Killing Zombies in Minecraft With Deep Reinforcement Learning

Hiroto Udagawa, Tarun Narasimhan, Shim-Young Lee

December 16, 2016

1 Introduction

In this project, we train an agent in the video game Minecraft with reinforcement learning to kill as many zombies as possible in a confined space. Our agent is only equipped with a sword to attack the zombies, which can themselves attack and kill the agent. The objective is to have the agent learn what policies to execute based on the visual information it receives about the surroundings from the raw pixels of the game play screen and the reward structure we have designed. Using a convolutional neural network and Q-function reinforcement learning, the agent showed definite improvement in its ability to kill zombies. This project displays the promise of combining deep learning with reinforcement learning to achieve non-trivial tasks.

2 Related Work

There have been several projects in this area which we drew upon for our project. The major inspiration for our work was DeepMind's 2013 paper on deep reinforcement learning, in which the agent was trained with a single model architecture (Deep Q Learning) that uses a convolutional neural network to perform a variant of Q-learning, and successfully played 7 different Atari 2600 games, in a few of which the agent outperformed human experts [1]. Instead of solving the standard action-value function (Q function) with Bellman equations, deep convolutional neural network was used as a nonlinear function approximator. DeepMind also employed a biology inspired technique known as experience replay, which attempts to fix the divergence due to correlation between sequential inputs. While a different network was trained for each game, the same architecture and hyperparameters were used across all of the games. Furthermore, no prior (or game specific) knowledge was used, but only the raw pixel values were fed to the network as inputs. DeepMind's major achievement was using extremely high-dimensional states (the visual input from the agent) and no information about the game rules to train an algorithm which could outperform skilled human players, and we aimed to replicate this in our project.

For the actual implementation of our project, we built on the work of Chen (2015), which also relied on DeepMind's DQN and applied the same model architecture and algorithm to the game Flappy Bird [3]. We sought to expand on his work by applying DeepMind's architecture to a more complicated setting. While Flappy Bird has only two possible actions and no opponents, Minecraft has a continuous action space and our setting includes zombies actively trying to kill the agent.

3 Minecraft and Project Malmo

For the setting of our project, we chose Minecraft, a popular sandbox video game that was released in 2011. It allows players to control an agent, through which they can explore the map, build structures, craft tools, and destroy computer-generated zombies. Project Malmo is an open source platform for Minecraft created by Microsoft that offers an interface for researchers to experiment with different approaches to artificial intelligence by controlling an agent through Python scripts.

Malmo is a useful environment for artificial intelligence research because it allows users to test endless situations within a relatively simple framework by controlling agents' movements and environments via code. By combining different states, actions, and spaces, researchers can create different environments and define various tasks for which the agent trains with reinforcement learning.

As a simulator, Malmo is more constrained than many other simulators which researchers have employed. Microsoft designed Malmo to be an interface with Minecraft, but not to provide the agent with any information about Minecraft. Thus, the default in Malmo is for the player-controlled agent to not have any information about its environment or the rules of the game. This feature makes Malmo very convenient for reinforcement learning as this replicates the exact kind of environment we want for reinforcement learning: in order to maximize difficulty, the agent should be trained only from its own visual input. For example, the agent does not "know" that the *attack* action kills zombies; it learns this through training.

An additional constraint posed by Minecraft is the fact that the field of view is first person. In many or the Atari games tested by DeepMind or in Flappy Bird, the game gives a bird's-eye view, which gives much more information than the agent's first-person view in Minecraft. For example, [**Hiro: do you want to discuss Minecraft/Malmo implementation issues here? Like the lag issue?**]

4 Method

4.1 Scenario

The agent begins in the middle of a completely flat room with a fixed number of randomly generated zombies. The room has a uniform color so that the algorithm will not have to account for differences in lighting and colors. The zombies automatically approach the agent and damage it if they get too close. The agent will be equipped with a sword to fend off the zombies. As the agent destroys zombies, new zombies continue to spawn (so that there is always a constant number of zombies), and the game continues to run until the character loses all of its health. At this time, the score will be calculated based on the total number of zombies that the character was able to kill and the scenario resets to the beginning.

