Universiteit Leiden
The Netherlands

Opleiding Informatica & Economie

The new AI frontier: Minecraft

Mathijs Laban

Supervisors:
Prof.dr. A. Plaat
Second reader:
Dr.M. Preuss

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

31/3/2021

Abstract

Machine learning is becoming a hot topic in the computer science world, since its a powerful tool where a lot is left to be discovered. To enable many practices of machine learning it is essential to test them and games are the perfect testing ground. A lot of work is done on board-games and many 2-dimensional games, while this delivered a lot of insight and uses they hardly represent the complex world around us. To start testing we would need therefore need more complex games and Minecraft is a perfect example of a complex 3-dimensional game while retaining a simple to interact with the world. Now there are 2 main branches in the world of learning in this environment generic learning algorithms and reinforcement learning algorithms. In this bachelor project, I propose an infrastructure to evaluate build neural networks to see how they would preform. Setting up Minecraft to become learn able harnesses a lot of problems and pitfalls which makes it important to take progress slow and focus on small parts at the same time. Thereby I propose while using the application it is essential to keep learn able parts small and constantly add more complexity overtime.

Contents

1 Introduction

As of 2020, there is a trend going on with countless Artificial Intelligence (AI) projects to solve a wide variance of games Rue [2019] such as snake Yeh et al. [2016], but also more complex games are being tackled such as OpenAI's Dota 2 project OpenAI et al. [2019] And Deepmind's Starcraft2 Vinyals, O et al. [2019] to name a few. This trend all started because of a problem both seen in the industry and games. The problem within the games started around the 1970s when games became more widely available to customers, for in those games the adversaries were static in their actions, thereby being independent of the player's actions. These static reactions were therefore easily predicted and exploited. Around 10 years later games such as Donkey Kong got set patterns to react to what the player does, this did show a lot of promise because games were getting more randomization. The reactions from the adversaries were a big step in the direction of smarter solutions, although they where still easily predicted and therefor exploited Guinnessworldrecords [2020]. To make the games more challenging, enemies had to react to the player directly to make them harder to predict and making more interesting gameplay. Through many years with small adjustments of the complexity of the actions and reactions, we saw that steadily those static patterns were replaced with interactive enemies as we see them now in big games like league of legends **?** or Dota 2. Although the bots as of 2020 are still improving they are still easy to predict and have to "cheat" in many games, such as the original Age of Empires 2 Benson [2019], to become challenging to the player. This is where self-learning AI comes in to breach this gap and with the success seen in chess, checkers, go, Age of Empires, and many others. Even Atari games and now the bigger projects of Dota 2 and Starcraft sees success suggesting that the computer is capable. Those AI's eventually got to a level of play seen in competitive tournaments and some cases even beating the world champions, showing that there is a big possibility to make adversaries difficult without resorting to cheats. The games that have been solved have been expanding from finite sets of possibilities such as chess to near-infinite 2-dimensional team-based strategy games such as Dota 2. In this bachelor project, I want to expand further in this new and interesting field by making an learnable environment for the first 3-dimensional game; Minecraft.

1.1 Goals

In this bachelor project I will see if it is possible to make an environment for an AI to play Minecraft using reinforcement learning or evolutionary algorithms. For reinforcement learning, I will use the Deep Q Network (DQN) learning algorithm Anschel et al. [2017] and for evolutionary learning will use the Neuroevolution of Augmenting Topologies (NEAT) algorithm Stanley and Miikkulainen [2002]. The final goal is to see if it is possible to make an AI for Minecraft upholding the rules of so-called "any% speedruns" with a random world seed without cheats [1]. The learning will be left out since I simply don't have the time to learn the AI. [2]

[1] To see the rules explained in further detail visit speedrun.com

[2] Due to the nature of learning all bots will have to join the same random seed, to speed up the process.

1.1.1 Research Questions

To test the viability of a Minecraft AI I will answer the following questions:

- Is it possible to get an AI-controlled bot to play Minecraft?
- What would be a good setup to start an AI learning in Minecraft?

To see the degree of solvability and the hidden problems that will identify Minecraft.

1.1.2 Delivering

This bachelor project will include all code required to replicate the results made. Given the limited timeframe, there will be some work required to finish some of the parts related to learning to fully utilize and run the AI.

1.2 Thesis overview

In chapter 2 (Background) I will lay the foundations needed to understand the terminology used in the rest of the thesis. In chapter 3 (Related work) I will show my reasons and strategy based on prior research in the same research area. In chapter 4 (Setup) I will go over the way I arranged my software and hardware to learn. In chapter 5 (Neural Network setup) I will explain my choices for the input and outputs of the neural networks to learn and work on. In chapter 6 (Problems) I will go over the problems encountered during the process and how I overcame most. In chapter 7 (Conclusion) and further research I will draw my conclusion, and suggest a few area's that need further research.

2 Background

2.1 Minecraft

Minecraft is a survival game based in a 3-dimensional world, where everything is built out of blocks [3]. This does not make it a simple game, however. Minecraft has many ways of playing from exploring the randomly generated world to defeating bosses and even building a computer is possible Price [2015]. In this bachelor project, I will focus on the traditional way of playing it, by killing the final boss "the end dragon". This is a difficult task and will take speedrunners, upholding the same rules as I, around 24 minutes at best. By generally performing a few steps with each step requiring more luck and skill to complete quickly.

[3] For more information about Minecraft visit their website at minecraft.net

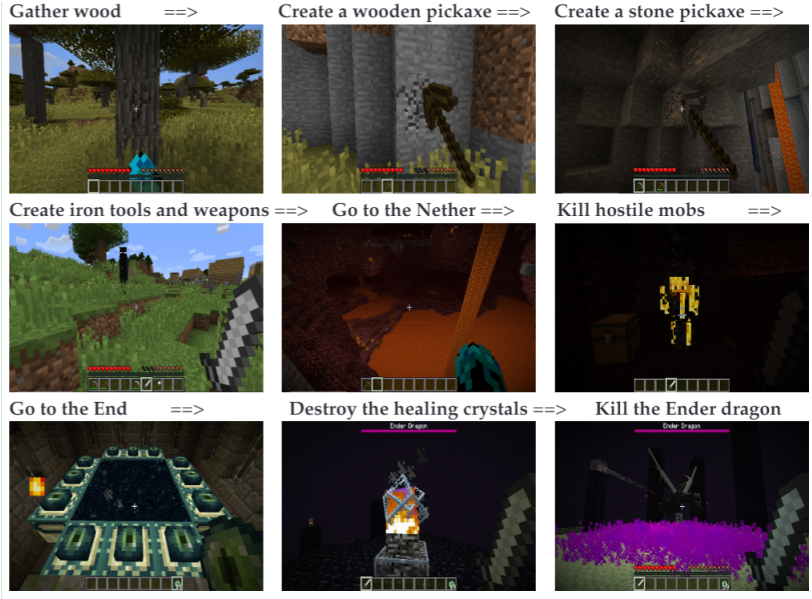

Figure 1: Steps to complete a Minecraft speedrun as of 2020

Most of these steps need to be completed in chronological order because the progression system in Minecraft doesn't allow you to gather materials without the correct tools. Therefore creating a problem for an AI because it will learn that some things are useless because they didn't have the tools required at the time. Random actions will be required in every new state to learn what things are unlocked due to the unlock they did before.

