World’s Hardest Game AI

Project Overview

The purpose of this project is to develop a machine learning model to play and complete World’s Hardest Game. The program captures a screenshot of the game, reduces the image’s dimensions, and determines the best path to take.

Game Overview

The World’s Hardest Game is a JavaScript game created by Stephen Critoph. It can be found at <http://www.coolmath-games.com/0-worlds-hardest-game/>.

You start the game as a red square placed in a bounded maze. Using the arrow keys, you navigate to the end of a level without touching any blue circles. Touching a blue circle returns you to the start of the current level and increments your death counter. Blue circles can be stationary or move throughout the level in cyclical patterns.

Green areas mark safe spaces where no blue circles can touch you and start and end zones. Some levels have yellow circles that must be collected before you can progress when reaching the ending zone.

There are no lives in the game, so you want to minimize your deaths as you progress through the game. After completing 30 levels, your final score is the displayed as the amount of deaths you received, with a perfect “run” being a perfect score.

Old Approach

The program will open The World’s Hardest Game in a web browser. It will start with the bot or manually and begin collecting samples. The program will download images of the game every 200ms and save it into a folder. The program will then down sample the image using bilinear transformation to 128 by 128 pixels wide, filling in blank spaces with black. Bilinear was chosen since it provides better discrete edges to detect walls. Other methods might be used if it’s found helpful. The down sampled images are then saved into another folder.

CNN reading a screenshot of the game. The image is scaled down to 128 width using bilinear transformation. The CNN should find the player location, where the blue circles are, where the coins are, any green zones, and the start and end locations. The output should be a 1D vector containing all the information as the top left pixel location of each. The output of the CNN will connect to a fully connected FF NN which will forward prorogate the pixel locations to find the arrow key to press.

200 images will be collected with 160 used for training and 40 used for testing. The sample size as to be small since the labels will be set manually. I can’t think of a better way to set the labels automatically since it would require knowing where to move already. More images might be needed to correctly train the network. I will see how well it performs before collecting more samples.

New Approach

Lmao what were we thinking before.

The problem of finding the exit while avoiding walls/enemies can be reduced to a path finding problem. The walls and enemies can be thought of as walls in a maze and the exit is the end of the maze. A\* can be used to find the exit of the maze; however, the dynamic nature of the enemies prevents the algorithm from working effectively. D\* is like A\* except with the addition of finding a new a new path once the only path becomes obstructed. Both of these algorithms cannot information about the cyclical nature of the enemies to make an informed decision on the agent’s movement. I hope to use reinforcement learning to learn and predict the movements of the enemies to better navigate the levels.

Although the problem can be reduced to a path finding problem, important preprocessing must be first done. A screen capture will be taken of the game. The image will then be transformed into a matrix containing the rewards for the agent moving to the location. The Walls and enemies will have negative rewards and the goal a positive reward. A negative reward will be given for each movement and staying still.

Once the reward matrix is created, the agent will move using a chosen policy. Q learning is the choice for now, but another policy may be chosen later. The choses of up, down, left, right, and stay will be given to the agent. The agent will pick a movement and move in the game.

Findings

* Making a grid based on the locations of the walls and the agent can solve the first two levels.
* The agent gets stuck in the fourth level likely due from overfitting from training on the first level for too long
* Performance improvements can be made by 1) tuning the CNN to better handle the input matrix, 2) increasing the resolution of the grid to represent more locations the agent is in, 3) changing the grid to show the rewards for moving in a direction like in the grid world example
* Other improvements are 1) adding diagonal movement, 2) changing the levels based on time to prevent overfitting, 3) finding a smoother method for moving the agent, 4) getting a better computer to increase training rate