พฤติกรรมการสำรวจในการเรียนรู้แบบเสริมกำลัง โดยประยุกต์ใช้กับเกมออนไลน์หลายผู้เล่น

The Effects of Exploration Behavior in Reinforcement Learning in Multiplayer Online Battle Arena (MOBA) game.

พัชรพล จันทนะ

ศุภวิชญ์ การสมพจน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

พฤติกรรมการสำรวจในการเรียนรู้แบบเสริมกำลัง โดยประยุกต์ใช้กับเกมออนไลน์หลายผู้เล่น

The Effects of Exploration Behavior in Reinforcement Learning in Multiplayer Online Battle Arena (MOBA) game.

พัชรพล จันทนะ

ศุภวิชญ์ การสมพจน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

ปริญญานิพนธ์ปีการศึกษา 2560

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สาถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง พฤติกรรมการสำรวจในการเรียนรู้แบบเสริมกำลัง
โดยประยุกต์ใช้กับเกมออนไลน์หลายผู้เล่น

The Effects of Exploration Behavior in Reinforcement Learning in Multiplayer Online Battle Arena (MOBA) game.

ผู้จัดทำ

1. นาย พัชรพล จันทนะ รหัสนักศึกษา 57010865
2. นาย ศุภวิชญ์ การสมพจน์ รหัสนักศึกษา 57011272

_____ อาจารย์ที่ปรึกษา

(ผศ.อัครเดช วัชระภูพงษ์)

ปัญญาประดิษฐ์สำหรับเกมออนไลน์หลายผู้เล่น

นายพัชรพล	จันทนะ	57010865
นายศุภวิชญ์	การสมพจน์	57011272
ผศ.อัครเดช	วัชระภูพงษ์	อาจารย์ที่ปรึกษา

ปีการศึกษา 2560

บทคัดย่อ

ในบทความนี้ได้มีการประยุกต์ใช้วิธีการในการสำรวจรูปแบบต่างๆ และวิธีการเรียนรู้แบบเสริมกำลังในการศึกษาสภาพแวดล้อมที่ต่อเนื่อง และมีรางวัลล่าช้า ซึ่งจะพบสภาพแวดล้อมในลักษณะนี้ได้ในเกมกลยุทธ์ทั่วๆไป เพื่อหาพฤติกรรมการสำรวจที่เหมาะสม เราได้ใช้เทคนิคการสำรวจในการเรียนรู้แบบเสริมกำลังแบบต่างๆ ได้แก่ ขั้นตอนวิธีแบบละโมบ โดยมีนัยยะอิงจากค่าคงที่ epsilon, การกระจาย Boltzmann, การรบกวนใน action, และการ bootstrapping. เพื่อที่จะหารูปแบบการเรียนรู้ที่เหมาะสมที่สุด เราจึงได้พัฒนาอัลกอริธึมการเรียนรู้แบบเสริมกำลังโดยประยุกต์ใช้กับเกม Dota2 ซึ่งเป็นเกม MOBA ยอดนิยมในเวลาปัจจุบัน เนื่องจากความสำเร็จของ Double Deep Q-Network (DDQN) อัลกอริทึมที่สามารถใช้ในปริภูมิสถานะที่มิติที่ซับซ้อนได้ และสามารถทำงานได้ดีในปริภูมิการกระทำที่ไม่ต่อเนื่อง เราจึงเลือกใช้กับการ Lasthit ซึ่งเป็นหนึ่งในงานๆหนึ่งของเกมนี้ ยิ่งไปกว่านั้นเราได้ใช้ Deep Deterministic Policy Gradient (DDPG) ในการแก้ปัญหาที่มีลักษณะของปริภูมิการกระทำที่ต่อเนื่อง และเรายังได้ใช้อัลกอริทึมนี้ในการปิดกั้นการเคลื่อนที่ของ Creep (ประเภทของตัวละครในเกม) นอกจากวิธีการสำรวจต่างๆ และ อัลกอริทึมทั้งสองที่ใช้ในการเรียนรู้แล้ว เรายังได้ใช้เทคนิค replay memory จากผลการทดลองแสดงให้เห็นว่าการที่เราเลือกอัลกอริธึมการเรียนรู้แบบเสริมกำลังและเลือกพฤติกรรมการสำรวจที่เหมาะสมสามารถใช้แก้ปัญหาในปัญหาย่อยๆทั้งสองของเกม Dota2 ได้

The Effects of Exploration Behavior in Reinforcement Learning in Multiplayer Online Battle Arena (MOBA) game.

Mr.Patcharapon	Jantana	57010865
Mr.Supawit	Kansompoj	57011272
Asst.Prof.Akkradach	Watcharapupong	Advisor

Academic 2017

ABSTRACT

In this paper, we apply some exploration methods with Reinforcement Learning method to continuous delayed reward environment, that was found in some strategy game. To find suitable exploration behavior, we implement some technics such as classical epsilon greedy, Boltzmann distribution, action space noise adding, and bootstrapping. In order to find the best solution for that exploration method. We also implement reinforcement learning algorithm working with Dota2 ,which is the popular one MOBA game in this time. Because of the success of Double Deep Q-Network (DDQN) in high dimensionally state space, perform very well over discrete action space, was chosen to used with a Lasthit Creep task. More over the accomplishment of Deep Deterministic Policy Gradient (DDPG) in continuous action space problem is remarkable result and we implemented this with Creep blocking task. All of them was used with memory replay technics and a few exploration method. The experiment show that our choosing reinforcement learning algorithm and choosing exploration behavior. Can complete our Dota2's subtask.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สามารถสำเร็จลุล่วงได้ด้วยดีด้วยความช่วยเหลือจากหลายฝ่าย โครงงานฉบับนี้สำเร็จลงไม่ได้ หากปราศจากการช่วยเหลือจากบุคคลเหล่านี้

อาจารย์ที่ปรึกษา ผศ.อัครเดช วัชระภูพงษ์ ที่ให้คำปรึกษา แนะนำ และให้ความช่วยเหลือในการทำโครงงาน

อาจารย์และบุคลากรต่าง ๆ ในภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้ให้คำแนะนำและความรู้ตลอดการศึกษา

ท้ายที่สุดขอขอบคุณบิดา มารดา และครอบครัวที่เลี้ยงดู สั่งสอน สนับสนุนโอกาสในการศึกษา และคอยให้กำลังใจเสมอมา

นายพัชรพล จันทนะ

นายศุภวิชญ์ การสมพจน์

สารบัญ

หน้า

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมา

การเรียนรู้แบบเสริมกำลัง (Reinforcement Learning) เป็นอัลกอริทึมที่กำลังเป็นที่น่าจับตามองมากในขณะนี้ เป็นผลมาจากความนิยมและความตื่นตัวทางด้านการเรียนรู้ของเครื่องจักร (Machine Learning) ที่เพิ่มขึ้นอย่างก้าวกระโดดในช่วง 3-4 ปีที่ผ่านมา มีงานวิจัยมากมายที่วิจัยเกี่ยวกับการเรียนรู้แบบเสริมกำลัง และหนึ่งในนั้นคือ การนำความรู้ทางด้านนี้มาสร้างระบบปัญญาประดิษฐ์เพื่อแก้ปัญหาภายในเกม ซึ่งในปัจจุบันยังไม่มีเกมใดที่สามารถสร้างระบบปัญญาประดิษฐ์ดังกล่าวมาใช้ในเชิงพาณิชย์ได้ เนื่องจากมีปัญหาในด้านการสร้างสภาพแวดล้อม (Environment) เพื่อฝึกฝนโมเดล, ทรัพยากรของเกมที่ต้องใช้ค่อนข้างมาก และเวลาในการฝึกฝนที่ค่อนข้างนาน ทำให้ไม่ทันต่อการใช้งานเป็นต้น

หนึ่งในปัญหาที่ทำให้ใช้เวลาฝึกฝนที่นานมาจากขั้นตอนที่เรียกว่าการสำรวจในการเรียนรู้แบบเสริมกำลัง เนื่องจากเป็นขั้นตอนที่ต้องลองผิดลองถูก ซึ่งมีอัลกอริทึมที่ใช้เพื่อการสำรวจมากมาย หากเรากำหนดเกณฑ์หรือเลือกใช้อัลกอริทึมได้ไม่เหมาะกับงาน อาจทำให้ใช้เวลาในการฝึกนานขึ้น โดยเฉพาะในช่วงแรกของการฝึก การสุ่มแบบไม่มีรูปแบบอาจจะไม่ใช่วิธีที่เหมาะสม

ทางผู้จัดทำจึงต้องการวิจัยเกี่ยวกับการเลือกใช้อัลกอริทึมการสำรวจเพื่อแก้ปัญหาทางด้านเกม โดยผู้จัดทำเลือกเกม Dota 2 เพราะมีเครื่องมือในการพัฒนาบอทให้ และมีกลุ่มผู้พัฒนาอยู่จำนวนหนึ่งอยู่แล้ว ทำให้ง่ายต่อการพัฒนาระบบควบคุมบอท นอกจากนี้ยังทำให้สามารถมุ่งเน้นไปที่การพัฒนาด้าน Machine Learning ในเกมดังกล่าวได้ โดยผู้จัดทำจะใช้การเรียนรู้แบบเสริมกำลังแก้ปัญหาสองอย่างคือ ปัญหา Lasthit แก้ด้วย Double Deep Q Nural Network (DDQN) ที่สามารถทำงานได้ดีในปริภูมิการกระทำที่ไม่ต่อเนื่อง (Discrete Action Space) และปัญหา Creep blocking แก้ด้วยวิธี Deep Deterministic Policy Gradient (DDPG) ที่สามารถการแก้ปัญหาที่มีลักษณะของปริภูมิการกระทำที่ต่อเนื่อง (Continuous Action Space) ได้ โดยแต่ละปัญหาจะทดลองใช้อัลกอริทึมในการสำรวจหลายๆรูปแบบ เพื่อตรวจสอบผลว่าอัลกอริทึมลักษณะใด เหมาะสมกับปัญหาเหล่านั้น

1.1 จุดมุ่งหมายและวัตถุประสงค์

1) เพื่อเปรียบเทียบว่าอัลกอริทึมในการสำรวจลักษณะไหนเหมาะสมในการแก้ปัญหา Creep blocking ในเกม Dota2 มากกว่ากัน ระหว่าง Ornstein-Uhlenbeck process และ การใช้การกระจาย Uniform โดยใช้ DDPG
2) เพื่อเปรียบเทียบว่าอัลกอริทึมในการสำรวจลักษณะไหนเหมาะสมในการแก้ปัญหา Lasthit ในเกม Dota2 มากกว่ากัน ระหว่าง Epsilon greedy และ Boltzmann distribution โดยใช้ DDQN

1.2 ประโยชน์ที่ที่คาดว่าจะได้รับ

1) สามารถนำอัลกอริทึมในการสำรวจที่เหมาะสมไปดัดแปลงเพื่อแก้ปัญหาอื่นๆในเกม Dota2 หรือเกมอื่นๆ รวมทั้งปัญหาต่างๆที่ใช้การเรียนรู้แบบเสริมกำลังได้

1.3 ขอบเขตของโครงงาน

1) เลือกใช้อัลกอริทึม DDQN และ DDPG ในการทดลองเท่านั้น
2) สำหรับปัญหา Lasthit รองรับเฉพาะฮีโร่ Sniper เท่านั้น
3) สำหรับปัญหา Creep blocking เลือกใช้เฉพาะฮีโร่ Shadow fiend

1.4 ขั้นตอนการดำเนินงาน

1) ศึกษาทฤษฎีของ Reinforcement Learning (RL)
   a. ศึกษาความรู้เบื้องต้นของ Machine Learning
   b. ศึกษาตัวอย่างปัญหาต่างๆที่สามารถใช้ RL มาแก้ปัญหาได้ พร้อมทั้งวิธีการแก้
   c. ศึกษางานวิจัยที่เกี่ยวข้อง
2) ศึกษา Library ในภาษา python ที่จำเป็นต้องใช้
   a. ศึกษา Tensorflow และ Keras สำหรับทำ RL
   b. ศึกษา flask สำหรับทำ server
3) ศึกษาการควบคุมตัวละครและสิ่งต่างๆภายในเกม Dota 2
4) เลือกวิธีการทาง Machine Learning เพื่อที่จะนำมาแก้ปัญหาในเกม Dota 2

5) ทดลองสร้าง RL สำหรับ Dota 2
   a. ออกแบบวิธีการทดลอง
   b. ออกแบบ input , output และ hypermarameter ต่างๆ ของ model
   c. สร้างส่วนอัพเดท model ด้วยภาษา python โดยจะทำงานบน server
   d. สร้าง environment และ agent โดยการเขียน vscript และ ภาษา Lua ในเกม
   e. เทรน model
   f. ตรวจสอบผลลัพท์ โดยการดู reward ที่ได้
6) ทดลองปรับเปลี่ยนอัลกอริทึมการสำรวจ
7) วัดผลและประเมินผล

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 ความรู้ทั่วไปเกี่ยวกับเกม Dota 2

Dota 2 เป็นเกมประเภท MOBA (Mutiplayer Online Battle Arena) ซึ่งถูกพัฒนาขึ้นโดยบริษัท Value Corporation ซึ่งตัวเกมจะมีลักษณะการเล่นหนึ่งตาจะถูกแบ่งเป็น 2 ฝ่ายได้แก่ ฝั่ง Radiant และ ฝั่ง Dire ซึ่งในแต่ละทีมจะต้องป้องกันไม่ให้ฐานตัวเองถูกทำลาย โดยผู้เล่นแต่ละคนต้องทำการควบคุมตัวละคร ซึ่งมักถูกเรียกว่า “ฮีโร่” โดยฮีโร่แต่ละตัวนั้นจะมีความสามารถที่แตกต่างกันและ มีสไตลการเล่นที่ต่างกัน ในระหว่างเกมการเล่นจะมีการเก็บสะสมค่าประสบการณ์ (Experience points) และ Items เพื่อใช้ในการต่อสู้กับฮีโร่ฝั่งตรงข้าม และทีมที่ชนะก็คือทีมที่สามารถทำลายสิ่งก่อสร้างที่เรียกว่า “Ancient” ของฝ่ายตรงข้ามได้ก่อน [1]

ก) ข)

รูปที่ 2.1 Ancient ของแต่ละฝ่าย

ก) Ancient ของฝั่ง Radiant

ข) Ancient ของฝั่ง Dire

2.1.2 โหมดที่ใช้ในการเล่นเกม (Dota 2 Environment)

โหมดต่างๆในเกมนั้น มีหลายโหมดมากมายซึ่งสามารถศึกษาเพิ่มได้จาก [2] ซึ่งในที่นี้ขอกล่าวถึงเฉพาะโหมดที่ใช้ในการศึกษาในงานวิจัยนี้ สำหรับโหมดที่ใช้นั้นคือ Custom Game เนื่อง จาก Custom Game รองรับการพัฒนาจาก Community ซึ่งสามารถพัฒนารูปแบบของเกม และกฎต่างๆ ได้อย่างอิสระ ในการพัฒนานั้น ต้องทำการพัฒนาผ่านทาง Dota 2 Workshop Tools และใช้ Dota 2 Bot script API ซึ่งรองรับภาษา LUA ในการพัฒนาส่วนที่ติดต่อควบคุมเกม

รูปที่ 2.2 แผนที่ภายในเกม และเลนการวิ่งของ Creeps

จากรูปที่ 2.3 แสดงให้เห็นถึงความซับซ้อนของ Environment ในเกม Dota 2 ได้มีหลายๆสิ่งเป็นตัวประเมินวัดว่าตอนนี้เราได้เปรียบหรือเสียเปรียบฝั่งตรงข้ามอยู่ และตัดสินใจทำอะไรในเกม ไม่ได้มีค่าคะแนนที่แน่นอนบอกถึงการได้เปรียบเสียเปรียบภายในเกม และสิ่งนี้ก็เป็นอีกปัญหาหนึ่งของเกม Dota 2 ที่ผู้เล่นส่วนใหญ่พบคือการตัดสินใจในเกม และการประเมินเกมการแข่งขัน [3]

