การพัฒนาโปรแกรมเกมหุ่นยนต์รบที่ควบคุมด้วยภาษาจาวา

DEVELOP ROBOT GAME WITH CONTROL BY JAVA PROGRAMMING LANGUAGE

ศิวกร ดำรงภัทร
สุตภัทร พุ่มศิริ
อดุลศักดิ์ สังข์เกิด

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2544

DEVELOP ROBOT GAME WITH CONTROL BY JAVA PROGRAMMING LANGUAGE

SIVAKORN DAMRONGPHATR
SUTTAPAT POOMSIRI
ADULSAK SUNGKIRD

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE
FACULTY OF SCIENCE
KING MONKUT'S INSTITUE OF TECHNOLOGY LADKRABANG
ACADEMIC YEAR 2001

ปัญหาพิเศษเรื่อง การพัฒนาโปรแกรมเกมหุ่นยนต์รบที่ควบคุมโดยภาษาจาวา

DEVELOP ROBOT GAME WITH CONTROL BY JAVA PROGRAMMING LANGUAGE

ชื่อนักศึกษา นายศิวกร ดำรงภัทร 41056110

นายสุตภัทร พุ่มศิริ 41056121

นายอดุลศักดิ์ สังข์เกิด 41056132

ภาควิชา คณิตศาสตร์ประยุกต์และวิทยาการคอมพิวเตอร์

สาขาวิชา วิทยาการคอมพิวเตอร์

อาจารย์ที่ปรึกษา อาจารย์วิสันต์ ตั้งวงษ์เจริญ

อาจารย์ชาญชัย ดีอ่วม

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร วิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2544

คณะกรรมการสอบ		ลายมือชื่อ
ประธานกรรมการ	อาจารย์พรชัย เจนจิระพงศ์เวช	
กรรมการ	ผู้ช่วยศาสตราจารย์กฤษฎา ไตรสุรัตน์	
กรรมการและอาจารย์ที่ปรึกษา	อาจารย์วิสันต์ ตั้งวงษ์เจริญ	
กรรมการและอาจารย์ที่ปรึกษา	อาจารย์ชาญชัย ดีอ่วม	

(ผู้ช่วยศาสตราจารย์ไพรบูลย์ พันธรักษ์พงษ์)

หัวหน้าภาควิชาวิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

ปัญหาพิเศษเรื่อง การพัฒนาโปรแกรมเกมหุ่นยนต์รบที่ควบคุมโดยภาษาจาวา
DEVELOP ROBOT GAME WITH CONTROL BY JAVA PROGRAMMING LANGUAGE

ชื่อนักศึกษา นาย ศิวกร ดำรงภัทร 41056110
นาย สุตภัทร พุ่มศิริ 41056121
นาย อดุลศักดิ์ สังข์เกิด 41056132

ปริญญา วิทยาศาสตรบัณฑิต

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์

สาขาวิชา วิทยาการคอมพิวเตอร์

ปีการศึกษา 2544

อาจารย์ที่ปรึกษา อาจารย์วิสันต์ ตั้งวงษ์เจริญ
อาจารย์ชาญชัย ดีอ่วม

บทคัดย่อ

ปัจจุบันแนวความคิดเชิง Object Oriented มีความแพร่หลายและได้รับความนิยมเป็นอย่างมากทั้งในหมู่ผู้พัฒนาระบบในหลายๆองค์กรและนักเรียนนักศึกษาและผู้สนใจทั่วไป เนื่องจากเป็นแนวความคิดที่ช่วยในการพัฒนาซอฟต์แวร์ให้มีประสิทธิภาพสามารถแก้ปัญหาการพัฒนาซอฟต์แวร์ในระบบเก่าที่เกิดขึ้นในหลาย ๆ ขั้นตอนของการพัฒนาได้ ไม่ว่าจะเป็นการค่าใช้จ่ายในการพัฒนาที่ค่อนข้างสูง สามารถใช้ได้เฉพาะงานนั้น ๆ เท่านั้นไม่สามารถนำกลับมาใช้ใหม่ได้อีก เมื่อผู้ใช้ต้องการเปลี่ยนความต้องการเกี่ยวกับการทำงานของระบบที่ทำเสร็จแล้วก็ยังเป็นการยุ่งยากเพราะต้องไปเริ่มใหม่อีก แต่ขั้นตอนการศึกษาและเรียนรู้ในเรื่อง Object Oriented Technology นั้นค่อนข้างยากต่อการทำความเข้าใจ ทั้งยังขาดแนวทางที่ถูกต้องและชัดเจนในการพัฒนาจนอาจทำให้เกิดความเบื่อหน่ายที่จะพัฒนาโปรแกรม

ดังนั้นการพัฒนาโปรแกรมเกมหุ่นยนต์รบที่ควบคุมโดยภาษาจาวาจึงเป็นแนวทางหนึ่งเพื่อให้ผู้ใช้หรือผู้เล่นเกมสามารถฝึกฝน เรียนรู้ ทำความเข้าใจและพัฒนาโปรแกรมเชิง Object Oriented ได้ โดยใช้เกมเป็นสื่อกลางเพื่อให้ผู้สนใจไม่เกิดความเบื่อหน่ายในการศึกษาและใช้ภาษา Java ซึ่งเป็นภาษาที่มีแนวความคิดเชิง Object Oriented และได้รับความนิยมสูงในขณะนี้มาดำเนินการทำให้การศึกษามีความสะดวกขึ้น

Special Project Title	Develop Robot Game With Control By Java Programming Language
Students	Mr. Sivakorn Damrongphatr 41056110
	Mr. Suttapat Pumsiri 41056121
	Mr. Adulsak Sungkird 41056132
Degree	Bachelor's Degree of Science
Department	Mathematics and Computer Science , Faculty of Science
Programme	Computer Science
Academic Year	2001
Special Project Advisor	Lecturer Wisan Tungwongjarean
	Lecturer Chanchai Dee-uam

Abstract

Nowadays, the concept of "objected oriented" is very popular among developer in many organizations, students, and all people who interested in. This is because this concept helps to develop the software to more efficient level. In addition, this concept can solve the problems that happen many times in the previous system during the developing process. For example, the problem of great expenses in the developing process and the problem of the limitation of developing ability that can only use the previous software in a particular work which you can not reuse the former one when you need it for another different work. However, the education of "Objected Oriented Technology" is very difficult to understand because it lacks the right and clear way of development that can make people so bored to develop the program.

Therefore, the development of "The Fighting Robot Game Program" which controlled by Java language is the one way for the consumers or players to practice, learn, understand and develop the "Object Oriented" program. People who interested in this game will use this game as a medium for getting rid of the boredom that may occur from the education and usage of Java language, the language that has a concept of Object Oriented. Hence, the use of the concept of Object Oriented which is now popular can make the education more convenient.

กิตติกรรมประกาศ

ในการทำปัญหาพิเศษเรื่อง การพัฒนาโปรแกรมเกมหุ่นยนต์รบที่ควบคุมด้วยภาษาจาวา สามารถสำเร็จลุล่วงไปได้ด้วยดี คณะผู้จัดทำต้องขอขอบพระคุณ อาจารย์วิสันต์ ตั้งวงษ์เจริญ และ อาจารย์ชาญชัย ดีอ่วม อาจารย์ผู้รับผิดชอบปัญหาพิเศษฉบับนี้ ที่กรุณาให้คำแนะนำ และเป็นที่ปรึกษาในการแก้ปัญหาต่างๆ รวมทั้งเป็นผู้ตรวจสอบความถูกต้อง ของปัญหาพิเศษฉบับนี้

ขอกราบขอบพระคุณ คุณพ่อ และคุณแม่ ที่คอยเป็นกำลังใจแก่คณะผู้จัดทำตลอดเวลา และขอขอบคุณพี่ๆ เพื่อนๆ น้องๆ ทุกคนของคณะผู้จัดทำที่มีส่วนช่วยเหลือในปัญหาพิเศษนี้

ขอขอบพระคุณอาจารย์ทุกท่านที่ประสิทธิ์ประสาทวิชาความรู้ทั้งในภาคทฤษฎีและภาคปฏิบัติแก่คณะผู้จัดทำ และขอขอบคุณเจ้าหน้าที่ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ที่ช่วยอำนวยความสะดวกในการใช้ห้องปฏิบัติการคอมพิวเตอร์และอำนวยความสะดวกในการเบิกอุปกรณ์ต่างๆที่ใช้ในการจัดทำปัญหาพิเศษ จนปัญหาพิเศษฉบับนี้สัมฤทธิ์ผลได้ด้วยดีทุกประการ

คณะผู้จัดทำ

มีนาคม 2545

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป(ต่อ)

รูปที่ **หน้า**

สารบัญรูป(ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหา

ปัจจุบันโปรแกรมภาษา object oriented ได้มีความแพร่หลายและมีความนิยมเป็นอย่างมากเนื่องจากในการพัฒนาซอฟแวร์ในระบบเก่าเกิดปัญหาขึ้นในหลายขั้นตอนของการพัฒนา ผู้พัฒนาต้องเสียค่าใช้จ่ายในการพัฒนาค่อนข้างสูงมากเมื่อเทียบกับผลตอบแทนที่ได้ เนื่องจากสามารถใช้ได้เฉพาะงานนั้นๆเท่านั้นไม่สามารถนำกลับมาใช้ใหม่ได้อีก เมื่อผู้ใช้ต้องการเปลี่ยนความต้องการเกี่ยวกับการทำงานของระบบที่ทำเสร็จแล้วก็ยังเป็นการยุ่งยากเพราะต้องไปเริ่มใหม่อีก Object Oriented Technology เป็นแนวทางหนึ่งในการพัฒนาซอฟต์แวร์ที่มีประสิทธิภาพสามารถแก้ปัญหาเหล่านี้ได้ และในปัจจุบันได้เป็นที่นิยมในหมู่ผู้พัฒนาระบบในหลายๆองค์กร แต่มีปัญหาที่ออบเจคนั้น ขั้นตอนการศึกษาและเรียนรู้นั้นยากต่อการเข้าใจได้โดยง่าย ทั้งยังขาดแนวทางที่ถูกต้องและชัดเจนในการพัฒนา และเกิดความเบื่อหน่ายที่จะพัฒนาโปรแกรม ดังนั้น จึงมีแนวคิดในการพัฒนาโปรแกรมเพื่อให้ผู้ใช้สามารถฝึกฝนการเขียนโปรแกรม object oriented โดยให้เกม ซึ่งเขียนโดยภาษา Java ซึ่งเป็นภาษาที่มีความนิยมสูงในขณะนี้ มาดำเนินการแทนจะทำให้การศึกษามีความสะดวกขึ้น พร้อมทั้งยังได้เกมที่มีความสามารถทำงานที่มีผู้เล่น มากกว่า 1 คน และใช้งานบนอินเทอร์เน็ตได้

1.2 วัตถุประสงค์ของการทำ

1. เพื่อให้ผู้ที่สนใจภาษา Java มีแนวทางการศึกษาให้ง่ายยิ่งขึ้น โดยใช้เกมเป็นสื่อกลาง
2. เพื่อให้ผู้พัฒนาโปรแกรมเบื้องต้นสำหรับควบคุม(Control) object ในแนวความคิด object oriented ได้
3. เพื่อพัฒนาเกมในลักษณะที่เป็น online โดยมีผู้เล่น มากกว่า 1 คน บน อินเทอร์เน็ตได้
4. เพื่อให้ได้เกมควบคุมหุ่นยนต์ซึ่งผู้เล่นสามารถควบคุมได้อย่างอิสระด้วยภาษาจาวา
5. เพื่อให้ได้เว็บไซต์ที่ให้ผู้เล่นได้สามารถดาวน์โหลดโปรแกรมเพื่อสร้างหุ่นยนต์และอัพโหลดหุ่นยนต์สำหรับนำไปต่อสู้กันบนเว็บไซต์ได้
6. เพื่อให้ได้โปรแกรมสำหรับใช้ในการสร้างหุ่นยนต์

1.3 ขอบเขตของปัญหา

1. เป็นเกมการต่อสู้ของหุ่นยนต์ที่มีผู้เล่น 2 ฝ่าย แต่ละฝ่ายจะสามารถเลือกส่วนประกอบต่างๆ เพื่อประกอบขึ้นเป็นหุ่นยนต์ของตัวเอง คือส่วนของ BODY และ WEAPON ซึ่งแต่ละส่วนก็

จะมีหลายแบบ มีข้อดีข้อด้อยแตกต่างกันไป เช่น ถ้าส่วนของ BODY น้ำหนักน้อยก็อาจจะเคลื่อนที่ได้เร็ว แต่มีพลังงานต่ำ เป็นต้น ในส่วนของ WEAPON ก็มีข้อจำกัดเช่นกัน ถ้า WEAPON นั้นมีพลังในการทำลายสูงก็จะมีจำนวนครั้งในการใช้ที่จำกัด เป็นต้น ฉะนั้นผู้เล่นก็ต้องเลือกส่วนประกอบต่างๆที่คิดว่าเหมาะสมที่จะสามารถสู้กับฝ่ายตรงข้ามได้ หลังจากที่ทั้งสองฝ่ายเลือกส่วนประกอบต่างๆเสร็จสิ้นแล้ว ก็จะเริ่มการต่อสู้โดยปล่อยให้หุ่นยนต์ต่อสู้กันเองตามแต่ความสามารถของหุ่นยนต์แต่ละตัวด้วยส่วนประกอบต่างๆของ BODY และ WEAPON ที่ได้เลือกไว้และสถานการณ์ที่เกิดขึ้นตามที่ได้กำหนดไว้ในโปรแกรม โดยที่ในการเล่นสามารถเล่นผ่าน internet ได้ด้วย

2. ในส่วนของสนามรบ จะเป็น output ที่ได้จากการคำนวณ สถานการณ์ต่างๆที่เกิดขึ้นโดยที่สถานการณ์ต่างๆ การเคลื่อนไหวของหุ่น การจู่โจมรวมถึงการเลือกแผนในการเล่นนั้นจะถูกเปลี่ยนโดยการคำนวณโดยเครื่องคอมพิวเตอร์ทั้งสิ้นโดย output ที่ได้นี้จะแสดงรูปร่างหน้าตาของหุ่นที่เกิดจากส่วนประกอบที่ถูกเลือก

1.4 ประโยชน์ที่คาดว่าจะได้รับ

- มีความรู้ความชำนาญในเรื่องของการเขียนโปรแกรมเชิงวัตถุและการเขียนโปรแกรมติดต่อผ่าน internet
- เพื่อเป็นแนวทางในการศึกษาและพัฒนาการเขียนเกมที่มีแนวคิดแบบเชิงวัตถุต่อไป
- เพิ่มทักษะในการคิดและวางแผนของผู้เล่นในการเลือกส่วนประกอบที่เหมาะสมของหุ่นยนต์
- เพื่อความสนุกสนาน ผ่อนคลาย และความบันเทิง
- มีการแลกเปลี่ยนความคิดเห็นระหว่างผู้เล่น

1.5 ขั้นตอนในการดำเนินงาน

1. ศึกษาการทำงานของภาษา java ที่เป็นภาษาเชิงวัตถุ ศึกษาการออกแบบภาษาเชิงวัตถุและศึกษาการเขียนโปรแกรมเพื่อการติดต่อสื่อสารกันผ่าน internet
2. วิเคราะห์และออกแบบการทำงานของเกม โดยแบ่งเป็นโครงสร้างและความสัมพันธ์ของส่วนประกอบต่างๆที่ประกอบขึ้นเป็นหุ่นยนต์ สถานการณ์ในการต่อสู้ระหว่างหุ่นยนต์ 2 ฝ่าย
3. เขียนโปรแกรมในส่วนต่างๆที่ได้ออกแบบไว้ รวมทั้งเขียนโปรแกรมในการติดต่อผ่าน internet
4. ทดลองเล่นเกมเพื่อทดสอบการทำงานของเกมส์ว่าเป็นไปตามจุดประสงค์ที่วางไว้หรือไม่
5. แก้ไขข้อบกพร่องและตกแต่งรายละเอียดต่างๆให้สมบูรณ์

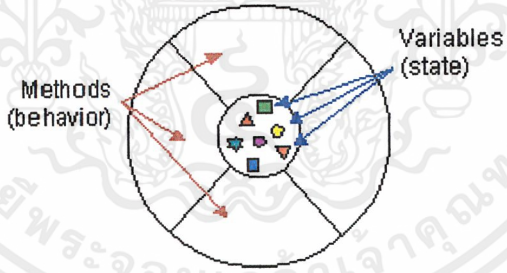
บทที่ 2
ทฤษฎีและวรรณกรรมที่เกี่ยวข้อง

2.1 ทฤษฎีของการโปรแกรมเชิงวัตถุ (Object Oriented Methodology)

เนื่องจากในการพัฒนาของซอฟแวร์ในระบบเก่าเกิดปัญหาขึ้นในหลายขั้นตอนของการพัฒนา ซึ่งทำให้ผู้พัฒนาต้องเสียค่าใช้จ่ายในการพัฒนาและค่าบำรุงระบบค่อนข้างสูงมากเมื่อเทียบกับผลตอบแทนที่ได้ เนื่องจากสามารถใช้ได้เฉพาะงานนั้น ๆ เท่านั้น ไม่สามารถนำกลับมาใช้ได้ใหม่อีก นอกจากนี้เมื่อ user ต้องการเปลี่ยนความต้องการเกี่ยวกับการทำงานของระบบที่ทำเสร็จแล้วก็ยังเป็นการยุ่งยากเพราะต้องไปเริ่มต้นใหม่อีกจึงมีผู้เชี่ยวชาญหลายท่านได้ทำการคิดค้นหาแนวทางที่จะแก้ปัญหาเหล่านี้และพบว่า Object Technology เป็นแนวทางหนึ่งในการพัฒนาซอฟแวร์ที่มีประสิทธิภาพและในขณะนี้ได้เป็นที่นิยมในหมู่ผู้พัฒนาระบบในหลาย ๆ องค์กร

2.1.1 ทฤษฎีของการโปรแกรมเชิงวัตถุ (Object Oriented Methodology)

คำว่า "Object " (Real World Object) โดยทั่วไปจะหมายถึง วัตถุใด ๆ จะมีสิ่งสำคัญคือ ลักษณะ,สถานะ และ พฤติกรรม (Attribute, State, Behavior) ส่วน Software Object จะพยายามเลียนแบบ Real World Object โดยแบ่งออกเป็น 2 ส่วน คือ ส่วนที่เป็นสถานะ (Data and variable) และพฤติกรรม (Method, Process, Operation)

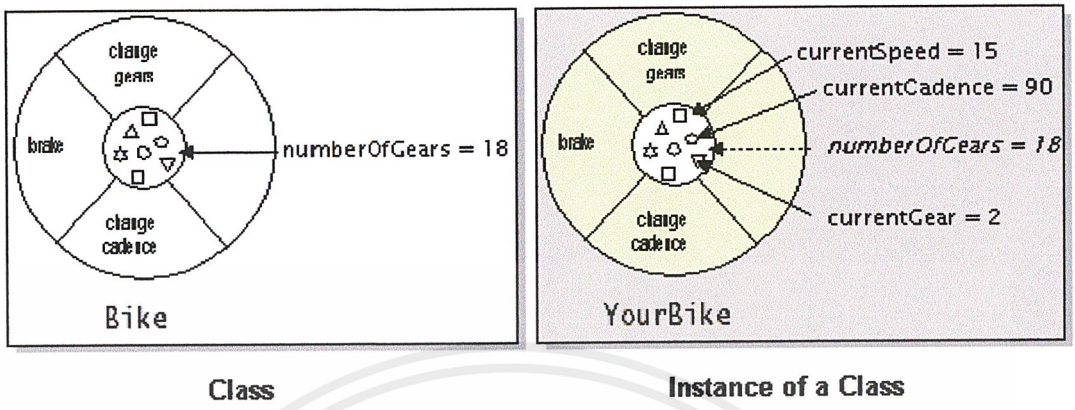


รูปที่ 2.1 แสดงองค์ประกอบของ Object

Object An object เป็น collection ของ data (Attribute, properties) และ function logic ซึ่ง data จะบอกถึงคุณสมบัติหรือสถานะของ object และ Method จะบอกถึงพฤติกรรมต่าง ๆ ของ object นั้น ๆ โดย Method จะแบ่งเป็น 2 ประเภท คือ interface method ซึ่งเป็น method ที่ได้ถูกใช้ได้จาก Object อื่น และ internal method ซึ่งเป็น method ที่จะถูกเรียกใช้ได้เฉพาะภายใน Object ที่เป็นเจ้าของเท่านั้น

และยังมีอีกคำหนึ่งที่ควรทราบคือ "Class" เป็นพิมพ์เขียวของ Object ที่ไม่สามารถนำมาใช้ได้โดยตรง โดยจะมีการบอกถึง method ที่ใช้ได้โดย Object และมีการแสดง data type ที่บอกถึง

สถานะของ Object โดยยังไม่ระบุค่าใน Attribute แต่ละตัวซึ่งถ้าเป็น Object จะมีการระบุค่าของ Attribute และ method เหมือน Class ของมัน



รูปที่ 2.2 แสดงความสัมพันธ์ระหว่าง Class และ Object

2.1.2 **คุณสมบัติของ** Object

2.1.2.1 **การปกป้องข้อมูลภายใน** (Encapsulation/Information hiding)

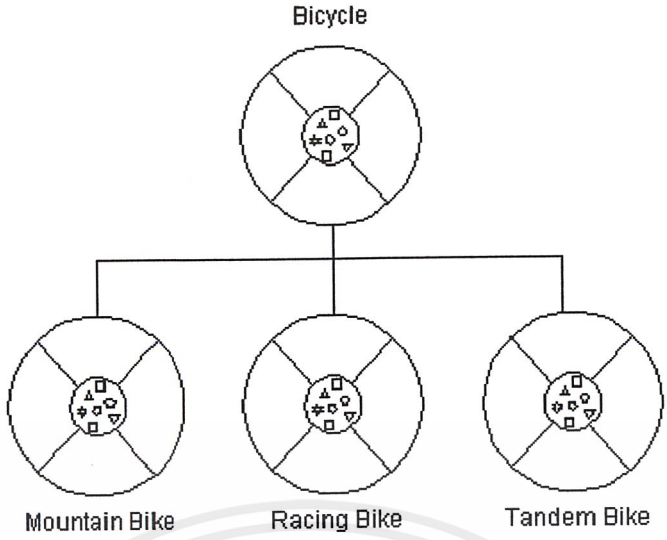
ทฤษฎีนี้ถูกคิดค้นโดย James Rambaugh ซึ่งหมายถึง การแยกลักษณะภายนอก (Interface) ซึ่งสามารถติดต่อกับ Object อื่น ๆ ออกจากส่วนที่ implement ภายในของ Object ซึ่งจะถูกใช้ได้เฉพาะตัว Object นั้น ๆ มีข้อดี คือ เพิ่มความสามารถในการปกป้องข้อมูลโดยไม่ให้ Object อื่นเข้ามาทำการเปลี่ยนแปลงข้อมูลก่อนได้รับอนุญาตทั้งนี้ตัว Object ควรรู้และทำการเปลี่ยนค่าด้วยตนเองซึ่งเป็นผลให้ maintain object ง่ายขึ้น

2.1.2.2 **การถ่ายทอดคุณสมบัติ** (Inheritance)

คือหลักการในการที่ Object ทุก ๆตัวใน generalized collection ได้มีการใช้ข้อมูล (structure) และพฤติกรรม (behavior) ร่วมกันซึ่งจะมีความสัมพันธ์แบบ is- a relationship โดยเรียก class ที่อยู่เหนือกว่าว่า parent class ซึ่งจะถ่ายทอดคุณสมบัติทั้ง Attribute และ method มายัง class ที่ต่ำกว่าเรียกว่า subclass ซึ่งจะมี Attribute และ method เพิ่มเติมจาก parent class มีข้อดีคือ

- เพิ่ม Consistency เพียงแค่เปลี่ยนเป็น Method และAttribute ที่ Super class ซึ่งเป็นผลให้ค่าที่ subclass เปลี่ยนไปด้วย

- เป็นการส่งเสริมการนำ Object กลับมาใช้ใหม่



รูปที่ 2.3 แสดงการสืบทอดคุณสมบัติ (Inheritance)

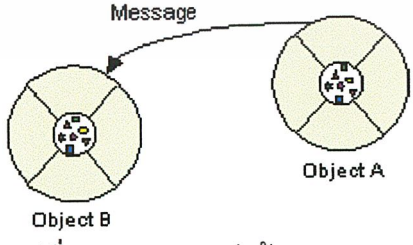
2.1.2.3 **การแสดงคุณสมบัติของวัตถุ** (Polymorphism)

อาจหมายถึง "having many forms" โดยการส่ง message เดียวกันให้กับ Object ที่ต่างกัน เป็นผลทำให้ Object แสดงพฤติกรรมที่แตกต่างกัน มีข้อดีคือ เป็นการสนับสนุนนำโปรแกรมกลับมาใช้ใหม่และสามารถเปลี่ยนแปลงซอฟต์แวร์ได้ในระหว่างการพัฒนา

2.1.3 **การติดต่อกันระหว่าง** object (Message Passing)

คือการที่ Object ติดต่อกันด้วยการส่งข้อความ (message) ถึงกันและกันซึ่งข้อความจะประกอบด้วยจุดหมายปลายทาง (destination) ของข้อความนั้นและข้อมูลที่สำคัญ (argument หรือ parameter)

Message Passing เปรียบได้กับ function call หรือ procedure call ที่มีใน structured programming โดยผ่าน Interface method ของ Object นั้น ๆ มีผลทำให้ Object ที่เป็นผู้รับข้อความ (received object) นั้นกระทำการอย่างใดอย่างหนึ่ง



รูปที่ 2.4 แสดงการส่งข้อความ (message) จาก Object A ไปยังObject B

2.1.4 การวิเคราะห์และออกแบบเชิงวัตถุ (Object Oriented Analysis and Design: OOAD)

ปัจจุบันเราอยู่ในโลกที่ซับซ้อนงานแต่ละงานจะมีความยุ่งยาก วัตถุหนึ่ง ๆ อาจเกิดจากการรวมของวัตถุต่าง ๆ ได้ วัตถุต่าง ๆก็สามารถมีความสัมพันธ์ต่อกันได้ ซึ่งวัตถุเหล่านี้อาจจะเป็นคน, สัตว์, สิ่งของ งานต่างๆ หรือจะเป็นสิ่งที่เราสนใจในการวิเคราะห์และออกแบบระบบอาจทำให้เกิดความสับสน ยังไม่มีเทคนิคในการวิเคราะห์และออกแบบระบบใดที่จะช่วยให้สามารถวิเคราะห์ลักษณะที่สำคัญจากโลกรอบตัวเรา ให้อยู่ในรูปของนามธรรม (Abstract) ให้มากที่สุด และทำให้สามารถจำแนกวัตถุต่าง ๆ ออกเป็นลำดับชั้นตามความสัมพันธ์และรูปแบบของมัน ซึ่งการวิเคราะห์และออกแบบระบบเชิงวัตถุจะเป็นวิธีที่ช่วยให้สามารถแปลงโลกแห่งความเป็นจริงเป็นซอฟต์แวร์ได้ง่ายมากขึ้น

การวิเคราะห์และออกแบบระบบเชิงวัตถุ คือการกำหนดแนวทางการปฏิบัติ (process) และสัญลักษณ์ (Notation) ที่ใช้ในการวิเคราะห์และออกแบบซอฟต์แวร์ ซึ่งจะอ้างอิงตามหลักการของวัตถุ (Object) คือ การรวมคุณลักษณะ (Attribute) และหน้าที่การทำงาน (Operation) ไว้ด้วยกัน การวิเคราะห์และออกแบบระบบเชิงวัตถุนั้น มีการกำหนดมาตรฐานขั้นตอนการปฏิบัติในการวิเคราะห์และออกแบบระบบ ซึ่งในมาตรฐานจะต้องมีการนิยามคำศัพท์ต่าง ๆ เพื่อความเข้าใจที่ตรงกัน สร้างมาตรฐานของโมเดล (Model) และสัญลักษณ์ (Notation) ที่ใช้ในการวิเคราะห์และออกแบบระบบ มีนักวิชาการเป็นจำนวนมากได้กำหนดมาตรฐานต่าง ๆ ซึ่งเป็นที่รู้จักดี เช่น Object Oriented Software Engineering : OOSE โดย Jacobson, Object Modeling Technique : OMT โดย Rumbaugh, OO analysis and design โดย Coad และYourdon เป็นต้น มาตรฐานต่าง ๆ ก็จะมีความเหมือนหรือแตกต่างกันไป แต่โดยรวมแล้วก็ต้องประกันได้ว่าทำงานได้จริงและถูกต้อง ในแต่ละ Phase การทำงานจะต้องสามารถคาดคะเนได้ว่าผลลัพธ์ที่ได้คืออะไร การทำงานถ้ามีความผิดพลาดเกิดขึ้นจะต้องกลับไปหาได้ว่าความผิดพลาดที่เกิดขึ้นอยู่ที่ขั้นตอนใด มาตรฐานที่ใช้ต้องสามารถรองรับได้ทั้งระบบงานเล็ก ๆ จนถึงระบบงานที่มีความซับซ้อนมาก ๆ ได้ โดยสรุปแล้วการวิเคราะห์และออกแบบระบบเชิงวัตถุมีโมเดลที่ใช้อยู่ 3 แบบ คือ

2.1.4.1 ฟังก์ชันนอลโมเดล (Functional Model) ฟังก์ชันนอลโมเดล เป็นโมเดลที่ใช้ในการแสดงความต้องการของระบบทั้งหมดช่วยในการอธิบายรายละเอียดหลัก ๆ ภายในวัตถุ แสดงให้เห็นการไหลของข้อมูลในแต่ละการทำงานโดยจะสนใจเพียงแค่ว่ามีงานอะไรบ้างที่ต้องทำจะยังไม่สนใจว่างานนั้น ๆ ทำอย่างไร เช่นในการพิมพ์รายงานสรุปประจำเดือน จะต้องรู้ว่างานนี้ใครเป็นผู้รับผิดชอบรายงาน รูปร่างหน้าตาของรายงานเป็นอย่างไร มีข้อมูลอะไรบ้างที่สนใจ และเมื่อออกรายงานแล้ว ใครเป็นผู้ที่รับรายงานนั้นเป็นต้น เครื่องมือที่ใช้ในการแสดงความต้องการ

ของระบบทั้งหมดในลักษณะที่ผู้ใช้งานสามารถเข้าใจได้ง่าย โดยจะถูกนำไปใช้ต่อไปใน Phase ต่าง ๆ ของการวิเคราะห์และออกแบบระบบ

2.1.4.2 **ออบเจ็กโมเดล** (Object Model) ออบเจ็กโมเดลเป็นโมเดลที่ใช้ในการแสดงโครงสร้างของระบบ โดยจะแสดงในรูปของคลาส (Class) ต่าง ๆ พิจารณาจากความต้องการของระบบที่แสดงอยู่ในฟังก์ชันนอลโมเดลเครื่องมือที่ใช้ในการแสดงโครงสร้างของระบบจะมีคอมโพเนนท์ไดอะแกรม (Component Diagram) และคลาสไดอะแกรม (Class Diagram) ซึ่งคอมโพเนนท์ไดอะแกรมแสดงให้เห็นถึงความสัมพันธ์ระหว่างคอมโพเนนท์ต่าง ๆ ภายในระบบ คลาสไดอะแกรมแสดงให้เห็นถึงคุณลักษณะ (attribute) คือ ข้อมูล (data) หรือตัวแปร (variable),หน้าที่การทำงาน (Operation)คือ เมธอท (method)ภายในคลาส และความสัมพันธ์ระหว่างคลาสต่าง ๆ ภายในระบบ

2.1.4.3 **ไดนามิคโมเดล** (Dynamic Model) ไดนามิคโมเดล เป็นโมเดลที่ใช้ในการแสดงถึงการทำงานระหว่างออบเจค(Object) ต่าง ๆ ตามการส่งข้อความ (Message) หรือเมื่อเหตุการณ์ (Event) ต่าง ๆ ได้เกิดขึ้น ออบเจคในที่นี้หมายถึงอินสแทนซ์ (Instance) ที่สร้างขึ้นจากคลาสที่ได้ออกแบบไว้ใน ออบเจคโมเดล โดยที่แต่ละออบเจคมีคุณสมบัติ และพฤติกรรมเช่นเดียวกับคลาสต้นแบบ ในการทำงานของระบบจะประกอบขึ้นจากการส่งข้อความไปมาระหว่างออบเจคเหล่านั้น เมื่อมีการทำงานไปเรื่อย ๆแล้วออบเจคอาจจะมีการเปลี่ยนสถานะ (State) ไปตามเหตุการณ์ที่เกิดขึ้นได้ เพื่อให้เป็นตามที่กำหนดไว้ในฟังก์ชันนอลโมเดล เครื่องมือที่ใช้คือซีเควนซ์ไดอะแกรม (Sequence Diagram) และสเตทไดอะแกรม (State Diagram) ซีเควนซ์ไดอะแกรมแสดงให้เห็นถึงลำดับขั้นตอนการทำงาน เมื่อมีเหตุการณ์หนึ่ง ๆ ขึ้นแล้วออบเจคต่าง ๆ มีการทำงานต่อไปอย่างไร วัตถุประสงหลักของซีเควนซ์ไดอะแกรม คือเพื่อให้ผู้เขียนโปรแกรมสามารถนำไปศึกษาและพัฒนาโปรแกรมหรือขยายขีดความสามารถของโปรแกรมต่อไป ส่วนสแตทไดอะแกรม เป็นไดอะแกรมที่แสดงให้เห็นถึงสถานะทั้งหมดที่เป็นไปได้ และเหตุการณ์ที่ทำให้เกิดการเปลี่ยนสถานะของออบเจคแต่ละตัว

2.1.5 **แนวความคิด** Object-Oriented Programming

เมื่อเริ่มต้นยุคคอมพิวเตอร์ การโปรแกรมทุกอย่างทำด้วยภาษาเครื่องเท่านั้น จึงทำให้การใช้งานคอมพิวเตอร์จำกัดอยู่กับผู้ที่เข้าใจโครงสร้างของคอมพิวเตอร์เป็นอย่างดี แต่เมื่อมีภาษา Fortran เป็นภาษาสำหรับโปรแกรมภาษาแรกที่มีคอมไพเลอร์และใช้งานได้จริง คอมพิวเตอร์ก็ถูกใช้งานอย่างแพร่หลายโดยบุคคลในสาขาต่างๆ โดยไม่จำเป็นต้องมีความรู้เกี่ยวกับคอมพิวเตอร์ชั้นสูง อย่างไรก็ตามเทคนิคการสร้างคอมไพเลอร์ในตอนนั้นยังมีข้อจำกัดอยู่มาก ภาษา Fortran จึงถูกออกแบบมาอย่างใกล้เคียงกับภาษาเครื่อง เพื่อให้ง่ายต่อการแปลภาษา และทำงานได้เร็ว แต่ส่งผลให้การสร้างโปรแกรมใหญ่ที่สลับซับซ้อนเป็นไปด้วยความยากลำบาก แนวโน้มการ

ออกแบบภาษาในยุคนั้นจึงหันไปสู่การสร้างภาษาที่เหมาะกับการใช้งานเฉพาะด้านอย่างเช่น ภาษา Cobol สำหรับงานธุรกิจ และภาษา Lisp สำหรับงานการสัญลักษณ์ เป็นต้น

ต่อมาภาษา Algol ถูกสร้างขึ้นเพื่อเป็นภาษาสำหรับโปรแกรมทั่วไป (general purpose language) เป็นภาษาแรก โดยเสนอกลไกสำหรับสร้างโครงสร้างข้อมูล (data structures) และประโยคที่เป็นโครงสร้าง (structured statement) เพื่อให้ง่ายต่อการนำโปรแกรมจำนวนมากมาประกอบกันเป็นโปรแกรมขนาดใหญ่ขึ้นและยังช่วยสนับสนุนความคิดในการเขียนโปรแกรมเป็นหน่วยย่อยๆซึ่งทำให้ง่ายทั้งการสร้างและจัดการ วิธีนี้ได้รับความนิยมอย่างแพร่หลาย กลายเป็นแม่แบบของหลายภาษาเช่น PL/I, Pascal, C, Modula-2, Ada และอื่นๆอีกมากมาย ภาษากลุ่มนี้เรียกว่า imperative programming โปรแกรมที่เขียนขึ้นแบบ imperative มักใช้ทรัพยากรของเครื่องคอมพิวเตอร์อย่างมีประสิทธิภาพและทำงานเร็ว แต่ก็ใช้ได้กับปัญหาที่ไม่ใหญ่มากนัก เมื่อมีปัญหาที่มีความซับซ้อนมากขึ้นก็เริ่มปรากฏความยุ่งยากเกินกว่าจะจัดการได้อย่างมีประสิทธิภาพ จึงต้องมีการค้นหาแนวคิดอื่นและภาษาสำหรับสร้างโปรแกรมแบบใหม่ ที่สามารถจัดการกับโปรแกรมขนาดใหญ่ได้อย่างมีประสิทธิภาพมากกว่า

ในปี 1967 ภาษา Simula67 ถูกพัฒนาขึ้นในประเทศ Norway โดยนำรูปแบบมาจากภาษา Algol และเพิ่มกลไกที่ช่วยในการเขียนโปรแกรมสำหรับจำลอง การทำงานของระบบใดๆ แม้ภาษา Simula67 ไม่ได้รับความนิยมเท่าที่ควร แต่กลไกของภาษาที่เกี่ยวกับการจำลองระบบได้รับความสนใจเป็นอย่างมากและถูกนำไปพัฒนาเพื่อใช้สำหรับสร้างโปรแกรมขนาดใหญ่

โดยทั่วไปในระบบที่ซับซ้อน จะประกอบด้วยวัตถุ หรือมองเป็นระบบย่อย จำนวนมากที่ทำงานเกี่ยวข้องกัน การสร้างโปรแกรมที่คำนวณผลลัพธ์ของวัตถุทั้งหมดออกมาโดยตรง จะยากกว่าการสร้างโปรแกรมที่จำลองการทำงานของแต่ละวัตถุในปัญหานั้น แนวคิดในการสร้างโปรแกรมที่ประกอบขึ้นจากวัตถุในปัญหานั้น แนวคิดในการสร้างโปรแกรมที่ประกอบขึ้นจากวัตถุเป็นหลักนี้เรียกว่าโปรแกรมเชิงวัตถุ (object oriented programming) และภาษาที่มีกลไกสนับสนุนการสร้างวัตถุจะเรียกว่าภาษาเชิงวัตถุ (object oriented languages)

ในราวต้น 1970s กลุ่มนักค้นคว้าของบริษัท Xerox พัฒนาภาษา Smaltalk ขึ้นเป็นภาษาที่เน้นการโปรแกรมเชิงวัตถุ อย่างแท้จริงเป็นภาษาแรก (ภาษา Simula67 มีการโปรแกรมเชิงวัตถุเป็นเพียงส่วนต่อเติมของภาษา) พร้อมกับการสร้างโปรแกรมเพื่อเป็นสภาพเป็นสภาพแวดล้อมช่วยในการเขียนโปรแกรมเชิงวัตถุด้วย แต่ผลงานนี้จำกัดอยู่แต่ในการค้นคว้า จนกระทั่งประมานกลาง 1980s บริษัท Xerox ตั้ง บริษัทลูกชื่อ ParcPlace System ขึ้นมาเพื่อพัฒนาภาษา Smaltalk ให้ใช้งานได้ในเชิงการค้า และได้รับความสนใจอย่างมาก ประกอบกับชั่งเวลานั้น ภาษา C กำลังได้รับความนิยมอย่างสูง จึงมีการพัฒนาขยายภาษา C ไปสู่การโปรแกรมเชิงวัตถุ เช่นภาษา Objective-C พัฒนาโดย Brad Cox จากบริษัท Stepstone Corporation และที่ประสบความ

สำเร็จอย่างสูงก็คือภาษา C++ พัฒนาขึ้นโดย Bjarne Stroustrup จากบริษัท AT&T ซึ่งในปี 1989 มีการปรับปรุงใหม่ประกาศเป็น C++ รุ่น 2.0 และใช้เป็นมาตรฐานในหลาย platform

เมื่อเข้าสู่ 1990s คอมพิวเตอร์ที่มีทรัพยากรพอจะทำงานโปรแกรมเชิงวัตถุได้อย่างมีประสิทธิภาพ เริ่มมีราคาถูกลงและเป็นที่แพร่หลายมากขึ้น แนวคิดของการโปรแกรมเชิงวัตถุจึงถูกทดสอบและถูกใช้งานจริงอย่างแพร่หลายมากขึ้น เพียงไม่นาน คนในวงการอุตสาหกรรมซอฟต์แวร์ก็เห็นประโยชน์และยอมรับในประสิทธิภาพของโปรแกรมเชิงวัตถุทำให้ความต้องการโปรแกรมเชิงวัตถุและเครื่องมือช่วยในการโปรแกรมมีมากขึ้น สังเกตได้ว่ากว่า 90% ของภาษาที่ถูกสร้างขึ้นในยุค 1990s นี้เป็นภาษาเชิงวัตถุ แม้แต่ภาษาที่มีมานานแล้วอย่างเช่น Fortran, Ada, Modula-2 และ Prolog ก็ถูกนำมาปรับปรุงใหม่ให้กลายเป็นภาษาที่สนับสนุนการโปรแกรมเชิงวัตถุเช่น Fortran90, Ada95, Modula-3 และ Prolog II

ภาษาที่ปรับปรุงมาจากภาษาที่ได้รับความนิยมมาก่อนจะได้เปรียบตรงที่มีผู้ใช้คุ้นเคยกับภาษาอยู่แล้วส่วนหนึ่ง ซึ่งไม่ต้องเรียนรู้จากภาษาใหม่ทั้งหมด แต่จะประสบปัญหา เนื่องจากกลไกการโปรแกรมเชิงวัตถุในภาษาประเภทนี้ มักถูกออกแบบเป็นเพียงสิ่งต่อเติมของภาษา ซึ่งผู้เขียนโปรแกรมจะใช้หรือไม่ใช้ก็ได้ ภาษาประเภทนี้จึงยังคงใช้เขียนโปรแกรมในแนวคิดแบบเดิมได้โดยไม่ต้องคิดในแบบเชิงวัตถุ ตัวอย่างคือ มีผู้ใช้ภาษา C++ จำนวนมากที่ไม่เคยใช้ประโยชน์ของการโปรแกรมเชิงวัตถุเลย ถือเป็นการใช้เครื่องมือไม่ถูกกับงาน เนื่องจากภาษา C++ มี overhead ทั้งในการคอมไพล์และในขณะทำงานสูงกว่าในภาษา C ดังนั้นหากเรายังคิดและเขียนโปรแกรมแบบ imperative ก็ควรใช้ภาษา C จะได้โปรแกรมที่มีประสิทธิภาพสูงกว่า

ในระยะหลังนี้มีภาษาเชิงวัตถุหลายภาษาที่พยายามออกแบบให้การคิดเชิงวัตถุเป็นแกนหลักของภาษา ไม่ใช่ส่วนต่อเติมจากแนวคิดเดิม วิธีนี้จะทำให้ผู้ใช้ต้องคิดและเขียนโปรแกรมเป็นเชิงวัตถุทั้งหมด ตัวอย่างเช่น Smalltalk และภาษา Eiffel ถูกออกแบบมาโดยเน้นการคิดแบบเชิงวัตถุอย่างผู้ใช้หลีกเลี่ยงไม่ได้ นั้นคือไม่สามารถใช้งานภาษานี้ได้เลยหากไม่คิดอย่างเชิงวัตถุ ข้อดีก้อคือผู้ใช้ต้องคิดและเขียนโปรแกรมเป็นเชิงวัตถุตั้งแต่พื้นฐานขึ้นมาเพื่อจะได้สานต่อขึ้นมาเป็นโปรแกรมที่ได้ประโยชน์จากความคิดเชิงวัตถุอย่างแท้จริง แต่ข้อเสียก็คือผู้ใช้ต้องผ่านการเรียนรู้และปรับเปลี่ยนแนวคิดเป็นแบบวัตถุ ซึ่งต้องใช้เวลา จึงทำให้ภาษาประเภทนี้เป็นที่แพร่หลายได้ช้ากว่า อีกทั้งความคิดเชิงวัตถุยต้องถูกนำมาใช้ตั้งแต่ขั้นตอนการออกแบบ ไม่ได้จำกัดอยู่กับการโปรแกรมเท่านั้น จึงต้องศึกษาเกี่ยวกับการออกแบบระบบแบบเชิงวัตถุด้วย ถือว่าเป็นการลงทุนระยะยาวที่ต้องใช้เวลามาก ผู้ใช้บางคนจึงคิดว่าภาษาเชิงวัตถุไม่ได้สะดวกในการใช้งานเมื่อเปรียบเทียบกับภาษาที่ใช้เวลาในการเรียนรู้น้อยกว่า โดยยึดถือการสร้างโปรแกรมขนาดเล็กๆเป็นข้ออ้างนั้นเป็นการเปรียบเทียบที่ไม่ยุติธรรมเพราะภาษาเชิงวัตถุถูกสร้างขึ้นมาใช้งานขนาดใหญ่ มีกลไกที่สนับสนุนการทำงานทีมผู้เขียนโปรแกรมหลายคน และทำงานเป็นเวลานาน หากนำมาใช้สร้าง

โปรแกรมขนาดเล็กอาจจะงุ่มง่ามกว่าภาษาที่ออกแบบมาโดยตรงสำหรับงานขนาดเล็กที่ไม่ต้องถูกดูแลรักษาให้ใช้งานได้นาน

