การเรียนรู้ของเครื่อง 1

Machine Learning 1

กรณ์เทพ กตัญญูประจักษ์

ทรงเกิด รัตนทีปาจัย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2556

การเรียนรู้ของเครื่อง 1

Machine Learning 1

ธรณ์เทพ กรัญญประสิทธิ์

พงษ์เลิศ รัตนชินาลัย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2556

ปริญญานิพนธ์ปีการศึกษา 2556

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การเรียนรู้ของเครื่อง 1

MACHINE LEARNING 1

ผู้จัดทำ

1. นายธรณ์เทพ	กรัญญประสิทธิ์	รหัสนักศึกษา	53010702
2. นายพงษ์เลิศ	รัตนชินาลัย	รหัสนักศึกษา	53011040

.....อาจารย์ที่ปรึกษา

(รองศาสตราจารย์ ดร. บุญธีร์เครือตราชู)

การเรียนรู้ของเครื่อง

นาย ธรณ์เทพ กรัญญประสิทธิ์ 53010702
นาย พงษ์เลิศ รัตนชินาลัย 53011040
รศ.ดร. บุญธีร์ เครือตราชู อาจารย์ที่ปรึกษา
ปีการศึกษา2556

บทคัดย่อ

ในยุคปัจจุบันคอมพิวเตอร์มีความสามารถและประสิทธิภาพสูงขึ้นจากอดีตอย่างมากจึงได้มีการนำคอมพิวเตอร์มาประยุกต์ใช้ในการเรียนรู้และตัดสินใจแก้ปัญหาต่างๆ แทนมนุษย์โดยการเรียนรู้ของเครื่องเป็นการใช้วิธีการต่างๆเพื่อให้คอมพิวเตอร์สามารถเรียนรู้และแก้ปัญหาเองได้ซึ่งในที่นี้จะใช้เกมโรโบโค้ดเป็นแพลทฟอร์มสำหรับการศึกษาในเรื่องการเรียนรู้ของตัวหุ่นยนต์ในเกมโรโบโค้ดว่าสามารถมีการเรียนรู้ด้านการเล็งยิงหุ่นยนต์ศัตรูได้ถูกเป้าหมายมากเพียงใดโดยใช้ทฤษฎีเรื่องการเล็งยิงแบบคาดเดาและจดจำ(GuessFactor Targeting) เข้ามาใช้ในการศึกษาปัญหานี้ว่า ทฤษฎีการเล็งยิงแบบคาดเดาและจดจำ ทำให้หุ่นยนต์ในเกมโรโบโค้ด มีการเรียนรู้ด้านการเล็งยิงหุ่นยนต์ของศัตรูได้อย่างไร และมีประสิทธิภาพเพียงใด

MACHINE LEARNING 1

Mr. Thonthep	Karunyaprasith	53010702
Mr. Ponglert	Rattanachinalai	53011040
Assoc. Prof. Dr.Boontee	Kruatrachue	Advisor
Academic Year 2013		

ABSTRACT

Nowadays, PC has the high capability and performance. We have a computer application to learn and to solve many problems. Machine learning is used in various ways for learning and solving their own problems. In this case, we use Robocode game as a platform for studying machine learning about how the robot can learn to aim its opponent accurately. By usingGuessFactor Targeting to study about how this method can make the robot be able to learn its opponent's aiming efficiently.

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สามารถสำเร็จได้ด้วยดีจากความอนุเคราะห์ของคณะอาจารย์และครอบครัว ของผู้จัดทำขอขอบคุณรศ.ดร.บุญธีร์เครือตราชูอาจารย์ที่ปรึกษาโครงงานที่ให้คำแนะนำในการทำโครงงานรวมทั้งสอนทฤษฎีที่เกี่ยวข้อง, ให้ตัวอย่างข้อมูลที่สำคัญและเสนอแนวความคิดที่สามารถประยุกต์ใช้ในการทำโครงงานขอขอบคุณสาขาวิชาวิศวกรรมคอมพิวเตอร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ได้จัดเตรียมสิ่งอำนวยความสะดวกเพื่อให้การศึกษาโครงงานเป็นไปอย่างสะดวก

คณะผู้จัดทำจึงขอขอบพระคุณมา ณ ที่นี้ด้วย

นาย ธรณ์เทพ กรัญญประสิทธิ์

นาย พงษ์เลิศ รัตนชินาลัย

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

สารบัญรูป (ต่อ)

หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของโครงงาน

การเรียนรู้ของเครื่อง (Machine Learning) คือการที่คอมพิวเตอร์สามารถเรียนรู้ปัญหาตลอดจนสามารถแก้ปัญหานั้นได้อย่างชาญฉลาดโดยการเรียนรู้ของเครื่องสามารถทำได้หลายวิธี

โครงงานนี้เป็นโครงงานที่ศึกษาเกี่ยวกับการเรียนรู้ของหุ่นยนต์ ในเกมโรโบโค้ด ในด้านการเล็งยิงศัตรู ด้วยทฤษฎีการเล็งแบบคาดเดาและจดจำ(GuessFactor Targeting) ว่าหุ่นยนต์ที่ใช้ทฤษฎีการเล็งแบบนี้ สามารถที่จะเรียนรู้การเล็งยิงศัตรูให้โดนได้มากน้อย และมีประสิทธิภาพเพียงใด จึงมีประโยชน์และเหมาะแก่การศึกษาให้เข้าใจเพื่อสามารถนำไปประยุกต์ใช้ในชีวิตจริงได้ และเพื่อเรียนรู้วิธีที่จะทำให้เครื่องสามารถแก้ปัญหาได้ดีและมีประสิทธิภาพมากขึ้นต่อไป

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อให้เข้าใจหลักการแก้ปัญหาของทฤษฎีการเล็งแบบคาดเดาและจดจำ
2. เพื่อศึกษาการนำทฤษฎีการเล็งแบบคาดเดาและจดจำไปใช้งานในโรโบโค้ด
3. เพื่อให้เข้าใจการทำงานของระบบที่มีการเรียนรู้โดยใช้โรโบโค้ด
4. เพื่อให้สามารถนำการเรียนรู้ของเครื่องไปใช้แก้ปัญหาต่างๆในชีวิตจริง

1.3 ขอบเขตของโครงงาน

1. คอมพิวเตอร์สามารถเรียนรู้การทำงานของระบบเองได้
2. คอมพิวเตอร์สามารถหาวิธีการที่ดีเพียงพอในการทำงานของระบบได้เอง

1.4 วิธีการดำเนินการ

1. ศึกษาเกมโรโบโค้ด
2. ศึกษาทฤษฎีการเล็งแบบคาดเดาและจดจำ
3. ศึกษาเรื่องตัวประกอบการคาดเดา(GuessFactor)
4. ศึกษาเรื่องประวัติการเคลื่อนที่(Movement Profile)
5. ศึกษาเรื่องค่าการเคลื่อนที่เฉลี่ย (Rolling Averages)
6. ศึกษาเรื่องเซกเมนเตชั่น(Segmentation)

7. ศึกษาเรื่องกระบอกปืนจำลอง(Virtual Gun)และกระสุนจำลอง(Virtual Bullet)

8. ศึกษาเรื่องคลื่น(Wave)

9. เลือกใช้โปรแกรม Eclipseเป็น Editor สำหรับเขียนโปรแกรม

10.ทดลองสร้างหุ่นยนต์ที่ใช้ทฤษฎีการเล็งแบบคาดเดาและจดจำเบื้องต้น

11.ทดสอบหุ่นยนต์ที่ใช้ทฤษฎีการเล็งแบบคาดเดาและจดจำเบื้องต้นเพื่อเก็บข้อมูลต่างๆ

12. สร้างหุ่นยนต์ที่ใช้ทฤษฎีการเล็งแบบคาดเดาและจดจำที่มีความสามารถมากขึ้นกว่าเดิม

13. ทดสอบหุ่นยนต์ที่ใช้ทฤษฎีการเล็งแบบคาดเดาและจดจำรุ่นใหม่ที่สร้างขึ้นสำหรับเก็บข้อมูลการศึกษา

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.เข้าใจหลักแนวคิดของทฤษฎีการเล็งแบบคาดเดาและจดจำ

2.สามารถนำแนวคิดของทฤษฎีการเล็งแบบคาดเดาและจดจำไปใช้งานในโรโบโค้ดได้จริง

3. สามารถพัฒนาหุ่นยนต์ในโรโบโค้ดให้มีความสามารถมากกว่าเดิมได้โดยยังคงยึดตามแนวคิดของทฤษฎีการเล็งแบบคาดเดาและจดจำโดยปรับปรุงอัลกอริทึมให้ดีขึ้น

4.สามารถนำทฤษฎีการเล็งแบบคาดเดาและจดจำไปประยุกต์ใช้กับเรื่องอื่นๆในชีวิตประจำวันนอกเหนือจากโรโบโค้ดได้

1.6 ส่วนประกอบของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้ได้แบ่งเนื้อหาออกเป็น 5 บทด้วยกันคือ

บทที่ 1 บทนำ กล่าวถึงความสำคัญและที่มาของโครงงาน วัตถุประสงค์ของโครงงาน ขอบเขตของโครงงาน วิธีการดำเนินการ ประโยชน์ที่คาดว่าจะได้รับ และส่วนประกอบของปริญญานิพนธ์

บทที่ 2 ทฤษฎีที่เกี่ยวข้อง กล่าวถึงทฤษฎีพื้นฐานที่ใช้ในโครงงาน ประกอบด้วยอะไรบ้าง ให้บรรยายทฤษฎีทั้งหมดโดยละเอียด

บทที่ 3 สภาพแวดล้อมของโรโบโค้ด และการออกแบบวิธีการเล็งยิงของหุ่นยนต์ กล่าวถึงรายละเอียดของโครงงานนี้ ส่วนที่ได้ออกแบบและพัฒนาขึ้น การทำงานของระบบหรือชิ้นงาน บรรยายโดยละเอียด

บทที่ 4 การทดลองและผลการทดลอง กล่าวถึงการเตรียมการทดลองทั้งการจัดเตรียมฮาร์ดแวร์ ซอฟต์แวร์ สภาวะแวดล้อมในการทำการทดลอง ข้อมูลทดสอบ การทำงานหรือการจำลอง

การทำงานของระบบ ผลการทดลอง ค่าสมรรถนะของระบบ การวัดประสิทธิภาพของระบบ การวิเคราะห์ผลการทดลองหรือผลการทำงานทั้งหมด

บทที่ 5 บทสรุป กล่าวถึงบทสรุปของโครงงาน ปัญหาอุปสรรคต่างๆ ของโครงงาน และแผนงานในอนาคตของโครงงาน

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1รูปแบบการเล็งแบบคาดเดาและจดจำ

ทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ (GuessFactorTargeting) ที่จะนำไปใช้ในหุ่นยนต์ ของเกมโรโบโค้ด เพื่อทดสอบว่าทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ สามารถทำให้หุ่นยนต์เรียนรู้การเล็งยิงหุ่นยนต์ศัตรูได้เป็นทฤษฎีที่อาศัยส่วนประกอบหลายๆอย่าง ในการเรียนรู้โดยหุ่นยนต์จะทำการจับคลื่น(Wave) ที่ปล่อยออกมาจากหุ่นยนต์ศัตรู เมื่อศัตรูทำการยิงกระสุน เพื่อคำนวนหาระยะทางระหว่าง หุ่นยนต์เรา กับหุ่นยนต์ศัตรู แล้วทำการเก็บข้อมูลลงในเซกเมนเตชั่น (Segmentation) จากนั้นหุ่นยนต์ก็จะทำการรวมรวมข้อมูลด้านความเร็ว และทิศทางการเคลื่อนที่ของหุ่นยนต์ศัตรู เพื่อทำประวัติการเคลื่อนที่(Movement Profile) และค่าเฉลี่ยการเคลื่อนที่(Rolling Averages) จากข้อมูลที่เรดาร์ของหุ่นยนต์เราเก็บข้อมูลมาได้ แล้วทำการให้ค่าน้ำหนัก(weight) การเคลื่อนที่แต่ละแบบ เพื่อใช้ในการตัดสินใจต่อไป และจากนั้นจะทำการใช้ตัวประกอบการคาดเดา (GuessFactor) ในการคาดเดาเพื่อยิงศัตรู ว่าหากเรายิงไปในมุม เช่น 1.0 หรือ 0.75 ในขณะที่หุ่นยนต์กำลังไปในทิศทางนี้ ด้วยความเร็วเท่านี้ มุมไหนยิงโดนหุ่นยนต์ศัตรูมากน้อยเพียงใด ก็จะทำการให้คะแนน หรือปรับน้ำหนักการเล็งยิงของหุ่นยนต์ไปเองเรื่อยๆ จนได้ค่าที่ดีที่สุด สำหรับรูปแบบการเคลื่อนที่แบบนั้นของหุ่นยนต์ออกมาเพียงเท่านี้หุ่นยนต์ของเราก็จะเรียนรู้จากพฤติกรรมการเคลื่อนที่ ที่มีรูปแบบของหุ่นยนต์ของศัตรูได้

2.1.1 วิธีการทำงาน

แนวคิดรูปแบบการเล็งแบบคาดเดาและจดจำในช่วงแรกนั้นได้มากจากพอล อีแวนส์(Paul Evans)และเดวิด อัลเวส(David Alves)โดยพวกเขาได้มีการอธิบายหลักการไว้ว่า

- คำนวณว่าหุ่นยนต์ของศัตรูจะเคลื่อนที่ไปข้างหน้าได้ไกลเท่าไร ถ้าศัตรูนั้นเคลื่อนที่ด้วยความเร็วสูงสุดแล้วลูกกระสุนของหุ่นยนต์ของเรายิงไปโดนศัตรู เราจะเรียกจุดนั้นว่า จุด A
- คำนวณว่าหุ่นยนต์ของศัตรูจะเคลื่อนที่ไปข้างหลังได้ไกลเท่าไร ถ้าศัตรูนั้นเคลื่อนที่ด้วยความเร็วสูงสุดแล้วลูกกระสุนของหุ่นยนต์ของเรายิงไปโดนศัตรู เราจะเรียกจุดนั้นว่า จุด B

จากนั้นเราจะทำการคาดเดา โดยให้ค่าที่จะคาดเดาเป็นเลขระหว่าง 1 และ -1 ซึ่งจะเป็นจุดที่หุ่นยนต์ศัตรูของเราจะไป โดยที่

- ถ้าเป็น -1 จะหมายถึง หุ่นยนต์ของศัตรูกำลังเคลื่อนที่ไปข้างหลัง และหยุดที่จุด B

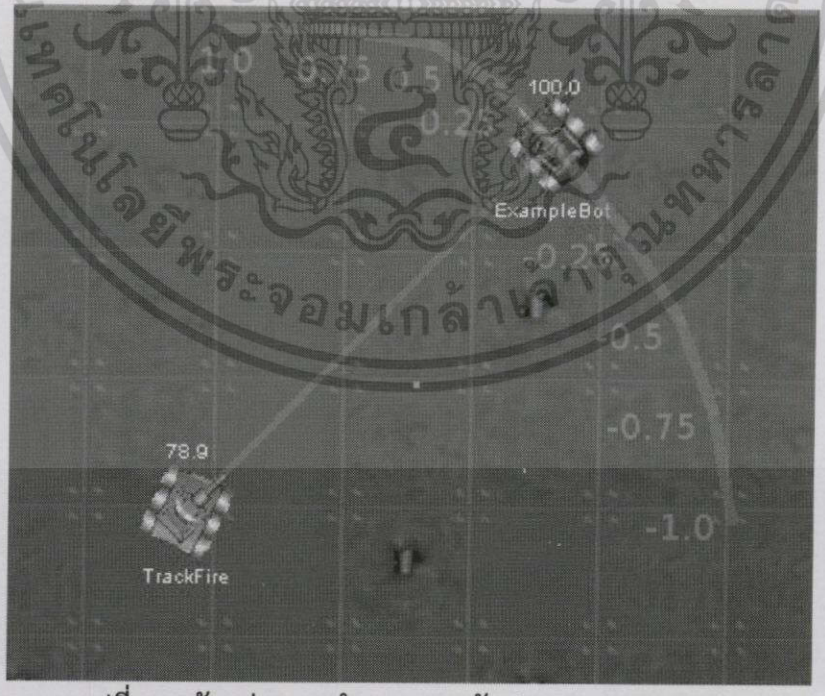
- ถ้าเป็น 1 จะหมายถึง หุ่นยนต์ของศัตรูกำลังเคลื่อนที่ไปข้างหน้า และหยุดที่ จุด A
- ถ้าเป็น 0 จะหมายถึง หุ่นยนต์ของศัตรูหยุดอยู่กับที่

หลังจากนั้นก็จะเป็นการติดตามผลว่าเลขไหนมีอัตราการยิงโดนมากที่สุด โดยถ้าเลขไหนมีอัตราการยิงโดนมากที่สุด เราก็จะให้หุ่นยนต์ของเรายิงไปยังทิศทางนั้นในอัตราที่สูงกว่าทิศทางอื่น

2.2 ส่วนประกอบของรูปแบบการเล็งแบบคาดเดาและจดจำ

2.2.1 ตัวประกอบการคาดเดา (GuessFactor)

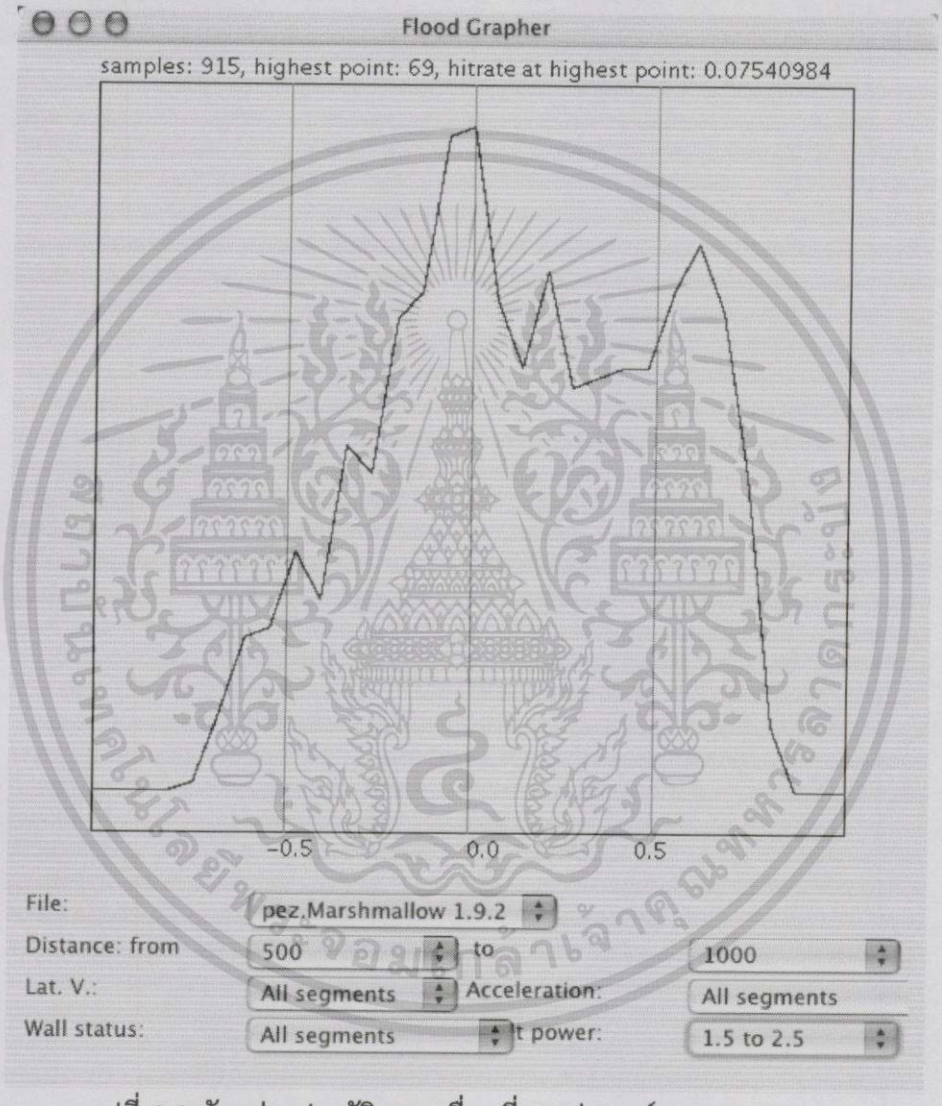
ค่าทิศทางที่จะทำการยิงไป โดยที่ถ้าเรายิงตรงไปในทิศทางที่เราเห็นศัตรู แล้วกระสุนของหุ่นยนต์เราโดนหุ่นยนต์ศัตรูเลย ค่าตัวประกอบการคาดเดา(GuessFactor)หรือเรียกสั้นๆว่าGF จะมีค่าเท่ากับ 0.0 ถ้าเรายิงล่วงหน้าไปยังทิศทาง และเวลาที่เป็นไปได้ ข้างหน้าศัตรู และกระสุนของหุ่นยนต์เราโดน หุ่นยนต์ศัตรูเลยก็จะมีค่าGF เป็น 1.0 และเช่นเดียวกันในทิศทางข้างหลังนั้นก็จะมีค่า GF เป็น -1.0 ซึ่งการที่เราจะหาค่าทั้งหมดนี้ เราจะต้องรู้ค่ามุมการยิงสูงสุด(Maximum Angle) ของหุ่นยนต์ที่จะเล็งยิงไปยังหุ่นยนต์ของศัตรูเราว่าศัตรูกำลังเคลื่อนที่รอบเราอยู่ทิศทางตามเข็มนาฬิกาหรือทิศทางทวนเข็มนาฬิกา



