Daniel Guerrero-Martin, Ellison Eitel, Justin Kerns, Luke Mayer, Rithvik Malladi
Professor Kim
CMSC 421
5/9/2024

Geometry Dash Reinforcement Learning and Neuroevolution Game Agents

The key idea of our project is to create an agent that can complete geometry dash levels after training on them. Creating a Geometry Dash model is useful for many reasons; it would not only allow users to beat levels in the game they would be struggling with, but it could also be used in testing for new levels to be added into the game or for testing user created levels. It could also be used for speedrunners or professional gamers to possibly obtain new strategies to beat levels that had not been used before. Extending and generalizing even further our method of using screenshots of a fast moving environment space and extrapolating information off of those screenshots to make decisions can really apply to anything that requires a computer to make actions given a series of visually cluttered and complex images or a complex video.

We created two solutions, a reinforcement learning model and a neuroevolution model. Our first model, the reinforcement learning model, can train relatively quickly if the hyperparameters are set correctly. Our algorithm we used for reinforcement learning was deep double q-learning which combines deep learning with q-learning to avoid some of the pitfalls of overestimation of actions that can stem from N-step q-learning. However, this model does have its limitations: we have to train the model on each level as it's not a generalized model and we also have to lower performance metrics such as framerate and graphics in order to not overwhelm the model with inputs that are too high dimensional.

Our second model, the neuroevolution model, is very good at getting to a consistent percentage of progress in the level and was much easier to implement than the reinforcement learning model. But tuning the hyperparameters for the neuroevolution model was extremely difficult and training was extremely slow for this specific game as there was no way for us to run multiple games at once.. Ideally the entire population would be run at once which is hard to do with a game like geometry dash so instead we needed to just run through each member of the population 1 by 1 and then after the first set was done it would move onto the next set. This is

obviously time consuming but nevertheless we were able to generate decent results with this method and with enough time it should be able to beat the level.

Our two solutions to the problem of beating the geometry dash levels, were built with certain similar components but ultimately were developed differently and work in very different ways. For reinforcement learning, we developed a gymnasium environment called GeometryDashEnvironment which handles the observation space, action space, step cost, and reward. In order to help with the processes getting snapshots of the game we also created FrameHelper which takes each frame of the game and uses the screen shot to obtain the information in the environment helper which determines: the progress made on the level, whether or not it's terminated, and the frames captured by the frame helper in order to determine the observation space. FrameHelper also processes the image by gray scaling, downscaling, and resizing the image.

The reinforcement learning model uses a combination of double q-learning to train the model and a keras neural network as the framework for the model. The neural network almost acts as a q-table that gets adjusted during training, predicting the next action with a combination of random exploration and neural network prediction. The neural network we made uses 2D convolution layers with padding as well as some other linear layers.

The neuroevolution model uses the neat python library to initialize neural network agents that play on images of the game as well as doing other things such evolving the population, evaluating the progress made using helper methods, and the ability to save, load, as well as show the model.

We used multiple existing libraries Keras, NEAT, numpy, gymnasium, random, pygetwindow, cv2, collections, mss, time, os, visualize, and pickle but we did not use any pretrained models. We chose keras as our neural network because of its variety of training api's as well as the amount of versatility in what systems we can import from keras. These factors made it our choice for the neural network implementation. We implement the backend of our Keras desktop implementation with pytorch. We chose to use gymnasium in order to create a gym api that would make implementing the observation and action space as well as the step and reward calculation easier for the reinforcement learning aspect of our model. We used the NEAT library in order to implement a neuroevolution network as it streamlines the controller for the evolution as well as allows for easy integration with the FrameHelper class. The FrameHelper

class we use getwindow and mss in order to get the window from Geometry Dash which we then use to fully implement the observation space using cv2 in order to process the frames as images. We also used numpy for array manipulation, random for the random choices necessary for exploration, and pynput for implementing the pressing of the keyboard in order to do the action in the step. We use time in order to perform a delay between training sets, collections in order to import the deque method, and lastly pickle is used in the NEAT model in order to save and load the model while visualize is used in that implementation in order to visualize said model. We chose these libraries because of our familiarity with them as well as how they all were able to be used together in conjunction to properly implement our models.

Our initial goal was to create a reinforcement learning agent that would be trained on and then be able to complete geometry dash levels, unfortunately we weren't able to meet this benchmark. Despite creating both NEAT and Reinforcement learning based models, there were many unforeseen difficulties that prohibited us from accomplishing our initial goals. Since we decided to use the actual game to train we weren't able to actually access the underlying game data to get more precise and efficient input data, instead we had to rely on screenshots of the frame data as our inputs. While the input was still reliable we had a balancing act of processing our images to reduce training time or not processing to retain as much information as possible. This is an issue because of how frame perfect jumps over obstacles need to be in geometry dash and while the window to actually perform a successful jump was large, if that window was even missed by 1 frame the player would die. Another issue we ran into for both models was how much jumping was favored. For the reinforcement learning when we had it as baseline N-Step q-learning jumping was an overestimated action and the model would just choose to repeatedly keep jumping which is why we had to implement the double q-learning with a deep keras network. On the NEAT side because the algorithm starts out with a simple model and evolves the model with more generations the issue of jumping would eventually solve itself. But because NEAT took so long to train for Geometry Dash we mostly just saw it constantly jumping with steady but slow progress. While the game has only two actions, jumping or not jumping, the levels themselves are large and complex which as a result leads to a large observation space which in turn inflates training times. Lastly, within individual levels there are different modes the player must go through with a side scrolling section leading into a spaceship dodging section leading into even more sections in later levels. We were able to accommodate these new

gameplay sections but weren't able to find enough time to train our models for them. These issues were further compounded by the length of time it takes to actually train the models all in all resulting in a much more difficult task than we initially expected.

