Attempt at Real-Time Minecraft AI with Reinforcement Learning[1]

Juncheol Yang
Newport High School
juneyang2005@gmail.com

Hyeoncheol Yang
Newport High School
henryyang0718@gmail.com

Abstract

Being people who enjoy playing video games, we have attempted to create an AI for the video game Minecraft, capable of determining actions by analyzing the features from the screen like a human player. It is specialized to the named minigame Sumo, where two players in the circular arena fight against each other to 'punch' the opponent outside, which requires the mastery of the game physics, including the ones that exploit its limitations. There already exist many kinds of "hacks" where modded bots intercept the opponent position received from the server and manipulate it to automate player movements and/or gain significant advantage. These traditional hacks could be coded so that they outperform any human, but they, almost always, modify how Minecraft functions and are easy to spot that they are nonhuman programs. Contrary to this, our goal was to create Artificial Intelligence that controls a player much like how a human would by using display output from Minecraft. To achieve this, the Artificial Intelligence was to capture screen directly from Minecraft, process through Neural Network, simulate player movement back in Minecraft to observe the outcome, and re-adjust the NN either according to the reward function that assesses the optimality or the outcome of the entire episode. Accordingly, we have created the training environment and major framework for the AI during the summer break. Our future plan is to complete the NN, train in the custom Minecraft world and in the popular server against real humans, and seek for the imperfections and improvements.

Introduction

Our names are Juncheol and Hyeoncheol, and we are creators of the Minecraft Sumobot. We were brainstorming for a challenging project to work over the summer break at 2021, and we came up with three top candidates. First was the Minecraft parkour bot. Parkour is arguably one of the most technical gamemode, which at the advanced level requires manipulation of the game physics to the extreme to complete a single, or a series of, block-to-block maneuvers. The second candidate was a bot for PvP, one of the most popular gamemode in Minecraft which includes a complex analysis of the situation to determine the best action in a fast-paced battle against the opponent. The third was the Minecraft sumo bot, a gamemode that has similar skill requirements to Minecraft PvP but the difference being two players fight to knock the other off the platform instead of forcing enemy health to 0. Among these three, we have determined to create Artificial Intelligence for the Sumo gamemode, because for Parkour, there always exists a clear solution to a possible "jump" which we knew for certain that a hard-coded algorithm would always perform better than by Machine Learning, and for PvP, we found more complexity in the gamemode than what we found playing Sumo gamemode. We also figured that it will be easy to train in Sumo gamemode because it is easy to implement repeated iterations to train on while being expected to be simple enough for the Neural Network to be optimized in a relatively short amount of training.

[1] This is a self-funded project and the project result with source code has been published at GitHub (https://github.com/TowlTheOwl/SumoBot).

System Architecture

The agent is composed of seven functions. The first function captures the image, second function detects the state of the game, third distributes the state to all other functions, fourth classifies the image and decides on the action, fifth executes the action, sixth records all the actions. The process starts with Multiprocessing starting all the functions, except sixth, to make the functions run at the same time. When the functions start, the first function starts to capture the screen and sends the array of the image to two other functions: the second and fourth functions. However, the fourth function, the function that classifies the image, does not start until the third function tells the function that the game has started. Then, the function classifies the image and gives a 7 digit binary number, followed by a 6 digit decimal number. The first 7 digit decides whether to press or not press keys such as w, a, s, and d. The next 6 digits decide by how much to move the mouse, which controls the aim, to hit the enemy. Then the function sends the 14-digit number to the fifth function, which executes all the actions, by pressing the buttons and moving the mouse. Then it starts the sixth function with the argument as the 14-digit number. The sixth function records the action.

The Minecraft map we created mimics the map of sumo in a popular multiplayer server "Hypixel." The platform is congruent to the one on Hypixel, except that it is not decorated to avoid the NN accidentally catching on decorations unrelated to the Sumo gamemode. Furthermore, it is possible to run a specific reward function we have designed in our map which cannot be used on Hypixel.

The reward function mentioned makes use of the data from the minecraft such as distance from the center of the circular arena and the opponent, the closeness towards facing the opponent, and the time the player was not on the ground, all collected using python program running inside mcpipy, the mod allowing Python scripts to run in Minecraft.

Key challenges and their solutions

The first challenge we faced was the speed at which the program captured the screen. When we made our first prototype of the capturing function, the FPS (frames per second) of the captured screen averaged 3. However, we were able to fix this problem by using the function "Image.save()" from Pillow instead of the "cv2.imwrite" function from OpenCV. We were able to increase FPS from 3 to 13. As we built the program on and on and implemented "mss," we increased FPS from 13 to average of 28, with recorded lowest of 20 fps and highest of 34.