รูปที่ 2.1 ตัวอย่างการคำนวณของตัวประกอบการคาดเดา

2.2.2ประวัติการเคลื่อนที่ (Movement Profile)

เป็นการรวมกันของข้อมูล ได้แก่ทิศทางที่เราควรจะยิงไปกับความถี่ของทิศทางการยิงแต่ละทิศที่มีการยิงถูกต้อง(กระสุนยิงโดนหุ่นยนต์ศัตรู) โดยประวัติการเคลื่อนที่นี้อาจเรียกได้อีกอย่างหนึ่งว่าข้อมูลของหุ่นยนต์



รูปที่ 2.2 ตัวอย่างประวัติการเคลื่อนที่ของหุ่นยนต์ Mashmallow ซึ่งถูกบันทึกโดยหุ่นยนต์FloodGrapher

2.2.3ค่าการเคลื่อนที่เฉลี่ย (Rolling Averages)

ในทางสถิติค่าการเคลื่อนที่เฉลี่ย(Rolling average หรืออีกชื่อหนึ่งคือ Moving Average) ถูกใช้งานในการวิเคราะห์เซตของจุดข้อมูล โดยการสร้างชุดของค่าเฉลี่ยของสับเซตข้อมูลทั้งหมด ดังนั้น ค่าการเคลื่อนที่เฉลี่ยจะไม่ใช่แค่ตัวเลขเพียงตัวเดียว แต่จะเป็นเซตของตัวเลข ซึ่งแต่ละตัวจะเป็นค่าเฉลี่ยตามส่วนของสับเซตของเซตที่ใหญ่กว่าของจุดข้อมูล ค่าแรกของค่าการเคลื่อนที่เฉลี่ยอาจจะเป็นค่าเฉลี่ยเลขคณิต(Arithmetic mean) ของจุดข้อมูลที่1 ถึง 25 หลังจากนั้นค่าต่อไปก็จะเป็นค่าเฉลี่ยของจุดข้อมูลที่ 2 ถึง 26 ไล่ไปแบบนี้เรื่อยๆจนสุดท้ายก็จะเป็นค่าเฉลี่ยของจุดข้อมูล 76 ถึง 100

ในส่วนของโรโบโค้ดนั้นค่าการเคลื่อนที่เฉลี่ยโดยปรกติทั่วไปจะหมายถึงค่าที่จะนิยมยอมให้หุ่นยนต์ของเราใช้ข้อมูลของหุ่นยนต์ศัตรูที่ใหม่กว่าข้อมูลเดิม ซึ่งปีนของหุ่นยนต์ที่ใช้รูปแบบการเล็งแบบคาดเดาและจดจำไปสู้กับหุ่นยนต์ที่มีการเปลี่ยนรูปแบบการเคลื่อนที่ ก็จะมีประสิทธิภาพต่ำ แต่ถ้าหากใช้ค่าการเคลื่อนที่เฉลี่ยเข้าช่วยด้วย ก็จะทำให้มีประสิทธิภาพสูงขึ้นกว่าเดิม

รูปที่ 2.3 ตัวอย่างการให้ค่าน้ำหนัก(Weight)ของค่าการเคลื่อนที่เฉลี่ย(Rolling average)

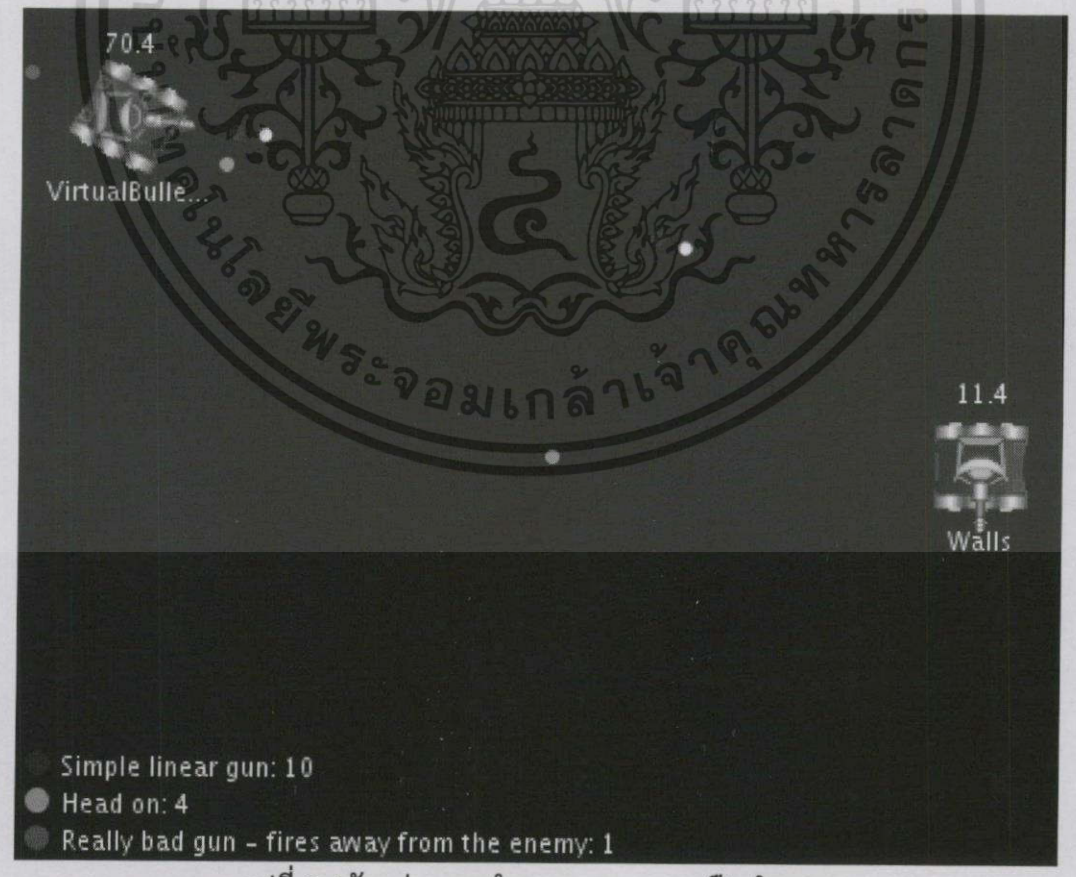
2.2.4 เซกเมนเตชั่น (Segmentation)

เซกเมนเตชั่น (Segmentation) เป็นค่าหรือข้อมูลบางอย่างที่หุ่นยนต์ของเราเก็บมาหรือเราเป็นคนใส่เพิ่มเข้าไป ซึ่งหุ่นยนต์ของเราสามารถดึงออกมาใช้ได้แทนข้อมูลปรกติที่ใช้โดยปกติข้อมูลดังกล่าวจะเก็บมาจากคลื่น (Wave) หรือกระสุนจำลอง (Virtual Bullet)

2.2.5กระบอกปืนจำลอง (Virtual Gun)และ กระสุนจำลอง (Virtual Bullets)

กระบอกปืนจำลองและกระสุนจำลองที่ถูกสมมติขึ้นในการยิง ทดสอบว่าถ้าหุ่นยนต์ของเรายิงไปในทิศทางหนึ่งๆ ณ เวลาหนึ่งๆ กระสุนที่ถูกยิงออกไปจะโดนหุ่นยนต์ของศัตรูหรือไม่

เมื่อหุ่นยนต์ยิงกระสุนจริงออกไป กระบอกปืนจำลองแต่ละกระบอก จะทำโพลล์ (poll)ของอองศาการยิงที่มันควรใช้ออกมา โดยจะเก็บ จุดที่ทำการยิง และความเร็วของกระสุนจำลองที่ทำการยิงออกไปไว้ด้วย หุ่นยนต์จะทำการติดตามผลว่าการยิงครั้งไหนโดนหรือไม่โดนหุ่นยนต์ศัตรู เพื่อไปปรับค่าการยิงต่างๆ ของหุ่นยนต์เรา ดังนั้นการที่หุ่นยนต์ของศัตรูมาไม่สามารถมองเห็นองศาการยิงของหุ่นยนต์เราที่ใช้ หุ่นยนต์ของศัตรูก็จะไม่สามารถปรับค่าการเคลื่อนที่ให้หลบกระสุนของหุ่นยนต์เราได้ ดังนั้นผลของการยิงจะแสดงให้เห็นถึงประสิทธิภาพกระบอกปืนจำลองแต่ละกระบอกนั่นเอง



รูปที่ 2.4ตัวอย่างการทำงานของกระบอกปืนจำลอง

2.2.6คลื่น (Wave)

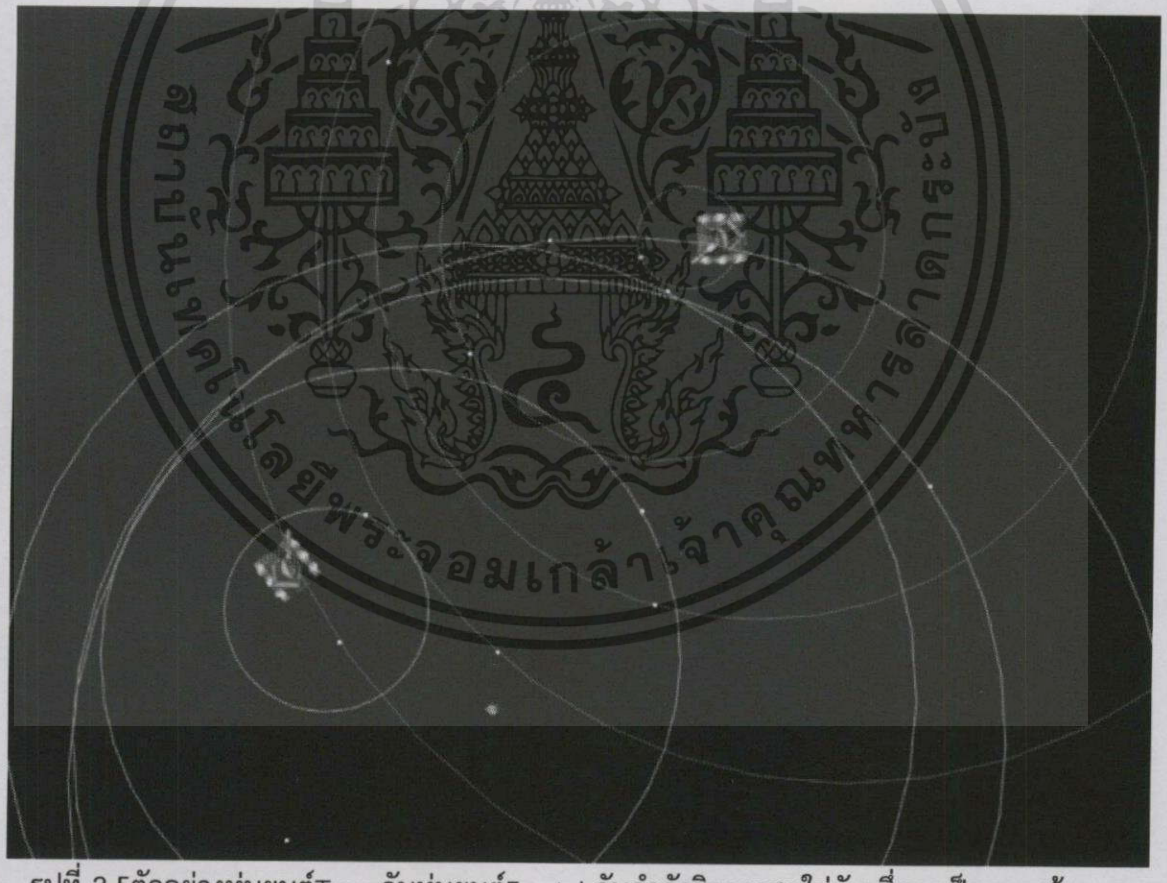
เป็นคลื่นที่หุ่นยนต์ของเราตรวจจับได้เพื่อหาทิศทางปืนของหุ่นยนต์ศัตรูว่าอยู่ทิศทางไหน เพื่อใช้สำหรับเป็นข้อมูลสำหรับไปประยุกต์ใช้ในการเคลื่อนที่หรือการเล็งเป้าของหุ่นยนต์ โดย คลื่นจะประกอบด้วย จุดกำเนิดคลื่น (หรือก็คือจุดที่หุ่นยนต์ศัตรูทำการยิงกระสุนออกมา), ความเร็วคลื่น (ขึ้นอยู่กับขนาดของกระสุนที่ศัตรูยิงมา),เวลาที่คลื่นถูกสร้างขึ้นมา, ทิศทางของปากกระบอกปืนของหุ่นยนต์ที่ยิงมา ณ เวลา ที่ยิงออกมา

คลื่นจะ“สลายตัว” หรือ “ชน” เมื่อระยะทางที่คลื่นเคลื่อนที่ได้ มีค่ามากกว่าระยะทางของจุดกำเนิดคลื่นถึงหุ่นยนต์ของศัตรู ซึ่งระยะทางที่คลื่นเคลื่อนที่ได้หาได้ดังนี้

- ระยะทาง = ความเร็วของคลื่น * (เวลา ณ ตอนนี้ – เวลาที่ยิงกระสุนออกมา)

เขียนเป็นภาษาอังกฤษได้ว่า

- Distance = wave_velocity * (time_now – time_fired)



รูปที่ 2.5ตัวอย่างหุ่นยนต์Tron กับหุ่นยนต์Fractal กับกำลังยิงกระสุนใส่กัน ซึ่งจะเป็นการสร้างคลื่น(Wave)ในแต่ละครั้งการยิง

2.3 เทคนิคการจัดกลุ่มแบบยืดหยุ่น

เทคนิคการจัดกลุ่มแบบยืดหยุ่น (Dynamic Clustering) เป็นเทคนิคที่ใช้ในการหาข้อมูล (entries)ในบันทึก(log) ที่มีลักษณะใกล้เคียงกับเหตุการณ์ในปัจจุบันมากที่สุด ซึ่งเทคนิคนี้ใช้วิธีการที่เรียกว่า "ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้สุด k ตัว" หรือ K-nearest neighbor algorithm และเทคนิคนี้ก็ไม่ได้มีความเกี่ยวข้องกับการการจัดกลุ่ม (Clustering) เสียทีเดียว เพียงแต่ว่าชื่อ "เทคนิคการจัดกลุ่มแบบยืดหยุ่น" นี้ ถูกใช้ในวงการโรโบโค้ดมานานจนเป็นที่ติดปากของผู้พัฒนาจนมาถึงผู้ใช้งาน

หลักการทำงานของเทคนิคนี้คือเก็บสถานะหรือเหตุการณ์ (state) สำหรับทุกๆข้อมูลในบันทึก ซึ่งเหตุการณ์นั้นจะประกอบไปด้วยข้อมูลอะไรก็ได้ที่ต้องการ เช่น ความเร็วของหุ่นยนต์ศัตรูที่วิ่งมาหาหุ่นยนต์ของเรา (Advancing Velocity) , ความเร็วของหุ่นยนต์ในแนวแกนตั้งฉากกับหุ่นยนต์ที่ใช้อ้างอิง (Lateral Velocity) , ระยะห่างจากหุ่นยนต์ศัตรู (Enemy Distance) เป็นต้น เหตุการณ์เหล่านี้จะถูกรวบรวมเป็นข้อมูล ซึ่งการที่จะนำข้อมูลเหล่านี้ไปใช้ได้

เรามีความจำเป็นที่จะต้องหา "ระยะห่าง" (Distance) ระหว่างเหตุการณ์ปัจจุบัน (Current state) กับเหตุการณ์ล่าสุดที่เพิ่งผ่านมา (Past state) ระยะห่างระหว่างเหตุการณ์ที่กล่าวมานั้นสามารถหาได้ด้วยวิธีการวัดระยะทางแบบยูคลิด (Euclidian)

$$\sqrt{(dist1 - dist2)^2 + (lat1 - lat2)^2 + \cdots} \tag{2.1}$$

หรืออีกวิธีหนึ่งคือหาด้วยวิธีวัดระยะทางแบบแมนฮัตตัน (Manhattan distance)

$$|dist1 - dist2| + |lat1 - lat2| + \cdots \tag{2.2}$$

หลังจากนั้นก็ให้หาข้อมูลที่มีระยะห่างระหว่างเหตุการณ์น้อยที่สุด และนำข้อมูลนั้นมาใช้เป็นส่วนหนึ่งของหุ่นยนต์ในโรโบโค้ด ไม่ว่าจะเป็นเรื่องการเล็งยิง หรือ การเคลื่อนที่ เพื่อให้หุ่นยนต์มีประสิทธิภาพตามที่ต้องการในสมัยยุคแรกๆ ที่ยังไม่ได้ใช้วิธียูคลิดหรือแมนฮัตตันนั้น ได้ใช้วิธีการคำนวณที่บันทึกโดยตรง โดยคำนวณจากระยะห่างของข้อมูลในบันทึก (Log entries) ไม่ได้คำนวณที่ตัวเหตุการณ์ (State) แบบในปัจจุบัน ทำให้เกิดปัญหาที่ว่า ถ้าบันทึกมีจำนวนมากก็จะทำให้การทำงานช้าลงมาก เพราะต้องคำนวณบันทึกที่เพิ่มขึ้นนั่นเอง

2.3.1 การแปลงเป็นจุดข้อมูล (Translating to data points)

