LET'S BEAT GEOMETRY DASH
APS360 PROJECT PROPOSAL

Joel Vadakken
Student# 10010089798
j.vadakken@mail.utoronto.ca

Skyler Han
Student# 1009794830
hs.han@mail.utoronto.ca

Ian Lu
Student# 1009972139
i.lu@mail.utoronto.ca

Jaden Dai
Student# 1009972228
jaden.dai@mail.utoronto.ca

ABSTRACT

Our team's project proposal is to create a bot that can play Geometry Dash using machine learning. The bot will be trained with two architectures, a convoluted neural network so that it can identify obstacles in the game, and with deep reinforcement learning so that the bot can learn to make the right choices. —-Total Pages: 8

1 INTRODUCTION

Geometry Dash is a popular platformer video game where the player attempts to navigate through obstacles in order to reach the end of a level. The game features a variety of game modes and unique objects.

A video of the first level can be seen here: https://www.youtube.com/watch?v=bbVEbqU9wPo

The player is controlled by one input which determines whether or not the player jumps. Our aim is to build an agent that can beat many of the game's levels. This will involve training two models. We aim to analyze screen information using a CNN architecture to find the positions of structures and hazards, then use a deep learning reinforcement architecture to determine agent behavior.

Our project was inspired by similar work in other games such as Mario. However, most other games simply require you to jump over basic obstacles and do not require complex decision making. Geometry Dash is unique and interesting because many levels feature "fakes" where jumping objects can trick the player into death. See Fig. 1 for an example.

Furthermore, many community-made levels often feature intricate designs that may confuse computer systems, as can be seen in Fig. 2 for example.

We believe that a deep-learning approach can enable a computer to tackle both these issues through object recognition and avoidance of fake objects and routes.

2 BACKGROUND AND RELATED WORK (4 MARKS)

The first trial of applying deep reinforcement learning to train a video game agent originates from a paper written by **?** at DeepMind. They created a program that could play Atari games using deep reinforcement learning. See Fig. 3 This paper demonstrates that a convolutional neural network can overcome the long training process of reinforcement learning to learn successful control policies from raw video data in complex RL environments. The network is trained with a variant of the Q-learning algorithm, with stochastic gradient descent to update the weights. And the game agent surpassed an expert human player in every game they implemented.

Figure 1: A screenshot from the 9th level "Cycles". Normally, interacting with the yellow orbs allows the players to jump higher and avoid obstacles. In this case, however, interacting with any of the yellow orbs will result in the player losing. (**?**)

Figure 2: A screenshot from the custom level "Edge of Destiny." Note the intricate details that make it harder to see the hazards. (**?**)

Figure 3: Screenshots from 5 Atari 2600 games: (Left-to-right) Pong, Breakout, Space Invaders, Seaquest, Beam Rider (**?**)

? created a relatively simple Neural Network to play Geometry Dash that takes in some information (ie. intersection with "orb" objects, distance to blocks, overhead blocks) and outputs an action. See Fig 4. However, we believe that this solution is inadequate as it runs on a simplified version of Geometry Dash instead of the official game. The distance and position of the obstacles are readily available information to the model, which would not be the case in the real version. Furthermore, the simplistic neural network essentially functions as a decision tree, which could fail in some cases to the aforementioned misleading objects.

? created a Youtube video that demonstrates that an AI can come to understand complex in-game physics systems and surpass human-level performance through training. Trackmania shares a similar overarching gameplay system to Geometry Dash in which a player considers their environment in order to control their in-game model. Furthermore, both games feature fairly complex physics

Figure 4: A screenshot of CodeNoodle's visualization of his neural network from his Youtube video. (**?**)

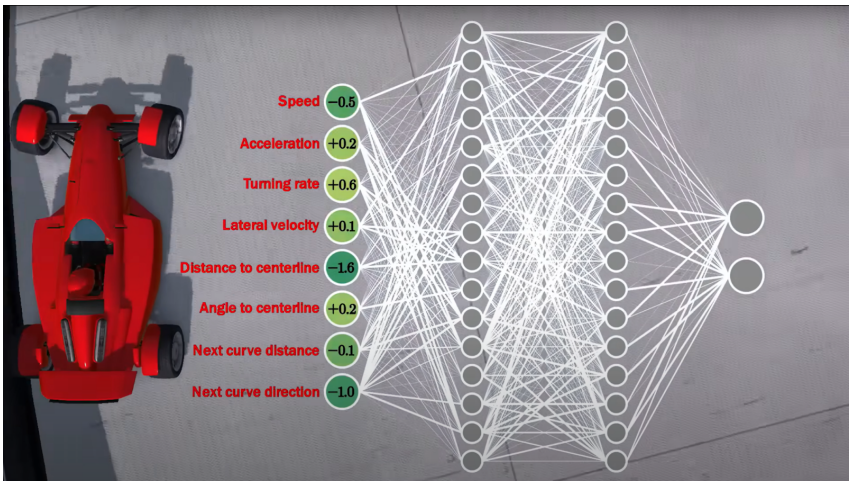

Figure 5: A screenshot of Yosh's visualization of his neural network for his trackmania bot. (**?**)