Within level 1 there is a regular geometry dash gameplay section that leads into a new mode at 34% where the player goes into a spaceship, changing the inputs. After about 1500 epochs our reinforcement model averaged about 20% progress for a given run peaking at the 34% threshold. As for our NEAT model, we only got to around 16% progress after about 8 hours of training but this is most likely due to how long it took to train. While it may seem as though the reinforcement agent may have performed better than the NEAT implementation, we feel as though it doesn't paint the full picture for three reasons. Firstly we were able to train the reinforcement learning model on a desktop. Secondly the way which we obtained data through screenshots was likely not as efficient for the NEAT implementation as larger observations/screenshots seemed to cause issues for the model computationally. Thirdly, the amount of training we could do on the NEAT model was limited as we could not run the generations in parallel.

Each group member contributed significantly to the project and its many different aspects. Luke worked a lot on the reinforcement learning neural network implementation, the frame helper, environment helper, as well as the neural net and GD environment drivers, writing a significant amount of code for each of those classes as well as contributing to a lot of the ideas for the project itself. Ellison worked a lot on the reinforcement learning neural network and q-learning implementations writing a lot of the code for q learning as well as the drivers for neural networks and contributing heavily to the idea for the project. Daniel worked alot on the reinforcement learning neural network implementation for the reinforcement learning setting up a lot of layers of the keras implementation and some coding/idea work on the drivers as well as writing the report and poster slides. Justin wrote a large part of the code for the NEAT model implementation as well as writing functions for getting the game state. Rithvik also contributed heavily to the implementation of the NEAT learning model as well as helping to write the frame helper and screen_capture files. Every person in our group contributed significantly to the final project and we believe that effort reflects in the final result we were able to achieve.

Literature Review

Our initial idea was to use concepts we learned in class like Q-learning to handle the task of playing Geometry Dash but we quickly realized that just Q-learning wouldn't be enough and so we decided to shift approaches to try out deep q-learning, double q-learning, and NEAT. Chandrakant [1] gave us an introduction to using reinforcement learning for training a neural network. Chapman [2] expanded on that knowledge of neural networks and more specific ways we could use Keras. Li & Rafferty [3] showed us some of the pitfalls to be wary of when attempting this problem, specifically the dangers of over punishing unnecessary actions which could lead the model to be too cautious. Nuer [4] introduced us to the idea of double q-learning, using a second target network that updates weights from the online model at less frequent intervals in order to stabilize training and prevent convergence on suboptimal policies. Phil [5] helped us overcome issues we had during the code implementation of these ideas, especially by walking through more optimal ways to implement memory buffers. SethBling [6] was our initial jumping off point for using NEAT to train a neural network to play geometry dash as in their video they discuss how they use NEAT to play Super Mario World. Stanley et al. [7] is the actual paper that introduced the NEAT algorithm and we used it to help with our own implementation of it in python using the python neat library.

Citations

[1]. Chandrakant, W. by: K. (2024, March 18). *Reinforcement learning with neural network.* Baeldung on Computer Science. https://www.baeldung.com/cs/reinforcement-learning-neural-network

[2]. Chapman, J., & Lechner, M. (2024, March 16). *Keras Documentation: Deep Q-learning for atari breakout*. Keras. https://keras.io/examples/rl/deep_q_network_breakout/

[3]. Li, T., & Rafferty, S. (2017). *Playing geometry dash with Convolutional Neural Networks.* Stanford.CS. https://cs231n.stanford.edu/reports/2017/posters/605.pdf

[4]. Nuer, J. (2022, May 20). *A comprehensive guide to deep Q-learning*. Medium.

https://medium.com/@jereminuerofficial/a-comprehensive-guide-to-deep-q-learning-8aeed632f52f

[5]. Machine Learning with Phil. (2019, August 5). *Double Deep Q Learning Is Simple with Keras* [Video]. YouTube. https://www.youtube.com/watch?v=UCgsv6tMReY

[6]. SethBling. (2015, June 13). *Mari/O - Machine Learning for Video games* [Video]. YouTube. https://youtu.be/qv6UVOQ0F44?si=mNP6A9265S-ZFnKb

[7]. Stanley, K., Miikkulainen, R., & The MIT Press. (n.d.). (2002). Evolving Neural Networks through Augmenting Topologies. In *The MIT Press Journals* [Journal-article]. 10(2):99-127, 2002.