองออกแบบได้อย่างเต็มที่

สิ่งที่อยากปรับปรุง

1). ชิ้นส่วนที่ใช้งานได้ต่างๆ ในเกม จะถูกใช้งานผ่านการควบคุมของผู้เล่นเองทั้งหมด
ทำให้เกิดความยุ่งยากที่ต้องคอยควบคุมเองอยู่ตลอดและข้อจำกัดต่างๆ ตามมา

บทที่ 3

การวิเคราะห์และออกแบบระบบ

ในบทนี้จะกล่าวถึงการวิเคราะห์และออกแบบระบบการโปรแกรมคำสั่งให้แก่หุ่นยนต์อันเป็นองค์ประกอบหลักของงานชิ้นนี้ ทั้งโครงสร้างของระบบ ลำดับการทำงาน และสิ่งที่ผู้เล่นสามารถทำได้ ให้สอดคล้องกับกฎกติกาของเกมที่ระบุไว้

3.1 ปัญหาและการออกแบบ

3.1.1 กฎกติกาของเกม

1). เกมเป็นเกมสองมิติจากมุมบน ควบคุมด้วยเม้าส์เป็นหลักและอาจมีการใช้คีย์บอร์ดเสริมบ้าง เพื่อป้อนข้อความต่างๆ ในการโปรแกรมคำสั่งให้หุ่นยนต์

2). เมื่อเริ่มเกมหุ่นยนต์หลักของผู้เล่นจะถูกสุ่มตำแหน่งเริ่มต้นในเกมที่มีแผนที่ที่ถูกสร้างขึ้นมาแบบสุ่ม พร้อมกับชิ้นส่วนเริ่มต้นอีกจำนวนหนึ่ง ที่สามารถนำมาประกอบกันเป็นหุ่นยนต์เพิ่มเติมเพื่อสนับสนุนหุ่นยนต์หลักได้

3). เป้าหมายของผู้เล่นคือเก็บสะสมคะแนน และเอาชีวิตรอดอยู่ให้ได้นานที่สุด เพราะเมื่อหุ่นยนต์หลักของผู้เล่นถูกทำลาย (พลังชีวิตหมด) จะถือเป็นการจบเกม และมีการแสดงคะแนนรวมของการเล่นในรอบนี้

4). ผู้เล่นสามารถสร้างหุ่นยนต์เพิ่มขึ้นมาได้ โดยการนำชิ้นส่วนที่มีมาประกอบกัน โดยจะมีชิ้นส่วนโครงที่จะเป็นตัวกำหนดลักษณะภายนอกของหุ่นยนต์และจำนวนช่องใส่ชิ้นส่วนที่สามารถนำเอาชิ้นส่วนอื่นๆ มาประกอบเสริมได้ โดยคุณลักษณะของหุ่นยนต์ที่ถูกสร้างขึ้นมา ก็จะถูกกำหนดโดยชิ้นส่วนเหล่านี้ ทั้งความเร็ว พลังชีวิต พลังป้องกัน วิธีการโจมตีและความเสียหายที่ทำได้จากการโจมตี หุ่นยนต์ของผู้เล่นที่ไม่ใช่หุ่นยนต์หลักเหล่านี้สามารถถูกทำลายได้โดยไม่นับว่าเป็นการจบเกม

5). ผู้เล่นสามารถคลิกเลือกหุ่นยนต์ของฝั่งตนเองเหล่านี้เพื่อควบคุม หรือเปิดหน้าต่างเพื่อแก้ไขชิ้นส่วนและโปรแกรมการทำงานของหุ่นยนต์ตัวที่เลือกได้

6). ทั้งนี้การทำงานในบางคำสั่งจะมีการกินพลังงานแบตเตอรี่ของตัวหุ่นยนต์ด้วย ในปริมาณที่แตกต่างกันไป เช่นการเคลื่อนที่ โจมตี หรือรักษาพลังชีวิต และถ้าพลังงานหมดจะไม่สามารถทำอะไรได้ เพื่อให้ได้ใช้ประโยชน์ของการโปรแกรมหุ่นยนต์โดยอัตโนมัติ คือไม่ต้องคอยเติมพลังงาน หรือขนส่งพลังงานอยู่เองเรื่อยๆ พลังงานแบตเตอรี่สามารถสร้างได้โดยชิ้นส่วนบางชนิด ที่ใช้พื้นที่ในการติดตั้งมาก ทำให้ติดตั้งกับหุ่นยนต์เล็กที่คอยเคลื่อนที่ไปมาไม่ได้ ต้องคอยขนส่งเอาจากหุ่นยนต์ใหญ่ ทั้งนี้หุ่นยนต์แต่ละตัวยังมีความจุแบตเตอรี่ที่จำกัดด้วย ทำให้สามารถบรรจุพลังงานแต่ละทีได้ในปริมาณที่จำกัด (สามารถเพิ่มความจุได้ด้วยการติดตั้งชิ้นส่วนเพิ่ม)

7). ในแผนที่ที่ถูกสร้างขึ้นมาแบบสุ่มนั้น จะประกอบไปด้วยพื้นปกติ พื้นที่หุ่นยนต์สามารถหรือไม่สามารถเคลื่อนที่ผ่านไปได้ รวมถึงบ้านซึ่งเป็นจุดเกิดของศัตรู (มีได้หลายจุดในแผนที่)

8). จุดเกิดของศัตรูนั้นจะแผ่ขยายขึ้นเรื่อยๆหากผู้เล่นไม่ไปทำลาย รวมถึงการให้กำเนิดศัตรูที่แข็งแกร่งขึ้นเรื่อยๆเพื่อเข้าทำร้ายผู้เล่น ทำให้เกิดสมดุลเกมที่ว่าเมื่อเริ่มเกมมาก็จะยังเจอแต่ศัตรูที่อ่อนแอ แต่หากเล่นนานๆไปศัตรูก็จะแข็งแกร่งขึ้นเรื่อยๆ และเมื่อขยายแผนที่ออกไปหาพื้นที่ใหม่ๆ บ้านของศัตรูที่เจอตอนหลังก็จะแข็งแกร่งกว่าเก่าด้วยเหตุผลเดียวกัน

รูปที่ 3.1 จุดเกิดของหุ่นยนต์ฝั่งศัตรู

9). ศัตรูในเกมนี้ล้วนเป็นหุ่นยนต์เช่นเดียวกับหุ่นยนต์ของฝั่งผู้เล่น ประกอบด้วยชิ้นส่วนต่างๆ เช่นเดียวกับหุ่นยนต์ของฝั่งผู้เล่น หากแต่ถูกโปรแกรมมาให้เข้าทำร้ายผู้เล่น และเมื่อหุ่นยนต์ศัตรูเหล่านี้ถูกทำลาย ก็มีโอกาสที่ผู้เล่นจะได้ชิ้นส่วนของหุ่นยนต์ที่ถูกทำลายนั้น เพื่อนำมาพัฒนาหุ่นยนต์ของฝั่งตัวเองต่อไป

3.1.2 หุ่นยนต์ที่มีภายในเกม

หุ่นยนต์แต่ละชนิดนั้นจะมีพื้นที่ติดตั้งชิ้นส่วนเพื่อเพิ่มคุณสมบัติต่างๆ ให้แก่หุ่นยนต์ แต่โดยพื้นฐานแล้วแต่ละชนิดก็ยังมีคุณสมบัติเริ่มต้นที่แตกต่างกันไป มีจุดดีจุดด้อยตามแต่ละชนิด ดังนี้

ตารางที่ 3.1 หุ่นยนต์ชนิดต่างๆ ที่มีภายในเกม

สีฝั่งผู้เล่น	สีฝั่งศัตรู	คำอธิบาย
	ไม่มี	**หุ่นยนต์หลัก (Core)** - เกมจบเมื่อถูกทำลาย - คุณสมบัติปานกลาง - พื้นที่ใส่ของ 50 หน่วย
		Assault Drone - คุณสมบัติปานกลาง - พื้นที่ใส่ของ 22 หน่วย
		Speed Drone - เคลื่อนที่เร็ว แต่พลังชีวิตน้อย - สร้างพลังงานใช้เองได้ (น้อย) - พื้นที่ใส่ของ 10 หน่วย
		Defend Drone - เคลื่อนที่ช้า พลังชีวิตสูง - พื้นที่ใส่ของ 18 หน่วย
		Fat Drone - โดยปกติไม่สามารถเคลื่อนที่ได้ พลังชีวิตสูง (อาจทำให้เคลื่อนที่ได้ด้วยการติดตั้งชิ้นส่วนเพิ่ม) - พื้นที่ใส่ของ 50 หน่วย
		Mini Drone - คุณสมบัติปานกลาง - เคลื่อนที่เร็วเล็กน้อย - พื้นที่ใส่ของ 18 หน่วย

3.1.3 คำสั่งหุ่นยนต์ที่มีภายในเกม

คำสั่งทั้งหมดที่สามารถสั่งให้หุ่นยนต์ทำได้ผ่านสคริปต์คำสั่งของหุ่นยนต์ แบ่งเป็นสองประเภทคือแบบ Soft ที่สามารถทำร่วมกับคำสั่งอื่นในรวดเดียวได้ เมื่อทำคำสั่งนี้แล้วจะไปหาคำสั่งอื่นที่ลำดับความสำคัญต่อๆ ไปทำต่อ กับแบบ Hard ที่ไม่สามารถทำร่วมกับคำสั่งอื่นได้อีก

ทั้งนี้หากเป้าหมายของคำสั่งไม่ได้มีวัตถุเดียว แต่เป็นกลุ่มของเป้าหมาย จะทำการดำเนินการเป้าหมายกับวัตถุที่อยู่ใกล้ที่สุดในกลุ่มเป้าหมาย

ตารางที่ 3.2 คำสั่งหุ่นยนต์ต่างๆ ที่มีภายในเกม

คำสั่ง	ประเภท	เป้าหมาย	คำอธิบาย
Move	hard	วัตถุ	เคลื่อนที่เข้าหาเป้าหมายที่ระบุ
Flee	hard	วัตถุ	เคลื่อนที่ออกห่างจากเป้าหมายที่ระบุ
Hold	hard	ไม่	อยู่นิ่งกับที่
Hold Fire	Soft	ไม่	พักการยิงของป้อมอัตโนมัติ (เนื่องจากกินพลังงานแบตเตอรี่เมื่อยิง)
Get Charge	hard	วัตถุ	ทำการถ่ายพลังงานแบตเตอรี่ของตนเองให้เป้าหมาย (ถ้าอยู่นอกระยะจะเป็นการเคลื่อนที่เข้าหาเป้าหมาย)
Heal	hard	วัตถุ	ทำการใช้พลังงานแบตเตอรี่ของตนเองเพื่อรักษาพลังชีวิตให้เป้าหมาย มีเป้าหมายเป็นตนเองได้ (ถ้าอยู่นอกระยะจะเป็นการเคลื่อนที่เข้าหาเป้าหมาย) * ต้องใส่ชิ้นส่วน repair kit ก่อน ถึงจะใช้งานคำสั่งนี้ได้
Add Tag	Soft	ข้อความ	ทำการเพิ่ม Hash Tag ตามที่ระบุให้กับตนเอง
Remove Tag	Soft	ข้อความ	ทำการลบ Hash Tag ตามที่ระบุออกจากตนเอง
Hold Tag	Soft	ข้อความ	ทำการถือ Hash Tag ตามที่ระบุค้างไว้กับตนเอง คือเพิ่ม Hash Tag ให้เมื่อเข้าคำสั่งนี้ และจะลบออกให้เองอัตโนมัติ เมื่อคำสั่งนี้ไม่ทำงานแล้ว

3.1.3 ชิ้นส่วนที่มีภายในเกม

ชิ้นส่วนแต่ละชนิดที่สามารถติดตั้งให้หุ่นยนต์ได้เพื่อเติมตั้งคุณสมบัติของหุ่นยนต์นั้นๆ อาจเป็นแค่การเพิ่มลดค่าเดิมต่างๆ ไปจนถึงเพิ่มความสามารถที่ไม่เคยมีมาก่อนในหุ่นยนต์ตัวนั้นได้ และชิ้นส่วนทุกชิ้นยังสามารถติดตั้งซ้ำให้กับหุ่นตัวเดิมได้เรื่อยๆ เพื่อเพิ่มประสิทธิภาพของชิ้นส่วนที่มีอยู่เดิมทบไปอีกขั้น ยกเว้นชิ้นส่วนประเภทป้อมปืน ที่แต่ละชิ้นจะทำงานแยกจากกันสิ้นเชิง ไม่ได้เป็นแค่การบวกความสามารถ

แต่ก็ยังมีสิ่งที่ต้องคำนึงถึงด้วยคือพื้นที่ที่ใช้ในการติดตั้ง โดยแต่ละชิ้นส่วนจะมีการระบุว่าใช้พื้นที่เท่าไร แล้วเมื่อหุ่นยนต์ติดตั้งจนพื้นที่เต็มแล้ว ก็จะไม่สามารถติดตั้งชิ้นส่วนเพิ่มอีกได้ จนกว่าจะถอดชิ้นส่วนที่เคยติดตั้งไปเดิมออกก่อนให้มีพื้นที่เหลือพอติดตั้ง

นอกจากนี้ยังต้องคำนึงถึงน้ำหนักของชิ้นส่วนที่ติดตั้งอีกด้วย โดยในเกมนี้จะสรุปค่าน้ำหนักออกมาเป็นความเร็วการเคลื่อนที่ของชิ้นส่วนนั้นๆ เลย ว่าถ้าใส่ชิ้นส่วนนี้แล้วจะลดค่าความเร็วของหุ่นยนต์เท่าไร

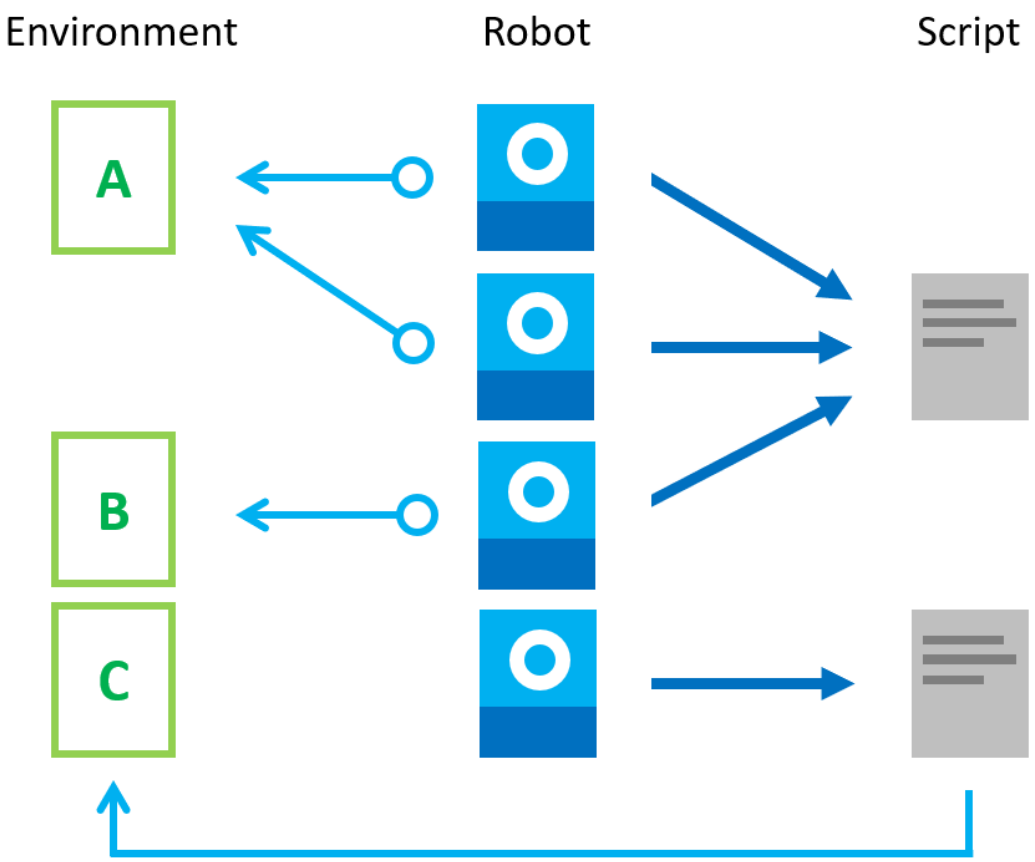
ตารางที่ 3.3 ชิ้นส่วนชนิดต่างๆ ที่มีภายในเกม

ภาพประกอบ	ใช้พื้นที่	คำอธิบาย
	3	Small Booster - เพิ่มความเร็วการเคลื่อนที่เล็กน้อย (100 หน่วย)
	4	Iron Scrap - เพิ่มพลังป้องกันเล็กน้อย - ลดความเร็ว 50 หน่วย
	12	Iron Plate - เพิ่มพลังป้องกันอย่างมาก - ลดความเร็ว 200 หน่วย
	7	Screw - เพิ่มความสามารถรักษาตัวเองอัตโนมัติ(เล็กน้อย) ให้หุ่นยนต์ - ไม่มีน้ำหนัก