systems. Since the Trackmania AI was able to master its game's physics system, we believe that a similar approach applied to Geometry Dash may also give strong performance. See Fig 5

? wrote a paper detailing how they split up an image for a robot into a bunch of different segments. This might be relevant for us as we could split up the images we attain from Geometry Dash into an array of squares, and try to sample at a specific framerate.

Figure 6: Video game with optimal path shown (left) vs Deep Q Agent Heat Map (right). (**?**)

? wrote a paper that explicitly uses a combination of CNN and Q-learning network to train an AI that can play a variety of games. They utilize CNNs to perform feature extraction and determine the best

course of action, while the Q- learning component aims to improve decision-making capabilities. They also utilize various pre-processing methods (resizing, etc.). Their work culminated in an agent that could effectively traverse some 2D environments. This paper showcases how a combination of CNN and Q-learning can enable an agent to plan a path through a level, which is fairly similar to what we are attempting to achieve. Their techniques may be a useful reference for our own project. See Fig 6.

? at Deepmind created a model, called SIMA, that can take visual observations from a game to carry out an instruction by outputting keyboard events. Although their model does not achieve the same end goal as us (following instruction vs autonomous gameplay), their agent's ability to understand complex visual information and map it to keyboard inputs supports our approach of utilizing screen capture for object detection and behavior planning. Furthermore, the team's approach to rewarding and punishing agents may serve as a useful reference for our specific game.

3 Data Processing (2 marks)

Our data for our two architectures will both come from the game itself. A CNN model will classify the data, and we will use that to create an environment to train the RL model.

In order to identify the types of terrain objects we propose the use of a CNN to classify different terrain objects into numerical values in a processed tensor that can be passed as an input into the RL model.

To reduce computational complexity and to preserve the accuracy of the CNN classifier, the CNN will include a grayscale filter. Each frame will be grayscale, and the background and each pixel will take the value of the mean colour in its associated area such that each pixel will consist of one colour value.

The CNN will be trained on various snapshot frames throughout Geometry Dash levels and the corresponding labels for each object terrain in each frame can be created using the GD MegaHack v7 mod which identifies the terrain object types on a given screen. The CNN will take as an input a single Geometry Dash frame, and will output a tensor of the same size with the rgb dimension removed. The processed values of the tensor will each be associated with a different terrain object in Geometry Dash. Ex:

- 0 for air (nothing)
- 1 for a platform
- -1 for a spike
- Some set of integers for portals
- Some set of integers for orbs
- Some set of integers for pads
- Other integers for other game elements

The RL model will use the CNN model as it trains in its environment. The tensor size of a singular dataset will consist of the number of frames in that geometry dash level, the number of pixels on a screen per frame, and their rgb values (number of frames, number of pixels to make up a column, number of pixels to make up a row, rgb values). The RL model will use this data to output a tensor of size 1 that dictates whether the bot should jump or not jump.

4 Architecture (2 marks)

4.1 CNN Model

We will utilize a CNN to analyze the game screen and find the positions of hazards, platforms, and other interactable objects, such as can be seen in Fig 7. A CNN is a type of ANN that retains the geometry of the data and performs convolutions through a kernel with variable parameters that goes across the input (usually a 2D image). This will leverage the CNN architecture's ability to

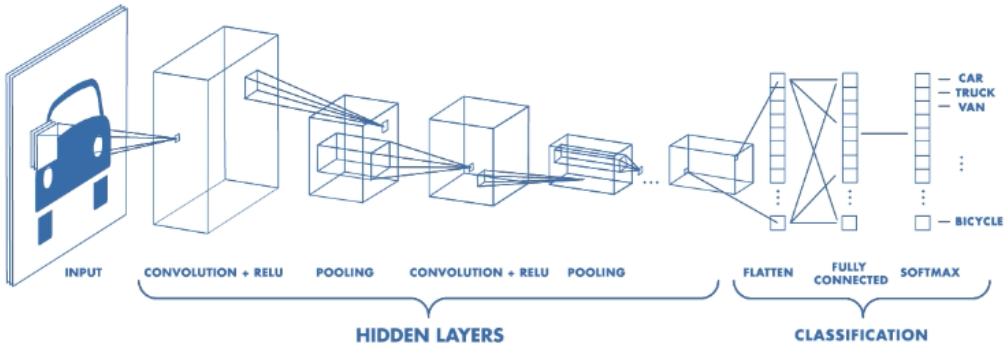

Figure 7: An example of how a CNN neural network might analyze our data.

understand complex shapes and patterns, and will enable accurate positional information that can later be used by the RL agent.