รูปที่ 2.3 Environment ต่างๆ ภายในเกม

2.1.3 คำศัพท์ทางเทคนิคในเกม

Creeps – ยูนิตพื้นฐานในเกมซึ่งผู้เล่นไม่สามารถควบคุมได้ โดยผู้เล่นสามารถได้เงิน (Gold) และค่าประสบการณ์ (Exp) จากการสังหาร Creeps ฝั่งตรงข้าม

Lasthit - การโจมตีในจังหวะสุดท้าย หรือคือการโจมที่ที่สามารถสังหาร Creeps ของฝ่ายตรงข้ามได้ และจะได้เงินและค่าประสบการณ์เป็นจำนวนมาก

Deny - การสังหาร Creeps ฝ่ายตัวเอง เพื่อให้ฮีโร่ฝั่งศัตรูไม่สามารถ Lasthit ได้ หากทำการ Deny สำเร็จ ฝั่งศัตรูจะได้รับค่าประสบการณ์เพียง 70% จากปกติ

Creep blocking – เทคนิคการบัง Creeps ฝ่ายตัวเองเพื่อให้เดินช้าลง เพราะจะทำให้ Creeps ของเราประทะกับ Creeps ของฝั่งตรงข้ามในตำแหน่งที่เราได้เปรียบ

MMR (Matchmaking Rating) - คะแนนที่แสดงความเก่งของผู้เล่น หากเล่นในโหมดจัดอันดับ โดยหากชนะในโหมดจัดอันดับ MMR จะเพิ่มขึ้น 25 คะแนน หากแพ้จะลดลง 25 คะแนน ซึ่งผู้เล่นส่วนมากจะมี MMR เฉลี่ยอยู่ที่ 3000 [4]

Dota 2 Workshop Tools - เป็นปลั๊กอินพัฒนา Bot และ Custom Game ของเกม Dota 2

Dota 2 Bot Script API - เป็น API สำหรับควบคุม Bot เกม Dota 2 สามารถควบคุมการทำงานของ Bot ได้ทุกอย่าง เช่น โจมตี, เคลื่อนที่ และซื้อไอเทม เป็นต้น

Vscript - เป็นระบบ Virtual Machine(VM) สำหรับให้เรารัน Script โดยจะทำงานใน Binding Layer ซึ่งอยู่ระหว่าง Source Engine (ระบบภายในเกมที่เราไม่สามารถเข้าถึงได้) และ Script ที่อยู่ภายนอกเกม

2.2 Exploration in Reinforcement learning

เนื่องจากตัดสินใจในหนึ่งครั้ง สามารถที่จะเลือกกระทำอย่างใดอย่างหนึ่งได้อย่างเดียว การใช้เรียนรู้แบบเสริมกำลัง (Reinforcement Learning) จำเป็นต้องมีการจัดการระหว่างการการสำรวจในสิ่งที่ไม่รู้ (exploration) และการทำสิ่งที่ดีที่สุดที่รู้อยู่แล้ว (exploitation) ให้เหมาะสมต่อการเรียนรู้ นอกจากนั้นแล้วถ้าหากจะทำการ Exploration ยังจำเป็นต้องเลือกด้วยว่าจะสำรวจ (explore) แบบไหนซึ่งการสำรวจที่ถูกใช้ในงานวิจัยชิ้นนี้ได้แก่

2.2.1 Epsilon Greedy

เป็นวิธีการสำรวจ (Exploration) โดยการกำหนดค่า threshold หรือค่าคงที่ epsilon(ε) ไว้เพื่อทำการสำรวจ จากนั้นทำการสุ่มค่าระหว่าง [0,1] ถ้าหากค่าที่สุ่มได้เกินค่า threshold จะทำการเลือกการกระทำ (action) โดยการสุ่ม

2.2.2 Boltzmann Distribution

เป็นการทดลองสำรวจ (Exploration) ด้วยความน่าจะเป็นที่มาจาก Boltzmann Distribution ดังแสดงในสมการที่ 2.1

$$P_t(a) = \frac{\exp\left(\frac{q_t(a)}{\tau}\right)}{\sum_{i=1}^{n} \exp\left(\frac{q_t(i)}{\tau}\right)} \quad (2.1)$$

โดย a คือ ลำดับของ action ที่ต้องการหาความน่าจะเป็น

$P_t(a)$ คือความน่าจะเป็นที่จะทำ action ที่ a

$q_t(a)$ คือค่า value function ของ action ที่ a

τ คือค่า temperature ที่เลือก

n คือ จำนวน action ที่เป็นไปได้

ซึ่งการใช้ Boltzmann Distribution ทำให้ความน่าจะเป็นที่จะทำแต่ละ action นั้นมีไม่เท่ากัน โดยมีการทำงานคล้ายกับ Softmax Function

2.2.3 Uniform Distribution

การสำรวจโดยการใช้ Uniform Distribution ในการสำรวจในปริภูมิการกระทำ (action space noise) โดยการใส่ noise ที่เป็น Uniform Distribution ดังสมการที่ 2.2

$$a_t = \pi(s_t) + \mathrm{N}(\sigma) \tag{2.2}$$

โดย a_t คือ action ณ เวลา t

$\pi(s_t)$ คือ policy function

$\mathrm{N}(\sigma)$ คือ noise ที่เป็น Uniform Distribution

2.2.4 Ornstein–Uhlenbeck process (OU)

การสำรวจโดยการใช้ OU ในปริภูมิสถานการณ์กระทำ (action space noise) โดยการใส่ noise ที่ได้จาก Ornstein–Uhlenbeck process ดังสมการที่ 2.3

$$a_t = \pi(s_t) + \mathrm{OU}(\mu, \sigma) \tag{2.3}$$

โดย a_t คือ action ณ เวลา t

$\pi(s_t)$ คือ policy function

$\mathrm{OU}(\mu, \sigma)$ คือ noise ที่เป็น Ornstein–Uhlenbeck process

ลักษณะคำคัญของการสุ่มโดยใช้ Ornstein–Uhlenbeck process นั้นจะมีลักษณะการสุ่มแบบมีความสัมพันธ์กับการสุ่มก่อนหน้า

2.3 คุณสมบัติของ task environment ในปัญญาประดิษฐ์

คุณสมบัติของ task environment ในปัญญาประดิษฐ์นั้นสามารถจำแนกได้ในเรื่องต่างๆ ดังต่อไปนี้

1) Fully observable vs partilally observable
2) Single agent vs multiagent
3) Deterministic vs stochastic
4) Episodic vs sequential
5) Static vs dynamic
6) Discrete vs continuous

2.4 Markov Decision Process (MDPs)

เป็น Mathmatical framework ที่ใช้ในการกำหนดปัญหาการตัดสินใจที่เป็นลำดับ (Sequential Decision Problems) ซึ่งแสดงความสัมพันธ์ระหว่างสถานะ (state), การกระทำ (action), รางวัล (reward), และ ทรานซิชั่น (transition) ดังแสดงในรูปที่ 2.4

S คือเซตของ state s ของ agent ใน environment

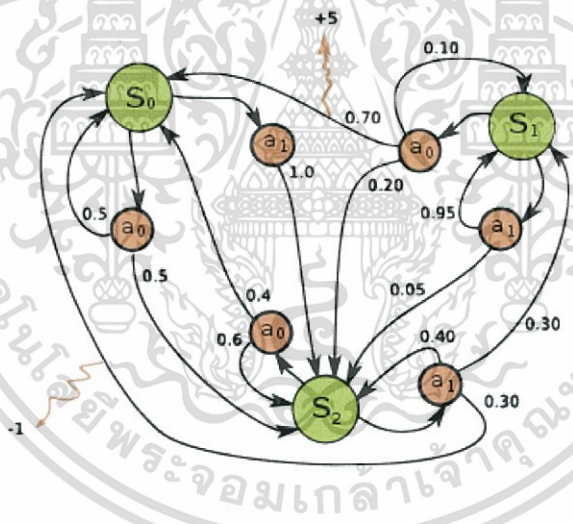
A คือเซตของ action a ของ agent

T คือ transiton ซึ่งคือ probability $\pi(S)$ ใน state s ซึ่งแสดงดังสมการที่ 2.4

$$T : S \times A \rightarrow \pi(S) \tag{2.4}$$

R คือ ค่ารางวัล (reward) ซึ่งจะมีค่าเป็นจำนวนจริง แสดงดังสมการที่ 2.5

$$r : S \times A \rightarrow \mathbb{R} \tag{2.5}$$



รูปที่ 2.4 แสดง MDPs ของ environment ที่ประกอบไปด้วย s,a,r,T

2.5 Policy และ value function

2.5.1 Policy

เป้าหมายของ Reinforcement Learning คือการหา policy $\pi : S \rightarrow A$ ที่ทำให้ค่า reward รวมมากที่สุดในระยะยาว

2.5.2 State value function

$V^{\pi} : S \rightarrow \mathbb{R}$ ฟังค์ชั่นที่ประมาณค่าผลรวมของ reward ในระยะยาวในแต่ละ state

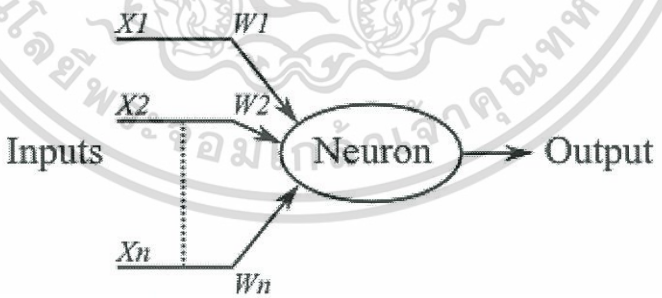
2.5.3 State action value function

$Q^{\pi} : S \times A \rightarrow \mathbb{R}$ ฟังค์ชั่นที่ประมาณค่าผลรวมของ reward ในระยะยาวในแต่ละ action ของแต่ละ state

2.6 โครงข่ายประสาทเทียม (Artificial Neural Network)

2.6.1 โครงสร้างของโครงข่ายประสาทเทียม

Artificial Neural Network หรือที่เรียกสั้นๆกันว่า Neural Network (NN) คือโมเดลทางคณิตศาสาตร์ที่เลียนแบบเซลล์ประสาทในระบบประสาท ซึ่งจะประกอบไปด้วย Neuron จำนวนมากมาเชื่อมต่อกัน โดยแต่ละ Neuron จะ ประกอบไปด้วย Input , Weight , Neuron และ Output ดังรูปที่ 2.5



รูปที่ 2.5 ภาพโครงสร้างของ Neuron

จะสังเกตได้ว่าการทำงานของ Neural Network นั้นการที่จะได้ Output ในแต่ละ Neural นั้นต้องมีการคำนวณทางคณิตศาสตร์ ผ่าน Activation Function ซึ่งอยู่ในแต่ละ Neural นั้น จะมีการคำนวนโดยใช้ $x_1, x_2x_3, \dots, x_n \in X$ เป็น Feature (Input) และค่าถ่วงน้ำหนัก (Weight) $w_1, w_2, w_3, \dots, w_n \in W$ ดังสมการที่ 2.6

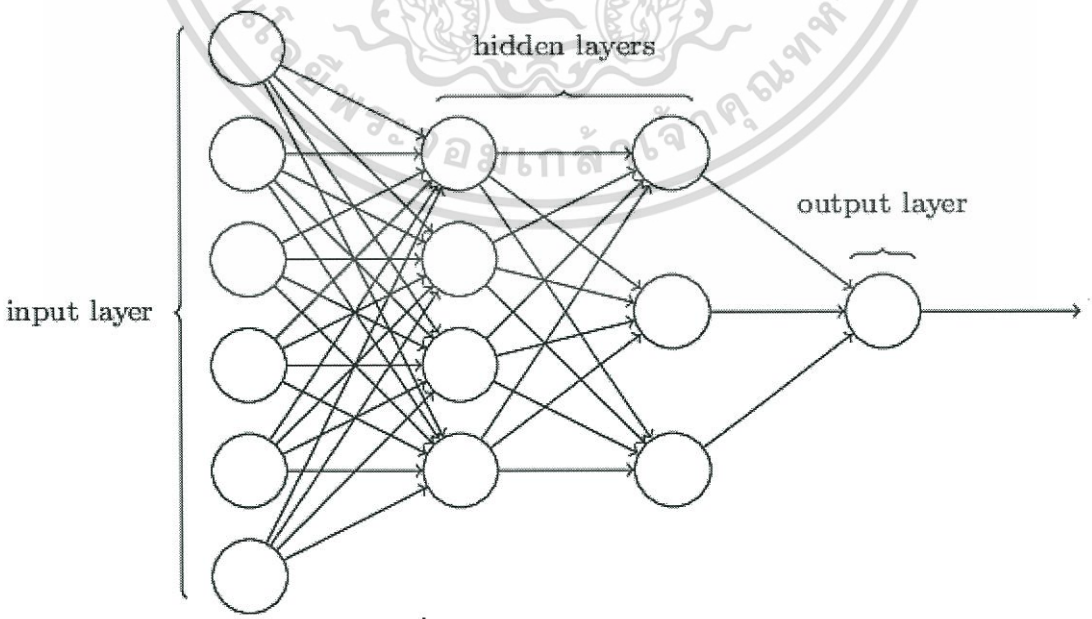
$$a = g(x_1w_1 + x_2w_2 + \cdots + x_nw_n) \tag{2.6}$$

โดย a คือ Neural (Activation Unit) ที่ i ใน layer j

g คือ Activation Function ซึ่งอาจจะเป็น Linear Funciton , Logistic Function , Relu Function , ฯลฯ

เมื่อ Neuron จำนวนมากมาเชื่อมต่อกันจะสามารถทำการคำนวณ และปรับค่า Weight ของแต่ละ Neural เพื่อที่จะจดจำคู่ Input และ Output อีกทั้งจำนวน Layer ยังส่งผลต่อความสามารถในการคำนวนที่ซับซ้อนยิ่งขึ้น โดยปกติแล้วการแบ่งโครงสร้าง Neuron Network จะถูกแบ่งออกเป็น 3 ส่วนใหญ่ๆ ได้แก่

1) **Input Layer** รับข้อมูลเข้าสู่ Neural network
2) **Output Layer** ปรับ Output ให้เป็นไปตามที่ผู้ใช้ต้องการ โดยขึ้นอยู่กับ Activation Function ที่ใช้
3) **Hidden Layer** คือ Layer ชั้นอื่นๆ ที่ ไม่ใช่ Input Layer และ Output Layer



รูปที่ 2.6 Neural Network Model

2.6.2 Activation function

คือฟังก์ชันที่ทำหน้าปรับผลลัพท์ของโหนดนั้นๆให้อยู่ในช่วงหรือลักษณะที่ต้องการ ในโครงงานเล่มนี้ได้ใช้ Activation function ต่างๆดังนี้

1) Identity function (Linear)

เป็นฟังก์ชันที่มี output เหมือน input สามารถใช้ได้หลากหลายไม่ว่าจะในส่วนของ Hidden Layer หรือ Output Layer โดยมีสมการดังนี้

$$f(x) = x \tag{2.7}$$

2) Hyperbolic tangent (tanh)

เป็นฟังก์ชันที่จะแปลงค่าให้อยู่ในช่วง [-1,1] โดยปกติแล้วจะใช้ใน Hidden Layer โดยมีสมการดังนี้

$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{2.8}$$

3) Rectified linear unit (ReLU)