ภาพประกอบ	ใช้พื้นที่	คำอธิบาย
	5	**Small Repairkit** - เพิ่มความสามารถรักษาให้แก่หุ่นยนต์ ทำให้เราสามารถสั่งหุ่นยนต์ที่ติดตั้งไปรักษาหุ่นยนต์ตัวอื่นหรือรักษาตัวเองได้ - กินพลังงานแบตเตอรี่เพื่อเพิ่มพลังชีวิต - ลดความเร็ว 50 หน่วย
	2	**AA Battery** - เพิ่มความจุแบตเตอรี่เล็กน้อย - ลดความเร็ว 25 หน่วย
	10	**D Battery** - เพิ่มความจุแบตเตอรี่อย่างมาก - ลดความเร็ว 300 หน่วย
	4	**USB3** - เพิ่มความเร็วในการถ่ายโอนพลังงานแบตเตอรี่ - ไม่มีน้ำหนัก
	3	**Old Antenna** - เพิ่มระยะในการถ่ายโอนพลังงานแบตเตอรี่หรือรักษาหุ่นยนต์อื่น - ลดความเร็ว 25 หน่วย
	4	**Solar Panel** - เพิ่มความสามารถชาร์จพลังงานแบตเตอรี่ตัวเองอัตโนมัติ(เล็กน้อย) ให้หุ่นยนต์ - ลดความเร็ว 75 หน่วย
	15	**Fusion Reactor** - เพิ่มความสามารถชาร์จพลังงานแบตเตอรี่ตัวเองอัตโนมัติ(มาก) ให้หุ่นยนต์ - ลดความเร็ว 200 หน่วย
	10	**Wireless Charger** - เพิ่มความสามารถชาร์จพลังงานแบตเตอรี่ให้หุ่นยนต์ตัวอื่นที่อยู่ในระยะได้โดยอัตโนมัติ

ภาพประกอบ	ใช้พื้นที่	คำอธิบาย
		- มีflowที่จำกัด ถ้ามีหุ่นยนต์ในระยะมาก ก็จะถูกหาร ทำให้ชาร์จให้หุ่นยนต์แต่ละตัวช้าไปด้วย (เพิ่มflowได้โดยใส่ชิ้นส่วนนี้เพิ่ม) - ลดความเร็ว 150 หน่วย
	6	**Uzi** - ป้อมปืนอัตโนมัติ คอยโจมตีศัตรูที่เข้ามาอยู่ในระยะ - พลังโจมตี (Damage) ปานกลาง - ความแม่นยำ (Accuracy) ปานกลาง - ความถี่ในการยิง (Fire Rate) สูง - ลดความเร็ว 50 หน่วย
	7	**Small Cannon** - ป้อมปืนอัตโนมัติ คอยโจมตีศัตรูที่เข้ามาอยู่ในระยะ - พลังโจมตี (Damage) สูงมาก - ความแม่นยำ (Accuracy) สูง - ความถี่ในการยิง (Fire Rate) ต่ำ - ลดความเร็ว 150 หน่วย
	3	**Mini Turret** - ป้อมปืนอัตโนมัติ คอยโจมตีศัตรูที่เข้ามาอยู่ในระยะ - พลังโจมตี (Damage) ต่ำ - ความแม่นยำ (Accuracy) ต่ำ - ความถี่ในการยิง (Fire Rate) ปานกลาง - ลดความเร็ว 25 หน่วย

3.2 การวิเคราะห์โครงสร้างของระบบ



รูปที่ 3.2 ลักษณะการทำงานของระบบ

Environment สภาพแวดล้อมภายนอก ที่หุ่นยนต์สามารถมีปฏิสัมพันธ์ได้ เช่น ตำแหน่งพื้นที่ หุ่นยนต์ตัวอื่น หรือศัตรู อาจระบุถึงโดยหุ่นยนต์ผ่านตัวแปร หรือระบุถึงได้โดยตรงผ่าน script คำสั่ง

Robot วัตถุภายในเกมที่ผู้เล่นควบคุม เพื่อให้ไปมีปฏิสัมพันธ์กับสิ่งอื่นๆ ภายในเกม โดยควมคุมผ่านสคริปต์คำสั่งที่ติดตั้ง หุ่นยนต์หนึ่งตัวต่อหนึ่งสคริปต์ แต่อาจมีปฏิสัมพันธ์กับสิ่งอื่นๆภายในเกมได้หลายสิ่งพร้อมๆ กัน

Script ชุดคำสั่งที่สามารถป้อนให้หุ่นยนต์ ออกแบบให้แยกออกมาจากตัวหุ่นยนต์เอง เพื่อลดความยุ่งยากในการแก้ไขในภายหลังเมื่อมีหุ่นยนต์จำนวนมากแต่ชุดคำสั่งนั้นเหมือนๆ กัน หรือเป้าหมายของคำสั่งที่แตกต่างกันเพียงเล็กน้อยก็สามารถระบุให้แก่หุ่นยนต์ได้ในภายหลังผ่านตัวแปร โดยไม่ต้องมาคอยแยกหรือแก้ไขสคริปต์อยู่เรื่อยๆ

โดยสิ่งที่ผู้เล่นสามารถทำได้ จะถูกแบ่งเป็นลำดับขั้นตอนดังนี้

1). ผู้เล่นสามารถคลิกเลือกที่ตัวหุ่นยนต์ เพื่อเปิดหน้าต่างการโปรแกรมคำสั่งขึ้นมา

2). ผู้เล่นสามารถสร้างตัวแปรใหม่ หรือระบุค่าของตัวโปรที่มีอยู่แล้วได้

3). ผู้เล่นสามารถเลือกสคริปต์ให้แก่หุ่นยนต์ ไปยังเมนูแก้ไขสคริปต์ เพื่อแก้ไขหรือสร้างสคริปต์ขึ้นมาใหม่ได้

4). ในเมนูแก้ไขสคริปต์ ผู้เล่นจะสามารถแก้ไขเหตุการณ์ที่มีอยู่แล้ว หรือเลือกสร้างเหตุการณ์ใหม่ให้แก่สคริปต์นั้นได้ โดยต้องคำนึงถึงการให้ลำดับความสำคัญจากซ้ายไปขวา

5). ในเหตุการณ์หนึ่งเหตุการณ์ ผู้เล่นจะได้ระบุเงื่อนไขของเหตุการณ์ รวมถึงระบุคำสั่งที่จะทำงานเมื่อเกิดเหตุการณ์ตามเงื่อนไข

6). ในการกำนดเป้าหมายของเหตุการณ์ ผู้เล่นสามารถระบุเป้าหมายไปยังสภาพแวดล้อมโดยตรงหรือติดค่าเป็นตัวแปรไว้ เพื่อให้สามารถคอยระบุหรือแก้ไขที่ตัวหุ่นยนต์ในภายหลังได้

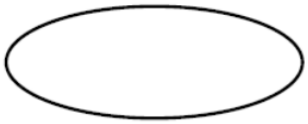
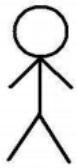
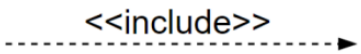
7). ในการระบุเป้าหมายของเหตุการณ์ ผู้เล่นสามารถคลิกเพื่อนเพิ่มฟิลเตอร์ต่างๆของเป้าหมายเพิ่มได้ทั้งลำดับที่ใช้ในการเลือกเป้าหมาย หรือเงื่อนไขที่ใช้กรองเป้าหมายที่ไม่สนใจเพิ่มได้

การทำงานเหล่านี้ จะถูกวิเคราะห์และเรียบเรียงใหม่ขึ้นด้วยยูเอ็มแอล (UML: Unified Modeling Language) อันเป็นโมเดลมาตรฐานที่ใช้ในการออกแบบเชิงวัตถุ เพื่อให้ชิ้นงานเป็นรูปร่างมากขึ้น โดยในโมเดลยูเอ็มแอล จะประกอบด้วยไดอะแกรมต่างๆ ดังนี้

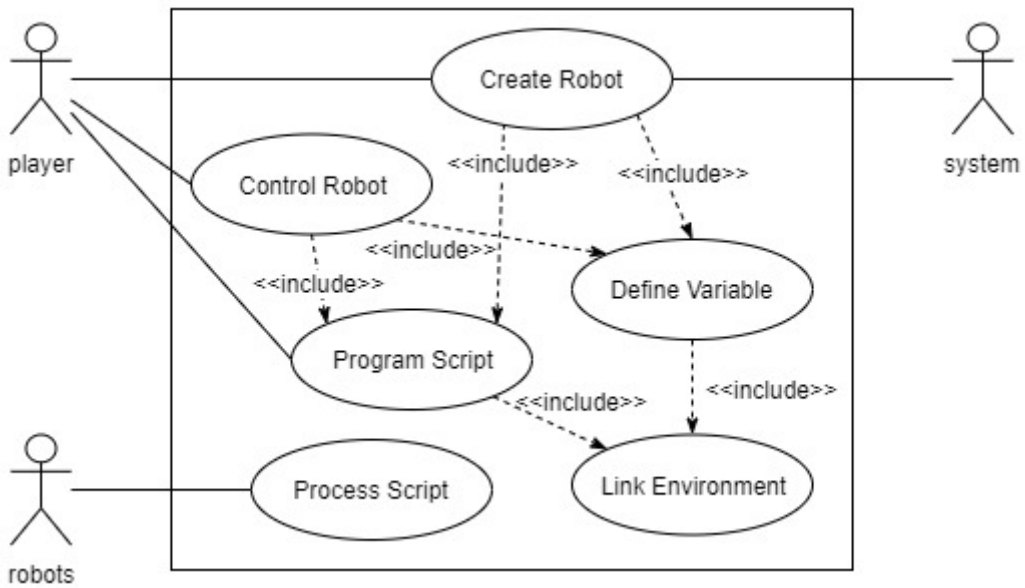
3.2.1 แผนภาพยูสเคส

แผนภาพยูสเคส (Use Case Diagram) สำหรับแสดงฟังก์ชันการทำงานของระบบ ผู้เกี่ยวข้อง รวมถึงความสัมพันธ์กับฟังก์ชันอื่นๆในระบบด้วย โดยมีสัญลักษณ์ต่างๆแสดงได้ดังตารางที่ 3.4 ดังนี้

ตารางที่ 3.4 สัญลักษณ์และความหมายของสัญลักษณ์ที่ใช้ในแผนภาพยูสเคส

รูปสัญลักษณ์	ชื่อสัญลักษณ์	ความหมาย
	Use Case	งานหรือกระบวนการที่เกิดขึ้นกับระบบ
	Actor	เชื่อมกับยูสเคสแสดงถึงบทบาทที่มีส่วนเกี่ยวข้องกับยูสเคสนั้นๆ
	Relationship	เส้นเชื่อมเพื่อแสดงการมีความสัมพันธ์ระหว่างยูสเคสกับแอคเตอร์
	Include Relationship	ความสัมพันธ์ระหว่างยูสเคสกับยูสเคส ใช้แสดงว่ายูสเคสต้นลูกศร มียูสเคสปลายลูกศรเป็นส่วนประกอบ (มีการเรียกใช้ต่อ)

แสดงได้เป็นแผนภาพ ดังนี้



รูปที่ 3.3 แผนภาพยูสเคสของเกมเอาชีวิตรอดหุ่นยนต์

จากรูปที่ 3.3 สามารถอธิบายยูสเคส ได้ดังนี้

ตารางที่ 3.5 Use Case Specification ของ Create Robot

Use Case Id	**UC01 - Create Robot**
Scenarios	ผู้เล่นหรือระบบต้องการสร้างหุ่นยนต์ตัวใหม่ขึ้นมาในแผนที่
Trigger Event	ผู้เล่นเลือกสร้างหุ่นยนต์ หรือ เมื่อระบบถึงกำหนดเวลา
Brief Description	ทำการสร้างหุ่นยนต์ตัวใหม่ขึ้นมา โดยระบุชิ้นส่วนและสคริปต์ที่ใช้
Actors	ผู้เล่น, ระบบ
Related Use Case	UC03 - Program Script UC04 - Define Variable
Precondition	ผู้เล่นไม่ได้อยู่ในเมนูอื่นๆ หรือระบบถึงกำหนดเวลา
Postcondition	สร้างหุ่นยนต์ตัวใหม่ขึ้นมาในแผนที่
Flow of Events	1. ทำการระบุชิ้นส่วนที่ใช้ 2. ทำการเลือกสคริปต์ หรือสร้างขึ้นมาใหม่ UC03 - Program Script 3. ทำการกำหนดค่าต่างๆให้ตัวแปร UC04 - Define Variable 4. สร้างหุ่นยนต์ตามคุณสมบัติที่ระบุขึ้นมาในแผนที่
Exception Condition	-

ตารางที่ 3.6 Use Case Specification ของ Control Robot

Use Case Id	**UC02 - Control Robot**
Scenarios	ผู้เล่นต้องการควบคุมหรือแก้ไขหุ่นยนต์ที่เลือก
Trigger Event	ผู้เล่นเลือกหุ่นยนต์เพื่อควบคุม
Brief Description	ทำการเลือกหุ่นยนต์ เพื่อแก้ไขคุณสมบัติต่างๆ ของหุ่นยนต์ใหม่ได้
Actors	ผู้เล่น
Related Use Case	UC03 - Program Script UC04 - Define Variable
Precondition	ผู้เล่นเลือกหุ่นยนต์ ที่เป็นของฝั่งผู้เล่น
Postcondition	หุ่นยนต์ที่ถูกแก้ไขคุณสมบัติ
Flow of Events	1. ทำการเลือกหุ่นยนต์เป้าหมาย 2. ทำการเลือกสคริปต์ หรือสร้างขึ้นมาใหม่ UC03 - Program Script 3. ทำการกำหนดค่าต่างๆ ให้ตัวแปร UC04 - Define Variable 4. บันทึกคุณสมบัติใหม่ให้แก่หุ่นยนต์
Exception Condition	ผู้เล่นทำการยกเลิกการแก้ไข

ตารางที่ 3.7 Use Case Specification ของ Program Script

Use Case Id	**UC03 - Program Script**
Scenarios	ผู้เล่นต้องการแก้ไขหรือสร้างสคริปต์คำสั่งใหม่
Trigger Event	ผู้เล่นเลือกแก้ไข หรือสร้างใหม่ ในเมนูสคริปต์คำสั่ง
Brief Description	ทำการเลือกสคริปต์คำสั่งเพื่อเข้าไปทำการแก้ไข
Actors	ผู้เล่น
Related Use Case	UC01 - Create Robot UC02 - Control Robot UC05 - Link Environment
Precondition	อยู่ในหน้าต่างการสร้างหุ่นยนต์ใหม่ ควบคุมหุ่นยนต์ หรือแก้ไขสคริปต์
Postcondition	สคริปต์ที่ถูกแก้ไขใหม่
Flow of Events	1. ทำการเลือกสคริปต์เป้าหมาย 2. ทำการเลือกเหตุการณ์ หรือสร้างเหตุการใหม่ 3. ระบุเงื่อนไข และคำสั่งของเหตุการณ์ และเป้าหมายโดยอาจมีการระบุถึงสิ่งแวดล้อมภายนอก UC05 - Link Environment 4. วนซ้ำขั้นตอนสองจนกว่าจะเขียนหมด 5. บันทึกสคริปต์
Exception Condition	ผู้เล่นทำการยกเลิกการแก้ไข

ตารางที่ 3.8 Use Case Specification ของ Define Variable

Use Case Id	**UC04 - Define Variable**
Scenarios	ผู้เล่นต้องการสร้างหรือกำหนดค่าของตัวแปรใหม่
Trigger Event	ผู้เล่นเลือกแก้ไขตัวแปร
Brief Description	ผู้เล่นสร้างหรือกำหนดค่าของตัวแปรใหม่ให้แก่หุ่นนต์ที่เลือก
Actors	-
Related Use Case	UC01 - Create Robot UC02 - Control Robot UC05 - Link Environment
Precondition	ผู้เล่นเลือกหุ่นยนต์ที่ต้องการแก้ไขตัวแปรแล้ว
Postcondition	ตัวแปรของหุ่นยนต์ที่ถูกแก้ไขใหม่
Flow of Events	1. ทำการเลือกหุ่นยนต์เป้าหมาย 2. ทำการระบุค่าให้ตัวแปรเก่าหรือใหม่โดย UC05 - Link Environment 3. ทำซ้ำขั้นตอน2จนหมด 4. บันทึกการแก้ไข
Exception Condition	ผู้เล่นทำการยกเลิกการแก้ไข

ตารางที่ 3.9 Use Case Specification ของ Link Environment

Use Case Id	**UC05 - Link Environment**
Scenarios	ผู้เล่นต้องการลิงก์ค่าของเป้าหมายไปยังสภาพแวดล้อมภายนอกหุ่นยนต์
Trigger Event	ผู้เล่นทำการกำหนดค่าตัวแปรหรือเป้าหมายของเหตุการณ์
Brief Description	ทำการเลือกสภาพแวดล้อมภายนอก พร้อมกับปรับแต่งฟิลเตอร์ เพื่อเป็นค่ากำหนดไปยังตัวแปรหรือเหตุการณ์
Actors	ผู้เล่น
Related Use Case	UC03 - Program Script UC04 - Define Variable
Precondition	ผู้เล่นเลือกหุ่นยนต์ที่ต้องการแก้ไขตัวแปรแล้ว
Postcondition	คืนค่าเป้าหมายไปยังยูสเคสที่เรียกใช้งาน
Flow of Events	1. ทำการเลือกสิ่งแวดล้อมเป้าหมาย 2. กำหนดฟิลเตอร์ลำดับความสำคัญ 3. ปรับแต่งฟิลเตอร์การกรองเพิ่มเติม 4. ตกลงเพื่อกำหนดค่า
Exception Condition	ผู้เล่นทำการยกเลิก