Pseudo Code Version 1:

```
Define Screen Capturing function
  Loop until stopped
    Grab screen with Pillow Imagegrab
    Save the image with cv2.imwrite()
```

Pseudo Code Version 2:

```
Define Screen Capturing functions
  Loop until stopped
    Grab screen with Pillow Imagegrab
    Save the image with Image.save()
```

Pseudo Code Final Version:

```
Define Screen Capturing function
  Loop until stopped
```

```
    Grab screen with mss
    Put the array into multiprocessing queue
```

The second challenge came from the speed of execution. Because a function could only be started after the previous function was finished, the speed of execution was very slow, which would lead to slower decision making or even making decisions that do not fit into the situation the bot could be in. We were able to solve this problem by using Multiprocessing, a built-in module that allows multiple functions to be executed at the same time. Since all functions could be executed at the same time, the program could capture screen, make decisions, execute the action at the same time. This would be beneficial to the program since the program will be making a quicker decision allowing the action to fit better into the situation that the AI is in during the fight.

Pseudo Code Before:

```
Define Screen Capturing Function: Parameter - None
  Capture Screen
  Return: Array of RGB values of each pixel on the screen

Define Decision Making Function: Parameter - Array of the screen
  Calculate using the array
  Output Decision
  Return: Action

Define Player Controlling Function: Parameter - 14-digit number (action)
  Execute the action

Loop
  Start Screen Capturing Function
  Use the output to Start Decision Making Function
  Use the output to start Player Controlling Function
```

Pseudo Code After:

```
Define Screen Capturing Function: Parameter - Queue 1
  Loop
    Capture Screen
    Put the array of RGB values of each pixel on the screen into the queue

Define Decision Making Function: Parameter - Queue 1, Queue 2
  Loop
    Use the value of Queue 1 to decide on an action
    Put the action into Queue 2

Define Player Controlling Function: Parameter - Queue 2
  Loop
    Set action as the value of Queue 2 (14-digit number; action)
    Execute action by pressing keys and moving the mouse

Start all functions with multiprocessing Process
```

The third challenge was transferring the data from one function to a different function. Learning about how to start each function with multiprocessing was easy, but learning about “pipe” and “queue”’ functions were difficult. When we looked at the document for multiprocessing, it explained “pipe” and “queue”, but it was too hard to understand, since we didn’t have much understanding in multiprocessing.

Unable to understand, instead of using any of the functions, we saved the image in the function that captured the screen, and loaded that image in the function that classified the image. With the output, we saved it to a text file, which was read in the function that executed the action. We also tried to save the image and action value into a variable, but it did not go as we expected since the variables did not change in a function even with the global function. However as we learned about “mss”, we also learned about “queue.” In the documentation about mss, there was also an explanation about how you could use mss with multiprocessing. Thanks to this documentation, we could successfully implement the “queue” function and mss into our program.

Pseudo Code before:

```
Initialize variables "image" and "action"

Define Screen Capturing Function: Parameter – None
  Use global variable "image"
  Loop
    Capture Screen
    Update variable image as the array of RGB values of each pixel on the
screen

Define Decision Making Function: Parameter – None
  Use global variables "image" and "action"
  Loop
    Use the value of image to decide on an action
    Update variable action as 14-digit number output

Define Player Controlling Function: Parameter – None
  Use global variable "action"
  Loop
    Execute action by pressing keys and moving the mouse

Start all functions with multiprocessing Process

**This did not work because the variables weren't being updated**
```

What the code was changed to can be seen above: "Pseudo Code after" in second challenge

Another problem we ran into was creating a socket to transfer data over from one computer to another since the python code that finds the value for the reward function cannot be run on both computers. . The reason why we needed a socket to transfer data is to make a reward when doing Agent vs Agent training. Some of the data we transfer are the distance from the center, and where the bots are facing and how far away that is from the opponent. Socket, being one of the modules we first learned not long ago, gave us problems because of our lack of knowledge in socket. We learned about sockets by watching videos about making online video games. We used similar code that we learned in the video in our programs too. We got errors such as “winerror 10053.” Even when we searched on the internet, we couldn’t find how to fix the code. We fixed this code by rewriting it, since we couldn’t figure out what the problem was. After rewriting the code and comparing it with the original one, we were able to find out that the problem occurred because of the decoding of the data. Since we coded it similarly with the code used in the video, and the code in the video used “pickle” that lets a code send a class instance over socket, we also had to change the decoding of the data, since we sent a single number rather than the whole class instance.