ปัญหาของภาษาเชิงวัตถุคือมี overhead ในระหว่างโปรแกรมทำงานสูง หากเขียนโปรแกรมทั้งหมดเป็นแบบเชิงวัตถุจะทำงานช้าและใช้หน่วยความจำสิ้นเปลืองมากจนแม้แต่เครื่องคอมพิวเตอร์ประสิทธิภาพสูงในปัจจุบันก็ไม่สามารถทำงานโปรแกรมเชิงวัตถุขนาดใหญ่ได้เร็วพอ ในทางปฏิบัติ ภาษา Java จึงเลือกทางออกโดยเน้นให้ใช้ความคิดแบบเชิงวัตถุเป็นส่วนใหญ่ แต่ยกเว้นในการจัดการเกี่ยวกับชนิดข้อมูลพื้นฐานและการคำนวณทางคณิตศาสตร์ทั่วไปแบบ imperative ด้วยเหตุนี้ภาษา Java จึงถูกจัดเป็นภาษาเชิงวัตถุภาษาที่ยังไม่เขียนโปรแกรมแบบ imperative ก็ได้ แต่การออกแบภาษา Java ประสบความสำเร็จในการสนับสนุนให้ผู้เขียนโปรแกรมคิดแบบเชิงวัตถุอย่างมาก นั้นคือหากผู้เขียนโปรแกรมคิดและสร้างโปรแกรมแบบเชิงวัตถุจะทำให้สร้างโปรแกรมได้ง่ายกว่าการไม่คิดแบบเชิงวัตถุ โดยไม่คุ้มค่าเลยถ้าจะเรียนรู้ภาษา Java โดยไม่ใช้ประโยชน์ความคิดเชิงวัตถุ และเราจะเปรียบเทียบการเขียนโปรแกรมแบบ imperative กับ object oriented ที่มีความแตกต่างกันเพื่อให้เข้าใจในพื้นฐานของแนวคิดเชิงวัตถุ จึงขออธิบายความคิดในการเขียนโปรแกรมแบบ imperative ก่อน

2.1.5.1 การเขียนโปรแกรมแบบ Imperative programming

การเขียนโปรแกรมแนวความคิดแบบ imperative คือการสร้างโปรแกรมที่ประกอบไปด้วยลำดับของคำสั่งให้คอมพิวเตอร์ทำตามเพื่อแก้ปัญหานั้น การเขียนโปรแกรมแบบนี้เหมือนกับการสอนคอมพิวเตอร์ว่าต้องการทำอะไบ้าง เป็นวิธีที่ใกล้เคียงกับการโปรแกรมหน่วยประมวลผล (cpu) ทั่วไปในระดับภาษาเครื่อง การแปลภาษาจากภาษาระดับสูงเป็นภาษาเครื่องจึงทำได้ง่ายและโปรแกรมทำงานได้เร็ว ภาษาแบบ imperative จึงได้รับความนิยมอย่างกว้างขวางและเป็นแนวคิดของการเขียนโปรแกรมในภาษาที่ได้รับความนิยมจำนวนมาก ความคิดแบบ imperative ถูกพัฒนาให้มีประสิทธิภาพมากขึ้น โดยการสร้างโปรแกรมสำหรับแก้ปัญหาที่ต้องใช้บ่อยๆ เก็บไว้เป็น procedures เพื่อเพื่อสามารถเรียกใช้ได้ ในความหมายที่สูงขึ้น คือผู้ใช้ไม่จำเป็นต้องเข้าใจรายละเอียดการทำงานของ procedure นั้น เพียงแต่ทราบเงื่อนไขที่จะใช้งาน และผลที่จะได้ รับก็สามารถงาน procedure นั้นได้ การเขียนโปรแกรมด้วยวิธีนี้จึงมักถูกเรียกว่า procedural programming ซึ่งก็ยังคงเป็นความคิดแบบ imperative ในแง่ที่ผู้เขียนโปรแกรมต้องระบุขั้นตอนการเรียกใช้ procedure สำหรับการแก้ปัญหานั้นๆ เหมือนกับการสอนคอมพิวเตอร์แก้ไขปัญหาด้วย procedures ต่างๆ อยู่นั้นเองแต่เป็นการเอาความคิดในระดับที่สูงขึ้น ขบวนความคิดในการเขียนโปรแกรมแบบ imperative เพื่อแก้ปัญหาหนึ่ง อาจสรุปเป็นขั้นตอนได้ดังนี้

1. หาวิธีการแสดงข้อมูลที่เกี่ยวกับปัญหา เป็นพื้นที่ในหน่วยความจำ ด้วยตัวแปร หรือโครงสร้างข้อมูลที่เหมาะสม

2. กำหนดลำดับของคำสั่งที่เมื่อทำงานแล้วจะส่งผลข้างเคียงต่อตัวแปรที่เตรียมไว้ จนกว่าจะได้ค่าเป็นคำตอบของปัญหานั้น

การสร้างโปรแกรมที่ส่งผลข้างเคียงแก่ตัวแปลในลักษณะเช่นนี้ก่อให้เกิดปัญหาคือ เป็นการยากที่จะเข้าใจผลลัพธ์ของโปรแกรม เนื่องจากพฤติกรรมของโปรแกรมถูกเปลี่ยนไปเป็นคำสั่งของคอมพิวเตอร์ และผลลัพธ์ของโปรแกรมจะขึ้นกับลำดับของผลข้างเคียงที่เกิดขึ้น เราไม่สามารถทราบว่าเกิดอะไรขึ้นบ้างโดยการอ่านโปรแกรมเท่านั้น แต่จำเป็นต้องจำลองการทำงานของโปรแกรมจึงจะทราบว่ามีผลข้างเคียงใดเกิดขึ้นและเกิดขึ้นในลำดับใด ซึ่งเป็นเรื่องที่ลำบากมากในการโปรแกรมขนาดใหญ่ และจะยุ่งยากเข้าไปอีกหากโปรแกรมนั้นมีการดำเนินที่แตกต่างกันไปโดยขึ้นอยู่กับค่าของอินพุทของโปรแกรม หากเราจะสามารถตรวจสอบจนทราบลำดับของผลข้างเคียงที่เกิดขึ้นแล้ว ก็ยังเป็นการยากที่จะแปลไปสู่ผลลัพธ์ของโปรแกรม เพราะยังต้องตีความว่า ตัวแปรในโปรแกรมแทนข้อมูลในข้อมูลใดในปัญหา และคำสั่งในโปรแกรมแทนกิจกรรมใด ซึ่งเป็นความยุ่งยากอย่างมากเพราะการทำงานของคอมพิวเตอร์แตกต่างจากกิจกรรมที่มนุษย์ใช้ในการแก้ปัญหา

2.1.6 เปรียบเทียบการเขียนโปรแกรมแบบ Imperative V.S. Object oriented

ความแตกต่างระหว่างการโปรแกรมแบบ imperative กับการโปรแกรมเชิงวัตถุ สรุปได้เป็นหัวข้อดังต่อไปนี้

2.1.6.1 แนวความคิดแบบ Top-down V.S. Bottom-up ภาษา imperative สนับสนุนแนวความคิดในการสร้างโปรแกรมแบบ top-down ซึ่งทำการแตกระบบทั้งหมดออกเป็นระบบย่อย แล้วพิจารณาทีละระบบย่อย หากระบบย่อยนั้นง่ายพอก็เขียนโปรแกรมสำหรับระบบนั้นโดยตรง แต่หากยังเป็นระบบที่ซับซ้อนก็ใช้วิธีที่แตกระบบย่อยนั้นออกเป็นระบบย่อยอีกต่อหนึ่งและพิจารณาทีละระบบย่อยเช่นเดิม ทำแบบนี้ซ้ำไปเรื่อยๆ จนกว่าจะได้ระบบย่อยที่ง่ายพอจะเขียนโปรแกรมโดยตรงได้ทั้งหมด วิธีการนี้เป็นหลักเกณฑ์ที่ผูกแน่นกับโครงสร้างภาษา Pascal ที่สนับสนุนให้ผู้ใช้แตกโปรแกรมออกเป็น procedures ย่อยและ procedures ย่อยๆ ลงไปได้อย่างไม่จำกัด แต่จากการสร้างโปรแกรมขนาดใหญ่พบว่า procedures ที่สร้างขึ้นในระหว่างการสร้างโปรแกรมแบบ top-down มักจะไม่สามารถใช้ในที่อื่นที่ต้องแก้ปัญหาคล้ายๆกัน กล่าวอีกอย่างว่าโปรแกรมที่สร้างไว้แล้วมักจะไม่สามารถนำกับมาใช้ได้อีกซึ่งเรียกว่า อัตราการนำกลับมาใช้ (reuse) ต่ำ เนื่องจากความคิดในการสร้างโปรแกรมแบบ top-down มักจะทำให้ได้โปรแกรมที่สร้างขึ้นเพื่อใช้งานเฉพาะที่ เมื่อต้องการใช้เท่านั้น ไม่ได้สนับสนุนให้สร้างโปรแกรมที่เป็นสากลเพียงพอที่จะทำงานได้ในปัญหาเดียวกันแต่อยู่ในสถานะอื่น ดังนั้นเมื่อต้องการโปรแกรมสำหรับทำงานอย่างหนึ่ง แต่ถึงแม้จะเคยมีโปรแกรมสำหรับงานคล้ายๆกันนั้นมาแล้วก็ไม่สามารถนำมาใช้งานได้ ต้องสร้างโปรแกรมขึ้นมาใหม่ ทำให้มีโปรแกรมทำงานคล้ายๆกันเป็นจำนวนมากในระบบ

ซึ่งเสียทั้งพื้นที่ในการเก็บโปรแกรมและเวลาในการพัฒนาโปรแกรมที่ซ้ำซ้อนกัน ต่างจากโปรแกรมเชิงวัตถุที่มุ่งเน้นการสร้างโปรแกรมแบบ bottom-up มากกว่า นั่นคือการสร้างโปรแกรมที่เป็นพื้นฐานเพื่อนำไปประกอบขึ้น เป็นโปรแกรมที่ใหญ่ขึ้นในภายหลังภาษาเชิงวัตถุจึงต้องมีกลไกทางภาษาเพื่อสนับสนุนการสร้างโปรแกรม ให้มีความเป็นสากลที่จะนำไปใช้งานในสถานการณ์ต่างๆ

2.1.6.2 **การออกแบบโปรแกรมแบบ** Procedure oriented V.S. Object oriented ในการออกแบบโปรแกรมสำหรับระบบขนาดใหญ่เรามักจะแบ่งระบบนั้นออกเป็นระบบย่อยตามวัตถุหรือชิ้นส่วนที่ทำงานอยู่จริงในระบบ เพื่อความสะดวกในการกำหนดหน้าที่และพฤติกรรมของแต่ละระบบย่อย แต่ความคิดในการเขียนโปรแกรมแบบ imperative จะเน้นการสร้าง procedures เป็นหลัก โดยที่ procedures เป็นกลไกของภาษาที่ไม่เหมาะสำหรับจำลองวัตถุที่มีจริงๆ ในธรรมชาติ เนื่องจาก procedure มีช่วงชีวิตจำกัดเพียงแค่เฉพาะเมื่อโปรแกรมดำเนินอยู่ใน procedure นั้น ทำให้เกิดข้อจำกัดว่าเวลาใดเวลาหนึ่งจะเพียง procedure เดียวเท่านั้น ที่ทำงานอยู่ แต่ในความเป็นจริงระบบส่วนใหญ่จะมีวัตถุหลายๆ วัตถุทำงานไปพร้อมๆ กัน อีกทั้งในระบบทั่วไปจะมีวัตถุที่ถาวรเป็นส่วนใหญ่ไม่ใช่เป็นแบบ proceduresที่ถูกสร้างขึ้นและทำได้หลาย รอบระหว่างการทำงานของโปรแกรม การสร้างโปรแกรมขนาดใหญ่ด้วย imperative จึงยาก ลำบากมาก เปรียบกับภาษาเชิงวัตถุที่สามารถจำลองพฤติกรรมการทำงานของวัตถุได้ใกล้เคียงกว่า โดยใช้หน่วยของโปรแกรมที่เรียกว่าคลาส (class) ซึ่งมีได้ทั้งข้อมูลและฟังก์ชั่นสำหรับกำหนดพฤติกรรมของวัตถุประเภทหนึ่ง ในการใช้งานคลาสเราจะต้องสร้าง instance ของคลาสนั้น ซึ่งเป็นตัวอย่างหนึ่งของวัตถุประเภทนั้น คล้ายกับการสร้างตัวแปรของชนิดข้อมูลหนึ่งขึ้นมา ช่วงชีวิตของ instance จะเริ่มจากเมื่อถูกสร้างขึ้นไปจนจบขอบเขตของโปรแกรมที่สร้าง instance นั้นขึ้นมา ดังนั้นบาง instance อาจจะมีช่วงชีวิตตลอดระหว่างโปรแกรมทำงานก็ได้ หรือ instance หนึ่งอาจมีชีวิตคงอยู่ได้เสมอไม่ว่าโปรแกรมจะดำเนินผ่านไปที่ใด และที่เวลาหนึ่งอาจมีหลาย instance มีชีวิตอยู่พร้อมกันได้

2.1.6.3 **การโปรแกรมแบบ** Algorithms V.S. Behavior ความคิดส่วนใหญ่ของการเขียนโปรแกรมแบบ imperative คือการสร้าง algorithm ซึ่งหมายถึงลำดับของคำสั่งที่แก้ปัญหาหรือทำงานอย่างหนึ่ง สำหรับคำนวณผลลัพธ์ของการทำงานของระบบ การเขียนโปรแกรมจึงเน้นไปในการสร้าง algorithm ที่มีประสิทธิภาพในการแก้ ปัญหานั้น ถือเป็นงานที่ยุ่งยากกว่าการโปรแกรมเชิงวัตถุซึ่งจะจำลองโครงสร้างและพฤติกรรมของแต่ละวัตถุแล้วนำมาประกอบกันเพื่อทำงานให้ได้ผลลัพธ์ออกมา การโปรแกรมแบบนี้จึงเน้นไปในการ กำหนดพฤติกรรมของวัตถุโดยระบุว่าวัตถุประเภทหนึ่งจะมีข้อมูลใดบ้างเป็นสถานการณ์ทำงานของวัตถุและมีฟังก์ชั่นใดบ้างสำหรับทำการเปลี่ยนสถานะของวัตถุหรือทำการติดต่อโต้ตอบกับวัตถุอื่น

2.1.6.4 กลไกการจัดการข้อมูลแบบ Share global V.S. Information hiding ในภาษา imperative นั้น procedures ต่างๆ จะไม่สามารถรักษาสถานะของการทำงานไว้ได้ ซึ่งก็คือค่าของตัวแปรเฉพาะที่ใน procedure นั้น เมื่อโปรแกรมดำเนินออกจาก procedure ไปแล้ว ค่าเหล่านั้นจะถูกเก็บไว้เป็น global data ซึ่ง procedures ใดๆสามารถอ้างถึงและเปลี่ยนแปลงได้ทำให้เกิดปัญหาเกี่ยวกับผลข้างเคียงที่อันตรายเมื่อนำ procedures จำนวนมากมาทำงานร่วมกัน ภาษาเชิงวัตถุไม่สนับสนุนการมี global data และมีกลไกที่เรียกว่า information hiding ที่สามารถจำกัดการมองเห็นข้อมูลที่อยู่ภายในวัตถุ ข้อมูลหรือฟังก์ชันที่ถูกระบุเป็น private ซึ่งจะถูกอ้างถึงได้เฉพาะภายในคลาสเท่านั้น ส่วนข้อมูลหรือฟังก์ชันที่ถูกระบุเป็น public จะถูกอ้างถึงจากภายนอกคลาสได้ โดยปกติเราจะระบุ ให้ฟังก์ชันที่เป็นจุดเชื่อมต่อระหว่างคลาส จึงต้องเป็น public เท่านั้น ด้วยเหตุนี้ภาษาเชิงวัตถุ จึงมักจะเรียกฟังก์ชันว่า methods ซึ่งสามารถกำหนดให้เป็นมาตรฐาน ได้คล้ายกับขาของตัว IC (integrated circuit) จากที่ผ่านมาจะเห็นว่าอุตสาหกรรมฮาร์ดแวร์เจริญก้าวหน้าได้รวดเร็วกว่าว่าอุตสาหกรรมซอฟต์แวร์อย่างมาก นั่นเป็นเพราะเขาประสบความสำเร็จในการชิ้นส่วนอุปกรณ์ที่ต่างคนต่างผลิตมาทำงานร่วมกัน โดยการกำหนดหน้าที่ของขาต่างๆ ของตัว IC แต่ละเบอร์เป็นมาตรฐาน เมื่อผู้ใช้นำไปใช้ในเงื่อนไขที่กำหนดก็จะทำงานได้ แม้ว่าภายในตัว IC เบอร์ เดียวกันที่ต่างยี่ห้ออาจมีวงจรต่างกันหรือแม้แต่ในเทคโนโลยีที่ต่างกัน แต่เมื่อสร้างให้มีขาติดต่อกับ วงจรภายนอกตามมาตรฐานแล้ว ผู้ซื้อจะสามารถนำไปเสียบแทนตัวเดิมและทำงานได้เสมอ ข้อดีประการอื่นของวิธีการนี้คือ ผู้ใช้ไม่จำเป็นเข้าใจรายละเอียดการทำงานของ IC ตัวนั้นเลยก็สามารถใช้งานได้ส่วนผู้ผิดก็สามารถปรับปรุงเปลี่ยนแก้ไขวงจรของ IC ตัวนั้นได้โดยไม่ส่งผลกระทบต่อการใช้งานและเป็นการนำสิ่งที่ มีอยู่แล้วมาใช้งานอื่นในแบบ bottom-up ช่วยในการพัฒนาวงจร ที่มีความซับซ้อนมากกว่าเป็นไปอย่างมีประสิทธิภาพ คนในวงการซอฟต์แวร์ปรารถนาจะผลิตโปรแกรมด้วยวิธีนี้มานานแล้วแต่ติดที่ภาษา imperative ไม่สนับสนุนความคิดนี้ จนกระทั่งมีการพัฒนาเกี่ยวกับ data abstraction ขึ้นในภาษา Modula-2 และ Ada ให้มีกลไก information hiding เป็นส่วนประกอบหนึ่งจึงเริ่มมีความเป็นไปได้เกิดขึ้น

2.1.6.5 กลไกการจัดการข้อมูลแบบ Programs-data separathion V.S. Encapsulation ภาษา imperative จะแบ่งแยกระหว่างโปรแกรมกับข้อมูลอย่างชัดเจน โปรแกรมเป็นสิ่งที่ทำงานได้และเปลี่ยนแปลงไม่ได้ในระหว่างโปรแกรมทำงาน ส่วนข้อมูลเป็นสิ่งที่ถูกอ่านหรือเขียนได้ ถูกสร้างขึ้นหรือถูกทำลายได้ แต่ให้ทำงานไม่ได้ กล่าวอีกอย่างหนึ่งคือ โปรแกรมเป็นผู้กระทำ (active) ที่สามารถเปลี่ยนแปลงข้อมูลได้ ส่วนข้อมูลเป็นผู้ถูกกระทำ (passive) เท่านั้น การสร้างโปรแกรมในภาษาแบบนี้จึงมักแบ่งแยกขั้นตอนการกำหนดโครงสร้างข้อมูลออกจากขั้นตอน การกำหนดโปรแกรมที่จะจัดการข้อมูลนั้น การพัฒนาโปรแกรมขนาดใหญ่ในภาษาแบบนี้มักจะสร้าง procedures ที่ใช้บ่อยๆ เก็บไว้เป็น library และนำมาใช้โดยการ include หรือ import

เมื่อต้องการ จึงเป็นความยุ่งยาก เมื่อจะใช้งานเพราะต้องสร้างข้อมูลให้มีโครงสร้างและชนิดของข้อมูลตรงกับที่ procedure นั้นจะจัดการได้ ซึ่งต้องเสียเวลาศึกษารายละเอียดและเขียนโปรแกรมเพื่อสร้างข้อมูลแบบนั้น ปัญหาอีกแบบหนึ่งคือ หากมีโปรแกรมหนึ่งจัดการกับข้อมูลที่มีชนิดข้อมูลอย่างหนึ่งแล้ว เราก็สามารถใช้โปรแกรมนั้นจัดการข้อมูลใดๆที่มีชนิดข้อมูลแบบนั้นได้เสมอ ไม่มีทางจำกัดให้โปรแกรมนั้นใช้ได้กับเฉพาะข้อมูลบางตัว หรือในทำนองคล้ายๆกัน ไม่มีทางจำกัดให้ข้อมูลบางตัวถูกกระทำได้โดยเฉพาะโปรแกรมบางโปรแกรมเท่านั้นเป็นการเปิดโอกาสให้โปรแกรมหรือ ข้อมูลถูกนำไปใช้อย่างผิดจุด ประสงค์ไม่ว่าจะเป็นไปโดยจงใจหรือไม่ก็ตาม ภาษาเชิงวัตถุมีกลไก encapsulation สำหรับนำข้อมูลและ method ที่จะกระทำต่อข้อมูลนั้นมาผูกติดกันกลายเป็นโครงสร้างหนึ่งหน่วยซึ่งก็คือ คลาสนั้นเอง เมื่อสร้าง instance ของคลาสนั้นขึ้นจะมีทั้งโปรแกรมและข้อมูลพร้อมใช้งาน โดยไม่จำเป็นต้องเข้าใจโครงสร้างของข้อมูลที่อยู่ในคลาสนั้นเลย จึงไม่มีปัญหาเกี่ยวกับ method จะจัดการกับข้อมูลไม่ได้ อีกทั้งคลาสยังสามารถจำกัดให้ methods ทำงานได้เฉพระข้อมูลที่อยู่ในคลาสนั้นเท่านั้น จึงไม่มีปัญหาเกี่ยวกับการนำ method ไปจัดการกับข้อมูลที่ไม่สมควรถูกจัดการ

2.1.6.6 **กลไกการจัดการข้อมูลแบบ** Data transportation V.S. Communication with message ในการโปรแกรมแบบ imperative มักจะมองว่า procedures เป็นสิ่งที่อยู่กับที่และมีการติดต่อสื่อสารกันโดยการส่งข้อมูล ไปมาระหว่าง procedures ในลักษณะของพารามิเตอร์หรือค่าที่ส่งกลับมา หากข้อมูลมีขนานใหญ่มากโปรแกรมจะเสียเวลาไปในการเคลื่อนย้ายข้อมูลมาก กว่าการคำนวณเพื่อแก้ปัญหาจริงๆ การโปรแกรมเชิงวัตถุไม่ส่งเสริมให้มีการส่งข้อมูลขนาดใหญ่ไปมาระหว่าง instance แต่จะออกแบบโปรแกรมให้ข้อมูลส่วนใหญ่อยู่กับที่ภายใน instance และมีการส่งข้อมูลไปมาระหว่างกันให้น้อยที่สุดเท่าที่จำเป็น เพื่อให้โปรแกรมทำงานอย่างมีประสิทธิภาพ ภาษาเชิงวัตถุเรียกสิ่งที่ส่งไปมาระหว่าง instance ว่า ข่าวสาร (messages) ซึ่งก็คือพารามิเตอร์ที่ถูกส่งไปในการเรียก method และค่าที่ถูกส่งกลับมานั้นเอง เราถือ เป็นข้อควรปฏิบัติในการเขียนโปรแกรมเชิงวัตถุที่ดีคือ ข่าวสารที่ถูกส่งไปมานี้ควรมีขนาดเล็ก

2.1.6.7 **การส่งข้อมูลแบบ** Single flow V.S. Multiple objects ปัญหาอีกอย่างหนึ่งของภาษาแบบ imperative คือมีการดำเนินการของโปรแกรม (execution flow) เพียงเส้นทางเดียว ผู้เขียนโปรแกรมจะต้องละบุลำดับของคำสั่งในการแก้ปัญหาให้สามารถยืดออกเป็นเส้นตรงเพียงเส้นเดียวให้ได้ จึงไม่เหมาะกับการแก้ปัญหาบางชนิดและลำบากในการสร้างระบบขนาดใหญ่ เพราะในการนำโปรแกรมขนาดเล็กจำนวนมากมาประกอบขึ้นเป็นโปรแกรมขนาดใหญ่ จะต้องพยายามให้การคำเนินการของโปรแกรมย่อยสามารถร้อยต่อกันเป็นเส้นเดียวให้ได้ ซึ่งเป็นการยุ่งยากมากทั้งในการสร้างและแก้ไขเปลี่ยนแปลง แต่ในความคิดแบบเชิงวัตถุจะมี instance เป็นหน่วยหลักที่ยึดติดกันไว้ด้วย methods ซึ่งเป็นช่องทางในการส่งข่าวสาร หากเราเปลี่ยนแปลง

ระบบด้วยการนำ instance หนึ่งออกไป ก็เพียงแต่ต้องหา instance ที่มี methods สำหรับเชื่อมต่อกับ instance อื่นแบบเดียวกันมาใส่แทน ดังนั้นการนำวัตถุจำนวนมากมาประกอบกันเป็นระบบที่ใหญ่ขึ้น จึงทำได้ง่ายกว่าทั้งในการสร้างและปรับปรุงเงื่อนไข

2.1.7 หลักการพื้นฐานของ Object- Oriented Programming

วัตถุทั่วไปในธรรมชาติจะเก็บสถานะ (state) เป็นข้อมูลของตัวเองได้และต้องสามารถทำกิจกรรมบาง อย่างได้ เช่นการคำนวณและการเปลี่ยนแปลงข้อมูลของตัวเองหรือทำการติดต่อโต้ตอบกับวัตถุอื่น หากเราสมมุติว่านักเรียนคนหนึ่งเป็นวัตถุ เขาก็ควรจะมีชื่อ มีที่อยู่ มีผลการเรียนเป็นข้อมูลของตัวเอง และเขาน่าจะสามารถตอบคำถามอย่างเช่น ชื่ออะไร ผลการเรียนอยู่ระดับใดหรือเมื่อมีการประกาศผลสอบ เข้าก็น่าจะจำผลสอบได้ เป็นต้น ตัวแปลในภาษา imperative ไม่สามารถจำลองพฤติกรรมและการทำงานของวัตถุอย่างนี้ ภาษาเชิงวัตถุจึงเสนอคลาสขึ้นเป็นกลไกสำหรับกำหนดโครง สร้างและพฤติกรรมของวัตถุเมื่อต้องการจำลองพฤติกรรมของวัตถุนั้น เราจะต้องสร้างสิ่งที่เรียกว่า instance ของคลาสนั้น ซึ่งจะมีหน่วยความจำสำหรับเก็บข้อมูลของวัตถุและมี methods สำหรับกระทำต่อข้อมูลของมันเองหรือตอบโต้แลกเปลี่ยนข้อมูลกัน instances อื่น อาจกล่าวได้ว่า instances เป็นเหมือนกับตัวแปรพื้นฐานที่เก็บข้อมูลได้พร้อมกับมี operations ที่เกี่ยวข้องกับหน้าที่ของมันด้วย แต่ instance มีช่วงชีวิตต่างจากตัวแปรในภาษา imperative

คลาสคือโครงสร้างและพฤติกรรมของวัตถุประเภทหนึ่ง หากเปรียบ instance ในภาษาเชิงวัตถุก็เหมือนกับตัวแปลในภาษา imperative แล้วคลาสก็เหมือนกับชนิดของข้อมูล (type) นั้นเอง คลาสถูกสร้างขึ้นเพื่อกำหนด โครงสร้างของ instance ของวัตถุประเภทหนึ่ง เราจึงมีคลาสก่อนจะสร้าง instances ของคลาสนั้น และเมื่อมีคลาสหนึ่งแล้ว จะสร้าง instances ของคลาสนั้นกี่ instances ก็ได้ โดยที่ instance ของคลาสเดียวกันจะมีโครงสร้างเหมือนกันแต่อาจจะมีข้อมูลต่างกันไป ในการกำหนดคลาส เราต้องระบุว่าคลาสนั้นมีสมาชิก (member) ใดบ้างสมาชิกของคลาสมี 2 ประเภทคือ

- สมาชิกที่เป็นข้อมูลเรียกว่า data members หรือ field อาจเป็น ค่าคงที่ ตัวแปรของชนิดข้อมูลพื้นฐาน array หรือแม้กระทั้ง instances ของคลาส

- สมาชิกที่เป็นฟังก์ชันเรียกว่า methods members หรือเรียกสั้นๆว่า methods ซึ่งอาจเป็นได้ทั้ง procedures คือฟังก์ชันที่ไม่ส่งค่าออกมาหรือฟังก์ชันที่ส่งค่าออกมา

ในยุค 1970s มีผู้พบข้อสังเกตว่าชนิดข้อมูลที่สร้างขึ้นโดยผู้เขียนโปรแกรม (user define data type) โดยใช้ array หรือ structures ที่มีในภาษา imperative ทั้งหลายนั้นไม่สามารถใช้งานได้อย่างปลอดภัยเหมือนกับชนิดของข้อมูลพื้นฐานที่มากับภาษาเช่น int หรือ float จึงมีการศึกษาและเสนอว่าชนิดของข้อมูลที่ใช้งานได้อย่างปรอดภัยนั้นควรมีลักษณะดังนี้คือ รายละเอียดของการเก็บแสดงข้อมูลนั้นต้องไม่มีความสำคัญต่อการใช้งานชนิดข้อมูลนั้นเนื่อง จากข้อมูลภายในจะ

ไม่ถูกอ้างถึงได้โดยตรง แต่ต้องการจัดการผ่านทาง operations ที่กำหนดไว้เท่านั้น ชนิดของข้อมูลที่มีลักษณะดังกล่าวนี้จะถูกเรียกว่า abstract data type และจากการศึกษาก็พบว่าภาษาที่จะสามารถสร้างชนิดข้อมูลแบบนี้ได้ต้องมีกลไกทางภาษาสองอย่างดังต่อไปนี้

1. Encapsulation สำหรับนำข้อมูลกับ operations มาผูกติดกันเป็นหน่วยหนึ่ง

2. Information hiding เพื่อกำหนดให้การอ้างถึงข้อมูลหรือ operations ภายในชนิดข้อมูลหนึ่งเป็นไปได้หรือไม่ได้

กลไกทั้งสองถูกเสนอขึ้นใช้งานในภาษา Modula-2 และภาษา Ada ซึ่งยังไม่ถือว่าเป็นภาษาเชิงวัตถุ และเมื่อมีการพัฒนาภาษาเชิงวัตถุขึ้นก็ต้องมีข้อกำหนดว่าภาษาเชิงวัตถุต้องมีอย่างน้อย 4 กลไกคือ encapsulation, information hiding, inheritance, dynamic binding

หลักการพื้นฐานของ Object- Oriented Programming ที่สำคัญและอาจจะเคยรู้มาบ้างแล้วในเรื่องของ Object Oriented Methodology แต่จะขอกล่าวโดยสรุปอีกทีมีดังนี้

1. Object : Object คล้ายกับกล่องๆหนึ่ง ซึ่งในกล่องมี Data (instance variable) และ Method รวมกันอยู่ภายใน และ Object นี้จะทำการรับและการส่ง message ระหว่าง Object เพื่อทำการใช้งาน Data และ Method ที่อยู่ภายใน โดยการรับและการส่ง message นี้จึงเปรียบเสมือน interface ของ Object นั้นๆด้วย

2. Class : เป็นการจัดกลุ่มของออบเจคตามคุณสมบัติและผู้ใช้สามารถติดต่อคลาสโดยผ่านทางเมธตอดภายในคลาส หรือเมธตอดในซุปเปอร์คลาส (super class หรือ parent class) เท่านั้น หรืออาจจะกล่าวได้ว่าเป็นพิมพ์เขียวของ Object

3. Method : เป็นการทำงาน (Operation) ที่สำคัญหรือเป็นวิธีการกระทำที่สามารถทำงานกับออบเจคได้ แต่ละคลาสจะมีเมธตอดของตัวเอง โดยการทำงานจะเริ่มจากการส่ง message ไปยัง Object ที่ต้องการ (Receiver Object) เพื่อใช้เรียกเมธตอดที่อยู่ภายใน Object นั้น

ตัวอย่าง การส่ง message ไปยัง Receiver Object

receiver.message_name(a1,a2,a3);

4. Encapsulation : คือการรวมกันของโครงสร้างข้อมูล(Data Structure) กับฟังก์ชัน (Method, Action) เกิดเป็นวัตถุใหม่ที่มีความสามารถในการซ่อนข้อมูลจากระบบภายนอกได้ (เหมือนเป็นยาเม็ดแคปซูลที่เราทานแก้หวัดซึ่งจะมองไม่เห็นยาภายในรู้แต่เพียงว่ายาเม็ดนี้ใช้รักษาโรคอะไรเท่านั้น) ทำให้ข้อมูลมีความมั่นคงขึ้น

5. Inheritance : คือการสร้างคลาสใหม่ขึ้นมา โดยมีการสืบทอดคุณสมบัติพื้นฐานที่มาจากคลาสเดิมแต่จะมีข้อมูลหรือเมธตอด เพิ่มขึ้นจากคลาสเดิม ถ้าคลาส B เป็นซับคลาสของ

คลาส A ออบเจคที่เป็นอินสแตนซ์ของคลาส B จะมีคุณสมบัติพิเศษกว่าอินสแตนซ์ของคลาส A แต่อย่างน้อยที่สุดจะต้องมีคุณสมบัติเหมือนอินสแตนซ์ของคลาส A

การถ่ายทอดคุณสมบัตินี้จะรวมถึง Data และ Method ของคลาส A และคลาส A จะถูกเรียกเป็นซุปเปอร์คลาสของคลาส B

6. Polymorphism : คือการที่เราส่งแมสเซจที่เหมือนกันไปในออบเจคที่ต่างกัน แต่ละออบเจคจะตอบสนองออกมาไม่เหมือนกัน ตามแต่ละชนิดและหน้าที่ของออบเจค ความสามารถที่ใช้ในเมสเซจเหมือนกัน

สำหรับการกระทำที่เหมือนกัน ไปยังออบเจคต่างชนิดกัน มีลักษณะเหมือนกับการที่มนุษย์คิดในการแก้ปัญหาหนึ่งๆ

ตัวอย่าง การ polymorphism

AB + CD ได้ผลลัพธ์ ABCD ซึ่ง concatenation โดย Operation จากคลาส String

5 + 3 ได้ผลลัพธ์ 8 ซึ่งadd โดย Operator จากคลาส Integer

ทั้ง 2 แบบ เป็นการส่งเมสเซจ '+' เข้าไปยังออบเจค ภายในคลาส String และ Inter ตามลำดับ

7. Dynamic Binding : คือ การนำโปรแกรมย่อยๆมาประกอบให้ใช้งานในขณะ run time โดยในขณะ Compile time นั้นจะเก็บโปรแกรมในรูปแบบของ Class ต่างๆไว้ เพื่อไม่ให้เกิดความยุ่งยาก ซับซ้อน ต่อจากนั้น เมื่อนำมาใช้ในขณะ run time เมื่อมีการเรียกใช้ Class นั้นมาไว้ในส่วนของโปรแกรม และเมื่อใช้งานเสร็จแล้วจะถูกลบออกจากหน่วยความจำ

8. Override : คือ การที่เราสร้าง Subclass ขึ้นมาใหม่ และมีการสร้าง Method ที่ซ้ำกับ Methodที่เป็นของ Superclass

9. Overload : คือ การที่เราสร้าง Method ชื่อเหมือนกัน แต่รับ Parameter ต่างกัน ภายใน Class เดียวกัน

2.2 ทฤษฎีของภาษาจาวา

2.2.1 แนะนำภาษาจาวา

ในช่วงปี 1990 ตลาดเครื่องใช้ไฟฟ้า มีมูลค่าสูงกว่าตลาดคอมพิวเตอร์หลายเท่าตัว อีกทั้งระบบคอมพิวเตอร์ขนาดเล็กสำหรับควบคุมเครื่องใช้ไฟฟ้าเหล่านี้ก็ถูกพัฒนาให้ดีขึ้นเรื่อยๆ บริษัท Sun Microsystems ซึ่งประสบความสำเร็จในตลาดระบบเครือข่ายคอมพิวเตอร์อย่างมากในตอนนั้น เห็นว่าควรใช้ความได้เปรียบ ทำการพัฒนาระบบซอฟต์แวร์สำหรับควบคุมเครื่องใช้ไฟฟ้าขนาดเล็ก พวกเขาใช้ภาษา C++ เขียนโปรแกรม ผลที่ได้เป็นโปรแกรมที่มักจะทำงานผิดพลาดจน

ต้องสรุปว่าภาษา C++ ไม่เหมาะสำหรับงานแบบนี้เพราะมีข้อจำกัดหลายอย่าง จึงต้องสร้างโปรแกรมสำหรับทำงานพื้นฐานเอง ปัญหาคือ หน่วยประมวลผลที่ใช้ในงานควบคุมมีมากหลายเบอร์ หลายยี่ห้อ และมีชุดคำสั่งไม่เหมือนกัน โปรแกรมที่ทำงานได้บนหน่วยประมวลผลรุ่นหนึ่งจะต้องถูกคอมไพล์ใหม่ จึงจะนำไปใช้งานบนหน่วยประมวลผลอีกรุ่นหนึ่งได้ ด้วยเหตุนี้พวกเขาจึงพัฒนาภาษาใหม่ ชื่อ Oak ให้เหมาะที่จะทำงานในระบบที่หน่วยประมวลผลน้อย พวกเขาแน่ใจว่าระบบควบคุมจะมีขนาดใหญ่และซับซ้อนขึ้นเรื่อยๆ จึงให้ Oak เป็นภาษาเชิงวัตถุ ปัญหาใหญ่ของการออกแบบภาษา Oak อยู่ที่ต้องการให้เป็นภาษาที่ทำงานบนหน่วยประมวลผลใดๆ ก็ได้ จึงนำเทคนิคการคอมไพล์โปรแกรมเป็นคำสั่งของหน่วยประมวลผลสมมติตัวหนึ่ง แล้วสร้าง interpreter ของหน่วยประมวลผลสมมติตัวนั้นให้แก่หน่วยประมวลผลตัวที่จะทำงานโปรแกรมนั้น ด้วยวิธีนี้โปรแกรมที่สร้างขึ้นจึงสามารถนำไปทำงานบนเครื่องที่มีหน่วยประมวลผลต่างรุ่นได้ เราเรียกคุณสมบัติอย่างนี้ว่า platform independent

ในปี 1993 เมื่อ HTML และ browser เปลี่ยนแปลง Internet ไปอย่างมาก และมีผู้ใช้มากขึ้นอย่างรวดเร็ว บริษัท Sun Microsystems มองเห็นความจำเป็นที่ต้องมีภาษาสำหรับสร้างโปรแกรมที่สามารถทำงานบนเครื่องคอมพิวเตอร์ระบบใดๆก็ได้ จึงนำภาษา Oak มาปรับปรุงใหม่ และทดลองสร้าง web browser ชื่อ WebRunner ที่ทำงานโปรแกรมภาษา Oak ได้และทดลองจนได้ผล จึงเปลี่ยนชื่อจาก Oak ซึ่งเป็นชื่อทางการค้าที่มีผู้ใช้อยู่ก่อน เป็น Java ในตอนต้นปี 1995 พร้อมกับเปลี่ยนชื่อ WebRunner เป็นHotJava

เมื่อเริ่มต้น Java ทำงานได้บนระบบ Solaris เท่านั้น แต่เพียงภายในฤดูร้อนของปี 1995 ก็พัฒนาให้ทำงานได้บน Windows NT, Windows 95 และ Linux พอถึงปลายปี 1995 บริษัท Nestcape ก็สร้าง Netscape 2.0 ให้ทำงาน Java ได้ หลังจากนั้นบริษัท Microsoft กับ IBM ก็ประกาศสนับสนุนภาษา Java ด้วย และในปลายปี 1995 บริษัท Sun Microsystems นำโปรแกรมชุดพัฒนาภาษา Java (JDK) รุ่น 1.0 ขึ้นแจกจ่ายใน Internet

2.2.2 ลักษณะหรือคุณสมบัติของภาษาจาวา

ภาษาจาวาถูกออกแบบมาให้มีลักษณะดังต่อไปนี้

1. จาวาเป็นภาษาที่เป็น Object – Oriented กล่าวคือมีการออกแบบให้มีโครงสร้างเป็นเชิงวัตถุ (Object) เพื่อให้การพัฒนาโปรแกรมเป็นไปโดยง่ายและรวดเร็ว และสามารถนำส่วนต่างๆ มาใช้ได้ใหม่โดยไม่ต้องแก้ไขหรือมีการแก้ไขที่น้อยที่สุด

ในภาษาจาวา เกือบทุกสิ่งทุกอย่างจะกำหนดในรูปเชิงวัตถุ ยกเว้นเฉพาะตัวแปรบางชนิด เช่น ตัวเลขและค่าทางตรรกะ จาวามีการจำแนกออกเป็นคลาส (Class) แต่ละคลาสจะมีชุดของเมธตอด (method) ซึ่งแสดงถึงคุณสมบัติเฉพาะของเชิงวัตถุนั้น คลาสสามารถถ่ายทอดคุณสมบัติจากคลาสอื่นได้ โดยที่คลาสเริ่มต้นจะเป็นคลาส Object เสมอ เช่นคลาสรถบรรทุกมีการถ่ายทอด

คุณสมบัติมาจากคลาสของรถ ซึ่งรถถ่ายทอดคุณสมบัติมาจากคลาสยานพาหนะและยานพาหนะถ่ายทอดคุณสมบัติมาจาก Object อีกทอดหนึ่ง

2. เป็นภาษาที่ง่ายในการเรียนรู้และใช้งาน ซึ่งตีความหมายได้หลายอย่างในแง่ต่างๆ ต่อไปนี้

- ภาษาจาวานำหลักไวยากรณ์ภาษาส่วนใหญ่มาจากภาษา C และ C++ ผู้ที่คุ้นเคยกับภาษา CหรือC++อยู่แล้ว จะเข้าใจไวยากรณ์ภาษาจาวาได้ง่าย หรือใช้เวลาศึกษาไม่นานนัก

- ภาษาจาวาตัดคุณสมบัติบางอย่างของภาษา C และ C++ ที่ยุ่งยากออกไปเช่น pointer arithmetic, default argument, scope resolution, protect and private inheritance และ operator overloading เมื่อจาวาเป็นภาษาเชิงวัตถุแล้วกลไกอย่างเช่น structures, unions, bit fields และ enumerated types รวมทั้ง typedef ก็ไม่มีความจำเป็นจึงถูกตัดออกไป นอกจากนี้กลไกที่ทำให้เกิดความกำกวม เช่น multiple inheritance และ copy constructor และกลไกที่อาจทำลายแบบแผนการเขียนโปรแกรมเชิงวัตถุที่ดีเช่น friend methods ก็ถูกตัดออกไปด้วย ซึ่งจาวาเป็นภาษาที่มีการถ่ายทอดแบบ Single – inheritance กล่าวคือ แต่ละคลาสจะถ่ายทอดคุณสมบัติจากคลาสใดคลาสหนึ่งเพียงคลาสเดียว ทำให้ความซับซ้อนของภาษาลดลง

- ภาษาที่ถูกเรียกว่า typed language จะทำการตรวจสอบเกี่ยวกับ type เพื่อช่วยให้โปรแกรมทำงานโดยไม่มีความผิดพลาดเกี่ยวกับ types แต่การเขียนโปรแกรมภาษาเช่นนี้มีข้อยุ่งยาก เช่น ตัวอักษรบวกกับเลขจำนวนเต็มไม่ได้ ภาษาจาวาเป็น typed language แต่เพื่อให้ใช้งานง่าย จึงสร้างกฎเกณฑ์เกี่ยวกับ automatic type coercion ซึ่งเป็นการเปลี่ยนแปลงค่าระหว่าง typeที่ต่างกัน ช่วยให้โปรแกรมง่ายขึ้นโดยลดภาระการเปลี่ยน types ไปจาก programmer

3.โปรแกรมที่สร้างขึ้นด้วยภาษาจาวาจะไม่มีความผิดพลาดจากข้อบกพร่องของภาษา ซึ่งไม่เกี่ยวกับตรรกะของโปรแกรม คุณสมบัติของภาษาอย่างนี้เรียกว่าความคงทน (robust) ภาษาจาวาถูกออกแบบให้มีความคงทนด้วยวิธีการดังนี้

- ภาษาจาวามีการใช้กลไก exception handling เพื่อให้โปรแกรมสามารถจัดการกับความผิดปกติบางอย่างที่เกิดขึ้นในขณะโปรแกรมทำงานให้โปรแกรมทำงานต่อไปได้โดยไม่ต้องหยุดลง ถึงแม้ว่าโปรแกรมที่คุณเขียนขึ้นจะมีข้อบกพร่องแฝงอยู่ก็ตาม ถ้าเป็นข้อบกพร่องที่ระบบไม่รู้จักหรือคาดหวังว่าจะเกิดขึ้น โปรแกรมจาวาจะค้นหาใน exception ต่าง ๆ และดำเนินการก่อนที่จะเกิดปัญหาในภายหลัง ที่เป็นเช่นนั้นเนื่องมาจากโปรแกรมในรูปแบบเดิมอนุญาตให้สามารถดำเนินการกับหน่วยความจำทั้งหมดของเครื่องได้ โปรแกรมสามารถเปลี่ยนค่าต่างๆในหน่วยความจำซึ่งอาจก่อให้เกิดปัญหาในภายหลัง โปรแกรมจาวาอนุญาตให้โปรแกรมดำเนินการกับหน่วยความจำเพียงบางส่วนเท่านั้น ทำให้โปรแกรมจาวาไม่สามารถแก้ไขค่าในส่วนที่ไม่ควรแก้ไขได้

จาวาใช้ Exception ในการบ่งถึงข้อผิดพลาดเมื่อคุณรันโปรแกรม แต่ในบางครั้ง exception เป็นการแจ้งถึงเหตุการณ์พิเศษที่โปรแกรมของคุณควรให้ความสนใจ

ถ้าคุณพยายามที่จะจัดการกับข้อผิดพลาดทุกอย่างที่อาจเกิดขึ้นในโปรแกรมของคุณโครงสร้างของโปรแกรมคุณก็จะซับซ้อนมากและยากต่อการเข้าใจ ข้อดีของ exception คือมีการแยกโค๊ดที่จัดการกับความผิดพลาดออกจากโค๊ดที่จัดการกับความผิดพลาดออกจากโค๊ดที่ทำงานเมื่อไม่มีข้อผิดพลาดเกิดขึ้น ข้อดีอีกข้อหนึ่งก็คือ มีการเตรียมการเป็นพิเศษเพื่อจัดการกับ exception ชนิดต่าง ๆ โปรแกรมของคุณจะต้องจัดการกับ exception นั้น ๆ มิฉะนั้นโปรแกรมของคุณก็จะไม่สามารถคอมไพล์ได้

exception ในจาวาก็คือ วัตถุซึ่งสร้างขึ้นเมื่อมีเหตุการณ์ผิดปกติเกิดขึ้นในโปรแกรม วัตถุนี้จะมีข้อมูลเกี่ยวกับธรรมชาติของปัญหา exception จะถูก thrown นั่นก็คือ วัตถุที่บ่งถึงรายละเอียดเกี่ยวกับ exception จะถูกส่งไปเป็นอาร์กิวเมนต์ไปให้ส่วนของโปรแกรมที่รับผิดชอบในการจัดการกับปัญหานั้น ส่วนของโปรแกรมที่รับวัตถุของ exception เข้ามานั้นเราเรียกว่า catch