ตารางที่ 3.10 Use Case Specification ของ Process Script

Use Case Id	**UC06 - Process Script**
Scenarios	หุ่นยนต์ทำการประมวลผลสคริปต์คำสั่งของตัวเองเพื่อแปรออกมาเป็นการทำงาน
Trigger Event	ทำงานโดยอัตโนมัติตลอดการเล่น
Brief Description	หุ่นยนต์ทำการอ่านสคริปต์คำสั่งของตนเองเพื่อหาเหตุการณ์ที่ตรงเงื่อนไขและทำตามคำสั่งของเหตุการณ์นั้นไปเรื่อยๆ จนกว่าจะเข้าเงื่อนไขอื่น
Actors	หุ่นยนต์
Related Use Case	-
Precondition	หุ่นยนต์ไม่ได้อยู่ระหว่างการควบคุมแก้ไขคุณสมบัติ
Postcondition	ทำการกระทำตามคำสั่งของเหตุการณ์ที่เข้าเงื่อนไข
Flow of Events	1. อ่านเหตุการณ์ในสคริปต์คำสั่ง 2. หากเงื่อนไขไม่เป็นจริงให้ไปทำข้อ1ใหม่ด้วยเหตุการณ์ที่อยู่ลำดับความสำคัญถัดไป 3. หากไม่สามารถทำตามคำสั่งได้ให้ไปทำข้อ1ใหม่ด้วยเหตุการณ์ที่อยู่ลำดับความสำคัญถัดไป 4. ทำตามคำสั่งของเหตุการณ์ที่ระบุ 5. ทำการวนซ้ำข้อ1หาเหตุการณ์ทำต่อไปเรื่อยๆจนครบจำนวนเทรดที่หน่วยประมวลผลของตนมี
Exception Condition	หาคำสั่งจนเหตุการณ์ที่สคริปต์มาหมด

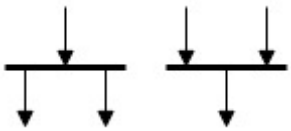
3.3 การวิเคราะห์พฤติกรรมของระบบ

เป็นส่วนหนึ่งของการอธิบายการทำงาน เพื่อให้เข้าใจถึงลำดับขั้นตอนและลักษณะการทำงานโดยละเอียด โดยอธิบายผ่านแผนภาพกิจกรรม (Activity Diagram) ที่อธิบายลำดับของกิจกรรมทั้งหมด และแผนภาพซีเควนซ์ (Sequence Diagram) เพื่ออธิบายลำดับการทำงานของแต่ละยูสเคสอีกที

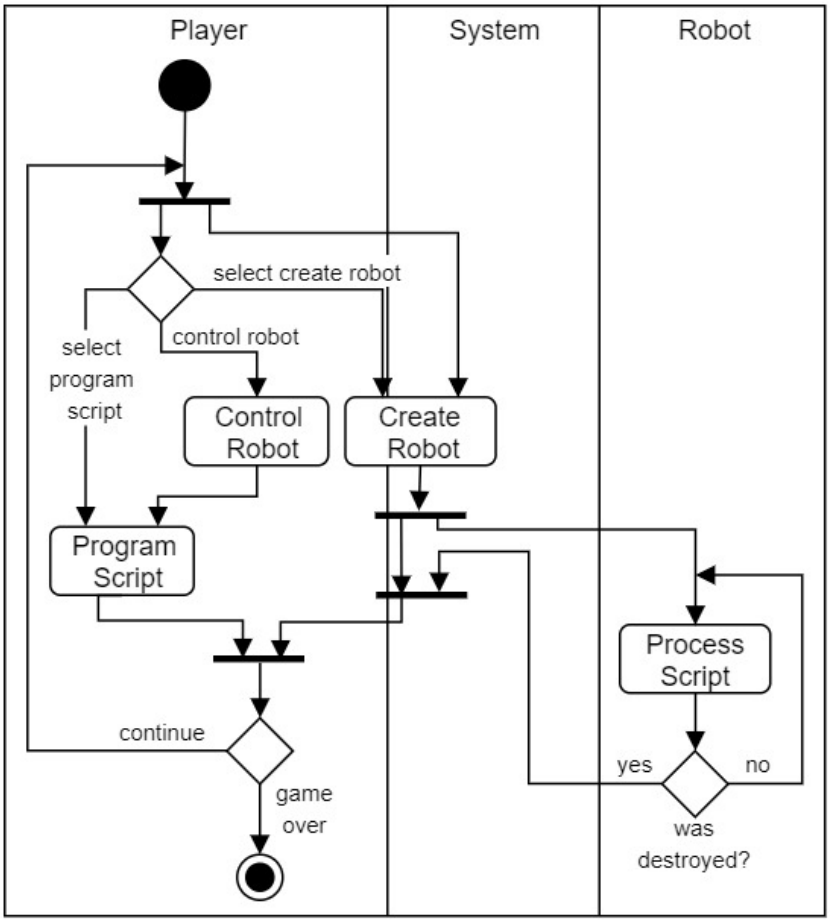
3.3.1 แผนภาพกิจกรรม (Activity Diagram)

แสดงภาพรวมของกิจกรรมทั้งหมดที่เกิดขึ้นในระบบรวมถึงบทบาทที่มีส่วนเกี่ยวข้อง ผ่านลำดับและเงื่อนไขในการทำงาน แสดงได้ด้วยสัญลักษณ์ดังตารางที่ 3.11

ตารางที่ 3.11 สัญลักษณ์และความหมายของสัญลักษณ์ที่ใช้ในแผนภาพกิจกรรม

รูปสัญลักษณ์	ชื่อสัญลักษณ์	ความหมาย
	Start	จุดเริ่มต้นของระบบ
	Terminate	จุดสิ้นสุดของระบบ
	Activity	กิจกรรม
	Control	แสดง flow ของระบบ
	Fork and Join	สัญลักษณ์ประสาน แยกและรวม flow เพื่อแสดงการทำงานของกิจกรรมที่เกิดขึ้นพร้อมๆกัน
	Condition	กรอบเงื่อนไข ให้สามารถเลือกทิศทางต่อไปของกิจกรรม

แสดงได้เป็นแผนภาพ ดังนี้



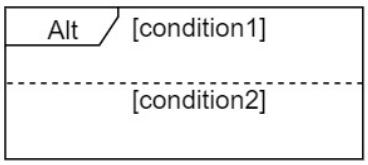
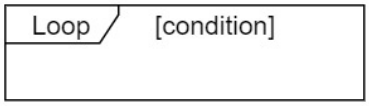
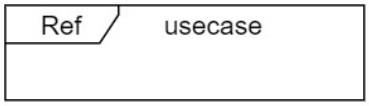
รูปที่ 3.4 แผนภาพกิจกรรมของเกมเอาชีวิตรอดหุ่นยนต์

เมื่อผู้เล่นเริ่มเกม การทำงานจะถูกแบ่งเป็นสองฝั่งคือฝั่งระบบที่จะคอย create robot ศัตรูออกมาโจมตีผู้เล่นเรื่อยๆ และฝั่งผู้เล่นที่จะเลือกการกระทำได้3อย่างคือ program script, control robot และ create robot การ program script คือการเข้าไปแก้ไขสคริปต์ที่มีอยู่แล้วโดยตรง ส่วน control robot คือการเข้าควบคุมหรือแก้ไขคุณลักษณของหุ่นยนต์ที่เลือก ซึ่งส่วนนี้สามารถ program script เพื่อแก้ไขคำสั่งของหุ่นยนต์ที่เลือกต่อได้ คำสั่งต่อมาที่ทั้งระบบและผู้เล่นทำได้คือ create robot สร้างหุ่นยนต์ขึ้นมาใหม่ในแผนที่ แล้วหุ่นยนต์ตัวนั้นจะ process script คำสั่งของตัวเองไปเรื่อยๆจนกว่าจะถูกทำลาย โดยคำสั่งทั้งหมดเหล่านี้จะทำงานวนไปเรื่อยๆจน game over (ผู้เล่นพลังชีวิตหมด) แล้วจบการทำงาน

3.3.2 แผนภาพซีเควนซ์ (Sequence Diagram)

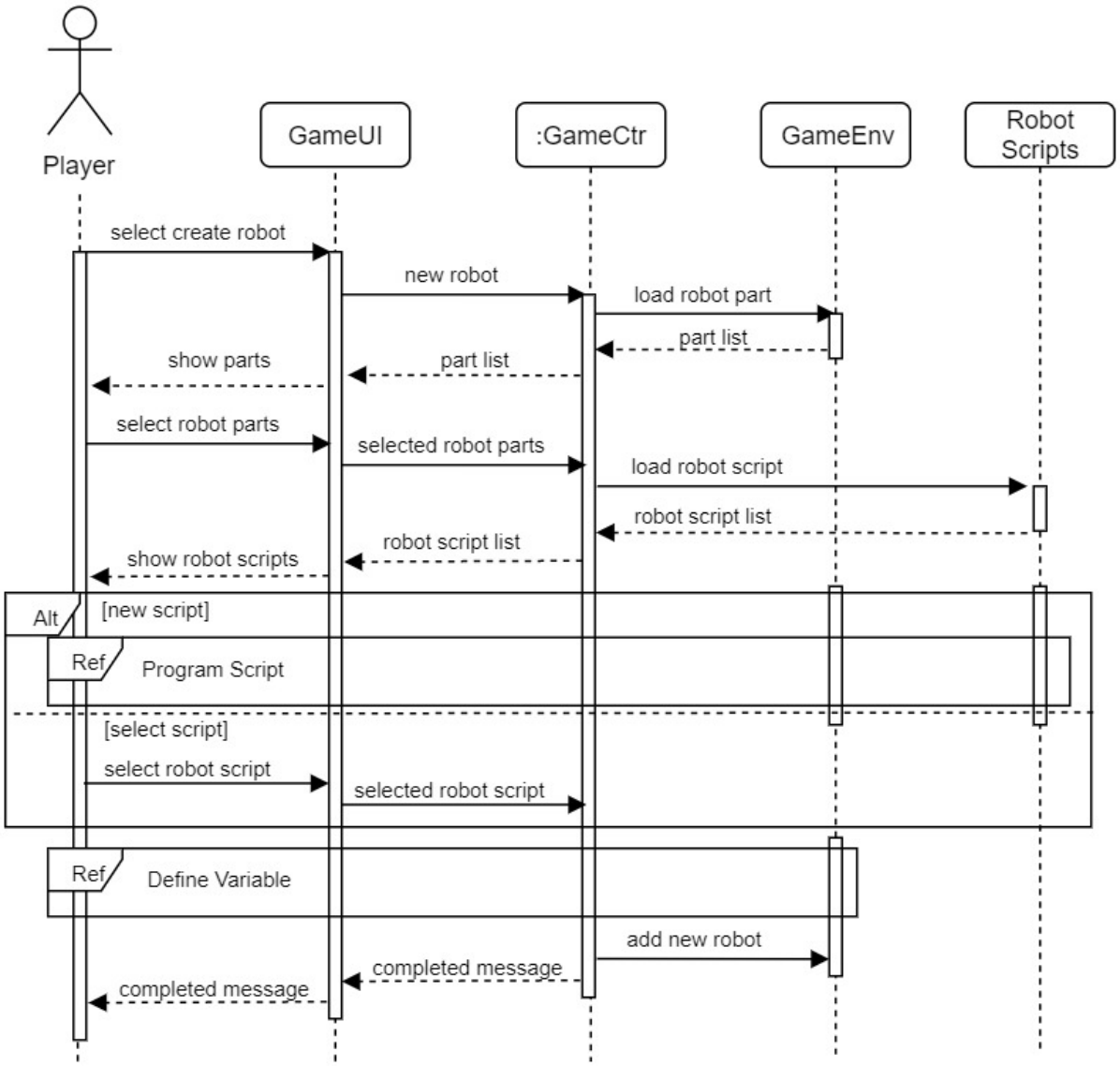
แผนภาพแสดงถึงการปฏิสัมพันธ์ระหว่างบทบาทต่างๆและออบเจ็กต์ภายในแต่ละยูสเคส โดยคำนึงถึงลำดับเวลาการทำงานเป็นสำคัญ แสดงโดยสัญลักษณ์ได้ดังตารางที่ 3.12

ตารางที่ 3.12 สัญลักษณ์และความหมายของสัญลักษณ์ที่ใช้ในแผนภาพซีเควนซ์

รูปสัญลักษณ์	ชื่อสัญลักษณ์	ความหมาย
	Object Class	คลาสหรือออบเจ็กต์ที่มีบทบาทในการตอบสนองต่อการทำงานในส่วนนี้
	Lifeline	เส้นชีวิต เป็นแกนในการแสดงเวลาของแต่ละคลาสหรือออบเจ็กต์
	Focus of control / Activation	การมีอยู่บนเส้นชีวิต แสดงช่วงเวลาของคลาสที่มี กิจกรรมเกิดขึ้นในช่วงชีวิต
	Message	เส้นแสดงการส่งสัญญาณระหว่างวัตถุ พร้อมทั้งระบุคำอธิบายสัญญาณกำกับ
	Callback / Self Delegation	การประมวลผล และมีการคืนค่าผลลัพธ์เกิดขึ้นแค่ภายในตัวเอง
	Condition	กรอบทางเลือกทำเพียงช่องใดช่องหนึ่งที่ตรงตามเงื่อนไข
	Loop	กล่องแสดงถึงการทำซ้ำซีเควนซ์ที่อยู่ภายในเรื่อยๆจนกว่าจะหลุดจาก condition
	Reference Fragment	กล่องแสดงถึงการเรียกใช้งานยูสเคสอื่นที่มีการ include