เป็นฟังก์ชันที่จะแปลงค่าลบให้เป็น 0 โดยปกติแล้วจะใช้ใน Hidden Layer โดยมีสมการดังนี้

$$f(x) = x^{+} = \max(0, x) \tag{2.9}$$

2.6.3 Loss function

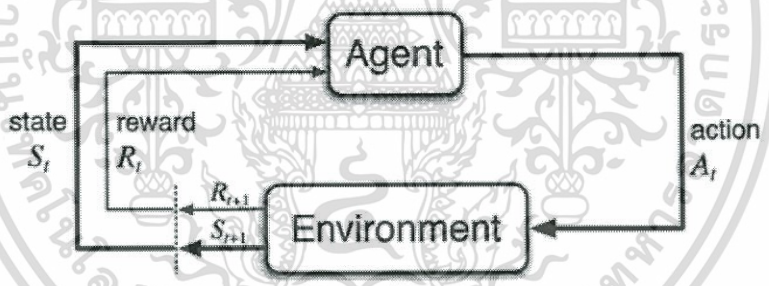
เป็นฟังก์ชันสำหรับคำนวณค่าความผิดพลาดในการทำนายผลของโมเดล ในโครงงานนี้ได้ใช้ Mean Square Error ในการคำนวณค่าความผิดพลาดของทั้งสองการทดลอง (DDQN และ DDPQ)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \tag{2.10}$$

2.7 กระบวนการที่ใช้ในการเรียนรู้ (Learning Methods)

2.7.1 Reinforcement Learning

เป็นการเรียนรู้ที่ เรียนรู้ว่าควรจะทำอะไร หรือกล่าวอีกนัยหนึ่งคือการหาความสัมพันธ์ระหว่างสถานะการณ์ต่างๆ (Situations) และการกระทำต่างๆ (Actions) เพื่อที่จะทำให้ได้ค่ารางวัล (Reward) มากที่สุด และ Agient ไม่จำเป็นต้องถูกบอกว่า การกระทำ (Actions) ไหนที่ควรทำ ซึ่งเหมือนๆกับกระบวนการเรียนรู้อื่นๆใน Machine Learning แต่ก็จะต่างกันตรงที่ Reinforcement Learning จะต้องมีการหาว่า การกระทำ (Actions) ไหนให้ค่า รางวัล (Reward) สูงที่สุด โดยการลองกระทำ Action นั้นๆ และที่น่าสนใจ และท้าทายในการเรียนรู้โดยใช้ Reinforcement Learning นั้นคือ Action ที่กระทำไปอาจจะไม่ได้ส่งผลแค่ รางวัลที่เกิดขึ้นในทันที (Immediate Reward) แต่ยังส่งผลกับรางวัลที่จะเกิดขึ้นในสถานการณ์ต่อๆไป (All Subsequent Rewards) ซึ่งลักษณะจำเพาะ 2 อย่างคือ Trial-and-error Search และ Delayed Reward เป็นสิ่งที่เป็นลักษณะจำเพาะที่สำคัญที่แยก Reinforcement Learning ออกจากวิธีการเรียนรู้โดยใช้ Machine Learining แบบอื่นๆ [5]



รูปที่ 2.7 การมีปฏิสัมพันธ์กันระหว่าง agent และ environment

จากรูปข้างต้น (รูปที่ 2.7) แสดงถึงความสัมพันธ์ระหว่าง Ageint และ Environment ใน Reinforcement Learning ผู้เรียนรู้ หรือผู้ตัดสินใจ ถูกเรียกว่า Agent และสิ่งที่อยู่รอบๆ Agent จะเรียกว่า Environment ความสัมพันธ์นี้เกิดขึ้นเรื่อยๆ เมื่อ Agent เลือก Action จากนั้น Environment จะเปลี่ยนไป State ใหม่ตาม Action นั้นๆ นอกจากนี้ยังให้ Reward ซึ่งจะมีลักษณะเป็นค่าตัวเลข (Numerial Value) ซึ่ง Agent จะพยายามในการเลือก Action ให้ได้ค่า Reward มากที่สุดเมื่อเวลาผ่านไปเรื่อยๆ (ในที่นี้ไม่ใช่การ Maximize Reward ให้มากที่สุด ณ ขณะนั้น แต่จะเป็นการ Maximize Reward ในอนาคตด้วย)

จะสังเกตเห็นว่าปัญหาที่ Reinforcement Learning พบคือปัญหาในการตัดสินใจที่เป็นลำดับ (Sequential Desion Problems) ซึ่งหมายการตัดสินใจ ณ ปัจจุบันจะส่งผลต่อการตัดสินใจ

ต่อๆ ไปและในการตัดสินใจในปัจจุบันจำเป็นต้องขึ้นอยู่กับการตัดสินในที่ผ่านมา โดยปัญหานี้ไม่เพียงพบใน Reinforcement Learning แต่ยังพบใน Optimal Control Problems และปัญหาอื่นๆที่เกิดขึ้นในชีวิตประจำวันของเรา

ซึ่งปัญหารูปแบบนี้ใน Reinforcement Learning ที่มี Environment ในลักษณะที่เป็น Stochastic Enivronments จะใช้ Mathematical framework คือ Markov decision process (MDPs) ในการแทนการตัดสินใจของปัญหาให้เป็นในรูปของสมการทางคณิตศาสตร์

2.7.2 Reinforcement Learning Methods

กระบวนการการเรียนรู้ของ Reinforcement learning สามารถจัดจำแนกได้ใหญ่ 3 วิธีได้แก่

2.7.2.1 Value-based methods

การเรียนรู้โดยการประมาณค่า value function และจะใช้ implecit optimal policy เช่น epsilon greedy การเรียนรู้รูปแบบนี้ถูกใช้ใน value iteration, Q-learning, Sarsa, ฯลฯ

2.7.2.2 Policy-based methods

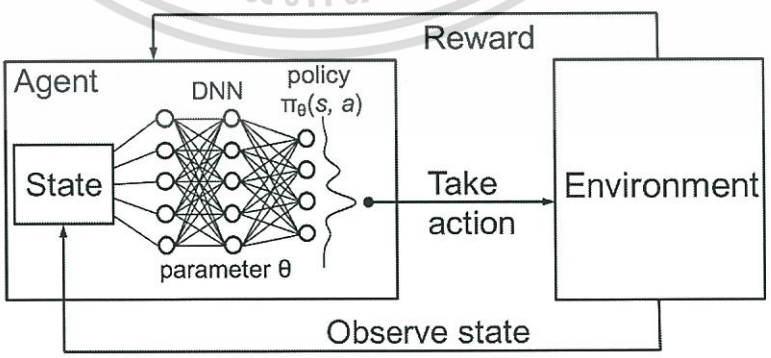
การเรียนรู้โดยการประมาณ policy และไม่มี value function การเรียนรู้รูปแบบนี้ถูกใช้ใน CMA-ES

2.7.2.3 Actor-critic methods

การเรียนรู้ที่มีการประมาณทั้ง policy และ value function การเรียนรู้รูปแบบนี้ถูกใช้ใน policy iteration, deterministic policy gradient, และ actor-critic algorithum หลายๆอย่าง

2.7.3 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) คือการนำเทคนิคของ Reinforcement Learning และ Deep Neural Network functions approximators มาใช้ร่วมกัน



รูปที่ 2.8 โครงสร้างของ Deep Reinforcement Learning

2.7.4 Mini-Batch Gradient Descent (MB-GD)

อัลกอริทึมหนึ่งใน Gradient Descent Algorithum ที่มีการอัปเดตค่า Parameters θ ของ Objective Functions $J(\theta)$ โดยแบ่งข้อมูลเป็นกลุ่มเล็กๆจาก Training examples ทั้งหมดตามจำนวน mini-batch size b และมี Learning Rate α เป็นตัวแปรค่าคงที่ ซึ่งมีรูปแบบสมการที่ 2.11

$$\theta = \theta - \alpha \sum_{i=0}^{b} \left(\nabla_{\theta} J\left(\theta ; x^{(i)}, y^{(i)}\right) \right) \tag{2.11}$$

2.7.5 Q-Learning

อัลกอริทึม Reinforcement Learning ที่ใช้ Value-based method ซึ่งประยุกต์ใช้หลักการของ Temporal Difference Learning เข้ามาช่วย โดยอัลกอริทึมนี้จะใช้ Value Function ที่เรียกว่า Q-Table โดย Q-Table จะเก็บค่า Reward รวมทั้งหมดที่จะได้จาก Episode นั้นๆ สามารถเขียนได้ในรูปของ $Q(s,a)$ โดย s คือ State และ a คือ Action Q-Learning มีอัลกอริทึมที่เอาไว้อัพเดทค่า Q-Tableดังนี้

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \, max_{a'} Q(s',a') - Q(s,a)) \tag{2.12}$$

โดย s คือ State ถัดไป

a' คือ Action เลือกทำใน State s' ซึ่งค่า Q-Table มากที่สุด

γ คือ Discount Factor มีค่า [0,1]

α คือ Learning Rate มีค่า [0,1]

ซึ่ง $r + \gamma \, max_{a'} Q(s',a')$ คือค่า Target Value และ $Q(s,a)$ คือค่าของ Predict Value เมื่อคำนวณแล้วจะได้ค่า Loss ออกมาเพื่อนำไปปรับ Q-Table ต่อไป โดย Q-Learning มีกระบวนการดังรูปที่ 2.9

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

รูปที่ 2.9 Q-Learning Pseudocode

2.7.6 Deep Q-Network (DQN)

เนื่องจากไม่สามารถนำ Q-Learning มาใช้แก้ปัญหาที่มี State Space มากได้ เพราะ Q-Table ไม่สามารถเก็บได้พอ จึงมีการดัดแปลงนำ Neural Network มาใช้แทน Q-Table ซึ่งวิธีการอัพเดทค่า Q-Table ยังมีสมการเช่นเดิม ดังรูปด้านล่างดังนี้

```
initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
      select an action a
            with probability ε select a random action
            otherwise select a = argmax_a' Q(s,a')
      carry out action a
      observe reward r and new state s'
      store experience <s, a, r, s'> in replay memory D

      sample random transitions <ss, aa, rr, ss'> from replay memory D
      calculate target for each minibatch transition
            if ss' is terminal state then tt = rr
            otherwise tt = rr + γmax_a' Q(ss', aa')
      train the Q network using (tt - Q(ss, aa))^2 as loss

      s = s'
until terminated
```

รูปที่ 2.10 Deep Q-Learning Pseudocode

แต่จะมีการเพิ่มเทคนิคที่ใช้คือ เมื่อใช้ Neural Network ทำให้สามารถใช้ Mini-Batch Gradient Descent ในการอัพเดตค่า Parameters และใช้เทคนิค Replay memory ในการนำข้อมูลจากเกมที่เล่นมาเก็บไว้ใน Memory จากนั้นจึงค่อย Random และนำมาอัพเดตค่า Paramenters ใน Neural Network

2.7.7 Double Deep Q-Network (DDQN)

เป็นเทคนิคที่เพิ่มเติมมาจาก DQN เพราะว่า DQN นั้นมีปัญหาเล็กน้อยเนื่องจากค่า weight ที่เปลี่ยนไปในระหว่างขั้นตอน Mini-Batch Training ส่งผลให้ Q-Target นั้นเปลี่ยนแปลงตลอดเวลา จึงได้มีการใช้ Neural Network ที่มีคุณสมบัติเหมือนกันกับ Q-Network ดังเดิม เพื่อทำหน้าที่คำนวณค่า Q-Target โดยเฉพาะ และ Network นี้จะอัพเดทค่า weight หลังจากที่ทำ Mini-Batch Training เสร็จแล้ว

2.7.8 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) อัลกอริทึมเป็นเป็นหนึ่งใน actor-critic algorithm ใช้สำหรับปัญหาที่มีปริภูมิการกระทำที่ต่อเนื่อง (continuous action space) ซึ่งคล้ายกับ DQN และมี critic ในการประมาณค่า Q-value function และประยุกต์ใช้ Bellman equation ในการอัพเดต ดังสมการที่ 2.13

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi_\theta(s_{t+1})) \quad (2.13)$$

โดย π_θ คือ actor หรือ policy

$Q(s_t, a_t)$ คือ state action value function หรือ Q-value function

γ คือ Learning Rate มีค่า [0,1]

โดยปกติแล้ว actor-critic algorithm ส่วนใหญ่เช่น policy gradient, compatible features, ฯลฯ มักจะใช้ stochastic policy ในการแก้ปัญหา continuous แต่สำหรับ DPG และ DDPG เป็นอัลกอริทึมที่ใช้ gradient ในการคำนวณ deterministic policies ซึ่งทำงานอย่างมีประสิทธิภาพและได้รับพิสูจน์ว่าสามารถแก้ปัญหาได้ (proof of convergence)

2.8 เทคนิคและเทคโนโลยีที่ใช้

2.8.1 ภาษา Python

เป็นภาษาเขียนโปรแกรมที่กำลังได้รับความนิยมในขณะนี้ เนื่องจากมี Library จำนวนมาก และ มี syntax ที่ง่ายในการเขียน

2.8.2 Tensorflow

เป็น Library สำหรับทำ Machine Learning ในภาษา Python

2.8.3 Keras

เป็น Library สำหรับทำ Machine Learning ในภาษา Python ซึ่งเป็น High Level API ของ Tensorflow ดังนั้นจึงสามารถใช้งานได้ง่ายกว่า

2.8.4 Flask

เป็น Web framework ในภาษา Python มีข้อดีคือสามารถพัฒนาระบบ Server ได้ง่าย

2.8.5 ภาษา Lua

เป็นภาษาโปรแกรมที่นิยมมากในการทำ API ทางด้านเกม เนื่องจากมีลักษณะเป็น Script ซึ่งจะทำงานอยู่ในฝั่งของเกม ทำให้ผู้เขียนไม่จำเป็นต้องคอมไพล์โค้ดเพื่อทดสอบโปรแกรม ซึ่งเกม Dota 2 จำเป็นต้องใช้ ภาษา Lua ในการควบคุมสิ่งต่างๆภายในเกม

2.8.6 Window 10

คือ ระบบปฏิบัติการคอมพิวเตอร์ระบบหนึ่ง (operating system) ที่สร้างขึ้นโดยบริษัทไมโครซอฟต์ ซึ่งกำลังเป็นที่นิยมอยู่ในปัจจุบัน โดย Dota2 Workshop Tools จะสามารถทำงานได้ภายในระบบปฏิบัติการ Window 10 เท่านั้น

2.9 งานวิจัยที่เกี่ยวข้อง

2.9.1 Human-level control through deep reinforcement learning

งานวิจัยนี้ [6] ได้มีการนำ Deep Reinforcement Learning มาใช้ในการแก้ปัญหาที่ Environment มีความซับซ้อน โดยการใช้ Deep Q-Network และ Reinforcement Learning มาใช้ในการแก้ปัญหาเกม Atari 2600 games ซึ่งจะรับ input เป็น pixel screen ของเกม จากงานวิจัยนั้นพบว่าการใช้ Deep Reinforcement Learning นั้นให้ผลลัพธ์ที่ดีกว่า (ได้คะแนนที่ดีกว่า) ในแทบทุกๆเกม

2.9.2 Mastering the game of Go with deep neural networks and tree search

งานวิจัยนี้ [7] ได้แสดงเทคนิคการใช้ Deep Neural Network , Monte Carlo tree search และ Reinforcemnt Learning ในการพัฒนา AlphaGo Zero ซึ่งเป็น AlphaGo เวอร์ชั่นล่าสุดที่มีการเรียนรู้ที่เริ่มจากศูนย์ มี Neural Network ที่ใช้ในการประมาณค่าความน่าจะเป็นของตาเดินแต่ละตา และมีการใช้ Monte Carlo tree search ในการจำลองตาเดินแต่ละตาเพื่อที่จะอัพเดต parameter ใน Policy Network ให้แม่นยำขึ้น และที่สำคัญได้มีการใช้เทคนิค Reinforcement Learning อีกด้วย

2.9.3 Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks

งานวิจัยนี้ [8] ได้มีการใช้ Deterministic Policy Gradient ในการแก้ปัญหา micromanagement task ในเกม StarCarft และงานวิจัยนี้ได้เสนอ heuristic reinforcement learning ซึ่งได้รวมการสำรวจแบบปกติ และเทคนิค backpropagation เข้าด้วยกัน