จุดข้อมูล (Data point) เป็นบริเวณหนึ่งในพื้นที่ข้อมูลจินตภาพ (Imaginary data space) โดยสนามรบในโรโบโค้ดมีลักษณะเป็นพื้นที่ข้อมูล (Data space) 2 มิติ ประกอบด้วยพิกัด (x,y) ซึ่งพิกัดเหล่านี้อาจเป็นพิกัดของตัวหุ่นยนต์, กระสุนปืน, คลื่น ฯลฯ เราสามารถคำนวณระยะทางได้โดยวิธีการวัดระยะทางระหว่างหุ่นยนต์สองตัวแบบยูคลิดได้ดังนี้

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2.3}$$

ซึ่งเราสามารถนำวิธีการนี้ไปใช้ในลักษณะพื้นที่ข้อมูลที่มากกว่า 2 มิติได้ เช่น ถ้าเป็น 5 มิติจะได้ว่า

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 + (p_1 - p_2)^2 + (q_1 - q_2)^2} \tag{2.4}$$

นี่คือวิธีการที่ใช้ในการวัดความคล้ายคลึงกันระหว่างเหตุการณ์ (Situation) 2 เหตุการณ์ โดยการแปลงเหตุการณ์ เหล่านั้นให้เป็นจุดข้อมูล และนำจุดข้อมูลนั้นมาคำนวณระยะทางระหว่างหุ่นยนต์ทั้งสองแบบยูคลิด โดยเราสามารถเก็บข้อมูลสำหรับการคำนวณในรูปแบบกี่มิติก็ได้ตามต้องการ การเก็บข้อมูลตั้งแต่ 6 มิติขึ้นไปก็ไม่ใช่เรื่องแปลก แต่ในที่นี้จะกล่าวถึงแค่ 3 มิติก็พอ เพื่อให้เข้าใจถึงหลักการแบบง่ายๆก่อน

สำหรับวิธีการแปลงเหตุการณ์ต่างๆให้อยู่ในรูปของจุดข้อมูล เราเริ่มต้นจากนำค่าตัวเลขที่อธิบายเหตุการณ์นั้นๆมาใช้งาน เช่น ความเร็ว (speed) ทิศทางที่หุ่นยนต์กำลังหันไปอยู่ (relative heading) และ พลังงาน (energy) ของหุ่นยนต์ศัตรู เราต้องการที่จะนอร์มัลไลซ์ (Normalize) ค่าทั้ง 3 ค่าให้อยู่ในช่วงเดียวกันคืออยู่ระหว่าง 0 กับ 1 ถ้าเราไม่ทำแบบนี้ ตัวแปรที่มีค่าตัวเลขมากจะมีน้ำหนักความสำคัญมากกว่าตัวแปรที่มีค่าตัวเลขน้อย จากโค้ดด้านล่างคือการแปลงสถานะของหุ่นยนต์ศัตรูให้เป็นจุดข้อมูล

2.3.2 การระบุความใกล้เคียงกันของเหตุการณ์ (Identifying similar situations)

หลังจากที่เราได้สร้างบันทึกของเหตุการณ์ในอดีตแล้ว เราต้องการที่จะหาบันทึกของเหตุการณ์ที่ใกล้เคียงกันกับเหตุการณ์ปัจจุบัน ซึ่งการจะทำแบบนี้ได้ จะต้องมีการคำนวณหลายอย่างมาก จึงเป็นเรื่องสำคัญที่จะต้องทำโค้ดให้ซับซ้อนน้อยที่สุดเท่าที่จะเป็นไปได้ กำหนดให้มีค่าต่างๆดังนี้ ได้แก่ บันทึกของข้อมูลที่เก็บเหตุการณ์ในอดีตไว้, จุดข้อมูลที่เก็บข้อมูลของเหตุการณ์ปัจจุบันไว้ และ

จำนวนเหตุการณ์ที่เราต้องการค้นหา ซึ่งด้านล่างนี้คือโค้ดเทียม (Pseudocode) สำหรับอัลกอริธึมแบบ "บรูทฟอร์ซ" ที่ได้รับการปรับปรุงอย่างดีแล้ว (Well optimized brute force algorithm) สำหรับการค้นหาเหตุการณ์ในอดีตที่มีความใกล้เคียงกับเหตุการณ์ปัจจุบันมากที่สุด

```
nearestNeighbors(allPoints, currentPoint, numNeighbors,
numDimensions)
definenearestPoints, collection of size <numNeighbors>
fillnearestPoints with the first <numNeighbors> points
from allPoints

definenearestDistancesSq, double array of size
<numNeighbors>
setnearestDistancesSq[0] to
distanceSquared(currentPoint, nearestPoints[0])
definelongestDistanceSq, initialize to
nearestDistanceSq[0]
definelongestIndex, initialize to 0

for x = 1 to numNeighbors-1
nearestDistancesSq[x] = distanceSquared(currentPoint,
nearestPoints[x]);
if (nearestDistancesSq[x] >longestDistanceSq)
longestDistanceSq = nearestDistancesSq[x]
longestIndex = x

for x = numNeighbors to allPoints.length-1
thisDistanceSq = distanceSquared(currentPoint,
allPoints[x]);
if (thisDistanceSq<longestDistanceSq)
nearestPoints[longestIndex] = allPoints[x]
nearestDistanceSq[longestIndex] = thisDistanceSq
find the new maximum value in nearestDistancesSq
setlongestDistanceSq to the maximum value
setlongestIndex to the array index of the maximum value

returnnearestPoints
end
```

รูปที่ 2.6 โค้ดเทียมการระบุความใกล้เคียงกันของเหตุการณ์

สังเกตว่าในโค้ดข้างต้นไม่ได้มีการคำนวณเกี่ยวกับสแควร์รูท (Square root) เพื่อหาค่าระยะทางของยูคลิดที่แท้จริง เพราะสแควร์รูทเป็นการคำนวณที่ช้ามาก และก็ไม่มีผลว่าระยะทางไหนใกล้หรือไกลกว่ากัน

2.3.3 สิ่งที่ต้องคำนึง(Considerations)

จากที่เห็นในโค้ดข้างต้นความเร็วที่ใช้ในการคำนวณจะแปรผันตรงกับขนาดของตัวแปร allPointsและจำนวนมิติในพื้นที่ข้อมูล (Data space) ถ้าหุ่นยนต์มีการเพิ่มข้อมูลทุกๆติ๊ก(tick) จะทำให้หุ่นยนต์นั้นต้องเก็บจุดข้อมูลอย่างน้อย 25,000 ชุดจากการแข่งขัน 35 รอบ ซึ่งแม้จะมีแค่ 3 มิติ แต่นั่นหมายถึงการคำนวณต่างๆประมาณ 100,000 สมการ เพื่อที่จะหาเหตุการณ์ในอดีตที่ใกล้เคียงกับเหตุการณ์ในปัจจุบันมากที่สุด ซึ่งถ้านำวิธีนี้ไปใช้ในการเล็งยิง หุ่นยนต์จะต้องทำการคำนวณตามที่กล่าวมาข้างต้นทุกๆครั้งที่มีการเล็งยิง ดังนั้นเราจึงต้องมีการล้างข้อมูลที่เก่ามากๆออกเพื่อให้มีพื้นที่สำหรับเก็บจุดข้อมูลใหม่ โดยอาจจะตั้งค่าไว้ว่าเมื่อเก็บบันทึกเหตุการณ์มาจนถึงเหตุการณ์ที่ 30,000 ถ้ามีการเก็บบันทึกเหตุการณ์อีก เหตุการณ์ที่เก่าที่สุดจาก 30,000 เหตุการณ์นั้นจะถูกลบทิ้งไปในทันที เป็นต้น

2.3.4 วิเคราะห์ความใกล้เคียงกันของเหตุการณ์ (Analyzing the similar situations)

หลังจากเราได้ดูบันทึกเหตุการณ์แล้วค้นพบกลุ่มของเหตุการณ์ที่มีความใกล้เคียงกันมากๆกับเหตุการณ์ในปัจจุบัน เรายังมีหลายๆสิ่งหลายๆอย่างที่ต้องทำก่อนจะนำบันทึกเหตุการณ์นั้นมาใช้งาน ไม่ว่าจะเป็นเรื่องการเล็งยิงศัตรู หรือการหลบกระสุนศัตรูเราอาจใช้วิธีการคาดเดาและจดจำ (GuessFactors) เพื่อระบุทิศทางในการยิงของหุ่นยนต์ของเราหรืออาจจะใช้วิธีการจำลองแบบเล่นไปข้างหน้า (Play-It Forward) เพื่อที่จะดูว่าหุ่นยนต์ศัตรูเคลื่อนที่ไปทิศทางไหนบ้างในอดีต เมื่อเราได้มุมยิงหลายๆมุมที่ใกล้เคียงแล้ว ก็จะมีการใช้ตัวประมาณความหนาแน่นของเคอร์เนลหลายๆแบบเพื่อที่จะช่วยตัดสินใจว่ามุมยิงไหนเป็นมุมยิงที่ดีที่สุด

2.3.5 การเก็บข้อมูลมุมที่ใช้ในการยิง (Storing firing angles)

เมื่อเริ่มมีการบันทึกจุดข้อมูล เราจำเป็นต้องบันทึกข้อมูลเพิ่มเติมที่เกี่ยวข้องกับเหตุการณ์ที่ทำให้เราสามารถระบุมุมยิงได้ ซึ่งข้อมูลเพิ่มเติมที่ว่าอาจจะเป็นตัวแปรคาดเดาและจดจำ (GuessFactor) ค่าออฟเซ็ทไปยังทิศทางศัตรู (Bearing Offset) เวกเตอร์การกระจัด (Displacement Vector) หรือบันทึกการเคลื่อนที่ของหุ่นยนต์ศัตรูที่เป็นไปตามช่วงเวลาหนึ่งๆ (สำหรับวิธีการแบบเล่นไปข้างหน้า หรือ Play-It Forward) ซึ่งในที่นี้เราจะใช้ GuessFactorsโดยสิ่งที่เราจะเพิ่มเข้าไปหลังจากเราเก็บข้อมูลคลื่น (Wave) แล้วมีดังนี้

- แปลงเหตุการณ์ที่มีการยิงให้เป็นจุดข้อมูลและเพิ่มจุดข้อมูลลงไปในคลื่นที่หุ่นยนต์เราเก็บข้อมูลมาได้

- เมื่อคลื่นตกกระทบที่กระบอกปืน หรือหุ่นยนต์ของเราถูกยิงระหว่างกำลังเคลื่อนที่ให้เพิ่มเหตุการณ์นั้นลงไปใน HashMapรวมกับจุดข้อมูลที่มีตัวแปรคาดเดาและจดจำที่คลื่นนี้สร้างขึ้นมา
- หลังจากนั้น เมื่อเราพบกลุ่มของเหตุการณ์ที่มีลักษณะใกล้เคียงกัน เราสามารถดูที่ตัวแปรการคาดเดาและจดจำที่ถูกสร้างขึ้นมาในแต่ละเหตุการณ์ และสามารถนำตัวแปรนั้นไปแปลงเป็นมุมในการยิงต่อไป

2.3.6 การตัดสินใจ (Making decisions)

หลังจากที่เราได้กลุ่มของเหตุการณ์ที่มีความใกล้เคียงกันแล้ว เราสามารถค้นหาหรือสร้างมุมยิงของแต่ละเหตุการณ์ขึ้นมาได้ อย่างในตัวอย่างนี้คือการดึงค่าตัวแปรการคาดเดาและจดจำออกมาจาก HashMapแล้วคำนวณมุมในการหลบหนีสูงสุด (Maximum Escape Angle (MEA)) สำหรับพลังของกระสุนที่เราเลือกใช้หรือคลื่นที่เราตรวจจับ โดยนำค่ามุมในการหลบหนีสูงสุดคูณกับค่าตัวแปรการคาดเดาและจดจำ เพื่อกำหนดทิศทางขึ้นมา

ซึ่งสิ่งที่เราจะทำกับมุมในการยิงเหล่านี้ขึ้นอยู่กับว่า ณ ตอนนั้นเราอยู่ในช่วงการเล็งยิง หรืออยู่ในช่วงการหลบกระสุนถ้าอยู่ในช่วงการเล็งยิง เราจะเลือกมุมยิงที่มีโอกาสยิงโดนศัตรูมากที่สุดโดยดูจากการเคลื่อนที่ของหุ่นยนต์ในบันทึกที่ผ่านมา แต่ถ้าอยู่ในช่วงการหลบหลีกศัตรู เราจะต้องวัดค่าความอันตรายของมุมยิงของหุ่นยนต์ศัตรูโดยดูจากลักษณะการเคลื่อนที่ของหุ่นยนต์ของเรา ขึ้นอยู่กับมุมยิงที่ได้มาจากเหตุการณ์ที่ใกล้เคียงกันนั้น มีความใกล้เคียงกันมากน้อยแค่ไหน

2.3.7ตัวอย่างการเล็งยิง (Aiming example)

ในตัวอย่างการเล็งยิงนี้ สมมติว่าเรากำลังเล็งยิงศัตรูอยู่ เราพบเหตุการณ์ที่มีความใกล้เคียงกันแล้ว กำหนดพลังงานกระสุนเรียบร้อย และหุ่นยนต์ศัตรูมีระยะห่างจากเราตามค่าตัวแปร distanceToEnemyตัวอย่างโค้ดเทียมเพื่อที่จะระบุมุมยิงจะมีลักษณะดังนี้

```
firingAngle(similarPoints, guessFactors, bulletPower,
distanceToEnemy)
definebestAngle
definebestDensity, initialize to 0
definebotWidthAngle, set to abs(36/distanceToEnemy)
for each dataPointA in similarPoints
define density, initialize to 0
definefiringAngleA, set to
```

```
firingAngleFromGuessFactor(guessFactors{dataPointA},
bulletPower)

for each dataPointB in similarPoints
if (dataPointA != dataPointB)
definefiringAngleB, set to
firingAngleFromGuessFactor(guessFactors{dataPointB},
bulletPower)

defineux = (firingAngleA - firingAngleB) /
botWidthAngle
if (abs(ux) <= 1)
density++

if (density >bestDensity)
bestAngle = firingAngleA
bestDensity = density

returnbestAngle
end
```

รูปที่ 2.7 โค้ดเทียมตัวอย่างการเล็งยิง

2.3.8 ตัวอย่างการเซิร์ฟคลื่น (Wave Surfing example)

สมมติว่าเรากำลังหลบหลีกศัตรูด้วยการเซิร์ฟคลื่น นั่นหมายถึงเรากำลังเลือกลักษณะการเคลื่อนที่ทุกๆติ๊ก (tick) สิ่งที่เราต้องการคือวัดความอันตรายของมุมยิงที่หุ่นยนต์ศัตรูยิงมาว่า จะยิงโดนหุ่นยนต์ของเราหรือไม่ หากเราเลือกลักษณะการเคลื่อนที่นี้ เพื่อที่เราสามารถเลือกลักษณะการเคลื่อนที่ที่ปลอดภัยที่สุดนั่นเอง

เราสามารถคำนวณความหนาแน่นของเคอร์เนล (Kernel density) สำหรับแต่ละมุมยิงของหุ่นยนต์ศัตรูได้ โดยใช้หลักการเดียวกันกับการเล็งยิงของหุ่นยนต์เราเองแต่แบบนี้จะมีปัญหาอยู่ 2 เรื่อง เรื่องแรกคือเราจะมีมุมยิงของหุ่นยนต์ศัตรูหลายๆมุมที่มีระดับความอันตรายเท่ากัน ซึ่งทำให้หุ่นยนต์ของเราจะอยู่บริเวณขอบของพื้นที่ที่คำนวณไว้แล้วว่ามีความอันตราย ดังนั้นเราจึงต้องการค่าความอันตรายที่มีความละเอียดมากขึ้น ซึ่งตัวประมาณค่าความหนาแน่นของเกาเซียน (Gaussian density estimator) สามารถทำงานได้ดีในเรื่องนี้ เรื่องที่สอง ข้อมูลของการเซิร์ฟส่วนใหญ่จะมีปริมาณน้อยกว่าข้อมูลการเล็งยิง ทำให้เหตุการณ์ที่ใกล้เคียงกันที่เราค้นพบ อาจจะไม่ได้ใกล้เคียงกันอย่างที่คิดก็ได้ ดังนั้นการให้ค่าน้ำหนักแก่เหตุการณ์ต่างๆโดยแบ่งตามระยะทางระหว่างหุ่นยนต์ศัตรูถึงจุดที่หุ่นยนต์เราเซิร์ฟคลื่นจะช่วยตรงจุดนี้ได้มาก

```
checkDanger(similarPoints, surfWaveDataPoint,
guessFactors, bulletPower,
projectedDistanceToEnemy, projectedFiringAngle)

define density, initialize to 0
definebotWidthAngle, set to
abs(36/projectedDistanceToEnemy)
for each dataPointA in similarPoints
definefiringAngleA, set to
firingAngleFromGuessFactor(guessFactors{dataPointA},
bulletPower)

defineux = (firingAngleA - projectedFiringAngle) /
botWidthAngle
density += e^(-0.5 * (ux^2))
          / distance(dataPointA, surfWaveDataPoint)

return density
end
```

รูปที่ 2.8 โค้ดเทียมตัวอย่างการเซิร์ฟคลื่น

ในเรื่องการเซิร์ฟ มีสิ่งที่ต้องคำนึงอยู่ 2 อย่าง

- เราต้องการเซิร์ฟเหตุการณ์ / จุดข้อมูล จากเวลาที่คลื่นถูกยิงออกมา ไม่ใช่เหตุการณ์ ณ ตอนนั้น
- เราไม่ควรจะคำนวณเหตุการณ์ที่ใกล้เคียงกันมากที่สุดทุกๆติ๊ก เราสามารถเก็บแคช (cache) เหตุการณ์ดังกล่าวสำหรับคลื่นแต่ละลูกหลังจากคำนวณไปแล้ว 1 ครั้ง ถ้าหุ่นยนต์เราถูกยิงหรือตรวจพบว่ามีกระสุนกำลังจะพุ่งมาโดน ให้เคลียร์แคชและรีเฟรช (refresh) ค่าเหตุการณ์ที่ใกล้เคียงกันมากที่สุดสำหรับคลื่นทุกๆลูก

2.4 ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้สุด k ตัว (k-nearest neighbors algorithm)

ในเรื่องการจดจำรูปแบบ (pattern recognition) ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้สุด k ตัว (k-Nearest Neighbors algorithm)เป็นวิธีการที่ใช้ในการจัดแบ่งคลาส โดยวิธีการนี้จะช่วยในการตัดสินใจ เพื่อให้คลาสสามารถนำมาใช้แทนเงื่อนไขหรือกรณีใหม่ๆ โดยการตรวจสอบจำนวน k

จำนวน ของกรณีหรือเงื่อนไขที่เหมือนกันหรือใกล้เคียงกันมากที่สุด โดยจะหาผลรวมของจำนวนเงื่อนไข หรือกรณีต่างๆ สำหรับแต่ละคลาส และกำหนดเงื่อนไขใหม่ๆ ให้คลาสที่เหมือนกันกับคลาสที่ใกล้เคียงกันมากที่สุด