- ภาษาจาวาไม่มีกลไกสำหรับคืน (de-allocation) หน่วยความจำที่ขอมาในขณะโปรแกรมทำงาน(dynamic memory allocation) อย่างที่มีในภาษา C และ C++ คือ free() และ delete() ภาษาจาวาอาศัย automatic garbage collector ทำหน้าที่เก็บหน่วยความจำที่ไม่สามารถอ้างถึงได้แล้ว กลับไปใช้งานใหม่โดยอัตโนมัติ

- ภาษาจาวาเป็นภาษาประเภท strongly typed หมายถึง ภาษาที่เน้นความถูกต้องของชนิดข้อมูล(type) ที่ใช้ในโปรแกรม คอมไพเลอร์ของภาษาประเภทนี้จะทำการตรวจสอบว่าโปรแกรมการจัดการกับชนิดข้อมูลของตัวแปรถูกต้องหรือไม่ เรียกกิจกรรมของคอมไพเลอร์นี้ว่า type checking ความผิดพลาดเกี่ยวกับชนิดข้อมูลทั้งหมดจะถูกปฏิเสธตั้งแต่ตอนคอมไพล์ โปรแกรมจึงไม่มีความผิดพลาดเกี่ยวกับชนิดข้อมูลเกิดขึ้นในระหว่างที่โปรแกรมทำงาน นอกจากนี้ยังมีการตรวจสอบอีกว่า ในระหว่างโปรแกรมทำงาน มีการเปลี่ยนชนิดข้อมูล (casting) ระหว่างค่าต่าง type ถูกต้องหรือไม่ และการอ้างถึงสมาชิกใน array หรือ string อยู่ในขอบเขตที่ถูกต้องหรือไม่

4.โปรแกรมภาษาจาวามักจะถูกส่งผ่านระบบเครือข่ายไปทำงานบนเครื่องคอมพิวเตอร์ ดังนั้นภาษาจาวาต้องมีหลักประกันว่าจะไม่ก่อให้เกิดอันตรายต่อเครื่องหรือระบบ ภาษาจาวาจึงมีข้อกำหนดหลายอย่างเพื่อให้โปรแกรมไม่สามารถทำอันตรายต่อระบบที่รับโปรแกรมนั้นไปทำงาน โดยแบ่งการป้องกันออกเป็นหลายระดับดังนี้

- Java interpreter มี byte-code verifier ทำหน้าที่ตรวจสอบโปรแกรมที่จะถูกทำงานว่ามีคำสั่งที่ผิดปกติ หรือมีการทำงานที่ไม่สมควรหรือไม่ หากพบก็จะปฏิเสธที่จะทำงานโปรแกรมนั้น

- ภาษาจาวามีระบบรักษาความปลอดภัยที่เรียกว่า sandbox model นั่นคือ โปรแกรมที่ถูกนำมาจากเครื่องอื่นผ่านทางระบบเครือข่ายจะถือว่าเป็นโปรแกรมที่ไม่น่าไว้ใจ (un-trusted codes) และถูกเก็บอยู่ในภาวะที่เรียกว่า sandbox โปรแกรมที่อยู่ใน sandbox จะมีข้อจำกัดในการทำงานหลายอย่าง ซึ่งถูกควบคุมโดย security manager เช่น ไม่สามารถอ่านหรือเขียนไฟล์เป็นต้น

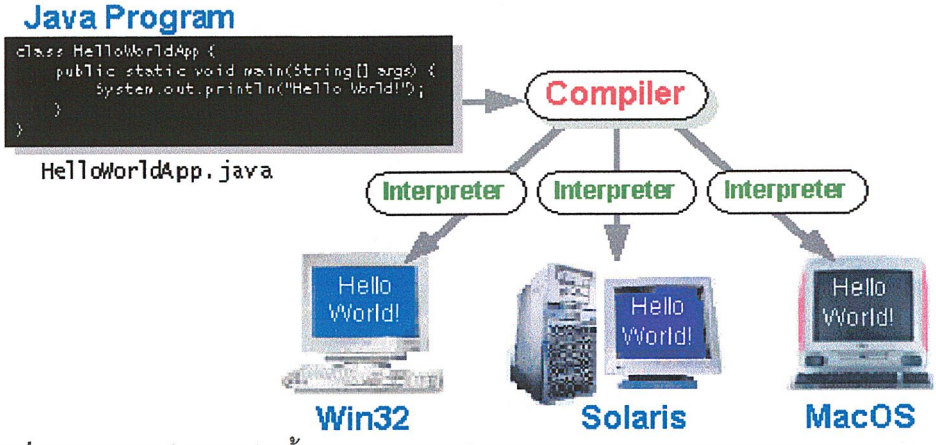
5.จุดมุ่งหมายสำคัญของการออกแบบภาษาจาวาคือ โปรแกรมต้องสามารถทำงานบนเครื่องต่างระบบกันได้ เรียกคุณสมบัตินี้ว่า "ไม่ขึ้นกับระบบ" (architecture neutral หรือ platform independent) เพื่อให้ได้คุณสมบัตินี้ ภาษาจาวาต้องใช้วิธีการแปลภาษาแบบทั้ง compilation และ interpretation รวมทั้งการกำหนด Java Virtual Machine

2.2.3 การทำงานของ Java Virtual Machine

ภาษาจาวานำความคิดการสร้างเครื่องจักรสมมติ (Virtual Machine) มาใช้ เพื่อให้โปรแกรมไม่ขึ้นกับระบบ โดยมีคอมไพเลอร์ทำการแปลภาษาให้เป็นโปรแกรมของ Java Virtual Machine (JVM) แล้วนำโปรแกรมนั้นมาทำงานด้วยเครื่องจักรสมมติที่จำลองขึ้นโดย Java interpreter สรุปได้ดังรูป

รูปที่ 2.5 แสดงการแปลงจากโปรแกรมจาวาไปเป็นโปรแกรมของ Java Virtual Machine

ในวิธีนี้โปรแกรมภาษาจาวาจะถูกคอมไพล์โดย Java compiler ได้เป็นโปรแกรมของ JVM แล้วสามารถนำไปทำงานบนเครื่องใดๆที่มี Java interpreter ได้ จึงมีคุณสมบัติไม่ขึ้นกับระบบ โปรแกรมของ JVM จะทำงานได้เร็วกว่าการใช้ interpreter เพียงอย่างเดียวเพราะขั้นตอนการทำ compilation ถูกแยกออกไปจากการ execution และด้วยการออกแบบคำสั่งของ JVM ให้ใกล้เคียงกับคำสั่งของหน่วยประมวลผลทั่วไป Java interpreter จึงเปลี่ยนคำสั่งของ JVM ไปสู่คำสั่งของหน่วยประมวลผลที่ใช้งานได้ง่าย การทำ interpreter โปรแกรมของ JVM จึงเร็วกว่าการ interpretation ของภาษาระดับสูงอื่นๆ ดูรูปประกอบ



รูปที่ 2.6 แสดงถึงความไม่ขึ้นกับระบบของโปรแกรมภาษาจาวา

ภายใน JVM จะมีหน่วยประมวลผลสมมติ ที่เรียกว่า virtual processor ทำหน้าที่ประมวลผลคำสั่งของ JVM โดยปกติ virtual processor ของ JVM ที่จำลองขึ้นบนคอมพิวเตอร์เครื่องหนึ่ง จะแปลคำสั่งของ JVM เป็นคำสั่งของหน่วยประมวลผลในคอมพิวเตอร์เครื่องนั้น เรียกว่า native code ที่ทำหน้าที่เดียวกันแล้วให้หน่วยประมวลผลทำงานคำสั่ง native code นั้น สังเกตว่า native code นั้นอาจเป็น Applicaiton Program Interface (API) ของระบบปฏิบัติการที่ใช้หรืออาจจะเป็น standard library ที่สร้างขึ้นสำหรับหน่วยประมวลผลนั้น ทำให้ JVM หนึ่งอาจใช้งานได้ในระบบต่างๆ โดยเปลี่ยนแปลงแต่ standard library เท่านั้น คำสั่ง (opcode) ของ JVM มีขนาด 1 byte ทุกคำสั่ง จึงเรียกโปรแกรม JVM ว่าโปรแกรม byte code

นอกจากที่กล่าวมาแล้ว ยังมีลักษณะหรือคุณสมบัติอื่นๆ ของภาษาจาวาอีกดังนี้

- จาวามีการทำงานแบบ Multi-thread

ระบบคอมพิวเตอร์สมัยใหม่ เช่น Unix และ Windows 95 การทำงานแบบ muti - tasking ซึ่งหมายความว่าคอมพิวเตอร์สามารถทำงานได้มากกว่า 1 งานในขณะเดียวกัน จาวาเป็นภาษาที่ถูกออกแบบให้เป็น multi - tasking

โปรแกรมจาวาสามารถมีได้มากกว่า 1 thread ของการทำงานเช่น thread หนึ่งอาจทำงานคำนวณบางอย่างที่ใช้เวลานานและอีก thread หนึ่งก็ทำการติดต่อกับผู้ใช้ ทำให้ผู้ใช้ไม่ต้องหยุดทำงานเพื่อรอให้จาวาเสร็จงานที่ใช้เวลาในการทำงานนานก่อน

โปรแกรมในลักษณะ multi - thread โดยทั่วไปจะยุ่งยากซับซ่อน เนื่องจากเหตุการณ์หลายเหตุการณ์สามารถเกิดขึ้นพร้อม ๆ กัน แต่จาวาได้ตระเตรียมระบบเพื่อจัดการในการเกิดเหตุการณ์พร้อมกัน ช่วยให้การเขียนโปรแกรมเป็นไปได้โดยง่าย

- จาวามีขนาดเล็ก

เนื่องจากจาวาได้ถูกออกแบบเพื่อใช้กับคอมพิวเตอร์ขนาดเล็ก ระบบของจาวาจึงค่อนข้างเล็ก สามารถรันได้อย่างมีประสิทธิภาพบนเครื่องคอมพิวเตอร์ที่มีตั้งแต่ 4 Mb RAM ตัวแปลภาษาจาวาใช้เนื้อที่หน่วยความจำเพียงไม่กี่ร้อยกิโลไบต์

โปรแกรมที่สร้างขึ้นด้วยภาษาจาวาแบ่งได้เป็น 2 ประเภท คือ Java Application และ Java Applet

2.2.4 การทำงานของ Java Application

Java Application คือ โปรแกรมที่สร้างขึ้นเพื่อถูกทำงานโดย Java interpreter ต้องเป็นโปรแกรมที่สมบูรณ์ มี main() เป็นจุดเริ่มต้น และสามารถควบคุมการดำเนินไปของตัวเองได้ ทำงานภายใต้ Java interpreter โดยไม่ต้องมีโปรแกรมอื่นช่วย บางคนจึงเรียกว่า stand alone program

ขั้นตอนการสร้างและทำงาน Java application มีดังนี้ เริ่มจากใช้ editor เขียนโปรแกรมสำหรับ application เก็บไว้ในไฟล์ที่ extension เป็น .java จากนั้นใช้คอมไพเลอร์ javac.exe ทำการคอมไพล์ จะได้ผลลัพธ์เป็นไฟล์ที่มี extension เป็น .class ซึ่งเป็นโปรแกรม JVM เมื่อต้องการทำงานก็ส่งไฟล์ที่มี extension เป็น .class นั้นให้แก่ java.exe

2.2.5 การทำงานของ Java Applet

Applet คือ โปรแกรมที่สร้างขึ้นเพื่อส่งไปกับ HTML page ให้ไปทำงานภายใต้ web browser ที่มีJava interpreter โดยสามารถติดต่อกับผู้ใช้และแสดงผลผ่านทาง web page นั้น หรือกล่าวได้ว่าเป็นโปรแกรมที่ฝากมากับ HTML page นั่นคือ จากใน HTML page จะสามารถเรียก applet มาทำงานและแสดงผลในพื้นที่ของ page นั้น โดยที่ web browser จะเป็นผู้โหลด applet ที่ถูกเรียกมาจาก server ที่ให้ HTML page นั้นมาและจัดการทำงานให้โดยอัตโนมัติ วิธีนี้ทำให้เราสามารถส่งโปรแกรมไปทำงานบนเครื่องใดๆในระบบอินเตอร์เน็ต โดยใช้ HTML page เป็นตัวกลาง

ขั้นตอนการสร้างและทำงาน applet มีดังนี้ ใช้ editor เขียนโปรแกรมสำหรับ applet เก็บไว้ในไฟล์ที่ extension เป็น .java จากนั้นใช้คอมไพเลอร์ javac.exe ทำการคอมไพล์ จะได้ผลลัพธ์เป็นไฟล์ที่มี extension เป็น .class ซึ่งเป็นโปรแกรม JVM ของ applet นั้น (สังเกตว่า java.exe จะไม่สามารถทำงานกับไฟล์ที่มี extension เป็น .class ที่ได้นั้นได้เนื่องจากไม่ใช่ Java application) จากนั้นก็ต้องแทรก Tag <applet code="file.class" width=100 height=100> </applet> ลงไปในเอกสาร HTML (file.class คือไฟล์ applet ที่เราต้องการ) ซึ่งไฟล์ของ applet นั้นจะถูกมองหาในไดเรกทอรีและเครื่องเดียวกันกับ HTML page นั้นหากเป็นกรณีที่ applet อยู่ที่เครื่องอื่นก็ต้องอ้างถึงไฟล์ของ applet นั้น และเมื่อต้องการดูผลของ applet นี้ก็ใช้ web browser ที่มี Java

interpreter เช่น Netscape หรือ Microsoft Internet Explorer เรียกดูไฟล์ HTML ที่มี applet อยู่ เมื่อไฟล์นี้ถูกจัดการถึงประโยคที่เรียก applet นั้น ไฟล์ applet นั้นก็จะถูกโหลดมาทำงาน นอกจากนี้ยังสามารถใช้โปรแกรม appletviewer.exe ดูผลของ applet นี้ได้ด้วย

2.2.6 การเปลี่ยนจากทฤษฎีเชิงวัตถุ เป็นโปรแกรมในภาษาจาวา

2.2.6.1 ตัวอย่างเรื่อง Class ,Object ,Variable and Method

จากหลักการของ Object Oriented ดังที่กล่าวมา สามารถอธิบายได้ด้วยภาษาจาวาซึ่งเป็นภาษาที่ใช้หลักการของ Object Oriented ได้ดังนี้

- ตัวอย่างที่ 1

```
public class Point{
        protected int x,y;  //coordinates of the Point
        //construtor
        public Point(int a,int b){
          x=a;
          y=b;
        }
        public int getX(){return x;}
        public int getY(){return y;}
}
```

จากตัวอย่างที่ 1 อธิบายได้ดังนี้

- ชื่อของ Class คือ Point
- Instance variables (ซึ่งก็คือData หรือ Attribute หรือ Variable นั่นเอง) มี 2 ตัวด้วยกัน คือ x และ y ซึ่งเป็นตำแหน่งของจุดหรือ Point
- Constructor เป็นชื่อเดียวกับชื่อ Class เมื่อเราสร้าง Object Point แล้ว ก็จะมีการทำงานใน method นี้ทันที

```
public Point(double a,double b){
        x=a;
          y=b;
    }
```

- method นอกจาก method Point ที่เป็น Constructor method แล้ว ใน class Point ยังมีอีก 2 method คือ getX() และ getY() ซึ่งจะทำการ return ค่าของ x และ y

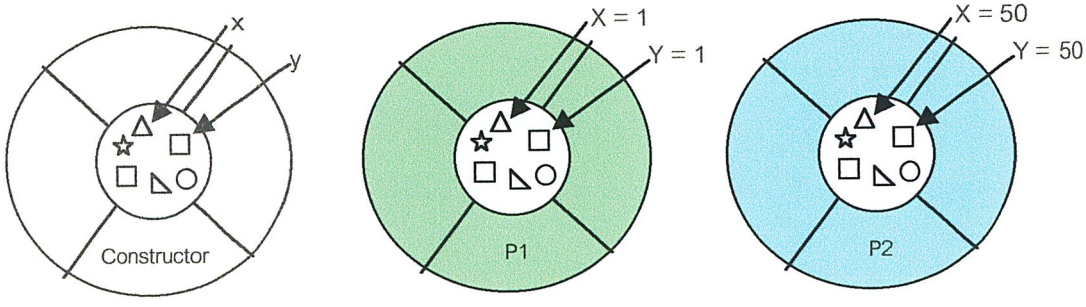
- ตัวอย่างที่ 2

```
import java.awt.*;
import java.applet.Applet;
public class Draw extends Applet{
        private Point p1,p2;
        public void init(){
            p1=new Point(1,1);
            p2=new Point(50,50);
        }
        public void paint(Graphics g){
            g.drawString("x of p1="+p1.getX(),20,20);
            g.drawString("x of p2="+p2.getX(),20,40);
        }
}
```

จากตัวอย่างที่ 2 นอกจากชื่อ Class , Variable และ Method ที่อธิบายไปแล้วในตัวอย่างที่ 1 สามารถอธิบายเพิ่มเติมได้ดังนี้ คือ ในmethod init() ซึ่งเป็น method ที่เรียกทันทีหลังจากที่ class Draw applet ถูกเรียกให้ทำงานจะมีการสร้าง Object ที่ชื่อ p1 และ p2 ซึ่งมี type เป็น Point

```
public void init(){
        p1=new Point(1,1);
    p2=new Point(50,50);
}
```

ดังนั้นจะมีการทำงานของ constructor method ของ class Point ทันที และทำการกำหนดค่าของ x เท่ากับ 1 และ y เท่ากับ 1 สำหรับ object p1 และกำหนดค่าของ x เท่ากับ 50 และ y เท่ากับ 50 สำหรับ object p2 ดูรูปภาพประกอบ



รูปที่ 2.7 แสดงการกำหนดค่าของ Construtor และการกำหนดค่าเมื่อสร้างเป็น Object

จากรูปด้านซ้ายจะแทนด้วย class Point ส่วนรูปตรงกลางและรูปด้านขวาแทน object p1 และ object p2 ตามลำดับ

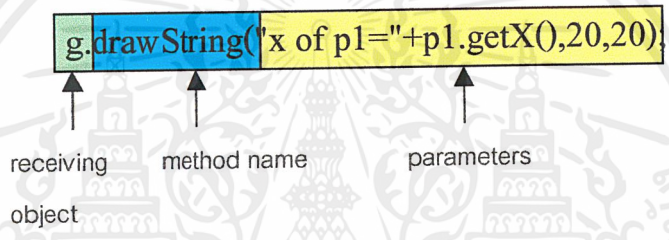
2.2.6.2 **ตัวอย่างเรื่อง** Messages

จาก code ของตัวอย่างที่ 2

```
g.drawString("x of p1="+p1.getX(),20,20);
g.drawString("x of p2="+p2.getX(),20,40);
```

จะเห็นว่ามีการติดต่อกันหรือส่ง Message ระหว่าง applet กับ object g ซึ่งเป็น object ที่ใช้ในการแสดงผล บรรทัดแรกเป็นการแสดงค่า x ของ object p1 ส่วนบรรทัดที่ 2 แสดงค่า x ของ object p2

รูปต่อไปนี้แสดงส่วนประกอบของการส่ง Message ของบรรทัดแรก



รูปที่ 2.8 แสดงส่วนประกอบของการส่ง message ระหว่าง object

2.2.6.3 **ตัวอย่างเรื่อง** Inheritance

ในการแสดงผลบน browser นั้น object จะต้องเป็น applet ซึ่งรับการถ่ายทอด (inherit) มาจาก applet class และจากตัวอย่างที่ 2 นั้น class Draw ก็มีการรับการถ่ายทอดมาจาก applet class ด้วยเช่นกันดัง code ข้างล่างนี้

```
public class Draw extends Applet{
    ...
}
```

extends Applet นี้ทำให้ Draw เป็น subclass ที่รับการถ่ายทอดคุณสมบัติมาจาก class Applet ซึ่งเป็น superclass ดังนั้น class Draw จึงสามารถใช้คุณสมบัติหรือ method ต่างๆของ Applet class ได้ด้วยเช่น ความสามารถในการกำหนดค่าเริ่มต้นใน method init() , ถูก start ใน method start() และ stopใน method stop() โดย browser ซึ่งในตัวอย่างก็มีการใช้ method init () และ method paint() ด้วย

2.2.6.4 ตัวอย่างเรื่อง Encapsulation and Information Hiding

ในภาษาจาวามีการกำหนดสิทธิในการเข้าถึงหรือเรียกใช้ variable และ method ด้วยการใช้คำเฉพาะนำหน้า variable และ method ดังนี้

ตารางที่ 2.1 แสดงสิทธิในการอ้างถึงของ variable และ method ในภาษาจาวา

Accessible to	private	Protected	public	package
Same class	✓	✓	✓	✓
Subclass	✗	✓	✓	✗
Package	✗	✓	✓	✓
World	✗	✗	✓	✗

- private สามารถอ้างถึงได้เฉพาะภายใน class เดียวกันเท่านั้น
- protected สามารถอ้างถึงได้ภายใน class เดียวกัน subclass และ ภายใน package
- public สามารถอ้างถึงได้ในทุกๆ ที่
- package สามารถอ้างถึงได้เฉพาะภายใน class เดียวกันและภายใน package เดียวกันเท่านั้น

สามารถดูตัวอย่าง code ได้ในตัวอย่างที่ 1 และ 2

2.2.6.5 ตัวอย่างเรื่อง Polymorphism

ในภาษาจาวาซึ่งมีกลไก inheritance และ dynamic binding ตามหลักการที่กล่าวมาแล้วในเรื่อง Object Technology ทำให้สามารถสร้าง method ที่มีลักษณะที่เรียกว่า polymorphism method คือ method ที่สามารถจัดการกับ instance ของ class ที่ต่างกันได้ ดูตัวอย่างต่อไปนี้

- ตัวอย่างที่ 3

```
class Animal{
        private String type;
        public Animal(String s){type=s;}
        public void sound(){}
}
class Dog extends Animal{
        public Dog(String s){super(s);}
        public void sound(){System.out.println("box box");}
}
```

```
class Cat extends Animal{
        public Cat(String s){super(s);}
        public void sound(){System.out.println("meaw meaw");}
}
public class testPoly{
        static Dog dog;
        static Cat cat;
        public static void main(String ar[]){
                dog=new Dog("dog");
                cat=new Cat("cat");
                dog.sound();
                cat.sound();
        }
}
```

จะได้ผลลัพธ์

```
box box
meaw meaw
```

จากตัวอย่างใน class testPoly มีการสร้าง object dog และ cat ซึ่งเป็นคนละ class กัน แต่ทั้งสอง class มีการ inherit มาจาก class Animal เหมือนกัน และใน class Animal นี้ก็มี method sound() ทำให้ทั้ง class Dog และ Cat สามารถเรียกใช้ method sound() ได้ทั้งคู่ เมื่อโปรแกรมทำงานจนถึงบรรทัดที่มีการเรียกใช้method sound() ดัง code

```
dog.sound();
cat.sound();
```

compiler จะทำการตรวจสอบว่าเป็นการเรียกจาก object ใน class ใดก็จะสั่งให้ method ใน class นั้นทำงาน (จากหลักการ dynamic binding) ซึ่งใน method sound() ใน class Dog และ Cat นั้นมีการ implement ที่ต่างกัน ทำให้ผลลัพธ์จากการเรียก mehtod sound() เหมือนกันของ object dog และ cat ได้ผลลัพธ์ที่ต่างกัน

2.3 การทำงานแบบ Thread

2.3.1 ความหมายของ Thread

ในระบบปฏิบัติการแบบ Multitasking ผู้ใช้ซึ่งอาจจะมีมากกว่าหนึ่งคน สามารถส่งโปรแกรมให้แก่ระบบปฏิบัติการทำงานมากกว่าหนึ่งโปรแกรมในเวลาหนึ่งได้ โดยโปรแกรมเหล่านี้จะไปเข้าคิวเพื่อรอเข้าทำงานในหน่วยประมวลผล บางระบบปฏิบัติการอาจกำหนดเวลาที่แต่ละ Process จะได้ทำงาน หาก Process ใดใช้เวลาทำงานเกินเวลาที่กำหนดให้ ก็จะถูกหยุดและออกมาเข้าคิวรอรอบต่อไป และโปรแกรมที่รอคิวจะเข้าไปทำงานแทน ความเร็วของคอมพิวเตอร์ทำให้การทำงานลักษณะนี้ดูคล้ายกับว่าโปรแกรมเหล่านั้นทำงานไปพร้อมๆ กัน ทั้งที่จริงแล้วเป็นการจัดคิวให้เข้าทำงานทีละโปรแกรม โปรแกรมส่วนใหญ่มักต้องทำงานหลายอย่างไปพร้อมๆ กัน เช่น ในขณะหนึ่งอาจอ่านข้อมูลจาก disk พร้อมกับแสดงตัวอักษรที่จอภาพ หากงานทั้งสองนี้ถูกแยกกันทำเป็นสอง Process และทำเป็นเวลานาน จะเห็นการพิมพ์ตัวอักษรที่จอภาพสะดุดเป็นช่วงๆ เนื่องจากมีการสลับกันทำงานระหว่าง Process ที่อ่านข้อมูลกับ Process ที่พิมพ์ตัวอักษร การเปลี่ยนการทำงานของ Process หนึ่งไปสู่อีก Process หนึ่ง เรียกว่า Context Switching ในขั้นตอนนี้ต้องมีการเก็บสถานะการคำนวณของ Process เดิมไว้ เพื่อให้สามารถกลับมาทำงานต่อจากจุดนั้นได้ และต้องโหลดสถานะการคำนวณของ Process ใหม่เข้าไปทำงานแทนที่ภาระการทำ Context Switching นี้เป็น Overhead ที่ทำให้โปรแกรมงานทำงานช้าลง ผู้ออกแบบระบบปฏิบัติการและภาษาสำหรับโปรแกรมจึงหาวิธีทำให้ overhead ของ Context Switching ลดลง ด้วยการเสนอกลไกใหม่ที่เรียกว่า Thread

2.3.2 การทำงานของ Thread

ระบบปฏิบัติการแบบ Multi-Threading จะสามารถแบ่ง Process ออกเป็นหน่วยที่เล็กกว่าซึ่งเรียกว่า Thread หรือบางทีเรียกว่า light-weight Process ระบบปฏิบัติการแบบนี้จะสามารถสร้างหลาย Threads ในหนึ่ง Process โดยปกติ Thread หนึ่งคือ execution flow ของทำงานอย่างหนึ่ง หากนำหลายๆ Threads มารวมกันใน Process หนึ่ง จะช่วยให้ Process นั้นสามารถทำงานได้มากกว่าหนึ่งอย่างไปพร้อมๆ กันโดยไม่ต้องมีการทำ Context Switching โปรแกรมจึงทำงานได้เร็วและ เป็นการใช้งานหน่วยประมวลผลได้อย่างมีประสิทธิภาพมากขึ้น

ในระบบ Multitasking โดยปกติหนึ่ง Process จะมีหนึ่ง PCB ส่วนในระบบ Multi-Threading หนึ่ง Process มีหนึ่ง PCB เช่นกัน แต่อาจมีหลาย Threads เหล่านั้นมีส่วนที่เป็นโปรแกรม (code segment) ร่วมกันและใช้หน่วยความจำสำหรับเก็บข้อมูล (data segment) บางส่วนร่วมกัน หาก Threads ใน Process เดียวกันใช้ตัวแปรบางตัวร่วมกัน เราเรียกตัวแปรเหล่านั้นว่า shared variable และถ้า Threads หนึ่งมีตัวแปรของมันเองที่ไม่ถูกใช้โดย Thread อื่น ก็เรียกตัวแปรนั้นว่า local variable โดยปกติ Thread หนึ่งจะทำงานในบางส่วนของ code segment

เท่านั้น แต่ละ Thread จะมีโปรแกรมของตัวเองมีจุดเริ่มต้นและจุดสิ้นสุดแตกต่างกันไป Thread แต่ละตัวมักจะทำงานเป็นอิสระต่อกันทำให้ Thread หลายๆ ตัวสามารถทำงานพร้อมกันได้ ถ้าเครื่องคอมพิวเตอร์นั้นมีหน่วยประมวลผลหลายตัวก็อาจแบ่งให้หน่วยประมวลผลแต่ละตัวทำงานแต่ละ Threadได้พร้อมกันแบบขนาน (Parallel) ซึ่งจะช่วยให้โปรแกรมทำงานเร็วขึ้น

การจัดการให้ Thread ใดถูกทำงานเมื่อใด และโดยหน่วยประมวลใด นั้นเป็นหน้าที่ของระบบปฏิบัติการ ไม่อยู่ในการควบคุมของผู้เขียนโปรแกรมโดยตรง ผู้เขียนโปรแกรมทำได้แต่เพียงสร้าง Threads ให้ทำหน้าที่ต้อง และควบคุมให้ Thread เหล่านั้นเริ่มต้นทำงาน และหยุดการทำงาน ในระหว่างที่ Thread ทำงาน โปรแกรมยังสามารถออกคำสั่งให้ Thread นั้นหยุดรอ และเริ่มต้นทำงานเมื่อถึงเวลาที่กำหนดได้

ข้อแตกต่างระหว่าง Thread กับ Process พอสรุปได้ดังนี้คือ

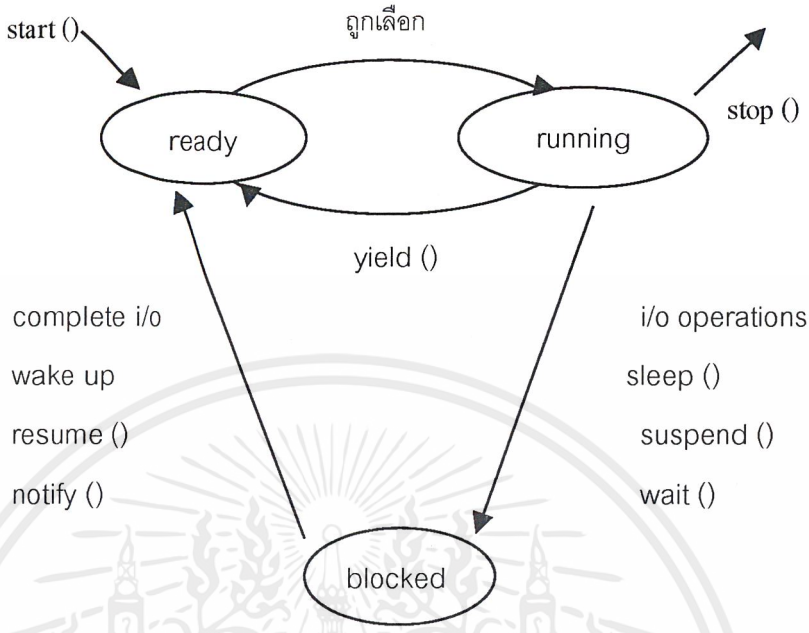
1. Dependency แม้ว่าแต่ละ Threads ของ Process เดียวกันจะทำงานที่แตกต่างกัน แต่มีความเกี่ยวข้องกันบางอย่างและต้องทำงานอยู่ภายใน environment เดียวกัน จึงอาจใช้ตัวแปรบางตัวร่วมกัน หรืออาจใช้โปรแกรมบางส่วนร่วมกัน แต่สำหรับ Process จะไม่มีความเกี่ยวข้องเลย เพราะ แต่ละ Process จะมี environment ของตัวเอง

2.Communication การติดต่อระหว่าง Threads ทำได้ง่าย โดยอาจใช้ shared variables แต่ สำหรับ Process ไม่มี shared variable เนื่องจาก เมื่อมีการทำ Context Switching จะเปลี่ยน environment ของ Process ด้วย วิธีเดียวที่ Process จะส่งค่าให้แก่กันได้คือทำผ่านทางไฟล์ เรียกว่า pipe ซึ่งเป็นวิธีที่สิ้นเปลืองมากและช้ากว่าการอ่านหรือกำหนดค่าตัวแปรอย่างมาก

3. Context Switching เมื่อ Process หนึ่งถูกนำเข้าทำงาน หาก Process นั้นมีมากกว่าหนึ่ง Thread ก็จะมีการสลับกันทำงานระหว่าง Threads เหล่านั้น โดยที่ไม่ต้องเปลี่ยน environment และ PCB ของ Process ทำให้การสลับการทำงานระหว่าง Threads เป็นภาระต่อเครื่องน้อยกว่าการทำ Context Switching ของการเปลี่ยน Process ส่งผลให้ Threads เหล่านั้นมีเวลาทำงานมากขึ้น

2.3.3 วัฏจักรของ Thread

วัฏจักร (lifetime) ของ Thread แสดงได้เป็นรูปดังรูปนี้



รูปที่ 2.9 แสดงวัฏจักรของ Thread

การเปลี่ยนแปลงสถานะของ Thread มีกฎเกณฑ์ดังนี้

1. โปรแกรมหนึ่งอาจสร้าง Thread ขึ้นมาได้โดยการ New instance ของคลาส Thread เมื่อ Thread ถูกสร้างขึ้น อาจมีการกำหนดค่าเริ่มต้นให้แก่ตัวแปรบางตัวของมันได้ แต่ยังไม่เริ่มทำงาน

2. เมื่อ Thread ที่อยู่ในสถานะ new ได้รับคำสั่ง start () จะเปลี่ยนไปสู่สถานะ ready คือไปรออยู่ในคิวของ Threads จนกว่าจะออกไปทำงานในหน่วยประมวล โดยปกติ Thread จะเปลี่ยนสถานะจาก ready ไป running เร็วมาก

3. หากไม่มี Thread ในสถานะ running ระบบปฏิบัติจะเลือก Thread หนึ่งจากคิวของ Thread ที่รออยู่ในสถานะ ready เพื่อไปทำงาน กฎเกณฑ์ที่ใช้ในการเลือกนี้ขึ้นกับระบบปฏิบัติการ (ไม่มีกำหนดในภาษา Java) เช่น ในระบบ Solaris 1.x จะใช้ priority ของ Thread เป็นตัวเลือก ส่วนในระบบ Win32 จะใช้ทั้ง priority กับ time-slice ของ Thread ในการเลือก

4. เมื่อ Thread เข้ามาอยู่ในสถานะ running จะเปลี่ยนสถานะต่อไปได้ 3 แบบคือ

4.1 เป็นสถานะ death เมื่อ Thread นั้นได้รับคำสั่ง stop() เมื่อ Thread เข้าในสถานะ death จะถูกนำพื้นที่หน่วยความจำคืนไป ไม่สามารถกลับเข้าสู่สถานะ ready ได้อีก

4.2 เป็นสถานะ ready เมื่อ Thread ถูก Interrupt หรือรับคำสั่ง yield () ผู้เขียนโปรแกรมอาจใช้คำสั่ง yield() เมื่อต้องการให้ Thread อื่นที่คอยในสถานะ ready เข้ามาทำงานแทน ทำให้ Thread ที่กำลังทำงานอยู่นั้นหยุดการทำงานชั่วคราว และจะกลับเข้าทำงานอีกเมื่อถูกเลือก

4.3 เป็นสถานะ blocked เมื่อ Thread นั้นรับคำสั่ง wait() , suspend () , หรือ sleep () จากโปรแกรม หรือทำการติดต่อกับ I/O หรือรอใช้ทรัพยากรของเครื่องที่มี Thread อื่นกำลังใช้อยู่

5. Thread ที่อยู่ในสถานะ blocked จะเปลี่ยนสถานะเป็น ready เมื่อได้รับคำสั่ง notify() หรือ resume() จากโปรแกรม หรือเมื่อหมดเวลานอนหลับจากคำสั่ง sleep() หรือเมื่อเสร็จสิ้นการติดต่อกับ I/O หรือได้รับทรัพยากรของระบบที่รอใช้งานอยู่

สรุปว่า Thread ที่เข้าสู่สถานะ blocked ด้วยคำสั่งด้านล่างนี้ จะกลับเข้าสู่สถานะ ready เมื่อมี เหตุการณ์ต่อไปนี้

wait(): เมื่อมี Thread อื่นออกคำสั่ง notify()

suspend(): เมื่อ Thread ได้รับคำสั่ง resume()

sleep(time): เมื่อครบเวลา time

ทำ I/O เสร็จ

2.3.4 Thread ในภาษาจาวา

ภาษา Java สนับสนุนการโปรแกรม multi-Threading ด้วยกลไกที่กำหนดขึ้นภายในภาษา ต่างกับภาษา C และ C++ ที่สนับสนุน Thread ด้วย library ที่เขียนขึ้นต่างหากสำหรับแต่ละคอมไพเลอร์และระบบปฏิบัติการ โปรแกรมภาษา Java ทุกๆ โปรแกรมจะต้องมี Thread อย่างน้อยหนึ่ง Thread เสมอ คอมไพเลอร์จะเป็นผู้สร้าง Thread ให้แก่ main() ของทุกๆ โปรแกรม เรียกว่า main Thread และ Java Interpreter จะเป็นผู้ควบคุม Main Thread ที่หลังจากทำงานแล้ว อาจจะมีการสร้าง Thread อื่นๆ ได้อีก

คลาส Thread มี data members สำหรับเก็บค่าคุณสมบัติของ Thread ดังต่อไปนี้

```
private  char  name [] ;
```

คือชื่อของ Thread เราสามารถตั้งชื่อ Thread โดยส่ง String เป็นพารามิเตอร์ในแก่ constructor หากไม่มีชื่อ Thread นั้นจะถูกตั้งชื่อให้โดยอัตโนมัติเป็น “Thread-1”+n โดย n เป็นเลขจำนวนเต็มที่เพิ่มขึ้นตามจำนวน Thread ที่ถูกสร้างขึ้น

```
private int priority ;
```

คือค่าระดับความสำคัญของ Thread ในการถูกระบบปฏิบัติการเลือกไปทำงาน

```
private boolean daemon = false;
```

เป็นค่าที่ระบุว่า Thread นั้นเป็น Daemon Thread หรือไม่

หาก Thread หนึ่งถูกสร้างขึ้นและถูกเริ่มการทำงานด้วย start() แล้ว จะถือว่ามีชีวิต ซึ่งเมื่อเรียก isAlive() จะเป็น true จนกว่าเมื่อสิ้นสุดการทำงานแล้ว isAlive() จะเป็น false

2.3.4.1 รายละเอียดและการทำงานของ Class Thread

คลาส Thread ถูกกำหนดใน package java.lang ใช้สำหรับสร้าง instance ซึ่งเราจะเรียกสั้นๆว่า Thread คลาส Thread มีโปรแกรมสำหรับการทำงานของ Thread ตัวหนึ่ง ที่มีการดำเนินของโปรแกรมแยกออกไปจากโปรแกรมที่สร้างมัน Thread ที่สร้างขึ้นจะยังไม่เริ่มต้นทำงานจนกว่า start() ของมันจะถูกเรียก เมื่อ start() ถูกเรียกแล้ว มันจะไปเข้าคิว จนเมื่อมันถูกเลือกไปทำงาน โปรแกรมใน run() ก็จะถูกทำงาน

คลาส Thread มี methods สำหรับควบคุมให้ Thread เข้าสู่สถานต่างๆ เช่น start(), yield(), sleep(), suspend(), resume() และ stop() โดยปกติเราจะไม่ override methods เหล่านี้ นอกจากนี้ยังมี

```
public void run()
```

ซึ่งเป็น method ที่ว่างเปล่า ที่โดยปกติเราจะต้อง override เพื่อระบุว่าเมื่อ Thread นั้นอยู่ในสถานะ running จะทำอะไรบ้าง หากไม่ทำการ override run() ก็จะได้ Thread ที่ไม่ทำอะไรเลย

2.3.4.2 รายละเอียดและการทำงานของ Interface Runnable

วิธีที่สองในการสร้าง Thread คือสร้างคลาสที่ implement Runnable interface ใน interface นี้มีเพียง run() เป็น abstract method คลาสที่ implementation Runnable interface ใน interface นี้มีเพียง run() เป็น abstract method คลาสที่ implementation Runnable จะต้องมี run() เพื่อระบุการทำงานของ Thread ในสถานะ running

Runnable Inerface ถูกกำหนดไว้ใน java.lang ดังนี้

```
Public abstract interface Runnable {
        Public abstract void run();
}
```

การใช้ Runnable สร้าง Thread อาจทำได้สองวิธี

1. สร้างคลาสที่ implements Runnable โดยมี Thread ตัวหนึ่งเป็นสมาชิก
2. สร้าง instance ของ คลาสที่ implement Runnable แล้วส่งให้เป็นพารามิเตอร์ของ constructor ของคลาส Thread

ตัวอย่างต่อไปเป็นการแสดงการสร้างคลาส implement Runnable ที่มี Thread ตัวหนึ่งเป็นสมาชิก

```
Class   MyThread implements Runnable {
            Thread t;
            MyThread (String n) {
                T = new Thread (this, n);
            }
            public void run () {
                for (int  i = 0 ; I < 100; i++)
                    System.out.print (t.getName()) ;
            }
    }
    Class  RunnableTest1 {
            Public static void main (String args[]) {
                new  MyThread ("A") . t.start();
                new  MyThread ("B") . t.start();
            }
    }
```

คลาส My Thread ในตัวอย่างนี้ทำการ implements interface Runnable ในคลาสนี้มี Thread t เป็นสมาชิก ส่วนใน constructor มีการสร้าง instance ของคลาส Thread และ กำหนดค่าให้แก่ t ในการเรียก Thread() นี้ เราส่ง this และ n เป็นพารามิเตอร์ โดย this คือ reference ของคลาสนี้ซึ่งมี run() เหตุที่ต้องส่ง this ไปก็เพื่อให้โปรแกรมในคลาส Thread ใช้ this เรียก run () ที่กำหนดใน instance นี้ทำงาน ส่วน String n จะถูกนำไปเป็นชื่อของ Thread คลาส My Thread นี้มีการกำหนด run() ซึ่งมีประโยค for ทำการพิมพ์ชื่อของ Thread นี้ออกมา 100 ครั้ง เช่นกัน ส่วนใน main() ทำการสร้าง Threads ขึ้นมาสอง Thread โดยการ new และ ตั้งชื่อให้เป็น "A" กับ "B" แต่การเรียก start() ให้ Thread เริ่มต้นทำงานต้องทำผ่านทาง reference t ใน คลาส My Thread นั้น

ตัวอย่างต่อไปแสดงการสร้างคลาสที่ implements Runnable แล้วส่งให้เป็นพารามิเตอร์ของ constructor ของคลาส Thread

```
// RunnableTest2.java
Class   MyThread implements Runnable  {
        public void run()  {
```

```
            For (int i = 0 ; i < 100 ; i++)
                System.out.print(Thread.currentThread( ).getName( )) ;
        }
}
Class   RunnableTest2 {
        public static void main (String args[ ] {
            new Thread (new MyThread ( ) , "A" ) .start ( ) ;
            new Thread (new MyThread ( ) , "B")  .start ( ) ;
        }
}
```

คลาส My Thread ในตัวอย่างนี้ implements Runnable แต่ไม่มี Thread t เป็นสมาชิก เหมือนในตัวอย่างที่แล้ว ดังนั้นเมื่อจะเรียก getName() ต้องเรียก Thread.currentThread() ก่อน เพื่อได้ reference ของ Thread ที่กำลังทำงานอยู่ ส่วนใน main() มีการสร้าง instances ของคลาส Thread ขึ้น โดย constructor มีพารามิเตอร์ตัวแรกเป็น Runnable ซึ่งมี run() ที่จะถูกทำงาน และตัวที่สองเป็น String ที่จะเป็นชื่อของ Thread นั้น

2.3.5 ฟังก์ชันของ Thread ในภาษาจาวา

2.3.5.1 วิธีการใช้ฟังก์ชัน Sleep()

ภาษาสำหรับโปรแกรมโดยทั่วไปมักจะมีกลไกหน่วงเวลาให้โปรแกรมทำงานช้าลง เช่น ในภาษา C มี delay() ใช้ประโยชน์ในหลายกรณี เช่น ทำให้โปรแกรมแสดงผลช้าลงให้ผู้ใช้อ่านได้ทัน เป็นต้น ในภาษา Java เราสามารถทำให้โปรแกรมทำงานช้าลงได้ โดยออกคำสั่ง

```
public static native void sleep (long m)   throws   InterruptedException
```

แก่ Thread นั้น โดยพารามิเตอร์ long m นี้เป็นระยะเวลาที่ Thread จะหยุดทำงาน หน่วยเป็นเศษหนึ่งส่วนพันของวินาที (millisecond) ผลของ sleep(time) จะทำให้ Thread นั้นเข้าสู่สถานะ blocked เป็นระยะเวลา m ที่ระบุแล้ว Thread ก็จะเข้าสู่สถานะ ready เอง

2.3.5.2 วิธีการใช้ฟังก์ชัน Suspend () and resume()

เมื่อ Thread หนึ่งกำลังทำงานอยู่ในสถานะ running หากมีคำสั่ง suspend() จะทำให้ Thread นั้น หยุดการทำงานและเข้าสู่สถานะ blocked ไปจนกว่าเมื่อได้รับคำสั่ง resume() ก็จะทำให้ Thread นั้นเข้าสู่สถานะ ready และพร้อมที่จะทำงานเมื่อถูกเลือก กลไกนี้มีประโยชน์ในการขัดจังหวะการทำงานของ Thread เพื่อรอเหตุการณ์บางอย่างจนกว่าจะถึงเวลาที่ต้องการก็ออกคำสั่งให้ Thread นั้นทำงานต่อไป ดังตัวอย่างการแสดงการทำงานของ การ suspend() และการ resume()

```
Class MyThread extends Thread{
        MyThread(String s) {super(s);}
        Public void run() {
                For(int I=0;I<50;I++){
                        System.out.print(getName());
                        Try{sleep(100);}
                        Catch(InterruptedException e){}
                }
        }
}
classs SusRes {
        Public static void main(String args[])          {
                MyThread a = new MyThread("A");
                MyThread b = new MyThread("B");
                a.start();
                b.start();
                try{
                        Thread.sleep(1000);
                        a.suspend();
                        System.out.println("\n A is suspended");
                        Thread.sleep(1000);
                        a.resume();
                        System.out.println("\n A is resumed");
                }catch (InterruptedException e){}
        }
}
```