บทที่ 3

การออกแบบและการพัฒนา

3.1 แนวคิดในการศึกษา

ปัจจุบันการนำการเรียนรู้แบบเสริมกำลังนั้นกำลังได้รับความสนใจเป็นอย่างมาก เนื่องจากสามารถแก้ปัญหาปริภูมิสถานะและปริภูมิการกระทำที่ซับซ้อนได้ รวมทั้งได้มีงานวิจัยที่เป็นเริ่มนำการเรียนรู้แบบเสริมกำลังมาใช้แก้ปัญหาเกมมายิ่งขึ้น จึงทำทำให้ผู้จัดทำสนใจเกี่ยวกับที่จะศึกษาการเรียนรู้แบบเสริมกำลัง และต่อมาเมื่อทำการศึกษาก็พบว่า

ในการใช้การเรียนรู้แบบเสริมกำลังกับปัญหาต่างๆ หรือแม้กระทั้งกับปัญหาในการเล่นเกมนั้น ปัญญาหาที่มักพบคือการที่ไม่มีการสำรวจที่มากพอในสภาพแวดล้อมนั้นๆของเกม ซึ่งเกิดการที่โดยปกตินั้นอัลกอริทึมที่ใช้ในการเรียนรู้แบบเสริมกำลัง จะมีการสุ่มจากการสุ่มการกระทำที่ไม่มีนัยยะเนื่องจาก state-action value function หรือฟังชั่นที่ใช้ในการวัดว่าการกระทำในสถานะนั้น เป็นสิ่งที่ดีหรือไม่ดี มีค่าความคลาดเคลื่อนสูง ในตอนเริ่มต้น ด้วยการเลือกการกระทำเพื่อสำรวจในรูปแบบนี้จะทำให้ใช้เวลาในการเรียนรู้ที่นาน หรืออาจจะไม่สามารถเรียนรู้การแก้ปัญหานั้นๆได้เลย

โดยงานวิจัยนี้จะเน้นไปในการศึกษา เพื่อหาวิธีการสำรวจที่เหมาะสมกับการแก้ปัญหา Lasthit และ block Creep ในเกม Dota2 โดยการทดลองเลือกวิธีการสำรวจ และค่าพารามิเตอร์ต่างๆ ให้เหมาะสมกับการสำรวจในแต่ละปัญหา และใช้อัลกอริทึม DDQN สำหรับปัญหา Lasthit ใช้ DDPG สำหรับปัญหา Creep blocking

3.2 แนวคิดในการเลือกใช้อัลกอริทึม

การเลือกใช้อัลกอริทึมกับปัญหานั้นๆโดยหลักๆผู้ศึกษาได้อิงตามความเหมาะสม คุณสมบัติของสภาพแวดล้อมนั้นๆ ประกอบกับข้อดีข้อเสียของอัลกอริทึม โดยจะพิจารณาการเลือกใช้อัลกอริทึมดังนี้

3.2.1 DDQN กับปัญหา Lasthit Creep

DDQN เป็นอัลกอริทึมที่ได้รับความนิยมในการแก้ปัญหาที่จำนวนของ action มีจำกัด (discrete action space), สามารถเข้าใจได้ง่าย มีตัวอย่าง source code ค่อนข้างมาก เช่น Google Deepmind ที่นำ DDQN มาแก้ปัญหาในเกม Atari หลายๆเกม ผู้พัฒนาจึงคิดว่า DDQN มีความเหมาะสมมากในการแก้ปัญหา Lasthit ดังตัวอย่าง

3.2.2 DDPG กับปัญหา Creep blocking

เนื่องจากปัญหา Creep blocking เป็นปัญหาที่มีความซับซ้อน และปริภูมิการกระทำที่ต่อเนื่อง อีกทั้งการที่ DDPG นั้นได้ใช้ policy ที่มีลักษณะเป็น deterministic policy ซึ่งมีประสิทธิภาพมากกว่า Policy gradient ที่ใช้ stochastic policy ในปัญหาที่มีปริภูมิการกรทำที่ต่อเนื่อง (continuous action space)

3.3 แนวคิดในการเลือกการสำรวจรูปแบบต่างๆ

3.3.1 ขั้นตอนวิธีแบบละโมบจากค่าคงที่ epsilon

เป็นขั้นตอนวิธีในการสำรวจขั้นพื้นฐาน ที่นิยมใช้กันทั่วไป เราใช้วิธีนี้เป็นพื้นฐานในการวัดผลการสำรวจวิธีอื่นๆ

3.3.2 การกระจายตัวแบบ Boltzmann

เป็นนำความรู้ทางด้านเคมีและสถิติมาประยุกต์ใช้ทางด้าน Reinforcment Learning วิธีนี้มีแนวคิดคล้ายกับ Softmax Function เราใช้วิธีนี้เพื่อวัดผลว่า ชนิดของการสำรวจมีผลมากน้อยแค่ไหนสำหรับการแก้ปัญหาในเกม Dota2

3.3.3 การรบกวนในปริภูมิการกระทำต่อเนื่องโดยการสุ่มแบบไม่มีนัยยะโดยการกระจาย Uniform

การรบกวนปริภูมิสถานะในลักษณะนี้จะมีการใช้การสุ่มโดยไม่มีนัยยะ หรือไม่มีความสัมพันธ์กับการสุ่มก่อนหน้า เพื่อทดสอบความสัมพันธ์ของการสุ่มในลักษณะนี้กับสภาพแวดล้อมของปัญหา เราจึงได้ใช้การสุ่มในลักษณะนี้ และให้มีการสุ่มในลักษณะการกระจายแบบ Uniform ในการทดลอง

3.3.4 การรบกวนในปริภูมิการกระทำต่อเนื่องโดยการสุ่มแบบมีความสัมพันธ์กับการสุ่มก่อนหน้าโดยใช้ Ornstein-Uhlenbeck Process

เพื่อที่จะได้การสุ่มในปริภูมิสถานะที่ค่อนข้างคงที่ในการสุ่มค่าในการสุ่มไม่แกว่งไปมาจนเกินไป จึงจำเป็นต้องใช้การสุ่มแบบมีความสัมพันธ์กับการสุ่มก่อนหน้าโดยการใช้ Ornstein-Uhlenbeck Process

3.4 การออกแบบการทดลอง

3.4.1 ปัญหา Lasthit

สำหรับการทดลองนี้จำเป็นต้องมีการกำหนดตัวแปรควบคุม เพื่อกำหนดกรอบของปัญหาที่เราต้องการศึกษา โดยมีรายละเอียดดังต่อไปนี้

1. **กำหนด object ภายในเกม**

   ประกอบด้วยสิ่งต่างๆดังต่อไปนี้

   a. ฮีโร่ฝั่ง Radian 1 ตัว

   b. ครีปฝั่ง Radian 4 ตัว

   c. ครีปฝั่ง Dire 1 ตัว

   d. แผนที่เกม Dota2 พื้นฐาน

2. **เร่งเวลาในภายในเกม 10 เท่า**

3. **กำหนด terminal state**

   Episode นั้นๆ จะจบเมื่อ Creep ฝั่ง Dire ตาย

4. **กำหนดฟังก์ชั่นคะแนน**

   การให้คะแนนในแต่ละ timestep จะแบ่งออกเป็นสองแบบคือ

   a. ได้คะแนน +100 เมื่อ Hero สามารถสังหาร Creep ฝั่งตรงข้ามได้

   b. ได้คะแนน -1 หากสังหารไม่ได้

รูปที่ 3.1 สภาพแวดล้อมสำหรับการทดลอง Lasthit

3.4.2 การทดลองในปัญหา Lasthit

การทดลองที่ 1 ใช้การ bootstrapping

วัตถุประสงค์

เพื่อนำคะแนนของวิธีนี้มาใช้เป็นมาตราฐานในการวัดผลวิธีอื่นๆ

สมมุติฐาน

ค่า reward มีค่าแกว่งอย่างสม่ำเสมอ

การทดลองที่ 2 ใช้การสำรวจแบบ Epsilon greedy ที่ใช้ 24x24 hidden layer

วัตถุประสงค์

เพื่อทดสอบการลู่เข้าของ reward โดยใช้ Epsilon greedy

สมมุติฐาน

สามารถลู่เข้าสู่คำตอบได้เร็วและมากกว่าแบบ bootstrapping

การทดลองที่ 3 ใช้การสำรวจแบบ Boltzmann ที่ใช้ 24x24 hidden layer

วัตถุประสงค์

เพื่อทดสอบการลู่เข้าของคะแนนโดยใช้ Boltzmann Distribution

สมมุติฐาน

สามารถลู่เข้าสู่คำตอบได้เร็วและมากกว่าแบบ Epsilon greedy

การทดลองที่ 4 ใช้การสำรวจแบบ Epsilon greedy ที่ใช้ 120x120 hidden layer

วัตถุประสงค์

เพื่อทดสอบการลู่เข้าของ reward โดยการเพิ่มจำนวนของ hidden layer

สมมุติฐาน

สามารถลู่เข้าสู่คำตอบได้เร็วและมากกว่าการทดลองที่ 3

การทดลองที่ 5 ใช้การสำรวจแบบ Epsilon greedy ที่ใช้ 120x120 hidden layer และมีการตั้งค่าพลังโจมตีตายตัว

วัตถุประสงค์

เพื่อทดสอบการลู่เข้าของ reward โดยใช้การแก้ไขค่าพลังโจมตีของ hero และ Creep จากเดิมที่เป็นค่าสุ่ม ให้เป็นค่าตายตัว เพื่อทำให้ state ไม่แกว่ง

สมมุติฐาน

สามารถลู่เข้าสู่คำตอบได้เร็วและมากกว่าทุกการทดลองก่อนหน้านี้

3.4.3 ปัญหา Creep blocking

สำหรับการทดลองนี้จำเป็นต้องมีการกำหนดปัจจัยควบคุม เพื่อกำหนดกรอบของปัญหาที่เราต้องการศึกษา ซึ่งจะกำหนดให้

a. ทดลองในปริภูมิการกระทำที่ต่อเนื่อง

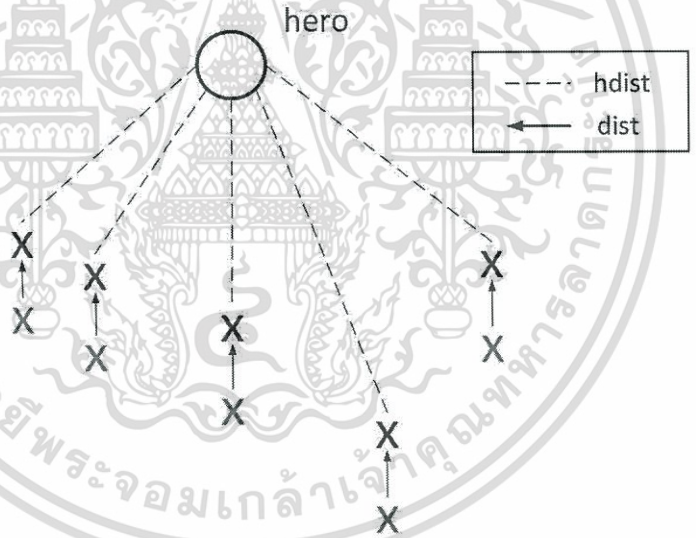
b. เร่งเวลาในภายในเกม 10 เท่า

c. กำหนด terminal state

1. Creep เดินนำหน้า hero
2. Hero เดินเลยเกินระยะที่กำหนดไว้

d. กำหนดฟังก์ชั่นคะแนน

ในการเรียนรู้แบบเสริมกำลังนั้นจำเป็นต้องมีการกำหนดฟังก์ชั่นคะแนนให้กับสภาพแวดล้อมเพื่อที่จะเป็นการกำหนดวัตถุประสงค์ให้ปัญญาประดิษฐ์ทำการหาวิธีการที่จะได้ค่าคะแนนรวมในระยะยาวที่มากที่สุด การให้คะแนนจะเกิดขึ้นเมื่อ hero สามารถขัดขวางการเดินของ Creep ได้รูปที่ 3.2 แสดงการให้คะแนนในกรณีที่ hero ขัดขวางการเดินของ Creep



รูปที่ 3.2 การขัดขวางการเดินของ Creep

ในทุกๆ 0.2 วินาที จะมีการหาจุดพิกัดของยูนิตแต่ละตัว และทำการคำนวณระยะห่างระหว่าง hero และ Creep **(hdist)** และ ระยะการเปลี่ยนแปลงจากตำแหน่งเดิมของ Creep **(dist)** และทำการให้คะแนนดังตารางที่ 3.1

ตารางที่ 3.1 แสดงการให้คะแนนเมื่อ hero ขัดขวางการเดินของ Creep

hdist	**reward**
น้อยกว่า 500	- ค่า **dist** มีค่าระหว่าง [0,20) คะแนน *+0.5* - ค่า **dist** มีค่าระหว่าง (20,40) คะแนน *+0.35* - ค่า **dist** มีค่าระหว่าง (40,60) คะแนน *+0.2* - ค่า **dist** มีค่าระหว่าง [60,∞) คะแนน *+0*
มากกว่าเท่ากับ 500	คะแนน *+0*

สำหรับการให้คะแนนในกรณีเมื่อ hero ไม่สามารถขัดขวางทางเดินของ Creep ได้นั้นจะเกิดขึ้นในลักษณะเดียวกับ c.1 และจะทำให้การเล่นตานั้นจบลง และให้คะแนนเท่ากับ *-1*

3.4.4 การทดลองในปัญหา Creep blocking

การทดลองที่ 1 ใช้การสำรวจแบบ boostraping

การทดลองนี้ทดลองโดยการใช้วิธีการ boostraping เพียงอย่างเดียว เพื่อศึกษารูปแบบพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ โดยมีสมมุติฐานคือ ค่าคะแนนที่ได้ควรจะค่อนข้างที่จะคงที่ เนื่องจากใช้การสำรวจแบบ boostraping

การทดลองที่ 2 สำรวจโดยการรบกวนในปริภูมิการกระทำโดยใช้การกระจาย Uniform

การทดลองมีวัตถุประสงค์เพื่อศึกษารูปแบบพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ โดยการรบกวนในปริภูมิการกระทำโดยใช้การสุ่มแบบการกระจาย Uniform ในช่วงที่แตกต่างๆกันได้แก่ 0 (ไม่มีการใช้ noise), ใส่ noise ในช่วง [-10,10], และใส่ noise ในช่วง [-20,20] ดังแสดงในตารางที่ 3.2

ตารางที่ 3.2 การเพิ่ม noise ต่างๆในการทดลองที่ 2

การทดลอง	**ช่วงของ noise จากการกระจายแบบ Uniform**
การทดลองที่ 2.1	0
การทดลองที่ 2.2	[-10, 10]
การทดลองที่ 2.3	[-20, 20]

โดยการทดลองนี้มีสมมุติฐานคือ การเรียนรู้สามารถเรียนรู้ในช่วงของ noise ที่เหมาะสมจะสามารถแก้ปัญหาได้

การทดลองที่ 3 สำรวจโดยการรบกวนในปริภูมิการกระทำโดยใช้การกระจาย Uniform กับการสำรวจแบบ boostraping

การทดลองมีวัตถุประสงค์เพื่อศึกษารูปแบบพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ ในกรณีที่มีการใช้ boostraping เป็น 20 เปอร์เซนต์ของการเล่น กับการรบกวนในปริภูมิการกระทำโดยใช้การสุ่มแบบการกระจาย Uniform ในช่วงที่แตกต่างๆกัน ได้แก่ 0 (ไม่มีการใช้ noise), ใส่ noise ในช่วง [-10,10], และใส่ noise ในช่วง [-20,20] ดังแสดงในตารางที่ 3.3

ตารางที่ 3.3 การเพิ่ม noise ต่างๆในการทดลองที่ 3

การทดลอง	**Boostraping (%)**	**ช่วงของ noise จากการกระจายแบบ Uniform**
การทดลองที่ 3.1	20	0
การทดลองที่ 3.2	20	[-10, 10]
การทดลองที่ 3.3	20	[-20, 20]

