CMPM 146 - Assignment 5: Evolving Mario Levels

Mika Peer Shalem, Anthony Umemoto

What you changed from the template and why, especially related to your selection strategies, fitness functions, crossover and mutation operators, etc.

For individual grid:
We decided to implement prefabs with built-in smaller sections. We then changed *random_individual* to add a bunch of random sections in the height/width of the map, while still ensuring that Mario and the flag are still on the map. We made a preferred array to check which sections go along next to each other.

In *generate_children*, we return two children, half of their genome comes from self and the rest from other. We checked if the neighbors of the current section worked, and if not, we switched it to the other genome. This way the crossover happens when needed.

Mutation - we add a mutation in 10% of our sections, ensuring more variability in the code. Moreover, we make sure that pipes would not start at the top of the screen in the mutation function.

Generate successor - we first sort all the individuals in the population based on their fitness, then using random selection we add individuals to a selected list. Our second selection strategy is to take the child with the lowest fitness and highest fitness to generate children (then we do the second-highest fitness with the second-lowest, etc.). We want to include both the best and worst fitness to ensure we are not getting rid of helpful mutations.

For individual DE:
How does it work?
Rather than representing the map as a matrix of blocks, it represents the map as a list of structures. Each structure has a type (e.g. “pipe”, “stairs”, “platform”, etc) to describe the actual shape, or sequence of blocks, it will create in the final grid. Each of these structures are stored as a tuple of needed values. For example, the pipe structure contains fields for its x position, and height.

During crossover, it will create random slices of each parents’ genome, then concatenate them together in 2 orderings: parent1 then parent2, and parent2 then parent1. This creates two distinct children that will then be mutated.

When mutating an individual, it picks one of these structures at random to change some of its values. This is done most commonly by pushing the value in either a positive or negative direction, and then clamping the value to keep it from tending towards an undesirable extreme.

We changed the way we calculated fitness to panelize having jumps that are longer than 3 blocks (it made the level really difficult), less than 3 stairs (made the level boring), and not enough platforms. We also changed the maximum height of the pipe in *random_individual* to make the game beatable. In mutate, we added changes in the x-value of the enemy and increased the percentage of mutations to add variability to the levels.

Something about each of your two favorite levels: Why do you like them? How many generations did it take and how many seconds to generate these levels

We liked the levels because they weren't too difficult but still challenging enough to not be able to beat them on the first try. They had many structures in different variations so you had to change your strategy throughout the level to win.
Both of our favorite levels took 6-7 generations and about 120 seconds to generate these levels.