The CNN will be comprised of a set number of layers and will take in as an input one frame from the game screen (640x480x3 Tensor), and output 640x480 array of integer values representing different object types.

4.2 Reinforcement Learning Model

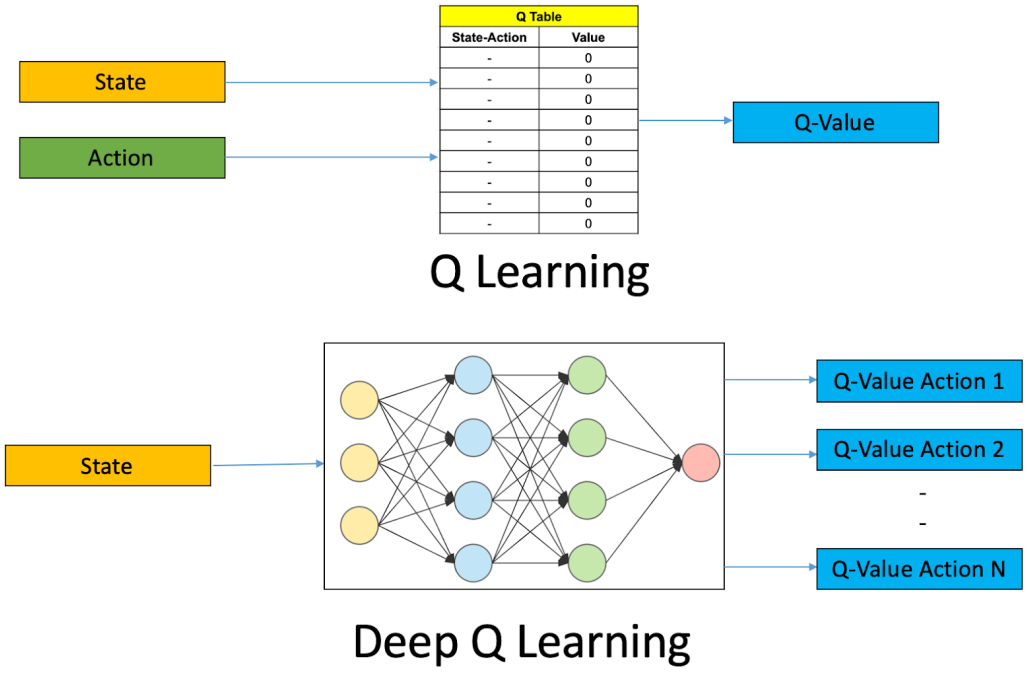

Figure 8: A flowchart that shows how Deep Q learning is used.

We will use a similar reinforcement learning model as was developed by **?**. This will include the following:

1. **Environment**
   - The Geometry Dash game environment needs to be set up so that it can interact with the agent.
   - The environment should provide the current state of the game (e.g., position of the player, obstacles) and allow the agent to perform actions (e.g., jump, no jump).
   - We would need to build the environment with OpenAI Gym to train the agent since we would need to interact with the RL modules.
2. **State Representation**
   - We can get the state from the feedback from the CNN by knowing where the obstacle is in the frame and the position of the player.
3. **Action Space**
   - The action space A consists of discrete actions the agent can take.
   - In Geometry Dash, this could be simplified to two actions: jump or no jump.
4. **Replay Memory**
   - We store the Geometry Dash agent's experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ in a dataset $D = \{e_1, \ldots, e_N\}$.
   - These experiences are pooled over many episodes into a replay memory.
   - We apply Q-learning updates to a small batch of the samples randomly chosen from the dataset.
5. **Reward Function**
   - Define a reward function that provides positive rewards for progress (e.g., distance covered, points scored) and negative rewards for failures (e.g., hitting an obstacle).
6. **Training Procedure**
   - Initialize the replay memory D to a fixed capacity.
   - Initialize the Q-network with random weights.
   - For each episode:
     * Initialize the starting state s_1.
     * For each time-step t:
       · With probability ϵ, select a random action a_t. Otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$.
       · Execute action a_t in the game and observe reward r_t and next state s_{t+1}.
       · Store experience $e_t = (s_t, a_t, r_t, s_{t+1})$ in replay memory D.
       · Sample a random minibatch of experiences (s_j, a_j, r_j, s_{j+1}) from D.
       · Compute the target Q-value:

$$y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{otherwise} \end{cases}$$

       · Perform a gradient descent step on the loss $(y_j - Q(s_j, a_j; \theta))^2$.
       · Update the state s_t to s_{t+1}.
     * Reduce ϵ gradually to reduce exploration over time.