2.2 Self-Learning Artificial Intelligence using Neural networks

Due to the big complexity of Minecraft, I will have to use Neural Networks to compress all the possibilities because table and tree based solutions will take up too much space to run efficiently. For example the possible input-size for blocks with a vision range of max 50 blocks around the bot. The bot will gather the location of the closest occurrence for 83 interesting blocks. Making it max $83! * 50^3$ different options, while growing exponentially with vision range. Note that this is only the block's vision, 1 of the 5 different input classes. Luckily this information field is possible to breakdown and simplify.
With those Neural Networks come the two learning algorithms NEAT and DQN, for more information about both the Neural Networks and the algorithms see the sections below.

2.2.1 Deep neural network

Deep neural networks work with nodes that are connected by weights, with the first layer generally being input data and the last layer being output data or predictions. Using the values and weights of the nodes the value of the next row of nodes will be determined using activation functions. On the weights there is usually some bias or dropout to stop the neural network from over-fitting its weights while learning. Therefore a neural network should have the shape:

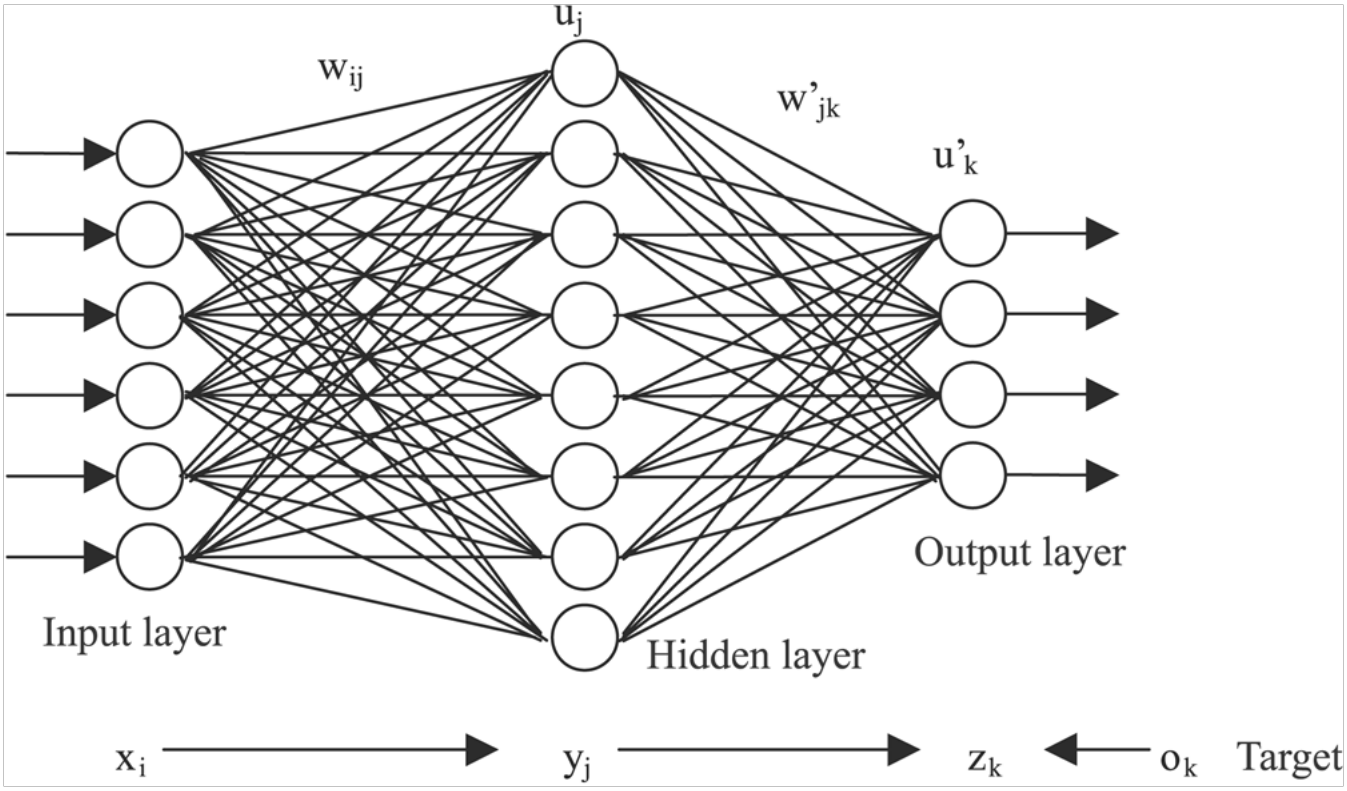

Figure 2: Standard simple neural network layout, with 6 input values and 4 output values

The more connecting lines the more information can be stored in a neural-network and due to the already discussed complexity, I have to use a Deep neural network to expect good results for the problem. The only difference between a standard neural-network and a deep neural-network is that there are multiple hidden layers. Therefore having the general shape:

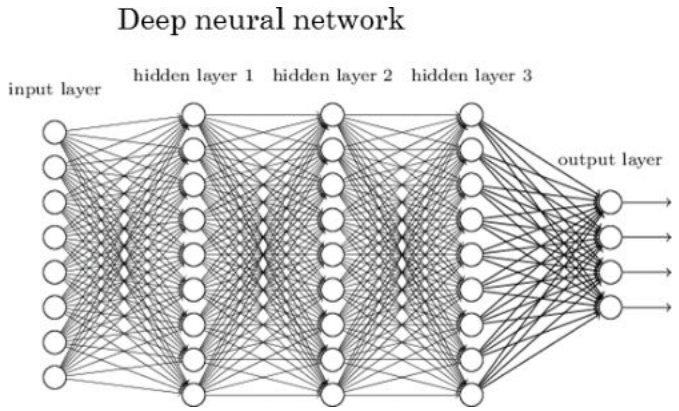

Figure 3: Standard simple Deep neural network layout, with 8 input values and 4 output values

2.2.2 NEAT

Neat is a generic algorithm that tries to learn based on the evolution theories of Darwin Darwin [1859], by using multiple (deep)neural-networks and grading them on fitness. Allowing the fittest to survive. This will therefore constantly drive up the average fitness of all the neural-networks Stanley and Miikkulainen [2002].
The Neat Algorithm will learn in generations where every generation is a group of actors/bots that attempt the problem. The best-performing r% neural networks, based on a points system, are getting randomly matched to create the next generation. The next generation will receive some random mutations and the cycle starts again.

