Procedural Platform Generation Management Using Machine Learning

Michael Probst
University of Kentucky, Lexington, KY USA
mppr222@uky.edu
GitHub project repository

Introduction

A popular subset of platforming-type games are infinite runners in which the player traverses a world that is endless and can only end when the player enters a fail state. In order to support and infinite world, the environment needs to be generated during runtime and must create objects some distance ahead of the player. Generally, these types of games randomly generate premade sections of platforms with some preference to sections that were rated as more or less difficult by the game designer. The purpose of this project was to attempt a different approach of platform generation in which a reinforcement learning agent learns the strengths and weaknesses of individual players and can cater a more interesting experience by creating a challenge that is specific to the skill of every player.

1. Resources

The purpose of this project was to experiment with machine learning techniques in an infinite runner game. Due to the limited time frame, an open source project was used so that there was no focus on the development of the core mechanics of the game.

2. Block Generation

The source had easily modifiable block generation code that allowed individual platforms to be instantiated in the world. These modifications included generating random widths and heights between blocks. Depending on a difficulty factor, the range of width and height differed making challenging jumps when the difficulty is high and easy jumps when the difficulty is low. The platforms themselves were pre-made sections of blocks that were chosen one at a time using a weighted probability.

2.1 Bock Features

5 features were determined to be the most difficult features of blocks that can occur:

1. Far Jump: Distance to next block is greater than half of the maximum block distance.
2. High Jump: Height to next block is greater than half of the maximum block height.
3. Below: Block is below the previous.
4. Narrow: Block is 1 or 2 tiles wide.
5. Has Enemy: There is an enemy on the block.

When a block is spawned, each feature that could apply to a block is considered. Each feature has its own probability associated with it that is determined using the number of

times the player has succeeded or failed on that feature.

3. Player Performance

3.1 Difficulty Management

When a player is doing well, it makes sense to present them with a more difficult challenge. Therefore, the performance of the player needed to be tracked. Criteria that determine the player's success are the average speed of the player, the time spent on each block, and the distance the player has traveled.

3.2 Block Probabilities

When a player successfully completes a jump or dies, the probabilities of each type of block is updated. The features on the block that is associated with a success or failure are added to the total success or failure counts for each feature that was present on that block. These success and failure counts are used to determine the probability of those features appearing on subsequent blocks. These probabilities give insight on the type of blocks that the player is good and bad at which are utilized so that the player is presented with blocks in which they are skilled when the difficulty is low, and they are presented with blocks they are not skilled in when the difficulty is high.

4. Testing Block Generation

Over time, it is expected that once the block management has learned what type of blocks the player is good and bad at, the better the player will perform because they are being presented with challenges that they are capable of completing in the beginning of the game. Also, it is expected that the frequency of the feature in which the player is not skilled will appear will decrease. To test these hypotheses, an agent was devised to play the game.

4.1 Agent Details

An agent was created that somewhat resembled human gameplay. It runs as long as it is grounded to a platform, and it jumps if it is about to run off. The agent is also capable of slowing itself down if it is about to overshoot a jump. This agent was vital to initial testing of the application of the block probabilities and the general use of the machine learning block generator, however, running the agent in real time was slow and a quicker testing environment was necessary for determining the effectiveness of the block generation. This was achieved by replicating the environment in a terminal-based setting.

