Grace Frizzell and Summer Nelson

Prof. Weiss

AI

8 May 2024

Project 3

This project endeavors to examine whether the application of genetic algorithms might be useful in the building of a maze runner to find paths through 2D array representations of mazes. In our design of the problem, each maze is represented by a 2D array of integers—ones and zeros—in order to show where possible paths and impassable walls exist, respectively. Our maze runner makes up, down, left, and right moves to generate a potential path. As such, we hypothesize that a genetic algorithm could be applied to the generation of paths through this maze. Given a starting population of randomly generated paths, it is possible that randomly prompted mutations and crossover events between other paths could lead to the generation of a legal and optimal path.

4 files are required for this project. We store movement and current locations in the moveNode class, and we store the mazes and their various attributes, like dimensions and size, in the Maze class. The Solver class features the genetic algorithm, crossover, mutate, and population generation functions, and the main class features the main method which calls all of the appropriate methods to solve a maze. For now, we instantiate a maze in this main method using a 2D representation of our potential maze, and we have been using other AI to develop random test cases for the program. After that, one should simply be able to run the program, and the program will run until a high value is found or a predetermined time elapses. Then, the path of the current moveNode is outputted.

The genetic algorithm first begins with the random generation of moveNodes with maximum size random paths through the maze. Then, each of these nodes is evaluated with the fitness function, which tracks the moves through the maze and sets values accordingly. If the runner ever backtracks or makes an illegal move, the value is set as low as possible. Each legal move increments the score by one, and a winning move sets the value as high as possible. After this, mutations and crossover events take place according to random chance, which we have set at 50% after multiple rounds of testing to see what rate of change resulted in helpful moves. In a mutation event, a random move change occurs to the path. In a crossover event, two parental paths are spliced together to produce a new child path, which are then added to the population. The program runs until it finds a winning sequence or until a certain period of time has passed, as set in the main class. All data structures we used were either integer arrays or ArrayLists, depending on the need for the structure to be expanded.

Our experience and observation of our algorithm indicates that it is working as indicated in the pseudocode; however, for this application the method might be too slow. In the random generation of paths, it is difficult to check for illegal moves based on prior knowledge, other than those that start on a border of the maze. Thus, a lot of the moveNodes feature impossible moves and then are valued accordingly, and the rate of change does not guarantee that an advantageous change will be made between the sub-optimal path and the next. A fifty percent rate of mutation and weighted crossovers only do so much to speed up the processing time, and crossover also does little to affect it. We experimented with increasing the rate of change, but the unintelligent choice of subsequent moves resulted in little change relative to the fifty percent model. Overall, we learned what the application of a genetic algorithm might look like and how to execute such a program, but we also learned that this might not be the best in-context application of the genetic algorithm. We might support that a different algorithm is used in the future, especially to generate more intelligent initial populations.