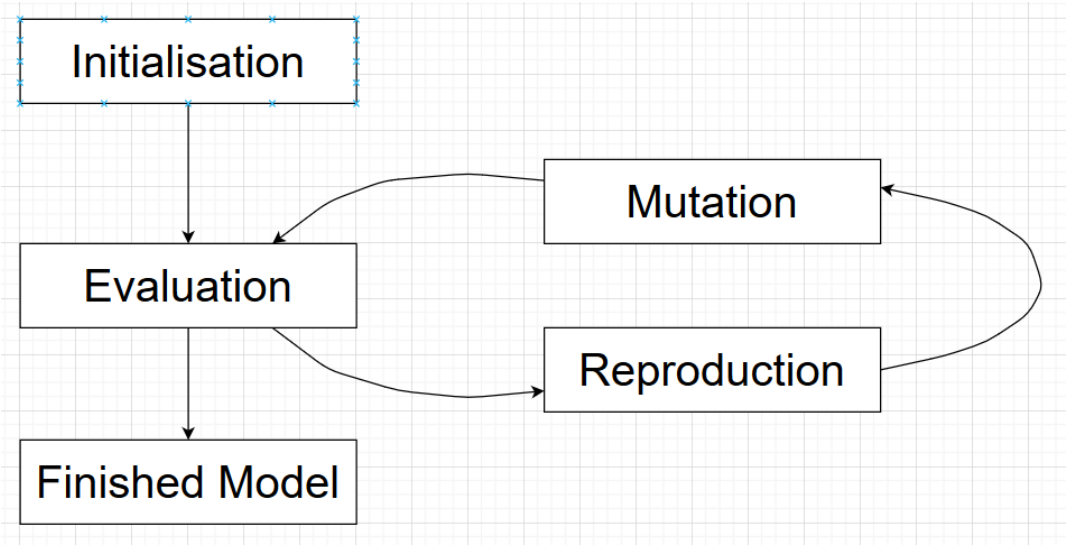

Figure 4: General steps in evolutionary learning

Example:
Let's say we got a simple problem to learn: x + y = z, using a population size of 3. The actors will all be grated on a test with 10 random x and 10 random y, for each generation. Where the points are how close they are.

For simplicity sake lets assume we already know what the genome should be of a perfect specimen 5.

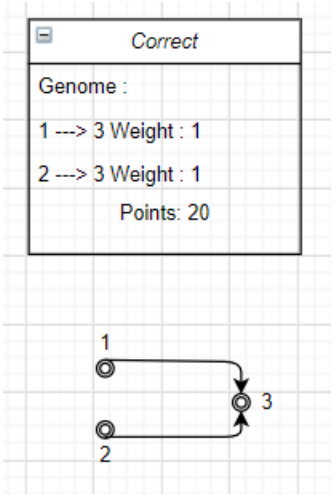

Figure 5: Perfect specimen that got the complete 20 points on every assignment of random x and y

We start off with a normally distributed first generation where the genome is set.

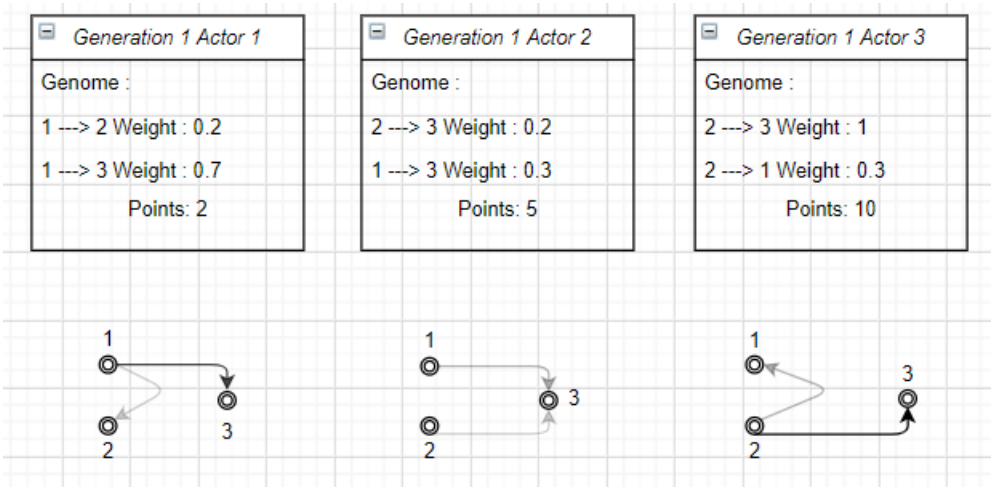

Figure 6: 3 individuals of the first generation

From these 3 we take the top 2 best performing individuals which are Actor 2 and Actor 3. They have a chance to give their genes over to the next generation.

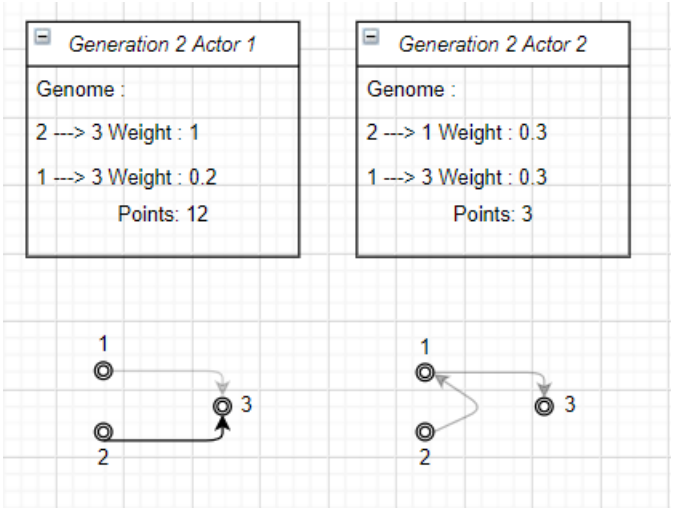

Figure 7: 2 individuals of the second generation

As you can see the second generation is comprised of trades from their parents. Actor 1 is comprised of 1 gen from first gens actor 2 and 1 from first gens actor 3, same goes for Actor 2. With this mixing up it is completely possible that individuals will score lower than their parents but if you make the set big enough they will always score equal to or higher than the last generation.
after the cross-section it is posible to get random mutations.

