## Intro:

This project is the implementation and research of reinforcement learning algorithms agents based on the video game Minecraft. The reinforcement learning is one of the fundamental aspects of the artificial Intelligent in games area. This project implemented three agents based on reinforcement Learning algorithms on Minecraft and analysis the training result of each bot. The objective of this project is impacts of the configurations of different functions on the learning performance of agent. The training agents using MARLO as framework which is a library built on top of Microsoft Malmo project based on Minecraft(Microsoft, 几几年). The goal of the implemented agents is the goals based on different maps of the Minecraft. The implemented agent achieves the goal by applying leaning functions. The actions of each agent taking is based on the reward rather than environment information or any guidance related to the goal of different maps. The leaning functions of the agents were implemented based on Monte Carlo Tree Search, Q-learning and Deep Q-learning.

This report will focus on the following part as the outline of this report.

1. Introduction of the MARLO framework
2. Background and Algorithms of the
3. Experimental Study
4. Analysis of the results obtained in the experiments
5. Overall conclusions

## MarLo

The target environment of this project is Minecraft which is a popular sandbox computer game. Project MalmÖ is an open source artificial intelligence experimentation and research platform built on top of Minecraft created by Microsoft. This platform offers an application programming interface that permit access to observations, actions and other general data that Minecraft provides. The providing support is powerful for the reinforcement learning, computer vision, multi-agent and related area experimentation and research. The MarLo official website claims that the framework is a high is a high-level application programming interface build on top of MalmÖ for the multi-agent and reinforcement learning area experimentation and research. The implemented agents were based on the MarLo framework and the reinforcement learning algorithms is the Monte Carlo Tree Search, Q-learning and Deep Q-learning.

## Background

Monte Carlo Tree Search is a heuristic search algorithm. The operation principle of the Monte Carlo Tree Search is that the agent will select random moves at each step and the game is played out to the terminal. Each step result of the game used to weight the node of the game tree. The benfit is the better choices for the future node or move selection. There are four basic steps of the Monte Carlo Tree Search: Selection, Expansion, Simulation and Backpropagation.

1. In selection step, the function will start from the current game state as the root and select child nodes until a leaf node L is reached.
2. In expansion steps, the function will create one or more nodes based on the child node and select one node from the created nodes.
3. In simulation step, the function will let the game randomly run from the node selected from the previous step.
4. In backpropagation step, the function will update information which get from the previous running result in the nodes on the path from the expansion selected node to this loop starting root.

Q-learning is a reinforcement learning algorithm. In Q-learning Q is the action-utility function which used to evaluate the pros and cons of taking an action in a particular status. The combination of status and action is limited. The meaning of Q (action-utility function) can be taken as a table that each row in the table records the status, and rewards when selecting different actions. The table is initialized to zero, then each row is updated by rewards through training.

To comprehend and optimize the given deep Q-learning algorithm, the lecture slides and the online materials about the basic idea of the deep Q-learning and the difference and inner relationship between the Q-learning and DQN has been looking through.

It is obviously unrealistic to maintain a large Q table by the traditional Q-learning method due to the excessive state. However, deep Q-learning is a model-free and off-policy reinforcement learning algorithm which solves the reinforcement learning task by playing games in the emulator following an episodes-greedy policy for exploration of the search space. To approximate the value of the Q-table, neural networks train the actual training sample data.

The basic algorithm of the DQN is that

1. Initialize replay memory and action-value function and target action-value function with each random weight theta.
2. For M episodes initialize the state and preprocessed sequence

For T time steps do

* 1. Use epsilon-greedy to select action from action value function
  2. Execute that action in emulator, observe the reward and the next action
  3. Set the next step state equals to this time state, this time action, reward and next time state. Set the preprocess. Then store the transition in the replay memory.
  4. Foreach memory reply, perform a gradient descent step with respect to the network parameters theta.
  5. Every step, setting the action-value function to the target action-value function.

literature review：

##### There are researches on the reinforcement learning in game playing area

## Experimental Study:

The maps training the agent:

In `MarLo-FindTheGoal-v0`, the agent born randomly in the 7\*7 room with a random cubic. The agent's goal is to get the cubic automatically within the given time. After getting the goal or running out of time, the game resets to the initial state.