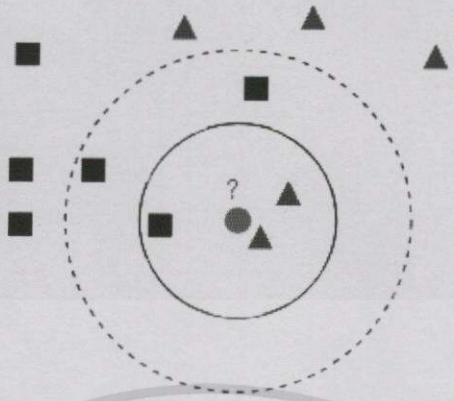
สำหรับการนำเทคนิคขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้สุด k ตัว ไปใช้ในนั้น เป็นการหาวิธีการวัดระยะห่างระหว่างแต่ละตัวแปร(Attribute)ในข้อมูล และจากนั้นคำนวณค่าออกมาให้ได้ ซึ่งวิธีนี้จะเหมาะสำหรับข้อมูลแบบตัวเลข แต่ตัวแปรที่เป็นค่าแบบไม่ต่อเนื่องนั้นก็สามารถทำได้ แต่อาจต้องใช้การจัดการรูปแบบอื่นเพิ่มขึ้นมา เช่น ถ้าเป็นเรื่องสี เราจะใช้สิ่งใดวัดความแตกต่างระหว่างสีแดงกับสีเหลือง ต่อจากนั้นเราต้องหาวิธีในการรวมค่าระยะห่างของตัวแปรทุกค่าที่วัดมาได้ เมื่อเราสามารถคำนวณระยะห่างระหว่างเงื่อนไขหรือกรณีต่างๆได้ จากนั้นจึงเลือกชุดของเงื่อนไขที่ใช้จัดคลาสมาเป็นฐานสำหรับการจัดคลาสในเงื่อนไขใหม่ๆ ได้ แล้วค่อยจะตัดสินได้ว่าขอบเขตของจุดข้างเคียงที่ควรเป็นนั้น ควรมีขนาดเท่าไร และอาจตัดสินใจได้อีกว่าจะนับจำนวนจุดข้างเคียงตัวมันได้ด้วยวิธีไหน

2.4.1 ขั้นตอนวิธีการ (Algorithm)

ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวมีขั้นตอนโดยสรุป ดังนี้

- กำหนดขนาดของ K (ควรกำหนดให้เป็นเลขคี่)
- คำนวณระยะห่าง (Distance) ของข้อมูลที่ต้องการพิจารณากับกลุ่มข้อมูลตัวอย่าง
- จัดเรียงลำดับของระยะห่าง และเลือกพิจารณาชุดข้อมูลที่ใกล้จุดที่ต้องการพิจารณาตามจำนวน K ที่กำหนดไว้
- พิจารณาข้อมูลจำนวน k ชุด และสังเกตว่ากลุ่ม (class) ไหนที่ใกล้จุดที่พิจารณาเป็นจำนวนมากที่สุด
- กำหนด class ให้กับจุดที่พิจารณา (class) ที่ใกล้จุดพิจารณามากที่สุด

จากรูป เป็นตัวอย่างการจัดกลุ่มข้อมูลของขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวอย่างง่ายโดยดูเป็นรูปภาพ



รูปที่ 2.9 ตัวอย่างการจัดกลุ่มของขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุด k ตัว อย่างง่าย

กำหนดให้จุดที่พิจารณาคือ วงกลมสีเขียว
ซึ่งวงกลมสีเขียวนี้ควรจะจัดกลุ่มให้จุดที่สนใจไปอยู่ในคลาสแรกของสี่เหลี่ยมสีน้ำเงินหรือคลาสสองของสามเหลี่ยมสีแดง
ถ้า k=3 แล้ว วงกลมสีเขียวจะอยู่ในคลาสสอง เพราะมี สามเหลี่ยม 2 รูป และ สี่เหลี่ยม 1 รูป อยู่ในวงกลมวงใน (เส้นทึบ)
ถ้า k=5 แล้ว วงกลมสีเขียวจะอยู่ในคลาสแรก เพราะมี สี่เหลี่ยม 3 รูป และ สามเหลี่ยม 2 รูป อยู่ในวงกลมวงนอก (เส้นประ)

2.4.2 การดำเนินการหลัก (Main operation)

การดำเนินการหลักจะมีอยู่ด้วยกัน 2 ฟังก์ชัน (Function) ได้แก่

- ฟังก์ชันระยะทาง(Distance Function)

  เป็นการคำนวณค่าระยะห่างระหว่างตัวเก็บบันทึก (Record) เพื่อที่จะมาวัดความคล้ายคลึงกันของข้อมูล

- ฟังก์ชันการแจกแจง(Combination Function)

  เป็นการรวมกันของผลลัพธ์ที่ได้จากการคำนวณค่าระยะห่าง (Distance) โดยทำการเรียงลำดับค่าระยะห่าง (Distance) จากน้อยไปมาก หลังจากนั้นดูจากค่า k ว่ากำหนดเป็นเท่าไร แล้วนำลำดับที่เรียงได้มาเทียบกับคลาสข้อมูลที่เรียงแล้วนำมาตอบ

2.4.3คุณสมบัติของฟังชันก์ระยะทาง(Distance Function)

- ค่าระยะทาง(ความห่าง) ที่คำนวณได้ต้องไม่ติดลบ

- ถ้าตำแหน่งเดียวกัน ฟังก์ชันต้องเป็นศูนย์(ค่าเหมือนกัน)
- การคำนวณวัดระยะทางไปกลับต้องเท่ากัน

2.4.4 การคำนวณค่าฟังชันก์ระยะทาง(Distance Function)

- ใส่ค่าสัมบูรณ์(Absolute)ให้กับค่าระยะห่าง(Distance) : |A-B|
- ยกกำลังสองให้กับค่าระยะห่าง(Distance) : (A-B)2
- ทำการปรับให้เป็นค่ามาตรฐาน: |(A-mean)/(SD)-(B-mean)/(SD)|

2.4.5 การรวมค่าระยะทาง(Distance)ตัวเก็บบันทึก

- การวัดระยะแบบแมนฮัตตัน(Manhattan distance)เป็นการนำค่าที่คำนวณได้ในตัวเก็บบันทึก1 อันมารวมกัน
- ระยะทางแบบยุคลิด(Euclidean distance)เป็นการหารากที่สอง(Square Root)ในแต่ละตัวแปร(attribute)แล้วนำมารวมกัน แล้วนำค่าที่คำนวณได้ในตัวเก็บบันทึก 1 อันมารวมกัน

Data	X1	X2	Y(Class)	Distance	Nearest Sign
D1	6	1	+	3.6	+
D2	7	6	+	2.2	+
D3	6	4	-	2	-
D4	8	6	-	2	-
D5	4	4	-	4	
D6	3	1	+	5.8	
D7	6	10	-	6.3	
D8	6	6	+	8	
D9	10	6	-	2.8	-
D10	3	3	?		

ตาราง 2.1 ตัวอย่างการนำขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุด k ตัวไปใช้งานกับข้อมูลตัวเลข

จากตัวอย่างข้างต้น กำหนดให้ k= 5 จงหาค่า Y(Class) ของ D10 ว่าเป็น + หรือ –

เมื่อคำนวณระยะทางแบบยุคลิด (Euclidean distance) จะพบว่า data ที่ใกล้กับจุด (3,3) มากที่สุดมี5data โดยเรียงลำดับจากน้อยไปมากดังนี้ คือD1(+), D2(+), D3(-), D4(-), D9(-) จากระยะทางที่ใกล้ที่สุดทั้ง 5 ให้สังเกตว่ากลุ่ม (class) ที่มีจำนวนมากที่สุด ปรากฏว่าคือ class (-) ดังนั้น จึงกำหนดกลุ่ม (class) ให้กับจุด (3,3) คือ class (-)

2.4.6 ตัวอย่างการประยุกต์ใช้งาน

ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวได้มีการนำไปใช้ประโยชน์ในหลากหลายด้าน เช่น

- การวิเคราะห์รูปแบบการบุกรุกเครือข่ายคอมพิวเตอร์ โดยขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวกล่าวคือ มีการประยุกต์ใช้ในการตรวจสอบความปลอดภัยในเครือข่ายระบบคอมพิวเตอร์ ซึ่งจะใช้ตัวแปร k ในการวิเคราะห์ข้อมูล และจำแนกประเภทของข้อมูลที่มีขนาดใหญ่ๆ ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวสามารถใช้ในการวิเคราะห์ข้อมูลได้เป็นอย่างดี
- ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวเป็นเทคนิคหนึ่งของการทำเหมืองข้อมูล(Data Mining) ที่ใช้ในการแก้ปัญหาแบบการจัดแบ่งคลาส(classification) ในการผลิต ผลิตภัณฑ์ด้านการทำเหมืองข้อมูลในปัจจุบัน
- ประยุกต์ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวและการคำนวณอย่างอ่อน(Soft Computing) เพื่อใช้ประมาณค่าสูญหายของข้อมูล โดยจะใช้ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวในการหาสมาชิกที่มีความใกล้ชิดกับข้อมูลที่สูญหายมา k ตัว แล้วนำจำนวน k มาหารเพื่อประมาณค่า และได้ปรับปรุงตัวหารด้วยการนำ หลักการคำนวณอย่างอ่อน(Soft Computing) เข้ามาช่วยในการปรับค่า k เรียกวิธีการนี้ว่า ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวอย่างอ่อน(Soft K-Nearest Neighbour)
- ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวใช้ในการหามัธยฐานของตัวเลข n ตัวที่ต่างกัน โดยมีประสิทธิภาพการทำงานเป็น O(n)

2.4.7 สรุป

ขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวเป็นขั้นตอนวิธีการที่ใช้ในการจัดกลุ่มข้อมูล โดยการจัดข้อมูลที่อยู่ใกล้กันให้เป็นกลุ่มเดียวกันซึ่งเทคนิคนี้จะทำให้ตัดสินใจได้ว่าคลาสไหนที่จะแทนเงื่อนไขหรือกรณีใหม่ๆได้บ้าง โดยการตรวจสอบจำนวน K ซึ่งถ้าหากเงื่อนไขของการตัดสินใจมีความซับซ้อน วิธีนี้สามารถสร้างโมเดลที่มีประสิทธิภาพได้ แต่ขั้นตอนวิธีการเพื่อนบ้านใกล้ที่สุดจะใช้ระยะเวลาในการคำนวณนานถ้าตัวแปร (Attribute) มีจำนวนมากจะเกิดปัญหาในการคำนวณค่า

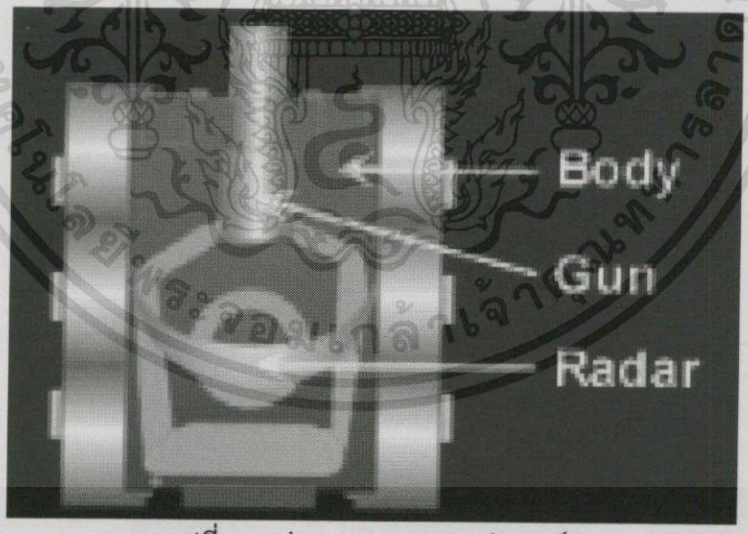
และค่อนข้างใช้ปริมาณงานในการคำนวณสูงมากบนคอมพิวเตอร์ เพราะเวลาที่ใช้สำหรับการคำนวณจะ เพิ่มขึ้นแบบแฟคทอเรียลตามจำนวนจุดทั้งหมด ดังนั้นเพื่อจะเพิ่มความรวดเร็วสำหรับเทคนิคขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวให้มากขึ้น ข้อมูลทั้งหมดที่ใช้บ่อยจะต้องถูกเก็บไว้ในหน่วยความจำ (Memory) โดยวิธีการเข้าถึงหน่วยความจำพื้นฐานอย่างมีเหตุผล (Memory-Based Reasoning) ซึ่งจะเป็นวิธีที่นำมาอ้างถึงเป็นประจำในการจัดเก็บกลุ่มคลาสของขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวในหน่วยความจำ และ ถ้าหากข้อมูลที่ต้องการหาคำตอบมีตัวแปรอิสระเพียงไม่กี่ตัวแล้ว จะทำให้เราสามารถเข้าใจโมเดลขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวได้ง่ายขึ้น ตัวแปรเหล่านี้ยังมีประโยชน์สำหรับนำมาสร้างโมเดลต่างๆ ที่เกี่ยวข้องกับชนิดของข้อมูลที่ไม่เป็นมาตรฐาน เช่น ข้อความ (Text) เพียงแต่อาจต้องมีมาตรฐานการวัดค่าสำหรับชนิดของข้อมูลดังกล่าวที่เหมาะสมด้วย นอกจากนี้ประสิทธิภาพของขั้นตอนวิธีการค้นหาเพื่อนบ้านใกล้ที่สุดk ตัวนี้ จะขึ้นอยู่กับจำนวนระยะห่าง การอธิบายระหว่างข้อมูลทั้งคู่ ที่สามารถแบ่งแยกอย่างมีประสิทธิภาพระหว่างข้อมูลปกติ และข้อมูลผิดปกติ การอธิบายจำนวนระยะห่างระหว่างขอมูลเป็นความท้าทายอย่างมากเมื่อข้อมูลมีความซับซ้อน อย่างเช่น ข้อมูลกราฟ และข้อมูลแบบลำดับ เป็นต้น

บทที่ 3
สภาพแวดล้อมของโรโบโค้ด และการออกแบบวิธีการเล็งยิงของหุ่นยนต์

สภาพแวดล้อม รวมถึงระบบฟิสิกส์ต่างๆ ของหุ่นยนต์ทุกตัวในโรโบโค้ด ไม่ว่าจะเป็นรูปร่าง ขนาด และความสามารถพื้นฐานปรกติ เช่น เรดาร์ตรวจจับข้อมูลเท่ากัน การรับรู้ต่างๆ ของหุ่นว่ายิงโดนศัตรูหรือไม่ หุ่นยนต์ทุกตัวได้เวลาแต่ละตาเท่ากัน เป็นต้น ทุกอย่างจะเหมือนกันทั้งหมด ไม่มีการได้เปรียบเสียเปรียบกันเรื่องเหล่านี้ หากแต่จะไปวัดกันที่ความสามารถที่ผู้พัฒนาหุ่นยนต์ พัฒนาความสามารถการวิเคราะห์สิ่งต่างๆ เช่น การเล็งยิงหุ่นยนต์ศัตรู หรือการหลบหลีกกระสุนของศัตรู เป็นต้น

3.1ส่วนประกอบของหุ่นยนต์ในโรโบโค้ด

หุ่นยนต์ในโรโบโค้ดประกอบด้วยกัน 3 ส่วน คือ



รูปที่ 3.1 ส่วนประกอบของหุ่นยนต์

3.1.1 ส่วนลำตัวหุ่นยนต์(Body)

เป็นส่วนล่างสุดที่ทำการแบกปืนไว้ และด้านบนสุดจะเป็นในส่วนของเรดาร์ ส่วนลำตัวเป็นส่วนที่ใช้สำหรับเคลื่อนที่ไปข้างหน้า ข้างหลัง หมุนไปด้านซ้าย และหมุนไปด้านขวา

3.1.2 กระบอกปืน (Gun)

กระบอกปืนถูกติดตั้งอยู่บนลำตัวของหุ่นยนต์ ทำหน้าที่สำหรับยิงกระสุน โดยกระบอกปืนนี้สามารถหมุนไปด้านซ้าย และหมุนไปด้านขวาเท่านั้น

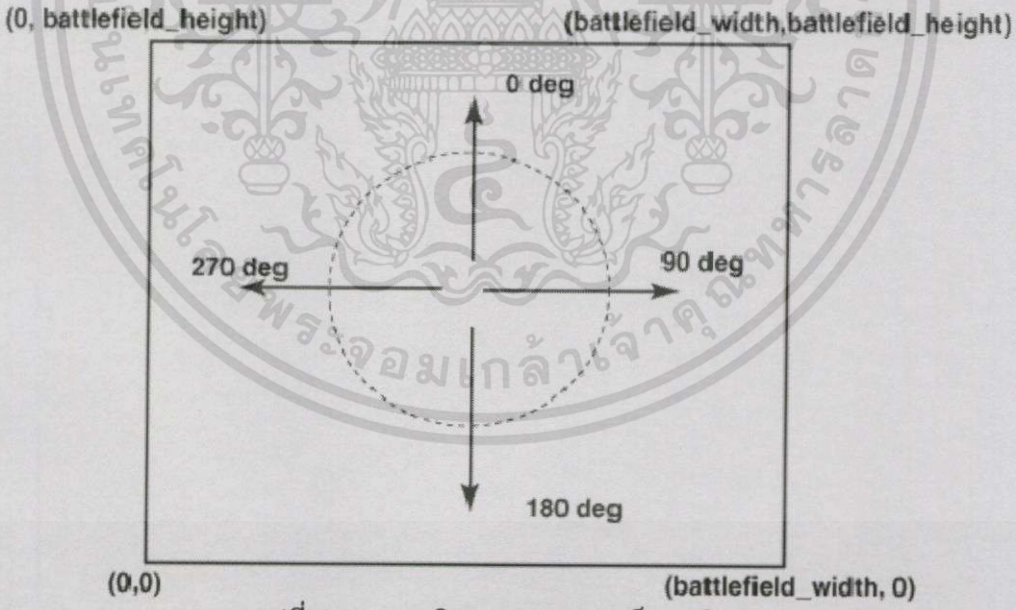
3.1.3 เรดาร์(Radar)

เรดาร์ถูกติดตั้งอยู่บนปืนของหุ่นยนต์ ทำหน้าที่ตรวจจับหุ่นยนต์ของศัตรูเมื่อทำการเคลื่อนที่ เรดาร์สามารถหมุนไปด้านซ้าย และหมุนไปด้านขวาเท่านั้น

3.2ระบบฟิสิกส์ของโรโบโค้ด

3.2.1 ระบบพิกัดและทิศทาง

- **ระบบพิกัด(Coordinates System):** โรโบโค้ดใช้ระบบพิกัดแบบคาร์ทีเซียน(Cartesian) ซึ่งหมายถึงพิกัด (0,0) จะอยู่ที่ตำแหน่ง ล่างซ้าย ของสนามรบ
- **ระบบทิศทางแบบตามเข็มนาฬิกา(Clockwise Direction):**โรโบโค้ดใช้ระบบทิศทางแบบตามเข็มนาฬิกา ซึ่ง ที่ 0 องศา จะเป็น “ทิศเหนือ”, ที่ 90 องศา จะเป็น “ทิศตะวันออก”, ที่ 180 องศา จะเป็น “ทิศใต้” และที่ 270 องศา จะเป็น “ทิศตะวันตก”



รูปที่ 3.2 ระบบทิศทางแบบตามเข็มนาฬิกา

3.3 เกณฑ์ของเวลาและระยะทาง ใน โรโบโค้ด

3.3.1เวลา(t)

เวลาในโรโบโค้ดจะวัดเป็น “ติก(tick)” หุ่นยนต์แต่ละตัวจะใช้เวลา 1 ตา ต่อ ติก (1 ติก= 1 ตา)