โดยมีสมมุติฐานคือ การเรียนรู้สามารถที่จะแก้ปัญหาได้ เนื่องจากมีการใช้ boostraping เข้ามาช่วย

การทดลองที่ 4 สำรวจโดยการรบกวนในปริภูมิการกระทำโดยใช้ Ornstein–Uhlenbeck process

การทดลองมีวัตถุประสงค์เพื่อศึกษารูปแบบพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ โดยการใช้ Ornstein-Uhlenbeck process โดยใช้ ค่าเฉลี่ย(mu) เท่ากับ 10 และใช้การกระจาย(sigma) เท่ากับ 30 โดยมีสมมุติฐานคือ การใช้ Ornstein-Uhlenbeck process นั้นจะได้ผลดีกว่าการใช้การกระจาย Uniform

การทดลองที่ 5 สำรวจโดยการรบกวนในปริภูมิการกระทำโดยใช้ Ornstein–Uhlenbeck process กับการสำรวจแบบ boostraping

การทดลองมีวัตถุประสงค์เพื่อศึกษารูปแบบพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ โดยการใช้ Ornstein-Uhlenbeck process โดยใช้ค่าพารามิเตอร์ดังแสดงในตารางที่ 3.4 และ boostraping เป็น 20 เปอร์เซนต์ของการเล่

ตารางที่ 3.4 การเพิ่ม noise ต่างๆในการทดลองที่ 5

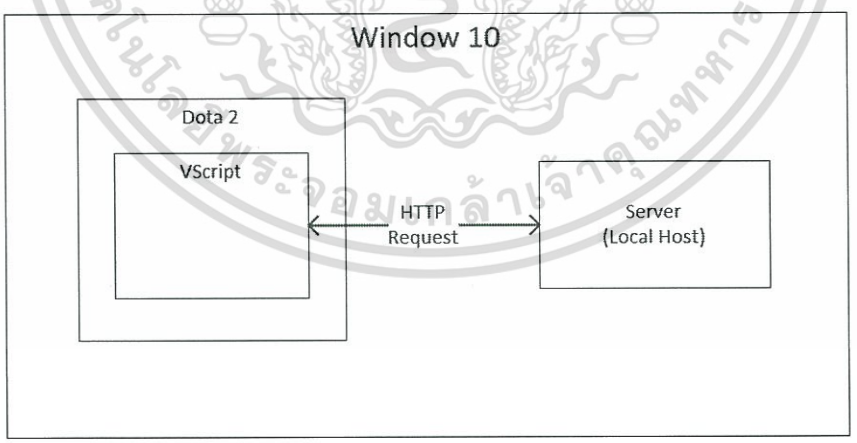
การทดลอง	Boostraping (%)	mu	sigma
การทดลองที่ 5.1	20	10	30
การทดลองที่ 5.2	20	-10	30
การทดลองที่ 5.3	20	0	30

โดยมีสมมุติฐานคือ สามารถเรียนรู้ที่จะแก้ปัญหาได้เร็วกว่าเนื่องจากมีการใช้ boostraping

3.5 ภาพรวมของระบบ

ในส่วนของภาพรวมของระบบนี้จะอธิบายถึง การออกแบบและองค์ประกอบของเครื่องมือที่ใช้ในการทดลอง ซึ่งจะประกอบไปด้วย 2 ส่วนหลัก ได้แก่ ส่วนของสิ่งแวดล้อมภายในเกม (Dota2 Environment) และ ส่วนของอัลกอริทึมที่ใช้ในการแก้ปัญหา (Server)

โดยอัลกอริทึมจะติดต่อกับสิ่งแวดล้อมของเกม เพื่อทำการเล่นเกมโดยเชื่อมต่อผ่านทาง HTTP Request และเมื่อเล่นเกมจบหนึ่งตา (episode) สิ่งแวดล้อมของเกมจะทำการส่งข้อมูลจากการเล่นเกมได้แก่ ค่าคะแนน (reward) และค่าสถานะภายในเกมต่างๆ (state) กลับไปยังเซิฟเวอร์เพื่อปรับปรุงค่าพารามิเตอร์ของอัลกอริทึม ซึ่งแสดงดังรูปที่ 3.3

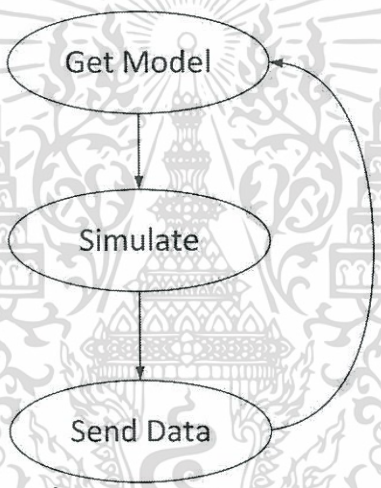


รูปที่ 3.3 แสดงการเชื่อมต่อระหว่าง ส่วนของสิ่งแวดล้อมภายในเกม และส่วนของอัลกอริทึมที่ใช้ในการแก้ปัญหา

3.5.2 ส่วนของสิ่งแวดล้อมภายในเกม (Dota2 Environment)

ในตัวเกม Dota2 สิ่งแวดล้อมภายในเกมสามารถพัฒนาได้โดยการใช้ Vscript ซึ่งทางผู้พัฒนาเกมได้เขียนขึ้นโดยใช้ภาษา Lua ทั้งนี้เราจำเป็นต้องพัฒนาส่วนของสิ่งแวดล้อมภายในเกมเพื่อควบคุมสิ่งแวดล้อมภายในเกมให้สามารถเล่นเกม และสามารถติดต่อกับส่วนของอัลกอริทึมที่ใช้ในการแก้ปัญหาได้ ซึ่งส่วนที่ต้องพัฒนาจะประกอบไปด้วย

- การควบคุม state ของเกมได้แก่ Get_Model, Send_data, และ Simulate (รูปที่ 3.4)
- การรับส่งข้อมูลกับส่วนอัลกอริทึม
- ฟังก์ชั่นการให้คะแนน
- โครงข่ายประสาทเทียมสำหรับการทำ forward pass เพื่อที่ได้ action
- ฟังก์ชั่นที่ใช้ในการสำรวจรูปแบบต่างๆ



รูปที่ 3.4 การควบคุม State ของเกม

3.5.3 ส่วนของอัลกอริทึมที่ใช้ในการแก้ปัญหา (Server)

ในส่วนของอัลกอริทึมที่ใช้ในการแก้ปัญหาจะต้องทำหน้าที่ในการรับคะแนนที่ได้จากการเล่น และ ข้อมูลสถานะต่างๆของเกม จากนั้นนำข้อมูลที่ได้มาใช้ในการอัพเดตค่าพารามิเตอร์ต่างๆในโครงข่ายประสาทเทียม ซึ่งในส่วนของอัลกอริทึมที่ใช้ในการแก้ปัญหา ที่ต้องพัฒนาจะประกอบไปด้วย

- การรับส่งข้อมูลกับส่วนของสิ่งแวดล้อมภายในเกม
- Replay memory
- โครงข่ายประสาทเทียมสำหรับอัลกอริทึม
- การกู้คืนอัลกอริทึม
- กราฟสรุปบน Tensorboard

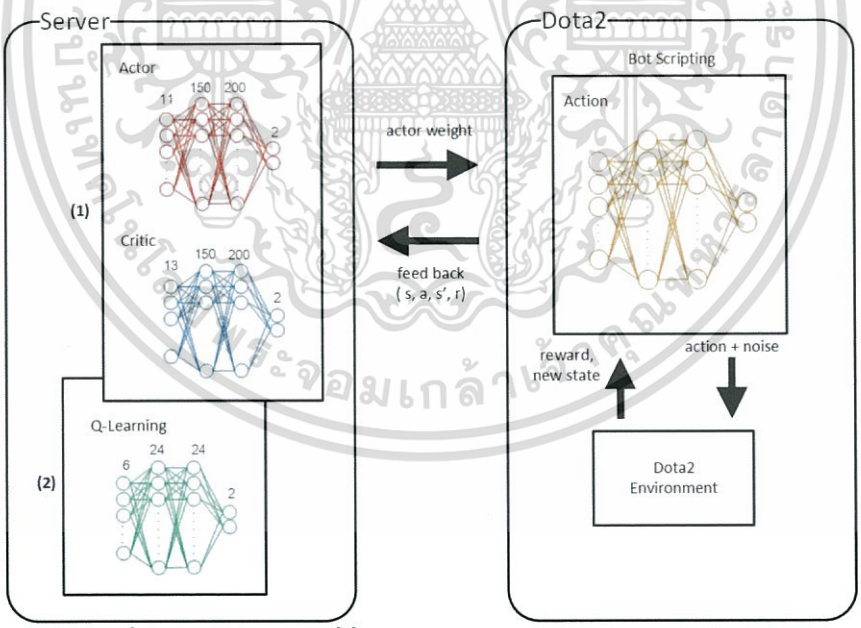
การสร้างเครื่องมือในส่วนนี้นั้นจะเขียนด้วยภาษา Python และใช้ Libaray ต่างๆดัง

ตารางที่ 3.5 Python Library ที่ใช้

Libaray ในภาษา Python	**รายละเอียด**
Flask==0.12.2	ใช้ในการสร้าง web service เพื่อติดต่อกับ Evironment ของเกม
tensorflow==1.4.0	ใช้ในการสร้างโครงข่ายประสาทเทียมสำหรับอัลกอริทึม และแสดงผลกราฟใน Tensorboard
Keras==2.1.2	ใช้ในการสร้างโครงข่ายประสาทเทียมสำหรับอัลกอริทึม
h5py==2.7.1	เพื่อบันทึกค่าพารามิเตอร์ต่างๆของโครงข่ายประสาทเทียม
requests==2.18.4	ใช้ในการสร้าง http request

3.5.4 โครงสร้างของระบบ

รูปที่ 3.3 แสดงข้อมูลที่สื่อสารกันระหว่าง server และ เกม Dota2 โดย server จะทำหน้าที่อัพเดท Neural network model และส่งกลับไปให้เกม Dota2 เพื่อให้นำไปฝึกฝนต่อไป



รูปที่ 3.5 แสดงข้อมูลที่สื่อสารกันระหว่าง server และ เกม Dota2

3.6 รายละเอียดการพัฒนา

3.6.1 ข้อมูลเข้าและออกของ Neural network

3.6.1.1 Lasthit

ในการทดลอง Lasthit ได้มีการออกแบบ input(state) และ output(action) ของ Neural Network ดังตารางที่ 3.6 และ 3.7

ตารางที่ 3.6 State ของ DDQN

ชื่อ	ช่วงข้อมูล	รายละเอียด
hp_Creep	[-1, 1]	ปริมาณเลือดของ Creep ฝ่ายตรงข้าม
delay_hero		ค่า delay ที่เหลือของฮีโร่ในการโจมตีครั้งถัดไป
delay_01		ค่า delay ที่เหลือของ Creep ตัวที่ 1 ในการโจมตีครั้งถัดไป
delay_02		ค่า delay ที่เหลือของ Creep ตัวที่ 2 ในการโจมตีครั้งถัดไป
delay_03		ค่า delay ที่เหลือของ Creep ตัวที่ 3 ในการโจมตีครั้งถัดไป
delay_04		ค่า delay ที่เหลือของ Creep ตัวที่ 4 ในการโจมตีครั้งถัดไป

ตารางที่ 3.7 Action ของ DDQN

ชื่อ	รายละเอียด
Idle	หยุดนิ่ง
Attack	โจมตีครีปฝั่งตรงข้าม

3.6.1.2 Creep Blocking

ในการทดลอง Creep Blocking ได้มีการออกแบบ input และ output ของ Neural Network ดังตารางที่ 3.8 และ 3.9

ตารางที่ 3.8 State ของ DDPG

ชื่อ	ช่วงข้อมูล	รายละเอียด
x_hero	[-1, 1]	พิกัดจริงของฮีโร่ในแกน x
y_hero		พิกัดจริงของฮีโร่ในแกน y
speed_hero		ความเร็วของฮีโร่
x_Creep01		พิกัดสัมพันธ์ของ Creep ตัวที่ 1 กับ ฮีโร่ในแกน x
y_Creep01		พิกัดสัมพันธ์ของ Creep ตัวที่ 1 กับ ฮีโร่ในแกน y
x_Creep02		พิกัดสัมพันธ์ของ Creep ตัวที่ 2 กับ ฮีโร่ในแกน x
y_Creep02		พิกัดสัมพันธ์ของ Creep ตัวที่ 2 กับ ฮีโร่ในแกน y
x_Creep03		พิกัดสัมพันธ์ของ Creep ตัวที่ 3 กับ ฮีโร่ในแกน x
y_Creep03		พิกัดสัมพันธ์ของ Creep ตัวที่ 3 กับ ฮีโร่ในแกน y
x_Creep04		พิกัดสัมพันธ์ของ Creep ตัวที่ 4 กับ ฮีโร่ในแกน x
y_Creep04		พิกัดสัมพันธ์ของ Creep ตัวที่ 4 กับ ฮีโร่ในแกน y

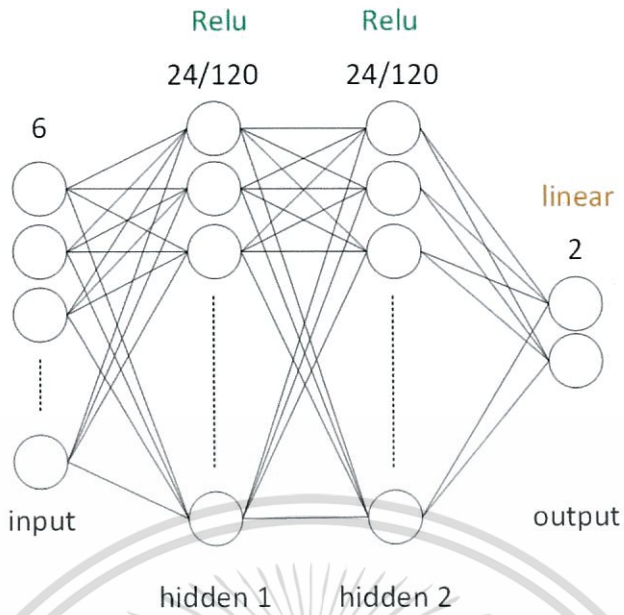
ตารางที่ 3.9 Action ของ DDPG

ชื่อ	รายละเอียด
x	ค่าระยะทางในแกน x ที่ฮีโร่ต้องเดิน
y	ค่าระยะทางในแกน y ที่ฮีโร่ต้องเดิน

3.6.2 โครงสร้างของ Neural Network

3.6.2.1 Lasthit

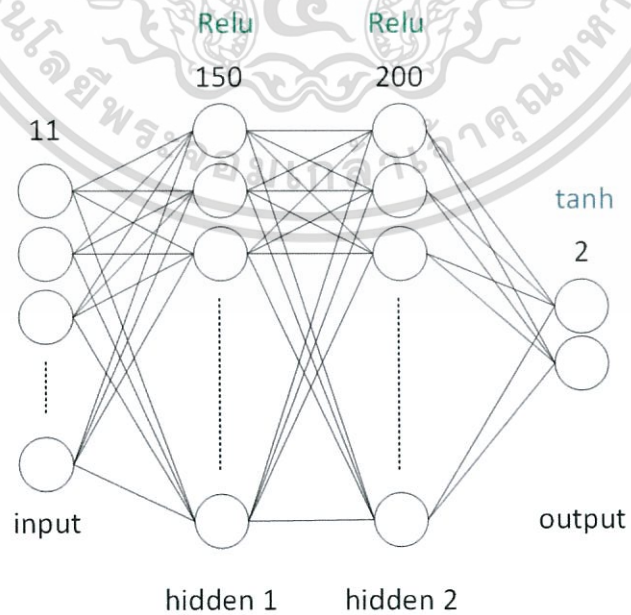
สำหรับปัญหา Lasthit นั้นได้ใช้ Neural Network ที่มี hidden layer 2 ชั้น และมีจำนวนโหนด 24x24 (สำหรับการทดลองที่ 2 และ 3) และ 120x120 (สำหรับการทดลองที่ 4 และ 5) ดังรูปด้านล่าง โดยจะมี Neural Network อยู่ 2 model ที่มี โครงสร้างเหมือนกันตามอัลกอริทึม DDQN



