Mid-term Progress Report

Miao Yinglong 1155046924

Change of Topic:

During the first few weeks, after spending one week trying to build the original paper code on Mac, I find it difficult to make progress on top of it, because of the complexity of the code. Instead, I decided to change to a new topic, with name: Reinforcement Learning in a Fighting Game, which I will build from scratch.

Project Goal

Use Reinforcement Learning techniques to build an AI to play a fighting game called Brawhalla from screenshot without control of the game system, which means the game runs simultaneously in real-time independent of the AI system.

The initial goal is to beat the inner AI of the game. If this goal is achieved, then the next direction would be to beat human players.

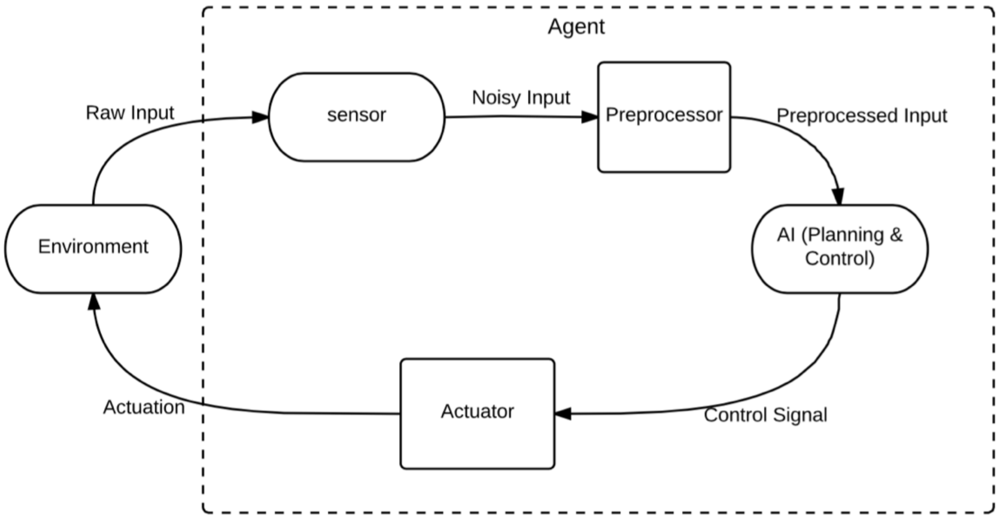
Significance of the Project

The project is significant for a couple of reasons:

1. Previous works in this field only apply Reinforcement Learning to games with access to the game engine. This work, however, wants to achieve the application without any control of it, which makes it more general and complicated. If the objective is achieved, it also makes building AI for games easier, as we no longer need to build the API layer of the game engine in between the AI agent and the game.
2. Previous games mainly focused on simple games like Atari. This work is applied to a real commercial video game called Brawhalla, which is more complex than previous games. Thus to achieve a good performance is one step further for Reinforcement Learning field.
3. The project also wants to utilize knowledge of game theory especially opponent modelling, game design and cloud computing. Thus it is a great intersection of those fields.
4.

Problem Statement & Formulation

The problem is to build an agent as the following diagram, which comes from the idea of modeling in Robotics.



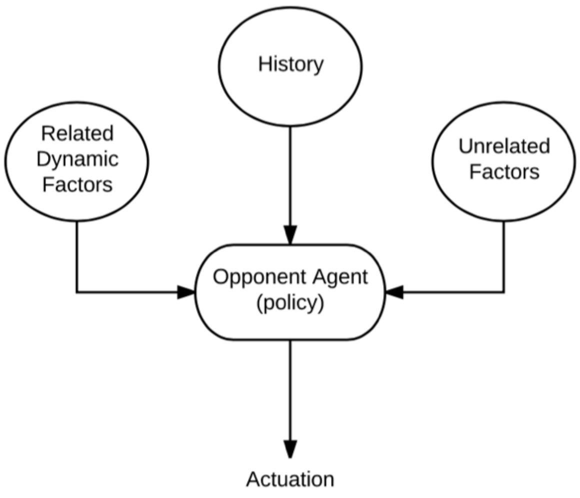
Here is the explanation of the components involved.