5 Baseline Model (2 marks)

Describe a simple, baseline model that you will compare your neural network against. This can be a simple machine learning model (e.g., SVM, Random Forests, etc.), a hand-coded heuristic model (that does not use machine learning), or something else. Expectations for the baseline model will vary from project to project.

5.1 Baseline Model for CNN

We will compare our CNN model against a hardcoded system that labels the obstacles in the game. Using OpenCV, this system will be able to identify the obstacles in the game and will be able to recognize the player's position. The information will then be fed into the baseline model for RL to make decisions.

5.2 Baseline Model for RL

We will compare our RL model against a simple decison tree that will jump based on the feedback from the baseline CNN model. The bot will jump when OpenCV detects an obstacle in the game and will not jump when there is no obstacle. The system will be able to play the game, but it will not be able to learn from its mistakes or improve its performance over time.

6 Ethical Considerations (2 marks)

Geometry Dash has online levels that users can play to earn rewards such as diamonds, orbs, and stars. These stars can be used to rank the player on the global leaderboard. Developing a bot that can complete Geometry Dash with machine learning can result in players gaining an unfair advantage over their peers as they climb the leaderboard, and result in people playing the game in a way the developers did not intend.

Another ethical consideration could be in the training of the models. It is unethical to use the work of others for profit without their permission. Levels that are created and published by Geometry Dash users online are their own creative works, and it may be unethical to use their work to train a model that can be used to generate an income without their consent. As a result, our group does not intend to commercialize this project.

7 Project Plan (4 marks)

CNN Tasks

Task	**Principal**	**Secondary**	**Internal Deadline**
Data Collection for CNN	Joel	Jaden	June 22
Create and Train CNN Model	Ian	Joel	June 22 and June 30
Create a OpenCV Baseline Model	Jaden	Ian	July 12

Table 1: CNN Tasks

RL Tasks

Task	**Principal**	**Secondary**	**Internal Deadline**
Setup the RL Environment	Skyler	Jaden	June 30
Create a Decision Tree Baseline Model	Jaden, Ian	Joel	July 19
Create and Train Deep Q Network	Skyler	Ian, Jaden	July 1 and July 30

Table 2: RL Tasks

8 Risk Registrar (4 marks)

The largest potential risk of this project is that we are unable to finish. We would like to train a bot that can complete Geometry Dash, but this would require that we finish training both the CNN model and the RL model in time. If we are unable to finish both, our group would just submit the CNN model for our final submission, as the CNN model would fulfill the requirements of this project, while the RL model is beyond the scope of this course. Although an RL model would be nice, we may have to sacrifice it if time does not permit.

There is also a risk that our group will not even finish the CNN model in time. There is a risk that our group will leave deliverables to the last minute, resulting in too little time to properly train our model and embark on the iterative process of trial and failure that marks all successful projects. To combat this tendency, we have decided as a team to implement many internal deadlines so that our model will have plenty of time to train.

Another potential risk of this project could be that our code only works on one machine, as the course instructor made it clear that the TA grading our project should be able to run our code. We intend to make the bot able to play Geometry Dash, which could require that our bot access the keyboard on its host computer, which could involve different libraries and processes for different machines and operating systems. Although we could try to mitigate this as much as possible by testing on various computers and making sure it works on all of our different machines, time constraints may disallow us from solving this issue in time. If we are unable to guarantee that it can run on any machine, we may have to preface our final submission with a warning that it only runs on a certain operating system. As of right now, our team plans to develop our model for Geometry Dash on Linux.

Lastly, there is the risk that one of our team members is unable to finish their portion of the project due to an outside situation. Fortunately, we have clearly outlined the responsibilities and tasks of each teammate, so if a teammate is unable to complete their tasks on time, it will be easy for the other teammates to recognize what still needs to be done.

9 GITHUB LINK (1 MARK)

https://github.com/J-Vadakken/APS360-Project