จะได้ผลลัพธ์ดังนี้

ABABABABABABABABABA

A is suspended

BBBBBBBB

A is resumed

ABABABABABABBABABABABABABABABABABABABABABABABABAAAA

AAAAA

2.3.5.3 วิธีการใช้ฟังก์ชัน Join ()

โดยปกติ Thread แม่ควรจะต้องหยุดรอให้ Threads ลูกทำงานจบเสียก่อนจึงจะได้ หาก Thread แม่จบการทำงานก่อน Threads ลูก ทรัพยากรที่ Threads ลูกใช้จะไม่ถูกจัดคืนไปใช้ใหม่

ในการทำให้ Thread แม่หยุดรอ Threads ลูกให้จบก่อนนั้น หาก Thread แม่ใช้ sleep() หรือ suspend() ก็ไม่สะดวก เพราะไม่ทราบว่าจะต้องหยุดรอเป็นเวลาเท่าใด Threads ลูกเหล่านั้นจึงจะทำงานจบทั้งหมด ในคลาส Thread จึงมี

public final synchronized void join(long m) throws InterruptedException

เมื่อ Thread หนึ่งเรียก join() ของอีก Thread หนึ่ง จะทำให้ Thread ผู้เรียกหยุดการทำงานไปจนกว่า Thread ที่ถูกเรียกจะทำงานเสร็จสิ้นหรือตายไปก่อนแล้ว Thread ผู้เรียกจึงจะทำงานต่อจากจุดที่เรียกนั้น

2.3.5.4 วิธีการใช้ฟังก์ชัน Yield ()

ในคลาส Thread มี public static native void yield()

สำหรับให้ Thread ที่กำลังทำงานอยู่นั้นออกจากสถานะ running ไปสู่สถานะ ready เพื่อให้ Thread อื่นเข้าทำงานและมันจะถูกเลือกกลับมาทำงาน อีกเมื่อโอกาส เราใช้ yield() ในกรณีที่ Threads นั้นทำงานนานมากๆ หรืองานที่ไม่ค่อยสำคัญนัก ก็ควรออกจากสถานะ running เองเป็นช่วงๆ เพื่อให้ Threads อื่นซึ่งอาจจะมีงานที่สำคัญกว่าเข้ามาทำงาน

2.3.5.5 วิธีการใช้ฟังก์ชัน Stop ()

เมื่อ Threads เข้าสู่สถานะ death ก็จะถูกนำทรัพยากรกลับคืนไป ไม่อาจกลับเข้าสู่สถานะ ready ได้อีก การเข้าสู่สถานะ death นั้นมีได้สองลักษณะ กรณีแรกคือ เมื่อสิ้นสุดการทำงานใน run() ของ Thread นั้น ถือว่าเป็นการจบลงอย่างปกติ กรณีที่สองคือเมื่อ Thread นั้นถูกเรียกคำสั่ง stop() แม้ว่า Thread นั้นอาจเรียก stop() ของตัวเองได้ แต่การปล่อยให้จบลงอย่างปกติจะง่ายกว่า โดยทั่วไปจึงมักเป็น Thread อื่นเรียก stop() ของ Thread ที่จะหยุดการทำงาน

กรณีที่ Thread เข้าสู่สถานะ death เองโดยการจบการทำงานอย่างปกติ ผู้โปรแกรมจะสามารถกำหนดกิจกรรมบางอย่างให้ Thread นั้นทำก่อนเข้าสู่สถานะ death ได้เช่น ทำการปิดไฟล์ หรือ socket แต่ถ้าเป็นกรณีที่ Thread นั้นถูกอีก Thread หนึ่งเรียก stop() ให้หยุดการทำงานนั้นผู้โปรแกรมไม่ทราบว่าการสั่ง stop() จะเกิดขึ้นเมื่อใด จึงเกิดปัญหาว่าหากต้องกำหนดกิจกรรมบางอย่างให้ Thread นั้นทำก่อนเข้าสู่สถานะ death จะวางโปรแกรมส่วนนั้นไว้ที่ใด ภาษา Java แก้ปัญหานี้โดย เมื่อมีคำสั่ง stop() แก่ Thread หนึ่งจะมี instance ของคลาส ThreadDeath

ถูกโยนให้แก่ Thread นั้น สังเกตว่า ThreadDeath เป็นคลาส ลูกของคลาส Error ปกติ โปรแกรมของ Thread จะไม่ดักจับ error นี้ ปล่อยให้ขึ้นสู่ Java interpreter ซึ่งจะส่งผลให้ Thread นั้นหยุดการทำงานลง แต่ในกรณีที่ Thread ต้องทำอะไรบางอย่างก่อนจะหยุดการทำงาน เราก็อาจใช้ catch block ของการดักจับ error นี้ทำการจัดการสิ่งที่ต้องการ

2.3.5.6 วิธีการใช้ฟังก์ชัน Daemon Threads

มีหลายกรณีที่ Thread หนึ่งต้องสร้าง Threads ลูกขึ้นมาเพื่อคอยสนับสนุนการทำงานของมัน โดยปกติ Threads ลูกพวกนี้จะถูกสร้างขึ้นมาในระหว่าง Thread แม่ทำงาน และจะทำงานร่วมกับ Thread แม่ไปจนเมื่อ Thread แม่จะสิ้นสุดการทำงานก็สั่ง stop() Threads เหล่านั้น แต่ในบางกรณีอาจยุ่งยากที่จะเก็บ references ของ Threads ลูกทั้งหมดที่สร้างขึ้น และ การตรวจสอบว่า Threads ใดตายไปแล้วหรือยัง หรือจะมีปัญหาถ้า Threads แม่จบการทำงานอย่างไม่ปกติโดยไม่มีโอกาสได้เรียก stop() ของ Threads ลูกที่มันสร้างขึ้น เราเรียก Threads ลูกที่ยังทำงานอยู่ แต่ Thread แม่ตายของมันไปแล้วว่า orphan Threads ซึ่ง Threads แบบนี้มักจะใช้ทรัพยากรเครื่องอย่างไม่ก่อให้เกิดประโยชน์

โดยปกติ หากยังมี Thread ใด Thread หนึ่งยังทำงานอยู่ Java virtual machine ก็จะปล่อยให้ทำงานต่อไปเรื่อยๆ แต่ ภาษา Java เสนอ Threads พิเศษขึ้นเรียกว่า daemon Threads หาก Java virtual machine พบว่า Threads ที่ยังทำงานอยู่นั้นมีแต่ daemon Threads เท่านั้น ไม่มี Threads ธรรมดาทำงานอยู่เลย มันก็จะหยุดตัวเองลง ส่งผลให้ daemon Threads เหล่านั้นตายโดยอัตโนมัติ ดังนั้นหาก Thread แม่ตัวหนึ่งมี Threads ลูกที่ทำงานสนับสนุนเป็น daemon Threads ทั้งหมด สมมติว่ามีเพียง Thread แม่ตัวเดียวที่เป็น Thread ปกติ เมื่อ Thread แม่ตาย Threads ลูกทั้งหมดก็จะตายลงเองด้วย

2.3.5.7 วิธีการใช้ฟังก์ชัน Racing Condition

หากมี Threads ตั้งแต่สองตัวขึ้นไป แย่งกันใช้ทรัพยากรหนึ่งอย่างไม่ถูกควบคุมก็อาจจะเกิดปัญหาขึ้นได้ เนื่องจากลำดับการใช้งานทรัพยากรนั้นของแต่ละ Thread อาจมีความสำคัญต่อผลลัพธ์ของโปรแกรมในบางกรณีอาจทำให้ โปแกรมเดียวกันให้ผลลัพธ์แตกต่างกันในการทำงานแต่ละครั้ง ปรากฏการณ์นี้เรียกว่า racing condition ของ Threads ในโปรแกรมนั้น

2.3.5.8 วิธีการใช้ฟังก์ชัน Synchronization

ภาษา Java มีกลไกสำหรับการควบคุมให้ที่เวลาหนึ่ง ทรัพยากรนั้นถูกใช้งานได้โดย Thread เดียวเท่านั้น โดยนำกลไก mutually exclusive ของระบบปฏิบัติการมาใช้ นั่นคือ หากนำ mutually exclusive ไปครอบไว้กับทรัพยากรใด จะจำกัดให้ที่เวลาหนึ่งมีเพียง Thread เดียวเท่านั้นที่สามารถเข้าครอบครองและใช้งานทรัพยากรนั้น ระหว่างที่ทรัพยากรนั้นมี Thread หนึ่ง ครอบครองอยู่ หากมี Thread อื่นต้องการใช้งานทรัพยากรนั้นก็จะหยุดรอจนกว่า Thread ที่ใช้งานอยู่จะยอม

ปล่อยทรัพยากรนั้นออกจากการครอบครองได้เพียงผู้เดียวในเวลาใดเวลาหนึ่งว่า mutually exclusive

ภาษา Java สามารถทำให้เกิด mutually exclusive ขึ้นกับ instance หนึ่งได้ หากคลาสของ instance นั้นมี method หนึ่งที่ถูกระบุด้วย keyword "synchronized" และเมื่อ method นั้นถูกเรียกจาก Thread ใดๆก็ตาม จะทำให้ที่เวลาหนึ่งมีเพียง Thread เดียวเท่านั้นที่ทำงาน method นั้นได้

สังเกตว่า เฉพาะสมาชิกที่เป็น methods เท่านั้น จึงจะมี keyword "synchronized" ได้ ไม่อาจใช้กับสมาชิกที่เป็น data members

เมื่อ Thread หนึ่งเรียก synchronized method หากตอนนั้นยังไม่มี Thread ใดทำงาน method นั้น Thread นั้นก็จะได้ทำงานหรือเรียกว่าครอบครอง แต่หากในระหว่างนั้นมี Thread อื่นกำลังทำงาน method นั้นอยู่ Thread ที่พยายามเรียกก็จะต้องรอจนกว่า Thread ที่ทำงานอยู่จะสิ้นสุดการทำงานใน method นั้น

2.3.5.9 วิธีการใช้ฟังก์ชัน Wait() and Notify()

ในคลาส Object มี wait(), notify() และ notifyAll() ทำให้ทุกๆ instance ของคลาสใดๆมี methods เหล่านี้เสมอ สังเกตว่า methods ทั้งสามนั้นถูกกำหนดเป็น final จึงไม่สามารถ override เปลี่ยนแปลงเป็นอย่างอื่นได้ methods เหล่านี้ใช้สำหรับจัดลำดับ Threads เข้าใช้งานทรัพยากรที่อยู่ใน mutually exclusive ดังจะอธิบายต่อไปนี้

```
public final void wait() throws InterruptedException
public final native void wait(long timeout) throws InterruptedException;
public final void wait(long timeout, int nanos) throws InterruptedException
```

wait() ใช้สำหรับทำให้ Thread หนึ่งเข้าสู่สถานะ blocked จนกว่าจะถูก Thread อื่นมาเรียก notify() หรือ interrupt() ของ Thread ที่กำลังรออยู่นั้น ก่อนที่ Thread หนึ่งจะเรียก wait() มันจะต้องได้ครอบครองทรัพยากรหนึ่งที่เป็น mutually exclusive อยู่ก่อน นั่นคือ wait() จะต้องถูกเรียกจากใน method ที่ถูกระบุเป็น synchronized หรือ อยู่ภายใน synchronized block เท่านั้น หาก Thread นั้นไม่ได้ยึดครองทรัพยากรใดในตอนที่เรียก wait() จะเกิด IllegalMonitorState-Exception ขึ้น แต่ exception นี้เป็นคลาสลูกของคลาส RuntimeException จึงอาจไม่ต้องมี catch block สำหรับจับก็ได้ เมื่อ Thread ออกจากสถานะ blocked (เพราะมี Thread อื่นเรียก notify() หรือ interrupt()) Thread นั้นจะแข่งกับ Threads อื่นกลับเข้าไป ครอบครองทรัพยากรนั้นอีก สำหรับ wait() ที่มี long timeout เป็นพารามิเตอร์ จะรออยู่ในสถานะ blocked เป็นเวลาไม่เกิน timeout msec เมื่อหมดเวลาจะออกจากสถานะ blocked เอง แต่ถ้าระหว่างรออยู่นั้นมีการ notify() หรือ interrupt() ก็จะทำงานเหมือนกับ wait() หาก timeout เป็นศูนย์ ถือว่าไม่มี timeout

ส่วน wait() ที่มีทั้ง long timeout และ int nanos เป็นพารามิเตอร์นั้นทำงานคล้ายกับ wait() ตัวที่มี long timeout แต่สามารถระบุ nanos เพิ่มให้แก่ timeout มีหน่วยเป็น nsec โดยปกติระบบปฏิบัติการไม่สามารถจัดการได้ละเอียดถึง nsec ดังนั้นจึงยังไม่มีการใช้งาน wait() แบบนี้เลย

public final native void notify() ;

notify() ใช้สำหรับให้ Thread ที่กำลังได้ครอบครองทรัพยากรหนึ่ง บอกกับ Threads ที่กำลังรออยู่ในสถานะ blocked เพื่อจะใช้ทรัพยากรนั้นว่า จะมีการปล่อยทรัพยากรนี้แล้ว หากมีเพียง Thread เดียวที่กำลังรอใช้ทรัพยากรนั้น Thread นั้นจะได้เข้าใช้งานทรัพยากร หากมีมากกว่าหนึ่ง Threads กำลังรออยู่ ก็จะเลือกเพียง Thread เดียวให้เข้าใช้งานทรัพยากรนั้น การเลือกว่าจะให้ Thread ใดเข้าทำงานนี้ ไม่อยู่ในการควบคุมของผู้โปรแกรม หากมีการเรียก notify() ในขณะที่ไม่มี Thread รอใช้งานทรัพยากรนั้น ก็จะไม่มีอะไรเกิดขึ้น เมื่อตอนที่ Thread หนึ่งจะเรียก notify() มันต้องได้ครอบครองทรัพยากรนั้นอยู่ก่อน มิเช่นนั้นจะเกิด IllegalMonitorState- Exceptionขึ้น

public fianl native void notifyAll();

notifyAll() ใช้ในหน้าที่คล้ายกับ notify() แต่เมื่อ notifyAll() ถูกเรียก ทุกๆ Threads ที่กำลังรอใช้ทรัพยากรนั้นจะถูก notified และจะมีเพียง Thread เดียวเท่านั้นที่ได้เข้าใช้งานทรัพยากรนั้น ส่วน Threads อื่นที่เหลือจะถูก wait() และกลับเข้าสู่สถานะ blocked เพื่อรอใช้ทรัพยากรนั้น การใช้งาน notifyAll() ทำให้ทุกๆ Threads ที่กำลังรออยู่ มีโอกาสถูกเลือกเข้าทำงานเท่าๆกัน แต่ก็ทำงานช้ากว่า notify()

2.3.5.10 วิธีการใช้ฟังก์ชัน Deadlock

หากมี Thread หนึ่งต้องรอให้อีก Thread หนึ่งทำงานสิ้นสุดลงเสียก่อนจึงจะทำงานต่อไปได้ เราเรียกว่า Thread นั้นขึ้นกับ (depend on) อีก Thread หนึ่ง โปรแกรม multiThreading ที่มีการขึ้นต่อกันระหว่าง Threads อาจมีปัญหาเกิดขึ้นในระหว่างทำงานได้ เช่นหาก Thread หนึ่งได้ครอบครองทรัพยากรที่เป็น mutually exclusive และถ้า Thread นั้นไม่สามารถจบการทำงานลงได้ Threads อื่นๆที่รอใช้ทรัพยากรนั้นก็จะไม่สามารถทำงานได้ ปัญหาแบบนี้ถือเป็นความผิดพลาดของการสร้างโปรแกรมที่พอตรวจสอบและแก้ไขได้ แต่ก็มีปัญหาอีกแบบหนึ่งที่ตรวจสอบและแก้ไขได้ยากกว่าคือ ปัญหา deadlock ซึ่งเกิดขึ้นกับ Threads มากกว่าหนึ่งตัวขึ้นไปที่ไม่สามารถทำงานต่อไปได้ โดยจะอยู่ใน blocked state ไปอย่างถาวร

Deadlock จะเกิดขึ้นได้เมื่อ Thread อย่างน้อยสอง Threads สมมติว่าเป็น Aและ B โดยที่ Thread A ได้ครอบครองทรัพยากร X ที่เป็น mutually exclusive และ Thread B ได้ครอบครองทรัพยากร Y ที่เป็น mutually exclusive เช่นกัน แต่ใจขณะนั้น Thread A ต้องการใช้ทรัพยากร Y และ Thread B ก็ต้องการใช้ทรัพยากร X ทำให้ Threads ทั้งสองต้องเข้าสู่สถานะ blocked เพื่อรอให้ทรัพยากร ที่รอใช้งานถูกปล่อยออกมาจึงจะทำงานต่อไปได้ แต่ทั้ง Thread A และ B จะไม่

ปล่อยทรัพยากรที่ตัวครอบครองอยู่เนื่องจากยังทำงานไม่เสร็จ นั่นคือ Threads ทั้งสองกำลังรอคอยสิ่งที่จะไม่เกิดขึ้น จึงต้องอยู่ในสถานะ blocked ตลอดไป

2.3.5.11 วิธีการใช้ฟังก์ชัน Interrupt ()

ทุกๆ Thread จะมี flag หนึ่งสำหรับบอกว่า Thread นี้ถูกสั่งขัดจังหวะแล้วหรือไม่ เมื่อ Thread หนึ่งถูก Thread อื่นหรือตัวเองเรียก

public void interrupt()

จะทำให้ interrupt flag ของ Thread นั้นเป็น true และตรวจสอบได้โดยใช้

public boolean isInterrupt()

ซึ่งจะให้ true ถ้า Thread นั้นถูกเรียก interrupt() แล้ว และจะให้ false ถ้า Thread นั้นยังไม่ถูกเรียก interrupt() หรืออาจใช้อีก method หนึ่งที่กำหนดอยู่ในคลาส Thread คือ

public static boolean interrupted()

ซึ่งจะให้ true ถ้า Thread ที่เรียก method นี้ถูกเรียก interrupt() แล้ว และจะให้ false ถ้า Thread ที่เรียก method นี้ยังไม่ถูกเรียก interrupt() และ Thread ที่กำลังทำงานอยู่ เมื่อถูกเรียก interrupt() ก็จะยังทำงานต่อไป

บทที่ 3
การออกแบบและพัฒนาโปรแกรม

3.1 ขอบเขตการทำงาน

1. สร้างบทละครของเกม อธิบายรายละเอียดของเกม
2. ออกแบบโปรแกรม คลาสต่าง ๆ จากบทละครของเกม

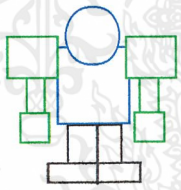
3.1.1 บทละคร (Scenario) ของเกม

เป็นเกมการต่อสู้ของหุ่นยนต์ 2 ฝ่าย โดยในระหว่างการต่อสู้นั้นจะเป็นการทำงานของหุ่นยนต์เองผู้เล่นไม่สามารถบังคับหุ่นยนต์ได้ การทำงานของหุ่นยนต์นั้นจะถูกเขียนคำสั่งควบคุมในเหตุการณ์ต่างๆ เพื่อบังคับหุ่นยนต์ไว้ล่วงหน้า เมื่อเกิดเหตุการณ์ใดๆ แล้ว หุ่นยนต์จะทำงานตามที่ได้เขียนคำสั่งไว้ในเหตุการณ์ที่เกิดขึ้น หุ่นยนต์ตัวใดที่ถูกสร้างให้มีความฉลาดในการต่อสู้มากกว่าก็จะเป็นผู้ชนะชนะในที่สุด

เกมจะประกอบด้วยส่วนของหุ่นยนต์และสนามรบ

3.1.1.1 ลักษณะและรายละเอียดของหุ่นยนต์

หุ่นยนต์จะประกอบด้วยส่วนต่างๆ คือ สมองหรือ CPU ลำตัวซึ่งติดกับหัว แขน 2 ข้าง ขา 2 ข้าง อาวุธระยะใกล้ อาวุธระยะไกลและไอพ่น ดังรูปที่ 3.1

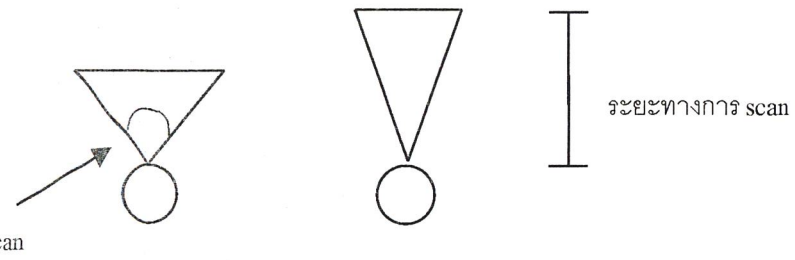


รูปที่ 3.1 แสดงส่วนประกอบของหุ่นยนต์

โดยส่วนของ CPU และลำตัวนั้นจำเป็นจะต้องมีในการสร้างขึ้นเป็นหุ่นยนต์ ส่วนส่วนอื่นๆไม่จำเป็นที่จะต้องมีก็จะถือว่าเป็นหุ่นยนต์ แต่ถ้าไม่มีส่วนใดจะไม่สามารถใช้งานและความสามารถในส่วนนั้นๆได้ แต่ละส่วนจะมีรายละเอียดดังนี้

CPU ใช้ในการประมวลผลเกี่ยวกับการ scan มีรายละเอียด คือ

- การ scan แบ่งเป็นระยะทางหรือความยาวของการ scan และมุมของการ scan ดังรูปที่ 3.2



รูปที่ 3.2 แสดงลักษณะ scan ของหุ่นยนต์

- Ability คือความสามารถในการใช้อาวุธที่ถนัด แบ่งเป็นถนัดในระยะใกล้และไกล ซึ่งจะแปรผันตรงกับความน่าจะเป็นในการยิงถูกคู่ต่อสู้ ความถนัดระยะใกล้ (near ability) จะเป็นความสามารถในการใช้อาวุธระยะใกล้ (short weapon) ซึ่งจะบอกเป็นเปอร์เซ็นต์ถ้ายิ่งมีเปอร์เซ็นต์มากจะทำให้ใช้ short weapon ได้แม่นยำขึ้น จะมีผลกับการใช้ short weapon โจมตีคู่ต่อสู้ โดยจะใช้ในการคำนวณว่าจะยิงโดนหรือไม่ ความถนัดระยะใกล้ (far ability) จะเป็นความสามารถในการใช้อาวุธระยะไกล (long weapon) ซึ่งจะบอกเป็นเปอร์เซ็นต์เหมือนกับความถนัดระยะใกล้ ถ้ายิ่งมีเปอร์เซ็นต์มากก็จะทำให้ใช้ long weapon แม่นยำขึ้น มีผลกับการใช้ long weapon โจมตีคู่ต่อสู่ว่าจะโดนหรือไม่

ลำตัว (Body) ใช้สำหรับเป็นส่วนกลางในการนำส่วนต่างๆ มาประกอบเป็นหุ่นยนต์ ถ้าไม่มีลำตัวจะไม่สามารถติดส่วนประกอบอื่นได้ มีค่าประจำหรือ attribute คือ พลังงานหรือ Energy ขนาดหรือ Size และน้ำหนักหรือ Weight

แขนซ้าย(Left Arm) และแขนขวา (Right Arm) เป็นส่วนสำหรับใช้ติดอาวุธ แขนแต่ละข้างจะสามารถติดอาวุธระยะใกล้ได้ 1 ชิ้นและติดอาวุธระยะไกลได้ 1 ชิ้น ค่าประจำตัวหรือ attribute ของแขนซ้ายและแขนขวาจะประกอบด้วย พลังงานหรือ Energy ขนาดหรือ Size และน้ำหนักหรือ Weight เช่นเดียวกับส่วนลำตัว

ขา (Leg) เป็นส่วนที่ใช้ในการเคลื่อนที่ของหุ่นยนต์ มีค่าประจำตัวหรือ attribute คือพลังงานหรือ Energy ขนาดหรือ Size น้ำหนักหรือ Weight และความเร็วหรือ speed ความเร็วของการเคลื่อนที่โดยใช้ขานั้นจะแปรผกผันกับน้ำหนักรวมของหุ่นยนต์

พลังงานหรือ Energy นั้นจะแปรผันตรงกับน้ำหนักหรือ weight

ไอพ่น (Jet) ใช้สำหรับการเคลื่อนที่ของหุ่นยนต์เช่นเดียวกับขาแต่จะมีความเร็วสูงกว่าการใช้ขา มี ค่า attribute ดังนี้

- ความเร็วหรือ Speed ระดับที่มีความเร็วสูงจะมีปริมาณน้ำมันน้อย ถ้าความเร็วต่ำจะมีปริมาณน้ำมันมาก น้ำหนักรวมของหุ่นยนต์จะไม่ผลต่อความเร็วของไอพ่น
- น้ำมันหรือ Oil จำนวนน้ำมันจะลดลงไปถ้ามีการเคลื่อนทีโดยใช้ไอพ่น

- น้ำหนักหรือWeight

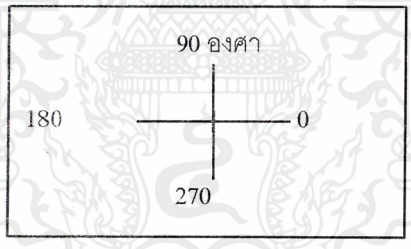
อาวุธหรือ Weapon แบ่งเป็นอาวุธระยะใกล้หรือ Short Weapon และอาวุธระยะไกลหรือ Long Weapon ประกอบไปด้วยค่า attribute ดังนี้

- Weight คือน้ำหนักของอาวุธชนิดนั้น
- Destroy Power คือพลังทำลายของอาวุธชนิดนั้น
- Distance คือระยะทางที่อาวุธนั้นสามารถยิงไปได้ไกลสุด
- Bullet Amount คือจำนวนกระสุนของอาวุธชนิดนั้น
- Bullet Speed คือความเร็วของกระสุนของอาวุธชนิดนั้น
- Shooting Rate คืออัตราการยิงอาวุธชนิดนั้นว่าในการยิงครั้งต่อไปต้องใช้เวลาเท่าไร

อาวุธระยะใกล้ จะมีพลังการทำลายต่ำแต่จำนวนกระสุนจะมากและมีอัตราการยิงที่เร็ว ส่วนอาวุธระยะไกลนั้นจะมีพลังการทำลายมากแต่จำนวนกระสุนจะน้อยและมีอัตราการยิงที่ใช้เวลานานกว่าอาวุธระยะใกล้

3.1.1.2 ลักษณะของสนามรบ

สนามรบเป็นพื้นที่ที่ใช้ในการต่อสู้ของหุ่นยนต์และแสดงผลทั้งหมดในการต่อสู้ มีพื้นที่เป็นสี่เหลี่ยมและมีทิศทางเป็นองศา ดังรูป



รูปที่ 3.3 แสดงลักษณะสนามรบ

3.1.1.3 กติกา วิธีในการเล่นเกม และรายละเอียดการทำงาน

หุ่นยนต์จะประกอบไปด้วยส่วนต่างๆตามที่ได้กล่าวมาแล้ว ส่วนที่มีความสำคัญมากที่สุดคือส่วนลำตัว ถ้าลำตัวถูกทำลายแล้วจะถือว่าหุ่นยนต์แพ้ทันที หุ่นยนต์จะสามารถติดอาวุธได้อย่างมากที่แขนข้างละ 2 ชิ้น คืออาวุธใกล้และอาวุธไกลตามที่ได้กล่าวมาแล้ว ถ้าแขนข้างใดถูกทำลายจนพลังงานหมดจะไม่สามารถใช้อาวุธที่ติดอยู่กับแขนข้างนั้นได้ ขาเป็นส่วนที่มี speed พื้นฐานในการเคลื่อนที่ของหุ่นยนต์อยู่ ถ้าขาถูกทำลายจนพลังงานหมดแล้วจะไม่สามารถเคลื่อนที่โดยใช้ขาด้วย speed พื้นฐานนี้ได้ ฉะนั้นในกรณีนี้หุ่นยนต์จะสามารถเคลื่อนที่ได้เมื่อใช้ไอพ่นเท่านั้น และถ้าน้ำมันหมดก็จะไม่สามารถใช้ไอพ่นในการเคลื่อนที่ได้

การเริ่มเล่นเกมนั้นผู้เล่นจะต้องสร้างหุ่นยนต์ขึ้นด้วยส่วนประกอบต่างๆที่ได้กล่าวมาแล้ว โดยโปรแกรมสำหรับช่วยสร้างหุ่นยนต์ ส่วนประกอบต่างๆนั้นจะมีหลายประเภทให้เลือกเพื่อประกอบขึ้นเป็นหุ่นยนต์ แต่ละประเภทจะมีค่า attribute ที่แตกต่างกันไป เมื่อเลือกส่วนประกอบต่างๆได้แล้ว ผู้เล่นจะสามารถเขียนคำสั่งเพื่อควบคุมการทำงานให้หุ่นยนต์ได้ โดยจะเขียนคำสั่งในเหตุการณ์ต่าง ๆ ที่อาจจะเกิดขึ้น จากนั้นผู้เล่นจะต้องทำการ compile เพื่อให้ได้ไฟล์ที่มีส่วนขยายเป็น .class และต้องส่งไฟล์นี้ไปให้กับสนามรบเพื่อให้สนามรบแสดงการต่อสู้ของหุ่นยนต์ ซึ่งการต่อสู้จะต้องเลือกหุ่นยนต์ 2 ตัวก่อนเพื่อจะเริ่มการต่อสู้ได้

การทำงานที่จะเกิดในสนามรบซึ่งสนามต้องทำการควบคุม แบ่งได้เป็น 5 ส่วนใหญ่ๆ ดังนี้

1. การเคลื่อนที่ สิ่งที่สัมพันธ์กันคือน้ำหนักของหุ่นยนต์และความเร็ว ซึ่งจะมีสมการในการคิดสำหรับการเคลื่อนที่ หุ่นยนต์จะส่งค่าพารามิเตอร์ของพิกัดที่จะเคลื่อนที่หรือค่าองศาและระยะทางจากจุดปัจจุบันไปยังสนามรบเพื่อให้สนามรบเป็นตัววาดกราฟิก

การชนขอบของสนามรบหรือชนกันเองระหว่างหุ่นยนต์จะทำให้หุ่นยนต์หยุดและเสียพลังงานในทุกส่วนโดยถ้ามีไอพ่นเป็นส่วนประกอบด้วยก็จะเสียปริมาณน้ำมันของไอพ่นไปเรื่อยๆ

2. การ scan มีการกำหนดพื้นที่สามเหลี่ยมของการ scan และมุมมองในการมองเห็นของหุ่นยนต์ซึ่งทุกตัวจะมีค่าเท่ากัน เมื่อมีการ scan พบศัตรู จะสามารถ get ค่าต่างๆได้ เช่น ตำแหน่งของศัตรู ความเร็วของศัตรู พลังของแต่ละส่วนของศัตรู และยังสามารถ บอกประเภทของสิ่งที่ scan ได้ เช่น ขอบของสนาม หุ่นยนต์ที่พบเป็นพันธมิตรหรือศัตรู

3.การถูกโจมตี หุ่นยนต์จะถูกโจมตีเมื่อจากฝ่ายตรงข้ามมีการสั่งยิงและในการยิงนั้นกระสุนมาโดนหุ่นยนต์ทำให้หุ่นยนต์เสียพลังงานของส่วนที่โดนซึ่งจะเป็นส่วนใดก็ได้

4.การโจมตี หุ่นยนต์สามารถทำการโจมตีได้ทุกเมื่อ โดยจะต้องระบุเป้าหมายของการโจมตีด้วย

5.การจัดการเกี่ยวกับเหตุการณ์ต่างๆที่อาจจะเกิดขึ้นได้ในขณะที่มีการต่อสู้กัน โดยในแต่ละเหตุการณ์นั้นผู้เขียนจะต้องเขียนคำสั่งเพื่อสั่งให้หุ่นยนต์ทำงานตามที่ต้องการและสนามจะจัดการทำตามคำสั่งนั้นๆ

เหตุการณ์ที่สามารถเกิดขึ้นได้ในระหว่างการต่อสู้ของหุ่นยนต์ประกอบด้วย 6 เหตุการณ์ด้วยกัน คือ

- เหตุการณ์ปกติ (Common)
- เหตุการณ์พบศัตรู (Found Enemy)
- เหตุการณ์โดนโจมตี (Attracked)
- เหตุการณ์พบขอบสนาม (FoundBorder)
- เหตุการณ์ชนศัตรู (CrashEnemy)

- เหตุการณ์ชนขอบสนาม(CrashBorder)

ในแต่ละเหตุการณ์นั้นสามารถที่จะเกิดขึ้นพร้อมๆ กันได้แล้วแต่ว่าจะเกิดเหตุการณ์ใดขึ้นบ้างในเวลานั้น คำสั่งที่ถูกเขียนขึ้นในแต่ละเหตุการณ์นั้นจะถูกประมวลผลทีละคำสั่งแบบลำดับ โดยคำสั่งแรกจะถูกทำงานจนเสร็จสิ้นก่อน คำสั่งต่อไปจึงจะสามารถทำงานได้ โดยมีข้อกำหนดของการทำงานของเหตุการณ์ต่างๆดังนี้

-ในตอนเริ่มต้นของเกมนั้นเหตุการณ์แรกที่จะถูกประมวลผลหรือถูกเรียกให้ทำงานก่อนคือเหตุการณ์ปกติ (Common) และเมื่อสิ้นสุดคำสั่งการทำงานของเหตุการณ์ปกติทั้งหมดแล้ว จะมีการทำงานซ้ำอีกครั้งตั้งแต่คำสั่งแรกจนถึงคำสั่งสุดท้ายอีกครั้งโดยอัตโนมัติ การทำงานจะวนอย่างนี้ไปเรื่อย ๆ จนกว่าจะจบเกมซึ่งในระหว่างนั้นก็สามารถเกิดเหตุการณ์อื่นพร้อม ๆ กันได้

- เมื่อเกิดเหตุการณ์พบศัตรูแล้วจะทำการประมวลผลคำสั่งที่ได้เขียนไว้จนกว่าจะหลุดจากเหตุการณ์พบศัตรู เมื่อหลุดจากเหตุการณ์นี้แล้ว คำสั่งของเหตุการณ์นี้จะถูกหยุดการประมวลผลทันที เมื่อเกิดเหตุการณ์นี้ใหม่อีกครั้งจะเริ่มทำการประมวลผลคำสั่งใหม่ตั้งแต่ต้นอีกครั้งหนึ่ง หากการทำงานสิ้นสุดลงหรือจบชุดคำสั่งทั้งหมดแล้ว ยังคงเกิดเหตุการณ์พบศัตรูอยู่จะมีการประมวลผลตั้งแต่คำสั่งแรกใหม่ตั้งแต่ต้นอีกครั้งหนึ่ง

- เหตุการณ์อื่นที่เหลือทั้งหมด คือ เหตุการณ์ โดนโจมตี เหตุการณ์พบขอบสนาม เหตุการณ์ชนศัตรู และเหตุการณ์ชนขอบสนาม เมื่อเกิดเหตุการณ์เหล่านี้ขึ้นแล้ว คำสั่งต่างๆที่ถูกเขียนไว้จะถูกประมวลผลทีละคำสั่งจนครบถึงคำสั่งสุดท้าย แม้ว่าในระหว่างที่ทำการประมวลผลคำสั่งต่างๆอยู่นั้น จะเกิดเหตุการณ์เหล่านี้ซ้ำอีกก็ตามจะถือว่าไม่เกิดผลใด ๆ คือจะยังทำการประมวลผลจนถึงคำสั่งสุดท้าย ต่อเมื่อประมวลผลคำสั่งจนครบหมดแล้ว เกิดเหตุการณ์นั้นอีกจึงจะเริ่มทำการประมวลผลคำสั่งใหม่อีกครั้งตั้งแต่คำสั่งแรก

3.1.2 การออกแบบโปรแกรมจากบทละคร

จากบทละครของเกมนั้นสามารถแบ่งออกได้เป็น 2 ส่วนหลักในการเล่นเกมคือ ส่วนที่ 1 จะต้องสร้างโปรแกรมที่ใช้สำหรับช่วยสร้างหุ่นยนต์โดยมีการเลือกส่วนประกอบต่างๆประกอบขึ้นเป็นหุ่นยนต์และเขียนโปรแกรมเพื่อสั่งให้หุ่นยนต์ทำงานในเหตุการณ์ต่างๆที่เตรียมไว้ ในส่วนนี้จะสร้างเป็น Java Application โดยใช้ Visual Cafe 4.0 ช่วยในการเขียนโปรแกรม ผลที่ได้จากการสร้างหุ่นยนต์ในส่วนนี้จะได้ไฟล์ที่ส่วนขยายเป็นจุดคลาสที่มีค่า attribute และคำสั่งการทำงานในเหตุการณ์ต่าง ๆ ของหุ่นยนต์ ส่วนที่ 2 เป็นส่วนของ web page ที่ใช้ในการแสดงรายละเอียดต่างๆของการต่อสู้ของหุ่นยนต์ 2 ตัวที่ได้เลือกไว้จากลิสต์ของหุ่นยนต์ที่ได้อัพโหลดมาแล้ว ใช้ในการดาวน์โหลดโปรแกรมที่ใช้สำหรับช่วยสร้างหุ่นยนต์ อัพโหลดไฟล์หุ่นยนต์ของผู้เล่น การแสดงผลเป็นรูปภาพ 2 มิติ ใช้ java swing ในการแสดงภาพ ส่วนนี้จะเป็นการเขียนโปรแกรมเป็น Java Applet

3.1.2.1 การออกแบบโปรแกรมสำหรับสร้างหุ่นยนต์

แบ่งออกเป็น 3 หน้า(Frame) ดังนี้

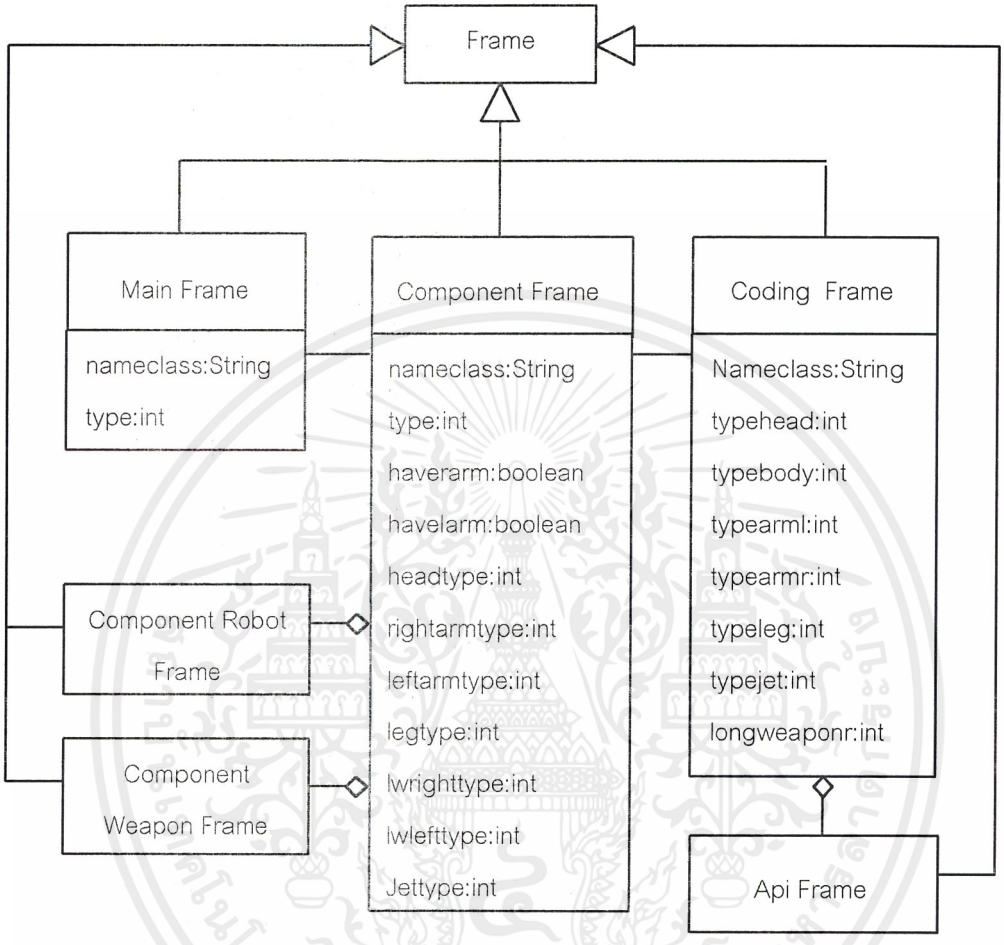
1.หน้าแรก เป็นหน้าหลักสำหรับเริ่มต้นสร้างหุ่นยนต์ โดยมีช่องสำหรับใส่ชื่อของหุ่นยนต์ตัวใหม่ พร้อมกับปุ่ม Next เพื่อเปลี่ยนไปยังหน้าถัดไป และมีช่องสำหรับให้ใส่ชื่อหุ่นยนต์ตัวเก่าที่เคยสร้างไว้แล้วเพื่อทำการแก้ไขเปลี่ยนแปลงได้ พร้อมกับปุ่ม Next เพื่อเปลี่ยนไปยังหน้าถัดไปอีกปุ่มหนึ่ง การทำงานของหน้า(Frame)นี้ จะมีการตรวจสอบชื่อของหุ่นยนต์ซึ่งกำหนดให้ชื่อของหุ่นยนต์ต้องขึ้นต้นด้วย "_" และตามด้วยตัวอักษรหรือตัวเลขอีก 7 ตัว นอกจากนี้ถ้าเป็นการเปิดตัวเก่าจะต้องค้นหาด้วยชื่อที่ใส่มา

2.หน้าที่สอง เป็นหน้าที่แสดงส่วนประกอบต่าง ๆ ของหุ่นยนต์เพื่อให้ผู้เล่นได้เลือกเพื่อประกอบขึ้นเป็นหุ่นยนต์ในกรณีสร้างหุ่นยนต์ตัวใหม่โดยการ new instance ให้โดยอัตโนมัติ หรือถ้าเป็นกรณีที่เป็นหุ่นยนต์ตัวเดิมจะแสดงส่วนประกอบต่างๆที่เคยได้เลือกไว้แล้วและสามารถเปลี่ยนแปลงได้ ส่วนประกอบต่าง ๆ ของหุ่นยนต์จะมี 3 ประเภทให้เลือกโดยจะมีค่า attribute แตกต่างกันไป ทำให้สามารถสร้างหุ่นยนต์ได้หลากหลายรูปแบบและมีความสามารถแตกต่างกันตามค่า attribute ที่แตกต่างกัน ดังนั้นในหน้านี้จะมี Choice ของส่วนประกอบต่าง ๆ มีปุ่ม Next เพื่อเปลี่ยนไปยังหน้าถัดไป และปุ่ม Back เพื่อกลับไปยังหน้าแรก ซึ่งถ้าไม่เลือกส่วน Body และ CPU จะไม่สามารถกดปุ่ม Next เพื่อเปลี่ยนไปยังหน้าถัดไปได้ เพื่อบังคับให้ผู้เล่นต้องทำการเลือกส่วน Body และ CPU ซึ่งจำเป็นต้องมีในหุ่นยนต์ทุกตัว และถ้าไม่เลือกแขนข้างใดก็จะไม่สามารถเลือกอาวุธของแขนข้างนั้นได้ด้วย นอกจากนั้นในหน้านี้จะมีปุ่ม Component of Robot สำหรับแสดงรายละเอียดของค่า attribute ของส่วนประกอบต่าง ๆ ของหุ่นยนต์ และปุ่ม Component of Weapon เพื่อแสดงรายละเอียดของค่า attribute ของอาวุธชนิดต่าง ๆ ด้วย ทั้ง 2 ปุ่มนี้เมื่อคลิกแล้วจะแสดงรายละเอียดในหน้า(Frame)ใหม่

3.หน้าที่สาม เป็นหน้าที่แสดง object ของส่วนประกอบต่าง ๆ ของหุ่นยนต์ตามที่ผู้เล่นได้เลือกไว้ซึ่งส่วนนี้จะไม่สามารถแก้ไขได้ ถ้าต้องการแก้ไขส่วนประกอบต้องกลับไปเลือกส่วนประกอบใหม่ที่หน้าที่สองเท่านั้น นอกจากนี้ยังเป็นหน้าสำหรับเขียนคำสั่งให้หุ่นยนต์ทำงานในเหตุการณ์ต่าง ๆ ด้วย ดังนั้นในหน้านี้จะมี Text Area สำหรับแสดงผล มีปุ่ม Attribute เพื่อให้เมื่อคลิกแล้วจะแสดง object ของส่วนประกอบต่าง ๆ ที่ผู้เล่นได้เลือกไว้ในพื้นที่ของ Text Area มีปุ่มของเหตุการณ์ต่าง ๆ 6 เหตุการณ์ เมื่อคลิกแล้วจะแสดงส่วนของ Method ของเหตุการณ์นั้นเพื่อให้ผู้เล่นได้เขียนคำสั่งควบคุม โดยแสดงรายละเอียดในพื้นที่ของ Text Area ที่กล่าวไปแล้ว นอกจากนี้ยังมีปุ่ม API เพื่อแสดง Method ที่ผู้เล่นสามารถเรียกใช้ได้ของคลาสต่าง ๆ ของส่วนประกอบของหุ่นยนต์ มีปุ่ม Generate เพื่อ Compile ไฟล์ของหุ่นยนต์นี้ให้เป็นไฟล์ที่มีนามสกุลจุดคลาสเพื่อส่ง

ไปทำงานใน web page และมี Text Area อีกส่วนหนึ่งสำหรับแสดงผลของการ Compile ในกรณีที่เป็นการเรียกหุ่นยนต์ตัวเก่านั้น จะสามารถแสดงคำสั่งต่าง ๆ ที่ผู้เล่นได้เคยเขียนไว้แล้วด้วย