Environment	Including game engine, opponent, Internet condition, etc.
Raw Input	The perfect and complete information of the environment at the current time.
Sensor	An encoder for the raw input. It is limited and thus noisy, mainly due to time delay and partial observability.
Noisy Input	The output of the sensor. It is noisy. In this situation, it is the screenshot of the game screen.
Preprocessor (Perception)	Get useful information from the raw input.
Preprocessed Input	Contain useful information of the current input.
AI (Planning & Control)	Responsible to make use of the information to generate a sensible control signal.
Control Signal	In this situation, it is just the same as the actuation output of the Actuator.
Actuator	Given control signal, generate the corresponding actuation input to the environment. It is also limited and noisy.

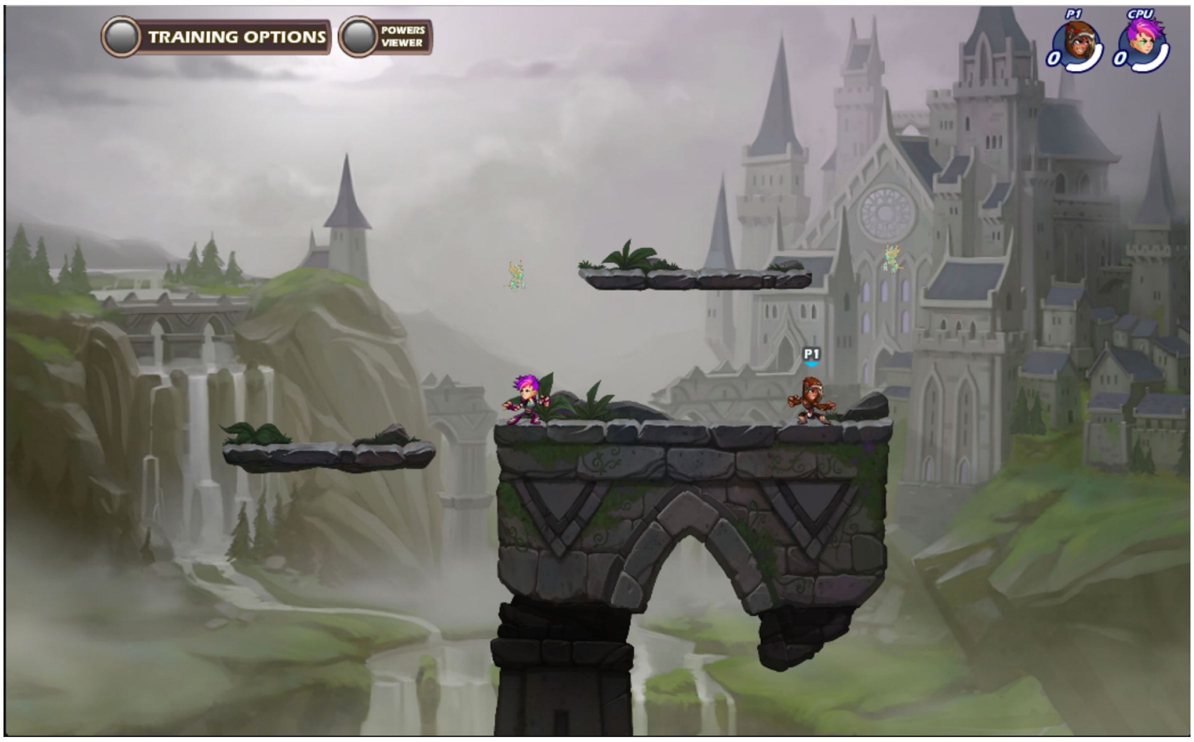
Then for each part, we will include a general description of the formulation.