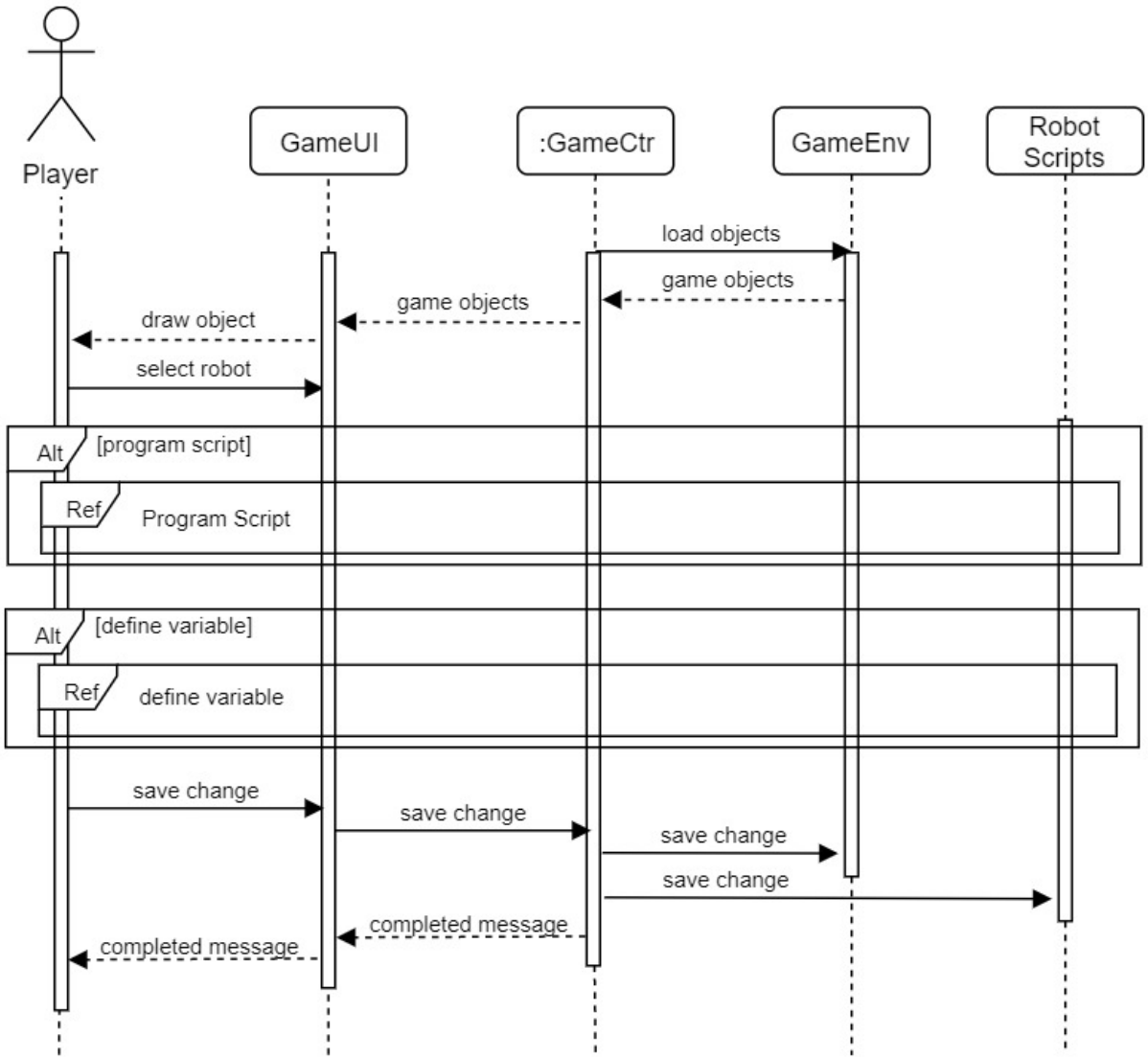
แสดงได้เป็นแผนภาพ ดังนี้

1) Sequence Diagram ของ Create Robot



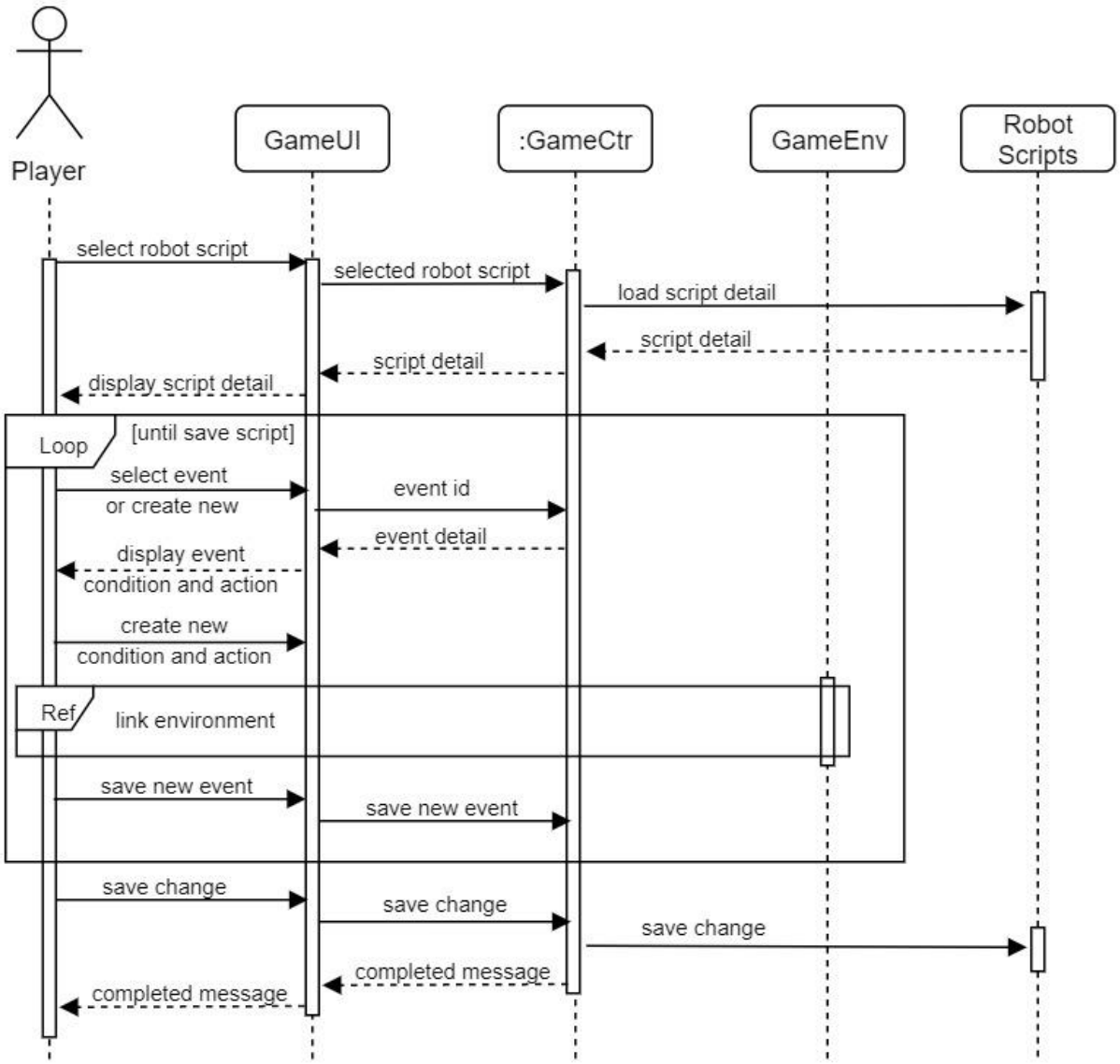
รูปที่ 3.5 Sequence Diagram ของ Create Robot

2) Sequence Diagram ของ Control Robot



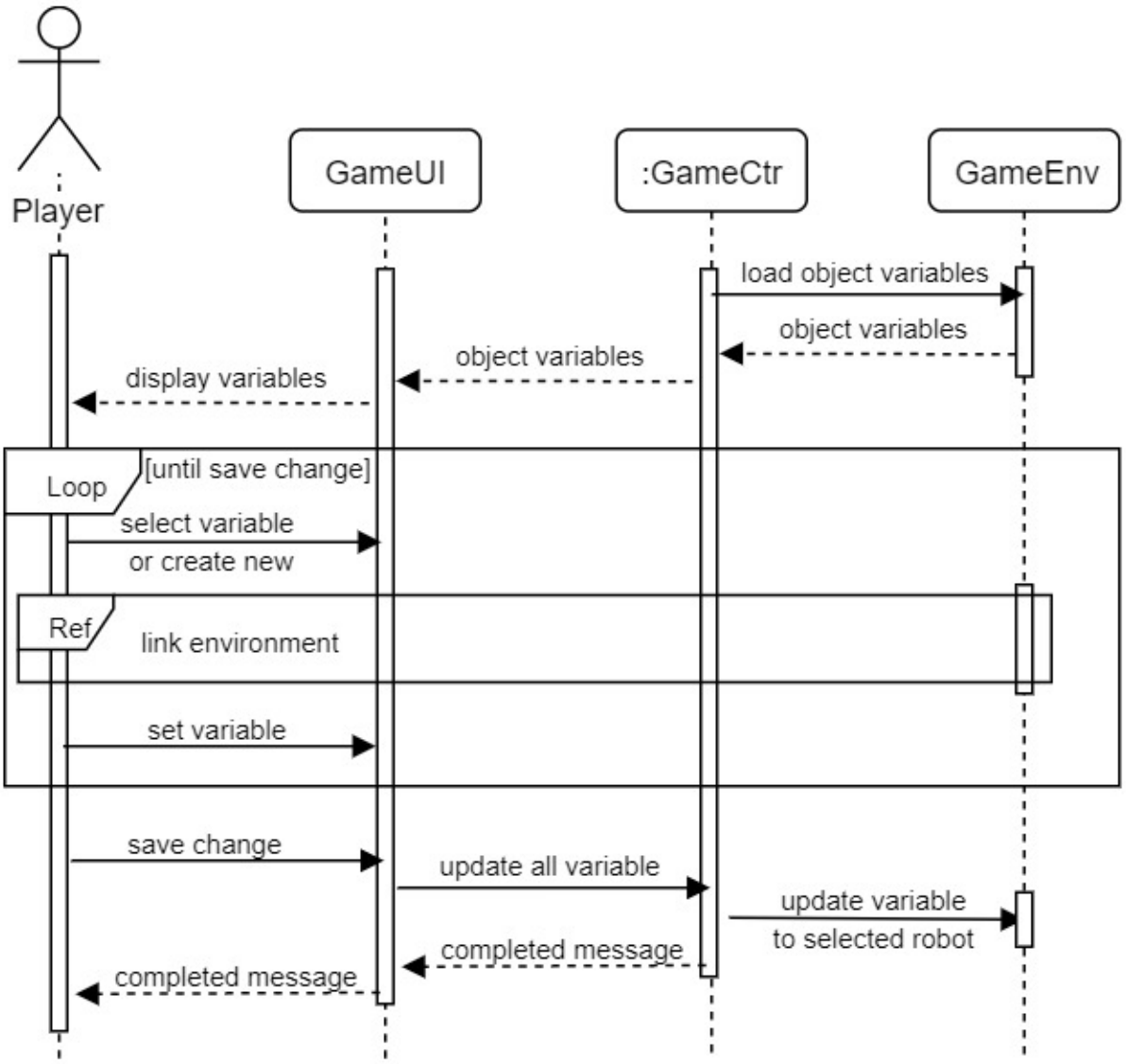
รูปที่ 3.6 Sequence Diagram ของ Control Robot

3) Sequence Diagram ของ Program Script



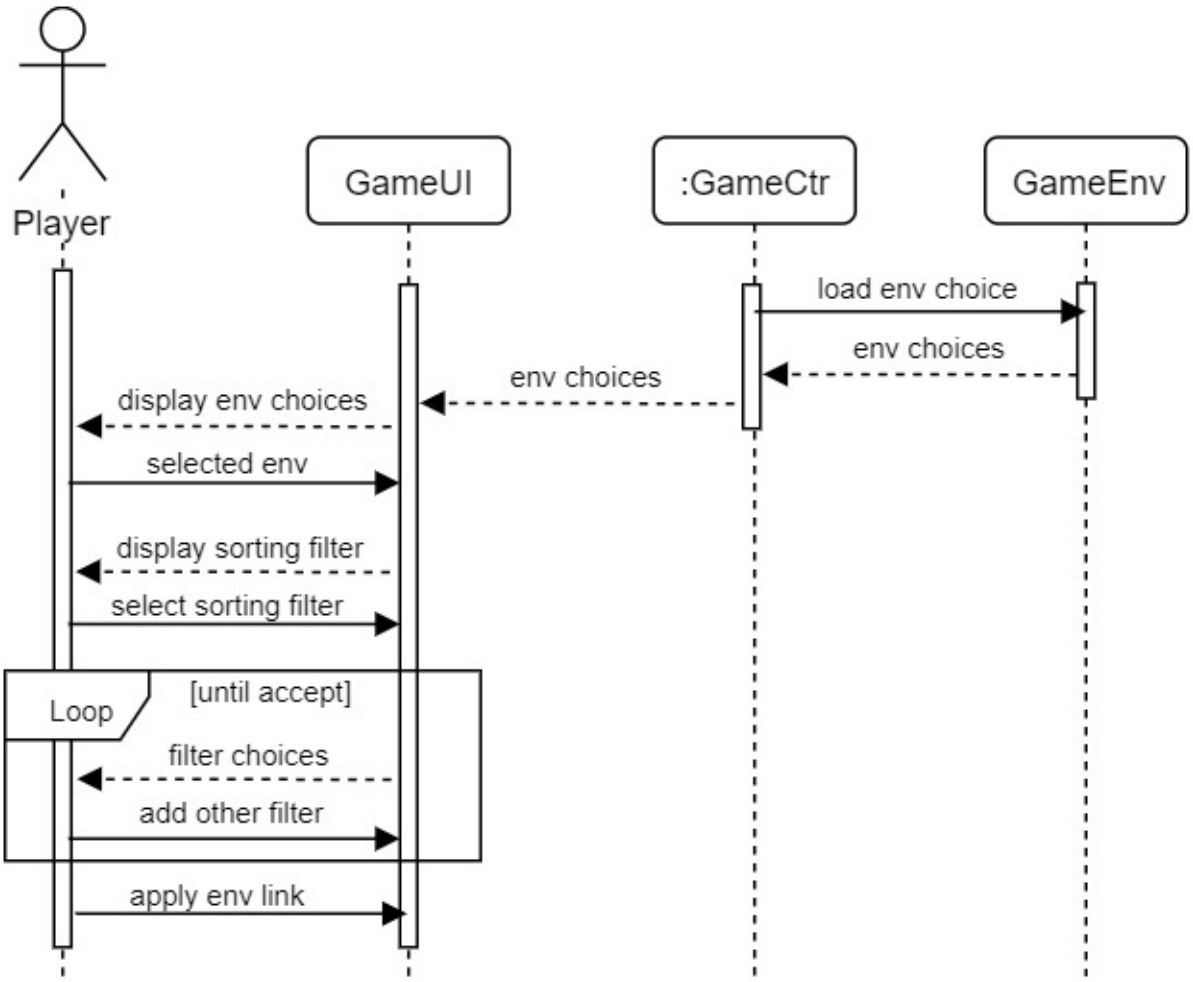
รูปที่ 3.7 Sequence Diagram ของ Program Script

4) Sequence Diagram ของ Define Variable



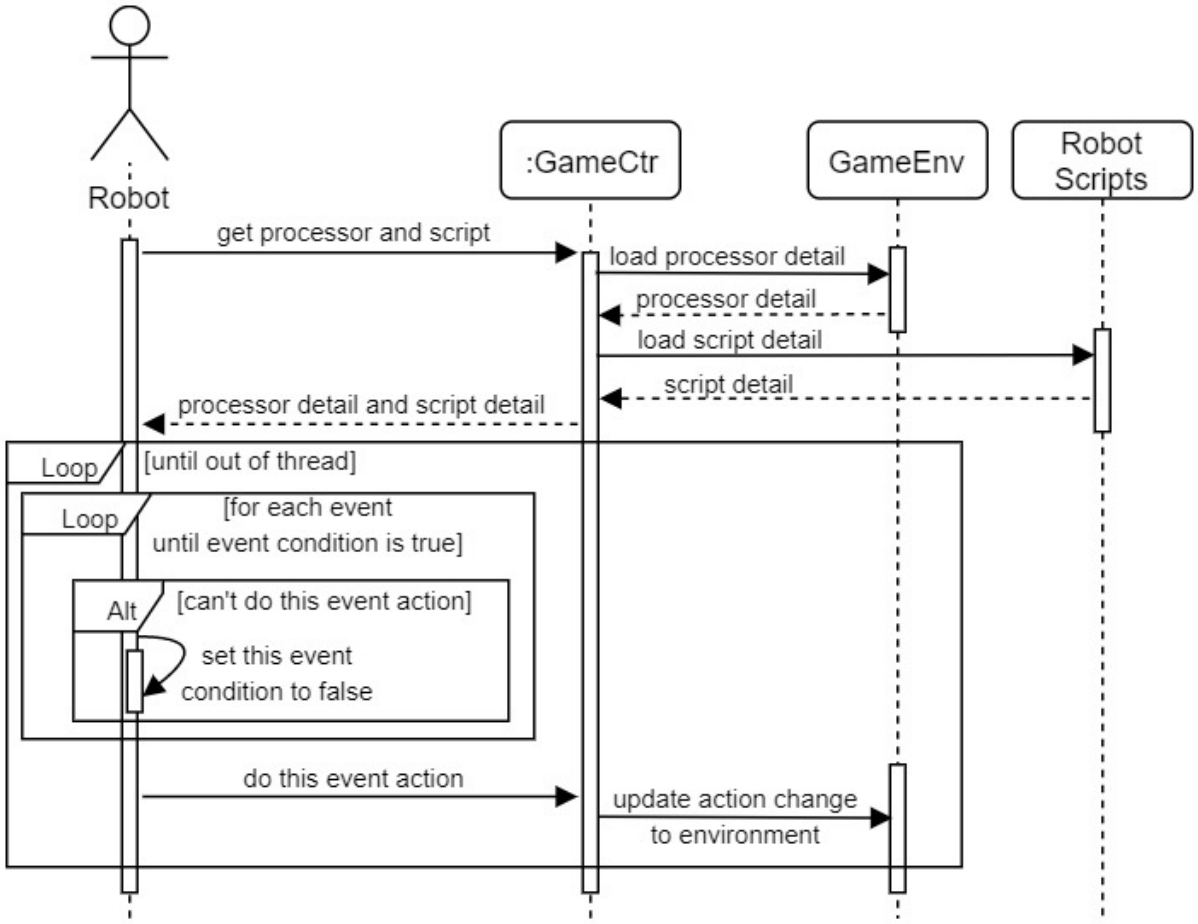
รูปที่ 3.8 Sequence Diagram ของ Define Variable

5) Sequence Diagram ของ Link Environment



รูปที่ 3.9 Sequence Diagram ของ Link Environment

6) Sequence Diagram ของ Process Script



รูปที่ 3.10 Sequence Diagram ของ Process Script

บทที่ 4

การออกแบบจอภาพ

เป็นส่วนสำคัญเพราะส่งผลต่อความประทับใจแรกของผู้เล่น ทั้งความสวยงาม ความง่ายต่อการทำความเข้าใจและใช้งาน บทนี้จึงได้แสดงถึงการวางแผนออกแบบให้เห็นถึงความหมายถึงความหมายของสิ่งต่างๆบนหน้าจอ และความสัมพันธ์ระหว่างหน้าต่างเมนูหน้าอื่นๆ อีกด้วย

4.1 หน้าต่างเกม

รูปที่ 4.1 หน้าต่างหลักของการเล่นเกม

จากรูปที่ 4.1 อธิบายรายละเอียดของภาพได้ดังนี้

ส่วนที่ 1 หุ่นยนต์ของฝั่งผู้เล่น สามารถดูได้จากการมีสีขาวเป็นส่วนใหญ่
สามารถเปิดหน้าต่างของหุ่นยนต์ได้จากการกดดับเบิลคลิกหุ่นยนต์ฝั่งตัวเองที่ต้องการในหน้าต่างนี้