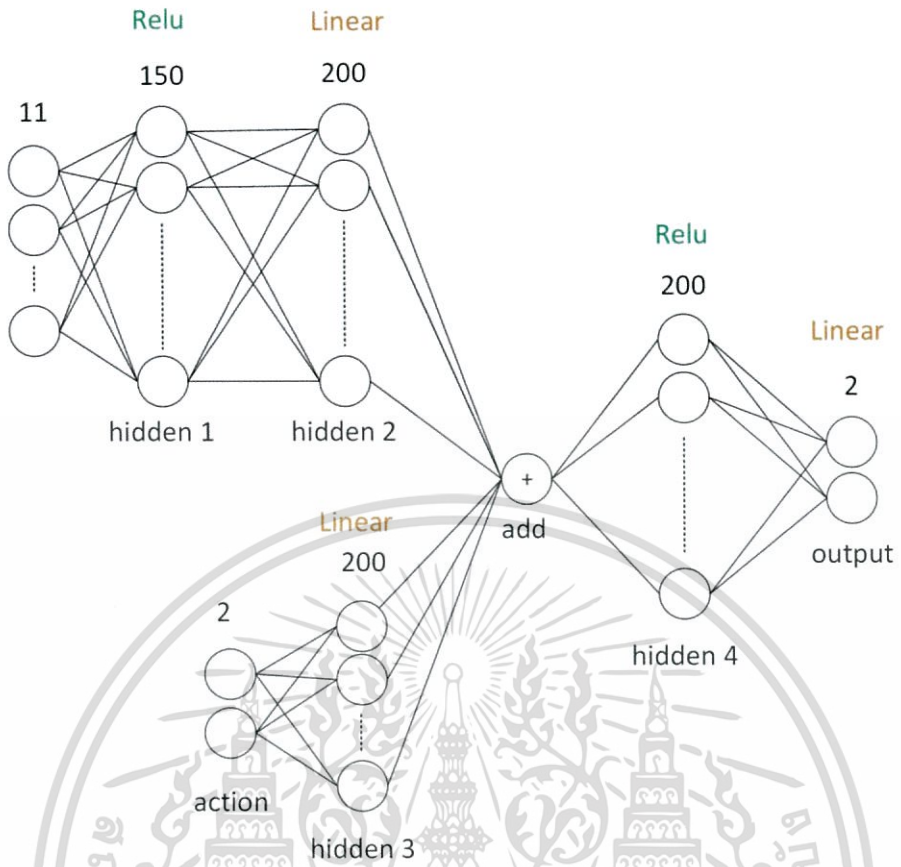
รูปที่ 3.6 DQN ของการทดลอง Lasthit

3.6.2.2 Creep Blocking

สำหรับปัญหา Creep Blocking นั้นได้ใช้ Neural Network จำนวนสองแบบคือ Actor Network ซึ่งมี hidden layer 2 ชั้นและมีจำนวนโหนด 150 และ 200 ตามลำดับดังรูปด้านล่าง ดังรูปที่ 3.7 และ Critic Network ซึ่งมีโครงสร้างที่ไม่เหมือน Neural network ปกติโดยจะมี hidden layer 4 ส่วน โดยจะนำค่าน้ำหนักจาก hidden ที่ 2 และ 3 มาบวกกันและสร้างเป็น hidden ที่ 4 ดังรูปที่ 3.8 ซึ่งเป็นไปตามอัลกอริทึม DDPG



รูปที่ 3.7 Actor Network สำหรับการทดลอง Creep Blocking

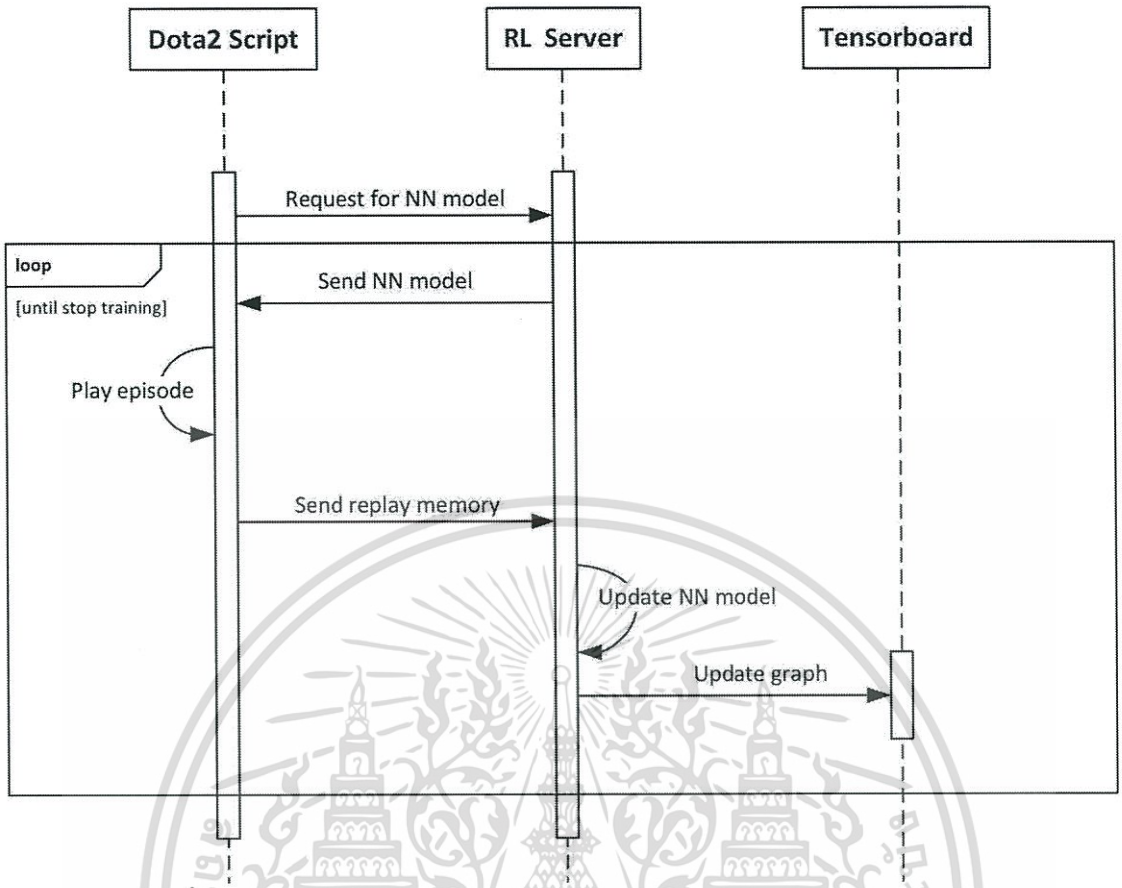


รูปที่ 3.8 Critic Network สำหรับการทดลอง Creep Blocking

3.6.3 ลำดับการทำงาน

การทำงานของระบบในขั้นตอนการฝึกของทั้งสองการทดลอง(Lasthit และ Creep blocking)นั้นมีการทำงานดังรูปที่ 3.9 โดยมีการทำงานดังนี้

1) script ภายในเกม Dota จะทำการร้องขอ Neural network model จาก server
2) server จะส่งข้อมูลของ weight และ bias ต่างๆผ่านทาง HTTP Response ในรูปแบบ json
3) script ในเกม Dota2 จะทำการเก็บอัพเดทค่า weight และ bias และทำการเก็บ replay memory จนกว่าจะจบ episode
4) ส่ง replay memory ให้ server
5) ทำการ update neural network weight และ bias
6) Server ส่งข้อมูลค่าคะแนนที่ได้ไปแสดงใน Tensorboard
7) ทำซ้ำตั้งแต่ข้อ 2 ไปเรื่อยๆ จนกระทั่ง เราหยุดการฝึก



รูปที่ 3.9 แสดง sequence diagram ของการฝึก Neural Network(NN) model

บทที่ 4

การทดลองและผลการทดลอง

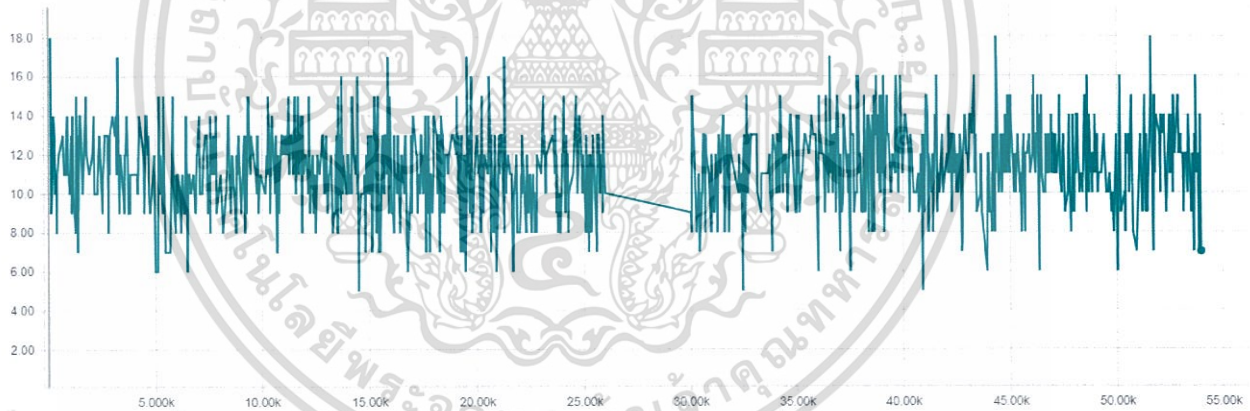
4.1 การทดลองแก้ปัญหา Lasthit

การทดลองทั้ง 5 แบบ เราได้พล็อตกราฟของจำนวน Creep ที่สังหารได้ในแต่ละ episode และเพื่อให้กราฟที่แสดงออกมาไม่แกว่งมากจนเกินไป เราพล็อตจุดในทุกๆ 20 timestep โดยจำนวน Creep ที่สังหารได้จะใช้ผลรวม(sum)ใน 20 รอบนั้น(มีค่า 0 -20)

4.1.1 การทดลองที่ 1 ใช้การ bootstrapping

ผลการทดลอง

เราได้ทำการใช้ boostraping เล่นจำนวน 55,000 ตา จากรูปที่ 4.1 แสดงจำนวน Creep ที่ฮีโร่สามารถสังหารได้ ซึ่งมีค่าค่อนข้างแกว่ง ส่วนมากอยู่ในช่วง 6-16 ตัว ค่าเฉลี่ยอยู่ที่ประมาณ 11 ตัว (55 %) และมีแนวโน้มว่าจะคงที่เป็นเช่นนี้ไปเรื่อยๆ โดยระหว่างรอบที่ 25,000 ถึง 30,000 เกิดการค้างของ environment จึงเกิดกราฟลักษณะดังกล่าว



รูปที่ 4.1 กราฟแสดงจำนวน Creep ที่สามารถสังหารได้ เมื่อใช้การ boostraping

4.1.2 การทดลองที่ 2 ใช้การสำรวจแบบ Epsilon greedy ที่มี 24x24 hidden node

ผลการทดลอง

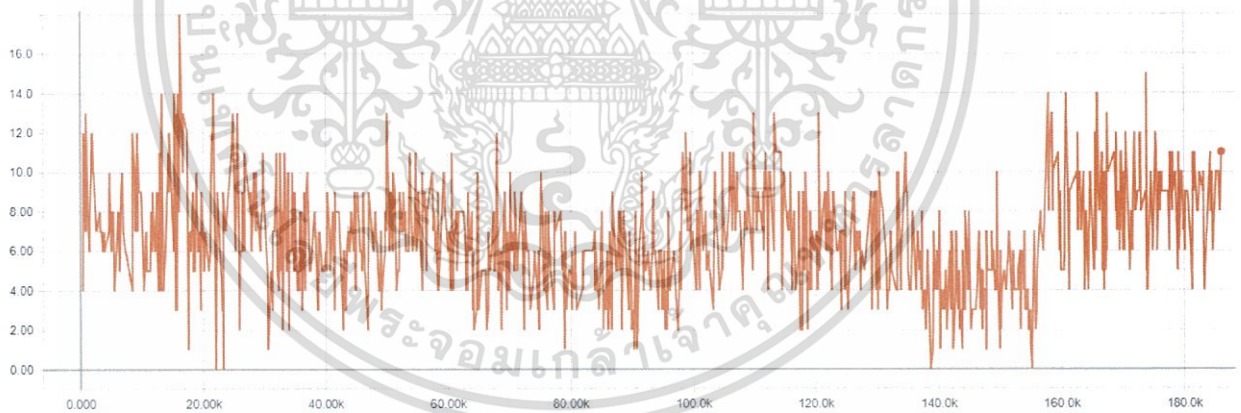
หลังจากเล่นไปประมาณ 113,000 ตา จากรูปที่ 4.2 แสดงจำนวน Creep ที่ฮีโร่สามารถสังหารได้ในแต่ 20 ตา ซึ่งมีค่าสูงสุดคือ 18 และต่ำสุดคือ 0 ตัว ค่าเฉลี่ยหลังจากเล่นไปแล้ว 40,000 ตาอยู่ที่ประมาณ 12 ตัว (60 %) และยังไม่มีแนวโน้มว่าจะเพิ่มขึ้น และส่วนที่กราฟตกเกิดจากการค้างของ environment

รูปที่ 4.2 กราฟแสดงจำนวน Creep ที่สามารถสังหารได้ เมื่อใช้การสำรวจแบบ Epsilon greedy ที่มี 24x24 hidden node

4.1.3 การทดลองที่ 3 ใช้การสำรวจแบบ Boltzmann ที่มี 24x24 hidden node

ผลการทดลอง

หลังจากเล่นไปประมาณ 180,000 ตา จากรูปที่ 4.3 แสดงจำนวน Creep ที่ฮีโร่สามารถสังหารได้ในแต่ 20 ตา ซึ่งส่วนมากจะมีค่าสูงสุดคือ 14 และต่ำสุดคือ 0 ตัว กราฟมีลักษณะคล้ายเดิมตลอด มีค่าเฉลี่ยประมาณ 7 ตัว (35 %) และยังไม่มีแนวโน้มว่าจะเพิ่มขึ้น



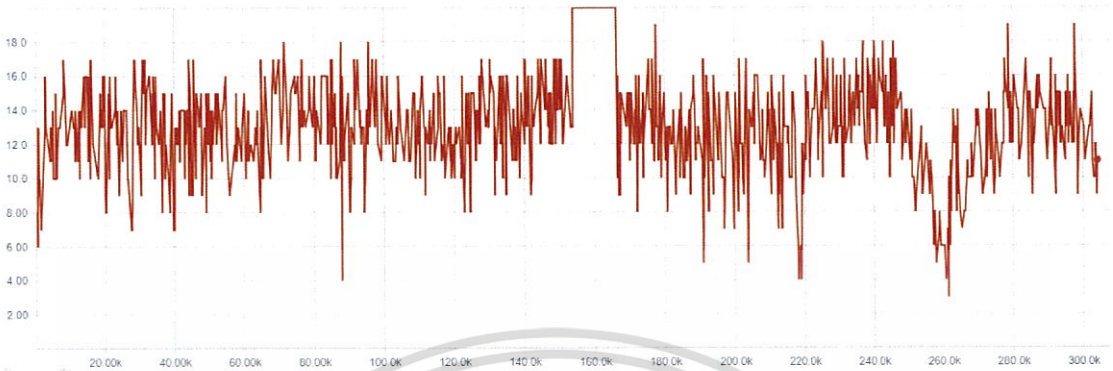
รูปที่ 4.3 กราฟแสดงจำนวน Creep ที่สามารถสังหารได้ เมื่อใช้การสำรวจแบบ Boltzmann ที่มี 24x24 hidden node