Environment:

Generally, it can be modeled as a Dynamic Markov Model, with the causal assumption of the real world, and the assumption that the casual graph for the opponent policy is the following:



We assume that Unrelated Factors mainly add some stationary randomness to opponent decisions, such as unpredictable human emotion changes.
Also, we assume that the Related Dynamic Factors are subject to the policy function of the opponent. Here we propose two possible ways to model this: (1). Assume it changes very slowly (2). Assume policy is composed of cumulative part and dynamic part (similar to function and its gradient), and assume the factors are also composed of two corresponding parts.
For simplicity, we also assume that the opponent only takes into account the history up to the most recent K time series. Another option for capturing the history is to use statistics data, such as taking weighted average or running average of the data.
As for the game, a typical scene is the following:



There are a couple of important features:

1. A player loses one life when he is hit faraway outside of the map.
2. The life bar on the right affects how far a player can be hit away.
3. A player can pick up weapons on the ground to perform different attack modes. There are at most two weapons for each character.

4. A player is also able to throw away the weapon to cause damage to the opponent. To throw away at a direction, the player needs to hold the throw key and use direction keys to achieve it.
5. Combo attacks are possible, and require fast connections between attacks. Some attacks can be connected into combos while others can't.
6. Charged attacks are also possible, and the attack damage is determined by how long the key has been held.
7. There are different characters in the game. Their basic attacks are the same, but the weapon attacks are different.

Raw Input:

We don't explicitly model the raw input, but just conceptually model it to have a better understanding of the system.

Sensor:

In this project, we consider a sensor that outputs the screenshot of the game. There are some natural questions concerning this part, specifically the following ones:

1. What is the frequency of the screenshot? Is it fixed or nonstationary?
2. What is the size that best balances the speed and the performance of the system?

The noise in the sensor is not negligible, and it may have a huge impact on the performance of the system. It mainly comes from the following sources:

1. Network delay, or lagging (for playing with human players)
2. Missed frames as a result of slow sensing

Preprocessor:

This is also an important part of the system. A good preprocessor will balance the information loss with the speed and performance of the system. As there is little knowledge about the game system, an important feature of the game is the reward, which must be captured by the preprocessor. For other parts, there are two choices to try:

1. Pixel input
2. Feature engineering (which will involve domain knowledge about computer vision and the game)

As feature engineering can filter out unnecessary information of the game, which may dramatically increase the performance of our system, but it is also not easy to implement, I'd like to leave this for an improvement of the system.

AI:

This is the main part. This project will mainly use Reinforcement Learning techniques to implement the AI, but some related knowledge such as opponent modeling in game theory and control theory will also come in handy.

For competing with built-in AI, opponent modeling is not so important as the built-in AI in commercial game is usually implemented by Finite State Machine, thus it can be assumed to be nonstationary. However, when competing with human players, we may need that knowledge to have a better performance.

The AI also has to take into account the noise of the system.

Actuator:

We achieve this by just using the output of the AI part. The noise is also not negligible, and it mainly comes from the following:

1. Time delay. The time that the actuation takes place is not the same as the planning time. Some time or frames would pass, and states may be changed due to this time delay, before the actuation executes.

2. It is possible that the input is not “accepted” by the game engine sometimes, which means that the actuation actually does not take place.

Proposed solutions

1. The frameworks of Sensors and actuators are easier to implement, and they definitely are a prerequisite of the AI component. The parameters of them can be further studied and tuned to tackle the noise problem.
2. Due to the recent development of Reinforcement Learning, some recent techniques may help for our design of the AI, thus an exploration of the recent papers are required.
3. There are mainly two methods for Reinforcement Learning: model-based and model-free. As model-free method has seen a lot of recent advances and easier to apply, it is reasonable to begin with this technique. However, model-based method may also help the performance of AI, thus a future work may look at the combination of both of them. It is also interested to look at how to generally learn a model to fix the problem of generality of model-based method.
4. If the input is raw pixel, as the state space is quite large, function approximation techniques should be used, such as Deep Learning. Some experiments of new algorithms in this direction, such as DQN, Actor-Critic method can be firstly tried.
5. For simplicity, we will firstly assume that the noise contributes to a stationary distribution in the dynamics of the model, namely, the transition f(s, a) is a stationary distribution. The analysis of the noise is subject to future study.
6. The proposed programming language to be used is Python. Also, the framework to be used is Pytorch.

Proposed Achievement for First Term:

Implement the initial version using DQN, taking pixel as input, for combating with the built-in AI using a fixed character.