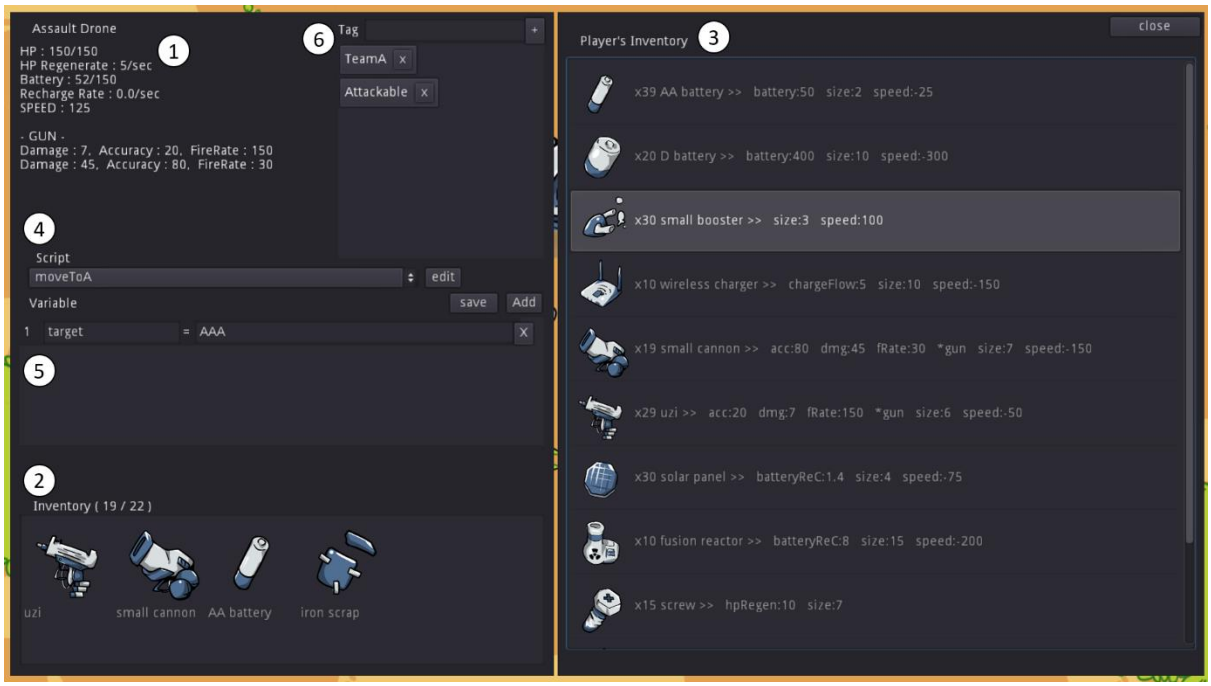
ส่วนที่ 2 หลอดพลังชีวิตของหุ่นยนต์แต่ละตัว จะไม่แสดงเมื่อพลังชีวิตเต็ม

ส่วนที่ 3 หุ่นยนต์ฝั่งศัตรูที่เข้ามาโจมตีผู้เล่น หุ่นยนต์ฝั่งผู้เล่นจะมีสีดำเป็นส่วนใหญ่

ส่วนที่ 4 ปุ่มเมนูในการเลือกแก้ไขสคริปต์คำสั่ง กดแล้วจะเปิดหน้าต่าง Program Script

4.2 หน้าต่างหุ่นยนต์

เป็นหน้าต่างที่จะเปิดขึ้นมาเมื่อผู้เล่นกดดับเบิลคลิก เพื่อเลือกชิ้นส่วนที่ต้องการติดตั้งหรือถอนการติดตั้ง



รูปที่ 4.2 หน้าต่างหุ่นยนต์

ส่วนที่ 1 สถานะต่างๆ ของหุ่นยนต์ ไม่ว่าจะเป็น ชื่อ, พลังชีวิต, แบตเตอรี่, ความเร็ว และปืนที่ติดตั้งอยู่

ส่วนที่ 2 ส่วนแก้ไขชิ้นส่วนหุ่นยนต์ สามารถ ดู ถอด และติดตั้งชิ้นส่วนใหม่ได้จากส่วนนี้ โดยจำนวนพื้นที่ที่สามารถติดตั้งชิ้นส่วนได้จะขึ้นกับชนิดของ case ที่หุ่นยนต์ติดตั้งอยู่

ส่วนที่ 3 ชิ้นส่วนที่เราเหลืออยู่

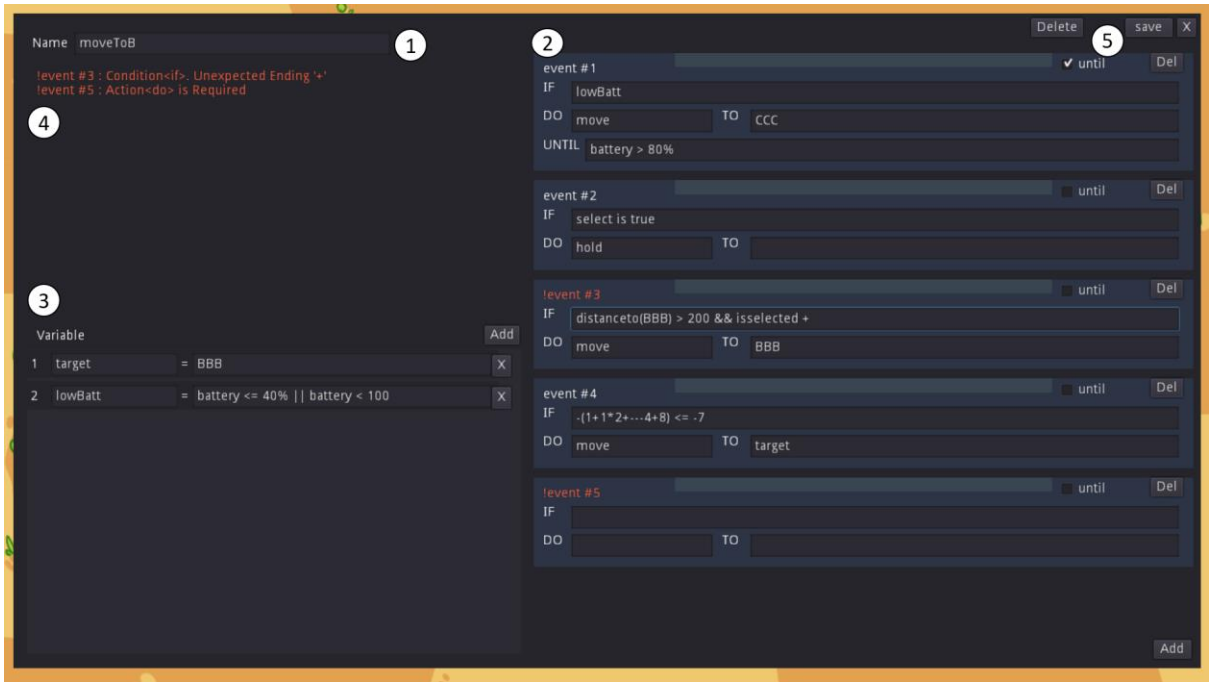
ส่วนที่ 4 ส่วนการติดตั้งสคริปต์ของหุ่นต์ตัวนี้ ผู้เล่นสามารถเลือกสคริปต์ที่มีอยู่แล้ว แก้ไข หรือสร้างขึ้นใหม่ได้ โดยจะเปิดหน้าต่าง program script ได้จากการกดedit

ส่วนที่ 5 ส่วนตัวแปรที่ใช้ของหุ่นยนต์ ตัวแปรเหล่านี้เป็นเพียงชื่อที่ผู้เล่นตั้งขึ้นเองแต่ความหมายจะขึ้นกับค่าที่ผู้เล่นกำหนด ตัวอย่างเช่นการกำหนดให้หุ่นยนต์มีเป้าหมายหลักให้ไปยังป้ายAAA

ส่วนที่ 6 ส่วนการTag หรือกรุ๊ปให้หุ่นยนต์ เป็นการระบุว่าหุ่นยนต์ตัวนี้จัดเข้าไปไว้ในประเภทไหนบ้าง จะได้กล่าวถึงได้ผ่านคำสั่งต่างๆ ในภายหลัง

4.3 หน้าต่างแก้ไขสคริปต์

เป็นหน้าต่างที่จะเปิดขึ้นมาเมื่อผู้เล่นเลือกสคริปต์แล้วเข้าสู่การแก้ไขจากหน้าต่างอื่น



รูปที่ 4.3 หน้าต่างแก้ไขสคริปต์

จากรูปที่ 4.3 อธิบายรายละเอียดของภาพได้ดังนี้

ส่วนที่ 1 ชื่อของสคริปต์คำสั่งนี้

ส่วนที่ 2 กล่องเหตุการณ์ต่างๆ โดยเรียงลำดับความสำคัญจากบนลงล่าง

ส่วนที่ 3 หน้าต่างตัวแปร ใช้เพื่อระบุความหมายอะไรบางอย่างเพิ่ม เพื่อความอ่านง่าย เป็นระเบียบ

ส่วนที่ 4 ส่วนแสดง error ของข้อมูลที่กรอกอยู่ ว่ามีการผิดหลักไวยกรณ์ที่รับได้หรือไม่

ส่วนที่ 5 ปุ่มบันทึกเหตุการณ์ จะไม่สามารถบันทึกได้หาก input ต่างๆ ยังมีที่ผิดหลักอยู่

4.4 หน้าต่างจบเกม

เป็นหน้าต่างสรุปผลที่จะปรากฏขึ้นมาแสดงให้ผู้เล่น เมื่อผู้เล่นเกมโอเวอร์

รูปที่ 4.4 หน้าต่างจบเกม

หน้าต่างจบเกม แสดงขึ้นเมื่อหุ่นยนต์หลักของผู้เล่นถูกทำลาย

พร้อมกับคะแนนที่ผู้เล่นทำได้ในรอบนี้

บทที่ 5

การพัฒนาระบบ

5.1 ภาษาและเครื่องมือที่ใช้ในการพัฒนา

5.1.1 ภาษาที่ใช้ในการพัฒนาระบบ

GDScript เป็นภาษา Script ที่ built-in มากับโปรแกรม Godot Engine เป็นภาษาระดับสูงที่มีความคล้ายคลึงกับภาษา Python ทำงานแบบ Dynamic ทำให้เก็บวัตถุชนิดต่างๆ กัน ไว้ใน Array หรือ Dictionary เดียวกันได้โดยไม่ต้องมีการระบุประเภทของตัวแปรก่อน จึงมีความง่ายเป็นอย่างมากเมื่อนำมาใช้กับเกมที่มีการกล่าวถึงกลุ่มวัตถุใดก็ได้ ที่อาจมีประเภทที่แตกต่างกัน แล้วนำมาดำเนินงานร่วมกัน โดยไม่ต้องมาคอยทำ Interface class เหมือนภาษาที่ต้องมีการประกาศชนิดของตัวแปรก่อนเสมอ ภาษานี้ถูกพัฒนามาควบคู่กับ Godot Engine

5.1.2 เครื่องมือที่ใช้ในการพัฒนาระบบ

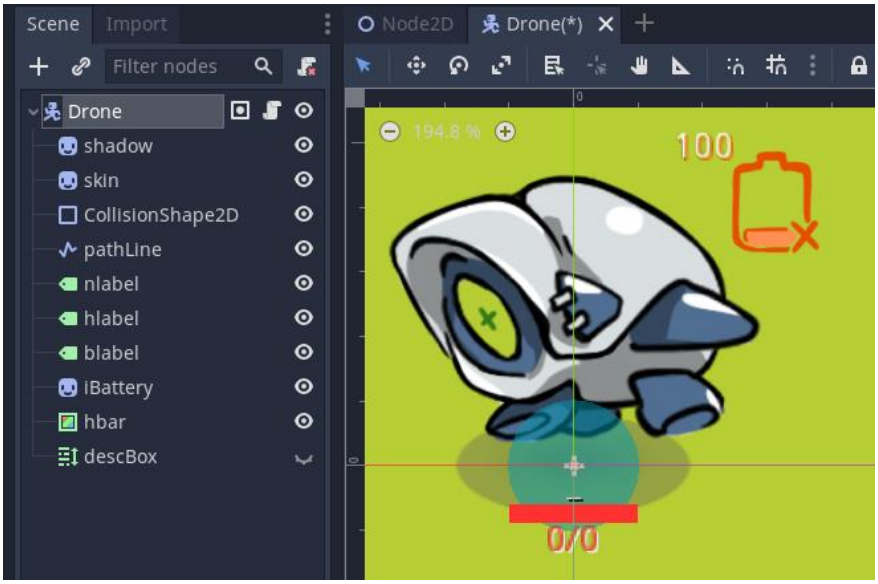
Godot Engine เป็น Game Engine ฟรีและ open-source ที่อาศัยแหล่งรายได้จากการบริจาคเป็นหลัก รองรับการพัฒนาเกมได้กับทั้ง2มิติและ3มิติ เกมที่พัฒนาสามารถ export ออกมาได้ในหลาย platform ทั้ง PC, Moblie และ Website

รูปที่ 5.1 logo ของ Godot Engine

5.2 เทคนิคที่ใช้ในการพัฒนา

5.2.1 Scene และการทำ Instance

ในโปรแกรม Godot เราจะมองวัตถุต่างๆเป็นโหนดๆหนึ่ง ที่สามารถเชื่อมต่อกับวัตถุต่างๆ เป็นโหนดลูกต่อๆไปได้จนเป็น tree ซึ่งโหนดเหล่านั้นก็จะมีจำนวนตามที่เราสร้างผ่าน tree ของโปรแกรม ถ้าเราอยากให้โหนดเหล่านั้นเป็นวัตถุที่สร้างใหม่ซ้ำๆ ได้ เราจะต้องทำการแปลงโหนดนั้นทั้งโหนดเป็น scene หนึ่งๆก่อน จากนั้นจึงทำการ import scene เหล่านั้นเข้ามาผ่านโค้ดคำสั่ง เพื่อที่จะทำให้สามารถสร้าง instance ของวัตถุขึ้นมาใหม่ในระหว่างรันเกมได้เรื่อยๆ



รูปที่ 5.2 ตัวอย่างของโหนด Drone ที่ถูกแปลงกลายเป็น Scene

5.2.2 การตรวจ Grammar และการทำ Parsing

เนื่องจากไม่ต้องการให้ผู้เล่นป้อนโค้ดคำสั่งผิดๆให้แก่หุ่นยนต์ ในตัวเกมจึงต้องมีการตรวจสอบความถูกต้องของนิพจน์ที่ผู้เล่นป้อนเข้ามาด้วย และจะไม่ยอมให้บันทึกเด็ดขาด หากนิพจน์เหล่านั้นยังมีไวยกรณ์ภาษาที่ผิดอยู่ หรือทั้งการที่นำนิพจน์ต่างๆ มาประมวณผลได้ผลลัพธ์ถูกต้อง อย่างที่มันควรจะเป็น โดยประยุกต์เอาจากรายวิชา Compiler และวิชาแขนงนี้อื่นๆ ที่ผู้พัฒนาเคยเรียน ไม่ว่าจะเป็น

- การตรวจรูปแบบของวงเล็บที่ผิด ทั้งไม่สมบูรณ์ หรือลำดับที่ขัดกัน
- การตรวจเครื่องหมายหรือคำที่ไม่รู้จัก ทั้งนี้ยังต้องเช็คอีกด้วยในกรณ์ที่บางเครื่องหมายเกิดจากหลายๆ อักษรที่ประกอบกัน ว่าสุดท้ายแล้วถ้าประกอบกันเสร็จ จะได้เครื่องหมายที่รู้จักหรือไม่ เช่น

เครื่องหมาย **<,=** หรือ **=,=** ต่อกันได้ และถ้าหากเป็นเครื่องหมายที่รู้จัก จะหั่นนิพจน์โดยถือว่าเครื่องหมายที่ติดกันเหล่านั้นเป็นคำเดียวกัน อย่างเช่น A==- BC จะต้องหั่นคำได้ว่า A, ==, -BC

- การตรวจเช็คลำดับของคำ ว่าเป็นลำดับที่ยอมรับได้หรือไม่ โดยหลังจากที่หั่นข้อความแยกออกเป็นคำๆ ได้แล้วพร้อมระบุชนิดของคำ ก็จะตรวจว่าอะไรต่อกับอะไรได้หรือไม่ได้ เช่น operator ไม่สามารถต่อกับ operator ได้
- การลำดับความสำคัญของ operator และวงเล็บ ว่าสุดท้ายแล้วต้องทำลำดับ operator ไหนก่อนหลัง ทั้งยังต้องคำนึงถึงการทำจากซ้ายไปขวาด้วยหากลำดับความสำคัญเท่ากัน

ทั้งนี้ในช่วงแรกผู้พัฒนาใช้วิธีรันข้อความให้เปรียบเสมือนโค้ดใน GDScript แต่พบว่าคำสั่งลัดบางอย่างจะไม่สามารถทำให้เป็นไปตามที่ผู้พัฒนาอยากให้เป็นได้ ดังนั้น เพื่อคงความเขียนง่ายของคำสั่งตามที่ได้เคยออกแบบไว้ เลยเปลี่ยนเป็นวิธีการอ่านและประมวณผลเองทั้งหมด