4.2 Terminal-Based Testing

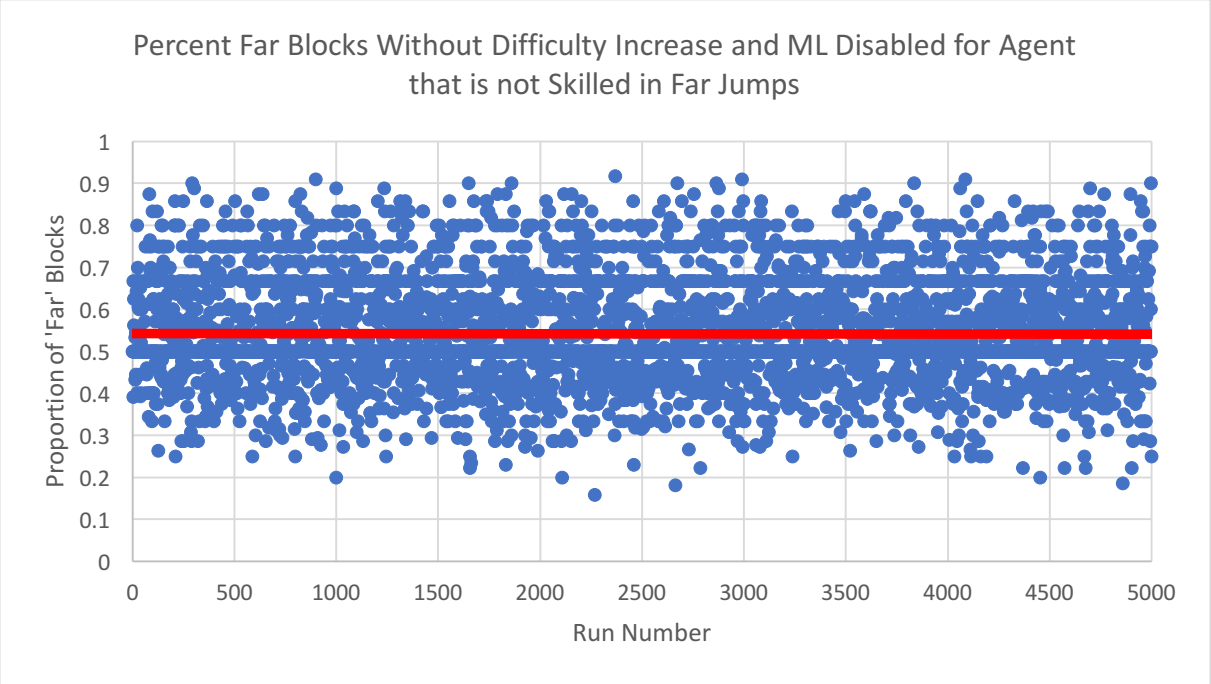
Using the same algorithms that determined the probabilities of the blocks and the selection of the sequence of blocks, a terminal-based environment was created. In this environment, the agent is presented with a block with features that are determined by their previous performance on blocks with similar features and are more or less aligned with the specific skill of the agent depending on the difficulty factor. The agent is given 5 skills, each correlating to the features that can appear on a given block: far, high, below, narrow, and enemy. These skills are assigned a value 0 to 1 that act as the probability of the agent succeeding on a jump given that the block has the feature correlating to that skill. Since not all blocks will always have all 5 of the features, skills relevant only to features that are present on the block will be taken into account, summed and averaged. For example, a block has the features far and enemy. The agent has a probability of success of 0 for far jumps and a probability of success

of 1 for blocks with enemies. The total probability of the agent completing this block is 0.5.