3.3.2 การวัดระยะทาง

ยูนิตในโรโบโค้ดโดยทั่วไปจะวัดในหน่วยพิกเซล(pixel) แต่มีข้อยกเว้น 2 อย่างคือ

1. ระยะทางทั้งหมดที่มีการวัดจะถูกจัดเก็บอยู่ในรูปของความเที่ยงสองเท่า (Double Precision) ดังนั้น เราสามารถเลื่อนพิกเซลทีละส่วนได้
2. โรโบโค้ดโดยปกติจะถูกปรับสเกลหน้าจอสนามแข่งให้พอดีกับความกว้างของหน้าจอ ในกรณีนี้ ทำให้ยูนิตของระยะทางจะมีขนาดเล็กกว่าพิกเซล

3.4 ระบบฟิสิกส์เรื่องการเคลื่อนที่ของหุ่นยนต์

3.4.1 ความเร่ง(a)

หุ่นยนต์จะเร่งความเร็วได้ที่

$$1\ pixel/turn^2 \quad (3.1)$$

และลดความเร็วได้ที่

$$2\ pixel/turn^2 \quad (3.2)$$

โดยโรโบโค้ดจะทำหน้าที่ในการตัดสินใจให้เองว่าจะใช้ความเร่งหรือไม่ ขึ้นอยู่กับระยะทางที่หุ่นยนต์ต้องการจะไป

3.4.2 สมการความเร็ว

$$\boldsymbol{v = at} \quad (3.3)$$

ความเร็วจะไม่สามารถเกิน 8 pixels/turn ได้ และในทางเทคนิคความเร็วจะเป็นปริมาณเวกเตอร์ แต่ในโรโบโค้ดจะสมมติทิศทางของเวกเตอร์เป็นทิศทางที่หุ่นยนต์หันหน้าไปด้านนั้น

3.4.3 สมการระยะทาง

$$d = vt \quad (3.4)$$

3.5 การหมุนของตัวหุ่นยนต์, กระบอกปืน และเรดาร์

3.5.1 อัตราหมุนสูงสุดของตัวหุ่นยนต์

$$\left(10 - 0.75 \times abs(velosity)\right) \ deg/turn \quad (3.5)$$

3.5.2 อัตราการหมุนสูงสุดของกระบอกปืน

$$20 \ deg/turn \quad (3.6)$$

3.5.3อัตราการหมุนสูงสุดของเรดาร์

$$45 \ deg/turn \quad (3.6)$$

3.6 ลูกกระสุนปืน

3.6.1 ค่าความเสียหายของลูกกระสุนปืน

$$4 \times Firepower \quad (3.7)$$

ถ้ากำลังกระสุน (Firepower) มีค่ามากกว่า 1 จะได้ค่าความเสียหายของกระสุนปืนเพิ่มมาอีก

$$2 \times (Firepower - 1) \quad (3.8)$$

3.6.2 ความเร็วของลูกกระสุนปืน

$$\mathbf{20 - 3 \times Firepower} \quad (3.9)$$

3.6.3 ค่าความร้อนของปืน (จากการยิงกระสุนปืน)

$$\mathbf{1 + \frac{Firepower}{5}} \quad (3.10)$$

หุ่นยนต์จะไม่สามารถยิงลูกกระสุนปืนออกไปได้ถ้า ความร้อนของปืน(GunHeat) มีค่ามากกว่า 0

3.6.4 ค่าพลังงานที่ได้กลับมา เมื่อลูกกระสุนที่ยิงออกไปถูกหุ่นยนต์ฝ่ายตรงข้าม

$$\mathbf{3 \times Firepower} \quad (3.11)$$

3.7 การชน

3.7.1 เมื่อชนกับหุ่นยนต์ตัวอื่น

เมื่อหุ่นยนต์ชนกับหุ่นยนต์ตัวอื่นจะได้รับความเสียหาย 0.6

3.7.2 เมื่อชนกับกำแพง

ถ้าชนกับกำแพงจะได้รับความเสียหายเท่ากับ

$$abs(velocity) \times 0.5 - 1$$

โดยค่าความเสียหายนี้จะต้องไม่น้อยกว่า 0

3.8 ระบบการนับคะแนน

เมื่อจบการต่อสู้ของหุ่นยนต์ในแต่ละตา ก็จะมีผลคะแนนต่างๆโชว์ออกมาให้เห็น

Results for 10 rounds

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	sample.Tracker	2889	800	160	1616	266	47	0	8	0	2
2nd	sample.RamFire	1990	350	0	1230	22	306	83	0	7	3
3rd	sample.MyFirstRo...	763	350	40	343	9	22	0	2	3	5

Save OK

รูปที่ 3.3 ผลการต่อสู้ของหุ่นยนต์

- Total Score– เป็นคะแนนของทุกอย่างรวมกัน ซึ่งจะเป็นตัวตัดสินผลการแข่งขันว่าหุ่นยนต์แต่ละตัวได้อันดับที่เท่าไร
- Survival Score–คะแนนการอยู่รอดในแต่ละรอบของหุ่นยนต์
- LastSurvivorBonus– คะแนนโบนัสพิเศษเมื่ออยู่รอดเป็นตัวสุดท้าย
- Bullet Damage– คะแนนที่ได้เมื่อหุ่นยนต์ยิงโดนหุ่นยนต์ศัครู
- Bullet Damage Bonus – คะแนนพิเศษเมื่อทำลายหุ่นยนต์ของศัครูได้
- Ram Damage – คะแนนที่ได้เมื่อหุ่นยนต์พุ่งเข้าชนทำความเสียหายแก่หุ่นยนต์ศัครู
- Ram Damage Bonus – คะแนนพิเศษเมื่อทำลายหุ่นยนต์ของศัตรูด้วยการพุ่งเข้าชน
- 1sts, 2nds, 3rds– ในส่วนนี้จะไม่เกี่ยวกับตัวคะแนน แต่จะเป็นส่วนที่บอกว่าหุ่นยนต์ตัวนี้มีจำนวนรอบที่อยู่รอดเท่าไร

3.9เริ่มต้นการออกแบบวิธีการเล็งยิงของหุ่นยนต์

```
Class Advanced Robot
{
        set variable

        runRobot()
        {
                setGunToTurnIndependentlyFromRobot(true)
```

```
                setRadarToTurnIndependenttlyFromGun(true)
                do
                {
                        turnRadarAround(1) // >0 = turn right , <0 = turn left
                } while(true)
        }

        environmentScanning(scannedEnvironment event)
        {
                energyManagement(event)
                surfTheWaveFromEnemyRobot()
                bulletFireControl()
        }

        robotIsBeingHitByEnemyBullet(hitByBullet event)
        {
                decreaseHitPoint(event)
                if(absolute(distanceOfEnemyRobotFromLocation -
waveSurfDistance) < 100)
                        saveLogHitOfRobotLocation()
        }

        bulletHitEnemyBullet(bulletHitBullet event)
        {
                getBulletLocation(event)
                loopCount(instant = 0; instant <enemyWaveSize; instant++)
                {
                        getEnemyWave(instant)
                        if(absolute(distanceOfEnemyRobotFromBulletLocation -
waveDistance) < 50)
```

```
                                        saveLogHitOfBulletLocation()
                }
        }

        bulletHitEnemyRobot (hitEnemyRobot event)
        {
                getEnemyHitPoint(event)
        }

        robotHitRobot (robotClashAgainstRobot event)
        {
                ramCountIncrease(event) // ram = clash != random access
memory
        }

        beingHitLogCollecting(data wave, distance robotLocation)
        {
                loopCount(instant = 46; instant >= 0; instant--)
                {
                        calculateTheWaveToRankTheDangerousOfArea(wave,
robotLocation)
                }
        }

        getFactorIndex(data wave, distance robotLocation)
        {
                calculateIndexForFactor(wave, robotLocation)
                getCalculateIndexForFactor()
        }
```

```
checkDanger(direction)
{
        set local variable
        do
        {
                if(thisMovement == predictedMovement)
                {
                        doNormalMovement()
                }
                calculateNewMovement()
        } while lastPredictedMovement = currentMovement
}

movementProjection(sourceLocation, angle, length)
{
        return new sourceLocationXintercept+sin(angle)*length,
sourceLocationYintercept+sin(angle)*length
}

assignSignValue(d)
{
        if(d < 0) return -1
        else return 1
}

avoidWall(angle, believeFactor)
{
        loop(!robotHitTheWall)
                angle += believeFactor*0.05
```

```
            return angle
    }

    calculateFieldAreaByUsingGun
    {
            returnfieldArea
    }

    class wave
    {
            set variable

            collectDistanceToPoint(d)
            {
                    return distance(d)
            }
            collectBelieveFactor(enemy)
            {
                    returnenemyLocationFactor(enemy)
            }
            booleanTest
            {
                    if(detectEnemyWave)
                    {
                            return true
                    }
                    return false
            }
    }
```

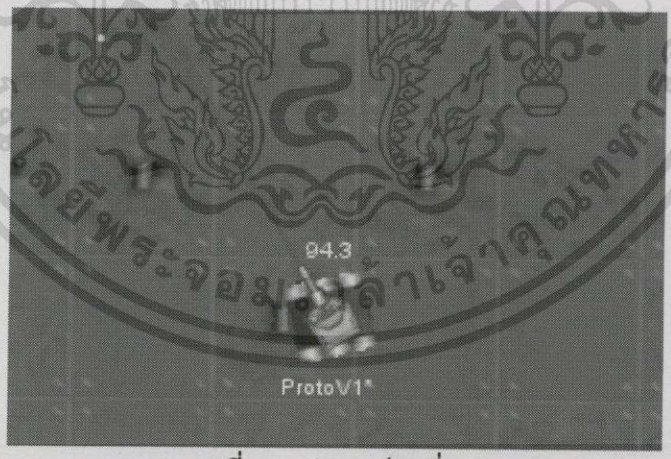
```
}
```

รูปที่ 3.4โค้ดเทียมของหุ่นยนต์

3.9.1 หุ่นยนต์รุ่นที่ 1

หุ่นยนต์รุ่นแรกที่ทำขึ้นมาเป็นหุ่นยนต์ที่ใช้ทฤษฎีการเล็งแบบคาดเดาและจดจำซึ่งประกอบไปด้วยตัวประกอบการคาดเดา(GuessFactor), เซกเมนเตชั่น(Segmentation) และคลื่น(Wave)โดย

- ให้ตัวประกอบการคาดเดามีค่าดังนี้
  - มีค่า 1 สำหรับหุ่นยนต์ของศัตรูที่เคลื่อนที่ตามเข็มนาฬิกาเมื่อเทียบกับการเคลื่อนที่ของหุ่นยนต์ของเรา
  - มีค่า -1 สำหรับหุ่นยนต์ของศัตรูที่เคลื่อนที่ทวนเข็มนาฬิกาเมื่อเทียบกับการเคลื่อนที่ของหุ่นยนต์ของเรา
  - มีค่า 0 คือยิงไปตรงๆแล้วโดนหุ่นยนต์ของศัตรูพอดี
- คลื่น ใช้สำหรับรวบรวมข้อมูลปรกติ
- เซกเมนเตชั่นใช้สำหรับเก็บข้อมูลเรื่องระยะทาง กับความเร็วของลูกกระสุนที่ใช้

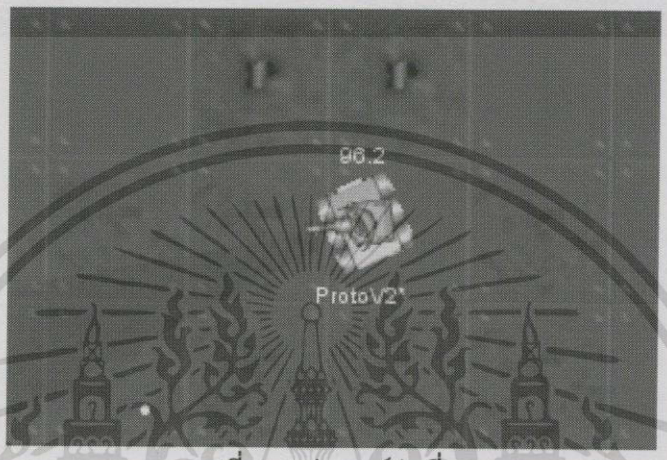


รูปที่ 3.5 หุ่นยนต์รุ่นที่ 1

3.9.2 หุ่นยนต์รุ่นที่ 2

หุ่นยนต์รุ่นที่ 2 โดยภาพรวมปรกติจะเหมือนรุ่นแรกเกือบทุกประการ แต่ในรุ่นนี้จะมีการเพิ่มในส่วนของกระบวนการเก็บค่าเฉลี่ยการเคลื่อนที่(Rolling Averages) เข้าไปเป็นส่วนใช้ในการเล็งยิงของหุ่นยนต์

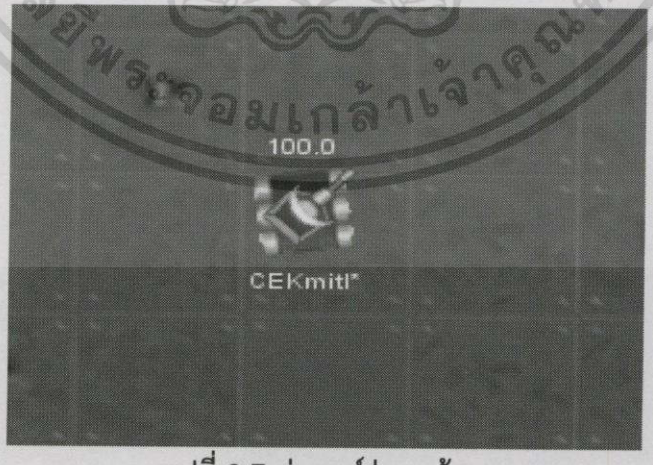
- กระบวนการเก็บค่าเฉลี่ยใช้สำหรับรวบรวมข้อมูลลงให้ int Array



รูปที่ 3.6 หุ่นยนต์รุ่นที่ 2

3.9.3 หุ่นยนต์รุ่นสุดท้าย

เป็นหุ่นยนต์รุ่นสุดท้าย ที่พัฒนาต่อเนื่องมากจากรุ่นที่1 และรุ่นที่2 โดยมีการเพิ่มความสามารถในการเรียนรู้ด้านการหลบหลีกกระสุน ของหุ่นยนต์ศัตรู ด้วยทฤษฎีการเล็งแบบคาดเดาและจดจำซึ่งประกอบไปด้วยตัวประกอบการคาดเดา (GuessFactor) เข้ามาประยุกต์



รูปที่ 3.7 หุ่นยนต์รุ่นสุดท้าย

บทที่ 4
การทดลองและผลการทดลอง

จากทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ (GuessFactorTargeting) ได้นำ มาใช้ในการทดลองกับหุ่นยนต์ที่พัฒนาขึ้นเป็นรุ่นสุดท้าย ซึ่งก็คือ CEKmitl และได้นำไปต่อสู้กับหุ่นยนต์ตัวอื่นๆ ทั้งแบบหุ่นยนต์ที่เป็นพื้นฐานทั่วไป ที่มีให้ทดลองใช้อยู่ในเกมโรโบโค้ดอยู่แล้ว และแบบสุดท้ายคือหุ่นยนต์ที่มีผู้พัฒนาไว้อยู่แล้วนำมาใช้ทดสอบ

4.1 แผนการทดลอง

4.1.1 กำหนดหุ่นยนต์ศัตรูที่จะนำมาทดลอง

ได้มาการกำหนดศัตรูที่จะนำมาทดสอบการเรียนรู้ของเครื่อง ด้วยทฤษฎีการเล็งแบบคาดเดาและจดจำโดยมีการรายชื่อหุ่นยนต์ที่จะนำมาทดลองดังนี้

- อัลฟา (Alpha)
- แบล็คไอซ์ (BlackIce)
- ดาเล็ค (Dalek)
- ดีเฟนเดอร์ (Defender)
- เฟรเดอริค (Frederick)
- โลคูตัส (Locutus)
- วูล์ฟเวอรีน (Wolverine)

4.1.2 กำหนดจำนวนรอบต่อตา

ได้มีการกำหนดจำนวนรอบที่ใช้ในการทดลองไว้ คือ10,000 รอบ ต่อ ตา

4.1.3 กำหนดสิ่งที่ต้องการจะทดลอง

ในการทดลอง เราจะทำการทดสอบ ทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ (GuessFactorTargeting) เมื่อนำไปใช้แล้ว หุ่นยนต์ที่ทดลองจะสามารถเรียนรู้การเล็งยิงหุ่นยนต์ของศัตรูโดน หรือไม่อย่างไร

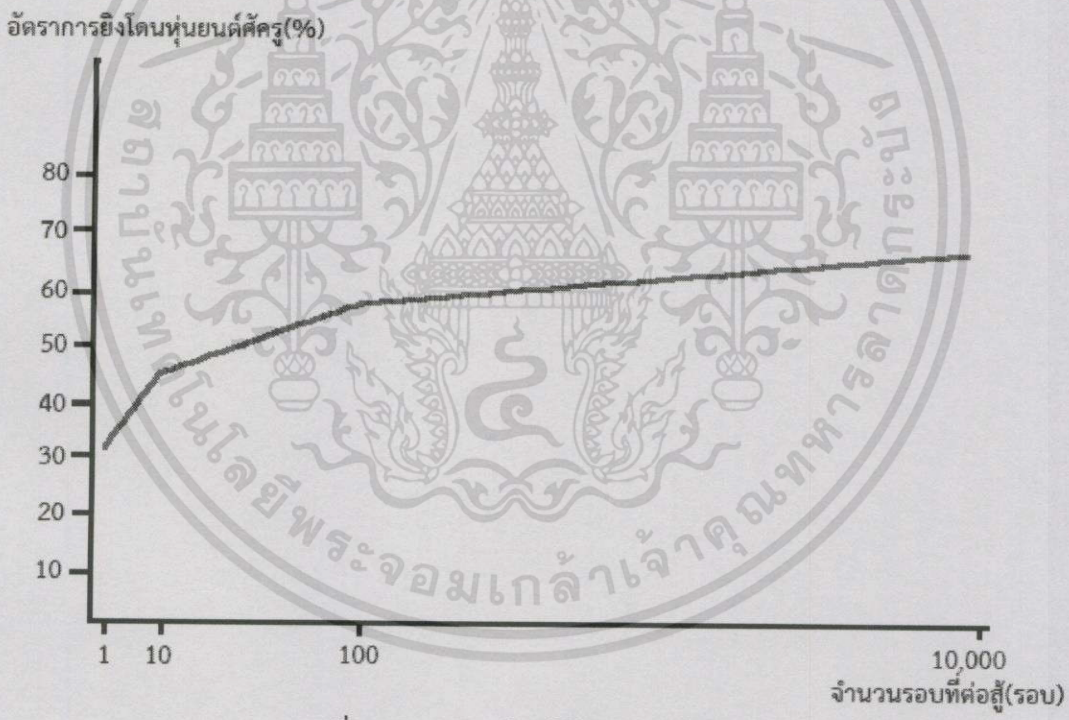
4.2 ผลการทดลอง

ผลการรันโปรแกรมโรโบโค้ด ที่ใช้ทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ (GuessFactorTargeting)ผ่านทางส่วนติดต่อกับผู้ใช้ โดยมีรายละเอียดดังนี้

1.ฮาร์ดแวร์ที่ใช้ในการดำเนินการเรียนรู้ของปัญญาประดิษฐ์เป็นคอมพิวเตอร์ระบบปฏิบัติการวินโดวส์ 7 ที่มีโปรเซสเซอร์ Intel Xeon E3-1275V2 3.5GHz 64bit-OS มีแรม 8.0GB

