Search-based procedural agent generation for a 1D “platformer”

Part I:

1. Introduction

For this task I have chosen to implement a simple evolutionary agent creation algorithm, which are tasked with reaching the finish line in a 1D “platformer” level, where there are certain obstacles like bushes and birds, for which the player has to move accordingly. The goal of the algorithm is to evolve a population of agents represented by sequences of movement commands that navigate the level from start to finish as effectively as possible.

2. Explanation of the algorithm and implementation

The algorithm begins with a randomly generated population of agents. Each agent is evaluated by simulating its moves in the level, and fitness is based on the distance reached. The next generation is created using genetic operators:

• **Crossover**: Combines two parent agents at a random point to produce a child with mixed moves from both parents.
• **Mutation**: Introduces random changes in the child’s moves to maintain diversity.
• **Elitism**: Copies the top 10% best agents unchanged to preserve high-quality solutions.

Key functions include:

• `**worldGenerate**()`: Generates the level, either manually or randomly, with obstacles represented as integers.
• `**evaluation**()`: Simulates agent movement and returns fitness as the furthest position reached.
• `**crossover**()` and `mutate()`: Implement genetic operations to create new agents.
• `**evolve**()`: Runs the evolutionary loop for several generations, applying selection, crossover, mutation, and elitism, while tracking performance statistics.
• `**plotEvolutionGraph**()`: Displays a graph of average fitness across generations to visualize evolutionary progress.

3. PCG taxonomy of the algorithm

The algorithm falls under Search-based Procedural Content Generation (SBPCG), where the content being generated is agent behavior rather than the level itself. It uses an indirect representation (move sequences) evolved via a genetic algorithm. The search space consists of possible agent genomes, and evolutionary operators such as crossover, mutation, and elitism guide the search toward better solutions. The fitness function is based on agent performance within a fixed, procedurally generated level.

4. Visualization of the algorithm

The algorithm is visualized in the windows created after the evolution is finished.

Firstly,, the graph window, which visualizes the evolution process by plotting the average, best, and worst fitness across generations, providing a clear view of progress and diversity within the population. This comprehensive visualization helps in analyzing convergence and exploring the range of agent behaviors over time. Additional visualizations, like agent paths or level layouts, could further enhance understanding but are not currently included.

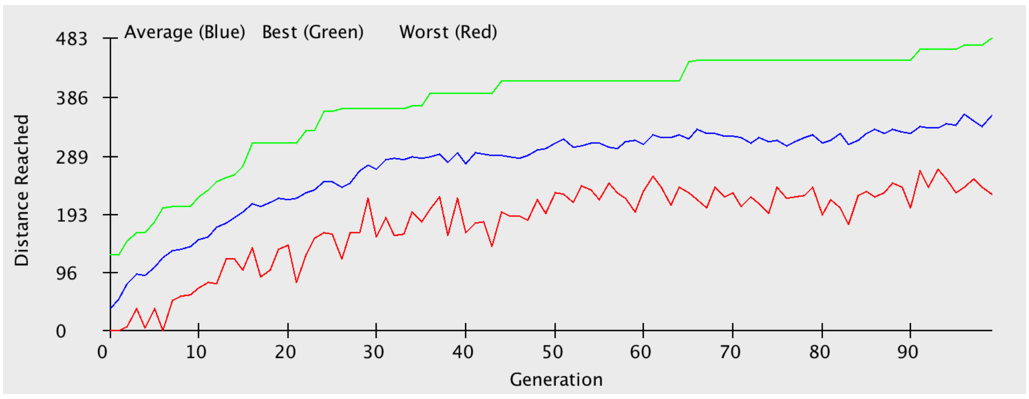

Figure 1: Evolution graph

And secondly, the algorithm also generates a color image representing the change of the best genome in every generation, green - meaning an unchanged gene and red - meaning a different gene (in this case a different move than before).

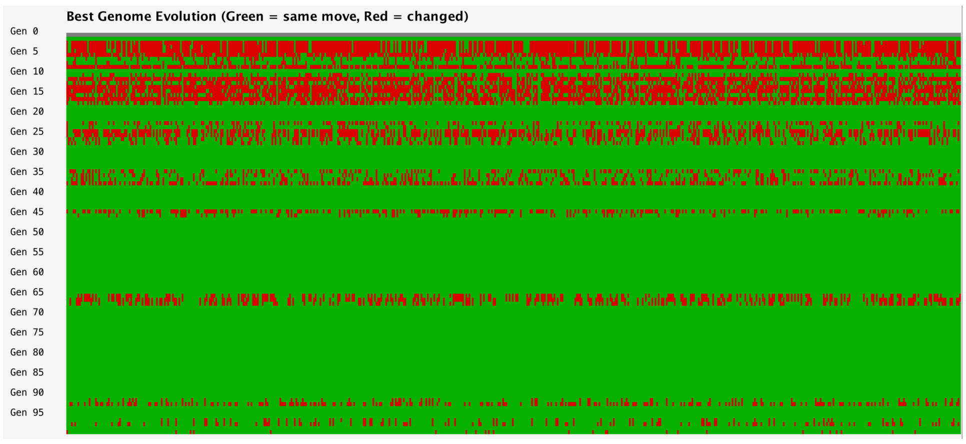

Figure 2: Genome evolution image

5. Expressive range of the algorithm

The algorithm's expressive range is currently limited by the 1D platformer setup and simple agent move representation. It can generate varied agent behaviors that navigate different obstacle configurations within this restricted environment. However, its capacity to produce diverse or complex content is constrained. Expanding to higher-dimensional levels or richer agent representations would significantly increase the range of possible behaviors and solution.

Part II:

1. Critical reflection over the algorithm, content and evaluation

The evolutionary algorithm successfully evolves agents to navigate a simple 1D level, demonstrating key SBPCG concepts. However, the 1D environment and basic fitness measure limit challenge complexity and evaluation depth. Visualization focuses on average fitness, missing insights from best and worst agents. Overall, it provides a solid proof of concept, with room for more complex environments, improved metrics, and richer analysis.

2. What would I do different, what worked, what didn't, what would I do next

In it's current state, the fitness function could be improved to better reward reaching the finish line and penalize inefficient moves. One noticeable behavior in the evolution process is that the distance curve always gets less steep after a few generations, indicating diminishing progress in terms of improvements. This means that initial generations make significant gains toward the goal, but later generations show slower improvement and reduced convergence speed.

3. The use of AI in the task

As in the previous task AI was used in this task for proof reading the report and to help generate some minor helper with discrete helper functions when manual effort could be better allocated elsewhere. Additionally, AI was used to help me with Java's GUI library in which I do not have much experience.