4.1.4 ทดลองที่ 4 ใช้การสำรวจแบบ epsilon greedy ที่มี 120x120 hidden node

ผลการทดลอง

หลังจากเล่นไปประมาณ 300,000 ตา จากรูปที่ 4.4 แสดงจำนวน Creep ที่ฮีโร่สามารถสังหารได้ในแต่ 20 ตา ซึ่งส่วนมากจะมีค่าสูงสุดคือ 16 และต่ำสุดคือ 8 ตัว กราฟมีลักษณะ

คล้ายเดิมตลอด มีค่าเฉลี่ยประมาณ 14 ตัว (70 %) และยังไม่มีแนวโน้มว่าจะเพิ่มขึ้น และส่วนที่กราฟขึ้นไปถึง 20 นั้น เกิดจากการค้างของ environment

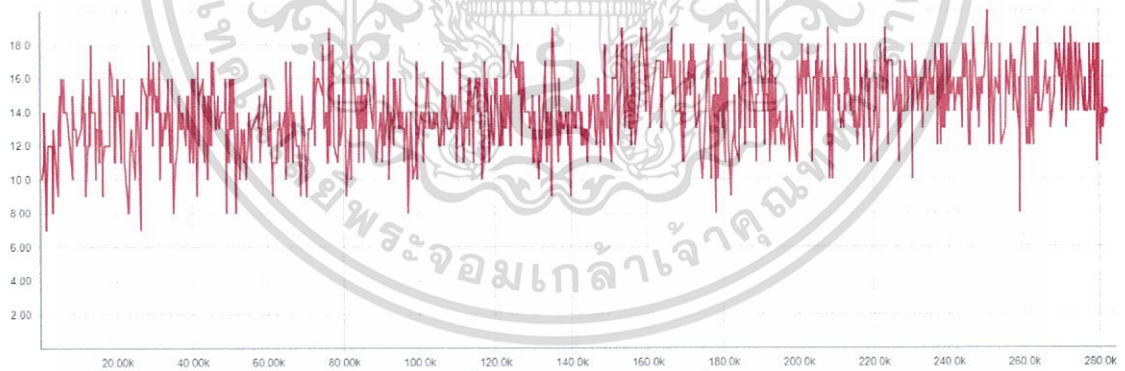


รูปที่ 4.4 กราฟแสดงจำนวน Creep ที่สามารถสังหารได้ เมื่อใช้การสำรวจแบบ Boltzmann

4.1.5 ทดลองที่ 5 ใช้การสำรวจแบบ epsilon greedy ที่มี 120x120 hidden node และมีการตั้งค่าพลังโจมตีตายตัว

ผลการทดลอง

หลังจากเล่นไปประมาณ 280,000 ตา จากรูปที่ 4.5 แสดงจำนวน Creep ที่ฮีโร่สามารถสังหารได้ในแต่ 20 ตา หลังจาก episode ที่ 200,000 ส่วนมากจะมีค่าสูงสุดคือ 18 และต่ำสุดคือ 12 ตัว มีค่าเฉลี่ยประมาณ 16 ตัว (80 %)



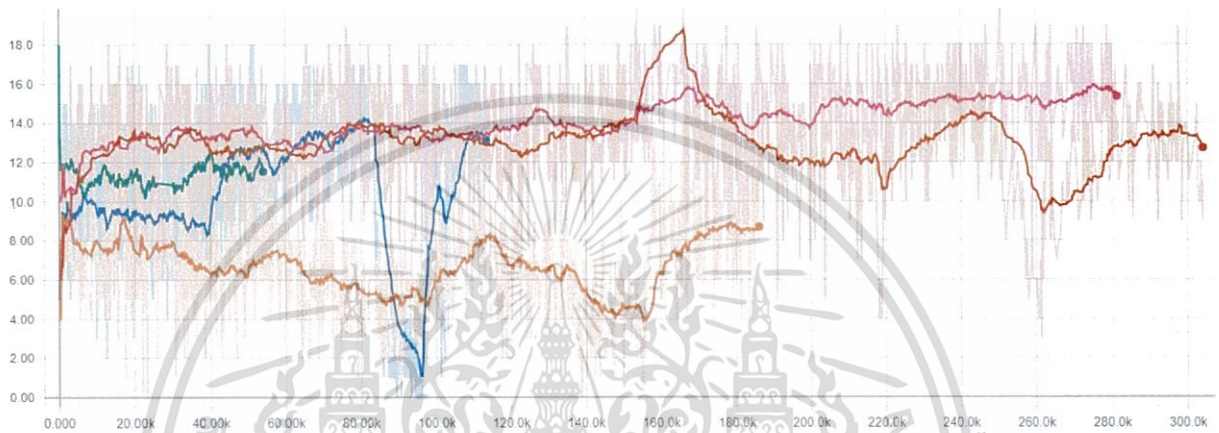
รูปที่ 4.5 กราฟแสดงจำนวน Creep ที่สามารถสังหารได้ เมื่อใช้การสำรวจแบบ epsilon greedy ที่มี 120x120 hidden node และมีการตั้งค่าพลังโจมตีตายตัว

4.1.6 สรุปผลการทดลอง

รูปที่ 4.6 เป็นการเปรียบเทียบระหว่าง 5 การทดลอง จะพบว่าการใช้ epsilon greedy สามารถให้ผลลัพท์ที่ดีกว่าการใช้ Boltzmann Distribution และจำนวน node ของ Neural Network ไม่มีผลมากนักการความแม่นยำในการสังหาร Creep กล่าวคือ การทดลองที่ 2 และ 4 มีความแม่นยำ

ที่ใกล้เคียงกันมาก แต่ต่างกับการทดลองที่ 5 ที่การปรับพลังโจมตีของ Creep ให้มีค่าตายตัวสามารถทำให้สังหาร Creep ได้แม่นยำขึ้นเรื่อยๆ และค่าแกว่งน้อยลง

โดยกราฟในรูปที่ 4.6 นั้นสีเขียวคือ boostraping, สีน้ำเงินคือ 24x24 epsilon greedy, สีส้มคือ 24x24 Boltzmann, สีแดงคือ epsilon greedy 120x120 และ สีชมพูคือ epsilon greedy 120x120 ที่มีการตั้งค่าพลังโจมตีตายตัว โดยกราฟที่แสดงนั้นใช้ Tensorboard ในการพล็อต และปรับค่า smoothing เป็น 0.968 เพื่อให้กราฟที่แสดงนั้นง่ายต่อการมองเห็นมากขึ้น



รูปที่ 4.6 กราฟเปรียบเทียบจำนวน Creep ที่สามารถสังหารได้ของการทดลองทั้งหมด

4.2 การทดลองแก้ปัญหา Creep blocking

4.2.1 การทดลองที่ 1 ใช้การสำรวจแบบ boostraping

วัตถุประสงค์

เพื่อศึกษาพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ

สมมุติฐาน

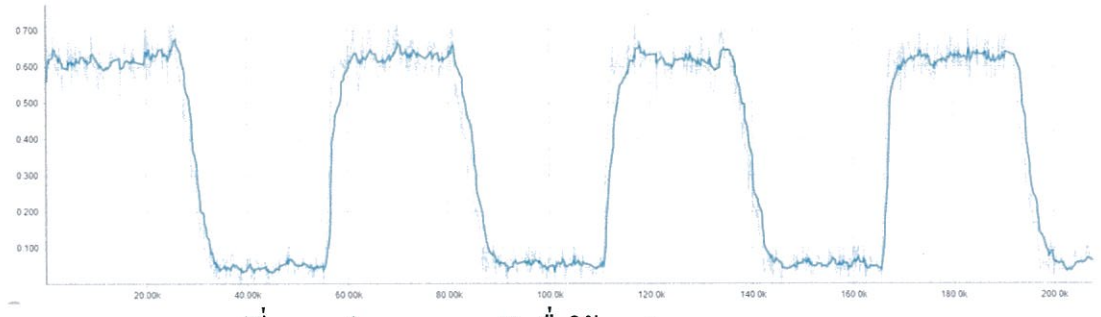
ค่าคะแนนที่ได้ควรที่จะคงที่ เนื่องจากใช้การสำรวจแบบ boostraping

ขั้นตอนการทดลอง

ทดลองรันโดยใช้การสำรวจแบบ boostraping เพียงอย่างเดียวโดยไม่ใช้อัลกอริทึมในการเรียนรู้แบบเสริมกำลังใดเลย

ผลการทดลอง

จากรูปที่ 4.7 แสดงค่าเฉลี่ยของคะแนนที่ได้จาก replay memory หลักจากการเล่นจำนวน 200,000 ตา เมื่อทำการสำรวจโดยการใช้ boostraping ซึ่งมีลักษณะของคะแนนขึ้นลงเป็นคาบละประมาณ 60,000 ตา โดยคะแนนสูงสุดจะประมาณ 0.6 และคะแนนต่ำสุดจะประมาณ 0.05 ซึ่งต่างกันมากถึง 0.55



รูปที่ 4.7 รูปแบบของรางวัลเมื่อใช้การสำรวจแบบ boostraping

4.2.2 การทดลองที่ 2 สำรวจโดยการรบกวนในปริภูมิการกระทำโดยใช้การกระจาย Uniform

วัตถุประสงค์

เพื่อศึกษารูปแบบพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ โดยการรบกวนในปริภูมิการกระทำโดยใช้การสุ่มแบบการกระจาย Uniform ในช่วงที่แตกต่างๆกัน

สมมุติฐาน

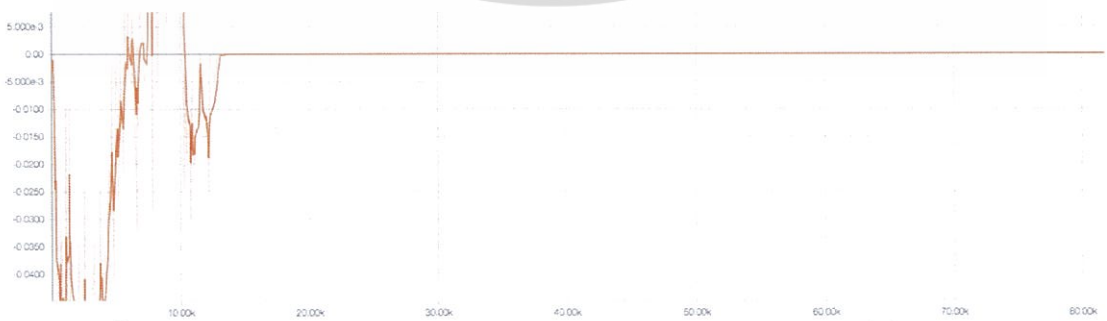
การเรียนรู้สามารถเรียนรู้ในช่วงของ noise ที่เหมาะสมจะสามารถแก้ปัญหาได้

ขั้นตอนการทดลอง

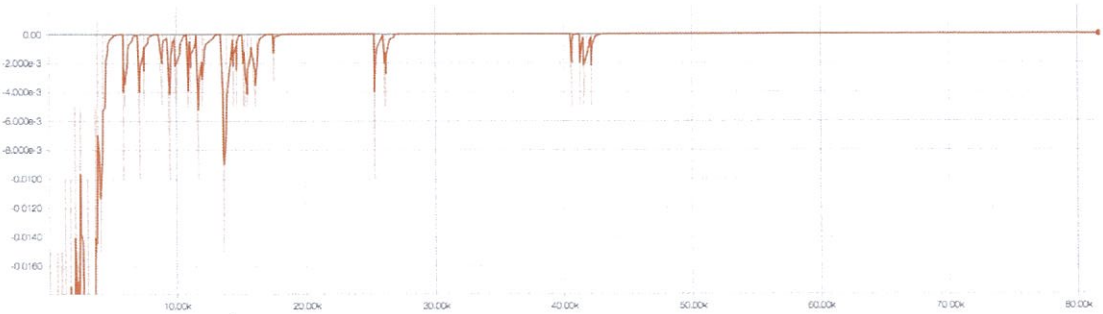
1) ทดลองรันโดยการไม่ใช้ noise
2) ทดลองรันด้วยการใช้ noise ระหว่าง -10 ถึง 10
3) ทดลองรันด้วยการใช้ noise ระหว่าง -20 ถึง 20

ผลการทดลอง

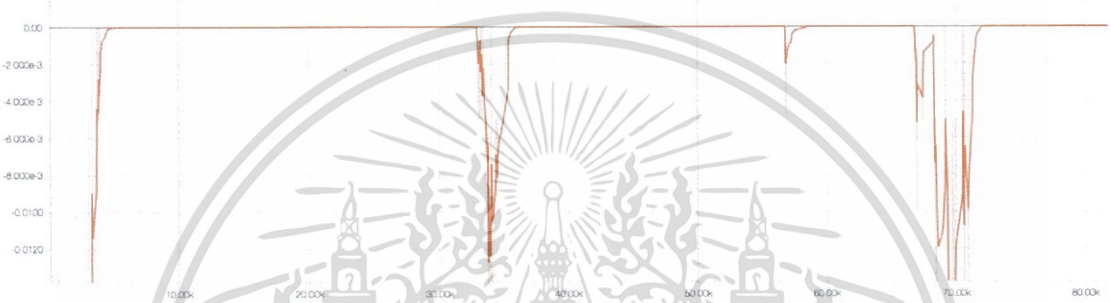
เมื่อทำการทดลองทั้ง 3 แบบจำนวน 80,000 รอบแล้วพบว่าคะแนนลู่เข้าใกล้ศูนย์ และไม่มีแนวโน้มจะสูงขึ้นทั้ง 3 การทดลองย่อย ดังรูปที่ 4.8



รูปที่ 4.8 การรบกวนในปริภูมิการกระทำโดยใช้การกระจาย Uniform เมื่อไม่มีการใช้ noise



รูปที่ 4.9 การรบกวนในปริภูมิการกระทำโดยใช้การกระจาย Uniform เมื่อช่วงของ noise ระหว่าง [-10, 10]



รูปที่ 4.10 การรบกวนในปริภูมิการกระทำโดยใช้การกระจาย Uniform เมื่อช่วงของ noise ระหว่าง [-20, 20]

4.2.3 การทดลองที่ 3 สำรวจโดยการรบกวนในปริภูมิการกระทำโดยใช้การกระจาย Uniform กับการสำรวจแบบ boostraping

วัตถุประสงค์

เพื่อศึกษารูปแบบพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ ในกรณีที่มีการใช้ boostraping โดยการรบกวนในปริภูมิการกระทำโดยใช้การสุ่มแบบการกระจาย Uniform ในช่วงที่แตกต่างๆกัน

สมมุติฐาน

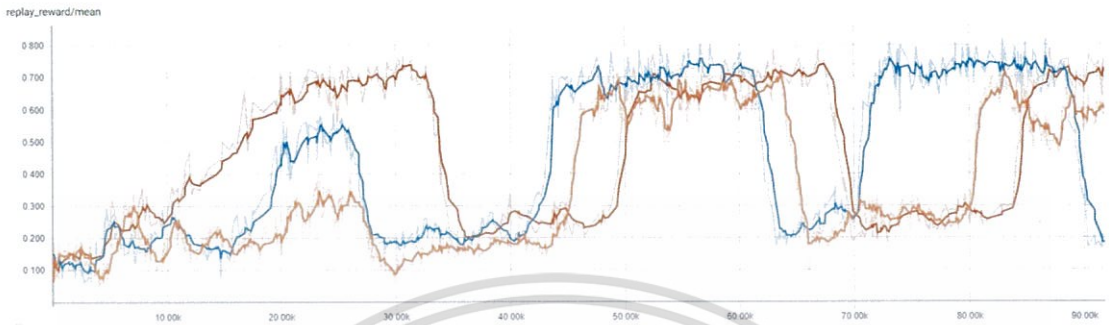
การเรียนรู้สามารถที่จะแก้ปัญหาได้ เนื่องจากมีการใช้ boostraping เข้ามาช่วย