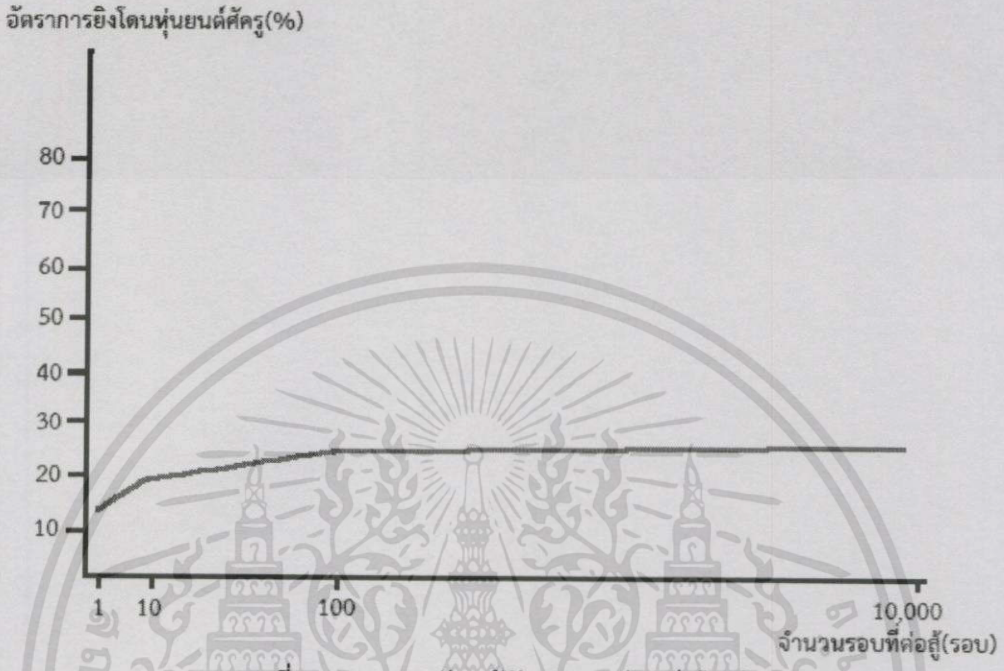
2.ซอฟต์แวร์ที่ใช้เป็นอีคลิปส์ แสตนดาร์ด 4.3(Eclipse Standard 4.3) ที่มีการติดตั้งไลบรารีโรโบโค้ด และซอร์ฟแวร์โรโบโค้ด

4.2.1 ผลการทดลองกับหุ่นยนต์อัลฟา(Alpha)



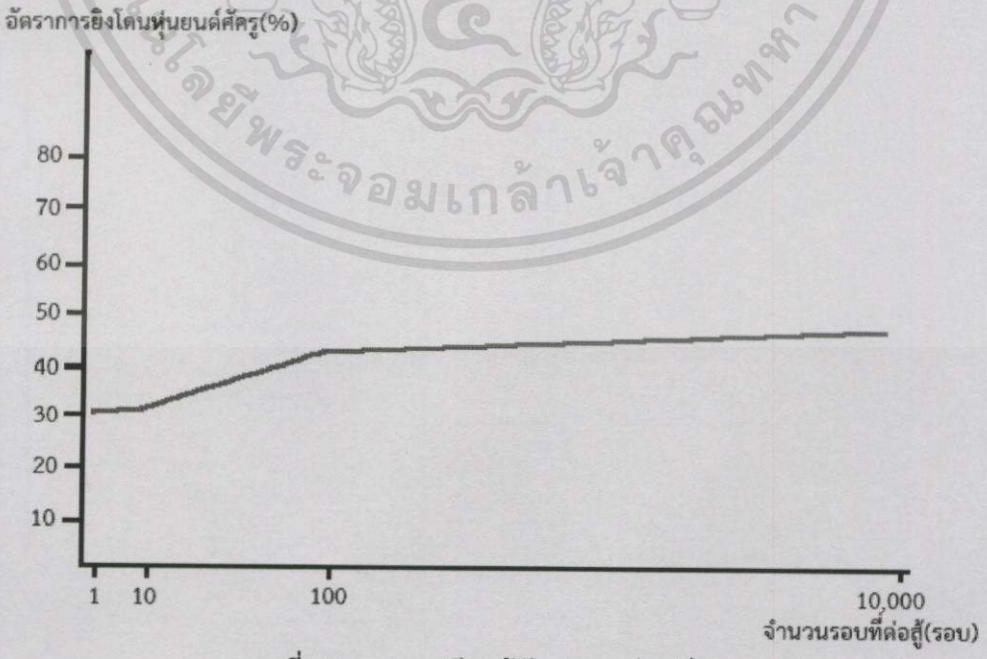
รูปที่ 4.1 ผลการเรียนรู้กับหุ่นยนต์อัลฟา

4.2.2 ผลการทดลองกับหุ่นยนต์แบล็คไอซ์ (BlackIce)



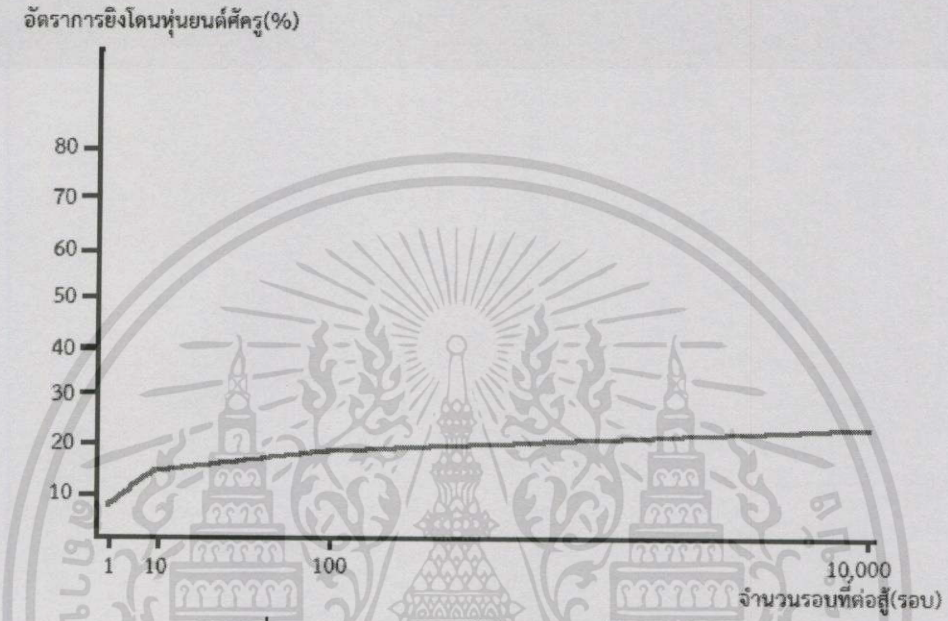
รูปที่ 4.2 ผลการเรียนรู้กับหุ่นยนต์แบล็คไอซ์

4.2.3ผลการทดลองกับหุ่นยนต์ดาเร็ค (Dalek)



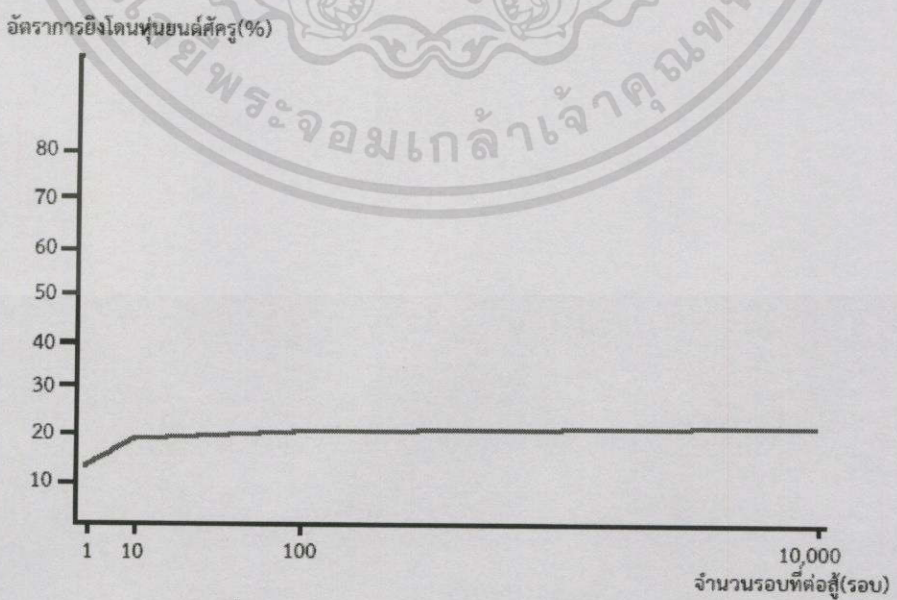
รูปที่ 4.3 ผลการเรียนรู้กับหุ่นยนต์ดาเร็ค

4.2.4 ผลการทดลองกับหุ่นยนต์ดีเฟนเดอร์ (Defender)



รูปที่ 4.4 ผลการเรียนรู้กับหุ่นยนต์ดีเฟนเดอร์

4.2.5 ผลการทดลองกับหุ่นยนต์เฟรเดอริค (Frederick)



รูปที่ 4.5 ผลการเรียนรู้กับหุ่นยนต์เฟรเดอริค

4.2.6 วิเคราะห์ผลการทดลอง

จากการทดลองพบว่าการเรียนรู้ของปัญญาประดิษฐ์ด้วยทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ (GuessFactorTargeting) ทดสอบในเกมโรโบโค้ด จะขึ้นกับปัจจัย 2 ปัจจัยด้วยกัน ปัจจัยแรกคือ ปัจจัยภายนอกหรือสภาพแวดล้อมในการเรียนรู้หากผู้เล่นฝ่ายตรงข้ามมีความสามารถที่เหนือกว่ามากจะทำให้การเรียนรู้มีประสิทธิภาพมากขึ้นไปด้วยเพราะปัญญาประดิษฐ์ด้วยทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ (GuessFactorTargeting) จะพยายามเรียนรู้จากสภาพแวดล้อมเพื่อเอาชนะฝ่ายตรงข้ามซึ่งมีความฉลาดกว่าตนสูงจึงเกิดแรงผลักดันให้ต้องปรับปรุงองค์ความรู้ที่มีอย่างมาก และในทางกลับกันหากผู้เล่นฝ่ายตรงข้ามมีความฉลาดไม่มากหรือน้อยการเรียนรู้ที่จะพัฒนาตนก็เป็นไปได้อย่างช้าเนื่องจากตัวปัญญาประดิษฐ์ด้วยทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ (GuessFactorTargeting) สามารถใช้องค์ความหรือประสบการณ์อันน้อยนิดก็สามารถเอาชนะได้อย่างง่ายดายจึงทำให้การปรับปรุงองค์ความรู้เป็นไปในทางที่ช้าและอาจไม่มีประสิทธิภาพ ปัจจัยที่สองคือจากในภายก็คือตัวปัญญาประดิษฐ์ด้วยทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ (GuessFactorTargeting) จะเรียนรู้ได้อย่างมีประสิทธิภาพนั้นก็ขึ้นอยู่กับผู้พัฒนาว่ามีการจัดสรรเครื่องมือในการเรียนรู้สำหรับตัวหุ่นยนต์มีประสิทธิภาพเพียงใด ในที่นี้คือสูตรต่างๆ ที่ใช้ในการคำนวณระยะทาง หรือความเร็วเป็นต้น แม้กระทั่งข้อมูลที่เก็บได้จากศัตรู แล้วนำมาใช้ก็เช่นเดียวกัน

บทที่ 5
บทสรุปและข้อเสนอแนะ

5.1 สรุป

การเรียนรู้หุ่นยนต์ในเกมโรโบโค้ดด้วยทฤษฎีรูปแบบการเล็งแบบคาดเดาและจดจำ (GuessFactorTargeting) เป็นหลักการทางการเรียนรู้ของเครื่องที่เลียนแบบพฤติกรรมจากการเรียนรู้ของสิ่งมีชีวิตอย่างหนึ่ง ตัวอย่างเช่นการ เล็งยิงวัตถุ ที่เคลื่อนที่อยู่ เราก็จะต้องทำการเล็งยิงล่วงหน้าไปในทิศทางที่ ที่วัตถุนั้นกำลังเคลื่อนที่ไปในทิศนั้น โดยหากเราไม่มีข้อมูลเลย ก็ต้องทำการสุ่มยิงล่วงหน้าไปเรื่อยๆ ว่าล่วงหน้าไปเท่านี้จะโดนหรือไม่ แล้วทำการเรียนรู้จากสิ่งที่ได้ลองไปปรับเปลี่ยนการคาดการของเราไปเรื่อยๆ เพื่อให้ได้ผลที่ดี ที่สุดออกมา และหากวัตถุนั้น มีพฤติกรรมที่เปลี่ยนแปลงได้ เช่นไปทิศเหนือที ไปทิศใต้ที ซึ่งมีรูปแบบที่แน่นอนอยู่แล้วว่าวัตถุนั้นจะมีโอกาสไปทิศเหนือ 70% และไปด้านซ้าย 30% แต่เรานั้นมิได้มีข้อมูล เราก็ต้องทำการสุ่มยิง เพื่อหาความถี่ของวัตถุว่ามีโอกาสไปในทิศไหนมากกว่ากัน และหากเราทำการสังเกตรวบรวมพฤติกรรมเหล่านี้เพื่อมาเป็นข้อมูลพื้นฐานก่อนแล้ว ก็จะทำให้การเล็งยิงของเรานั้นมีประสิทธิภาพมากขึ้นไปอีกซึ่งจะเป็นการอาศัยการเก็บข้อมูล ด้านความเร็วในการเคลื่อนที่, ทิศทางการเคลื่อนที่ และ ตำแหน่งของหุ่นยนต์ศัตรู เพื่อนำมาทำสถิติ ว่าหุ่นยนต์ ของเราจะยิงกระสุนโดนหุ่นยนต์ของศัตรูด้วยมุมการเล็ง แบบไหน และตำแหน่งของเราต้องเป็นอย่างไร โดยหุ่นยนต์ของเราจะเรียนรู้ และปรับตัวตามสถิติที่มีการ ยิงโดนหุ่นยนต์ศัตรูไปเรื่อยๆ นั่นเอง

5.2 ปัญหาอุปสรรคและแนวทางการแก้ไข

1. ทฤษฎีที่ทำการศึกษาไม่มีกฎเกณฑ์ที่ตายตัวว่าจะต้องทำการออกแบบหรือเขียนโปรแกรมอย่างไร เพราะมีแค่รูปแบบและวิธีคิดเท่านั้น ซึ่งมีแนวทางการแก้ปัญหาคือต้องศึกษาแนวคิดให้เข้าใจและสมาชิกในกลุ่มช่วยกันเรียบเรียงวิธีการการออกแบบและเขียนโปรแกรมการเรียนรู้ตามที่ได้ทำการศึกษามาแล้ว

2.ในการทดลองหาหุ่นยนต์ศัตรูที่มีรูปแบบการเคลื่อนไหวตายตัวได้ยาก เพราะโรโบโค้ด เป็นเกมที่ต้อง อาศัยการเรียนรู้ของเครื่องเข้ามาช่วยอยู่แล้ว ซึ่งก็หมายความว่าหุ่นยนต์ศัตรูก็ต้องมีความสามารถในการเรียนรู้อยู่เช่นกัน

3.ในส่วนการคำนวณบางอย่างต้องใช้คณิตศาสตร์เข้ามาคำนวณเพื่อความแม่นยำที่มากขึ้น

5.3ข้อเสนอแนะ

1. ออกแบบ และประยุกต์ใช้ทฤษฎีที่สามารถใช้ได้กับหลายๆ สถานการณ์
2. นำผลการทดลองจากทฤษฎีต่างๆ มาเปรียบเทียบ เพื่อหาการประยุกต์ใช้ที่เหมาสม
3. ควรหาหุ่นยนต์คู่ต่อสู้ที่หลากหลาย เพื่อการเรียนรู้หลายๆ รูปแบบ

บรรณานุกรม

[1] EthemAlpaydin. **Introduction to Machine Learning.** Cambridge, MA : MIT Press, 2004.

[2] HojjatAdeli and Shin-Lin Hung. **Machine Learning.** New York : John Wiley & Sons, Inc, 1995.

[3] ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. **ปัญญาประดิษฐ์.**กรุงเทพมหานคร : บริษัท เคทีพี คอมพ์ แอนด์ คอนซัลท์ จำกัด. พ.ศ. 2552.

[4] Kawigi, **"Guess Factor Targeting Tutorial."**[Online]. Available: http://robowiki.net/wiki/GuessFactor_Targeting_Tutorial. 2003.

[5] ABC, **"Dynamic Clustering Tutorial."**[Online]. Available: http://robowiki.net/wiki/Dynamic_Clustering_Tutorial. 2003.

[6] FlemmingLarsen, **"Robocode / Eclipse / Create A Project."**[Online]. Available: http://robowiki.net/wiki/Robocode/Eclipse/Create_a_Project. 2007.