Class Diagram ของโปรแกรมสำหรับสร้างหุ่นยนต์



รูปที่ 3.4 แสดง Class Diagram ของโปรแกรมสำหรับสร้างหุ่นยนต์

- ตัวแปร nameclass เป็นตัวแปรที่เก็บข้อมูลของชื่อหุ่นยนต์

- ตัวแปร type เป็นตัวแปรที่เก็บข้อมูลว่าการทำงานเป็นการสร้างหุ่นยนต์ตัวใหม่หรือใช้หุ่นยนต์เก่า - ตัวแปร headtype rightarmtype leftarmtype legtype lwrighttype lwlefttype Jettype shwrighttype shwlefttype bodytype ใน Component Frame เป็นตัวแปรที่เก็บรายละเอียดของส่วนประกอบของหุ่นยนต์ ซึ่งถ้าเป็นหุ่นยนต์ตัวเก่าค่าเหล่านี้จะถูกกำหนดไว้ เมื่อเฟรมถูกเปิดขึ้น

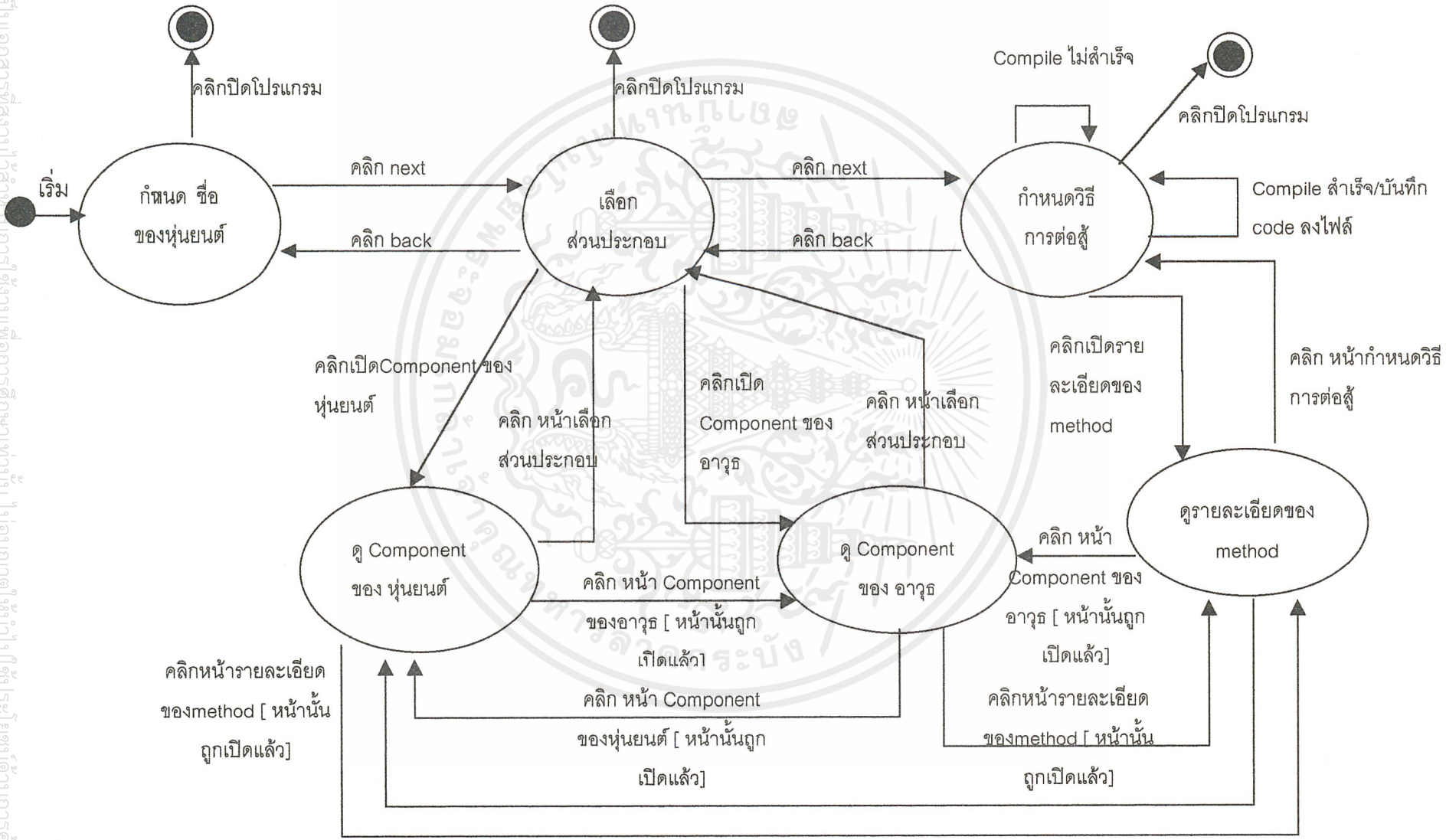
- ตัวแปร nameclass typehead typebody typearml typearmr typeleg typejet longweaponr longweaponl shortweaponl shortweaponr ใน Coding frame เป็นตัวแปรที่เก็บรายละเอียดของส่วนประกอบของหุ่นยนต์ ซึ่งถูกส่งมาจาก Component Frame

Main Frame จะเป็น Object ที่เรียก Component Frame เนื่องจากเวลาคลิก Next จะมีการสร้าง Object ของคลาส Component Frame และแสดงหน้า Component Frame ขึ้นมา ดังนั้น Main Frame จึงมีความสัมพันธ์แบบ 1-1 กับ Component Frame นอกจากนี้ Main Frame ต้องส่งข้อมูลของตัวแปร nameclass,type และ Object ของ Main Frame เองเพื่อให้สามารถคลิก Back เพื่อกลับมายังหน้าเดิมได้ ไปให้กับ Object ของ Class Component Frame ที่สร้างขึ้นมา

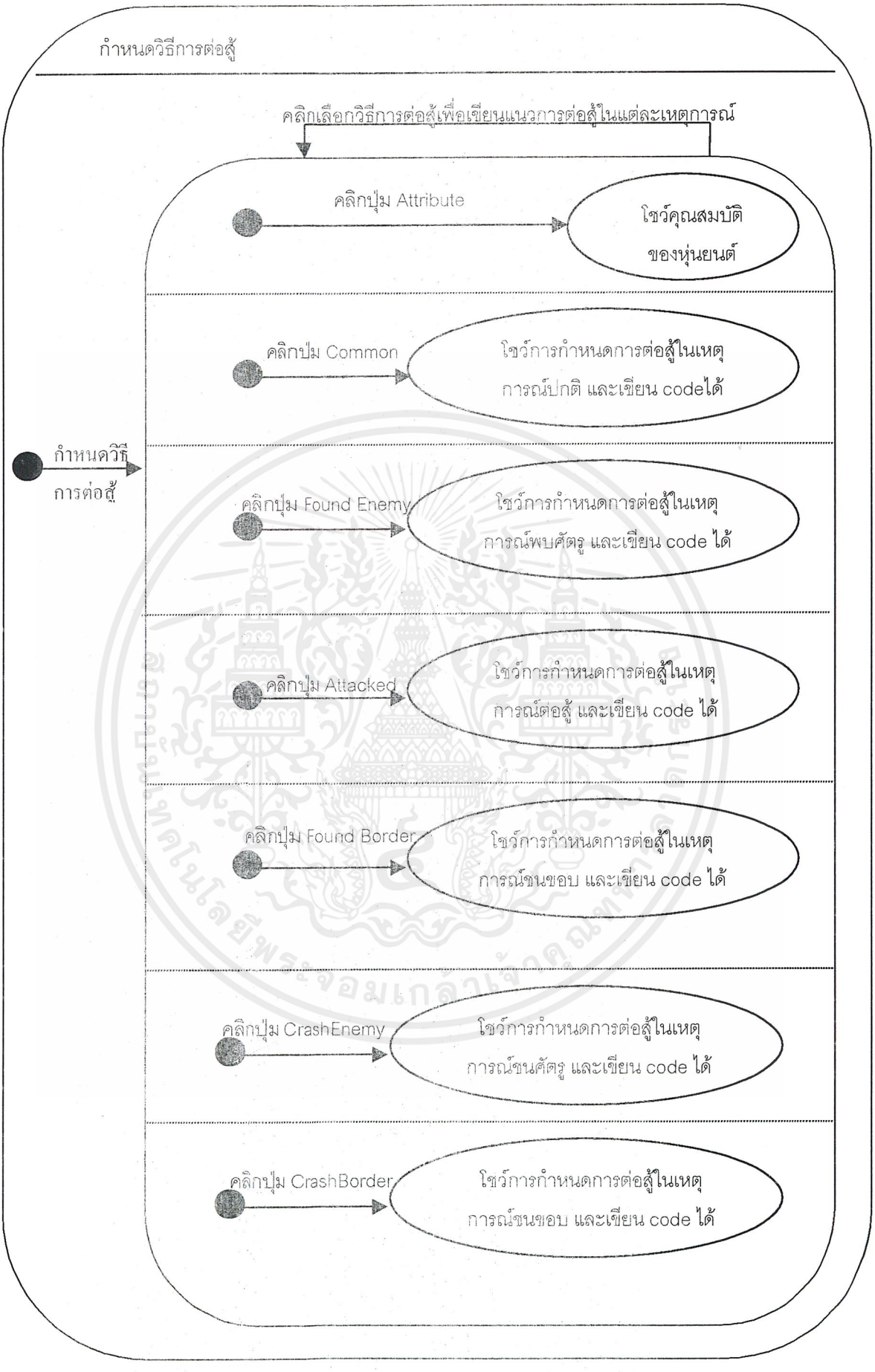
Component Frame จะมี Object ของคลาส Component Robot และ Component Weapon เป็นส่วนประกอบ เนื่องจากเวลาคลิกปุ่ม show Component Robot จะมีการเปิดหน้า Component Robot เช่นเดียวกับเวลาคลิก show Component weaponจะมีการเปิดหน้า Component weapon ดังนั้น Component Frame จึงมีความสัมพันธ์แบบ 1-1 กับ Component Frame และ Component Weapon นอกจากนี้คลาส Component Frame จะมีการสร้าง Object หรือ new instance ของคลาส Coding Frame เพื่อส่งข้อมูลของตัวแปร headtype rightarmtype leftarmtype legtype lwrighttype lwlefttype Jettype shwrighttype shwlefttype bodytype ซึ่งเป็นข้อมูลขององค์ประกอบของหุ่นยนต์ไปให้กับ Object ของคลาส Class Coding Frame ที่สร้างขึ้นซึ่งรับค่าของตัวแปรเหล่านี้โดย ตัวแปร nameclass typehead typebody typearml typearmr typeleg typejet longweaponr longweaponl shortweaponl shortweaponr และแสดงหน้าของ Object ของคลาส Coding Frame พร้อมทั้งส่ง Object ของคลาส Component Frame นี้ไปให้กับ Object ของคลาส Coding Frame ด้วยเพื่อให้สามารถที่จะคลิก Back เพื่อกลับไปยังหน้า Component Frame ตัวเดิมได้

คลาส Coding Frame จะมี Object ของคลาส API Frame เป็นส่วนประกอบ เมื่อต้องการดูรายละเอียดของ method ที่กำหนดให้กับผู้เล่นก็สามารถที่จะคลิกให้แสดงได้จากปุ่มที่เป็นส่วนประกอบในคลาส Coding Frame นี้

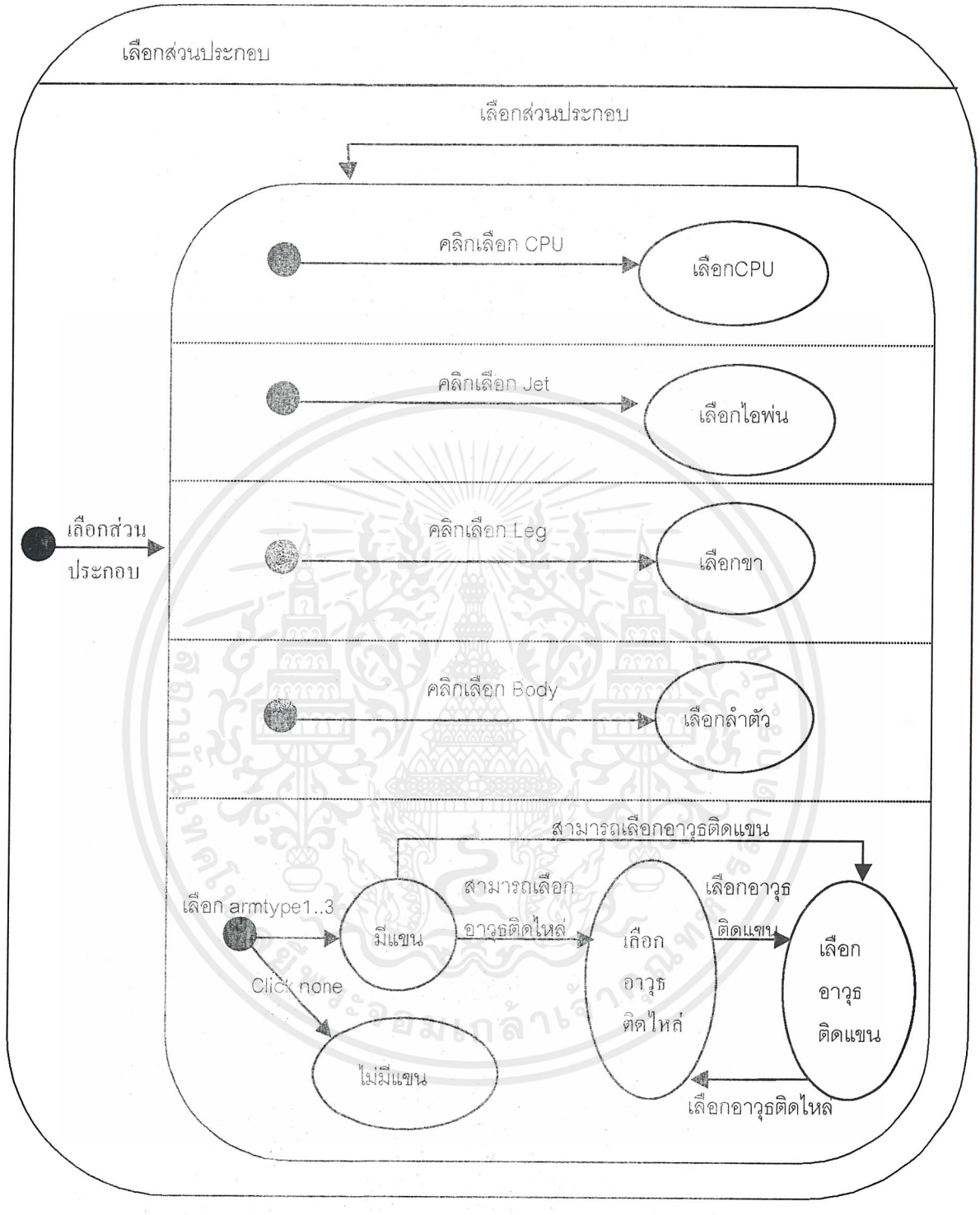
State Diagram ของโปรแกรมสำหรับสร้างหุ่นยนต์



รูปที่ 3.5 State Diagram ของ Application



รูปที่ 3.6 แสดง State Diagram ของ Application กำหนดวิธีการต่อสู้



รูปที่ 3.7 แสดง State Diagram ของ Application เลือกส่วนประกอบ

3.1.2.2 การออกแบบ Web Page

จะมีการแบ่งหน้า web ออกเป็น Frame โดย Frame ทางซ้ายจะเป็น link ต่าง ๆ Frame ตรงกลางเป็น Frame ที่แสดงรายละเอียดจาก link ของ Frame ทางซ้าย link ต่าง ๆ ประกอบด้วย

- Introduction แนะนำที่มาที่ไป จุดมุ่งหมายของเกม
- แนะนำเกมว่าเป็นอย่างไร
- วิธีการเล่น กฎ กติกา
- เมทอธที่ผู้เล่นสามารถเรียกใช้ได้
- link ให้ผู้เล่นดาวน์โหลดซึ่งประกอบด้วย โปรแกรม JDK ,โปรแกรมสำหรับสร้างหุ่นยนต์ ,คู่มือวิธีการเล่นและรายละเอียดของเมทอธที่ผู้เล่นสามารถเรียกใช้ได้
- link สำหรับอัพโหลดหุ่นยนต์เข้าสู่ server
- ปุ่มสำหรับแสดงรายชื่อของหุ่นยนต์ที่มีอยู่ที่ server จากการที่ได้อัพโหลดไว้ ซึ่งเมื่อคลิกแล้วก็จะแสดงรายชื่อของหุ่นยนต์เพื่อให้ผู้เล่นได้เลือก เมื่อผู้เล่นได้เลือกหุ่นยนต์แล้วสามารถคลิกปุ่มเพื่อเริ่มการต่อสู้ของหุ่นยนต์และแสดงรายละเอียดการต่อสู้ของหุ่นยนต์ในหน้าต่างใหม่เนื่องจากต้องการพื้นที่ในการแสดงผลขนาดเต็มจอ

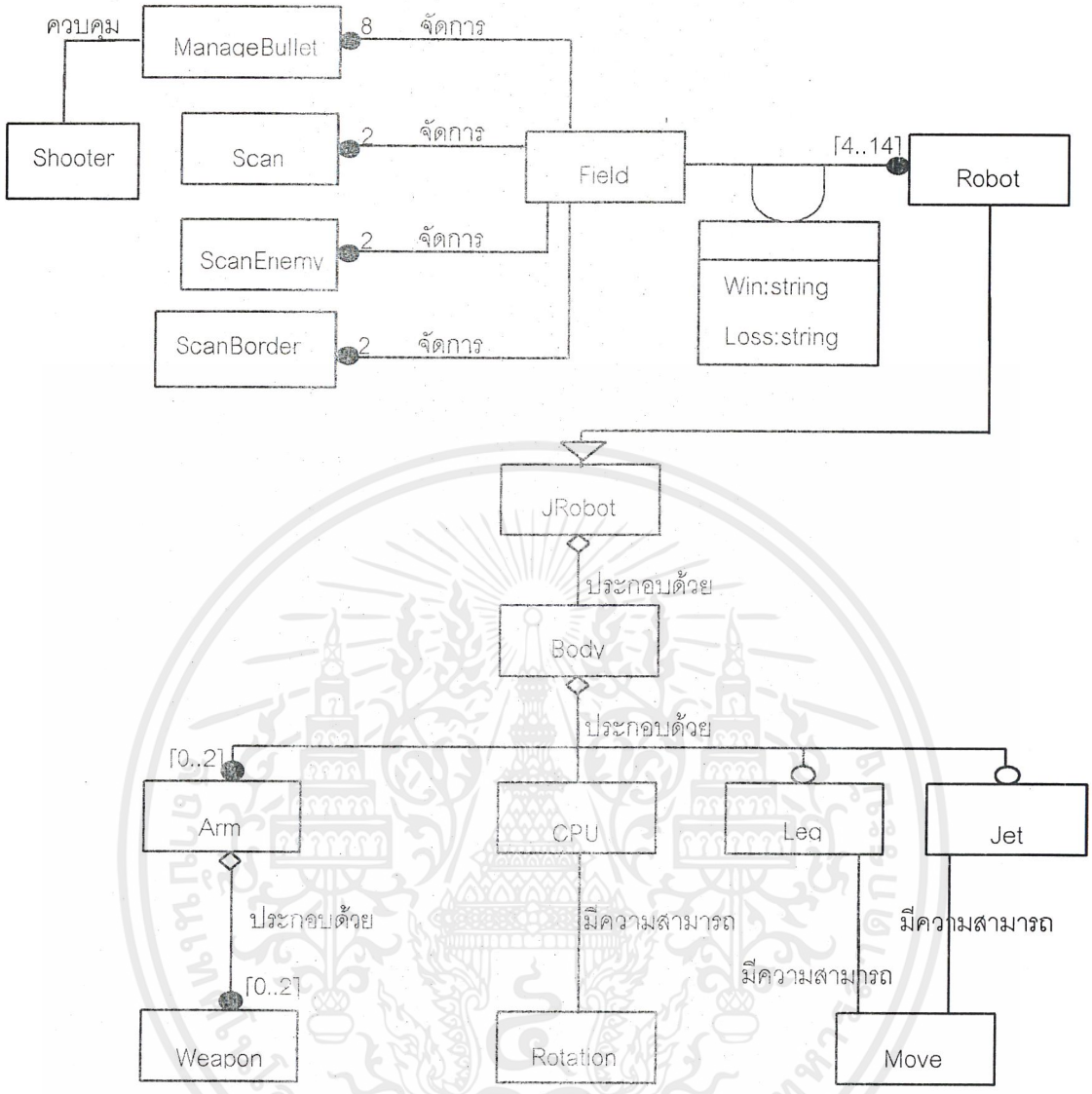
จากบทละครของหุ่นยนต์ที่ได้กล่าวไปแล้วในหัวข้อ 3.1.1 นั้นจะต้องมีการทำงานดังต่อไปนี้

ในการสร้างหุ่นยนต์นั้น ผู้สร้างจะต้องทำการสร้างหุ่นยนต์โดยการรับการถ่ายทอด (extend) มาจากคลาส Jrobot ที่เตรียมไว้ให้ซึ่งคลาสนี้จะมีการทำงานเป็น Thread เนื่องจากมีหุ่นยนต์ 2 ตัวที่ต้องทำงานไปพร้อมกัน หลังจากสร้างและอัพโหลดหุ่นยนต์และเลือกหุ่นยนต์ที่ต้องการต่อสู้ได้แล้ว จะส่งค่าหุ่นยนต์ที่เลือกมาให้สนาม สนามจะมีการสร้างหุ่นยนต์โดยการ new instance ของหุ่นยนต์แต่ละตัวขึ้น 2 instance เป็นหุ่นยนต์ตัวกลางที่เป็นตัวเก็บค่าต่าง ๆ 1 instance อีก1 instance จะถูกกำหนดให้เป็นหุ่นยนต์แทนเหตุการณ์ปกติเพื่อเริ่มทำงานจากเหตุการณ์ปกติ ฉะนั้นในตอนเริ่มต้นนี้จะมีการสร้าง instance ขึ้นทั้งหมด 4 instance คือ หุ่นยนต์ตัวที่ 1 จำนวน 2 instance และหุ่นยนต์ตัวที่ 2 อีก 2 instance นั่นเอง นอกจากนี้จะมีการสร้าง instance ของการ scan ของหุ่นยนต์ขึ้น 2 instance สร้าง instance ของการ scan คู่ต่อสู้ของหุ่นยนต์ขึ้น 2 instance สร้าง instance ของการ scan ขอบสนามของหุ่นยนต์ขึ้น 2 instance การทำงานทั้งหมดนี้จะมีการทำงานเป็น Thread เนื่องจากมีการคำนวณมากดังที่กล่าวไปแล้ว จากนั้นเมื่อเริ่มเล่นจะมีการสั่ง Thread ทั้งหมดให้ทำงาน (start Thread) นอกจากนี้จะมีคลาสการหมุน คลาสการเคลื่อนที่ของหุ่นยนต์และคลาสการเคลื่อนที่ของลูกกระสุนซึ่งมีการทำงานเป็น Thread ด้วยเหมือนกัน คลาสเหล่านี้จะทำงานต่อเมื่อมีการสั่งให้หุ่นยนต์หมุน เคลื่อนที่หรือสั่งให้ยิงนั่นเอง เมื่อมีการสั่งแล้วจะมีการสร้าง instance ของคลาสทั้งสองคลาสนี้ขึ้นและสั่งให้ทำงาน

ในการทำงานของทุก Thread นั้นจะมีการใช้ clock ที่กำหนดขึ้นเป็นตัวควบคุมการทำงานเพื่อให้การทำงานดำเนินไปพร้อมกัน โดยมีหลักคือในตอนเริ่มต้นจะให้ clock ของทุก instance นั้นมีค่ามากกว่า clock ของสนามอยู่ 1 เพื่อให้เมื่อเริ่มต้นสนามจะได้ทำงานก่อน 1 clock จากนั้นสนามจะเพิ่มค่า clock ขึ้น 1 Thread ต่าง ๆ จะสามารถทำงานได้ก็ต่อเมื่อ clock ของ Thread นั้น ๆ น้อยกว่าหรือเท่ากับ clock ของสนามดังนั้นเมื่อสนามทำงานแล้วจะทำให้ Thread อื่น ๆ ได้ทำงาน และหลังจาก Thread ต่าง ๆ ได้ทำงานแล้วจะมีการเพิ่มค่า clock ของ Thread นั้น ๆ ขึ้น 1 และจะสามารถทำงานได้อีกเมื่อ clock ของสนามเพิ่มค่าขึ้นอีก ซึ่ง clock ของสนามจะเพิ่มค่าก็ต่อเมื่อรอให้ Thread ต่าง ๆ ทำงานจนเสร็จใน clock นั้นเสียก่อน และหลังจากนั้นสนามจะเป็นตัวแสดงกราฟิกและเพิ่มค่า clock ของสนาม ทำให้ Thread อื่น ๆ ได้ทำงานต่อ นอกจากนี้ก่อนที่ clock ของสนามจะเพิ่มขึ้นนั้น สนามจะมีการตรวจสอบด้วยว่าเกิดเหตุการณ์ต่างๆ ขึ้นหรือไม่ ถ้าเกิดเหตุการณ์ใด ๆ ขึ้นแล้วจะมีการสร้าง instance ของหุ่นยนต์ตัวที่เกิดเหตุการณ์ขึ้นอีก 1 instance แทนเหตุการณ์ที่เกิดขึ้นและสั่งให้ instance นั้นเริ่มทำงาน instance ที่แทนเหตุการณ์ต่าง ๆ ที่เกิดขึ้นนี้จะทำการกำหนดค่าต่าง ๆ ที่หุ่นยนต์ตัวกลาง เนื่องจากเมื่อหุ่นยนต์ที่แทนเหตุการณ์นั้นสิ้นสุดการทำงานลงก็จะถูกทำลาย และจะถูกสร้างขึ้นใหม่อีกครั้งเมื่อเกิดเหตุการณ์นั้น ๆ ขึ้นอีกจึงต้องกำหนดค่าที่ตัวกลางแทน ในการทำงานของเหตุการณ์ต่าง ๆ ที่เกิดขึ้นนั้น สามารถมีการทำงานได้พร้อม ๆ กันโดยจะมีการอนุญาต ด้วยค่าที่ตัวกลางเช่น ใน clock หนึ่ง ๆ นั้น Thread ของเหตุการณ์ที่ได้ทำงานจะทำการ lock เพื่อไม่ให้ Thread ที่แทนเหตุการณ์อื่นสามารถเข้ามากำหนดค่าให้หุ่นยนต์ตัวกลางได้ ซึ่งการเข้ามากำหนดค่าที่หุ่นยนต์ตัวกลางนี้จะแยกเป็นการหมุนกับการเคลื่อนที่ ฉะนั้นใน clock นั้น ๆ อาจมีการเคลื่อนที่ของหุ่นยนต์ไปพร้อมกับการหมุนได้ซึ่งเป็นผลจากการสั่งจากคนละเหตุการณ์ เนื่องจากถ้าเป็นเหตุการณ์เดียวกันแล้ว จะไม่สามารถสั่งให้หุ่นยนต์เคลื่อนที่ไปพร้อมกับการหมุนได้เพราะจะมีการทำงานไปทีละคำสั่งจนจบนั่นเอง การทำงานจะดำเนินไปจนกว่าจะหมดเวลาหรือเกิดผลแพ้ชนะขึ้น จึงจบการทำงาน

จากการทำงานนี้สามารถออกแบบได้เป็นคลาสต่าง ๆ ที่เกี่ยวข้องดัง diagram ต่อไปนี้

Class Diagram ของสนามรบ



รูปที่ 3.8 แสดง Class Diagram ของสนามรบ

จาก Class diagram ข้างต้นอธิบายได้ดังนี้

- Class JRobot ประกอบด้วย Class Body , JRobot จะมี Body ได้เพียง 1 Body

- Class Body ประกอบด้วย Class Arm , Class CPU , Class Leg และ Class Jet โดย Body จะประกอบด้วย แขนอย่างมาก 2 ข้างหรือไม่มีเลยก็ได้ ,ขา 1 คู่หรือไม่มี , CPU 1 ตัว และ Jet 1 ตัวหรือไม่มี

- Class Arm ประกอบด้วย Class weapon โดยที่ แขน 1 ข้าง สามารถมี weapon ได้อย่างมาก 2 ชิ้น

- Class Leg และ Class Jet มีความสัมพันธ์กับ Class Move ซึ่งจะทำให้หุ่นยนต์มีความสามารถในการเคลื่อนที่ได้ โดยการเคลื่อนที่โดย Class Leg กับ Class Jet จะมีความเร็วต่างกัน

- Class CPU มีความสัมพันธ์กับ Class Rotation ซึ่งจะทำให้หุ่นยนต์สามารถหมุนได้

- Class Robot inherit มาจาก Class Jrobot ซึ่งเป็น Abstract Class ที่ให้ ผู้เล่น extend

- Class Robot มีความสัมพันธ์กับ Class Field หรือสนามรบโดยใน 1 Field จะมี Robot ที่ต่อสู้กันได้เพียง 2 ตัว และมีการสร้าง object แทนเหตุการณ์ปกติไว้ตั้งแต่เริ่มการต่อสู้ และหากเกิดเหตุการณ์ต่าง ๆ ขึ้นก็จะมีการสร้าง object ขึ้นอีกจึงต้องมีอย่างน้อย 4 ตัวและอย่างมาก 14 ตัว

- Class Scan ทำหน้าที่จัดการคำนวณตำแหน่งของจุด scan ของหุ่นยนต์ เมื่อมีการ scan

- Class ScanEnemy ทำหน้าที่จัดการคำนวณเกี่ยวกับการค้นหาศัตรู

- Class ScanBorder ทำหน้าที่จัดการคำนวณตำแหน่งเกี่ยวกับการค้นหาขอบเขตของสนาม

-Class ManageBullet ทำหน้าที่จัดการเกี่ยวกับการยิงของหุ่นยนต์ประกอบด้วยอาวุธใกล้ของแขนซ้าย อาวุธไกลของแขนซ้าย อาวุธใกล้ของแขนขวา และอาวุธไกลของแขนขวาของหุ่นยนต์ทั้งสองตัวรวมเป็นทั้งหมด 8 Instances

- Class Scan , Class ScanEnemy และ Class ScanBorder มีความสัมพันธ์กับ Class Field โดยใน Class Field จะมีการกำหนดไว้ Class ละ 2 Instances เป็นของหุ่นยนต์แต่ละตัว

- Class Robot สัมพันธ์กับคลาส Field ได้ Link attribute คือ ผลแพ้ชนะ ในสนาม

- รายละเอียดของ attribute และ method โดยละเอียดมีดังนี้

BodyHead
Weight:int //น้ำหนักของส่วน body Size:int //ขนาด body Energy:int //พลังงาน body

BodyHead(weight:int,size:int,energy:int) //Constructor
getWeight():int // คืนค่าน้ำหนักของ body
getSize():int // คืนค่าขนาดของ body
getEnergy():int // คืนค่าพลังงานของ body
setEnergy(energy:int) // กำหนดค่าพลังงานให้ body

รูปที่ 3.9 แสดง Attribute และ Method ของ Class BodyHead ใน Class Diagram ของสนามรบ

Arm
Weight:int //น้ำหนักของส่วนแขน Size:int //ขนาดส่วนแขน Energy:int //พลังงานของส่วนแขน

Arm(weight:int,size:int,energy:int) //Consrtuctor
getWeight():int // คืนค่าน้ำหนักของแขน
getSize():int // คืนค่าขนาดของแขน
getEnergy():int // คืนค่าพลังงานของแขน
setEnergy(energy:int) // กำหนดค่าพลังงานให้แขน

รูปที่ 3.10 แสดง Attribute และ Method ของ Class Arm ใน Class Diagram ของสนามรบ

Leg
Weight:int //น้ำหนักของส่วนขา Size:int //ขนาดของขา Energy:int //พลังงานของขา Speed:int //ความเร็วของขาที่สามารถเคลื่อนที่ได้

Leg(weight:int,size:int,energy:int,speed:int) //Consrtuctor
getWeight():int // คืนค่าน้ำหนักของขา
getSize():int // คืนค่าขนาดของขา
getEnergy():int // คืนค่าพลังงานของขา
getSpeed():int // คืนค่าความเร็วของขา
setEnergy(energy:int) // กำหนดค่าพลังงานของขา
move(cpu:CPU,degree:int,direction:Boolean,distance:int) // สั่งให้เคลื่อนที่โดยใช้ขา

รูปที่ 3.11 แสดง Attribute และ Method ของ Class Leg ใน Class Diagram ของสนามรบ

CPU
ScanDistance:int // ระยะความยาวของพื้นที่ Scan ซึ่งเป็นรูปสามเหลี่ยม
ScanDegree:int // ความกว้างเป็นองศาของมุม Scan ซึ่งเป็นรูปสามเหลี่ยมโดยยึดจากตัวหุ่นยนต์เป็นหลัก
ScanSpeed:int // ความเร็วในการหมุนของหุ่นยนต์ในการ Scan
NearAbility:int // เปอร์เซ็นต์ความสามารถในการยิงระยะใกล้
FarAbility:int // เปอร์เซ็นต์ความสามารถในการยิงระยะไกล
FaceDegree:int // ทิศทางองศาปัจจุบันของหน้าหุ่นยนต์
CurrentX:int // ตำแหน่งปัจจุบันของหุ่นยนต์ที่จุด X
CurrentY:int // ตำแหน่งปัจจุบันของหุ่นยนต์ที่จุด X
MyRobot:int // ค่าที่บอกว่าเป็นหุ่นยนต์ตัวที่หนึ่งหรือตัวที่สอง
BulletDegree:int // ทิศทางองศาของตำแหน่งที่หุ่นยนต์โดนยิงโดยเทียบกับทิศของหุ่นยนต์
ScanXmiddle:int // จุดกึ่งกลางของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
ScanYmiddle:int // จุดกึ่งกลางของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
ScanXleft:int // จุดทางซ้ายของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
ScanYleft:int // จุดทางซ้ายของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
ScanXright:int // จุดทางขวาของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
ScanYright:int // จุดทางขวาของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
CanSetCurrent:boolean // ค่าที่บอกว่าในขณะนั้นสามารถกำหนดค่าตำแหน่งปัจจุบันของหุ่นยนต์ได้หรือไม่
CanSetFace:boolean // ค่าที่บอกว่าในขณะนั้นสามารถกำหนดค่าทิศทางองศาปัจจุบันของหุ่นยนต์ได้หรือไม่
canShoot:boolean // ค่าที่บอกว่าในขณะนั้นสามารถยิงกระสุนได้หรือไม่
CommonStop:boolean // ค่าที่บอกว่าในขณะนั้นการทำงานในเหตุการณ์ปกติของหุ่นยนต์หยุดอยู่หรือไม่
FoundEnemyStop:boolean // ค่าที่บอกว่าในขณะนั้นเหตุการณ์พบศัตรูของหุ่นยนต์หยุดอยู่หรือไม่
AttackedStop:boolean // ค่าที่บอกว่าในขณะนั้นเหตุการณ์ถูกยิงของหุ่นยนต์หยุดอยู่หรือไม่
FoundBorderStop:boolean // ค่าที่บอกว่าในขณะนั้นเหตุการณ์พบขอบสนามของหุ่นยนต์หยุดอยู่หรือไม่
CrashEnemyStop:boolean // ค่าที่บอกว่าในขณะนั้นเหตุการณ์ชนศัตรูของหุ่นยนต์หยุดอยู่หรือไม่
CrashBorderStop:boolean // ค่าที่บอกว่าในขณะนั้นเหตุการณ์ชนของสนามของหุ่นยนต์หยุดอยู่หรือไม่
CPU(scandistance:int,scandegre:int,scanspeed:int,nearability:int,farability:int,facedegree:int)
// Constructor โดยมีการกำหนดค่าต่างๆด้วย
Rotate(degree:int,direction:boolean) // สั่งให้หุ่นยนต์หมุนโดยระบุองศาและทิศทางที่ต้องการ
CanSetCurrentXY():boolean // คืนค่าว่าในขณะนั้นสามารถกำหนดค่าตำแหน่งของหุ่นยนต์ได้หรือไม่
CanSetFaceDegree():boolean // คืนค่าว่าในขณะนั้นสามารถกำหนดค่าองศาของหุ่นยนต์ได้หรือไม่
getCurrentX():int // คืนค่าตำแหน่งปัจจุบันของหุ่นยนต์ของจุด X
getCurrentY():int // คืนค่าตำแหน่งปัจจุบันของหุ่นยนต์ของจุด Y
setCurrentX(curX:int) // กำหนดค่าตำแหน่งปัจจุบันของหุ่นยนต์ของจุด X
setCurrentY(curY:int) // กำหนดค่าตำแหน่งปัจจุบันของหุ่นยนต์ของจุด Y
setCanSetCurrent(can:boolean) // กำหนดค่าว่าในขณะนั้นสามารถกำหนดค่าองศาของหุ่นยนต์ได้หรือไม่

รูปที่ 3.12 แสดง Attribute และ Method ของ Class CPU ใน Class Diagram ของสนามรบ

CPU
CommonStop() // สั่งให้การทำงานในเหตุการณ์ปกติหยุด
FoundEnemyStop() // สั่งให้การทำงานในเหตุการณ์พบศัตรูหยุด
AttackedStop() // สั่งให้การทำงานในเหตุการณ์พบศัตรูหยุด
FoundBorderStop() // สั่งให้การทำงานในเหตุการณ์พบขอบสนามหยุด
CrashEnemyStop() // สั่งให้การทำงานในเหตุการณ์ชนศัตรูหยุด
CrashBorderStop() // สั่งให้การทำงานในเหตุการณ์ชนขอบสนามหยุด
CommonStart() // สั่งให้การทำงานในเหตุการณ์ปกติทำงาน
FoundEnemyStart() // สั่งให้การทำงานในเหตุการณ์พบศัตรูทำงาน
AttackedStart() // สั่งให้การทำงานในเหตุการณ์โดนยิงทำงาน
FoundBorderStart() // สั่งให้การทำงานในเหตุการณ์พบขอบสนามทำงาน
CrashEnemyStart() // สั่งให้การทำงานในเหตุการณ์ชนศัตรูทำงาน
CrashBorderStart() // สั่งให้การทำงานในเหตุการณ์ชนขอบสนามทำงาน
getScanDistance():int // คืนค่าระยะความยาวของพื้นที่ Scan ซึ่งเป็นรูปสามเหลี่ยม
getScanDegree():int // คืนค่าความกว้างเป็นองศาของมุม Scan ซึ่งเป็นรูปสามเหลี่ยมโดยยึดจากตัวหุ่นยนต์เป็นหลัก
getScanSpeed():int // คืนค่าความเร็วในการหมุนของหุ่นยนต์ในการ Scan
getNearAbility():int // คืนค่าเปอร์เซ็นต์ความสามารถในการยิงระยะใกล้
getFarAbility():int // คืนค่าเปอร์เซ็นต์ความสามารถในการยิงระยะไกล
getFaceDegree():int // คืนค่าทิศทางองศาปัจจุบันของหน้าหุ่นยนต์
getScanXmiddle():int // คืนค่าจุดกึ่งกลางของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
getScanYmiddle():int // คืนค่าจุดกึ่งกลางของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
getScanXleft():int // คืนค่าจุดทางซ้ายของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
getScanYleft():int // คืนค่าจุดทางซ้ายของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
getScanXright():int // คืนค่าจุดทางขวาของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
getScanYright():int // คืนค่าจุดทางขวาของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
setFaceDegree(facedegree:int) // กำหนดทิศทางองศาปัจจุบันของหน้าหุ่นยนต์
getEnemyDegree(boolean right):int // คืนค่าองศาของตำแหน่งศัตรูหากสามารถ Scan ศัตรูพบ
getEnemyDistance():int // คืนค่าระยะห่างของศัตรูหากสามารถ Scan ศัตรูพบ
getBulletDegree(boolean Right):int // คืนค่าองศาของตำแหน่งที่หุ่นยนต์โดนยิงโดยเทียบกับทิศของหุ่นยนต์

รูปที่ 3.12 (ต่อ) แสดง Attribute และ Method ของ Class CPU ใน Class Diagram ของสนามรบ

Jet
Speed:int // ความเร็วของไอพ่นที่สามารถเคลื่อนที่ได้ Weight:int // น้ำหนักของไอพ่น Oil:int // น้ำมันของไอพ่นที่มีอยู่

Jet(field:Field,speed:int,weight:int,oil:int) // Constructor
Move(degree:int,direction:Boolean,distance:int) // สั่งให้เคลื่อนที่โดยใช้ไอพ่นโดยระบุองศาและทิศทางของการเคลื่อนที่
getOil():int // คืนค่าปริมาณน้ำมันของไอพ่น
getSpeed():int // คืนค่าความเร็วของไอพ่น
getWeight():int // คืนค่าน้ำหนักของไอพ่น

รูปที่ 3.13 แสดง Attribute และ Method ของ Class Jet ใน Class Diagram ของสนามรบ

Weapon
DestroyPower:int // พลังทำลายของอาวุธ Distance:int //ระยะทางการยิงได้ของอาวุธ BulletSpeed:int // ความเร็วของกระสุนของอาวุธ BulletAmount:int // จำนวนลูกกระสุนเริ่มต้น CurrentBulletAmount:int // จำนวนลูกกระสุนที่มีอยู่ในขณะนั้น ShootingRate:int // อัตราการยิงลูกกระสุน Weight:int // น้ำหนักของอาวุธ

Weapon(name:String,destroypower:int,distance:int,bulletspeed:int,bulletamount:int,shootingrate:int,weight:int) // Consrtuctor
Shoot(cpu:CPU;degree:int,direction:Boolean) // ยิงโดยระบุองศาและทิศทางของการยิง
getDestroyPower():int // คืนค่าพลังทำลายของอาวุธ
getDistance():int // คืนค่าระยะทางการยิงของอาวุธ
getBulletSpeed():int // คืนค่าความเร็วของกระสุน
getBulletAmount():int // คืนค่าจำนวนกระสุนเริ่มต้น
getCurrentBulletAmount():int // คืนค่าจำนวนกระสุนในขณะนั้น
getShootingRate():int // คืนค่าอัตราการยิงของอาวุธ
getWeight():int // คืนค่าน้ำหนักของอาวุธ

รูปที่ 3.14 แสดง Attribute และ Method ของ Class Weapon ใน Class Diagram ของสนามรบ

Move	
DestinationX:int	// ตำแหน่งเป้าหมายในการเคลื่อนที่ของจุด X
DestinationY:int	// ตำแหน่งเป้าหมายในการเคลื่อนที่ของจุด Y
Distance:int	// ระยะทางในการเคลื่อนที่
Speed:int	// ความเร็วในการเคลื่อนที่
Degree:int	// องศาในการเคลื่อนที่
Direction:Boolean	// ทิศทางในการเคลื่อนที่
FaceDegree:int	// องศาปัจจุบันของหน้าหุ่นยนต์
LegOrJet:Boolean	// เป็นการเคลื่อนที่ของขาหรือไอพ่น
move(LegOrJet:boolean,cpu:CPU,degree:int,direction:boolean,distance:int,speed:int) // สั่งให้เคลื่อนที่โดยบอกว่าเป็นการเคลื่อนที่ของขาหรือไอพ่นพร้อมทั้งบอกองศาทิศทาง ระยะทาง และความเร็วในการเคลื่อนที่	
FindDestination()	// คำนวณหาเป้าหมายที่ต้องเคลื่อนที่ไป
run()	// สำหรับเริ่มต้นการทำงานของ thread

รูปที่ 3.15 แสดง Attribute และ Method ของ Class Move ใน Class Diagram ของสนามรบ

Shooter	
BeginBulletX	// ตำแหน่งเริ่มต้นของกระสุนของจุด X
BeginBulletY	// ตำแหน่งเริ่มต้นของกระสุนของจุด Y
CurrentBulletX:int	// ตำแหน่งปัจจุบันของกระสุนของจุด X
CurrentBulletY:int	// ตำแหน่งปัจจุบันของกระสุนของจุด Y
DestinationX:int	// ตำแหน่งเป้าหมายของกระสุนของจุด X
DestinationY:int	// ตำแหน่งเป้าหมายของกระสุนของจุด Y
Degree:int	// องศาของการยิงกระสุน
Distance:int	// ระยะทางของการยิงกระสุน
BulletSpeed:int	// ความเร็วของกระสุน
ShootingRate:int	// อัตราการยิงกระสุน
Shooter(cpu:CPU,right:boolean,distance:int,bulletspeed:int,shootingrate:int) // Constructor	
getBeginBulletX():int	// คืนค่าของตำแหน่งเริ่มต้นของกระสุนที่จุด X
getBeginBulletY():int	// คืนค่าของตำแหน่งเริ่มต้นของกระสุนที่จุด Y
getCurrentBulletX():int	// คืนค่าของตำแหน่งปัจจุบันของกระสุนที่จุด X
getCurrentBulletY():int	// คืนค่าของตำแหน่งปัจจุบันของกระสุนที่จุด Y
run()	// สำหรับเริ่มต้นการทำงานของ thread

รูปที่ 3.16 แสดง Attribute และ Method ของ Class Shooter ใน Class Diagram ของสนามรบ

Field	
MainRobot1:JRobot	// หุ่นยนต์หลักที่ใช้ต่อสู้ตัวแรก
MainRobot2:JRobot	// หุ่นยนต์หลักที่ใช้ต่อสู้ตัวที่สอง
R1Event[]:JRobot	// หุ่นยนต์ที่แทนเหตุการณ์ต่างๆของตัวแรก
R2Event[]:JRobot	// หุ่นยนต์ที่แทนเหตุการณ์ต่างๆของตัวที่สอง
Robot1Scan:Scan	// ตัวที่จัดการเกี่ยวกับการ Scan ของหุ่นยนต์ตัวแรก
Robot2Scan:Scan	// ตัวที่จัดการเกี่ยวกับการ Scan ของหุ่นยนต์ตัวที่สอง
R1scanEnemy:ScanEnemy	// ตัวที่จัดการเกี่ยวกับการ Scan คู่ต่อสู้ของหุ่นยนต์ตัวแรก
R2scanEnemy:ScanEnemy	// ตัวที่จัดการเกี่ยวกับการ Scan คู่ต่อสู้ของหุ่นยนต์ตัวที่สอง
R1scanBorder:ScanBorder	// ตัวที่จัดการเกี่ยวกับการ Scan ขอบสนามของหุ่นยนต์ตัวแรก
R2scanBorder:ScanBorder	// ตัวที่จัดการเกี่ยวกับการ Scan ขอบสนามของหุ่นยนต์ตัวที่สอง
R1ManageBullet:ManageBullet	// ตัวที่จัดการเกี่ยวกับการยิงของหุ่นยนต์ตัวแรก
R2ManageBullet:ManageBullet	// ตัวที่จัดการเกี่ยวกับการยิงของหุ่นยนต์ตัวที่สอง
init()	// จัดการทำงานเกี่ยวกับตอนเริ่มต้นของ Applet
start()	// จัดการทำงานเกี่ยวกับตอน Start ของ Applet
paint(g:Graphics)	// จัดการทำงานเกี่ยวกับการแสดงผลของ Applet
update(g:Graphics)	// จัดการทำงานเมื่อต้องการแสดง Graphic ของ Applet
stop()	// จัดการเมื่อหยุดการทำงานของ Applet