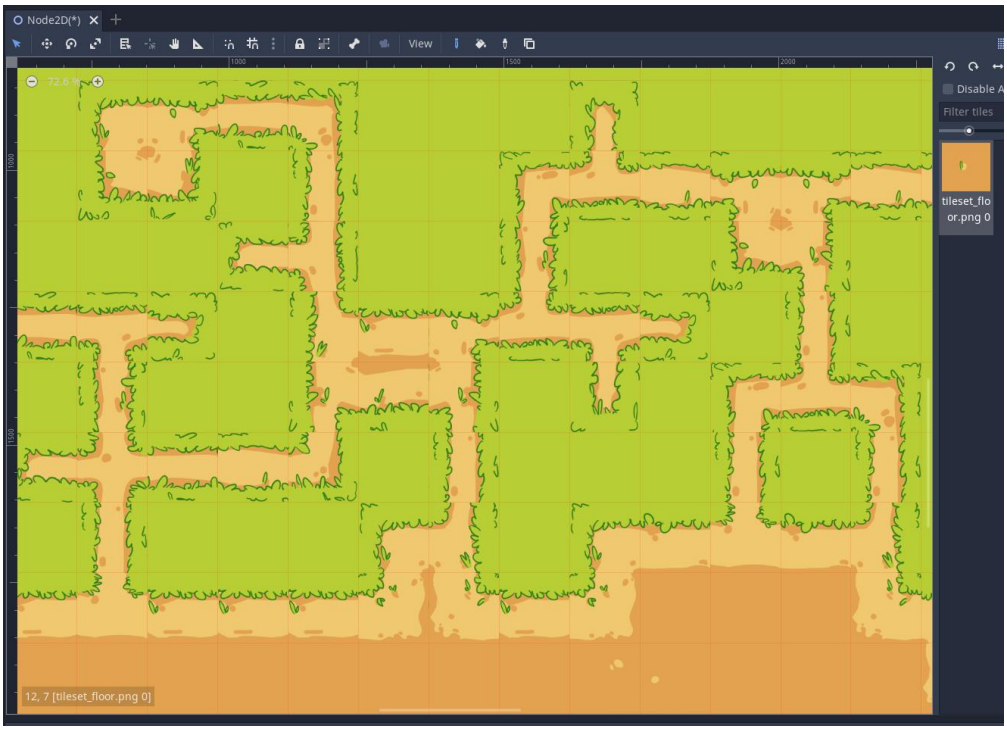
5.2.3 การ Pack Scene และการเขียนไฟล์

เพื่อให้เกมรองรับระบบบันทึกและโหลดเกม จำเป็นต้องให้ตัวเกมสามารถเขียนไฟล์เพื่อที่เวลาปิด-เปิดตัวเกมใหม่แล้วจะยังโหลดเกมใหม่ได้ แต่ปัญหาที่พบคือในการทำดังกล่าวต้องการบีบอัดข้อมูลทุกอย่างจากนั้นจึงนำไปเขียนเป็นไฟล์ได้ เนื่องจากเกมนี้มีโครงสร้างที่ใหญ่และกระจัดกระจายมาก การจะบีบอัดข้อมูลทั้งหมดเป็นปัญหาใหญ่ ทางผู้พัฒนาแก้ปัญหานี้โดยการใช้คลาส PackedScene ซึ่งคลาส PackedScene นี้ทำให้การบีบอัดโหนดและซีนต่างๆ ของเกมทำได้โดยง่าย อย่างไรก็ตาม ผู้พัฒนาก็ยังจำเป็นต้องบีบอัดข้อมูลเองในบางส่วนอยู่ เนื่องจากบางคลาสเป็นวัตถุที่ผู้พัฒนาเขียนขึ้นมาเองแล้วหาวิธีให้รองรับกับ PackedScene ของโปรแกรมไม่ได้

5.3 การพัฒนาระบบ

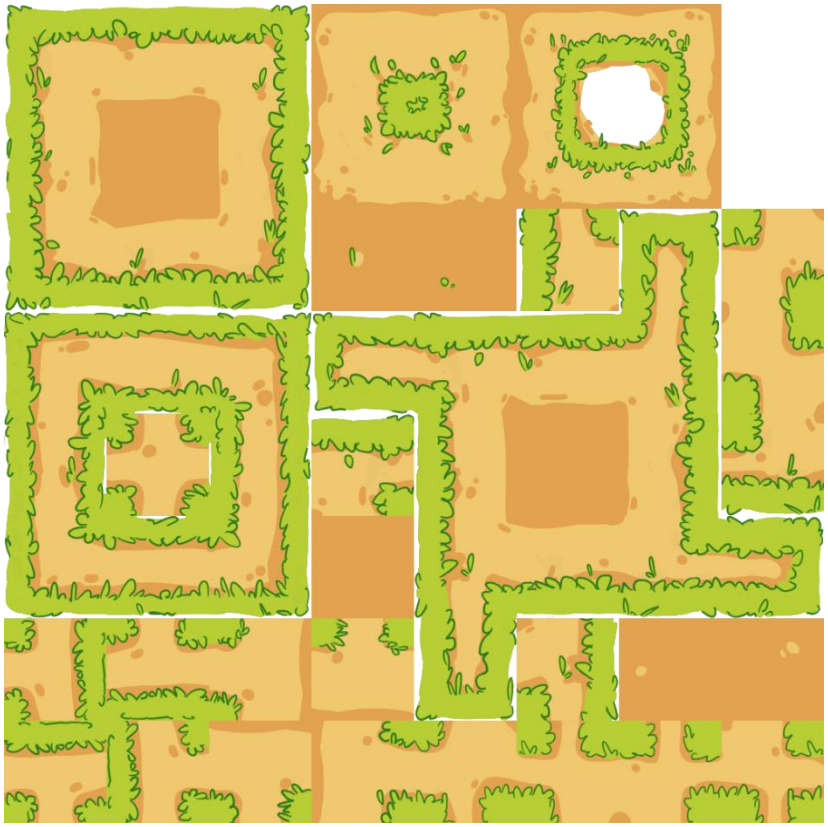
5.3.1 แผนที่ภายในเกม

เนื่องจากเป้าหมายของเกมคือการควบคุมวัตถุจำนวนมาก จึงต้องมีขนาดแผนที่ในเกมที่ใหญ่เพื่อให้สอดคล้องกันด้วย ผู้พัฒนาเลยเขียนระบบให้ทำการสุ่มสร้างแผนที่ขึ้นมาเองแทน และเพื่อให้แต่ละส่วนของแผนที่มันแสดงผลออกมาเป็นรูปลักษณ์ได้จากโค้ดที่เขียน ผู้พัฒนาจึงได้ใช้ Auto tile ที่เป็นเครื่องมือสร้างแผนที่ของโปรแกรมช่วย เพื่อให้แต่ละ tile (กระเบื้อง) มันคอยเลือกรูปที่เหมาะสมต่อกันให้เองตามที่ตั้งไว้



รูปที่ 5.3 ตัวอย่างการใช้ Auto Tile เพื่อให้ระบบคอยต่อรูปแผนที่ให้เอง

ทั้งนี้เราก็ต้องออกแบบรูปแบบกระเบื้องทั้งหมดที่เป็นไปได้ก่อน เพื่อนี่จะได้นำมาแบ่งเป็นชิ้นๆ และประกอบใหม่ในโปรแกรม



รูปที่ 5.4 รูปต้นฉบับก่อนที่จะนำมาหั่นและใช้ใน Auto Tile

5.3.2 ชิ้นส่วนหุ่นยนต์ และคุณสมบัติของหุ่นยนต์ในเกม

เพื่อที่จะให้ชิ้นส่วนต่างๆ ที่มีลักษณะการทำงานที่ต่างกัน มันทำงานร่วมกันได้ โดยไม่ต้องไปแยกโค้ดโปรแกรมเป็นสำหรับแต่ละไอเทม ผู้พัฒนาจึงออกแบบให้ทุกคุณสมบัติมันมีอยู่ในตัวหุ่นยนต์ไว้ก่อนแล้ว แล้วค่อยให้ชิ้นส่วนเหล่านั้นมันไปปรับค่าคุณสมบัติที่มีอยู่เดิมแทน และถ้าไอเทมไม่มีการกล่าวถึงค่าไหน ก็แสดงว่ามันไม่มีการปรับคุณสมบัตินั้น ดังตัวอย่าง

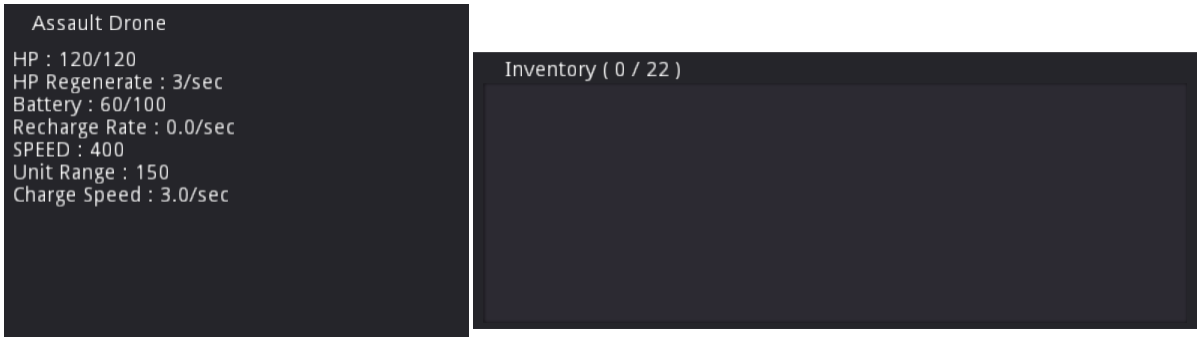
```
'screw_1':{
    "name" : "screw",
    "texture" : "res://assets/item_screw_1",
    "hpRegen" : 5,
    "size" : 7
  },
'repairkit_1':{
    "name" : "small repairkit",
    "texture" : "res://assets/item_repairkit_1",
    "healSpeed" : 5,
    "speed" : -50,
    "size" : 5
  },
```

```
'usb_1':{
    "name" : "USB3",
    "texture" : "res://assets/item_usb_1",
    "chargeSpeed" : 5,
    "size" : 4
  },
'antenna_1':{
    "name" : "old antenna",
    "texture" : "res://assets/item_antenna_1",
    "unitRange" : 50,
    "speed" : -25,
    "size" : 3
  },
```

รูปที่ 5.5 ตัวอย่างของการกำหนด value ของชิ้นส่วนติดตั้งต่างๆ ภายในเกม

ด้วยวิธีการดังกล่าว จึงสามารถทำให้หุ่นยนต์มีคุณสมบัติที่แตกต่างกันได้ แม้เดิมทีจะเป็นหุ่นยนต์ชนิดเดียวกัน ดังตัวอย่างการติดตั้งชิ้นส่วนที่แตกต่างกัน3แบบให้กับหุ่นยนต์ตัวเดิม

1. หุ่นยนต์ Assault Drone ที่ไม่ได้ติดตั้งอะไรเลย

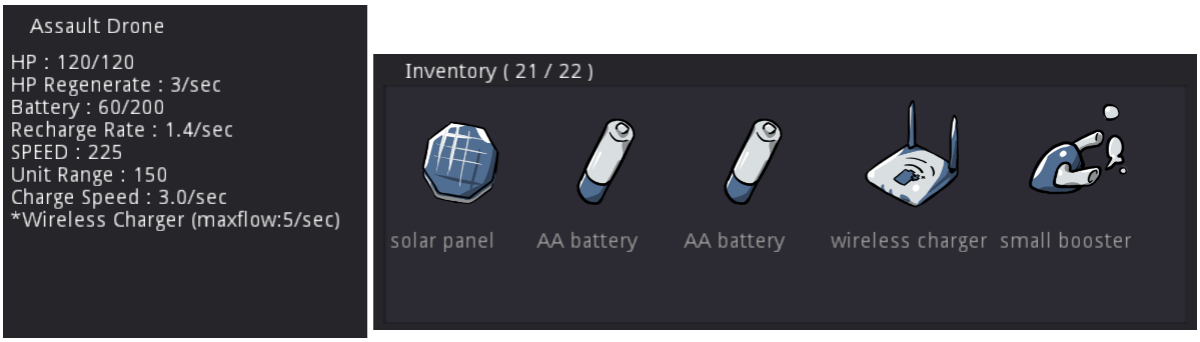


รูปที่ 5.6 ตัวอย่างการติดตั้งชิ้นส่วนให้หุ่นยนต์แบบที่ 1

2. หุ่นยนต์ Assault Drone ที่ติดตั้งป้อมปืน เกราะ และอุปกรณ์รักษาตัวเอง

รูปที่ 5.7 ตัวอย่างการติดตั้งชิ้นส่วนให้หุ่นยนต์แบบที่ 2

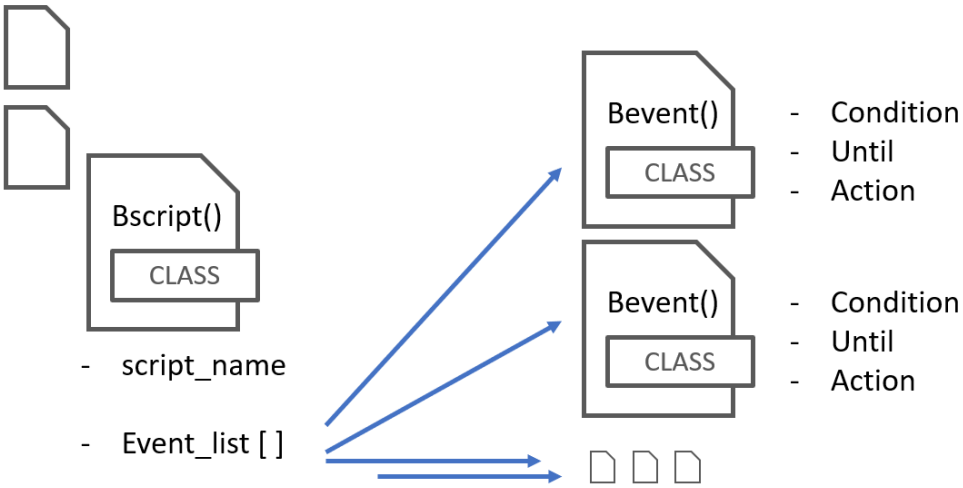
3. หุ่นยนต์ Assault Drone ที่ติดตั้งอุปกรณ์แบตเตอรี่ และ booster เพิ่มความเร็ว



รูปที่ 5.8 ตัวอย่างการติดตั้งชิ้นส่วนให้หุ่นยนต์แบบที่ 3

5.3.3 การทำงานของสคริปต์หุ่นยนต์

เพื่อให้กลุ่มของคำสั่งต่างๆ มีโครงสร้างการทำงานได้อย่างที่ผู้พัฒนาได้เคยออกแบบไว้ ผู้พัฒนาจึงได้ทำการสร้างคลาสขึ้นมาสองคลาสในโปรแกรม คือคลาสของสคริปต์หุ่นยนต์ และคลาสของเหตุการณ์หุ่นยนต์ โดยมีความสัมพันธ์ดังรูปที่ 5.9



รูปที่ 5.9 โครงสร้างของสคริปต์คำสั่งในหุ่นยนต์

จากรูป Bscript (สคริปต์คำสั่ง) แต่ละสคริปต์ จะเก็บ Array ของ Bevent (เหตุการณ์) หลายๆ เหตุการณ์ไว้ในตนเอง และเมื่อ Bscript ถูกเรียกประมวณผล ก็จะไปตามเรียกประมวณผล Bevent ของตัวเองต่ออีกทีตามลำดับ หาก Bevent ไหนไม่ตรงเงื่อนไข ก็จะไปเรียกประมวณผล Bevent อันต่อๆไปต่อ และเมื่ออันใดถูกเงื่อนไขก็จะเรียกทำคำสั่งของ Bevent นั้น แล้วตรวจดูว่าคำสั่งนั้นเป็นประเภทhardหรือ soft หากเป็นhardก็จะเลิกการทำงานทันที แต่หากเป็นsoftก็จะเรียกประมวณผลคำสั่งต่อๆ ไป ต่อจนหมด

5.3.4 การสร้างสร้างนิพจน์ต่างๆเพื่อใช้ใน Condition

ในการสร้างเงื่อนไขให้เหตุการณ์ จะสามารถเขียนเงื่อนไขได้ในรูปแบบของนิพจน์ (Expression) คือ ประกอบด้วยตัวแปรหรือค่าคงที่ต่างๆ และตัวดำเนินการ (Operator)

ตารางที่ 5.1 ตัวแปรต่างๆ ที่เป็นคุณลักษณะของหุ่นยนต์

คุณลักษณะ	ชื่อตัวแปร
พลังชีวิต	health, hp, hitpoint
พลังชีวิตสูงสุด	maxHealth, maxHp, maxHitpoint
อัตราการรักษาพลังชีวิตอัตโนมัติ	healthRegen, hpRen, hitpointRegen
พลังงานแบตเตอรี่	battery, bttr
พลังงานแบตเตอรี่สูงสุด	maxBattery, maxBttr
อัตราการชาร์จพลังงานอัตโนมัติ	bttrRegen, batteryRegen
ความเร็วในการเคลื่อนที่	speed, moveSpeed
ความเร็วในการดึงหรือส่งพลังงาน (ชาร์จ)	chargeSpeed
ระยะในการชาร์จหรือรักษา	ChargeRange, healRange, unitRange, range
Flow สูงสุดที่สามารถแชร์พลังงานได้	chargeFlow
พื้นที่ใส่ของที่ใช้แล้ว	storage
พื้นที่ใส่ของทั้งหมด	maxStorage
ถูกผู้เล่นเลือกอยู่ไหม	IsSelected, selected, select'

** ตัวแปรเหล่านี้ เป็นการเทียบแบบ case-insensitive

ตารางที่ 5.2 ตัวดำเนินการที่สามารถใช้ได้

ตัวดำเนินการคณิตศาสตร์	+, -, *, /, %
ตัวดำเนินการเปรียบเทียบ	> , <, >=, <=, ==, !=
ตัวดำเนินการตรรกะ	and, &&, or, ||, not, !