In `MarLo-CliffWalking-v0`, the game begins in a closed room. The goal of the agent is to reach the coin. However, there are only one way to get that. If the agent taking actions lead fall into the lava, the character dead which means fail to get the goal.

In `MarLo-Eating-v0`, the game begins in open flat space. The goal of the agent is to get the items as much as possible.

The algorithm of this bot is based on the Monte Carlo Tree Search.

1. the selection steps were implemented as a calculate action function which select the actions from the action pool (go forward, move left and move right) using UCB1 function.

UCB1 公式：

The required information to calculate action is the location of initial location instead of the current location of the character. The locations during the moving of the characters are excessively accurate values that lead the Monte Carlo Tree extremely large. The size of the tree could prolong the time span to reach convergence. To minimize the tree size, the coordinate of the characters was ignored expect that the agent was born. Reward could be directly obtained from the MARLO framework. Previous actions were stored in the global dictionary (the variable name is: stateDataAll).

1. There are two circumstances of the expansion steps, all depend on the initial location and the current action chain. If the key, which is a combination of two values mentioned above, cannot be found in the dictionary (stateDataAll), which means this step has not been explored before, agent will create a new state using the key. Otherwise, agent will follow the child action stored in corresponding value of the key.
2. In simulation step, the action of characters taking based on the last action of the previous action chain.
3. In backpropagation step, the function is followed the Monte Carlo Tree Search backpropagation algorithm.

Update value function: 公式

Calculate action function: 公式

1. Q-learning bot

Setting Reward:

1. +0.5 for found the goal
2. -0.1 for out of time
3. -0.01 for every step
4. Initialize: There are five actions (move forwards, backwards, turn left, turn right, remain) of the marlo framework. To simplify the status (Delta x, Delta z, Yaw) recording, the value of the location recorded as the approximated location value which extract from the game. The map was separated by the approximated value function into 49 blocks. To limit the size of the table, there are four actions (move forwards, turn left, turn right, remain) taken for the Q-learning agent from the marlo framework.

Initialize Q table as the dictionary

1. Repeat M times game
   1. Initialize state
   2. Repeat each step of until this state terminal
      1. Choose one action from the current state using the epsilon-greedy policy
      2. Taken action a, observe reward and the next state
      3. Update the Q table and current as the next state.
2. Draw computational graph
3. Deep Q-learning bot

Techniques Implemented

The mechanism of the given algorithm of the Deep Q-learning is that:

1. The function initializes the arguments, dictionaries, hidden layers and channels.

The algorithm initializes an environment and connect to the Minecraft client(marlo-server --port 10000).

Initializes the action-value function(q-function), replay buffer, explorer function and Stochastic Gradient Descent(SGD) optimizer function.

Check available GPU

Initialize a replay buffer and its capacity and experiment profile.

Initialize the agent.

1. Trains the agent while regularly evaluating it using the training and evaluating function from marlo framework following the basic DQN algorithm.
2. Draw the computational graph and save it in the output directory.

However, we change some of the given part:

?????

Experimental Study:

描述为每种算法和游戏进行的实验研究。

When testing the Monte Carlo Tree Search bot, parameters which changed to train the agent were the step size, the learning rate and the maps. Each time training, there are 250 rounds for the agent to learn the way to reach the maps goal by update the state value. When setting the step size, ….. The value of the learning rate of the agent is 0.2, 0.4, 0.7 based on the same step size and map. The observation is that the higher the learning rate is the early the agent find the goal. ….

Action chain

Q-learning

When the map is setting to the `MarLo-CliffWalking-v0`, the bot prefers to stay remain to maximize the reward getting. The reason is that most of the moving action of the bot might lead the bot fall into the lava and fail to get the goal in each round. In Q-learning reward setting, the agent dead reward is -100 that the penalty parameter is much higher than out of time which is 0. So that the agent prefers to stay remain which is a way to maximize the reward. That caused the agent exploration stop and fail to update more state in Q-table. After 250 times training, the agent cannot get the maps goal.

Marlo offical web : https://github.com/crowdAI/marLo