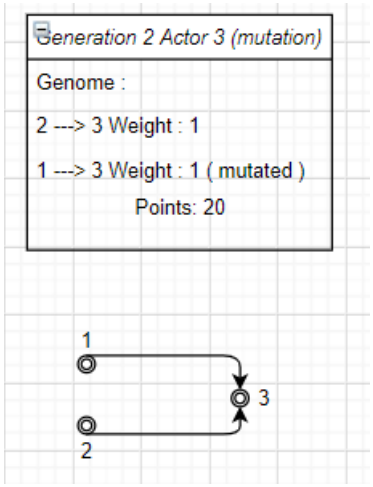

Figure 8: A Mutation occurred in Actor3 the mutated gen is marked

This is a very lucky mutation for it could have gotten anything even adding for example more complicated lines from 2 –> 1 or the other way around. Now we have reached the fitness we wanted to reach we stop the generations and use actor 3 to solve the problems.

2.2.3 DQN

DQN is a reinforcement learning algorithm that tries to learn based on actions a bot has done in its environment. Actions that where beneficial will be learned making the bot do them more often,

while action that where harmfull will be disincentivesed. DQN uses only one neural network and calculates its way to perform better Anschel et al. [2017].
The DQN algoritm will collect input information, output information, action preformed, its reward it has got for the performed action and if it is done with its run. As long as its run is not finished it will create a big backlog of information. After the run a part of this information is used to learn, where things that received a lot of points are prioritised. After calculating gradient decent the neural network is updated and a new run will start.
example: let's say we have the same problem as we faced with evolutionary learning. and let's assume we have a perfect brain that can solve this issue given by Figure 9

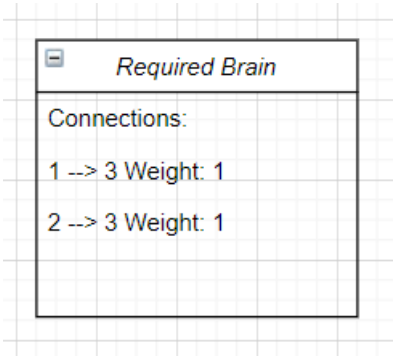

Figure 9: A perfect brain given the problem

Reinforcement learning works based on the individual answers and later getting the value of the action performed. Actions with a better than random value will be rewarded and therefore the brain will be recalculated to allow the weights to give a closer answer to the action performed. After the calculation, the weight will therefore have come closer to the perfect answer. While worse than average answers will be disincentivized also pushing the answer closer to the optimal until the optimum is reached. There are 2 types of answers possible: exploration (taking a random action) or exploitation (using the already learned strategy). the higher the exploration the better the chance to find new strategies while having a higher chance to fail. as we see in figure 10 the brain constantly keeps improving on the problem.

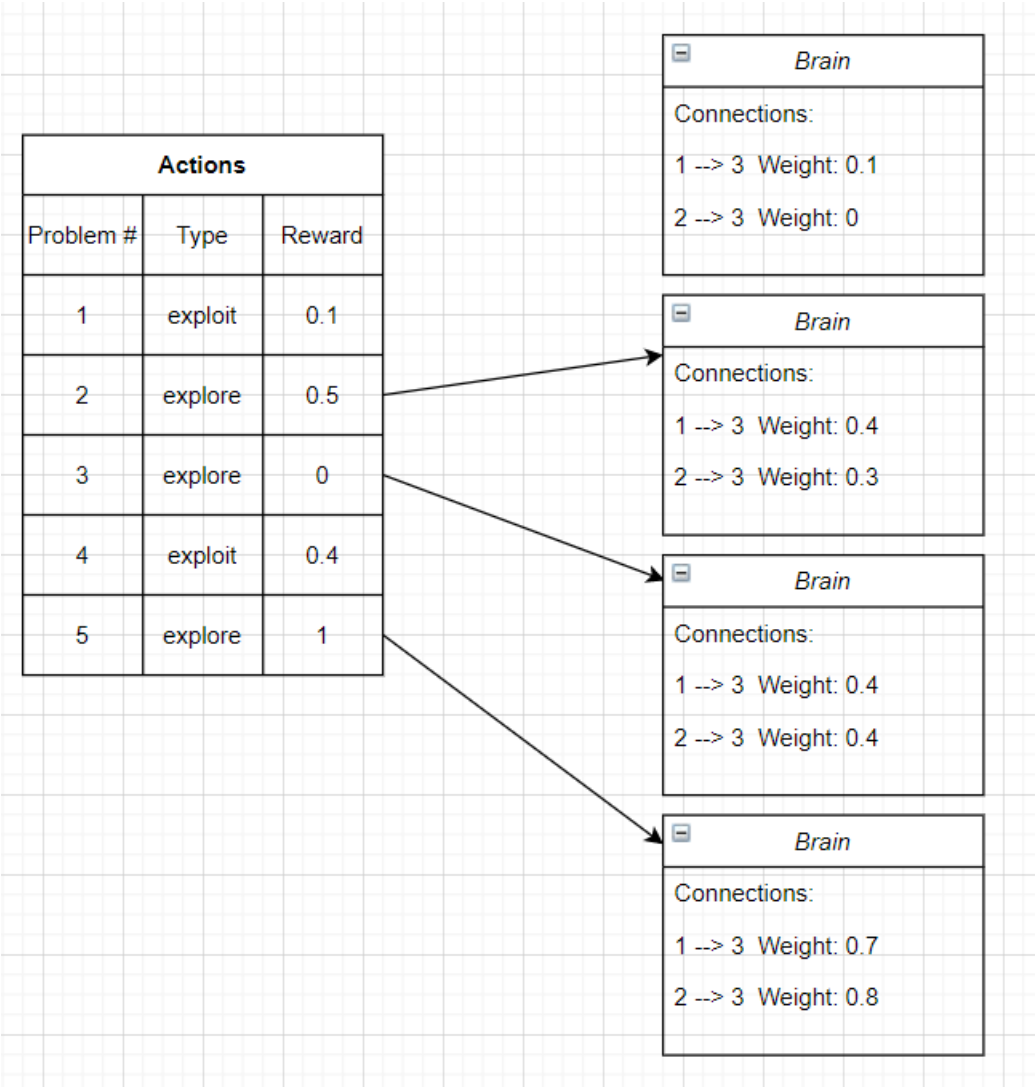

Figure 10: A simplified view on DQN

3 Related Work

Minecraft as explained in 2.1 is a game with complex states, space, and solutions to advance tiers in armor weapons and tools to complete the game. This makes it a complex problem to solve since traditionally this makes it difficult to learn. This difficulty in learning combined with the large state-space will therefore most likely fail to find the solution quickly. Making it an interesting problem, but when OpenAI managed to create an AI for both Starcraft 2 and Dota 2. Starcraft 2 being a challenge on strategy and countering the enemy, and Dota 2 the power of teamwork. It should be possible if there is enough computing power involved.