นำตัวแปรและตัวดำเนินการดังที่กล่าวมามาผสมกันจนเกิดเป็นเงื่อนไขต่างๆดังตัวอย่าง

ตารางที่ 5.3 ตัวอย่างการเปลี่ยนประโยคต่างๆ ให้กลายเป็นเงื่อนไขในเกม

ประโยค	เขียนได้ว่า
ถ้าพลังชีวิตต่ำ (น้อยกว่า20)	hp < 20
ถ้าสามารถสร้างพลังงานแบตเตอรี่เองได้	bttrRegen > 0
ถ้าสามารถสร้างพลังงานแบตเตอรี่เองได้ และมีพลังงานอยู่50หน่วยขึ้นไป	bttrRegen > 0 and bttr >= 50
ถ้าพลังชีวิตเหลืออยู่ต่ำกว่า 30%	(hp / maxHp) * 100 < 30
ถ้าพลังชีวิตเหลืออยู่ต่ำกว่า 30%	Hp < 30%

สังเกตได้ว่าเครื่องหมาย % นั้น นอกจากจะสามารถใช้ในการ modulo ได้แล้ว ผู้เขียนยังทำให้มันสามารถนำมาเปรียบเทียบเชิงปริมาณได้ด้วย หากใช้ต่อท้ายตัวเลข

เมื่อหุ่นยนต์คำนวณนิพจน์เงื่อนไขจนได้ผลลัพธ์ของนิพจน์แล้ว หุ่นยนต์จะถือว่าถูกเงื่อนไขของเหตุการณ์นั้นแล้วทำคำสั่งก็ต่อเมื่อ ผลลัพธ์ของนิพจน์เป็นจริง หรือมีอยู่จริง ซึ่งจะได้ขยายความของคำว่า มีอยู่จริง ด้วยเรื่องของกลุ่มหุ่นยนต์ในเนื้อหาต่อไป

5.3.5 การเรียกหาหุ่นยนต์ที่ถูก tag และการจัดการกลุ่มของหุ่นยนต์ (list)

นอกจากตัวแปรที่เป็นคุณลักษณะของหุ่นยนต์ตัวเองแล้ว หุ่นยนต์ยังสามารถเขียนตัวแปรเป็น tag ต่างๆ เพื่อกล่าวถึงหุ่นยนต์ทั้งหมดที่มีการติด tag เหล่านั้นด้วย เช่นการเขียนว่า ally เพื่อแทนถึงหุ่นยนต์ฝั่งเราทั้งหมด หรือ enemy เพื่อแทนหุ่นยนต์ฝั่งศัตรูทั้งหมด

เวลากล่าวถึงหุ่นยนต์ผ่าน tag นั้น เราจะได้ list ของหุ่นยนต์ที่ใช้ tag นั้นๆ กลับมา หาก tag นั้นไม่เคยถูกปักไว้ที่ไหนเลย เราก็จะได้ list เปล่าๆที่ไม่ได้ชี้ไปที่หุ่นยนต์ตัวไหนเลยกลับมา ซึ่ง list เหล่านี้เราก็ยังสามารถนำไปดำเนินการต่อได้ เพื่อให้ได้ผลลัพธ์ที่เจาะจงหรือครอบคลุมมากขึ้นตามที่เราต้องการ

การกรอง list หุ่นยนต์ (filter)

เหมือนกับการกล่าวถึงคุณลักษณะต่างๆของหุ่นยนต์ตัวเอง เราสามารถเข้าถึงคุณลักษณะต่างๆ หุ่นยนต์อื่นได้ผ่านตัวแปรใน ตาราง5.1 ด้วย ด้วยการเขียน tag หุ่นยนต์ เครื่องหมายจุด (.) คั่น แล้วตามด้วยชื่อคุณลักษณะ เช่นพลังชีวิตของหุ่นยนต์ฝั่งเราเขียนได้ว่า ally.hp หากแต่เมื่อเรานำกลุ่มหุ่นยนต์นี้ไปใช้กับตัวดำเนินการเปรียบเทียบ เราจะไม่ได้ค่าความจริงกลับมา แต่จะเป็นการกรองแทน ด้วยการเทียบกับหุ่นยนต์ใน list ทุกตัว หากตัวไหนได้ค่าเป็นเท็จจะถูกกรองออกไป เป็นเหตุว่าทำไมเราต้องเช็คเงื่อนไขเป็นการมีอยู่จริงแทน ดังตัวอย่าง

ตารางที่ 5.4 ตัวอย่างการใช้การกรอง (filter) กับเงื่อนไขของเหตุการณ์

ประโยค	เขียนได้ว่า
ถ้ามีศัตรูอยู่	enemy
ถ้ามีศัตรูที่พลังชีวิตมากกว่า200	enemy.hp > 200
ถ้ามีศัตรูอยู่ในระยะ300หน่วย	enemy.distance < 300
ถ้ามีศัตรูที่พลังชีวิตมากกว่า200 และอยู่ในระยะ300หน่วย	(enemy.hp > 200).distance < 300

* หากเป็นการอ่านคุณลักษณะของหุ่นยนต์ตัวอื่น จะมีตัวแปร distance เพิ่มมา หมายถึงระยะห่างระหว่างหุ่นยนต์ตัวเองถึงหุ่นยนต์ตัวนั้นๆ

การดำเนินการ Set

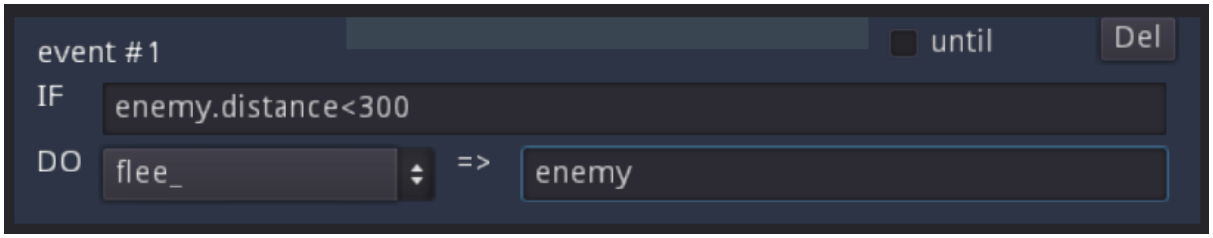
ในการดำเนินการกับ list นั้น เครื่องหมาย and, or หรือ – นั้นจะมีความหมายในทางอื่นด้วย คือการ Intersect, Union และ Minus ตามลำดับ ดังตัวอย่าง

ตารางที่ 5.5 ตัวอย่างการใช้ตัวดำเนินการ Set

ประโยค	เขียนได้ว่า
ศัตรูที่พลังชีวิตมากกว่า200 และอยู่ในระยะ300หน่วย	(enemy.hp > 200) and (enemy.distance < 300)
ศัตรูที่พลังชีวิตมากกว่า200 และอยู่ในระยะ300หน่วย	(enemy.hp > 200) - (enemy.distance >= 300)

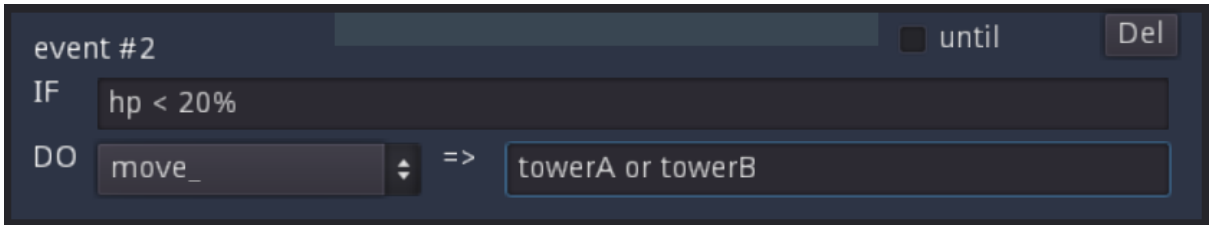
สังเกตุได้ว่าการเขียนนิพจน์เหล่านี้ก็ไม่ได้ใช้ได้แค่กับ Condition เท่านั้น เนื่องจากผลลัพธ์ที่ได้ไม่ได้เป็นแค่ค่าคงที่ แต่เป็นกลุ่มของหุ่นยนต์ซึ่งสามารถนำไปสื่อถึงเป้าหมายของคำสั่งในเหตุการณ์ต่างๆได้ด้วย

- ถ้ามีศัตรูอยู่ในระยะ 300 หน่วย ให้เดินหนีจากศัตรู



รูปที่ 5.10 ตัวอย่างการเขียนเหตุการณ์ให้หุ่นยนต์ 1

- ถ้าพลังชีวิตเหลือต่ำกว่า 20% ให้เดินไปที่ towerA หรือ towerB



รูปที่ 5.11 ตัวอย่างการเขียนเหตุการณ์ให้หุ่นยนต์ 2

บทที่ 6

ผลสรุป

หลังจากการทดสอบระบบต่างๆ ภายในเกม เพื่อดูว่าโปรแกรมเป็นตามวัตถุประสงค์ที่ตั้งไว้หรือไม่ รวมถึงดูข้อบกพร่องที่พบเจอในโปรแกรม และตลอดการดำเนินงาน เพื่อนำไปเป็นประสบการณ์ในการพัฒนางานต่อๆ ไปได้ ได้รายงานปัญหา ข้อสรุป และข้อเสนอแนะดังนี้

6.1 การดำเนินงาน

1. เกมทำงานบนระบบปฏิบัติการวินโดวส์ พัฒนาบน Godot Engine
2. เกมเป็นเกมเอาชีวิตรอดจากหุ่นยนต์ฝั่งตรงข้าม โดยสู้เพื่อแย่งเอาทรัพยากรมา สร้างกองกำลังหุ่นยนต์ของตัวเองให้พัฒนาไปเรื่อยๆ Game Over เมื่อหุ่นยนต์หลักของผู้เล่นถูกทำลาย
3. หุ่นยนต์มีหลายชนิด มีคุณลักษณะแตกต่างกันไป ทั้งยังสามารถติดตั้งชิ้นส่วนต่างๆ ให้หุ่นยนต์เพื่อปรับแต่งคุณลักษณะหรือความสามารถของหุ่นยนต์
4. เราสามารถโปรแกรมให้หุ่นยนต์ทำงานอัตโนมัติได้ ด้วยการโปรแกรมสคริปต์คำสั่งให้หุ่นยนต์ ด้วยภาษาใหม่ที่ผู้เขียนสร้างขึ้นมาเอง เพื่อให้ใช้ง่ายและเหมาะสมกับตัวเกมที่สุด
5. สคริปต์ของหุ่นยนต์ประกอบไปด้วย เหตุการณ์หลายๆ เหตุการณ์ ที่ผู้เล่นกำหนดเงื่อนไขขึ้นมาเอง ว่าถ้าเกิดเหตุการณ์นี้ขึ้นมา จะให้หุ่นยนต์ดำเนินการอย่างไรต่อไป
6. สคริปต์ของหุ่นยนต์นั้นอยู่เป็นเอกเทศกับตัวหุ่นยนต์เอง จะได้สามารถติดตั้งสคริปต์ร่วมกันในหลายๆ หุ่นยนต์ได้ ไม่ต้องมาคอยเขียนใหม่ซ้ำๆ ทั้งยังรองรับระบบตัวแปรที่สามารถละค่าจากในสคริปต์ แล้วมากำหนดในตัวหุ่นยนต์ภายหลัง เพื่อให้หุ่นยนต์มีเป้าหมายที่หลากหลายแม้จะใช้สคริปต์คำสั่งร่วมกัน
7. ในการกำหนดค่าของตัวแปรนั้น ผู้พัฒนาได้ใช้ระบบ Tag ที่สามารถแปะป้ายชื่อให้สิ่งต่างๆ ในเกมได้ เพื่อที่จะได้นำไปกล่าวถึงได้ในสคริปต์คำสั่ง

6.2 ผลการทดลองโปรแกรม

1. เกมสามารถสร้างสคริปต์คำสั่งขึ้นมาใหม่ และกำหนดให้แก่หุ่นยนต์ตัวใดๆ ได้
2. หุ่นยนต์สามารถดำเนินการตามคำสั่งต่างๆ ที่ระบุไว้ในสคริปต์ของหุ่นยนต์ได้
3. การรันคำสั่งของหุ่นยนต์ มีการเช็คเงื่อนไขได้ถูกต้องรวมถึงทำงานตามลำดับความสำคัญที่กำหนด
4. ตัวแปรของหุ่นยนต์มีการเขียนทับสคริปต์ และทำงานกับเป้าหมายที่เฉพาะเจาะจงได้
5. มีการ tag วัตถุแวดล้อมได้ รวมถึงเรียกหาและจัดการกับกลุ่มวัตถุที่ถูก tag ไว้ได้อย่างถูกต้อง
6. มีการสร้างหุ่นยนต์ของฝั่งตรงข้ามขึ้นมาเพื่อต่อสู้ และจบเกมเมื่อหุ่นยนต์หลักของผู้เล่นถูกทำลายได้

6.3 ปัญหาที่พบ

1. โครงสร้างของเกมมีความซับซ้อนสูง ทำให้ต้องใช้เวลามากในการวางระบบการทำงาน
2. ไม่ควรจะมีคำสั่งของผู้เล่นที่ไม่ถูกต้องเกิดขึ้นในสคริปต์การทำงานของหุ่นยนต์ จึงต้องมีการทำระบบตรวจสอบความถูกต้องของคำสั่งต่างๆ ที่ผู้เล่นป้อนเข้ามาด้วย ทำให้ใช้เวลามากกว่าที่วางแผนไว้
3. ปัญหา Performance ของเกมเมื่อมีหุ่นยนต์อยู่จำนวนมาก เนื่องจากหุ่นยนต์แต่ละตัวก็ต้องประมวณผลสคริปต์ของตัวเองอยู่ตลอดเวลา

6.4 ข้อสรุปและข้อเสนอแนะ

ข้อสรุป เกมที่พัฒนาสามารถทำงานบนคอมพิวเตอร์ ในระบบปฏิบัติการวินโดวส์ได้ อาจมีการล่าช้าในการพัฒนาบ้าง เนื่องจากพบกับปัญหาต่างๆ ที่ไม่คาดคิด แต่สุดท้ายผลลัพธ์ก็ยังออกมาใกล้เคียงกับที่ตั้งใจไว้แต่แรก อาจมีการปรับเปลี่ยนระหว่างการพัฒนาบ้างตามความเหมาะสม

ข้อเสนอแนะ หน้าจอแสดงผลต่างๆ ในเกมยังต้องการพัฒนาอีก เพื่อความเหมาะสมเมื่อเล่นกับขนาดหน้าจอแตกต่างออกไป ทั้งขนาดของกล่องข้อความต่างๆ และตัวหนังสือ ที่ไม่ได้ออกแบบมาให้ยืดหยุ่นตาม

เอกสารอ้างอิง

[1] SpriteBox LLC. (2558). Light Bot. https://lightbot.com/. วันที่สืบค้น 20 สิงหาคม 2562.

[2] Tomorrow Corporation. (2558). Human Resource Machine.
https://tomorrowcorporation.com/humanresourcemachine . วันที่สืบค้น 20 สิงหาคม 2562.

[3] GFX47. (2562). Gladiabots. https://gladiabots.com/. วันที่สืบค้น 20 สิงหาคม 2562.

[4] Rovio Entertainment. (2555). Bad Piggies. https://www.rovio.com/channel/bad-piggies .
วันที่สืบค้น 17 ตุลาคม 2562.

[5] Intel Hyper-Threading Technology, Technical User's Guide. (2546.1). Multithreading. p. 6
วันที่สืบค้น 17 ตุลาคม 2562.

[6] Unity Technologies. (2562.2). Unity User Manual, Event Trigger.
2562.https://docs.unity3d.com/Manual/script-EventTrigger.html. วันที่สืบค้น 17 ตุลาคม 2562.

[7] Juan Linietsky, Ariel Manzur. (2557). Godot Engine. https://godotengine.org/.
วันที่สืบค้น 17 ตุลาคม 2562.

[8] Juan Linietsky, Ariel Manzur. (2557). GDScript.
https://docs.godotengine.org/en/3.1/getting_started/scripting/gdscript/index.html.
วันที่สืบค้น 17 ตุลาคม 2562.

[9] Juan Linietsky, Ariel Manzur. (2557). InputEvent.
https://docs.godotengine.org/en/3.1/tutorials/inputs/inputevent.html.
วันที่สืบค้น 17 ตุลาคม 2562.

[10] Juan Linietsky, Ariel Manzur. (2557). YSort.
https://docs.godotengine.org/en/latest/classes/class_ysort.html. วันที่สืบค้น 17 ตุลาคม 2562.

[11] Juan Linietsky, Ariel Manzur. (2557). Physics introduction.
http://docs.godotengine.org/en/latest/tutorials/physics/physics_introduction.html.
วันที่สืบค้น 17 ตุลาคม 2562.