รูปที่ 3.17 แสดง Attribute และ Method ของ Class Field ใน Class Diagram ของสนามรบ

ManageBullet	
LeftShortBullet[]:Shooter	// จัดการเกี่ยวกับกระสุนของอาวุธระยะสั้นทางแขนซ้ายของหุ่นยนต์
RightShortBullet[]:Shooter	// จัดการเกี่ยวกับกระสุนของอาวุธระยะสั้นทางแขนขวาของหุ่นยนต์
LeftLongBullet[]:Shooter	// จัดการเกี่ยวกับกระสุนของอาวุธระยะไกลทางแขนซ้ายของหุ่นยนต์
RightLongBullet[]:Shooter	// จัดการเกี่ยวกับกระสุนของอาวุธระยะไกลทางแขนขวาของหุ่นยนต์
getLeftShortBulletObject(int i):Shooter	// คืนค่า Object กระสุนของอาวุธระยะสั้นทางแขนซ้ายของหุ่นยนต์
getRightShortBulletObject(int i):Shooter	// คืนค่า Object กระสุนของอาวุธระยะสั้นทางแขนขวาของหุ่นยนต์
getLeftLongBulletObject(int i):Shooter	// คืนค่า Object กระสุนของอาวุธระยะไกลทางแขนซ้ายของหุ่นยนต์
getRightLongBulletObject(int i):Shooter	// คืนค่า Object กระสุนของอาวุธระยะไกลทางแขนขวาของหุ่นยนต์
Shoot(cpu:CPU)	// ยิงโดยดูว่าเป็นการยิงของอาวุธใดและมีการส่งระยะทาง ความเร็ว และอัตราการยิงของอาวุธ

รูปที่ 3.18 แสดง Attribute และ Method ของ Class ManageBullet ใน Class Diagram ของสนามรบ

Scan	
Xmiddle:int	// จุดกึ่งกลางของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
Ymiddle:int	// จุดกึ่งกลางของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
Xleft:int	// จุดทางซ้ายของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
Yleft:int	// จุดทางซ้ายของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
Xright:int	// จุดทางขวาของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
Yright:int	// จุดทางขวาของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
ScanDistance:int	// ระยะความยาวของพื้นที่ Scan ซึ่งเป็นรูปสามเหลี่ยม
ScanDegree:int	// ความกว้างเป็นองศาของมุม Scan ซึ่งเป็นรูปสามเหลี่ยมโดยยึดจากตัวหุ่นยนต์เป็นหลัก
SideScanDistance:int	// ระยะความยาวของพื้นที่ Scan ทางด้านข้างซึ่งเป็นรูปสามเหลี่ยม
Scan(cpu:CPU)	// Constructor
getXmiddle():int	// คืนค่าจุดกึ่งกลางของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
getYmiddle():int	// คืนค่าจุดกึ่งกลางของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
getXleft():int	// คืนค่าจุดทางซ้ายของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
getYleft():int	// คืนค่าจุดทางซ้ายของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
getXright():int	// คืนค่าจุดทางขวาของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด X
getYright():int	// คืนค่าจุดทางขวาของขอบเขตการ Scan ซึ่งเป็นรูปสามเหลี่ยมของจุด Y
FindScanPoint(degreetype:int,degresize:int)	// คำนวณหาจุดขอบเขตของพื้นที่ Scan
run()	// สำหรับเริ่มต้นการทำงานของ thread

รูปที่ 3.19 แสดง Attribute และ Method ของ Class Scan ใน Class Diagram ของสนามรบ

ScanEnemy	
MyXmiddle:int	// จุดกึ่งกลางของพื้นที่ Scan ของหุ่นยนต์ที่จุด X
MyYmiddle:int	// จุดกึ่งกลางของพื้นที่ Scan ของหุ่นยนต์ที่จุด Y
MyXleft:int	// จุดทางซ้ายของพื้นที่ Scan ของหุ่นยนต์ที่จุด X
MyYleft:int	// จุดทางซ้ายของพื้นที่ Scan ของหุ่นยนต์ที่จุด Y
MyXright:int	// จุดทางขวาของพื้นที่ Scan ของหุ่นยนต์ที่จุด X
MyYright:int	// จุดทางขวาของพื้นที่ Scan ของหุ่นยนต์ที่จุด Y
EnemyXcenter:int	// ตำแหน่งกึ่งกลางของหุ่นยนต์ศัตรูที่จุด X
EnemyYcenter:int	// ตำแหน่งกึ่งกลางของหุ่นยนต์ศัตรูที่จุด Y
FoundEnemy:boolean	// ค่าที่ตรวจสอบว่า Scan พบศัตรูหรือไม่
ScanEnemy(myRobot:int)	// Constructor โดยบอกว่าเป็นของหุ่นยนต์ตัวใด
getFoundEnemy():int	// คืนค่าว่า Scan พบศัตรูหรือไม่
setFoundEnemy()	// กำหนดค่าว่า Scan พบศัตรูหรือไม่

รูปที่ 3.20 แสดง Attribute และ Method ของ Class ScanEnemy ใน Class Diagram ของสนามรบ

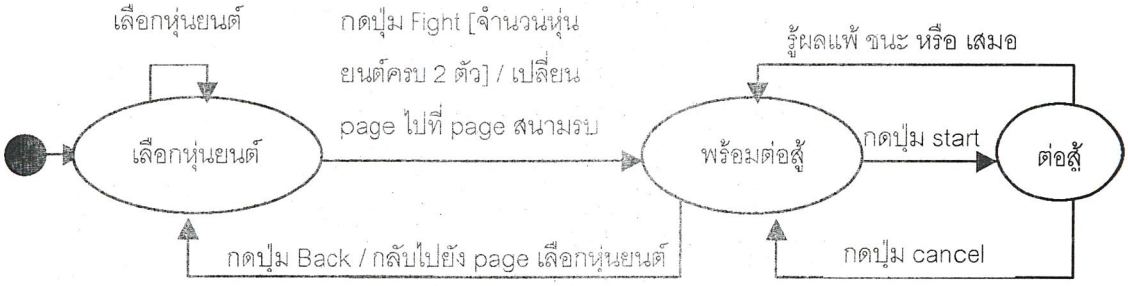
ScanBorder	
CenterpointX:int	// จุดกึ่งกลางของพื้นที่ Scan ของหุ่นยนต์ที่จุด X
centerpointY:int	// จุดกึ่งกลางของพื้นที่ Scan ของหุ่นยนต์ที่จุด Y
leftpointX:int	// จุดทางซ้ายของพื้นที่ Scan ของหุ่นยนต์ที่จุด X
leftpointY:int	// จุดทางซ้ายของพื้นที่ Scan ของหุ่นยนต์ที่จุด Y
rightpointX:int	// จุดทางขวาของพื้นที่ Scan ของหุ่นยนต์ที่จุด X
rightpointY:int	// จุดทางขวาของพื้นที่ Scan ของหุ่นยนต์ที่จุด Y
Direction:int	// ทิศทางที่พบขอบสนามโดยเทียบกับตัวหุ่นยนต์
BorDer:int	// ขอบสนามที่ Scan พบว่าเป็นขอบสนามด้านไหน
Distance:int	// ระยะห่างจากขอบสนามที่พบ
FoundBorder:boolean	// ค่าที่ตรวจสอบว่า Scan พบขอบของสนามหรือไม่
ScanBorder(myRobot:int)	// Constructor โดยบอกว่าเป็นของหุ่นยนต์ตัวใด
getDirection():int	// คืนค่าทิศทางที่พบขอบสนามโดยเทียบกับตัวหุ่นยนต์
getBorder():int	// คืนค่าขอบสนามที่ Scan พบว่าเป็นขอบสนามด้านไหน
getFoundBorder():boolean	// คืนค่าที่ตรวจสอบว่า Scan พบขอบของสนามหรือไม่

รูปที่ 3.21 แสดง Attribute และ Method ของ Class ScanBorder ใน Class Diagram ของสนามรบ

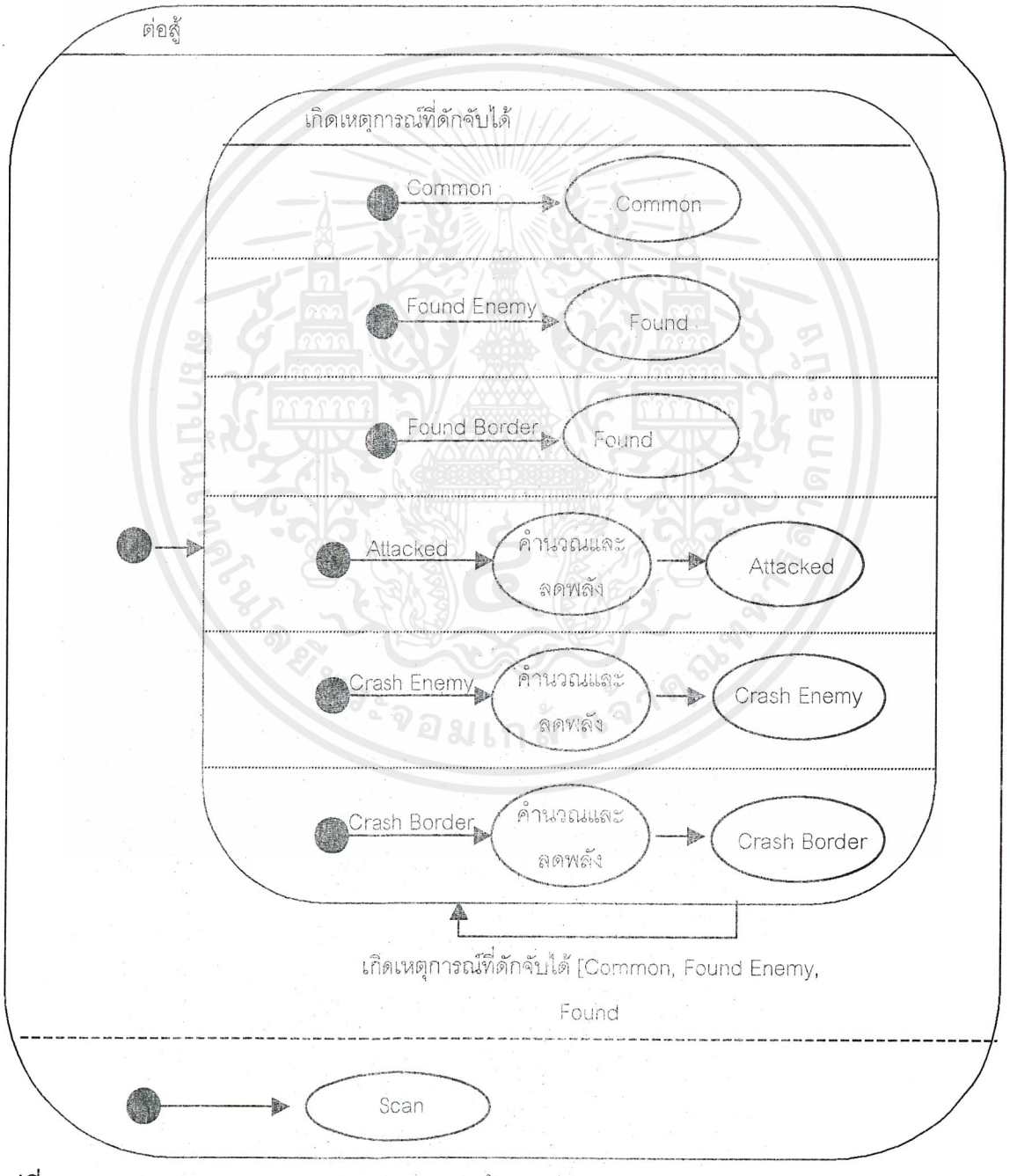
JRobot	
MyEvent:int	// เป็นค่าที่บอกว่าเป็นเหตุการณ์อะไร
Common()	// เหตุการณ์ปกติ
FoundEnemy()	// เหตุการณ์พบศัตรู
Attacked()	// เหตุการณ์ถูกยิง
FoundBorder()	// เหตุการณ์พบขอบสนาม
CrashEnemy()	// เหตุการณ์ชนศัตรู
CrashBorder()	// เหตุการณ์ชนขอบสนาม
run()	// สำหรับเริ่มต้นการทำงานของ thread
getCpuObject():CPU	// คืนค่า object cpu
getBodyHeadObject():BodyHead	// คืนค่า object ลำตัว
getLeftArmObject():Arm	// คืนค่า object แขนซ้าย
getRightArmObject():Arm	// คืนค่า object แขนขวา
getLegObject():Leg	// คืนค่า object ขา
getJetObject():Jet	// คืนค่า object ไอพ่น
getLeftShortGunObject():Weapon	// คืนค่า object อาวุธปืนระยะใกล้ทางซ้าย
getLeftLongGunObject():Weapon	// คืนค่า object อาวุธปืนระยะไกลทางซ้าย
getRightShortGunObject():Weapon	// คืนค่า object อาวุธปืนระยะใกล้ทางขวา
getRightLongGunObject():Weapon	// คืนค่า object อาวุธปืนระยะไกลทางขวา

รูปที่ 3.22 แสดง Attribute และ Method ของ Class JRobot ใน Class Diagram ของสนามรบ

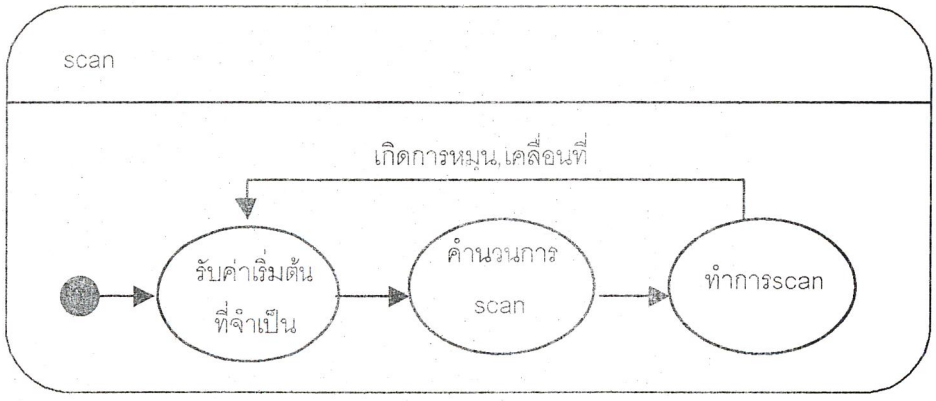
State Diagram ของสนามรบ



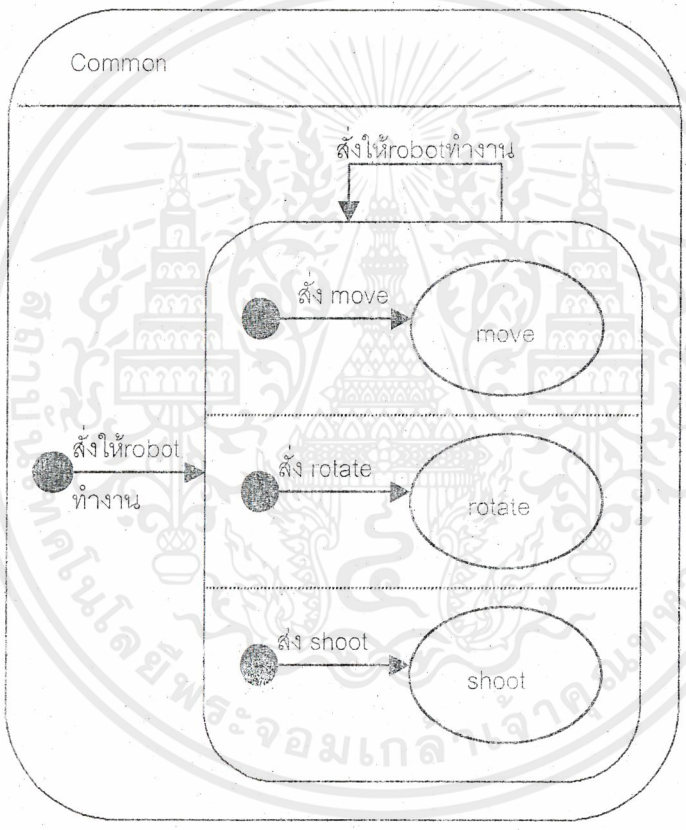
รูปที่ 3.23 แสดง State Diagram ของสนามรบ



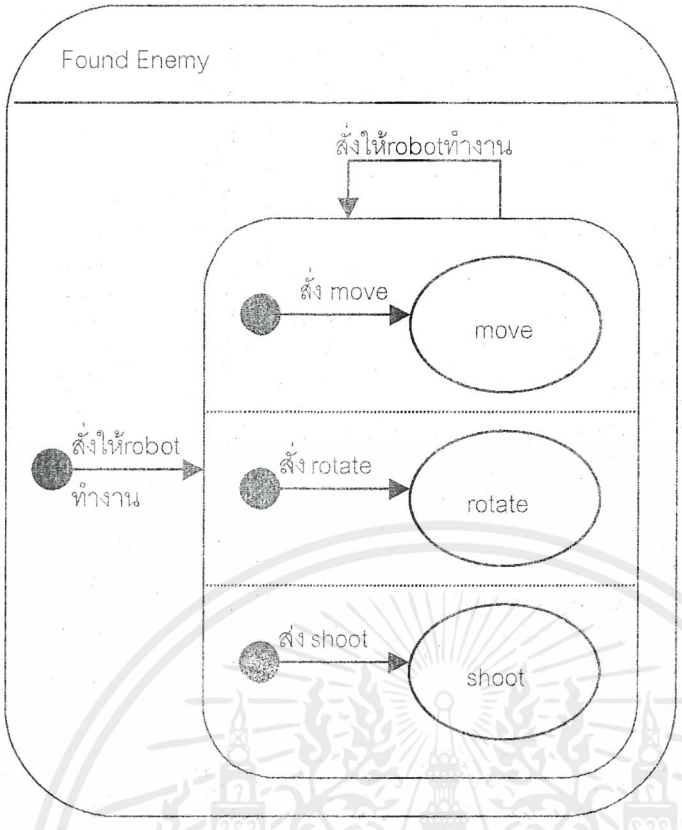
รูปที่ 3.24 แสดง State Diagram ย่อยของการต่อสู้



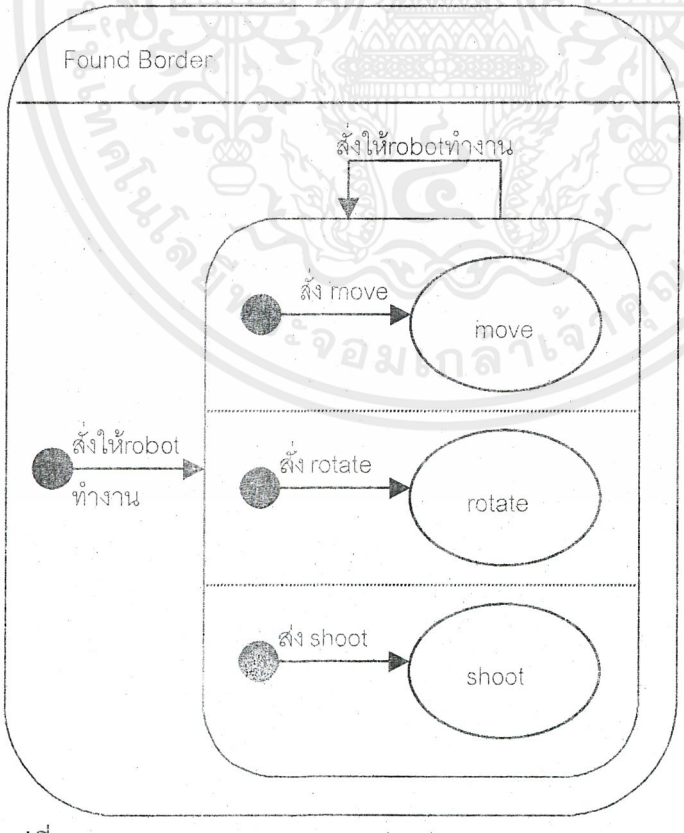
รูปที่ 3.25 แสดง State Diagram ย่อยของการ scan



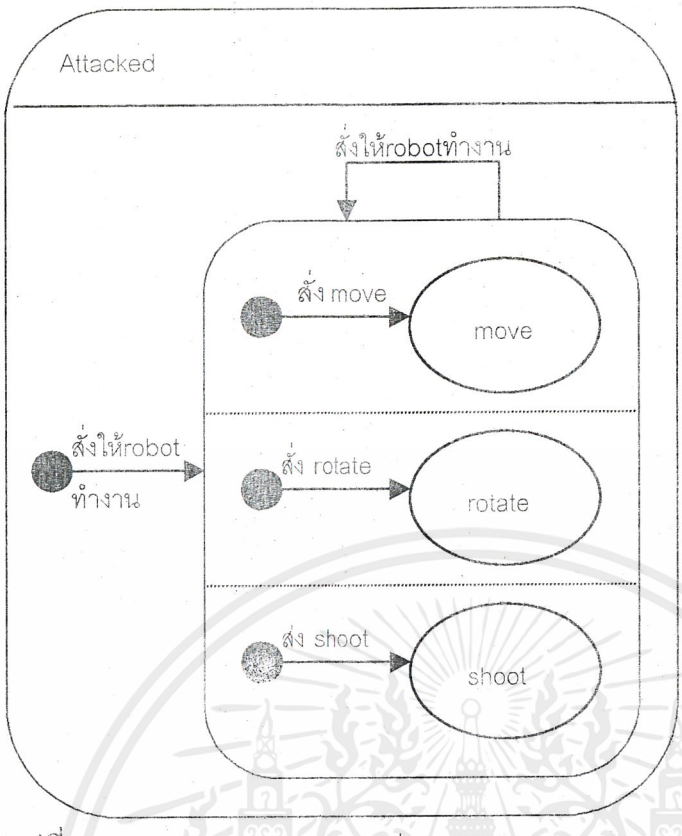
รูปที่ 3.26 แสดง State Diagram ย่อยของ process Common



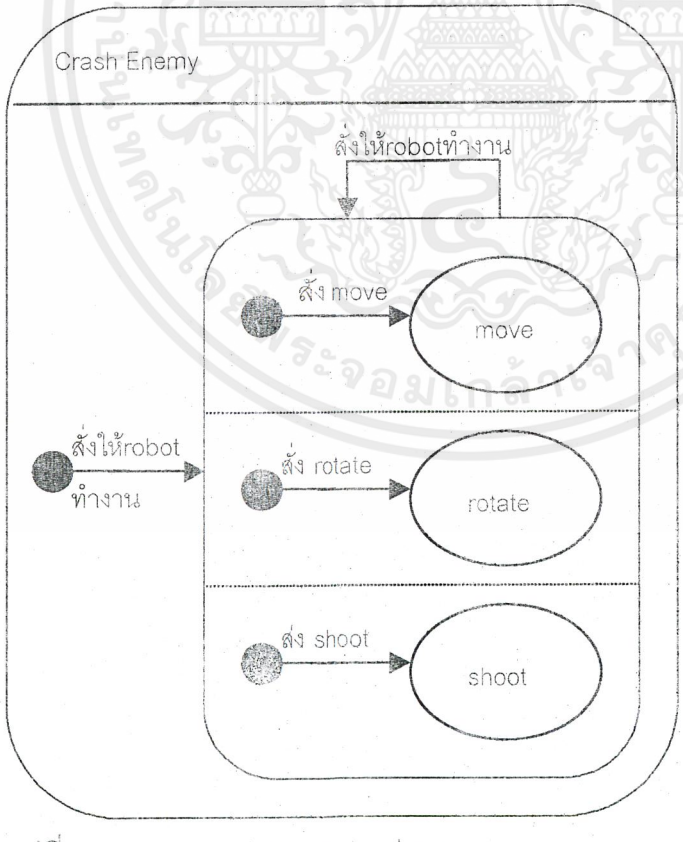
รูปที่ 3.27 แสดง State Diagram ย่อยของ process Found Enemy



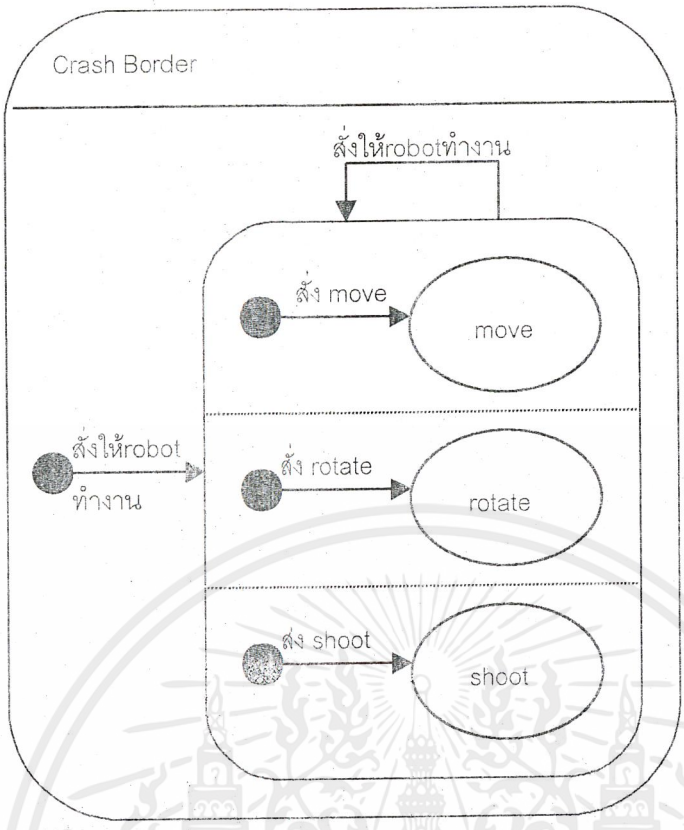
รูปที่ 3.28 แสดง State Diagram ย่อยของ process Found Border



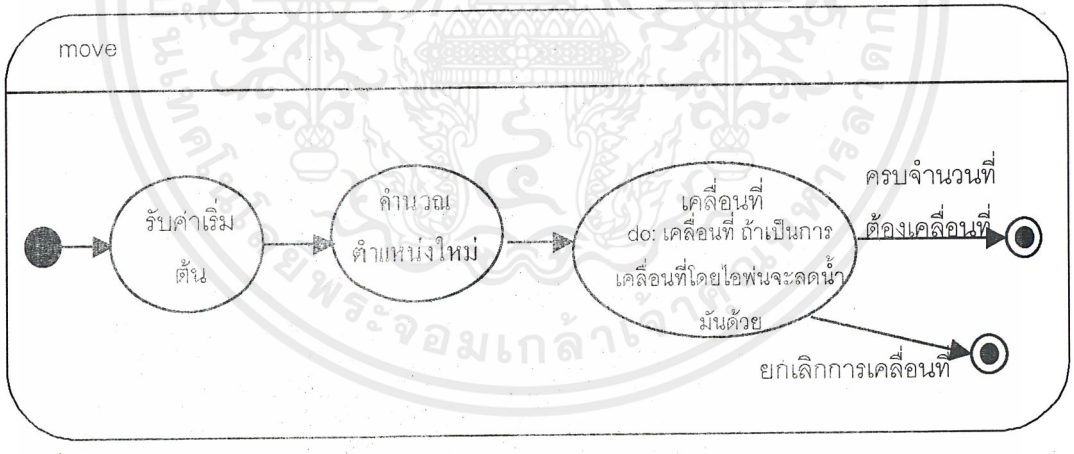
รูปที่ 3.29 แสดง State Diagram ย่อยของ process Attacked



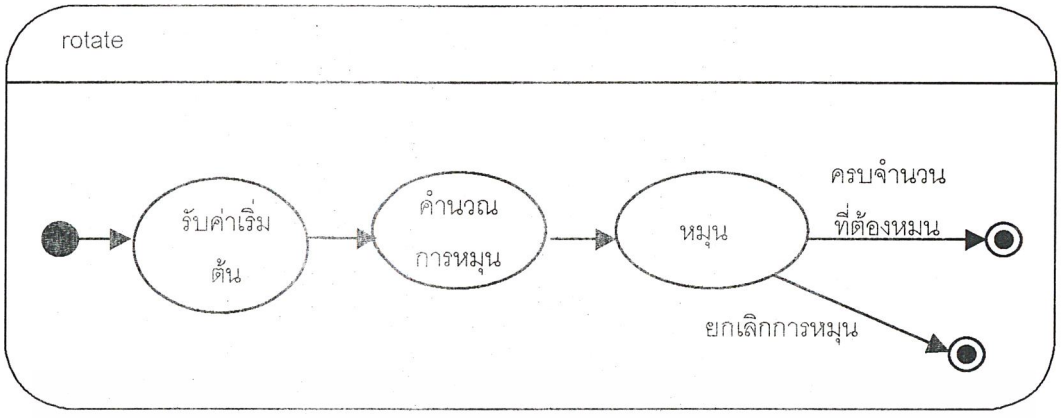
รูปที่ 3.30 แสดง State Diagram ย่อยของ process Crash Enemy



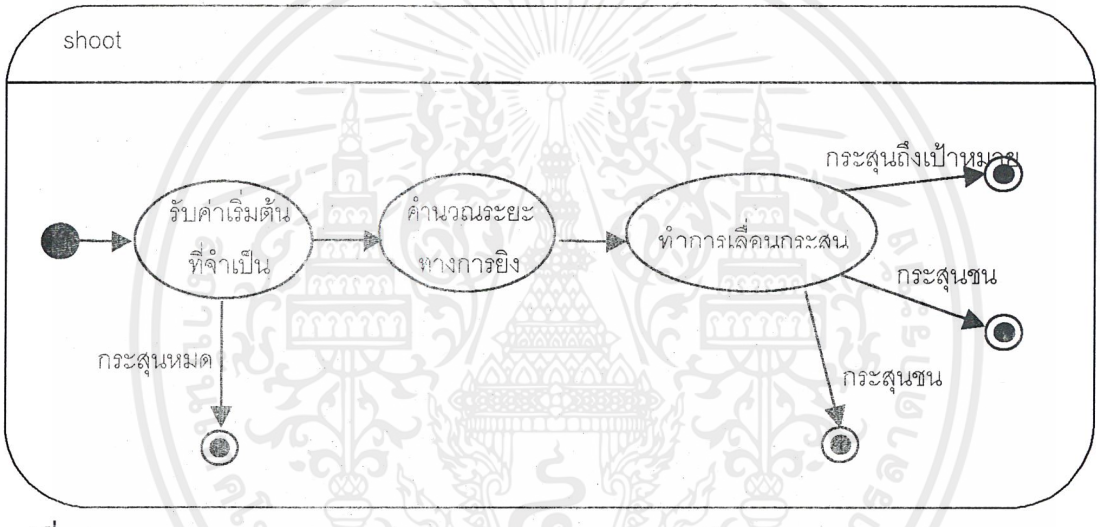
รูปที่ 3.31 แสดง State Diagram ย่อยของ process Crash Border



รูปที่ 3.32 แสดง State Diagram ย่อยของการ move



รูปที่ 3.33 แสดง State Diagram ย่อยของการ rotate



รูปที่ 3.34 แสดง State Diagram ย่อยของการ shoot

จากบทละครของเกมในหัวข้อ 3.1.1 นั้น สิ่งที่สนามรบจะต้องจัดการคือ

1. การเคลื่อนที่ของหุ่นยนต์ จะมีการจัดการอยู่ในคลาส Move ซึ่งจะมีจัดการการเคลื่อนที่ของหุ่นยนต์ทั้งโดยการใช้ขา (Leg) และการใช้ไอพ่น (Jet) ในการเคลื่อนที่ เมื่อมีการสั่งให้เคลื่อนที่จะต้องระบุองศา ทิศทางว่าเป็นทางซ้ายหรือขวาและระยะทางที่ต้องการ จึงต้องมีการคำนวณหาตำแหน่งเป้าหมายโดยรายละเอียดการทำงานของการหาตำแหน่งเป้าหมายและการเคลื่อนที่นั้นได้อธิบายไว้ในหัวข้อที่ 3.2.1 และ 3.2.2

2. การ scan เป็นการทำงานเกี่ยวกับจุดมุมทั้งสามของพื้นที่ scan ซึ่งเป็นรูปสามเหลี่ยมโดยการทำงานจะอยู่ในคลาส Scan รายละเอียดวิธีการคำนวณอธิบายไว้ในหัวข้อที่ 3.2.3

3. การโจมตี เกิดขึ้นเมื่อหุ่นยนต์สั่งยิงจากอาวุธที่หุ่นยนต์ตัวนั้นมีและสามารถที่จะใช้งานได้ โดยการสั่งยิงนั้นจะมีการทำงานจากคลาส Weapon โดย คลาส Weapon นี้จะทำงานร่วมกับคลาส ManageBullet ซึ่งเป็นคลาสที่ควบคุมการปล่อยลูกกระสุน และการยกเลิกการสั่งยิงหากต้องมีการหยุด สำหรับการปล่อยลูกกระสุนนั้นจะสามารถยิงได้ทางตรงเท่านั้น ซึ่งหากถ้าต้องการยิงในทิศทางอื่นต้องทำการหมุนหุ่นยนต์เสียก่อน เนื่องจากอาวุธแต่ละชนิดนั้นจะมีอัตราการยิงหรือ shooting rate ดังนั้นต้องรอให้ถึงเวลาที่กำหนดไว้เสียก่อนกระสุนลูกต่อไปจึงจะถูกปล่อยออกไปได้ โดยการรอนั้นจะรอด้วย clock ของสนามดังที่ได้กล่าวไปแล้วในตอนต้นในการทำงานของสนาม เมื่อถึงเวลาที่ต้องปล่อยลูกกระสุนจะมีการคำนวณตำแหน่งเป้าหมายจากทิศทางปัจจุบันของหน้าหุ่นยนต์และระยะทางของกระสุนจากอาวุธนั้นที่สั่งยิงซึ่งรายละเอียดการคำนวณนั้นจะใช้หลักการเดียวกับการคำนวณตำแหน่งเป้าหมายและการเคลื่อนที่ของหุ่นยนต์ที่ได้อธิบายในหัวข้อที่ 3.2.1 และ 3.2.2 ซึ่งจะมีการทำงานอยู่ในคลาส Shooter

4. การถูกโจมตี จะเกิดขึ้นเมื่อมีการสั่งยิงจากหุ่นยนต์ตัวใดตัวหนึ่งหรือทั้งคู่โดยมีการคำนวณดังต่อไปนี้

1. ความถนัดในการใช้อาวุธของคู่ต่อสู้จะนำมาใช้ในการคำนวณว่าการยิงครั้งนั้นจะโดนโจมตีหรือไม่

2. หากคำนวณได้ว่าต้องถูกโจมตีแล้ว จะมีการสุ่มตัวเลขของการโดนยิงว่าจะโดนส่วนใดของหุ่นยนต์จากความน่าจะเป็นของส่วนต่างๆคือ แขนข้างละ 15% ขา 25% ลำตัว 50%

3. เมื่อคำนวณได้แล้วว่าจะโดนส่วนใดของหุ่นยนต์แล้ว จะใช้เลขสุ่ม 1-5 ถ้าได้เลข 5 แสดงว่าจะต้องเสียพลังด้วยพลังโจมตีของอาวุธนั้นสูงสุด และถ้าได้เลข 1แสดงว่าจะเสียพลังด้วยพลังโจมตีของอาวุธนั้นต่ำ โดยเป็นเปอร์เซ็นต์เทียบกับพลังทำลายของอาวุธนั้น ๆ

5. การจัดการกับเหตุการณ์ต่าง ๆ ทั้ง 6 เหตุการณ์

- เหตุการการณ์ปกติ จะมีการ new Instance ตั้งแต่เริ่มการต้อสู้ เมื่อจบการทำงานจากการตรวจสอบว่าไม่มีคำสั่งใดได้ทำงาน จะมีการ new Instance ขึ้นใหม่และสั่งให้เริ่มทำงานอีกครั้ง

- เหตุการณ์พบศัตรู จะมีการตรวจสอบว่าศัตรูอยู่ในขอบเขตของพื้นที่ scan หรือไม่จากคลาส ScanEnemy ซึ่งจะมีการนำเอาตำแหน่งของจุด scan ทั้งสามมาเปรียบเทียบกับตำแหน่งของคู่ต่อสู้ โดยหลักการคำนวณได้อธิบายไว้ในหัวข้อที่ 3.2.4 เมื่อได้รับค่าว่า scan พบคู่ต่อสู้แล้วจะมีการ new Instance ขึ้นและสั่งให้เริ่มทำงาน ในระหว่างนั้นหากรอบต่อไปของการแสดงผลจากสนามหุ่นยนต์หลุดจากเหตุการณ์พบศัตรูจะมีการสั่งให้ Instance ที่สร้างขึ้นมานั้นจบการทำงาน และจะสร้าง Instance ขึ้นอีกครั้งเมื่อจบการทำงานทั้งหมดและยังเกิดเหตุการณ์พบศัตรูอยู่หรือเกิดเหตุการณ์พบศัตรูขึ้นใหม่อีกครั้ง

- เหตุการณ์ถูกโจมตี เกิดขึ้นจากหลักการคิดที่ได้กล่าวไปแล้ว เมื่อเกิดเหตุการณ์นี้จะมีการสร้าง Instance ที่แทนเหตุการณ์นี้ขึ้นและสั่งให้เริ่มทำงาน เมื่อทำงานจบแล้วหากเกิดเหตุการณ์นี้อีกก็จะมีการสร้าง Instance ขึ้นอีก

- เหตุการณ์พบขอบสนาม จะมีคลาส ScanBorder เพื่อทำการคำนวณว่าพื้นที่ scan นั้นพบขอบของสนามแล้วหรือยังซึ่งหลักการคำนวณจะอธิบายไว้ในหัวข้อที่ 3.2.5 เมื่อเกิดเหตุการณ์นี้ขึ้นจะมีการสร้าง Instance ที่แทนเหตุการณ์นี้ขึ้นและสั่งให้เริ่มทำงาน เมื่อทำงานจบแล้วหากเกิดเหตุการณ์นี้อีกก็จะมีการสร้าง Instance ขึ้นอีกเช่นเดียวกับเหตุการณ์ถูกโจมตี

- เหตุการณ์ชนศัตรูและเหตุการณ์ชนขอบสนามจะมีการตรวจสอบว่าพื้นที่ของหุ่นยนต์นั้นเท่ากับหรือเกินขอบเขตของสนามหรือพื้นที่ตัวของศัตรูหรือไม่ หากใช่ก็จะมีการการสร้าง Instance ที่แทนเหตุการณ์นี้ขึ้นและสั่งให้เริ่มทำงาน เมื่อทำงานจบแล้วหากเกิดเหตุการณ์นี้อีกก็จะมีการสร้าง Instance ขึ้นอีกเช่นเดียวกับเหตุการณ์ถูกโจมตี การชนศัตรูหรือชนขอบสนามนั้นจะต้องมีการลดพลังงานของทุกส่วนของหุ่นยนต์ด้วย

นอกจากนี้ยังมีคลาส Rotation ที่จะทำการหมุนหรือเปลี่ยนองศาของหน้าหุ่นยนต์โดยทำการลดหรือเพิ่มองศาขึ้นทีละหนึ่งองศาตาม clock ของสนาม

3.2 ทฤษฎีที่ใช้ในการทำงานของเกม

3.2.1 ทฤษฎีการคำนวณหาตำแหน่งเป้าหมายของหุ่นยนต์และลูกกระสุน

เนื่องจากหุ่นยนต์นั้นมีทิศทางหรือองศาของหน้าของหุ่นยนต์โดยเทียบกับสนาม คือมีองศาตั้งแต่ 0 – 360 องศา ฉะนั้นในการคำนวณการเคลื่อนที่ของทั้งหุ่นยนต์และลูกกระสุนจึงต้องนำองศาของหุ่นยนต์มาใช้ในการคำนวณเพื่อหาตำแหน่งเป้าหมายด้วยโดยมีขั้นตอนคือ

1.กรณีเป็นการเคลื่อนที่ไปทางซ้ายของหุ่นยนต์ ให้นำองศาของหุ่นยนต์มาบวกกับองศาที่ต้องการเคลื่อนที่ไป ถ้าเป็นการเคลื่อนที่ไปทางขวาของหุ่นยนต์ให้นำองศาของหุ่นยนต์ไปลงกับองศาที่ต้องการเคลื่อนที่ไป ถ้าเป็นการเคลื่อนที่ตรงๆ จะข้ามขั้นตอนที่ 2 และ 3 ไป

2. ทำองศาที่ได้ให้อยู่ในขอบเขตของ 360 องศา ในกรณีที่ค่าที่ได้มากกว่า 360 องศา ไม่ว่าค่าองศาจะเป็นค่าบวกหรือลบ ให้นำองศาที่ได้มาทำการ Mod หรือการหารแบบเอาเศษ ก็จะได้ค่าองศาในขอบเขตของ 360 องศา

3. ถ้าค่าองศาที่ได้มีค่าติดลบให้นำค่าองศาที่ติดลบนี้ไปบวกกับ 360

4. ใช้หลักการตรีโกนมิติมาคำนวณจุดเป้าหมาย เช่นกรณีที่องศาอยู่ในจตุภาคที่ 1 จะได้ว่าค่า X จะเท่ากับ ระยะทางที่จะต้องเคลื่อนที่ คูณด้วยค่า cos ขององศานั้น ค่า Y เท่ากับ ระยะทางที่จะต้องเคลื่อนที่ คูณด้วยค่า sin ขององศานั้น เป็นต้น ในจตุภาคอื่นจะมีการคำนวณคล้ายกันแต่

จะทำการลบค่าองศาที่ตกอยู่ในจตุภาคอื่น ๆ ให้มีค่าอยู่ในช่วง 90 องศา เพื่อให้คำนวณค่า cos , sin ได้ง่ายขึ้นและต้องดูด้วยว่าเป็นการเทียบค่าองศากับแกน X หรือแกน Y

3.2.2 ทฤษฎีในการเคลื่อนที่

ในการเคลื่อนที่ของหุ่นยนต์และการเคลื่อนที่ของลูกกระสุนนั้นจะใช้หลักการเดียวกันคือ ใช้หลักการการลากเส้นตรงของ Bresenham หรือ Bresenham ' s Algorithm หลักการของ Bresenham มีอยู่ว่าจะมีการคำนวณว่าจุดต่อไปนั้นจะลงนั้น จะเป็นตำแหน่งถัดไป หรือเป็นตำแหน่งเดิม เพื่อลากเป็นเส้นตรง โดยมีรายละเอียดดังนี้

กรณี | m | < 1

จะใช้การขยับค่าในแกน X ทีละหนึ่ง ส่วนค่าในแกน Y นั้นจะมีการคำนวณ ว่าจะลงจุดถัดไปหรือลงที่ตำแหน่งเดิม โดยมีสูตร คือ

1. ต้องทราบจุด เริ่มต้น (X_0, Y_0) และจุดปลายทาง (X_m, Y_m)
2. ลงจุดเริ่มต้น(X_0, Y_0)
3. กำหนดให้ $P_K = 2dY - dX$

   โดย P_K คือการลงจุด ณ ตำแหน่งใดๆ

   dY คือค่าสัมบูรณ์ของผลต่างระหว่างค่า Y

   dX คือค่าสัมบูรณ์ของผลต่างระหว่างค่า X
4. หาค่า Y (Y_K) ต่อไปที่จะลง โดยเริ่มจาก k = 0

   - ถ้าค่า $P_K < 0$ จุดต่อไปที่จะลงคือ (X_K+1 , Y_K) ซึ่งก็คือตำแหน่ง X เดิม +1 และตำแหน่ง Y เดิม นั่นเอง จากนั้นกำหนดให้ค่าของการลงจุดครั้งต่อไปคือ $P_{K+1} = P_K + 2dY$

   - ถ้าค่า $P_K >= 0$ จุดต่อไปที่จะลงคือ (X_K+1 , Y_K+1) ซึ่งก็คือตำแหน่ง X เดิม +1 และตำแหน่ง Y เดิม +1 จากนั้นกำหนดให้ค่าของการลงจุดครั้งต่อไปคือ $P_{K+1} = P_K + 2dY - 2dX$
5. ทำซ้ำขั้นตอนที่ 4 จนกว่าค่า X จะถึงเป้าหมาย

กรณี | m | > 1

จะใช้การขยับค่าในแกน Y ทีละหนึ่ง ส่วนค่าในแกน X นั้นจะมีการคำนวณ ว่าจะลงจุดถัดไปหรือลงที่ตำแหน่งเดิม โดยมีสูตร คือ

1. ต้องรู้จุด เริ่มต้น(X_0, Y_0) และจุดปลายทาง (X_m, Y_m)
2. ลงจุดเริ่มต้น(X_0, Y_0)
3. กำหนดให้ $P_K = 2dX - dY$

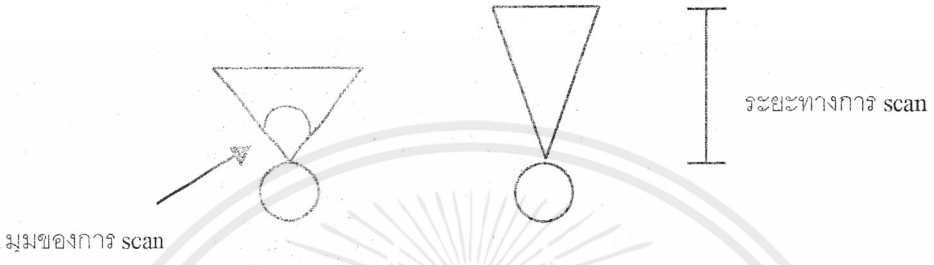
   โดย P_K , dY และ dX จะเหมือนกับกรณี | m | <1
4. หาค่า X (X_K) ต่อไปที่จะลง โดยเริ่มจาก k = 0