3.1 Starcraft 2 Deepmind

Starcraft 2 is a strategy game where both players need to create an economy mining rocks while making army's to stop the other player from doing the same. Although it is an older game there are constantly new strategies found and exploited to win. The team of Deepmind managed to solve this with success even managing to beat competitive players showing that the computer might be better at Starcraft 2 then humans Statt [2019].

3.2 Dota 2 OpenAI

Dota 2 is a multiplayer online battle arena pitting 5 players vs another 5. Where both teams must gather gold from monsters or killing the other team to be able to destroy the enemy towers and finally the ancients. This requires not only individual strategy but also teamwork. 10 months with 800 PFLOPs/s-days later the AI made by OpenAI managed to defeat the top teams of competitive players OpenAI [2019]. Creating a complete benchmark of 99.4% win-rate against players in public matches Wiggers [2019].

The key-points: both pieces of research have struggled with the problem of exploration, where Minecraft would be an excellent testing ground. They both also started building small and added more complexity into the systems as the project went on.

Microsoft also saw the possibility that lies within their game Minecraft, and started a challenge; could a self-learning AI find a diamond.

3.3 Microsoft Challenge

With a big price pool and many smart minds playing the game the solution should not have been the scale of the problem it became; for it still isn't solved Microsoft. To aid the combatants Microsoft even created a separate launcher and learn-able data snaps. This data however is tightly linked to the goal of the challenge of obtaining a diamond.
Although this sounds great it also restricts the user not being able to do the same development cycles as the games above. As both games used "cheat" mechanics to know information that should not have been known by a normal player playing. Due to this extra information, the AI managed to learn quickly. After the key mechanics were mastered the extra information wasn't needed anymore.

The key point to take away from the Microsoft challenge and both Dota's and Starcraft's AI's was that solving Minecraft would require a researcher to find a way to make exploration quicker by using extra information and a humble beginning.

Some ambitious projects have already started within the Minecraft community itself like Baritone and Chaos-craft.

3.4 Baritone

Baritone was an adaptive A* algorithm to get from A to B in Minecraft, but it managed to implement neural networks and started learning. Now being able to do lots of small tasks such as

fishing and farming. Showing that making small portions it could constantly get more complicated. Now it can even build entire structures on its own. Baritone [2019]

The key point to take away from baritone was that a Minecraft AI is possible if the tasks were to be split into smaller portions see 2.1. This is completely in line with other big projects as mentioned before. Smaller steps are easier to learn and created a stepping stone for the learning process. Smaller portions are a potential way of dealing with the exploration. If every step were to reset its epsilon, the chance of doing random actions, it would learn as if it's tackling a new problem while keeping most of the old data, what it essentially is therefore creating a "multi-step neural network". But using Baritone itself would not be beneficial.

3.5 Chaos-craft

Chaos-craft is a project done by a single Youtuber "Schematical" who was working on making a self-learning AI using a generic algorithm. It looked promising yet he got stuck on the exploration because of the large number of options a bot could use or do Schematical [2018].

The key point to take away from it was that it would certainly be possible to make it work if only there is a way to combat the problem of the large action possibilities on every single moment. This problem I already partly solved looking through the tips and tricks of the other projects, splitting the problem into smaller parts, and starting with less complicated state-action spaces to learn the fundamentals and reinforcing that into the more complicated neural networks later.

3.6 Key design criteria gathered from similar projects

To stand a chance in building a good design and knowing if you are going in the right direction you need extensive knowledge about the game you are working on. This works in two different aspects, information reduction, and state evaluations. To create an AI it is essential to keep the required input and output data as small as possible while getting as large as possible impact on performance. To evaluate in a point system it is also important to know the relative impact of certain items and actions for the future improvement of the bot.
Another essential part of information reduction is gathered through building up the AI in small steps, therefore being able to remove or reduce information that is already captured from previous input streams. While keeping the stability of the code and the complexity down. Here it is essential not to rush advancements for the first few generations will require time to hone into the problem before becoming stable enough to expand. From all projects mentioned above, it is seen as essential to create a stable baseline with essential interactions first before expanding and looking for more performance.
When learning it is usually accepted that in the earlier stages it is more advantageous to learn quickly. Most of the information at this stage stored in the neural network is next to useless. This allows for a higher learning rate. While later on in the process you want to keep most information intact in the neural network, therefore, having to learn slower. Therefore it is a good practice to gradually lower the learning rate.

Niche actions have always been a problem for it requires a lot of exploration to find while exploration becomes more expensive for it has usually already found the best solutions for most problems. Therefore splitting the learning process around known niches will speed up the exploration. As seen with the Roshan problem in Dota 2. Understanding this allows the builder to assign extra effort to these parts and creating a multi-stage neural network. By resetting the epsilon decay around these essential parts. Increasing the number of random actions and increasing the exploration. Dealing with niche actions in this way will limit the amount of freedom in the learning process, however, to gain increased learning speed.
To speed the learning process up further it is nearly essential to start with extra information that should not be known to a bot as compensation for instinct and prior knowledge of the human counterpart. For example showing blocks that are underground, allowing the bot the knowledge to know that diamonds are deep in the ground. This is only required for the learning process for it will not be required when the bot has grasped the underlying insight.

4 Setup

4.1 Design

The setup relies on 4 parts:

- The Control-Server
- The Bot-Controller
- The Minecraft-Server
- The Learning-Center

The complete setup should look like Figure 11.