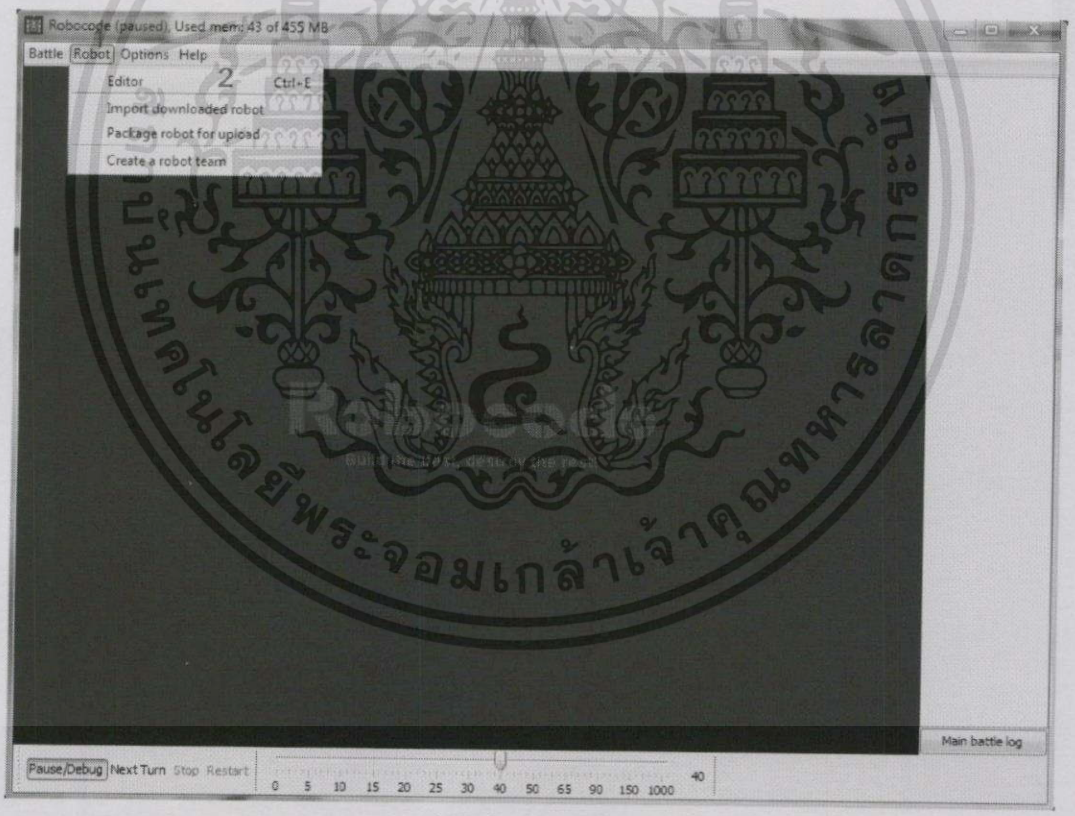
ภาคผนวก ก
เกมโรโบโค้ด

ก.1 หุ่นยนต์ตัวแรก

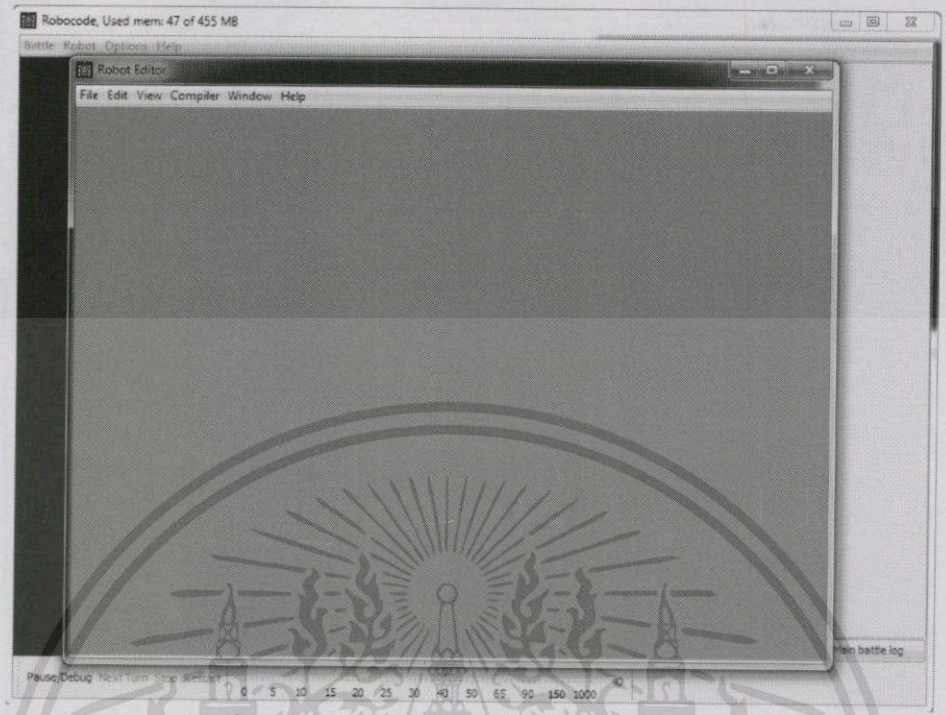
เกมโรโบโค้ดนั้น ตอนที่ติดตั้งโปรแรกม จะมีหุ่นยนต์พื้นญานปรกติทั่วไป ที่เราสามารถใช้โรโบโค้ด อิดิทเตอร์ (Robocode Editor) เพื่อที่จะดูว่าหุ่นยนต์เหล่านี้มีการทำงานอย่างไรเพื่อให้เกิดแนวคิดต่างๆ ในการสร้างหุ่นยนต์ของเราเองขึ้นมาได้

ก.2 โรโบโค้ด อิดิทเตอร์ (Robocode Editor)

ลำดับแรกสุดที่เราจะเปิดโรโบโค้ด อิดิทเตอร์ (Robocode Editor)ได้นั้น เราต้องเริ่มจากตรงส่วนหน้าหลักของโรโบโค้ด จากนั้นคลิดที่เมนู Robot จากนั้นเลือก Editor



ก.1 Robot > Editor



ก.2หน้าตาโรโบโค้ด อิดิทเตอร์ (Robocode Editor)

ก.3 สร้างหุ่นยนต์ตัวใหม่

```
package man;
importrobocode.*;

public class MyFirstRobot extends Robot {
public void run() {
while (true) {
ahead(100);
turnGunRight(360);
back(100);
turnGunRight(360);
        }
    }

public void onScannedRobot(ScannedRobotEvent e) {
fire(1);
    }
 }
```

โดยส่วนที่เราสนใจก็คือส่วนที่เป็นตัวหนาสีเขียว แต่ก่อนอื่นเรามารู้จักส่วนอื่นๆ กันก่อน

```
package man;
```

```
importrobocode.*;

public class MyFirstRobot extends Robot {
public void run() {
     }
 }
```

- Import robocode.*; คือการบอกกับจาวา (JAVA) ว่าเราจะใช้ออบเจ็กRobocode ในหุ่นยนต์เรานะ
- Public class MyFirstRobot extends Robot โดย Robot คือ type และ MyFristRobot คือชื่อของหุ่นยนต์
- Public void run() {} เป็นส่วนที่เกมจะเรียกไปยังmethodrun() เมื่อเริ่มการต่อสู้

ก.4 เริ่มให้หุ่นยนต์เคลื่อนตัวไปสักที่

```
while (true) {
ahead(100);
turnGunRight(360);
back(100);
turnGunLeft(360);
 }
```

โดยในโค้ดนี้จะเป็นการให้หุ่นยนต์เรา

1. เคลื่อนที่ไปข้างหน้า 100 พิกเซล (Pixel)
2. หมุนกระบอกปืนไปทางขวา 360 องศา
3. เคลื่อนที่กลับไปด้านหลัง 100 พิกเซล (Pixel)
4. หมุนกระบอกปืนไปทางซ้าย 360 องศา

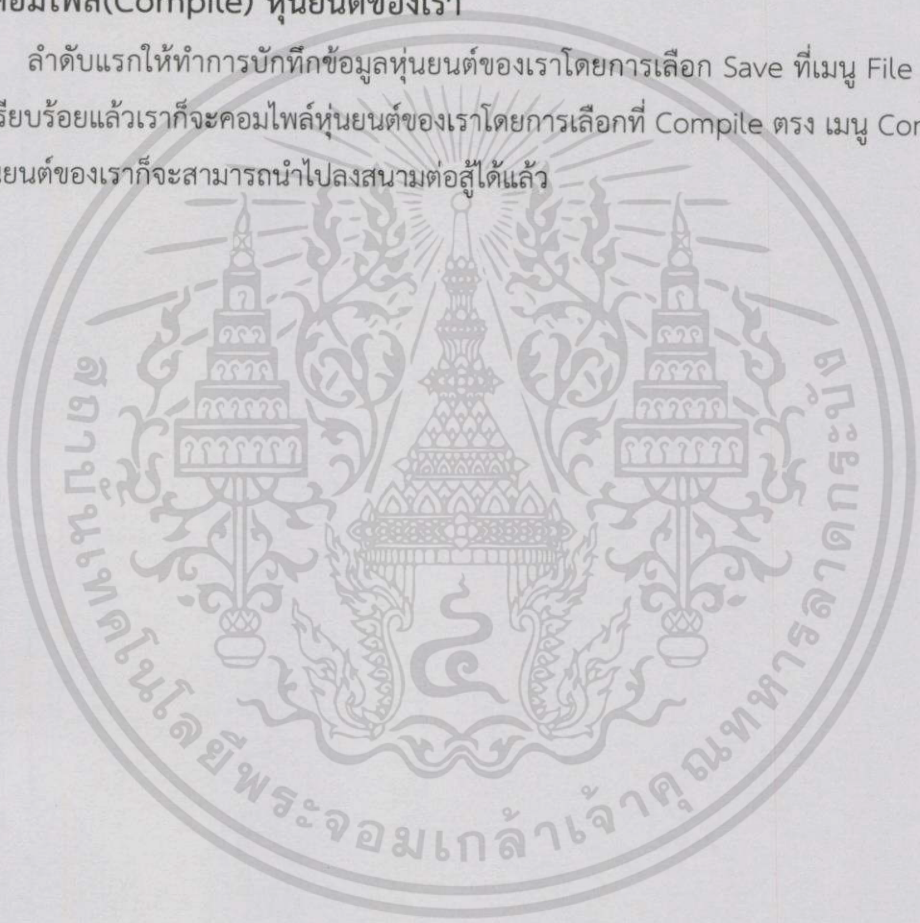
ก.5 การยิงกระสุน

เมื่อหุ่นยนต์เราตรวจหาหุ่นยนต์ศัตรู และต้องการที่จะยิงไปยังหุ่นยนต์ศัตรู

```
public void onScannedRobot(ScannedRobotEvent e) {
fire(1);
 }
```

ก.6 คอมไพล์(Compile) หุ่นยนต์ของเรา

ลำดับแรกให้ทำการบักทึกข้อมูลหุ่นยนต์ของเราโดยการเลือก Save ที่เมนู File พอบักทึกเสร็จเรียบร้อยแล้วเราก็จะคอมไพล์หุ่นยนต์ของเราโดยการเลือกที่ Compile ตรง เมนู Compiler ที่นี้หุ่นยนต์ของเราก็จะสามารถนำไปลงสนามต่อสู้ได้แล้ว



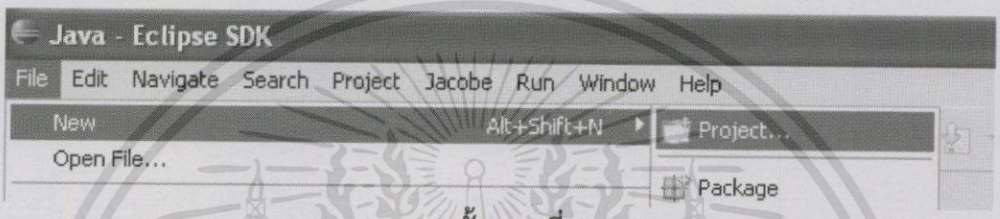
ภาคผนวก ข

ไลบรารี่โรโบโค้ด

ข.1 สร้างโปรเจ็คสำหรับหุ่นยนต์ตัวแรก

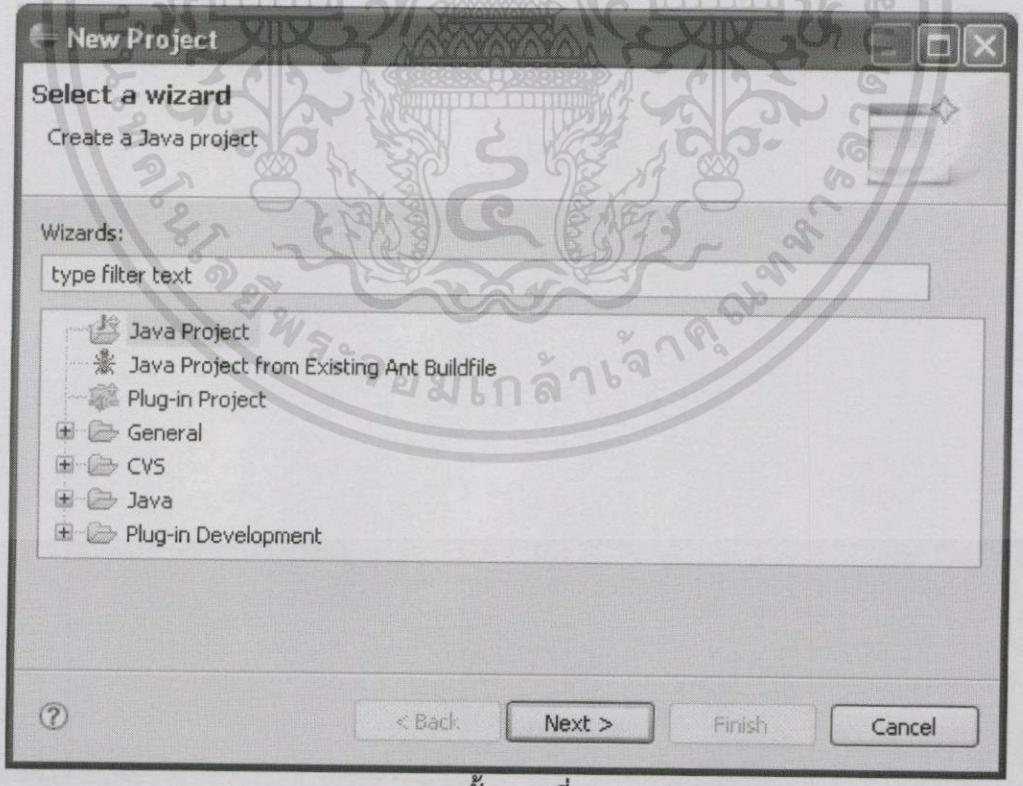
โรโบโค้ดนั้น สามารถที่จะให้ตัว Editor อื่นนอกจาก Editor ของโปรแกรมเองได้ ซึ่งก็คือโปรแกรม Eclipse นั่นเอง ซึ่งมีขั้นตอนดังนี้

เริ่มแรก เลือก File -> New -> Project



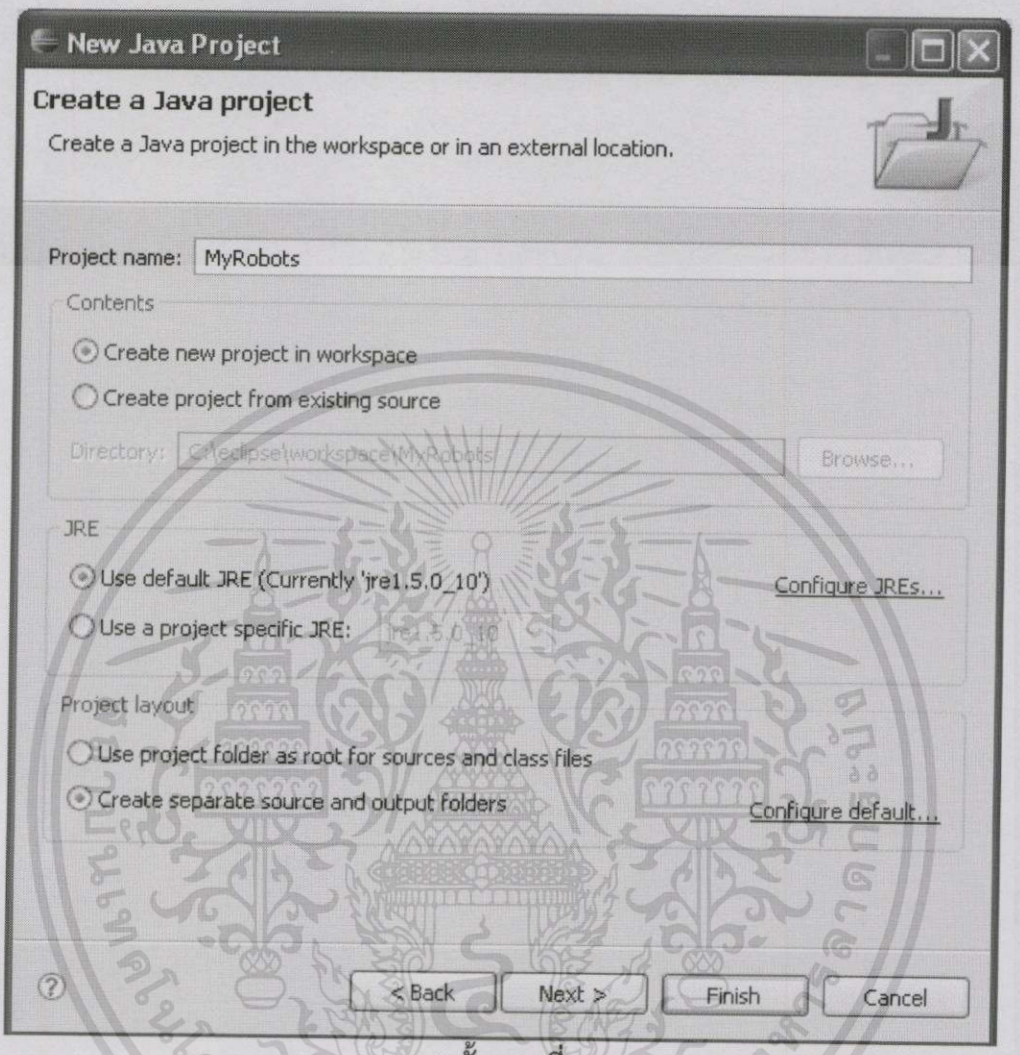
ข.1 ขั้นตอนที่ 1

จากนั้นเลือก Java Project



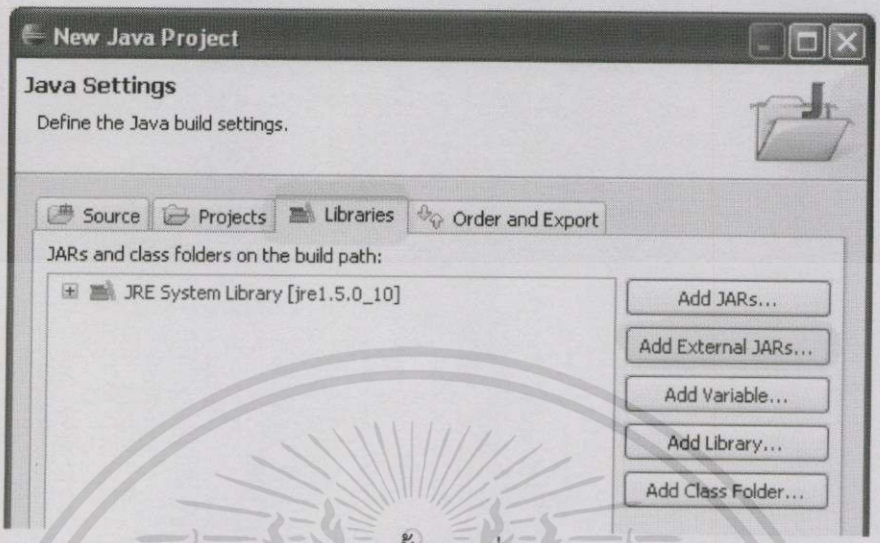
ข.2 ขั้นตอนที่ 2

เขียน Project Name ให้เรียบร้อย จากนั้นกดที่ Next



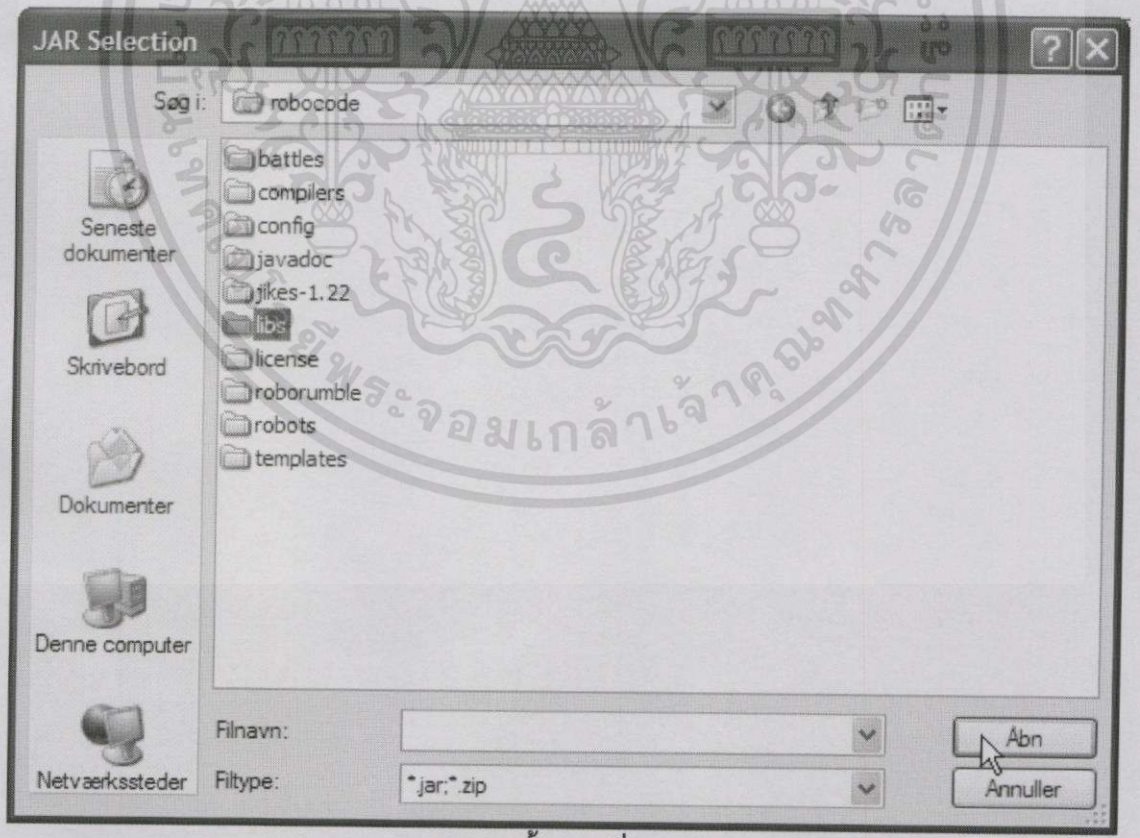
ข.3 ขั้นตอนที่ 3

เลือกแทป Libraries จากนั้นคลิกที่ Add External JARs



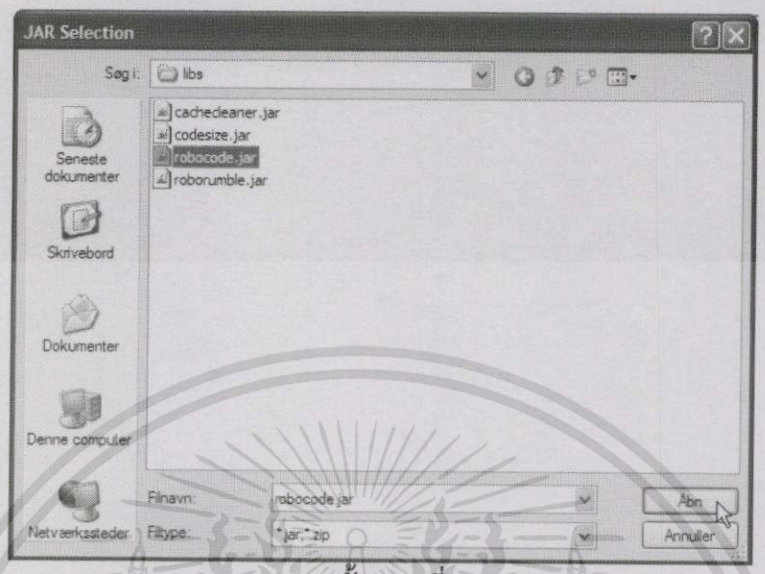
ข.4 ขั้นตอนที่ 4

ให้เลือก Browse ไปยังไดเร็คทอรี่ที่เราติดตั้งเกมโรโบโค้ดไว้ จากนั้นเลือก ศริหและคลิก Add