- ถ้าค่า $P_K < 0$ จุดต่อไปที่จะลงคือ (X_K , Y_K +1) ซึ่งก็คือตำแหน่ง X เดิม และตำแหน่ง Y เดิม +1 นั่นเอง จากนั้นกำหนดให้ค่าของการลงจุดครั้งต่อไปคือ $P_{K+1} = P_K + 2dX$

- ถ้าค่า $P_K >= 0$ จุดต่อไปที่จะลงคือ (X_K+1 , Y_K +1) ซึ่งก็คือตำแหน่ง X เดิม +1 และตำแหน่ง Y เดิม +1 จากนั้นกำหนดให้ค่าของการลงจุดครั้งต่อไปคือ $P_{K+1} = P_K + 2dX - 2dY$

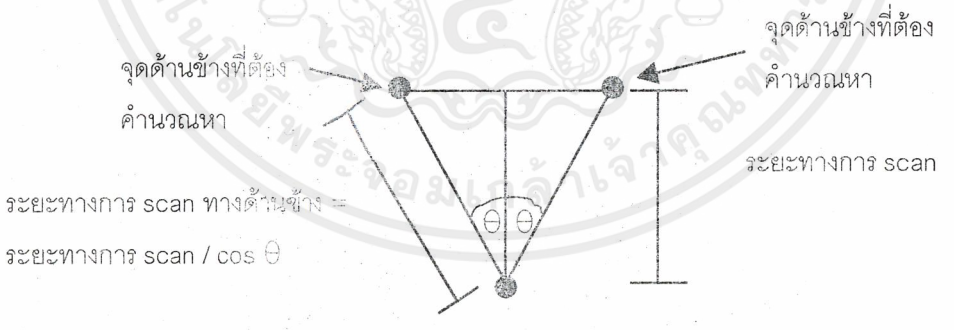
5. ทำซ้ำขั้นตอนที่ 4 จนกว่าค่า Y จะถึงเป้าหมาย

3.2.3 ทฤษฎีในการ scan ของหุ่นยนต์



รูปที่ 3.35 แสดงมุมการ scan และระยะทางการ scan

พื้นที่ scan ของหุ่นยนต์จะเป็นพื้นที่สามเหลี่ยม ซึ่งจะต้องมีการคำนวณตลอดเวลาไม่ว่าหุ่นยนต์จะมีการเคลื่อนที่หรือไม่ก็ตาม โดยหลักการจะเหมือนกับการคำนวณหาจุดเป้าหมายในการเคลื่อนที่ แต่เป็นการคำนวณหาจุดสามเหลี่ยมทางซ้ายและทางขวาแทน ระยะทางด้านข้างของพื้นที่ scan จะหาได้จาก ระยะทางการ scan หารด้วยค่า cos ของครึ่งหนึ่งของมุมการ scan จากหลักการของ สามเหลี่ยมพีทาโกรัส และหลักตรีโกนมิตินั่นเอง



รูปที่ 3.36 แสดงรายละเอียดของพื้นที่ scan

3.2.3.1 การ scan คู่ต่อสู้

หลังจากที่เราได้ค่าจุดทั้งสามของพื้นที่สามเหลี่ยม scan ของหุ่นยนต์แล้ว จะมีการนำจุดทั้งสามนั้นมาคำนวณในการ scan หาคู่ต่อสู้ว่าอยู่ในพื้นที่ scan นั้นหรือไม่ โดยมีวิธีการคือ

1.คำนวณหาจุดกึ่งกลางของพื้นที่สามเหลี่ยม scan จากการนำค่า X ของทั้งสามจุดมาบวกกันแล้วหารด้วย 3 จะได้ค่า X ที่เป็นจุดกึ่งกลาง และ นำค่า Y ของทั้งสามจุดมาบวกกันแล้วหารด้วย 3 จะได้ค่า Y ที่เป็นจุดกึ่งกลาง ซึ่งจุดกึ่งกลางที่ได้นี้จะเป็นจุดที่อยู่ในพื้นที่สามเหลี่ยม scan

2.นำค่าจุดกึ่งกลางที่ได้นี้ไปหาอสมการของเส้นทั้งสามของพื้นที่ scan เมื่อได้ค่าอสมการแล้วให้นำจุดที่ต้องการทดสอบว่าอยู่ในพื้นที่สามเหลี่ยม scan นี้หรือไม่ ซึ่งในที่นี้จะใช้จุดที่ทดสอบถึง 8 จุด เนื่องจากหุ่นยนต์นั้นมีขนาดพื้นที่เป็นสี่เหลี่ยมจัตุรัส จึงนำจุดมุมทั้ง 4 และจุดกึ่งกลางระหว่างมุมอีก 4 จุด รวมเป็น 8 จุดมาใช้แทนค่าในอสมการทั้งสามเส้น ถ้าแทนทั้งสามเส้นแล้วได้ค่าเป็นจริง แสดงว่าจุดที่นำมาทดสอบนั้นอยู่ในพื้นที่สามเหลี่ยม scan ทำให้ได้ว่าการ scan ครั้งนั้นพบคู่ต่อสู้ แต่ถ้าทดสอบครบทั้ง 8 จุดแล้ว ไม่เป็นจริงแสดงว่าการ scan ครั้งนั้นไม่พบคู่ต่อสู้

3.2.3.2 การ scan ขอบสนาม

จะนำจุดทั้งสามของพื้นที่ scan มาคำนวณเปรียบเทียบว่า จุดทั้งสามนั้นเท่ากับหรือเกินขอบเขตของสนามหรือไม่ ถ้าใช่จะมีการคืนค่าด้วยว่าเป็นการ scan พบทางด้านไหนของพื้นที่ scan คือ ทางซ้าย ทางขวา หรือทางตรง ของ scan นอกจากนี้ยังคืนค่าระยะห่างจากขอบสนามที่พบด้วย

3.3 ความต้องการของระบบในการเล่น

3.3.1 ความต้องการทางด้าน HARDWARE

อย่างต่ำ	แนะนำ
- CPU :Pentium 233	- CPU : Pentium III 450
- RAM : 32 MB	- RAM : 64 MB
- Display : ขนาด 4 MB	- Display : ขนาด 16 MB
- Sound card 16 Bit	- Sound card 16 Bit
- Modem สำหรับเชื่อมต่อกับ Internet	- Modem 56 Kbps
- ความละเอียดของหน้าจอ 720 x 576	- ความละเอียด 800*600

3.3.2 ความต้องการทางด้าน SOFTWARE

- OS : Windows 95, Windows 98 Windows Me, Windows NT 4.0 หรือ Windows 2000 หรือสูงกว่านี้ , LINUX, Macintoch
- Browser : เช่น Internet Explorer 4 หรือสูงกว่านี้ , Netscape หรือ browser อื่นๆ
- JDK 1.2.2 หรือรุ่นที่สูงกว่านี้

บทที่ 4
การใช้งานโปรแกรม

จากการทดลองเราได้สร้าง 2 ส่วนคือส่วนของ web site ที่ใช้ในการแสดงรายละเอียดต่างๆ ของการเล่น ใช้ในการ Download Application, Upload robot และใช้ในการเล่นเกมโดยการเลือก robot 2 ตัว ที่ทำการ Upload ไว้มาต่อสู้กันโดยการแสดงเป็นรูปภาพ 2 มิติ และ Application จะเป็นส่วนที่ใช้สำหรับการสร้างหุ่นยนต์โดยมีการเลือกส่วนประกอบต่างๆประกอบขึ้นเป็นหุ่นยนต์ และเขียนโปรแกรมเพื่อสั่งให้หุ่นยนต์ทำงานในเหตุการณ์ต่างๆ ซึ่งจะสามารถดูรายละเอียดเพิ่มเติมว่าในแต่ละส่วนสามารถทำอะไรได้บ้างได้ในภาคผนวก

4.1 การทำงานของโปรแกรม

การทำงานทั้งหมดแบ่งได้เป็น 2 ส่วนดังนี้

1. ส่วนของโปรแกรมสำหรับสร้างหุ่นยนต์ ประกอบด้วย

ตารางที่ 4.1 แสดงรายละเอียดส่วนของโปรแกรมสำหรับสร้างหุ่นยนต์

ส่วนประกอบ	รายละเอียด
หน้าสำหรับใส่ชื่อหุ่นยนต์	สามารถใส่เป็นชื่อใหม่เพื่อสร้างหุ่นยนต์ตัวใหม่หรือใส่ชื่อเดิมเพื่อนำหุ่นยนต์ตัวเดิมมาปรับเปลี่ยนได้ นอกจากนี้ในหน้านี้ยังมีการตรวจสอบการตั้งชื่อของหุ่นยนต์ให้ถูกต้องด้วย
หน้าสำหรับเลือกส่วนประกอบ	สามารถเลือกส่วนประกอบต่างๆเพื่อประกอบขึ้นเป็นหุ่นยนต์ ซึ่งจะต้องเลือกส่วนของ cpu และ body ด้วยจึงสามารถไปยังหน้าต่อไปเพื่อทำงานต่อได้ นอกจากนี้ยังสามารถกดปุ่ม back เพื่อกลับไปยังหน้าสำหรับใส่ชื่อหุ่นยนต์ได้ และมีปุ่มสำหรับแสดงรายละเอียดของส่วนประกอบแต่ละแบบของหุ่นยนต์และรายละเอียดของอาวุธแต่ละประเภทด้วย
หน้าสำหรับใส่คำสั่ง	สามารถแสดงส่วนประกอบของหุ่นยนต์ที่ได้เลือกไว้จากหน้าเลือกส่วนประกอบ สามารถใส่คำสั่งการทำงานของหุ่นยนต์ในเหตุการณ์ทั้ง 6 เหตุการณ์ มีปุ่มสำหรับคอมไพล์ code การทำงานของหุ่นยนต์เพื่อให้เป็นไฟล์ .class และบันทึกส่วนประกอบของหุ่นยนต์และการทำงานของเหตุการณ์ทั้ง 6 เพื่อให้สามารถนำมาปรับเปลี่ยนได้ในภายหลังอีกครั้ง นอกจากนี้ยังมี

ตารางที่ 4.1 (ต่อ) แสดงรายละเอียดส่วนของโปรแกรมสำหรับสร้างหุ่นยนต์

ส่วนประกอบ	รายละเอียด
	ปุ่มเพื่อแสดงรายละเอียดของ method ต่าง ๆ ที่ผู้เล่นสามารถเรียกใช้ได้แบบย่อ และมีปุ่ม back สำหรับกลับไปยังหน้าเลือกส่วนประกอบเพื่อปรับเปลี่ยนส่วนประกอบใหม่ได้

สำหรับตัวอย่างการทำงานแสดงไว้ในหัวข้อที่ 4.2

2.ส่วนของ web site ประกอบด้วย

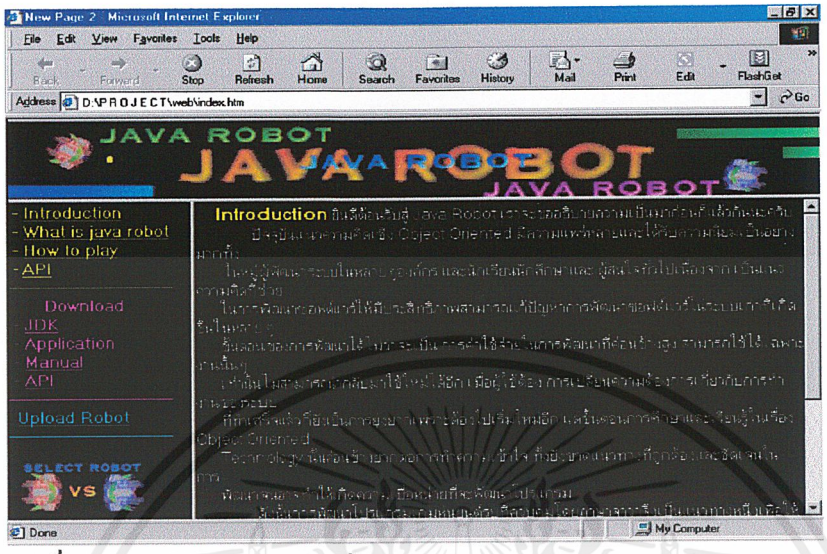
ตารางที่ 4.2 แสดงรายละเอียดส่วนของ web site

ส่วนประกอบ	รายละเอียด
Introduction	เป็นหน้าสำหรับเกริ่นแนะนำ
What is java robot	เป็นหน้าสำหรับอธิบายความหมายเกม
How to play	เป็นหน้าสำหรับอธิบายวิธีการเล่น พร้อมทั้งกฎเกณฑ์ต่างๆ
API	เป็นหน้าสำหรับแสดง method ต่างๆที่ผู้เล่นสามารถเรียกใช้ได้
Download	เป็นส่วนที่ link ไปยังส่วนของการดาวน์โหลดโปรแกรม JDK โปรแกรมสำหรับสร้างหุ่นยนต์ คู่มือของเกมและ เอกสาร API
Upload robot	เป็นหน้าสำหรับการ upload หุ่นยนต์
Select robot	เป็นหน้าสำหรับการเลือกหุ่นยนต์ที่ได้ upload ไว้แล้ว เพื่อต่อสู้
หน้าแสดงผล	เป็นหน้าสำหรับการแสดงผลการต่อสู้ของหุ่นยนต์ จากเหตุการณ์ต่างๆ 6 เหตุการณ์ที่สามารถเกิดขึ้นได้ คือ เหตุการณ์ปกติ เหตุการณ์พบศัตรู เหตุการณ์โดนโจมตี เหตุการณ์พบขอบสนาม เหตุการณ์ชนศัตรู และเหตุการณ์ชนขอบสนาม ซึ่งคำสั่งหรือการทำงานของหุ่นยนต์ในเหตุการณ์ต่าง ๆนั้นจะถูกคำนวณจัดการในส่วนหน้าแสดงผลนี้

สำหรับตัวอย่างการทำงานแสดงไว้ในหัวข้อที่ 4.2 นอกจากนี้การทำงานของหุ่นยนต์ในเหตุการณ์ต่างๆ ซึ่งเกิดจากการสั่งจากส่วนประกอบต่าง ๆ ของหุ่นยนต์ สามารถทำงานได้ตามที่แสดงไว้ในหัวข้อ method ที่ผู้เล่นสามารถเรียกใช้ได้ หรือ API ที่อยู่ในภาคผนวก

4.2 ผลการเล่นเกม

เริ่มต้นเราจะทำการ online เข้าสู่ internet ไปที่ web site ของ Java Robot ดังรูปที่ 4.1

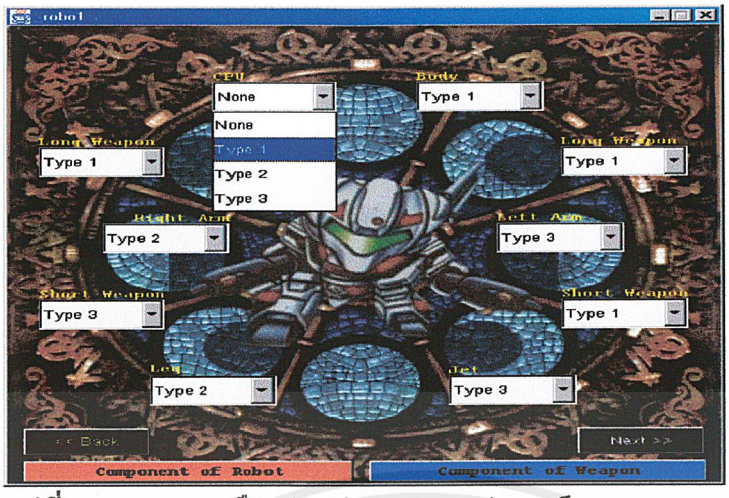


รูปที่ 4.1 แสดง web page หน้าแรกของ Java Robot

จากนั้นคลิก download JDK และ download Application ที่ใช้ในการสร้างหุ่นยนต์เพื่อติดตั้ง หลังจากติดตั้ง JDK และ Application เรียบร้อยแล้ว เริ่มต้นทำการสร้างหุ่นยนต์โดยเปิดโปรแกรม Application เมื่อเปิดโปรแกรมขึ้นมาหน้าแรกจะมีลักษณะดังรูปที่ 4.2 ทำการใส่ชื่อ _robo1 เป็นชื่อของหุ่นยนต์เพื่อทำการสร้างหุ่นยนต์ใหม่ที่ช่อง Build new robot จากนั้นคลิกปุ่ม Next

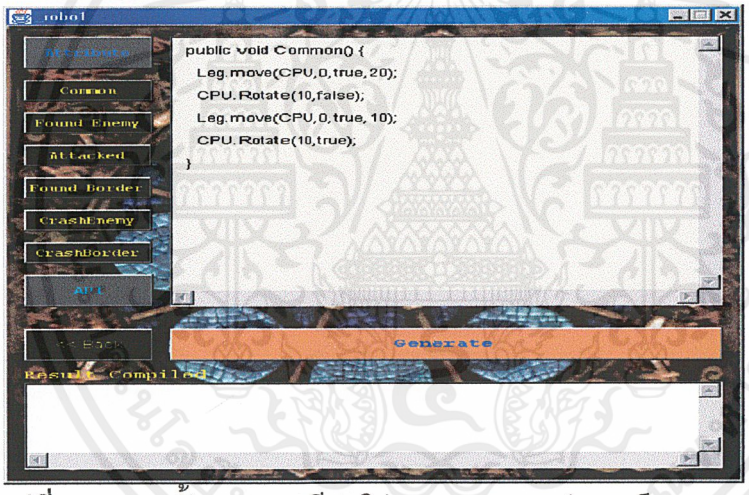
รูปที่ 4.2 แสดงหน้าแรกสำหรับใส่ชื่อของหุ่นยนต์ของ Application สำหรับสร้างหุ่นยนต์

ขั้นต่อไปทำการเลือกส่วนประกอบต่างๆของหุ่นยนต์ ดังรูปที่ 4.3 เสร็จแล้วก็คลิก Next



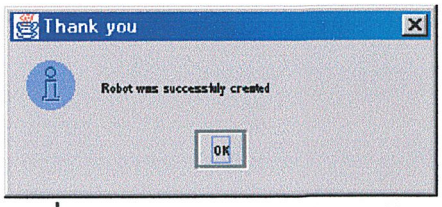
รูปที่ 4.3 แสดงการเลือกส่วนประกอบของหุ่นยนต์

จากนั้นทำการเขียนโปรแกรมด้วยภาษาจาวาในเหตุการณ์ต่าง ๆ ดังรูปที่ 4.4



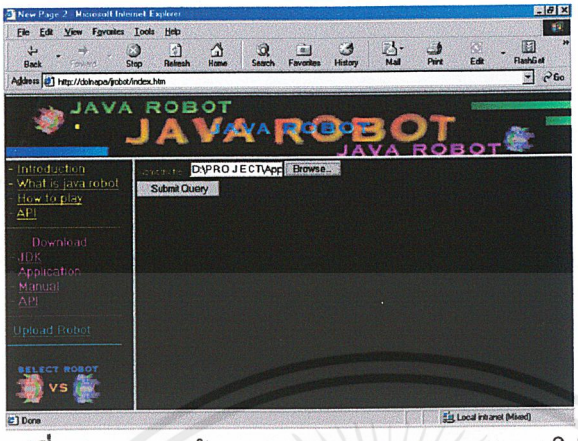
รูปที่ 4.4 แสดงขั้นตอนการเขียนโปรแกรมควบคุมหุ่นยนต์

จากนั้นคลิกปุ่ม Generate เพื่อ Compile โปรแกรม เมื่อ compile ผ่านแล้วจะขึ้น message ดังรูปที่ 4.5



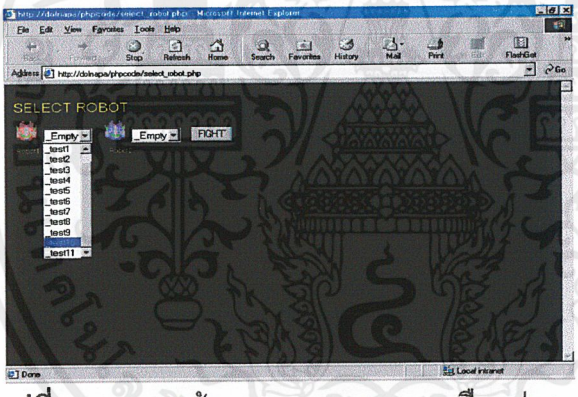
รูปที่ 4.5 แสดง message box หลังจาก compile โปรแกรมเสร็จแล้ว

จากนั้นนำไฟล์ _robo1.class ที่ได้หลังจากการ compile แล้ว อัพโหลดขึ้นไปบน web page ในหน้าแรกของ web site Java Robot ดังรูปที่ 4.6



รูปที่ 4.6 แสดงหน้า web page ของการ upload โปรแกรมหุ่นยนต์ที่สร้างขึ้น

เมื่อ upload เสร็จแล้วให้คลิกที่รูป SELECT ROBOT เพื่อเลือกหุ่นยนต์ที่จะต่อสู้ดังรูปที่ 4.7

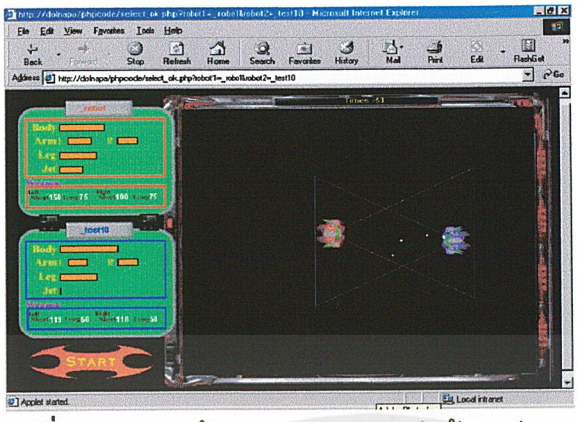


รูปที่ 4.7 แสดงหน้า web page ของการเลือกหุ่นยนต์เพื่อต่อสู้

เมื่อเลือกหุ่นยนต์ที่ต้องการต่อสู้ได้แล้วก็คลิกที่ปุ่ม FIGHT เพื่อไปที่หน้าต่อสู้ ดังรูปที่ 4.8

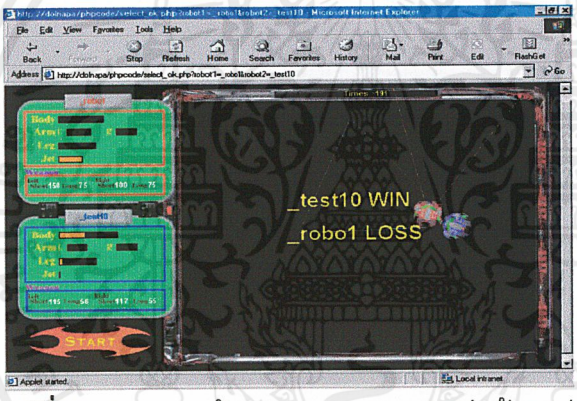
รูปที่ 4.8 แสดงหน้า web page ของการต่อสู้ของหุ่นยนต์

คลิกปุ่ม START เพื่อเริ่มการต่อสู้ของหุ่นยนต์ จากนั้นก็ปล่อยให้หุ่นยนต์ต่อสู้กัน ดังรูปที่ 4.9



รูปที่ 4.9 แสดงหน้า web page การต่อสู้ของหุ่นยนต์

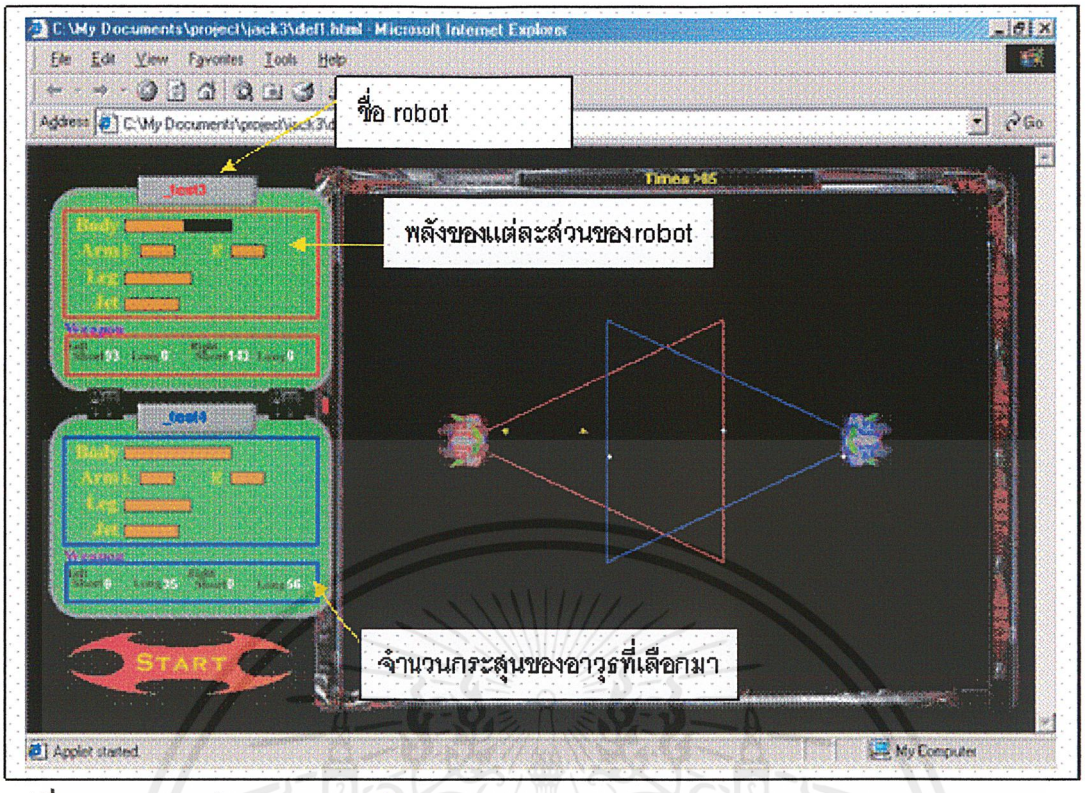
เมื่อรู้ผลแพ้ชนะก็จะขึ้นผลการต่อสู้ดังรูปที่ 4.10 เป็นอันจบสิ้นการเล่นเกม



รูปที่ 4.10 แสดงหน้า web page ผลการต่อสู้ของหุ่นยนต์

4.3 ผลการต่อสู้ของหุ่นยนต์ในเหตุการณ์ต่าง ๆ

การต่อสู้ของหุ่นยนต์นั้นขึ้นอยู่กับส่วนประกอบต่างๆ ของหุ่นยนต์แต่ละตัว ถ้าหุ่นยนต์ที่มีน้ำหนักมากก็จะทำให้หุ่นยนต์ตัวนั้นช้าแต่ก็จะมีพลังมากตามไปด้วย และอาวุธที่เลือกก็มีผลเช่นดังรูปที่ 4.11

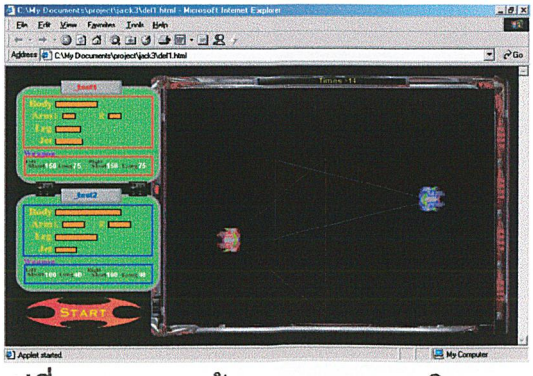


รูปที่ 4.11 แสดงหน้า web page ข้อแตกต่างระหว่างอาวุธ

จากรูปจะเห็นว่า robot _test3 ได้เลือกอาวุธระยะใกล้มาจึงทำให้โจมตีได้ใกล้กว่า robot _test4 ซึ่งได้เลือกอาวุธระยะไกลมา แต่จะเห็นได้ว่าอาวุธใกล้นั้นจะมีจำนวนกระสุนมากกว่าอาวุธไกล และยังมีข้อแตกต่างของอาวุธแต่ละชนิดเช่น อัตราการยิง หรือ ความเร็วของกระสุนด้วย

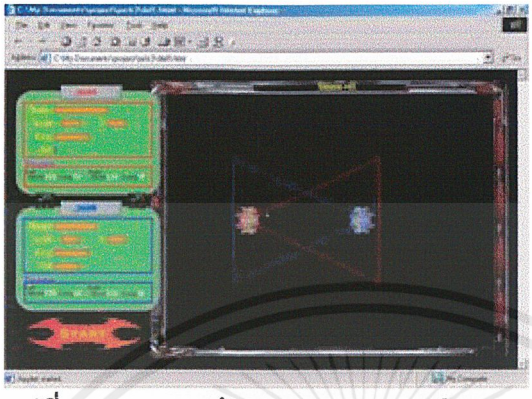
การเกิดเหตุการณ์สามารเกิดได้ดังต่อไปนี้

- การเกิดเหตุการณ์ Common มีการสั่งให้ robot ทั้ง 2 ตัวเคลื่อนที่ไปทางขวา 90 องศาrobot ทั้ง 2 ตัวก็จะเคลื่อนที่ไปทางขวาตามที่ได้สั่งไว้ และจะเห็นได้ว่า robot ทั้ง 2 ตัวมีพลังต่างๆ กัน robot สีน้ำเงินจะมีพลังเยอะกว่าสีแดงแต่ก็เคลื่อนที่ได้ช้ากว่าตามที่ได้กล่าวมาในตอนแรก และ robot ทั้ง 2 ตัวมีระยะ scan ต่างกันตามที่ได้เลือกไว้ ดังรูป 4.12

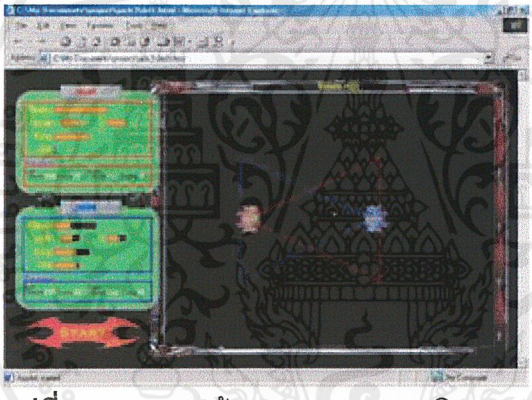


รูปที่ 4.12 แสดงหน้า web page การเกิดเหตุการณ์ Common

- การเกิดเหตุการณ์ FoundEnemy และ Attacked robot สีแดงมีการสั่งว่าถ้า scan พบศัตรูให้ยิงและ robot สีน้ำเงินเมื่อถูกโจมตีให้เคลื่อนที่ไปข้างหลัง robot ทั้ง 2 สามารถทำตามเหตุการณ์ทั้ง 2 ได้อย่างถูกต้อง ดังรูป 4.13 และ 4.14

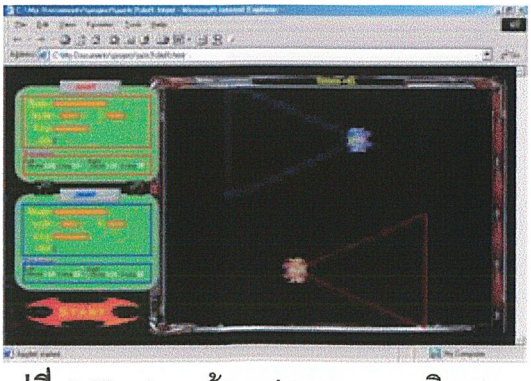


รูปที่ 4.13 แสดงหน้า web page การเกิดเหตุการณ์ FoundEnemy

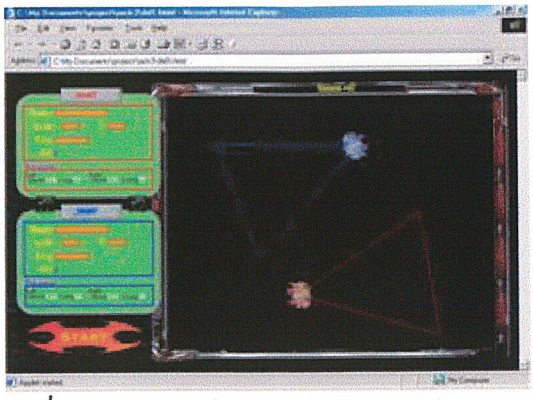


รูปที่ 4.14 แสดงหน้า web page การเกิดเหตุการณ์ Attacked

- การเกิดเหตุการณ์ FoundBorder robot ทั้ง 2 ตัวเมื่อ scan พบขอบสนามรบทางด้านขวาได้สั่งให้ robot หมุนมาซ้าย robot ทั้ง 2 ตัว สามารถทำตามได้อย่างถูกต้อง ดังรูป 4.15 และ 4.16

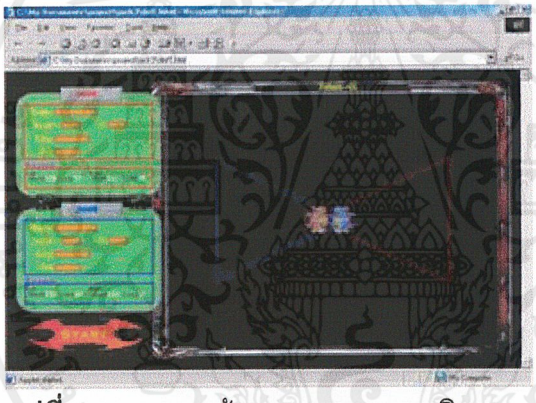


รูปที่ 4.15 แสดงหน้า web page การเกิดเหตุการณ์ FoundBorder รูปที่ 1

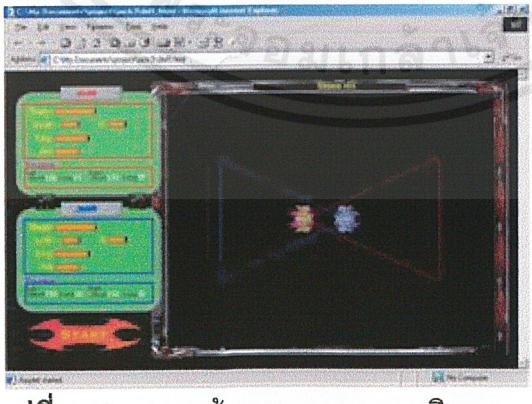


รูปที่ 4.16 แสดงหน้า web page การเกิดเหตุการณ์ FoundBorder รูปที่ 2

- การเกิดเหตุการณ์ CrashEnemy robot ทั้ง 2 ตัวเมื่อชนศตรูได้สั่งให้ robot ถอยหลัง robot ทั้ง 2 ตัว สามารถทำตามได้อย่างถูกต้อง ดังรูป 4.17 และ 4.18

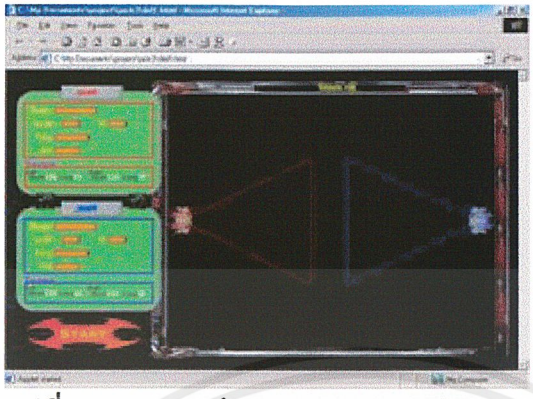


รูปที่ 4.17 แสดงหน้า web page การเกิดเหตุการณ์ CrashEnemy รูปที่ 1

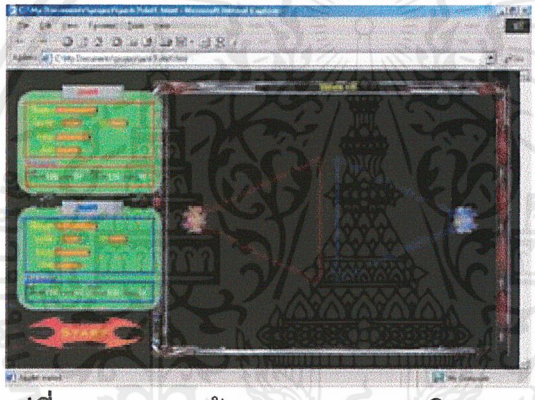


รูปที่ 4.18 แสดงหน้า web page การเกิดเหตุการณ์ CrashEnemy รูปที่ 2

- การเกิดเหตุการณ์ CrashBorder robot ทั้ง 2 ตัวเมื่อชนขอบสนามรบได้สั่งให้ robot เดินหน้า robot ทั้ง 2 ตัว สามารถทำตามได้อย่างถูกต้อง ดังรูป 4.19 และ 4.20



รูปที่ 4.19 แสดงหน้า web page การเกิดเหตุการณ์ CrashBorder รูปที่ 1



รูปที่ 4.20 แสดงหน้า web page การเกิดเหตุการณ์ CrashBorder รูปที่ 2

** ตั้งแต่เหตุการณ์ FoundEnemy จนถึงเหตุการณ์สุดท้าย เราได้ stop เหตุการณ์ Common เมื่อเกิดเหตุการอื่นเพื่อไม่ให้เกิดเหตุการณ์ซ้อนกัน

บทที่ 5
สรุปผลการศึกษา ข้อเสนอแนะ และแนวทางการพัฒนาต่อ

5.1 สรุปผลปัญหาพิเศษ

5.1.1 ในการพัฒนาโปรแกรมเกมหุ่นยนต์รบที่ควบคุมโดยภาษาจาวา ในปัญหาพิเศษฉบับนี้สามารถนำมาใช้ในการเล่นเกมได้อย่างมีประสิทธิภาพทั้งส่วนของในเว็บไซต์ และส่วนของโปรแกรมที่ผู้ใช้ต้องทำการ download เพื่อนำไปสร้างหุ่นยนต์

5.1.2 เนื่องจากเกมหุ่นยนต์รบที่ควบคุมโดยภาษาจาวาถูกออกแบบให้เป็นภาษาจาวา ซึ่งการประมวลผลจะทำที่ บราวเซอร์ ทำให้เวลาในการประมวลผลใช้ค่อนข้างมากกว่าการเล่นเกมทั่วไป

5.1.3 ผู้เล่นเกมจะสามารถพัฒนาทักษะในการเขียนโปรแกรมภาษาจาวา เหมาะกับผู้ที่จะฝึกฝนในการเขียนโปรแกรมจาวาและช่วยทำให้การศึกษาเกี่ยวกับทฤษฎีหลักการโปรแกรมเชิงวัตถุเข้าใจได้ง่ายขึ้นโดยใช้เกมเป็นสื่อในการฝึกฝน

5.1.4ผู้พัฒนาได้รับความรู้เกี่ยวกับการเขียนโปรแกรมเชิงวัตถุ และการติดต่อสื่อสารกันระหว่าง object โดยเฉพาะในภาษาจาวา

5.2 ปัญหา

5.2.1 การเล่นเกมหุ่นยนต์รบที่ควบคุมโดยภาษาจาวา ควรจะศึกษาถึงความพร้อมของอุปกรณ์ เนื่องจากการเล่นจะต้องมีระบบเครือข่ายที่ดี เนื่องจากต้องส่งผ่านข้อมูลผ่านระบบเครือข่ายเป็นจำนวนมาก รวมไปถึงประสิทธิภาพของคอมพิวเตอร์ที่นำมาใช้เล่นด้วย ถ้าคอมพิวเตอร์มีการประมวลผลช้า เกมจะมีการทำงานได้ช้าและขาดความตื่นเต้นในการเล่นเกม

5.3.3 เหตุการณ์ที่นำมาให้ผู้เล่นกำหนดวิธีการต่อสู้ของหุ่นยนต์ ปัจจุบันมี 6 เหตุการณ์ คือ ปกติ โจมตี ถูกโจมตี ชนขอบ ชนศัตรู พบศัตรู ซึ่งสามารถที่จะขยายเพิ่มอีกหลายๆเหตุการณ์ได้ จะทำให้เกมมีความซับซ้อนมากขึ้นและมีความสนุกสนานมากขึ้น

5.3.4 ส่วนประกอบที่นำมาให้ผู้เล่นเลือกเพื่อจะนำไปใช้ในประกอบขึ้นเป็นหุ่นยนต์ ปัจจุบันมี 10 ส่วน ซึ่งสามารถที่จะขยายเพิ่มอีกหลายๆส่วนได้ จะทำให้เกมมีความซับซ้อนมากขึ้นและมีความสนุกสนานมากขึ้นโดยเฉพาะส่วนของอาวุธโจมตี ซึ่งปัจจุบันมี 2 ประเภทคืออาวุธระยะใกล้และระยะไกล ในแต่ละประเภทก็มีอาวุธให้เลือกเพียง 3 ชิ้นและสามารถให้ติดกับแขนของหุ่นยนต์ได้ข้างละ 2 ชิ้น คืออาวุธระยะใกล้ 1 ชิ้นและอาวุธระยะไกลอีก 1 ชิ้น

5.3.5 ปัจจุบันในสนามสามารถนำหุ่นยนต์เข้าต่อสู้ได้ครั้งละ 2 ตัวเท่านั้น การพัฒนาสามารถเพิ่มในส่วนที่ทำให้หุ่นยนต์สามารถต่อสู้กันเป็นทีมได้ อาจเป็นทีมละ 2 ตัว หรือ ทีมละ 3 ตัว หรือว่าจะสามารถต่อสู้ทีละหลายๆตัวได้พร้อมๆกัน

5.3.6 ปัจจุบันในสนามต่อสู้จะเป็นอวกาศ ซึ่งไม่มีสิ่งกีดขวางใดใดเลย การพัฒนาต่อสามารถเพิ่มในส่วนของสิ่งกีดขวาง เช่น มีบริเวณที่จะป้องกันการ สแกนของศัตรู มีดาวที่สามารถการป้องกันอาวุธของศัตรูได้ มีไฟฟ้าที่ถ้าเข้าไปในบริเวณนั้นๆ จะเสียพลัง เป็นต้น

5.3.7 ปัจจุบันหลังจากการต่อสู้แล้วจะมีการแสดงผลแพ้ชนะหรือเสมอเท่านั้น ในการพัฒนาสามารถเพิ่มการจัดเก็บผลแพ้ชนะเป็นคะแนนเก็บไว้เป็นประวัติหรือเป็นฐานข้อมูลของหุ่นยนต์แต่ละตัว

ภาคผนวก ก.
การติดตั้งโปรแกรม

การติดตั้งโปรแกรม

- JDK โดยในที่นี้เป็นการขั้นตอนการติดตั้ง jdk 1.2.2

1. ดับเบิ้ลคลิกที่ตัวโปรแกรม jdk-1_2_2.exe โปรแกรม JDK จะถูก Install

2. Set Class path เพื่อให้สามารถเรียกใช้ Javac.exe และ Java.exe ได้จากในไดเรกทอรีใดๆ โดยพิมพ์ค่าเหล่านี้ลง ไปในไฟล์ autoexec.bat "SET PATH= C:\JDK1.2.2\BIN"

3. เมื่อเราต้องการที่จะ Compile โปรแกรมที่อ้างถึงคลาสต่างๆ จะทำได้โดยเพิ่มประโยค "SET CLASSPATH=.;C:\JDK1.2.2\LIB\CLASSES.ZIP" ที่ ไฟล์ autoexec.bat

4. ปกติเราจะใช้ default class path โดยจะ กำหนด ดังนี้ คือ

".;$JAVA\CLASSES;$JAVA\LIB\CLASSES.ZIP" ไฟล์ autoexec.bat

- Application for creating robot

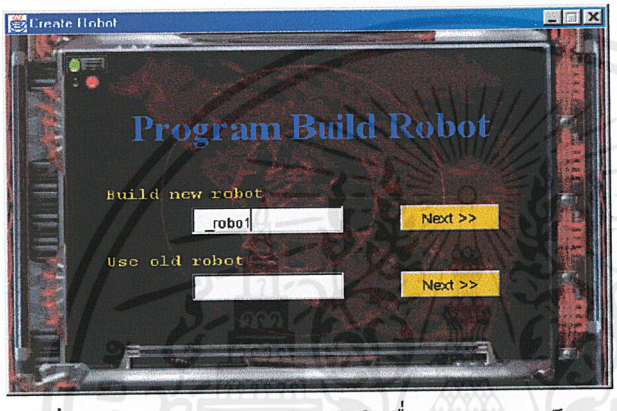
1. ดาวน์โหลดโปรแกรมจาก web site
2. ดับเบิลคลิกที่ ไฟล์ JRobot.exe ที่ดาวน์โหลดมาได้
3. โปรแกรมจะถูกเปิดขึ้น ซึ่งจะสามารถสร้างหุ่นยนต์ได้

ภาคผนวก ข.
คู่มือการใช้โปรแกรม

คู่มือการใช้ส่วนของ Application

คู่มือการใช้ในส่วนของ Application เป็นส่วนของโปรแกรมที่ใช้สำหรับให้ผู้เล่นสร้างหุ่นยนต์โดย สามารถ Downloadโปรแกรมนี้ได้จาก wed page

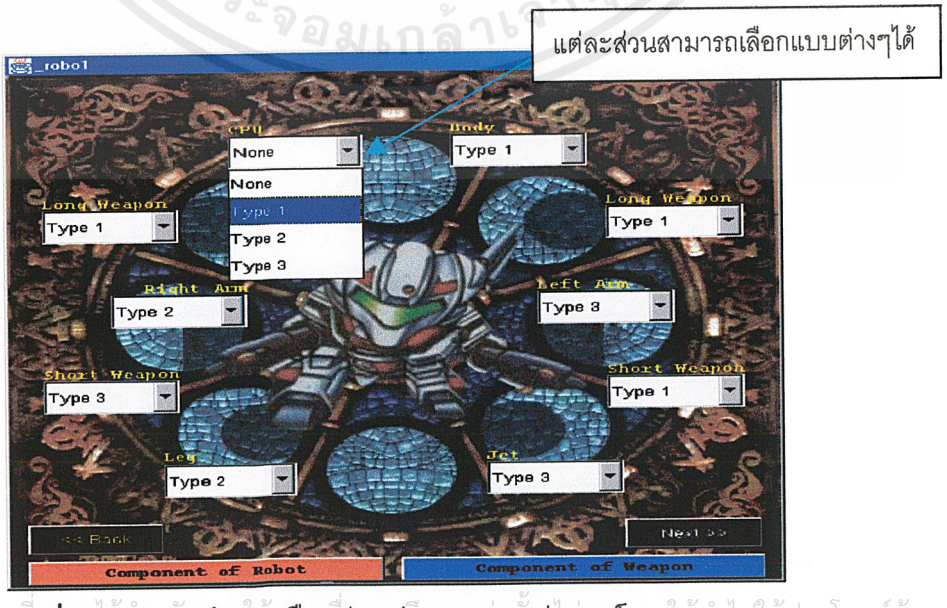
เมื่อเปิดโปรแกรมขึ้นมาหน้าแรกจะมีลักษณะดังรูปที่ ข-1 ให้ทำการใส่ชื่อ robot เพื่อทำการสร้าง robot