ขั้นตอนการทดลอง

1) ทดลองรันโดยการไม่ใช้ noise และใช้ boostraping 20 เปอร์เซนต์
2) ทดลองรันด้วยการใช้ noise ระหว่าง -10 ถึง 10 และใช้ boostraping 20 เปอร์เซนต์
3) ทดลองรันด้วยการใช้ noise ระหว่าง -20 ถึง 20 และใช้ boostraping 20 เปอร์เซนต์

ผลการทดลอง

เมื่อทำการทดลองทั้ง 3 แบบจำนวน 90,000 รอบแล้วพบว่าการเพิ่ม noise จะทำให้คะแนนสูงขึ้นเร็วกว่า ดังแสดงใน รูปที่ 4.11



รูปที่ 4.11 การรบกวนในปริภูมิการกระทำโดย ไม่ใช้ noise (สีส้ม), noise ระหว่าง [-10,10] (สีน้ำเงิน), noise ระหว่าง [-20,20] (สีแดง)

4.2.4 การทดลองที่ 4 สำรวจโดยการรบกวนในปริภูมิการกระทำโดยใช้ Ornstein–Uhlenbeck process

วัตถุประสงค์

เพื่อศึกษารูปแบบพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ โดยการใช้ Ornstein-Uhlenbeck process

สมมุติฐาน

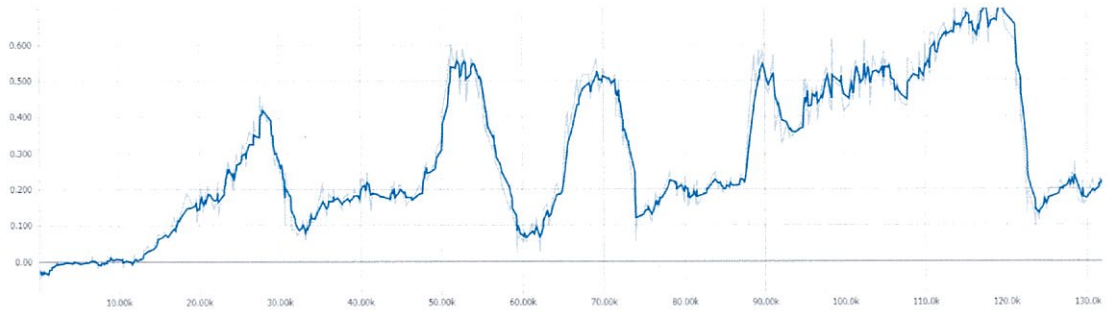
การใช้ Ornstein-Uhlenbeck process นั้นจะได้ผลดีกว่าการใช้การกระจาย Uniform

ขั้นตอนการทดลอง

ทดลองโดยการรันการสำรวจโดยการใช้ Ornstein-Uhlenbeck process และใช้ค่าเฉลี่ย(mu) เท่ากับ 10 และใช้การกระจาย(sigma) เท่ากับ 30

ผลการทดลอง

เมื่อทำการทดลองจำนวน 130,000 รอบแล้วพบว่าคะแนนเฉลี่ยขึ้นไปสูงถึง 0.7 ดังแสดงในรูปที่ 4.12



รูปที่ 4.12 การรบกวนในปริภูมิการกระทำโดยใช้ OU

4.2.5 การทดลองที่ 5 สำรวจโดยการรบกวนในปริภูมิการกระทำโดยใช้ Ornstein–Uhlenbeck process กับการสำรวจแบบ boostraping

วัตถุประสงค์

เพื่อศึกษารูปแบบพฤติกรรมการสำรวจ และลักษณะของคะแนนที่ได้จากการสำรวจ โดยการใช้ Ornstein-Uhlenbeck process และใช้ boostraping

สมมุติฐาน

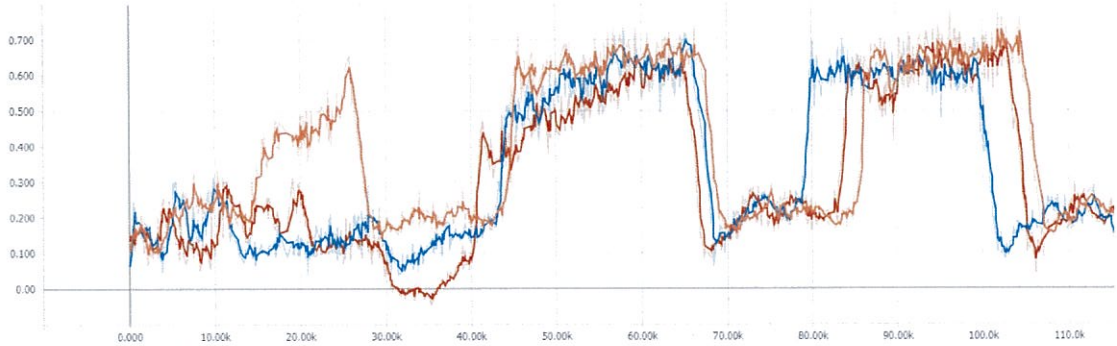
การเรียนรู้สามารถเรียนรู้ที่จะแก้ปัญหาได้เร็วกว่าการใช้ Ornstein-Uhlenbeck ที่ไม่มี boostraping

ขั้นตอนการทดลอง

1.) ทดลองรัน โดยการใช้วิธี Boostraping 20 เปอร์เซ็นต์ และใช้ Ornstein-Uhlenbeck โดยให้ mu เท่ากับ 10 sigma เท่ากับ 30
2.) ทดลองรัน โดยการใช้วิธี Boostraping 20 เปอร์เซ็นต์ และใช้ Ornstein-Uhlenbeck โดยให้ mu เท่ากับ -10 sigma เท่ากับ 30
3.) ทดลองรัน โดยการใช้วิธี Boostraping 20 เปอร์เซ็นต์ และใช้ Ornstein-Uhlenbeck โดยให้ mu เท่ากับ 0 sigma เท่ากับ 30

ผลการทดลอง

เมื่อทำการทดลองทั้ง 3 ขั้นตอนจำนวน 110,000 รอบพบว่า การใช้ mu เท่ากับ 10 ทำให้คะแนนพุ่งสูงขึ้นเร็วกว่า พารามิเตอร์อื่นๆ ดังแสดงในรูปที่ 4.13



รูปที่ 4.13 การรบกวนในปริภูมิการกระทำโดยใช้ OU โดย mu เท่ากับ 10 (สีส้ม), โดย mu เท่ากับ -10 (สีน้ำเงิน), โดย mu เท่ากับ 0 (สีแดง)

4.2.6 สรุปผลการทดลอง

จากการทดลองทั้ง 5 การทดลองที่ผ่านมาพบว่าค่าคะแนนเฉลี่ยสูงสุดของ boostraping จะอยู่ประมาณ 0.6 แต่ถ้าหากใช้การบวนการเรียนรู้แบบเสริมกำลังสามารถเพิ่มคะแนนเฉลี่ยได้ถึง 0.7 คะแนน ทำให้สามารถสรุปได้ว่าการเรียนรู้แบบเสริมกำลังในที่ใช้ค่าพารามิตเตอร์ของการสำรวจที่เหมาะสมสามารถแก้ปัญหา Creep blocking ได้ ยกเว้นการใช้การกระจาย Uniform ที่ไม่ใช้ boostraping ในการทดลองที่ 2 สาเหตุที่ไม่สามารถแก้ปัญหาได้อาจจะเนื่องจากการสำรวจโดยใช้การกระจายแบบ Uniform ทำให้พฤติกรรมการเดินของ hero จะสุ่มเดินไปรอบๆแบบไร้ทิ้งทิศทาง ซึ่งต่างจาก Ornstein–Uhlenbeck process ที่พฤติกรรมการเดินจะมีความต่อเนื่องมากกว่าในทิศทางนั้นๆในแต่ละรอบ เนื่องจากมีการคำนวณค่าการสุ่มโดยการอิงจากการสุ่มก่อนหน้าด้วย ทำให้ถึงแม้จะไม่มีการใช้ boostraping ก็สามารถแก้ปัญหาดังกล่าวได้

บทที่ 5
บทสรุปและข้อเสนอแนะ

5.1 บทสรุป

ปัญหา Lasthit และปัญหา Creep blocking ซึ่งเป็นเป็นหาของเกม Dota2 ทั้งสองสามารถแก้ได้ด้วยการใช้การเรียนรู้แบบเสริมกำลัง แม้ว่าการทดลองบางอย่างผลอาจจะไม่ดีมากนัก และการใช้การสำรวจรวมทั้งค่าพารามิเตอร์ของการสำรวจที่เหมาะสม นอกจากนั้นยังพบว่า การสร้างปัญหาประดิษฐ์สำหรับเกม Dota2 นั้นค่อนข้างจะมีปัญหาในเรื่องของ environment ในเกมอีกทั้ง ยังเป็นที่ไม่แน่ชัดในเรื่องของการไม่คงที่ของค่ารางวัลที่มีช่วงที่แกว่งไปมา ซึ่งอาจจะเกิดจากการที่ environment ทำงานผิดพลาดเมื่อรันเป็นระยะเวลานาน

ซึ่งทางผู้ศึกษาหวังว่าความรู้ที่ได้ทำการศึกษาอาจมีประโยชน์ต่อผู้ที่ต้องการทำการศึกษาเกี่ยวกับการเรียนรู้แบบเสริมกำลังในเกม Dota2 และหรือแม้กระทั้งเกมอื่นๆ

5.2 ขอบเขตและข้อจำกัด

1) ข้อจำกัดของการเพิ่ม noise เข้าไปใน continuous action space นั้นไม่สามารถใช้ได้กับเครื่องจักร หรือหุ่นยนต์ในอุตสาหกรรม เนื่องจากเครื่องมือไม่สามารถใส่ noise เข้าไปได้
2) อัลกอริทึมสำหรับการสำรวจนั้นสามารถทำงานได้ดีต่างกันในแต่ละปัญหา ทำให้ยากที่จะบอกว่าวิธีไหนคือวิธีที่ดีที่สุด
3) จำเป็นต้องเปิดเกม Dota2 ตลอดเวลาในขณะฝึกฝนโมเดล

5.3 ปัญหาและอุปสรรค

5.3.1 ปัญหาด้าน environment ของเกม Dota2

1) มีการตัวเกมมีการอัพเดตอยู่เสมอซึ่งทำให้ไม่สามารถรันได้ ซึ่งต้องคอยอัพเดตให้เป็นเวอร์ชั่นปัจจุบัน
2) เกม Dota2 ยังไม่มีโหมดที่สามารถทำงานได้โดยไม่มี UI จึงทำให้การฝึกค่อนข้างช้าและเกิดปัญหาคอขวดขึ้น
3) การเขียน Environment เพื่อให้สามารถทำงานได้ตามต้องการนั้น ค่อนข้างมีความยุ่งยาก

5.3.2 ปัญหา ด้านเทคโนยีที่ใช้

1) พื้นฐานด้าน Machine Lerning ของคณะผู้จัดทำมีค่อนข้างน้อยจึงทำให้ศึกษาเกี่ยวกับ Reinforcement Learning ได้ค่อนข้างช้า
2) Reinforcement Learning เป็นเรื่องที่ค่อนข้างใหม่ และไม่เป็นที่นิยมเท่ากับ Supervise Learning จึงทำให้หาอาจารย์หรือผู้เชี่ยวชาญที่สามารถให้คำปรึกษาได้ยาก

5.3.3 ด้านอื่นๆ

1) คอมพิวเตอร์ 1 เครื่องสามารถรันเกม Dota2 ได้เพียงอันเดียวเท่านั้น ทำให้ไม่สามารถทดลองหลายๆงานภายในคอมพิวเตอร์เครื่องเดียวได้

5.4 ข้อเสนอแนะ

1) การทดลองควรวัดผลให้จำนวนรอบมากกว่านี้ เพื่อให้ผลมีความน่าเชื่อถือมากขึ้น
2) หากเป็นไปได้ไม่ควรนำเกม Dota2 มาแก้ปัญหาด้วยวิธี Reinforcement Learning เนื่องจากมีปัญหาหลายๆอย่างทำให้ไม่สามารถรันแบบใช้ GPU เพื่อเร่งความเร็วได้ เช่น ตัวเกมไม่สามารถเร่งเวลาภายในเกมมากๆ ได้, การเชื่อมต่อระหว่างภาษา Lua(Environment) กับ Python(Agent) จำเป็นต้องทำผ่าน HTTP Request จึงทำให้ความเร็วช้าลง และตัวเกมเมื่อรันโปรแกรมเป็นเวลานานมักจะค้างหรือมีปัญหาแปลกๆตามมา เป็นต้น ดังนั้นหากต้องการนำ Reinforcement Learning มาแก้ปัญหาทางด้านเกม ควรจะเป็นเกมที่ Environment กับ Agent สามารถส่งข้อมูลหากันได้โดยตรง และสามารถประมวลผลโดยใช้ GPU ได้

5.5 แนวทางการพัฒนา

สำหรับโครงงานนี้เป็นเพียงการศึกษาและทดลองเกี่ยวกับการใช้อัลกอริทึมทางด้าน Reinforcement Learning และการสำรวจต่างๆ เพื่อแก้ปัญหาย่อยๆภายในเกม Dota2 สำหรับผู้ที่ต้องการต่อยอดโครงงานนี้ สามารถทดลองแก้ปัญหาอื่นๆภายในเกมได้ เช่น การต่อสู้ในโหมด 1v1 หรือการออกไอเท็ม เป็นต้น นอกจากนี้ยังสามารถนำแนวคิดและวิธีการไปปรับใช้กับปัญหาอื่นๆนอกเหนือจากเกม Dota2 ได้

เอกสารอ้างอิง

[1] McDonald, Tim. 2013. A Beginner's Guide to Dota 2: Part One – The Basics. [Online]. Available: https://www.pcinvasion.com/a-beginners-guide-to-dota-2-part-one-the-basics

[2] Dota 2 WIKI contributors. 2017. **Dota 2 WIKI Game modes.** Online] . Available: https://dota2.gamepedia.com/Game_modes.

[3] Circis1. 2017. **Decision making: The though process of a 6k player.** [Online]. Available: https://www.reddit.com/r/DotA2/comments/729m6v/decision_making_the_though_process_of_a_6k_player/

[4] OpenAI. 2017. **Dota 2 Solo MMR Distribution.** [Online]. Available: https://www.opendota.com/distributions

[5] Richard S. Sutton and Andrew G. 2012. **Reinforcement Learning: An Introduction Second edition, in progress.** Cambridge, Massachusetts London, England: The MIT Press

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. **Human-level control through deep reinforcement learning.** Nature, 518(7540):529–533, 2015.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. **Mastering the game of go with deep neural networks and tree search.** Nature, 529(7587):484–489, 2016.

[8] N. Usunier, G. Synnaeve, Z. Lin, S. Chintalausunier, et al. **Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks.** arXiv:1609.02993v3, 2016

[9] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, et al. **Parameter Space Noise for Exploration.** arXiv:1706.01905v2, 2018

[10] Edouard Leurent. **Why do we use the Ornstein Uhlenbeck Process in the exploration of DDPG?** [Online] Available: https://www.quora.com/Why-do-we-use-the-Ornstein-Uhlenbeck-Process-in-the-exploration-of-DDPG

[11[Uhlenbeck, G. E.; Ornstein, L. S. (1930). **On the theory of Brownian Motion**. Phys. Rev. 36: 823–841. doi:10.1103/PhysRev.36.823.

[12] Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. **Deterministic policy gradient algorithms.** In ICML, 2014.

[13] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. **Continuous control with deep reinforcement learning.** CoRR, abs/1509.02971, 2015.