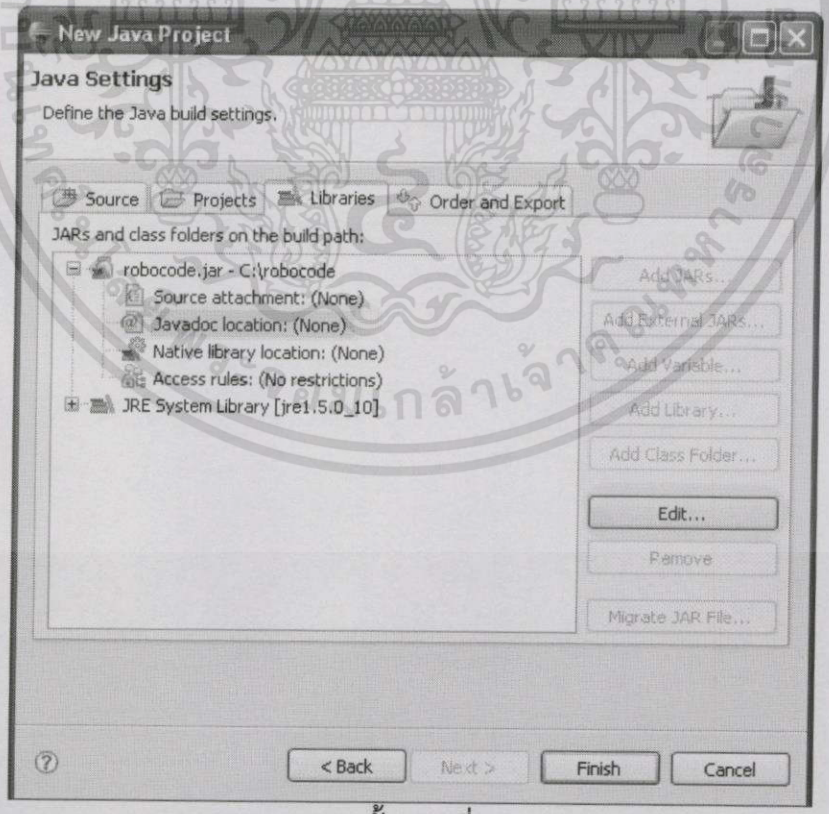
ข.5 ขั้นตอนที่ 5

ต่อมาให้เลือก robocode.jar และจากนั้นก็คลิด open



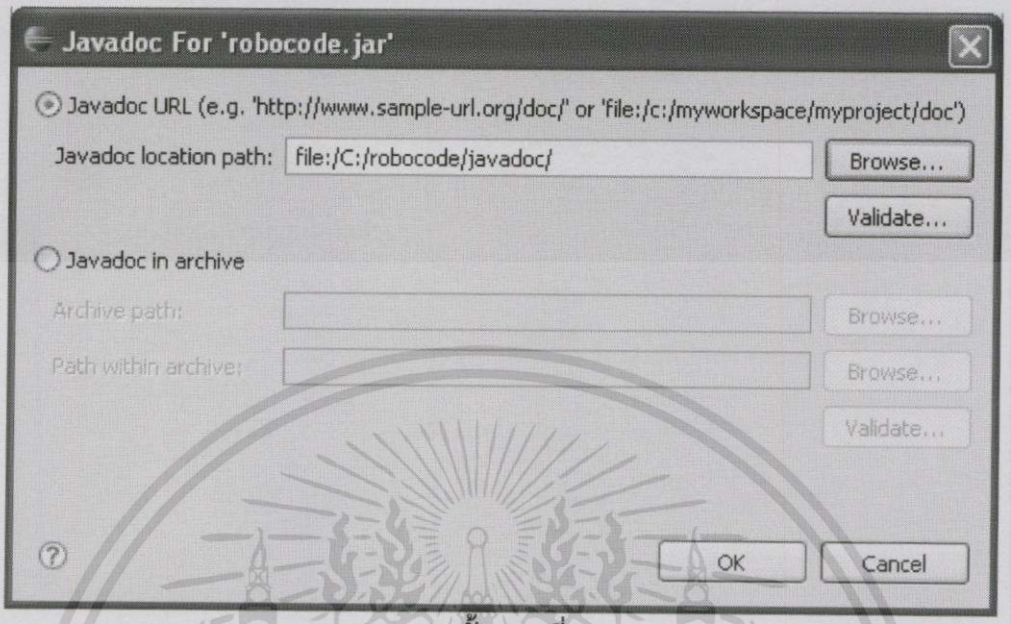
ข.6 ขั้นตอนที่ 6

จากนั้นเราจะต้องเลือกที่ Javadoc location: (None) ที่ใต้ robocode.jar และคลิกที่ Edit



ข.7 ขั้นตอนที่ 7

ต่อมาให้ Browse path ไปยังjavadocในrobocode หรือจะพิมพ์ path ตามรูปด้านล่างเลยก็ได้



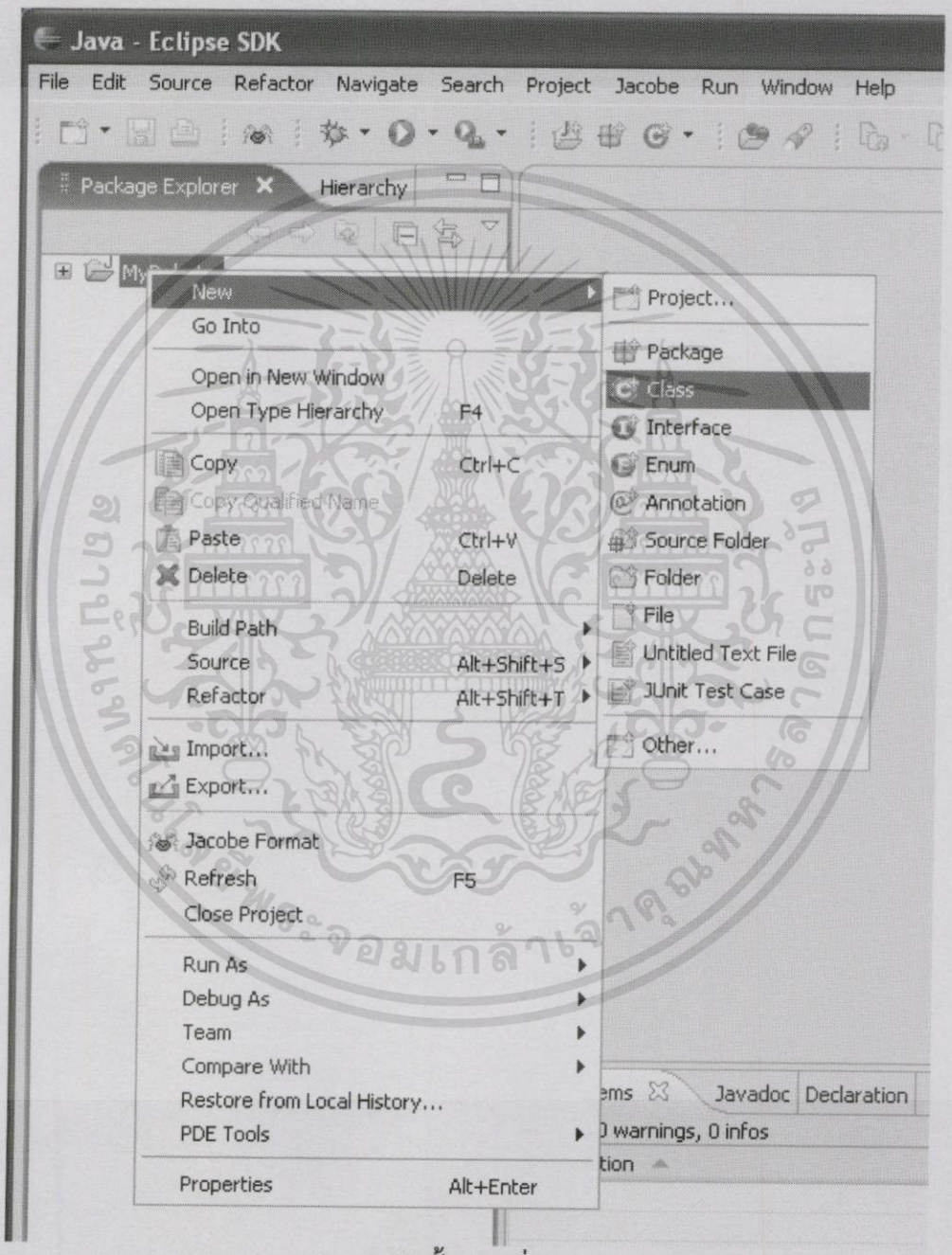
ข.8 ขั้นตอนที่ 8

คลิกที่ OK จากนั้นก็คลิก Finish เพื่อที่จะสร้างโปรเจ็ค

ข.2 สร้างหุ่นยนต์ใน Eclipse

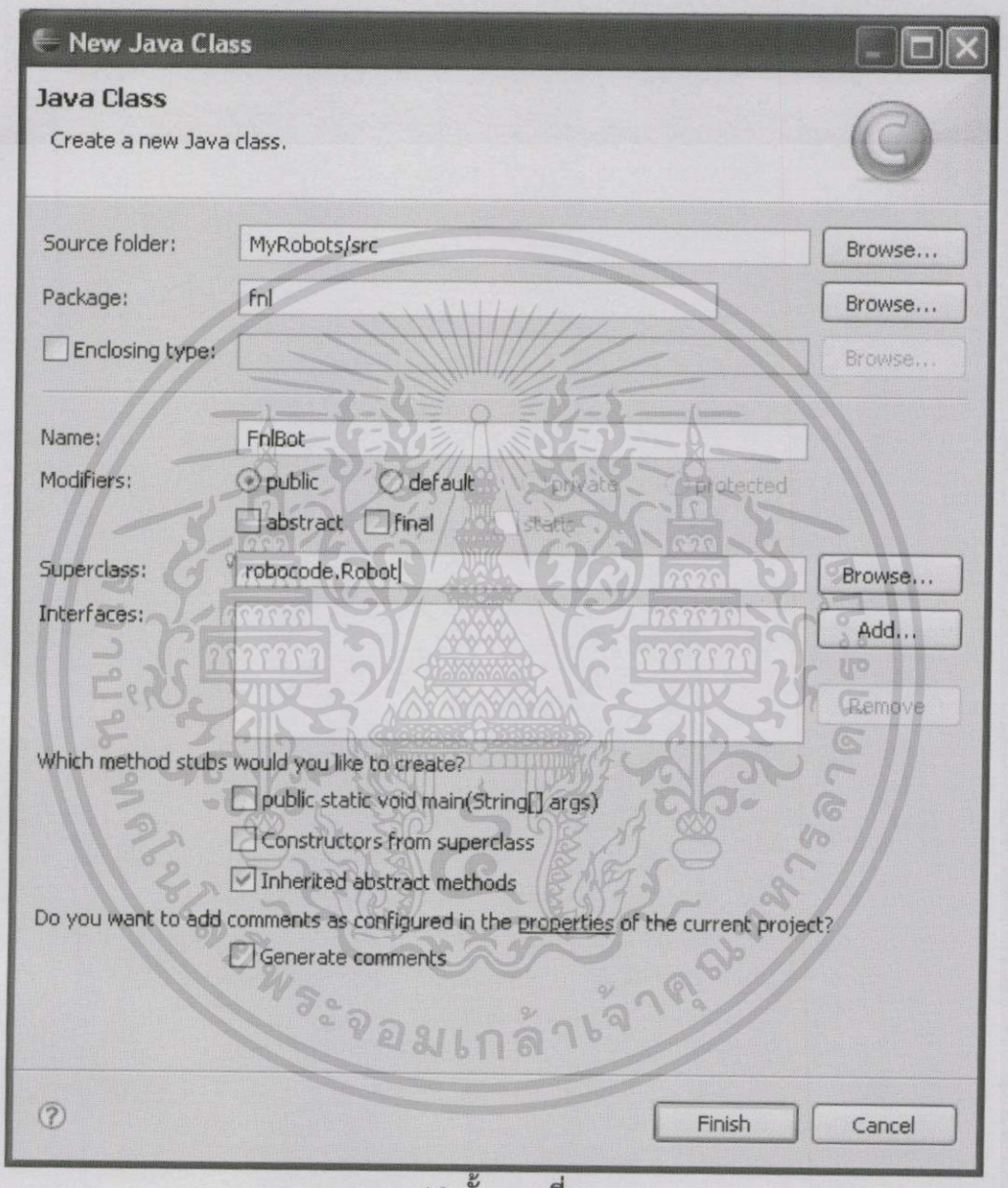
ที่นี้เราจะได้โปรเจ็ค ของเราขึ้นมาแล้ว จากเราเราจะมาเริ่มสร้างหุ่นยนต์กัน

เริ่มแรก ให้คลิกจวาที่โปรเจ็คMyRobots จากนั้นเลือก New -> Class



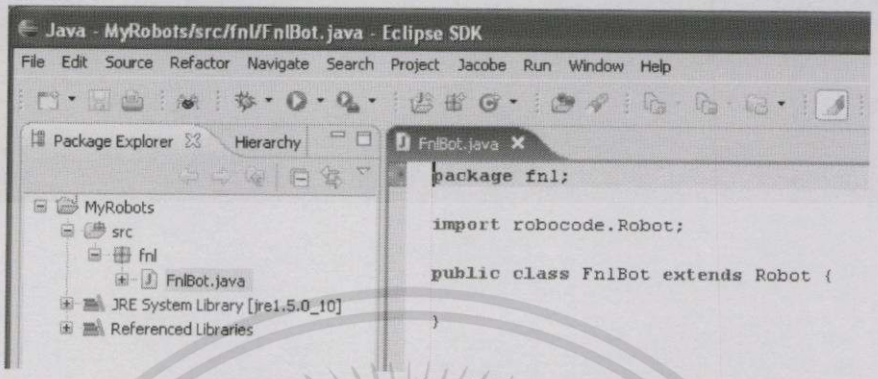
ข.9 ขั้นตอนที่ 1

ต่อมา ให้ใส่ชื่อ Package และชื่อหุ่นยนต์ที่ช่อง Name และเปลี่ยนตรง Superclass เป็น robocode.Robot



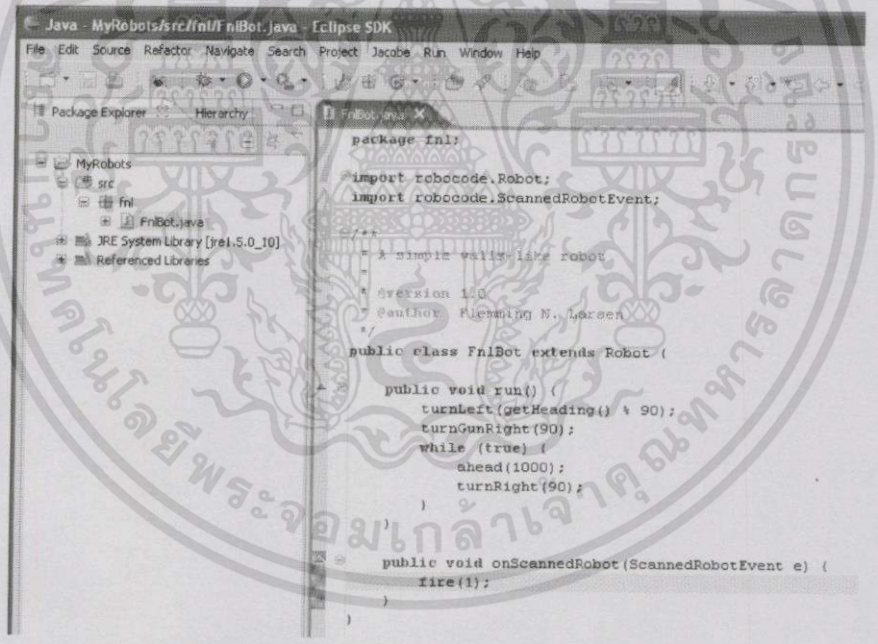
ข.10 ขั้นตอนที่ 2

จากนั้นกด Finish เราก็จะได้คลาสของหุ่นยนต์เรามาเรียบร้อย



ข.11 คลาสของหุ่นยนต์เราในโปรแกรม Eclipse

ที่นี้เราก็จะสามารถเขียนหุ่นยนต์ของเราขึ้นมาได้แล้ว



ข.12 เขียนโค้ดให้หุ่นยนต์ของเราตามต้องการ

โดยเราต้องการบันทึกหุ่นยนต์ของเราตามปรกติคือ File -> Save หรือกด CTRL+S
ซึ่งในตอนนี้เราไม่จำเป็นที่จำต้องคอมไฟล์หุ่นยนต์ของเราแล้ว เพราะเกมโรโบโค้ดจะจัดการส่วนนั้นให้เองแบบอัตโนมัติ