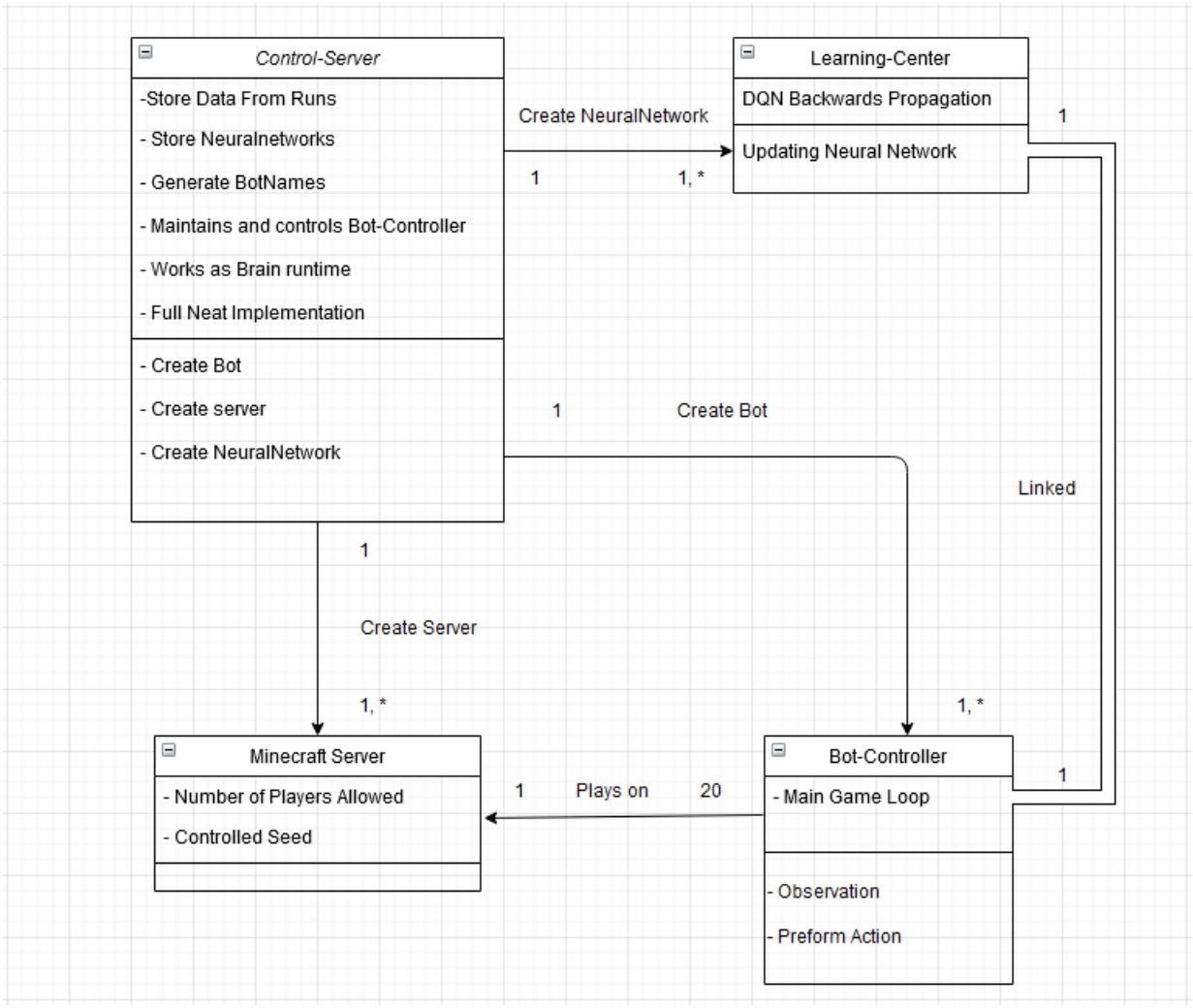

Figure 11: The design of parts

4.1.1 The Control-Server

The control server is the essential part of the entire project because it would have to keep everything linked and under control.

"localhost"	Startup of a Minecraft server
"localhost/startbots"	Startup of a bot on the minecraft server with a default new model file setup for DQN
"localhost/startbots?neat=True"	Will set the bot to be learning with neat
"localhost/startbots?model=<YOURMODELFILEPATH>"	Will load the neural network from a file, this only works for DQN for now.
"localhost/global"	The place to receive learn-able data in Json form

Table 1: Startup Options

The interaction with the control-server is limited to the starting of the Minecraft server and the bot-controllers. The bot-controllers do have multiple options to make them more suitable for the occasion as seen in 1. it's only possible to start up one Minecraft server and 20 bots on it. Though the setup is easily expandable since all components do not have much overhead. The low overhead is mainly due to the nature of nodejs ability to work parallel. The bot-controller needs to communicate it's environment back to the server through the following options 2.

"localhost/vision"	Bot-Controllers way of communicating information to the Control-Server
"localhost/deathnotification"	Bot-Controllers way of communicating of death of the bot

Table 2: communication Options

Due to its many connections, the control-server would have to keep up for essential communication it would have to be an online interface. This would make it possible to connect and run bots from remote locations. Making a Flask-webserver a good choice. Due to making the application in python it allowed for direct implementation of Keras the overlay of Tensorflow for the neural networks.

4.1.2 The Bot-Controller

This part of the setup is using an application programming interface (API) for Minecraft to make "bots" that could do all the things a normal player can when playing Minecraft.

The Bot-Controllers tasks are mainly tied to the game mechanics of Minecraft including the main game loop for the bot. It collects observations and performs the actions given by the control-server for more information see Neural Network setup 5.

I used the same API as Choas-craft named Mineflayer which translated most actions into a high-level API and gave me the ability to run bots on a remote server.

4.1.3 The Minecraft-Server

This part of the setup is the world where the bots will learn and hopefully conquer their task of destroying the final boss the Ender dragon.

The Minecraft-server is set without spawn protection to make sure the bot can do everything, this doesn't impact future generations because the world gets reset after every generation. All the bots are also spawned at the same time or within insignificant time apart to make interference small. The seed of the server is set so learned locations of resources will speed up the learning process.
To make learning possible I will use Minecraft version 1.12. This is an older version of the game that doesn't include the underwater theme because this update brought a lot of new items and enemies into the game that would make learning difficult. 1.12 is an excellent version for it was a very stable one where most of the useful items and mechanics were added to complete the game.

Complications: To join a Minecraft server you are required to own a Minecraft account for every player you join with. This caused a big problem since I don't have multiple spare accounts to learn therefore threatening to clock the entire process down by learning with one account. I had to find my way to cheat around this, an "offline" version of the server. Offline servers in Minecraft are dangerous since it will not check the accounts with Microsoft, nor will it check the connection safety for they have no internet connection yet it is still possible to join these servers from remote locations if the creator would port forward. This made it possible to run multiple accounts at a time.

4.1.4 Learning-Center

This part of the setup is the updater for the neural-networks and will learn after a generation of bots have done their run.
Due to this process taking a long time I had to separate this from the main server not to overload it, it causes extra challenges for this process still needs to be made elsewhere.

4.2 Required Hardware

The hardware required to run everything is split-able in the 4 main sections as seen in Figure 11, making it more accurate to calculate and allowing optimized setups when ran on a bigger scale. The average update time of gathering information and performing the actions is 3 seconds which is good enough for the game Minecraft, for most actions take longer, but could be improved with better hardware.