Proposed Achievement for Second Term:

Able to combat with a human player with a fixed character.

Proposed Directions for Second Term:

Transfer learning from AI trained with built-in AI, opponent modeling, feature engineering to accelerate learning

Proposed Achievement for Future Work:

1. Build a **general** agent that learns **fast** and performs **well** with **different characters**, that can combat with **human player**.
2. Generalize this framework to other games.
3. Conduct an AI for general commercial games using this framework.

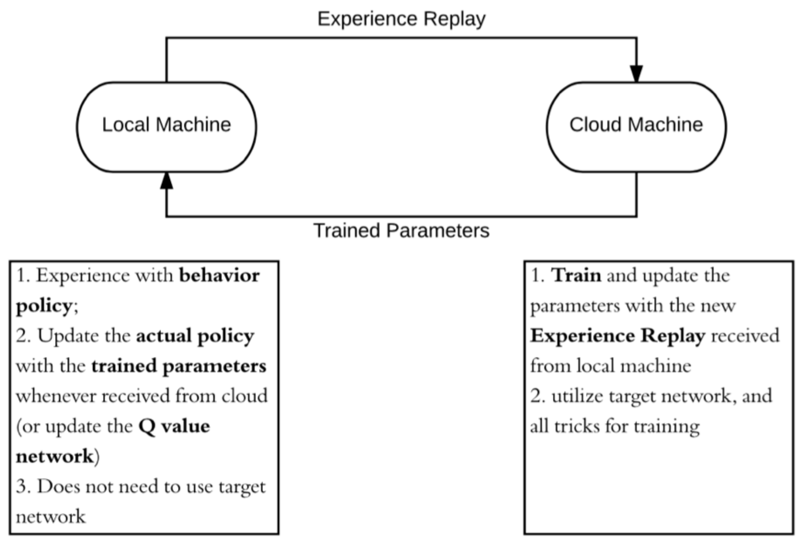
Recent Progress:

1 Oct. --- 7 Oct.	Explord on the previous topic, mainly about Reinforcement Learning and the connection with Control Theory. At the same time, built the paper’s work on Mac.
8 Oct. --- 15 Oct.	Changed to a new topic. Studied about Reinforcement Learning by reading the Introduction textbook by

	Richard Sutton. Read about papers of Reinforcement Learning in games, and tried to **formulate** the problem theoretically. Did simple experiments of **opponent modeling** in **Rock-Paper-Scissor**.
16 Oct. --- 22 Oct.	Built the **screen capturing,** and **keyboard input** functions of the project.
23 Oct. --- 30 Oct.	Did simple experiments of Reinforcement Learning in a simple Bayesian graph, and benchmarked Dynamic Programming method and Q-learning with Linear Regression for policy evaluation. Implemented Deep Q Learning for Atari game Boxing with Tensorflow from scratch. Implemented Deep Q Learning for Atari game Boxing with Pytorch from Pytorch DQN tutorial.

Recent Discovery:

1. Training with DQN is too slow in Macbook. A possible solution might be to use cloud computing, and transfer **experience replay information** to the **cloud server**, and train the parameters on the it, then transfer the parameters back with a certain frequency to the local machine in order to improve the policy. This may also decrease the time delay for training. This approach can be illustrated in the following diagram:



1. Experience with **behavior policy**;
2. Update the **actual policy** with the **trained parameters** whenever received from cloud (or update the **Q value network**)
3. Does not need to use target network

1. **Train** and update the parameters with the new **Experience Replay** received from local machine
2. utilize target network, and all tricks for training

2. Instead of pure DQN, we can utilize the idea of TD method to take into account the noise in the system. Also, some new tricks may also be helpful

Proposed Next Step:

1 Nov. --- 7 Nov.	Try applying DQN to the network, try local machine method to see if it works
8 Nov. --- 15 Nov.	Try cloud training method to see if it works
16 Nov. --- 27 Nov.	To be decided later.
28 Nov. --- 31 Nov.	Ready to work on the report, and prepare for the presentation.
2 Dec.	Presentation