รูปที่ ข-1 แสดงหน้าแรกสำหรับใส่ชื่อของหุ่นยนต์ของ Application สำหรับสร้างหุ่นยนต์

- Build new robot จะให้ใส่ชื่อ robot ในกรณีที่ผู้เล่นต้องการ robot ตัวใหม่หรือ ยังไม่เคยมี robot เลยโดยการใส่ชื่อจะต้องมี "_ "นำหน้าชื่อ robot เสมอ เช่น _Robot1 เป็นต้น

- Use old robot จะให้ใส่ชื่อ robot ในกรณีที่ผู้เล่นมี robot อยู่แล้วต้องการมาแก้หรือเพิ่มเติม robot ตัวนั้น แล้วกด next จะไป ดังรูปที่ ข-2 เพื่อทำการสร้าง robot ต่อไป



รูปที่ ข-2 แสดงหน้าการเลือกส่วนประกอบของหุ่นยนต์

รูปที่ ข-2 จะเป็นส่วนที่ให้ผู้เล่นทำการเลือกส่วนประกอบต่างๆของหุ่นยนต์ ซึ่งแต่ละส่วนจะมีอยู่อย่างละ 3 ประเภทให้เลือกใช้ได้โดยมีความสามารถต่างกันออกไป ผู้เล่นจึงสามารถเลือกส่วนต่างๆได้ แล้วแต่ความชอบของแต่ละคนว่าจะให้ robot ที่สร้างมีความสามารถอย่างไรบ้าง

- ในส่วนนี้จะให้เลือกส่วนประกอบต่างๆของหุ่นยนต์ ถ้าเป็นหุ่นยนต์ตัวเดิมจะมีส่วนประกอบต่างๆที่ได้เลือกไว้แล้วและสามารถเปลี่ยนแปลงได้ โดยแต่ละส่วนจะมีให้เลือก 3 แบบ คือ Type1, Type2, Type3

* ถ้าไม่เลือก Body จะไม่สามารถเลือกส่วนอื่นได้ และถ้าไม่เลือก Right Arm ก็จะไม่สามารถเลือก weapon ข้างขวาได้ เช่นเดียวกัน ถ้าไม่เลือก Left Arm ก็จะไม่สามารถเลือก weapon ข้างซ้ายได้

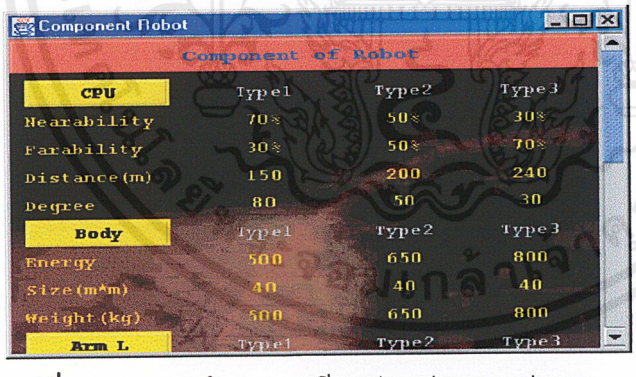
- ปุ่ม Component of Robot จะมีรายละเอียดของ Component ของ Robot ให้ดูว่าแต่ละแบบเป็น อย่างไร มีความสามารถแบบใดบ้าง ดังรูปที่ ข-3

- ปุ่ม Component of Weapon จะมีรายละเอียดของ Component ของ Weapon ให้ดูว่าแต่ละแบบเป็น อย่างไร มีความสามารถแบบใดบ้าง ดังรูปที่ ข-4

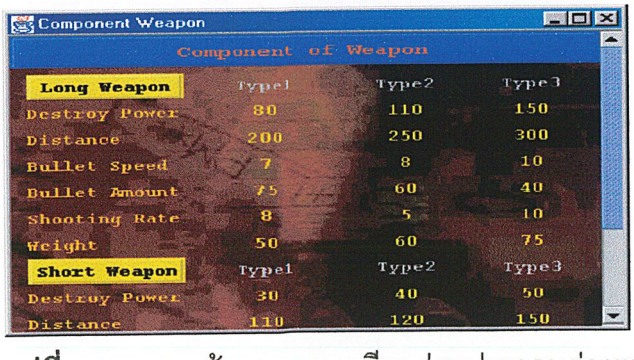
- ปุ่ม next จะไปที่หน้าต่อไป สำหรับทำการสร้าง code ของ class robot ของผู้เล่นและทำการ new ส่วนประกอบที่ผู้เล่นเลือก

* ถ้าไม่เลือก Body และ Cpu จะไม่สามารถกดปุ่ม next ได้

- ปุ่ม Back ทำให้เปลี่ยนหน้ากลับไปยังหน้าแรก ดังรูป ข-1



รูปที่ ข-3 แสดงหน้ารายละเอียดส่วนประกอบต่างๆของหุ่นยนต์



รูปที่ ข-4 แสดงหน้าของรายละเอียดส่วนประกอบต่างๆของอาวุธ

รายละเอียดของหน้า Component of Robot และ Component of Weapon

Cpu

- Near ability จะเป็นความสามารถในการใช้ Short weapon ซึ่งจะบอกเป็นเปอร์เซ็นต์ถ้า ยิ่งมีเปอร์เซ็นต์มากก็จะทำให้ใช้ Short weapon แม่นยำขึ้น ซึ่งจะมีผลกับการใช้ Short weapon โจมตีคู่ต่อสู่ว่าจะโดนหรือไม่

- Far ability จะเป็นความสามารถในการใช้ Long weapon ซึ่งจะบอกเป็นเปอร์เซ็นต์ถ้า ยิ่งมีเปอร์เซ็นต์มากก็จะทำให้ใช้ Long weapon แม่นยำขึ้น ซึ่งจะมีผลกับการใช้ Long weapon โจมตีคู่ต่อสู่ว่าจะโดนหรือไม่

- Distance จะเป็นระยะทางในการ scan ว่า robot ตัวนั้นจะสามารถ scan ได้ไกลแค่ไหน แต่ระยะทางการ scan มีผลต่อมุมของการ scan ด้วย ยิ่งระยะทางในการ scan มากก็จะทำให้มุมของการ scan น้อยลงไปด้วย

Body, Arm Right, Arm Right, Leg

- Energy จะบอกพลังของส่วนนั้นว่าจะสามารถทนการโจมตีได้แค่ไหนถ้า energy ของ body หมดจะทำให้ robot ตัวนั้นถูกทำลายทันที แต่ถ้า energy ของ arm ข้างใดหมดข้างนั้นก็จะไม่สามารถใช้ weaponข้างนั้นได้ และ ถ้า energy ของ leg หมดจะทำให้ robot ตัวนั้นไม่สามารถเคลื่อนที่ได้ นอกจากจะใช้ jet

- Size จะบอกขนาดของส่วนนั้น ซึ่งยิ่งมีขนาดมากก็ยิ่งทำให้โอกาสทำให้โดนโจมตีถูกง่ายขึ้น

- Weight จะบอกน้ำหนักของส่วนนั้น ซึ่งถ้าน้ำหนักรวมของทุกส่วนยิ่งมากก็จะทำให้ robot เคลื่อนที่ได้ช้าลงไปด้วย

- Speed (จะมีในส่วน leg ส่วนเดียว) ซึ่งใช้ในการเคลื่อนที่ โดย speed จะขึ้นกับน้ำหนักรวมของหุ่นยนต์ ด้วย เช่น ถ้า speed เท่ากันแต่น้ำหนักรวมของ robot ต่างกันก็จะทำให้ ความเร็วในการเคลื่อนที่ต่างกัน

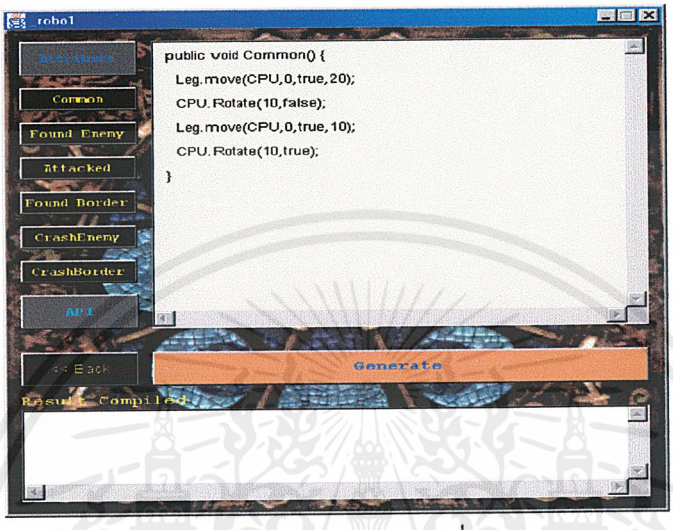
Jet

- Speed ความเร็วของการใช้ jet น้ำหนักของ robot จะไม่ผลต่อความเร็วของ jet
- Oil จำนวนน้ำมันที่ใช้ได้ จะลดลงไปถ้ามีการเคลื่อนทีโดยใช้ jet
- Weight น้ำหนักของ jet

รูปที่ ข-4 เป็นส่วนของ Component of Weapon จะบอกรายละเอียดของ Weapon ต่างๆ ทั้ง 3 ประเภท ทั้งที่เป็น Long Weapon ที่เป็นอาวุธระยะไกล ที่ส่วนมากจะมีพลังการทำลายมากแต่จำนวนกระสุนจะน้อยและ Short Weapon ที่เป็นอาวุธระยะใกล้ ที่ส่วนมากจะมีพลังการทำลายต่ำแต่จำนวนกระสุนจะมาก

- Weight คือน้ำหนักของอาวุธชนิดนั้น

- Destroy Power คือพลังทำลายของอาวุธชนิดนั้น
- Distance คือระยะทางที่อาวุธนั้นสามารถยิงไปได้ไกลสุด
- Bullet คือจำนวนกระสุนของอาวุธชนิดนั้น
- Bullet Speed คือความเร็วของกระสุนของอาวุธชนิดนั้น
- Shooting Rate คืออัตราการยิงอาวุธชนิดนั้นว่าในการยิงครั้งต่อไปต้องใช้เวลาเท่าไร

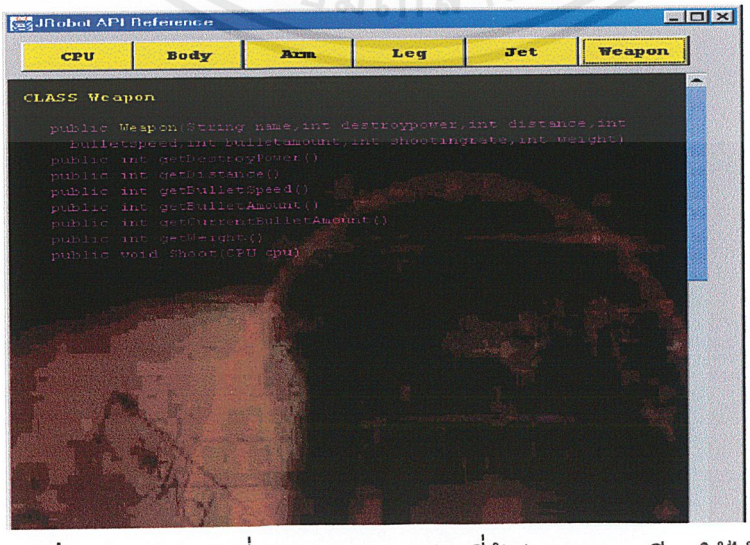


รูปที่ ข-5 แสดงหน้าสำหรับการเขียนคำสั่งควบคุมหุ่นยนต์

รูปที่ ข-5 เป็นส่วนของการใส่ Code ให้กับ robot เหมือนการใส่ความรู้ให้กับ robot โดยการใส่คำสั่งต่างๆในแต่ละเหตุการณ์ ถ้าเกิดเหตุการณ์นั้นจะให้ robot ทำอะไร

- ปุ่ม attribute ให้รู้ว่าผู้เล่นมีส่วนประกอบของหุ่นยนต์อะไรบ้างที่ถูกสร้างขึ้นมา โดยการ new instance ส่วนประกอบนั้นขึ้นมา ซึ่งจะไม่สามารถแก้ไขในส่วนนี้ได้

- ปุ่ม API จะแสดงให้เห็นว่าแต่ละส่วนของ robot มี methodใดที่สามารถเรียกใช้ได้ในรูปที่ ข-6 สำหรับรายละเอียดของ API ที่ผู้เล่นสามารถเรียกใช้ได้จะแสดงไว้ใน web และภาคผนวก ค.



รูปที่ ข-6 แสดงหน้าที่แสดง method ต่างที่ผู้เล่นสามารถเรียกใช้ได้

- ผู้เล่นจะสามารถ Override Event Code ของเหตุการณ์ต่างได้ เมื่อทำการคลิกเลือกเหตุการณ์ที่ต้องการที่จะ Override ก็จะมี method ของ event นั้นให้ทำการเขียน code เป็นภาษา java โดยสามารถใช้ method ของส่วนประกอบที่เลือกมาได้โดยดูจาก API ซึ่งมีเหตุการณ์ที่สามารถเลือกได้ดังต่อไปนี้

- Common เหตุการณ์ปกติ

- Found Enemy เหตุการณ์พบศัตรู

- Attacked เหตุการณ์โดนโจมตี

- Found Border เหตุการณ์พบขอบสนาม

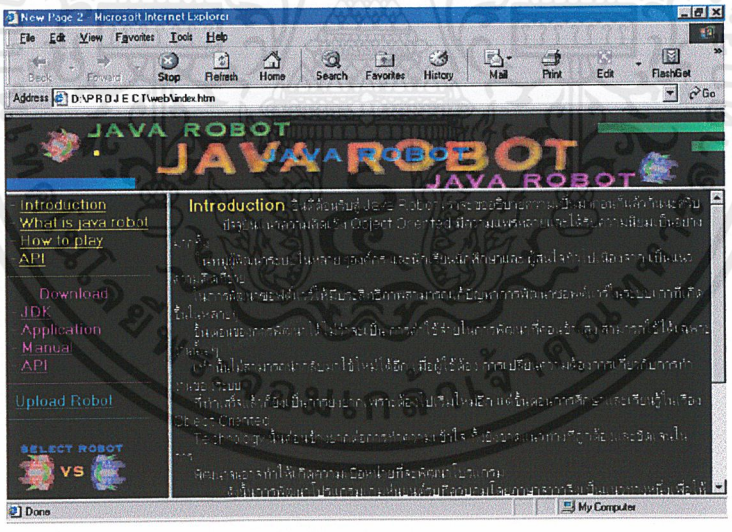
- Crash Enemy เหตุการณ์ชนศัตรู

- Crash Border เหตุการณ์ชนขอบสนาม

- กดปุ่ม Generate จะได้ code ของหุ่นยนต์ของผู้เล่นซึ่งเป็น fileชื่อ robot ที่เป็น.class ถ้าผู้เล่นเขียน Code ผิดหรือ Generate เสร็จจะแสดงผลใน result compiled

- ปุ่ม Back ทำให้เปลี่ยนหน้ากลับไปยังหน้าก่อนหน้านี้ ดังรูป ข-2

คู่มือการใช้ส่วนของ web



รูปที่ ข-7 แสดงหน้าแรกของ web

รูปที่ ข-7 เป็นหน้าแรกของ web นี้ประกอบด้วย Frame 3 Frame คือ Frameบนที่ใช้สำหรับใส่ title ของ java robot Frame ทางด้านซ้าย ที่เป็นหัวข้อต่างๆ ที่สามารถคลิกเลือกได้ซึ่งจะแสดงรายละเอียดของหัวข้อของที่ Frame ทางด้านซ้าย และ Frame ทางด้านขวาใช้แสดงรายละเอียดต่างๆ

Frame ทางด้านซ้าย จะประกอบไปด้วย 4 ส่วนคือ

1. หัวข้อต่างๆที่ใช้อธิบายรายละเอียดต่างๆของ java robot

- Introduction จะอธิบายที่มาของ java robot

- What is java robot จะอธิบายรายละเอียดและลักษณะของ หุ่นยนต์ และ สนามรบ

- How to play จะอธิบายวิธีการเล่นเกม และ กฎ กติกาต่างๆ

- API ใช้อธิบาย method ต่างๆของแต่ละส่วนประกอบต่างๆของหุ่นยนต์ที่สามารถใช้ได้ในการเล่นเกม

2. เป็นส่วนที่ให้ Downlond โปรแกรมที่จำเป็นในการเล่น

- JDK ผู้เล่นจะต้องดาวน์โหลดเพื่อใช้ประกอบการเขียนโปรแกรมซึ่งใช้ภาษาจาวาและใช้ในการแสดงของการต่อสู้ด้วยโดยจะต้องมี JDK version 1.2 ขึ้นไป

- Application ผู้เล่นที่ต้องการสร้างหุ่นยนต์ขึ้นเองจะต้องดาวน์โหลด Application สำหรับสร้างหุ่นยนต์เพื่อใช้ประกอบหุ่นยนต์และเขียนคำสั่งควบคุมกาทำงาน แต่สำหรับผู้ที่ไม่ต้องการสร้างเองก็สามารถเลือกหุ่นยนต์ที่มีอยู่ได้

- Munual เป็นคู่มือแสดงรายละเอียดการทำงานทั้งหมดของ Java Robot

- API เป็นเอกสารแสดง method ต่าง ๆ พร้อมทั้งรายละเอียด ที่ผู้เล่นสามารถเรียกใช้ได้ในการสั่งให้หุ่นยนต์ทำงาน

3. Upload เป็นส่วนสำหรับ upload โปรแกรมของหุ่นยนต์ที่สร้างขึ้น เพื่อนำเข้าไปไว้ในserver สำหรับเลือกเพื่อสต่อสู้

4. SELECT ROBOT สำหรับเลือกหุ่นยนต์ที่ต้องการต่อสู้และเริ่มต้นการต่อสู้พร้อมทั้งแสดงผล ของการต่อสู้ด้วย

ตัวอย่างของหุ่นยนต์ และคำอธิบายหุ่นยนต์

ใน web จะมีตัวอย่างของ Robot ที่ได้ทำการ upload ไว้ให้เป็นตัวอย่าง เพื่อให้เข้าใจการสร้างและ การเขียนคำสั่งควบคุม Robot มากขึ้น จะมีตัวอย่างดังนี้

_Empty คือ Robot ที่ไม่ได้ใส่คำสั่งใดเลยให้กับมันทำให้มันไม่สามารถทำอะไรได้เลย

_test1 และ _test2 คือ Robot ที่มีการสร้างขึ้นมาให้เห็นถึงข้อแตกต่างกันของการเลือกส่วนประกอบที่มีน้ำหนักมากกับน้ำหนักน้อย โดย _test1 จะเป็น robot ที่มีน้ำหนักเบาโดยส่วนประกอบต่างๆ จะเลือกใช้ type1 เป็นส่วนใหญ่เพื่อให้ได้ robot มีน้ำหนักเบา และ _test2 จะเป็น robot ที่มีน้ำหนักมากโดยส่วนประกอบต่างๆจะเลือกใช้ type3 เป็นส่วนใหญ่ทำให้ robot มีน้ำหนักมาก ในการเขียนคำสั่งในการควบคุม robot จะมีการเขียนคำสั่งในเหตุการณ์เดียวคือ Common ของ robot ทั้งสองตัวที่เหมือนกันดังนี้

```
public void Common() {
  Leg.move(CPU,90,true,100);
  ShortWeaponLeft.Shoot(CPU);
  ShortWeaponRight.Shoot(CPU);
  LongWeaponLeft.Shoot(CPU);
  LongWeaponRight.Shoot(CPU);
  Leg.move(CPU,90,false,100);
 }
```

จากคำสั่งจะเห็นได้ว่าคำสั่งแรกจะให้ robot เคลื่อนที่ไปทางซ้าย 90 องศา เป็นระยะทาง 100 M จากนั้นจะทำการยิงโดยใช้อาวุธระยะใกล้ทางซ้าย อาวุธระยะใกล้ทางขวา อาวุธระยะไกลทางซ้าย และอาวุธระยะไกลทางขวาตามลำดับ และทำการเคลื่อนที่ไปทางขวา 90 องศา เป็นระยะทาง 100 M เป็นคำสั่งสุดท้าย

เมื่อทำการนำ _test1 และ _test2 มาต่อสู้กันจะเห็นได้ว่า _test1 จะมีพลังในแต่ละส่วนน้อยกว่า _test2 แต่จะมีความเร็วมากกว่าเนื่องจากมีน้ำหนักน้อยกว่า

_test3 และ _test4 มีการเปรียบเทียบให้เห็นการยิงระหว่างอาวุธระยะใกล้กับอาวุธระยะไกล โดยในการเขียนคำสั่งในการควบคุม robot จะมีการเขียนคำสั่งในเหตุการณ์เดียวคือ Common ของ_test3 และ _test4ดังนี้

```
public void Common() { //เป็นคำสั่งของ _test3 จะเลือกอาวุธระยะใกล้มาใช้เท่านั้น
  ShortWeaponLeft.Shoot(CPU);
  ShortWeaponRight.Shoot(CPU);
 }
```

จากคำสั่งจะสั่งให้ _test3 ใช้อาวุธระยะใกล้ทางซ้ายก่อนแล้วใช้ทางขวา

```
public void Common() { //เป็นคำสั่งของ _test4 จะเลือกอาวุธระยะไกลมาใช้เท่านั้น
  ShortWeaponLeft.Shoot(CPU);
  ShortWeaponRight.Shoot(CPU);
 }
```

จากคำสั่งจะสั่งให้ _test4 ใช้อาวุธระยะไกลทางซ้ายก่อนแล้วใช้ทางขวา

เมื่อทำการนำ _test3 และ _test4 มาต่อสู้กันจะเห็นได้ว่า _test3 จะใช้อาวุธใกล้ซึ่งจะมีจำนวนกระสุนมากแต่ยิงไปได้ไม่ไกล ส่วน _test4 จะใช้อาวุธไกลซึ่งจะมีจำนวนกระสุนน้อยแต่ยิงไปได้ไกลและจะเห็นความแตกต่างของความเร็วของกระสุนและอัตราการยิงของอาวุธแต่ชนิดด้วย ซึ่งจะเป็นไปตามความสามารถของอาวุธชนิดนั้นๆ

ส่วนตัวอย่าง robot ตั้งแต่ _test5 ถึง _test9 จะแสดงให้เห็นการเกิดเหตุการณ์ต่างของ robot ดังนี้

_test5 จะแสดงเหตุการณ์พบศตรูโดยจะมีการเขียนคำสั่งใน 2 เหตุการณ์คือ Common และ FoundEnemy ดังนี้

```
public void Common() {
 CPU.Rotate(30,false);
 CPU.Rotate(30,true);
 Leg.move(CPU,0,true,50);
}
public void FoundEnemy() {
 CPU.CommonStop();
 ShortWeaponLeft.Shoot(CPU);
 ShortWeaponRight.Shoot(CPU);
 LongWeaponLeft.Shoot(CPU);
 LongWeaponRight.Shoot(CPU);
 }
```

ขอแนะนำให้ใช้ _test5 กับ Empty ในการต่อสู้ ตอนเริ่มต้น เหตุการณ์ Common จะทำการทำงานก่อนโดยทำการหมุนไปทางซ้าย 30 องศา หมุนไปทางขวา 30 องศา และทำการเคลื่อนที่ไปข้างหน้า 50 M ตามลำดับไปเรื่อยๆจนกระทั้ง scan พบศัตรู เหตุการณ์ FoundEnemy ก็จะเริ่มทำงาน เนื่องจากคำสั่งแรกเราทำการหยุดเหตุการณ์ Common ดังนั้น robot จะหยุดการทำงานของเหตุการณ์ Common และไม่สามารถกลับไปทำงาน ในเหตุการณ์ Common ได้อีกถ้าไม่มีการส่ง start อีกครั้ง หลังจากนั้น _test5 จะทำการยิงโดยใช้อาวุธระยะใกล้ทางซ้าย อาวุธระยะใกล้ทางขวา อาวุธระยะไกลทางซ้าย และ อาวุธระยะไกลทางขวาตามลำดับไปเรื่อยๆ แต่ถ้าเรามีการสั่ง start ในเหตุการณ์ Common คำสั่งสุดท้ายในเหตุการณ์ FoundEnemy แต่ _test5 ยัง scan เจอศัตรูอยู่ จะเห็นได้ว่า robot จะกลับมาทำคำสั่งในเหตุการณ์ Common อยู่นิดหน่อย และจึงจะกลับไปทำคำสั่งในเหตุการณ์ FoundEnemy อีก

_test6 จะแสดงเหตุการณ์โดนโจมตี โดยจะมีการเขียนคำสั่งใน 2 เหตุการณ์คือ Common และ Attacked ดังนี้

```
public void Common() {
  Leg.move(CPU,0,false,20);
 }
```

```
public void Attacked() {
  CPU.CommonStop();
  Jet.move(CPU,180,false,100);
  CPU.CommonStart();
}
```

ขอแนะนำให้ใช้ _test6 กับ _test5 ในการต่อสู้ ตอนเริ่มต้น เหตุการณ์ Common จะทำการทำงานก่อนโดยทำการเคลื่อนที่ไปข้างหน้า 20 M เรื่อยๆ จนกระทั้งโดน _test5 โจมตี เมื่อ _test6 โดนโจมตีเหตุการณ์ Attacked ก็จะเริ่มทำงานเนื่องจากคำสั่งแรกเราทำการหยุดเหตุการณ์ Common ดังนั้น robot จะหยุดการทำงานของเหตุการณ์ Common และทำการใช้ jet เคลื่อนที่ไป 180 องศาทางซ้าย 100 M หรือคือการเคลื่อนที่ไปข้างหลังนั้นเอง และทำการสั่ง start เหตุการณ์ Common จะเห็นได้ว่า robot จะกลับมาทำคำสั่งในเหตุการณ์ Common อีกครั้ง และเมื่อถูกโจมตีอีกก็จะกลับไปทำเหตุการณ์ Attacked อีกวนไปเรื่อยๆ

_test7 จะแสดงเหตุการณ์ scan เจอขอบสนาม โดยจะมีการเขียนคำสั่งใน 2 เหตุการณ์คือ Common และ FoundBorder ดังนี้

```
public void Common() {
  Leg.move(CPU,45,true,30);
}
public void FoundBorder() {
   CPU.CommonStop();
   if(CPU.getFoundBorderDirection()==0)
     CPU.Rotate(180,true);
   else if(CPU.getFoundBorderDirection()==1)
     CPU.Rotate(150,true);
   else if(CPU.getFoundBorderDirection()==2)
     CPU.Rotate(150,false);
   CPU.CommonStart();
}
```

ขอแนะนำให้ใช้ _test7 กับ _test7 ในการต่อสู้ ตอนเริ่มต้น เหตุการณ์ Common จะทำการทำงานก่อนโดยทำการเคลื่อนที่ไป 40 องศาทางขวา 30 M เรื่อยๆ จนกระทั้ง _test7 scan เจอขอบสนาม เหตุการณ์ FoundBorder ก็จะเริ่มทำงานเนื่องจากคำสั่งแรกเราทำการหยุดเหตุการณ์ Common ดังนั้น robot จะหยุดการทำงานของเหตุการณ์ Common และคำสั่งต่อมาจะมีการใช้คำสั่ง if ....

Else ในภาษาจาวา ส่วนคำสั่ง CPU.getFoundBorderDirection() ซึ่งเป็นหนึ่งใน method ที่มีให้ใช้ใน cpu และยังมี method อื่นๆ ให้ใช้ได้อีกโดยดูได้จาก API แต่ method นี้จะเป็นการ get ค่าของ scan ว่าscan เจอด้านไหนของ scan 0 จะเป็นตรงกลาง, 1 ทางซ้าย และ 2 ทางขวา จากคำสั่งจะเห็นได้ว่าถ้า getค่าได้ 0 จะทำการหมุนไป 180 องศาทางขวา, 1 จะทำการหมุนไป 150 องศาทางขวา, 2 จะทำการหมุนไป 150 องศาทางซ้าย จากนั้นจะทำการสั่ง start เหตุการณ์ Common จะเห็นได้ว่า robot จะกลับมาทำคำสั่งในเหตุการณ์ Common อีกครั้งและเมื่อ scan เจอขอบสนามอีกก็จะกลับไปทำเหตุการณ์ FoundBorder อีกวนไปเรื่อยๆ

_test8 จะแสดงเหตุการณ์ชนศตรู โดยจะมีการเขียนคำสั่งใน 2 เหตุการณ์คือ Common และ CrashEnemy ดังนี้

```
public void Common() {
  Leg.move(CPU,0,false,30);
}
public void CrashEnemy() {
  CPU.CommonStop();
  Jet.move(CPU,180,false,100);
  CPU.CommonStart();
}
```

ขอแนะนำให้ใช้ _test8 กับ _test8 ในการต่อสู้ ตอนเริ่มต้น เหตุการณ์ Common จะทำการทำงานก่อนโดยทำการเคลื่อนที่ไปข้างหน้า 30 M เรื่อยๆจนกระทั้งโดน _test8 ชนกับศตรู เหตุการณ์ CrashEnemy ก็จะเริ่มทำงานเนื่องจากคำสั่งแรกเราทำการหยุดเหตุการณ์ Common ดังนั้น robot จะหยุดการทำงานของเหตุการณ์ Common และทำการใช้ jet เคลื่อนที่ไป 180 องศา ทางซ้าย 100 M หรือคือการเคลื่อนที่ไปข้างหลังนั้นเอง และทำการสั่ง start เหตุการณ์ Common จะเห็นได้ว่า robot จะกลับมาทำคำสั่งใน เหตุการณ์ Common อีกครั้ง และเมื่อชนกับศตรูอีกก็จะกลับไปทำเหตุการณ์ CrashEnemy อีกวนไปเรื่อยๆ

_test9 จะแสดงเหตุการณ์ ชนขอบสนาม โดยจะมีการเขียนคำสั่งใน 2 เหตุการณ์คือ Common และ CrashBorder ดังนี้

```
public void Common() {
  Leg.move(CPU,180,false,20);
}
public void CrashBorder() {
  CPU.CommonStop();
```

```
        Jet.move(CPU,0,true,100);
        CPU.CommonStart();
    }
```

ขอแนะนำให้ใช้ _test9 กับ _test9 ในการต่อสู้ ตอนเริ่มต้น เหตุการณ์ Common จะทำการทำงานก่อนโดยทำการเคลื่อนที่ไปข้างหลัง 20 M เรื่อยๆจนกระทั้งโดน _test9 ชนกับขอบสนาม เหตุการณ์ CrashBorder ก็จะเริ่มทำงานเนื่องจากคำสั่งแรกเราทำการหยุดเหตุการณ์ Common ดังนั้น robot จะหยุดการทำงานของเหตุการณ์ Common และทำการใช้ jet เคลื่อนที่ไป ไปข้างหน้า100 M และทำการสั่ง start เหตุการณ์ Common จะเห็นได้ว่า robot จะกลับมาทำคำสั่งในเหตุการณ์ Common อีกครั้งและเมื่อชนขอบสนามอีกก็จะกลับไปทำเหตุการณ์ CrashEnemy อีกวนไปเรื่อยๆ

ดังนั้นผู้เล่นจึงควรเขียนคำสั่งต่างๆให้เหมาะกับเหตุการณ์ต่างๆ และพยายามควบคุม robot ไม่ให้ชนสนามหรือชนศัตรู เพราะจะทำให้เสียพลังไปโดยเปล่าประโยชน์

ภาคผนวก ค.
เมธอทสำหรับสั่งให้หุ่นยนต์ทำงาน (API specification)

class CPU

constructor

public *CPU*(int scandistance,int scandegree,int nearability,int farability)

สร้าง Object ของ CPU โดยมีการกำหนดค่าต่างๆ เป็นพารามิเตอร์คือ scandistance ซึ่งเป็นระยะความยาวของพื้นที่ Scan ที่เป็นรูปสามเหลี่ยม, scandegree คือความกว้างเป็นองศาของมุม Scan ซึ่งเป็นรูปสามเหลี่ยมโดยยึดจากตัวหุ่นยนต์เป็นหลัก, near ability คือเปอร์เซ็นต์ความสามารถในการยิงระยะใกล้ และ farability ซึ่งเป็นเปอร์เซ็นต์ความสามารถในการยิงระยะไกล

ตารางที่ ค-1 แสดง public method ของคลาส CPU

Method	Detail
int *getFaceDegree*()	คืนค่าทิศทางองศาปัจจุบันของหน้าหุ่นยนต์
int *getScanDistance*()	คืนค่าระยะความยาวของพื้นที่ Scan ซึ่งเป็นรูปสามเหลี่ยม
int *getScanDegree*()	คืนค่าความกว้างเป็นองศาของมุม Scan ซึ่งเป็นรูปสามเหลี่ยมโดยยึดจากตัวหุ่นยนต์เป็นหลัก
int *getScanSpeed*()	คืนค่าความเร็วในการหมุนของหุ่นยนต์ในการ Scan
int *getNearAbility*()	คืนค่าเปอร์เซ็นต์ความสามารถในการยิงระยะใกล้
int *getFarAbility*()	คืนค่าเปอร์เซ็นต์ความสามารถในการยิงระยะไกล
void *Rotate*(int degree,boolean direction)	สั่งให้หุ่นยนต์หมุนโดยระบุองศาและทิศทางที่ต้องการ
void *CommonStop*()	สั่งให้การทำงานในเหตุการณ์ปกติหยุด
void *FoundEnemyStop*()	สั่งให้การทำงานในเหตุการณ์พบศัตรูหยุด
void *AttackedStop*()	สั่งให้การทำงานในเหตุการณ์พบศัตรูหยุด
void *FoundBorderStop*()	สั่งให้การทำงานในเหตุการณ์พบขอบสนามหยุด
void *CrashEnemyStop*()	สั่งให้การทำงานในเหตุการณ์ชนศัตรูหยุด
void *CrashBorderStop*()	สั่งให้การทำงานในเหตุการณ์ชนขอบสนามหยุด
void *CommonStart*()	สั่งให้การทำงานในเหตุการณ์ปกติทำงาน
void *FoundEnemyStart*()	สั่งให้การทำงานในเหตุการณ์พบศัตรูทำงาน
void *AttackedStart*()	สั่งให้การทำงานในเหตุการณ์โดนยิงทำงาน
void *FoundBorderStart*()	สั่งให้การทำงานในเหตุการณ์พบขอบสนามทำงาน
void *CrashEnemyStart*()	สั่งให้การทำงานในเหตุการณ์ชนศัตรูทำงาน

ตารางที่ ค-1 (ต่อ) แสดง public method ของคลาส CPU

Method	Detail
void *CrashBorderStart*()	สั่งให้การทำงานในเหตุการณ์ชนขอบสนามทำงาน
boolean *getCommonStop*()	คืนค่าเป็นจริงถ้าในขณะนั้นเหตุการณ์ปกติหยุด ถ้าไม่จะคืนค่าเท็จ
boolean *getFoundEnemyStop*()	คืนค่าเป็นจริงถ้าในขณะนั้นเหตุการณ์พบศัตรูหยุด ถ้าไม่จะคืนค่าเท็จ
boolean *getAttackedStop*()	คืนค่าเป็นจริงถ้าในขณะนั้นเหตุการณ์โดนยิงหยุด ถ้าไม่จะคืนค่าเท็จ
boolean *getFoundBorderStop*()	คืนค่าเป็นจริงถ้าในขณะนั้นเหตุการณ์พบขอบสนามหยุด ถ้าไม่จะคืนค่าเท็จ
boolean *getCrashEnemyStop*()	คืนค่าเป็นจริงถ้าในขณะนั้นเหตุการณ์ชนศัตรูหยุด ถ้าไม่จะคืนค่าเท็จ
boolean *getCrashBorderStop*()	คืนค่าเป็นจริงถ้าในขณะนั้นเหตุการณ์ชนขอบสนามหยุด ถ้าไม่จะคืนค่าเท็จ

ถ้าเกิดเหตุการณ์ Scan พบศัตรู สามารถใช้ public method ต่อไปนี้ได้

ตารางที่ ค-2 แสดง public method ของคลาส CPU ถ้าเกิดเหตุการณ์ Scan พบศัตรู (FoundEnemy)

Method	Detail
int *getEnemyDegree*(boolean right)	คืนค่าองศาของตำแหน่งศัตรูโดยเทียบกับองศาของหุ่นยนต์ โดยคืนค่าองศาทางด้านขวาถ้าพารามิเตอร์ที่ส่งเป็นจริง ถ้าเป็นเท็จจะคืนองศาทางซ้าย
int *getEnemyEnergy*(int part)	คืนค่าพลังงานของส่วนต่างๆของศัตรูโดยที่ถ้า part=0 คือ body ,part=1 คือ left arm ,part=2 คือ right arm ,part=3 คือ leg
int *getEnemyWeight*(int part)	คืนค่าน้ำหนักของส่วนต่างๆของศัตรูโดยที่ถ้า part=0 คือ body ,part=1 คือ left arm ,part=2 คือ right arm ,part=3 คือ leg , part=4 คือ jet
int *getEnemySize*(int part)	คืนค่าขนาดของส่วนต่างๆของศัตรูโดยที่ถ้า part=0 คือ body ,part=1 คือ left arm ,part=2 คือ right arm ,part=3 คือ leg

ตารางที่ ค-2 (ต่อ) แสดง public method ของคลาส CPU ถ้าเกิดเหตุการณ์ Scan พบศัตรู (FoundEnemy)

Method	Detail
int *getEnemySpeed*(int part)	คืนค่าความเร็วในการเคลื่อนที่ของศัตรูโดยที่ถ้า part=3 คือความเร็วของleg , part=4 คือ ความเร็วของ jet
int *getEnemyJetOil*()	คืนค่าปริมาณน้ำมันที่มีอยู่ในขณะนั้นของศัตรู
int *getEnemyScanDistance*()	คืนค่าระยะความยาวของการ Scan ของศัตรู
int *getEnemyScanDegree*()	คืนค่าความกว้างของพื้นที่ Scan ของศัตรู
int *getEnemyNearAbility*()	คืนค่าเปอร์เซ็นต์ความสามารถในการยิงระยะใกล้ของศัตรู
int *getEnemyFarAbility*()	คืนค่าเปอร์เซ็นต์ความสามารถในการยิงระยะไกลของศัตรู
int *getEnemyFaceDegree*()	คืนค่าทิศทางองศาปัจจุบันของหน้าศัตรู
int *getEnemyWeaponDestroyPower*(int part)	คืนค่าพลังการทำลายของอาวุธคู่ต่อสู้โดยถ้า part=0 คือ left short gun ,part=1 คือ right short gun .part=2 คือ left long gun ,part=3 คือ right long gun
int *getEnemyWeaponDistance*(int part)	คืนค่าระยะทางการยิงสูงสุดของอาวุธคู่ต่อสู้โดยถ้า part=0 คือ left short gun ,part=1 คือ right short gun .part=2 คือ left long gun ,part=3 คือ right long gun
int *getEnemyWeaponBulletSpeed*(int part)	คืนค่าความเร็วลูกกระสุนของอาวุธคู่ต่อสู้โดยถ้า part=0 คือ left short gun ,part=1 คือ right short gun .part=2 คือ left long gun ,part=3 คือ right long gun
int *getEnemyWeaponCurrentBulletAmount*(int part)	คืนค่าจำนวนลูกกระสุนที่มีอยู่ในขณะนั้นของอาวุธคู่ต่อสู้ โดยถ้า part=0 คือ left short gun ,part=1 คือ right short gun .part=2 คือ left long gun ,part=3 คือ right long gun

ถ้าเกิดเหตุการณ์ Scan พบขอบสนาม สามารถใช้ public method ต่อไปนี้ได้

ตารางที่ ค-3 แสดง public method ของคลาส CPU ถ้าเกิดเหตุการณ์ Scan พบขอบสนาม (FoundBorder)

Method	Detail
int *getBorder*()	คืนค่าขอบสนามที่พบโดยที่ถ้าค่าเป็น 0 =ขอบด้านบน ,1 = ขอบด้านขวา ,2 =ขอบด้านล่าง ,3=ขอบด้านซ้าย .4=มุมขอบด้านซ้ายบน ,5=มุมขอบด้านขวาบน ,6=มุมขอบด้านล่าง ,7=มุมขอบด้านซ้ายล่าง

ตารางที่ ค-3 (ต่อ) แสดง public method ของคลาส CPU ถ้าเกิดเหตุการณ์ Scan พบขอบสนาม (FoundBorder)

Method	Detail
int *getFoundBorderDirection*()	คืนค่าทิศทางที่พบขอบสนามโดยถ้าค่าเป็น 0 =พบในทิศหน้าตรงของหุ่นยนต์,1=พบด้านซ้ายของหุ่นยนต์ ,2=พบด้านขวา
int *getBoderDistance*()	คืนค่าระยะห่างจากขอบสนามที่พบ

ถ้าเกิดเหตุการณ์โดนยิง สามารถใช้ public method ต่อไปนี้ได้

ตารางที่ ค-4 แสดง public method ของคลาส CPU ถ้าเกิดเหตุการณ์โดนยิง (Attacked)

Method	Detail
Int *getBulletDegree*(boolean Right)	คืนค่าองศาของตำแหน่งที่โดนยิงโดยเทียบกับทิศของหุ่นยนต์

class BodyHead

constructor

public *BodyHead*(int weight,int size,int energy)

สร้าง object ของ BodyHead ด้วยน้ำหนัก ขนาดและพลังงานที่เป็นพารามิเตอร์

ตารางที่ ค-5 แสดง public method ของคลาส BodyHead

Method	Detail
int *getWeight*()	คืนค่าน้ำหนักของ Body
int *getSize*()	คืนค่าขนาดของ Body
int *getEnergy*()	คืนค่าพลังงานของ Body

class Arm

constructor

public *Arm*(int weight,int size,int energy)

สร้าง object ของ Arm ด้วยน้ำหนัก ขนาดและพลังงานที่เป็นพารามิเตอร์

ตารางที่ ค-6 แสดง public method ของคลาส Arm

Method	Detail
int *getWeight*()	คืนค่าน้ำหนักของแขน
int *getSize*()	คืนค่าขนาดของแขน
int *getEnergy*()	คืนค่าพลังงานของแขน

class Leg

constructor

public *Leg*(int weight,int size,int energy,int speed)

สร้าง object ของ Leg ด้วยน้ำหนัก ขนาด พลังงานและความเร็วที่เป็นพารามิเตอร์

ตารางที่ ค-7 แสดง public method ของคลาส Leg

Method	Detail
int *getWeight*()	คืนค่าน้ำหนักของขา
int *getSize*()	คืนค่าขนาดของขา
int *getEnergy*()	คืนค่าพลังงานของขา
int *getSpeed*()	คืนค่าความเร็วในการเคลื่อนที่ของขา
void *move*(CPU cpu,int degree,boolean direction,int distance)	สั่งให้หุ่นยนต์เคลื่อนที่ด้วยองศา ทิศทาง และระยะทางที่เป็นพารามิเตอร์ โดยองศาเป็นองศาที่เทียบกับทิศของหุ่นยนต์ และต้องส่ง object cpu ไปด้วย

class Jet

constructor

public *Jet*(int weight,int speed,int oil)

สร้าง object ของ Jet ด้วยน้ำหนัก ความเร็วและปริมาณน้ำมันที่เป็นพารามิเตอร์

ตารางที่ ค-8 แสดง public method ของคลาส Jet

Method	Detail
int *getWeight*()	คืนค่าน้ำหนักของ
int *getSpeed*()	คืนค่าความเร็วในการเคลื่อนที่ของไอพ่น
int *getOil*()	คืนค่าปริมาณน้ำมันของไอพ่น
void *move*(CPU cpu,int degree,boolean direction,int distance)	สั่งให้หุ่นยนต์เคลื่อนที่ด้วยองศาซึ่งเทียบกับหน้าหุ่นยนต์ ทิศทางซ้ายหรือขวา ถ้าเป็นเท็จจะเป็นทางซ้าย ถ้าเป็นจริง

ตารางที่ ค-8 (ต่อ) แสดง public method ของคลาส Jet

Method	Detail
	จะเป็นด้านขวา และระยะทาง นอกจากนี้ต้องส่ง object cpu ไปเป็นพารามิเตอร์ด้วย

class Weapon

constructor

public Weapon(String name,int destroypower,int distance,int bulletspeed,int bulletamount,int shootingrate,int weight)

สร้าง object Weapon ด้วยชื่อ พลังการทำลาย ระยะทางการยิงสูงสุด ความเร็วของลูกกระสุน จำนวนลูกกระสุน อัตราการยิง และน้ำหนักที่เป็นพารามิเตอร์

ตารางที่ ค-9 แสดง public method ของคลาส Weapon

Method	Detail
int *getDestroyPower*()	ค่าพลังทำลายของอาวุธ
int *getDistance*()	ค่าระยะทางการยิงของอาวุธ
int *getBulletSpeed*()	ค่าความเร็วของกระสุน
int *getBulletAmount*()	ค่าจำนวนกระสุนเริ่มต้น
int *getCurrentBulletAmount*()	ค่าจำนวนกระสุนในขณะนั้น
int *getWeight*()	ค่าน้ำหนักของอาวุธ
void *Shoot*(CPU cpu)	สั่งยิงโดยต้องส่ง object cpu เป็นพารามิเตอร์เข้าไปด้วย

บรรณานุกรม

ดร.วีระศักดิ์ ซึงถาวร, Java Programming Volume I,2000

ดร.วีระศักดิ์ ซึงถาวร, Fundamental of JAVA Programming Volume 2,2000

กิตติ ภักดีวัฒนะกุล, Java **ฉบับโปรแกรมเมอร์**, 2001

Jesus Castagnetto Harish Rawat Sascha Schumann Chris Scollo Deepak Veliath,Professional PHP Programming, 2000