4.2.1 The control-server

This server doesn't need much as it comes to hardware for it mainly does communication, though it needs a stable internet connection, and a good CPU to calculate the predictions from the neural network. During my runs I have used my laptop containing: i5-8265U CPU, 1 GB RAM, no graphics card, and stable 20 Mb/s internet connection.
This is overkill for the problem, but the control-server is the most crucial part of the software if this falters everything will suffer.

4.2.2 The bot-controller

The bot controller is dependant on CPU power to perform its actions, due to the still unoptimized software it could only run 8 bot-controllers on an i5-8256U CPU for longer periods. This is why running multiple bot-controllers will have to be done on multiple workstations.

4.2.3 Minecraft server

The Minecraft server doesn't need much in terms of CPU power or GPU power but is very memory heavy requiring 1 GB to keep the server running and a furthermore 300 MB for each additional bot ran on the server to keep the server stable.

4.2.4 Learning Center

The learning center requires GPU power to perform its tasks. How much is needed however is until now undetermined.

5 Neural Network setup

5.1 Input Information

To create a good working neural network it should have all relevant information, therefore I have made choices on the information needed to make them. There are 5 main sources:

- Bot-vitals
- Layout of the land
- Blocks

- Entities
- Inventory

5.1.1 Bot-vitals

To make decisions the AI needs certain measures of health, food, armor, and others to decide if taking damages is possible to reach a goal. With vitals is mend the status of the agent or bot. With this version of the Neural Network the vitals Health, Food, and Location of the bot are added since they are required to do the basic things to survive longer amounts of time. Extra vitals to add with further modifications are time (of the ingame world), armor, weapon, breath and perhaps also poison, wither-effect, and other none essential effects.

5.1.2 Layout of the land

Giving the bot the information on where it could walk given terrain, therefore being able to avoid deadly falls. Hereby I gave it a $5x5x5$ vision of the terrain with a Boolean if there is a block at the location.

5.1.3 Blocks

The bot should know what blocks are around him to be able to find useful blocks such as iron. Hereby I gave the bot the closest of all useful blocks within a 50 blocks distance.

Concessions: It would be more human-like to have the information of all blocks visible but this would cost too much time and information fed to the neural network. Therefore only the first and closed block is shown for it would always be the quickest option to get. The bot is getting more information than a normal human for the 50 blocks in radius is including x-ray visions, therefore, being able to look through solid blocks.

5.1.4 Entities

Entities are both enemies, friendlies, and dropped items in Minecraft, and knowing their locations are vital for survival.
In this version of the bot-controller, all entities with their location are shown to the bot. In further versions, it might be beneficial to add the health of enemies to the information block and perhaps limit the distance of enemies visible.

5.1.5 Inventory

The items in the bot inventory, to know crafting possibilities and evaluating the bots. Therefore the bot gets a list of items and their quantity that is present in the bots inventory. In further versions, it might be beneficial to add "tool tiers" in the inventory.

5.2 Output Actions

To play the game the bot needs to perform a wide range of actions, such as mining, crafting, and fighting. The actions are ordered into 5 different groups: Movement, Crafting, Smelting & cooking, Fighting and Using Items.

5.2.1 Movement

Movement is essential to find all the necessary items required to complete the run. Therefore all movement is added in this version, in further versions, it might even be able to add are building to create new paths.

5.2.2 Crafting

To be able to get new items and be able to get achieve all goals. Crafting without metadata is in this version of the bot-controller, in future versions, it might even be possible to add metadata to the crafting to repair tools.

5.2.3 Smelting & Cooking

Preparing raw materials for crafting and cooking for food replenishment. This isn't working in this version of the bot-controller, but will have to be added in future versions for getting iron ingots requires smelting.

Concessions: This is not working correctly at the moment, this requires adjustments in the neural network and isn't required for the first steps of the run.

5.2.4 Fighting

Fighting the main boss is not the only thing required to complete the run and multiple enemies need to be killed. In this version of the bot-controller fighting isn't implemented, in future versions, it will get essential for multiple steps to rely on it.

Concessions: Fighting is not jet implemented for it both brings a lot of extra actions to the learning thereby making it more difficult it is also not required till later steps in the problem.

5.2.5 Using Items

Using items such as ender purls, the eye's of the ender to complete tasks. This isn't implemented in this version of the bot-controller though this will be the first to be added in future versions for it adds eating, building, throwing items, and much more.

Concessions: This will have to be adjusted since the use of items is not fully tested, though it will not be required for the first view generations.

6 Problems

In this section, I will go over all the problems I had while making the AI.

6.1 Tensorflow-Nodejs

Tensorflow is currently one of the most versatile options as it goes to neural networks, with a large user base. At the start of my project, the beta version of Tensorflow-Nodejs was released, the perfect solution since I was already bound to Nodejs due to Mineflayer. Sadly it didn't have the required versatility that I needed for this project and would also not work well in a webserver. Therefore I decided to change it to the standard version of Tensorflow in Python.
This standard version of Tensorflow is very complicated, therefore I used the Keras overlay to simplify the process. This worked wonderfully for the simplification outweighs the versatility loss.

6.2 Running multiple instances of bot-controller at once

I started programming the server and bot-controller without async function calls which caused a lot of problems with handling the information that I got from run-time and practically made it impossible to run multiple bots at the same time. This was solved by adding most of the async functions in the bot-controller and being careful with the data-storage by only adding information when necessary or in-class objects saving me the hassle for extra locks and async protection.

6.3 Multiple window crash of API

Mineflayer also had a few surprises and bugs within itself, for example, it would randomly crash when crafting on a crafting table was initiated due to opening too many "windows".

6.4 Numerical and categorical data

There are many forms of data that the AI needs to accurately predict its next best moves, this included both numerical and categorical data. The categorical data is hard to represent in a neural network for red is not two times the value of green or blue is not halve the value of red. This was solved by only giving the distance to the closed object for each kind of useful object in view. This both reduced required information, and still gave all the information needed for a bot to make its decisions.

6.5 Different planes of view

The bot eventually needs a lot of information to make decisions jet this would quickly become the problem of "the curse of dimensionality" for the more inputs the more information it needs before it has all the values correct. Therefore the neural network needed to be split up into smaller

portions that would give a dance representation of the information needed in 5 different planes.

- How am I doing in terms of food, health, and location?
- Where are the blocks around me?
- What is the distance to all different blocks I can see?
- Where are the closest enemies?
- What is in my inventory?

To reduce this into learn-able input a lot of things needed to be simplified as explained in 5.

6.6 Plugin Problems

When the crafting problem was solved, everything should have worked perfectly yet, the updated crafting system required the old not plugin version of block finder that is required for the scouting of the blocks see 5.1.3. This is not solvable by dynamically loading the plugin for that would take too much time.