Pseudo Code before:

```
Connect to server established with another code
```

```
Server sends data
Receive, decode the data
Send "get" to receive another data (calculated reward value)
Save the data as variable

**ERROR**
```

Pseudo Code after:

```
Connect to server established with another code
Server sends data
Receive, decode the data
Send "get" to receive another data (calculated reward value)
Decode the received data
Save the data as variable
```

Some challenges came from the creation of the training environment. As said, one of the reasons why we chose Sumo as the AI's specialization was the fact that this gamemode was simple and easily trainable. However, it was a challenge to come up with a method to make the Python program handling the AI understand the end of the episode, i.e. the Win/Loss was determined, or the game was forcefully ended for other reasons. It would have been possible to implement the Python code that send such information to the agents using mcpipy, but we already planned a code that would send various in-game data (the environment data that is used for the reward, but not visible to the AI) back to the program that should run in a minimal time. Therefore, we decided to make an algorithm that returns the information required for the training using “Command Blocks,” which allows programming within Minecraft. The system of Command Blocks recognizes two players, sets them up in the Sumo arena, announces the start, broadcasts different messages to each player depending on the result at the end of the episode, and repeats the process after delay. Specifics can be understood by referring to the pseudocode below:

Note that "splash" depicted represents showing text splash on the designated player's screen

Pseudocode of the Command block system in *Minecraft* that controls the *Sumo* game:

```
When two players are present in Location0-0 and Location0-1:

  Warp both to Sumo Arena
  Wait for a few seconds
  Send splash to both players "GO" with White text

  Repeat:
    If Player1 fell out of the arena:
      Place block on Location1
      break

    Else if Player2 fell out of the arena:
      Place block on Location2
      break

    Else if (arbit.) seconds have passed:
      Place block on Location3
      break

    Else:
      Continue
```

```
  If Location1 has block:
    Send splash to Player1 "####" with Red text
    Send splash to Player2 "####" with Green text
    Clear block

  If Location2 has block:
    Send splash to Player1 "####" with Green text
    Send splash to Player2 "####" with Red text
    Clear block

  If Location3 has block:
    Send splash to both players "####" with Red text
    Clear block

  Warp Player1 to Location0-0

  Warp Player2 to Location0-1
```

When the text splash is printed out to the screen, the screen is captured, then a Python function can acknowledge the end of the episode by matching the pixel:

```
Receive Image data from Multiprocessing queue

Initialize list of tuples pixels using the Image data on specific pixels
Initialize list of tuples tv using pre-determined pixel data [1]
Initialize integer match as 0

for each item in tv and pixels as i and j:
  if i==j:
    match = match + 1

If match > (arbitrary threshold):
  set [variable/data that can be seen across different functions] as [Win /
Loss] (depending on what the program was checking for in [1])
```

*Note that the pseudocode above only depicts a part of the hard-coded classification that handles Agent vs Agent training.

The final challenge was being able to control player movements from the supposed Neural Network's output. We were able to acknowledge that the Minecraft player could be controlled using the simulated output. Therefore, we sought for and found libraries that do such, being keyboard and mouse. They could be used such that the output of the NN, which is then determined to be 14 digits, 8 binary digits and 2 3-digit integers, that simulates 8 keys that control the player movements and where the player was facing using those libraries.

Pseudocode:

```
  ...

  Receive variable action from Queue 2
  Release all keys

  If action[0] == 1:
    press w

  If action[1] == 1:
```

```
    press a

  [...]

  Move mouse by (action[8]*100+action[9]*10+action[10]) in x direction and
(action[11]*100+action[12]*10+action[13]) in y direction

  ...
```

Related Works and Backgrounds

Being people that enjoy playing video games, we were also exposed to a lot of different contents about video games. And one of them was a "bot," which acts like a player to play a specific game. Almost every single video game we played had some kind of a bot that executed whatever action it was designed to do. For example, for the game Rocket League, there was a bot that played the game to score goals and defend its own goal. For the game Piano Tiles, a bot was designed to press on the keys that needed to be pressed. This was very interesting to us since we enjoy both video games and coding, and a bot is a combination of both. Some of the programs we saw were also called "hacks," since it gave a player an unfair advantage over other players. While playing video games, we also came across many different hacks, which are mostly relevant in Minecraft, since it is easy to control everything client-side. However, we did not want to create a hack. So rather than creating a part of the game, we created the whole bot to control the keys and mouse of the game to move and make its own actions.

Conclusion and Future Work

We were able to create the basic framework of a real-time Minecraft AI which now can successfully capture screen and execute action. However, since this AI cannot make its own action, we would need to implement Neural Network into the program so that it can make its action and execute it to fight the opponent, rather than making the random actions as it currently does. Also, we would like to develop the neural network in many different ways, so that we could see which way is the best for a Minecraft AI to develop in, and record the winrates of the bots after developing enough to make accurate decisions, in an online environment. One challenge we expect is speed of classification, or the decision making times. Since the array first function, the function that captures the screen, gives to the fourth function, the function that makes the decision, is 1080 x 1920 x 4, the size of the input exceeds 8 million integer values. Because this is too big and will take too long for a neural network to make a decision, we would need to find a way to speed up the process and compress the image in a way that it is readable for the program while not being too large that it will take too long before it can make its decisions. Also, we were able to learn about different modules in Python such as multiprocessing and socket. And as we move forward with the project, we hope we can learn more about Python while coding the new parts of the program.