The Evolution of Mario - A Genetic Algorithm PCG for Mario levels

Neal Kaviratna and Rich Li

When working on this project we opted to implement a Genetic Algorithm to help make our procedurally generated levels. The reason we wanted to procedurally generate levels was to cater them to individual players (hunter, collector, etc). So if they like to collect coins, the ideal is that our we would procedurally generate a level that will have more coins, therefore become more fun for this player. Using procedural generation has many benefits, including but not limited to, automatic design to save manpower, ability to create large worlds for relatively little resources, and ability to allow the same game to provide different, unrepetitive player experiences. Specific data points to look for would include: the frequency of gaps and stairs in a level for players that like to jump, the frequency of coins and blocks in a level for players that like to collect, and the frequency of enemies in a level for those who enjoy stomping on their foes!

Our genetic algorithm functions by creating an initial pool of 2000 randomly generated levels for the algorithm to start with. From these 2000 ‘living’ levels, we evaluate each level with a Fitness function, which will be discussed in depth later on. After determining the relative fitness of every level, we take the 200 fittest levels and mark them as progenitors. These 200 levels are randomly bred together to generate 1800 children levels, which have a small chance of mutating upon creation. This process is repeated again and again with the new pool of 2000 levels, and so on for 100

generations. After this, the level with the highest fitness is selected as the ‘apex level’ to be returned from the algorithm. With an admissible fitness function, this algorithm should create a fun level for the provided player profile.

Along with the genetic algorithm needs a function to determine the “goodness” of a certain level for the player. This function is called the fitness function. The source code provided for us already had a way of collecting data from players, and we were left with taking using that data to judge a certain level. Unfortunately, the source code had a in which it could not handle deaths, so we could not use that to evaluate our level. Our fitness function uses what the player did, like number of jumps, and percentage of monsters killed, and multiplies that with the number of opportunities in our levels to do that action with a certain multiplier. The specific multipliers were found by balancing certain premade player archetypes so our metrics don’t favor any specific archetype over another. The aggregation of these things resulted in a double that we could compare with other levels to determine their rank on how good that level is. It is important to note that this fitness function is made specifically for this situation and is not generic for any situation.

We wanted to implement our level representations so they were easily mutable/extensible, allowing us to focus on the Genetic Algorithm and the Fitness function used with it. Our levels consist of sections such as ‘straight’, ‘pipes’, ‘coins’ that we designed ourselves. The levels are initially made with a random combination of these sections, which we represent as an intmap of sections as well as the bytemap of blocks necessary to render the level inside the mario engine. When we need to mutate

a level, we can just change a section from one type to another, and regenerate the byte array. To create children, we can take half the sections from one level, and half from another, then generate the corresponding byte array. This abstraction of the levels innards helped us implement our Genetic Algorithm easily.

Several sections we made were created to showcase our algorithm's ability to create levels to match the given Player profile. These sections include a section with a large block of coins, a section where the stairs over a gap are guaranteed to be generated, though at a random height, and a section with a horde of enemies. These sections make it easily apparent what the Genetic algorithm thinks that the player prefers.

Overall we feel we are happy with our Genetic Algorithm, it creates fun mario levels that suit the player playing them. We did encounter some fun non-solutions along the way as well. Recommendations include levels made only of sections with Bullet Bill cannons and levels with sections built inside of one another. Future work on this project would strive to create more varied sections, as well as refine the balance of the fitness function further.

Code for both the Genetic Algorithm as well as the Fitness Function can be found in MyLevelGenerator.java