6.7 Sending information though query-strings

URL-query strings are very useful and quick options for sending information to a website but there is a limit of 2048 characters in the size of those URLs making them not viable for the bigger version of this project. This is expandable however but becomes vulnerable to DDOS attacks in the process. Therefore request bodies needed to be used.

6.8 Promise Objects

Some promise-all requests simply take too much time and have to release their data midrun to make the bots react quickly to the information that is obtained. Though because this is midrun it is hard to either reject or resolve all cases leaving the data in a promise object and not usable.

6.9 Sparsity of objectives

Minecraft is a game with a problem that obtaining goals will constantly get more challenging for the higher quality of the material the rarer it is. Therefore it will be likely that though strictly random actions it will take way too long to find anything and therefore will need to be stimulated in additional ways to speed this up.

6.10 Neural Network Integration

To make DQN work you need to supply it with a good neural network to work with but with the large amount of information required to make educated guesses the neural networks will also get big, and therefore slow.

7 Conclusions and Further Research

Minecraft is a complex game due to its many options. The large amount of information needed to make good choices and to learn effectively will have to be simplified or compressed. Otherwise, it will take too long to get a working bot. On top of the amount of information and actions, some actions are niche and are desensitized by previous learning steps to complete and rely on random actions. AI has two well-used options to solve niche problems by either giving the agents a baseline played by a human to start, or splitting the game into small steps to elevate most of the niche problems. By giving a baseline you limit the amount of learning space a lot and this was not the intended way of solving this game. Therefore splitting the game into small learn-able portions is the way to go.

7.1 Is it possible to get an AI-controlled bot to play Minecraft?

Running an AI in Minecraft is possible for both reinforcement learning and generic learning algorithms, the only problem with Minecraft is that the game has a lot of options and information that is due to its simplistic look underestimated. Therefore a project such as Minecraft will take a lot of time to make the more things you take into consideration quickly creating a complex neural network to solve even the basic actions done by a bot.

7.2 What would be a good setup to start an AI learning in Minecraft?

As for a start/setup for Minecraft, it is recommended to start small and to not overcomplicate things for this would lead to a lot of problems later on in the project as seen in mine. It is also advisable to split the game into small portions that are learnable for it would otherwise drown in its many niche problems. The setup should be split as much as possible too to make full use of the parallel running of bots and still keeping the amounts of computing power down. The setup of splitting the problem into four parts as seen in Setup 4.1 worked well for me.

7.3 Further research

Because this program is not fully learning there is a lot still to be done. But the main area's that need improvement are:

- Simplifying input and output data.
- Connecting with multiple computers and workstations.
- Increasing the amount of data input and output.
- Implementing better asynchronous options for long-taking processes.
- Better solving strategies with niche actions.

7.3.1 Simplifying input and output data

Simplifying input and output data while keeping most if not all relevant information would optimize the algorithms a lot. for example, the nearest dirt or grass blocks usually don't give much information and might be entirely irrelevant saving on input data required. This is the same edge with actions for mining the closed block of a certain kind is usually enough information for the gathering saving on having to choose between blocks. But also idea's as not showing locations for blocks and entities when the required items are not unlocked yet could save on data.

7.3.2 Connecting with multiple computers and workstations

Everything is better when you do it together! This is also the case with learning the more computers are working on learning the quicker the bots will perform. Therefore it needs good and safe protocols to share bot-controller files and share the current up to date neural network models to continue learning. This would require a good control system to make sure work is not done double.

7.3.3 Increasing the amount of data input and output

By adding more options for the bot, the bot can come up with more strategies and perhaps even find better strategies to solve the game than known now.

7.3.4 Implementing better asynchronous options for long-taking processes.

Some processes within the bot-controller take a lot of time due to its search-space this could perhaps be done more efficiently by doing more asynchronous work or improving the existing algorithms to search for items or perform actions.

7.3.5 Better solving strategies with niche actions

All the problems that require a good balance between exploration of new actions and using known best moves are still a big problem within the entire field of self-learning AI's and this shows itself with problems that are in itself are niche. Though splitting the problem into smaller bits is effective, there might be way better solutions for this exact problem.

References

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 176–185, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/anschel17a.html.

Git Baritone. Git project baritone, 2019. URL https://github.com/cabaletta/baritone.

Julian Benson. The ai in 'age of empires 2' has been cheating all these years. 2019. URL https://www.gameinformer.com/preview/2019/08/23/living-up-to-its-name.

Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859. or the Preservation of Favored Races in the Struggle for Life.

Guinnessworldrecords. Robbie lakeman:highest score on donkey kong, 2020. URL https://www.guinnessworldrecords.com/world-records/highest-score-on-donkey-kong.

Microsoft. Minerl competition 2019. URL https://minerl.io/competition/.

OpenAI. Openai five defeats Dota 2 world champions. 2019. URL https://openai.com/blog/openai-five-defeats-dota-2-world-champions/.

OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. 2019. URL https://arxiv.org/abs/1912.06680.

Rob Price. The 11 most ambitious virtual computers ever built inside minecraft. 2015. URL https://www.businessinsider.com/virtual-computers-built-inside-minecraft-2015-2?international=true&r=US&IR=T.

Noah Rue. Artificial intelligence & game development: Recent trends, 2019. URL https://becominghuman.ai/artificial-intelligence-game-development-recent-trends-a08a67769a63.

Schematical. Git project chaoscraft, 2018. URL https://github.com/schematical/chaoscraf.

K. O. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1757–1762 vol.2, 2002.

Nick Statt. Deepmind's starcraft 2 ai is now better than 99.8 percent of all human players, 2019. URL https://www.theverge.com/2019/10/30/20939147/deepmind-google-alphastar-starcraft-2-research-grandmaster-level.

Vinyals, O, Babuschkin, I, Czarnecki, W. M., and & Et, al. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nature*, 575:350–354, 2019. doi: 10.1038/s41586-019-1724-z. URL http://dx.doi.org/10.1038/s41586-019-1724-z.

Kyle Wiggers. Openai's Dota 2 bot defeated 99.4 procent of players in public matches, 2019. URL https://venturebeat.com/2019/04/22/openais-dota-2-bot-defeated-99-4-of-players-in-public-matches/.

Jia-Fong Yeh, Pei-Hsiu Su, Shi-Heng Huang, and Tsung-Che Chiang. Snake game ai: Movement rating functions and evolutionary algorithm-based optimization. 11 2016. doi: 10.13140/RG.2.2.33593.36969. URL http://dx.doi.org/10.13140/RG.2.2.33593.36969.