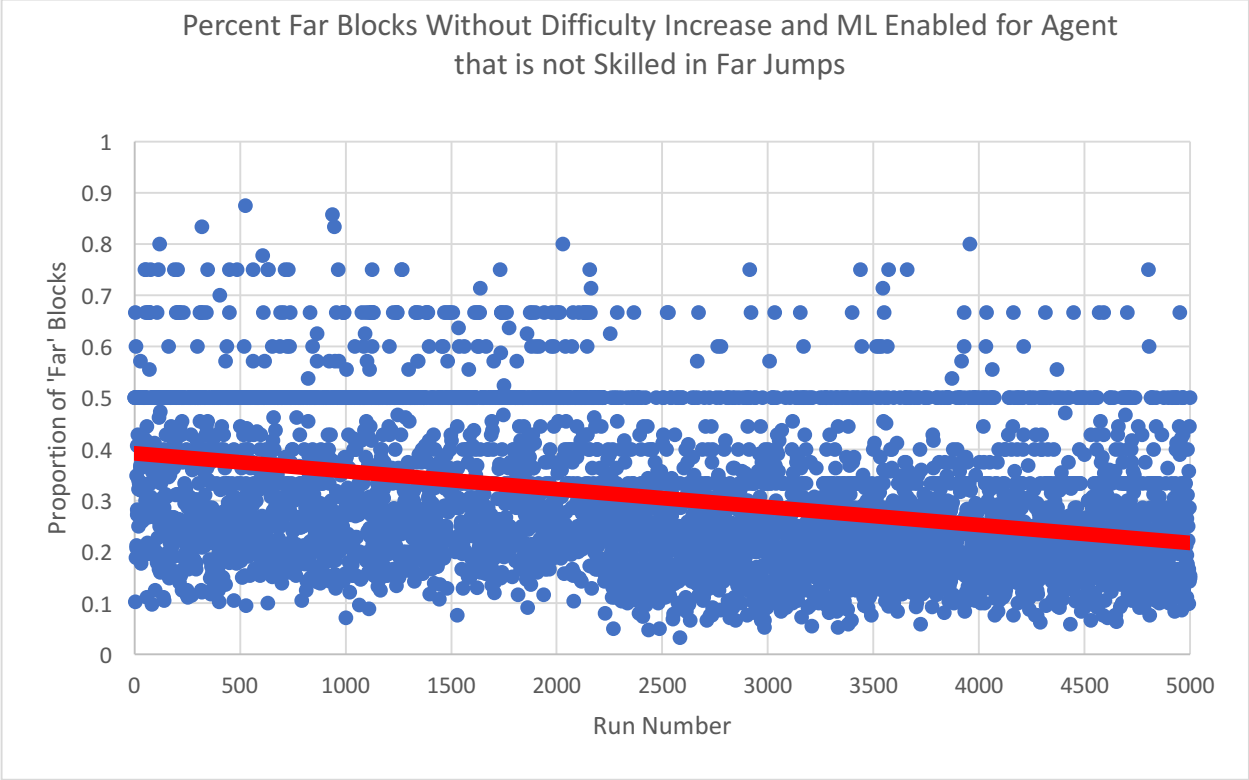
In a specific test, an agent was given the following probabilities of success for each feature: P(far) = 0, P(high) = 1, P(below) = 1, P(narrow) = 1, and P(enemy) = 1. The agent simulated running through the environment one block until it fails or 100 blocks have been completed. It is expected that as the agent completes a run, the block manager will generate blocks that cater to the specific skills of the agent. In this case, it is expected that over time there are significantly less blocks labeled as a far jump than any other label. This experiment was conducted without any difficulty adjustment to increase the likelihood of the observing blocks that the agent is not skilled in. A second experiment to act as a comparison is set up the same as the first, but without updating the probabilities of the features of the blocks, meaning the features observed on blocks is always random. This will act as a measure to truly see the results of the effects of the machine learning block manager. Each scenario ran the agent through the simulation 5000 times.

5. Results

The following graphs show the results of the runs made by the agent with the machine learning block manager, and without.



The graph above are the results of the controlled test where the agent ran the simulation 5000 times without the block manager updating the probabilities of each feature appearing on the blocks. The graph represents the proportion of the blocks in each run that were classified as a ‘far’ jump. This data shows that over time, the proportion of blocks that were classified as far did not change and on average the proportion of blocks classified as ‘far’ was 0.54. This is reasonable because there were no modifications to the proportions at which features should appear. On average, agents in this test completed 9.14 successive blocks.



The graph above shows the results of the agent completing 5000 runs with the machine learning block manager active and updating the probability of assigning a feature to a block dependent upon the number of agent successes and failures on each block feature. It is clear that over time, the proportion of 'far' blocks decreased at a steady rate. In total, the proportion of far blocks decreased by 0.2. On average, the proportion of 'far' blocks for a run was 0.30, which is 0.24 less than when the machine learning block manager is not applied. So, it is clear that the block manager was working as intended in terms of accommodating the skillset of the agent. Also, the average number of blocks the agent successfully completed was 20.97, which is about twice as much as without the machine learning disabled.

These results show that not only is the machine learning block manager spawning less of the blocks that the agent is not skilled in, but as a result, the blocks spawned result in a dramatic improvement in agent performance.

6. Future Work

It should be noted that the increase in performance is great after 5000 plays of the game. For an agent, this is trivial, but not many humans are willing to play the same game 5000 or more times. The unfortunate drawback of machine learning is that a lot of data is required to make a difference. A potential solution to this problem is to gather data from a human's run, then simulate additional runs with an agent that is similar in skill to the human so that more data can be collected to contribute to the effectiveness of the block manager. Also, the game is currently in a state that all blocks are fairly simple which is not very

interesting. Additional content such as new enemies, block terrain, and other obstacles would be more fun and engaging for players.