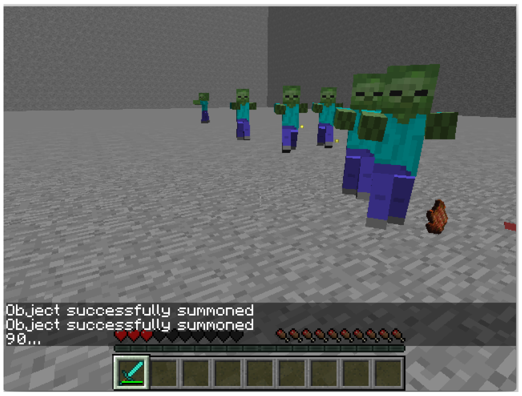
4.2 MDP Formulation

4.2.1 Actions and States

The actions for this problem are the discrete movements of the agent (the agent can move forward, back, left, or right, or remain stationary) as well as the ability to attack with a sword. Minecraft provides the agent with continuous movement actions, but for the purposes of this project we have discretized this into five possible actions.

The states are composed of the image of the player's vision represented by the pixels of the screen. Project Malmo provides the functionality to retrieve the image from the screen and feed this back into our algorithm. In this way, the agent reacts to its environment based on visual input. In order to give the agent temporal information, we store the last four frames of the game and make those the agent's state at any given time.

Figure 1: Minecraft screenshot



Object successfully summoned
Object successfully summoned
90...

4.2.2 Reward

We assign the algorithm a reward of +1 for a successful hit on a zombie and a reward of +5 for a successful kill. There is also a reward of 0.03 for every frame for which it stays alive and a penalty of -1 for every time the agent gets hit. These are designed to train the agent's behavior to incentivize staying alive; holding the kill count equal, we wanted the agent to have a higher reward for staying alive longer.

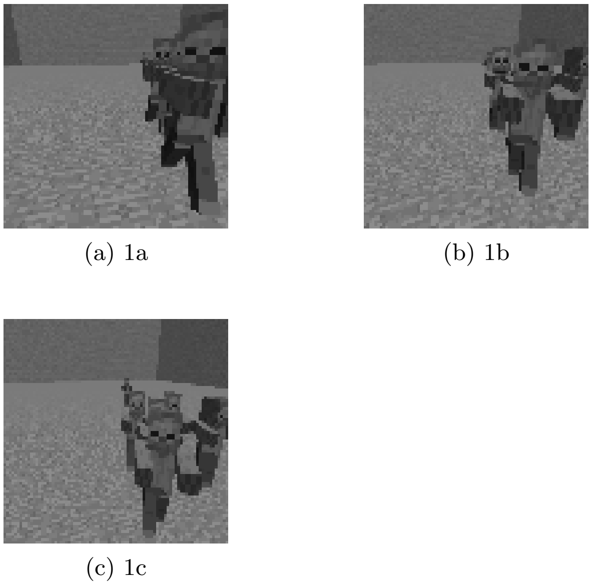
4.2.3 Deep Q-Learning and Theory

We use Q-learning to have our agent learn a policy to best fight these zombies. We utilize the work done by Google Deep Mind to use a convolutional neural network to approximate the Q-function iteration after iteration.

Algorithm 1: Adapted from Matiisen, 2015

```
initialize Q function with random weights ;
observe initial state $S$ ;
repeat
    with probability $\epsilon$ generate random action ;
    otherwise select $a = \arg\max_{a'} Q(s, a')$ ;
    carry out action $a$ ;
    observe reward $r$ and new state $s'$ ;
    update $D$ with new data: $< s, a, r, s' >$ ;
    sample random transitions from $D$ ;
    update network ;
until terminated;
```

Figure 2: Images of processed features



(a) 1a

(b) 1b

(c) 1c

5 Results and Analysis

The agent kicks ass and takes names like Chuck Norris on cocaine.

Table 1: Learning rate under varying Reward Structures

Table 2: Learning rate under varying ϵ

Table 3: Learning rate under varying learning rates

Table 4: Learning rate under varying difficulties

We ran into the unexpected problem that...

6 Conclusions and Future work

We succesfully trained a Deep Q Network to move around a room and kill zombies by taking visual input from the agent. Our results suggest that *foo* and *bar* methods worked best because [...]

Future work on this task might focus on expanding the agent's action space so that the agent can execute continuous movements rather than the discretized actions we have programmed. Additionally, researchers could focus on improving DeepMind's convolutional neural network to further enhance the performance or learning curve of the algorithm. This might involve different gradient algorithms or initial weights.

References

[1] Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Graves, Alex; Antonoglou, Ioannis; Wierstra, Daan; Riedmiller, Martin. *Playing Atari with Deep Reinforcement Learning*, December 2013.

[2] Matiisen, Tambet. *Demystifying Deep Reinforcement Learning*. December 2015. `https://www.nervanasys.com/demystifying-deep-reinforcement-learning/`

[3] Chen, Kevin. *Deep Reinforcment Learning for Flappy Bird*