Modifications to ga.py

We do not intend to compete in the competition

DE mutate

Firstly the mutate only has a 10% chance to actually mutate anything at all. If mutations are to happen, the code chooses a random area in the genome and checks what structure type is there. Once it knows what structure it is modifying, it does modifications that are tailored to the structure type. After doing said modification it then returns the modified genome.

Modifications that I made to DE mutate include increasing the possible range for the y offset in 4_block, 5_qblock, and 7_pipe. This was done in order to increase the height variance for each of those blocks, hopefully leading to levels with more verticality.

DE generate_children

This function utilizes variable-point crossover. First you find random values that split the length of each genome into two pieces. Then you take the front split of the first genome and attach it to the back split of the second genome. Then take the front split of the second genome and attach it to the back split of the first genome. Then it returns both new genomes while calling mutate on both.

DE generate_fitness

generate fitness modifies the linearity to keep un traversable paths from forming also it limits the number of holes as to stop the frequency of large holes from appearing

Generate_sucessor

Before any generation begins I sort the population by their fitness. Then, I generate two children 50 times using an elitist selection that randomly chooses two genomes from the top 5 fitnesses in the population. Then I generate two children 50 times using a random selector which chooses two genomes completely at random. Then I add all of the generated children to the list and return it.

Grid mutate

Each block in the genome has a 10% chance to have the ability to mutate. If it actually can't mutate, the type of block that is changed is completely dependent on the blocks around it. For instance, in order to prevent the formation of massive walls that Mario can't jump over, a mutated block can only form if: the block to the right of it is a block, the block to the right of it is a block, the block diagonally below it to the right is a block, or the block diagonally below it to the left is a block. This, while not perfect, has significantly prevented the amount of unscalable walls that can generate. Leading to easier levels. Another thing we did was when we generated question blocks, we checked to make sure that there were 3 blocks of empty space between the

question block and the ground. And while blocks could eventually spawn underneath them to block off the boxes. It still made it much easier to access the boxes than pure randomness. Finally, for the enemies, I made it so that they would only generate if the space beneath them was a box. While this sometimes has led to enemies spawning in enclosed areas, the majority of the enemies spawned have been interactable with the player.

Grid generate_children

This generate_children utilizes single point crossover to create said children from their parents. First it makes deepcopies of both parents. Then it randomly sorts the two parents so that one randomly goes before the other, the front_parent and back_parent. Then it finds a random value between 1 and the length of the levels - 1. Once it has that, it splits both front_parent and back_parent in half along that random value and swaps the front halves out with each other. Finally, it returns both modified children while calling mutate on both.

Something about our favorite level
Personally I quite liked the variety in types of obstacles in the level. At first, it is empty and flat, allowing the player to run forwards as fast as possible. Over time, more and more things begin to get in the way, leading to players having to become more and more deft with their controls to maintain their speed. Culminating in two valleys of blocks that the player must navigate through precisely to maintain as much speed as possible to the finish. This level was generated using Individual_DE and was generated in generation 54, each generation took on average 3.643 seconds to compute.