[12] Juan Linietsky, Ariel Manzur. (2557). Class packedscene.
https://docs.godotengine.org/en/3.1/classes/class_packedscene.html.
วันที่สืบค้น 17 มีนาคม 2563

ภาคผนวก

ภาคผนวก ก คู่มือการติดตั้งเกมเอาชีวิตรอดหุ่นยนต์

การติดตั้งจากไฟล์ **zip**

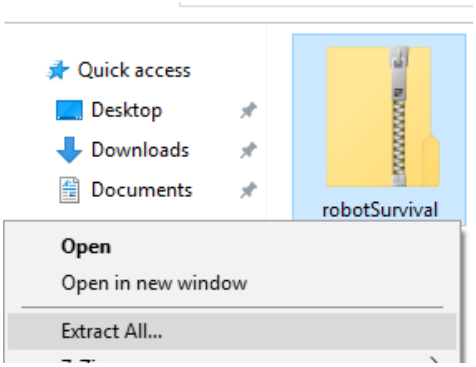
1. หากมีไฟล์อยู่แล้ว ให้ข้ามไปทำที่ข้อ 4.

   หากไม่มี สามารถโหลดไฟล์จาก Google Drive ได้ ตามข้อ 2.

2. ทำการเข้า website จาก URL ต่อไปนี้ http://bit.ly/rs_dl (อาร์เอส_ดีแอล)

3. กดเครื่องหมาย download  ที่มุมขวาบน เพื่อโหลดไฟล์

4. คลิกขวาตรงไฟล์ **robotSurvival.zip** ที่ได้ แล้วเลือก Extract All



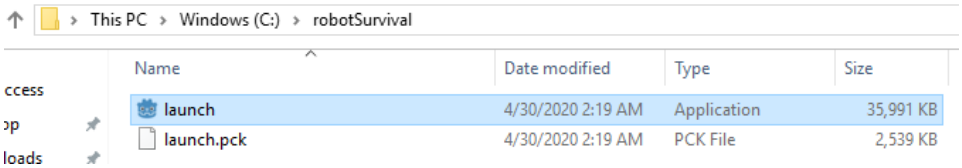
รูปที่ ก.1 การ unzip ไฟล์

   จะได้แฟ้มข้อมูลชื่อ **robotSurvival** มา

5. ทำต่อที่ข้อ 3 ของ การติดตั้งโดยใช้แผ่นซีดี

6. เมื่อเปิดแฟ้มข้อมูลจะพบกับไฟล์ชื่อ **launch.exe**

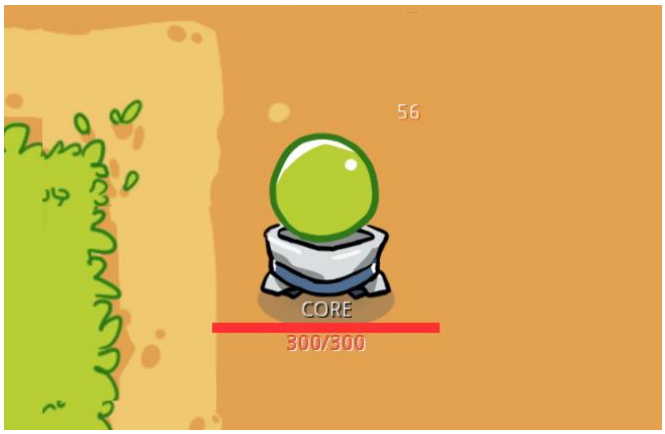
7. กดดับเบิลคลิกเปิดไฟล์ **launch.exe** เพื่อเริ่มเล่นเกม !



รูปที่ ก.2 ตัวเล่นเกม

ภาคผนวก ข คู่มือการเล่นเกมเอาชีวิตรอดหุ่นยนต์

1. เมื่อโหลดเกมเสร็จจะพบกับหุ่นยนต์หลักของผู้เล่นอยู่ตรงกลางหน้าจอ



รูปที่ ข.1 หุ่นยนต์หลักของผู้เล่น

เมื่อกดเลือกที่หุ่นยนต์ จะมีการแสดงค่าต่างๆ ของหุ่นยนต์ตัวนั้น อธิบายได้ดังนี้

- CORE คือชนิดของหุ่นยนต์ (หุ่นยนต์หลักของผู้เล่นมีชนิดเป็น CORE)
- 300/300 คือพลังชีวิตของหุ่นยนต์ พร้อมหลอดพลังชีวิตกำกับ เมื่อพลังชีวิตหมดจะถูกทำลาย โดย 300/300 หมายถึงตอนนี้มีค่าพลังชีวิต 300 จากพลังชีวิตเต็มที่ 300
- 56 (เลขด้านขวาบนของหุ่นยนต์) คือ ค่าพลังงานแบตเตอรี่ที่หุ่นยนต์ตัวนี้ม

2. เป้าหมายของเกมคือควบคุมหุ่นยนต์ เพื่อเอาชีวิตรอดให้นานที่สุด หากหุ่นยนต์หลักของผู้เล่นถูกทำลายจะ game over พร้อมมีสรุปคะแนนที่ผู้เล่นทำได้ในการเล่นในรอบนี้

โดยการควบคุมต่างๆ ของเกมทั้งหมด มีดังนี้

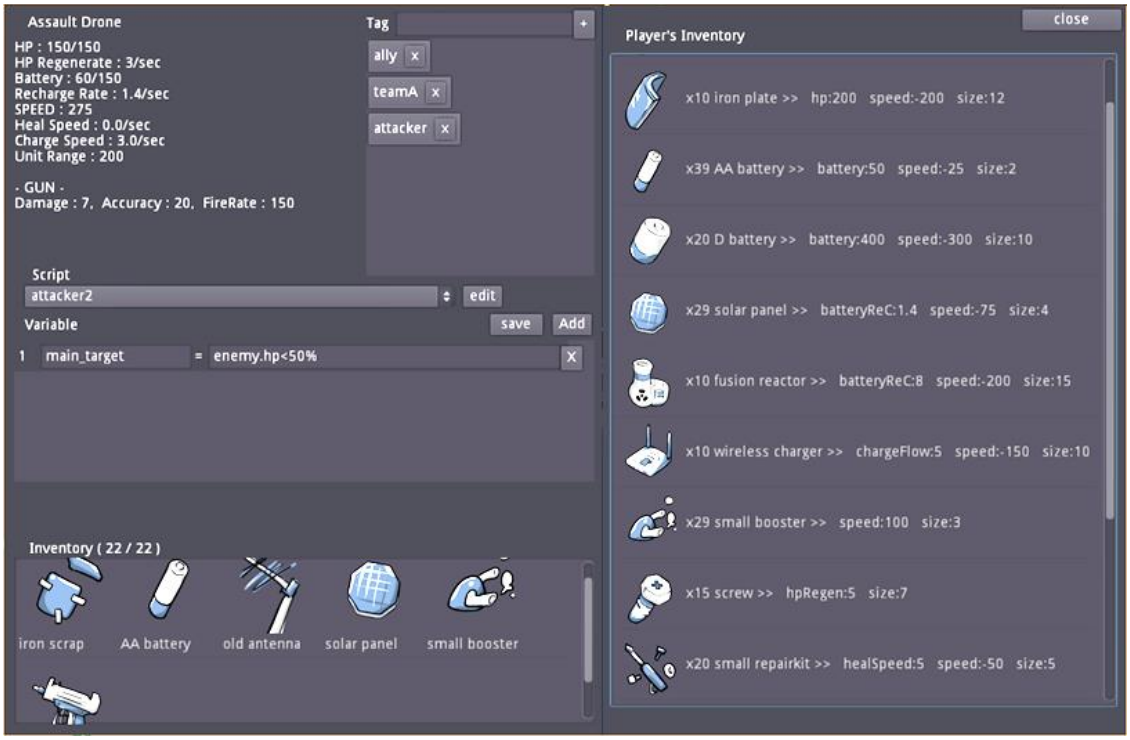
2.1 การควบคุมมุมกล้อง

- เลื่อนเมาส์ไปอยู่ที่ขอบของหน้าต่างเกม เพื่อเลื่อนหน้าจอไปตามทิศนั้น
- กดสกอร์เมาส์ หรือ สเปซบาร์ ค้างไว้ แล้วลากเมาส์ เพื่อเลื่อนหน้าจอ
- หมุนสกอร์เมาส์ เพื่อซูมกล้องเข้าออก

2.2 การควบคุมเกม

- คลิกซ้ายเพื่อเลือกหุ่นยนต์ สามารถคลิกค้างแล้วคลุมได้ เพื่อเลือกทีละมากๆ
- กด Shift ค้างแล้วเลือกที่หุ่นยนต์ จะเป็นการเลือกเพิ่ม แต่ถ้าเลือกไปที่หุ่นยนต์ที่กำลังเลือกอยู่แล้วจะเป็นการยกเลิกการเลือก
- คลิกขวา เพื่อสั่งให้หุ่นยนต์ที่กำลังเลือกอยู่เคลื่อนที่ไปยังจุดที่คลิก

- กดดับเบิลคลิกที่หุ่นยนต์ จะเป็นการเปิดหน้าต่างของหุ่นยนต์ตัวนั้นขึ้นมา เพื่อโปรแกรมคำสั่งหรือติดตั้งชิ้นส่วนต่างๆ ให้หุ่นยนต์ โดยหน้าต่างด้านขวาจะเป็นชิ้นส่วนที่เรามี หน้าต่างด้านซ้ายล่างคือชิ้นส่วนที่กำลังติดตั้งให้หุ่นยนต์อยู่ กดดับเบิลคลิกที่ชิ้นส่วนเหล่านั้นเพื่อย้ายมาติดตั้ง หรือถอดกลับไป



รูปที่ ข.2 หน้าต่างเมนูของหุ่นยนต์

- กด Ctrl พร้อมกับตัวอักษร A-Z เพื่อทำการปักป้าย tag ไปยังจุดที่เลือก โดยจะมีชื่อเป็น AAA-ZZZ ตามตัวอักษรที่กด กดเลือกวัตถุเพียงวัตถุเดียว แล้วปักป้ายขณะที่ชี้ไปที่วัตถุนั้น เพื่อปักป้ายไปยังวัตถุ
- กด Ctrl + Shift พร้อมกับตัวอักษร A-Z เพื่อเป็นการลบป้ายชื่อ tag เหล่านั้น
- กด P เพื่อหยุดเกม

2.3 การ Save และ Load เกม

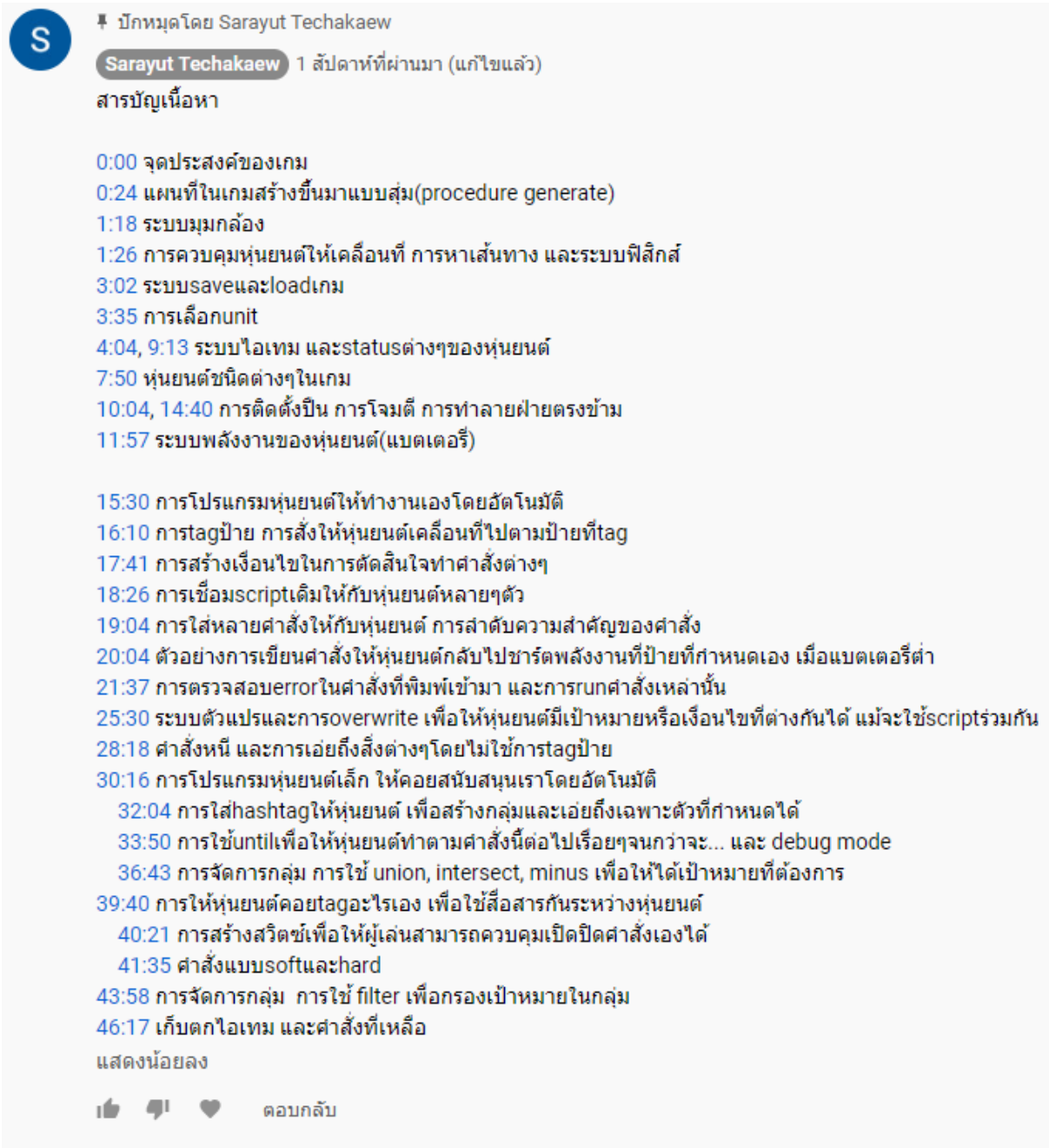
เกมนี้มีช่องให้บันทึกทั้งหมด10ช่อง โดยสามารถกดบันทึกและโหลดได้ตลอดเวลาเมื่อไม่ได้อยู่ในเมนู และสามารถโหลดบันทึกขึ้นมาได้ แม้จะทำการปิดเกมและเปิดขึ้นมาใหม่แล้ว ผ่านการกดปุ่มต่างๆ ดังนี้

- กดเลข 0-9 เพื่อโหลดเกมจากช่องต่างๆ ที่บันทึกไว้
- กด Shift พร้อมกับเลข 0-9 เพื่อทำการบันทึกไปยังช่องเหล่านั้น

2.4 การโปรแกรมคำสั่งให้หุ่นยนต์

พื้นฐานการสร้างเงื่อนไข การประกอบนิพจน์ เพื่อโปรแกรมหุ่นยนต์สามารถดูได้ที่ หัวข้อ 5.3.4 และหัวข้อ 5.3.5 ที่หน้า 56 เป็นต้นไป พร้อมกับดูความหมายของคำสั่งต่างๆจาก ตาราง 5.2 หน้า 23 ประกอบ

และเนื่องจากต่อจากนี้เป็นเนื้อหาที่ค่อนข้างซับซ้อน ผู้พัฒนาจึงได้นำเสนอผ่านการอัดวิดีโอตัวอย่าง โดยมีการแบ่งเนื้อหาต่างๆ ไว้ในความคิดเห็นของวิดีโอ หากต้องการดูเฉพาะเนื้อหาไหน ก็สามารถกดข้ามไปยังเวลาเหล่านั้นได้เลย โดยสามารถเข้าดูได้จาก URL ดังนี้ https://bit.ly/rs_demo



รูปที่ ข.3 สารบัญเนื้อหาต่างๆ ของคลิปบรรยายประกอบเกม Robot Survival

2.5 คำสั่งลัดภายในเกม

เพื่อให้ผู้เล่นสามารถทดสอบระบบต่างๆ ของเกมอย่างทั่วถึง มีอิสระในสร้างสถาณะการในแบบที่ผู้เล่นต้องการ ผู้พัฒนาจึงได้เปิดโอกาสให้ผู้เล่นเข้าถึงคำสั่งลับต่างๆ ได้ ไม่ว่าจะเป็นการโยกย้ายชิ้นส่วนของหุ่นยนต์ฝ่ายตรงข้าม ควบคุมหุ่นยนต์ฝ่ายตรงข้าม หรือกระทั่งคำสั่งลับต่างๆ เพื่อเป็นทางลัดของเกมดังนี้

- กด A เพื่อสร้าง Assault Drone ขึ้นมาในตำแหน่งที่เม้าส์อยู่
- กด S เพื่อสร้าง Speed Drone ขึ้นมาในตำแหน่งที่เม้าส์อยู่
- กด D เพื่อสร้าง Defend Drone ขึ้นมาในตำแหน่งที่เม้าส์อยู่
- กด F เพื่อสร้าง Fat Drone ขึ้นมาในตำแหน่งที่เม้าส์อยู่
- กด X เพื่อสร้าง Mini Drone ขึ้นมาในตำแหน่งที่เม้าส์อยู่
- กด Shift ค้าง พร้อมกับปุ่ม A, S, D, F, X เหมือนกับก่อนหน้า เพื่อสร้างหุ่นยนต์ฝั่งศัตรู
- กด W เพื่อสร้างหุ่นยนต์ใหม่ที่มีลักษณะทุกอย่างเหมือนหุ่นยนต์หนึ่งตัวที่กำลังเลือกอยู่
- กด E เพื่อทำลายหุ่นยนต์ทุกตัวที่กำลังเลือก
- กด O เพื่อสลับการ เปิด/ปิด การเกิดของหุ่นยนต์ฝ่ายตรงข้าม