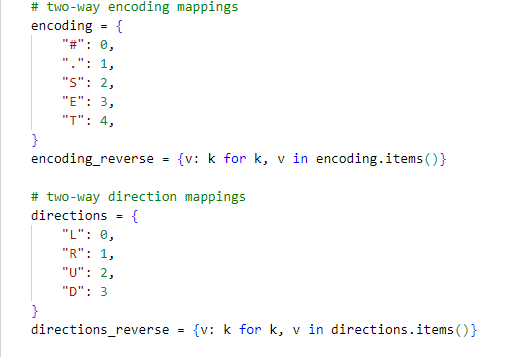
Intelligent systems: Seminar 1

Vid Hanžel & Aleksander Piciga

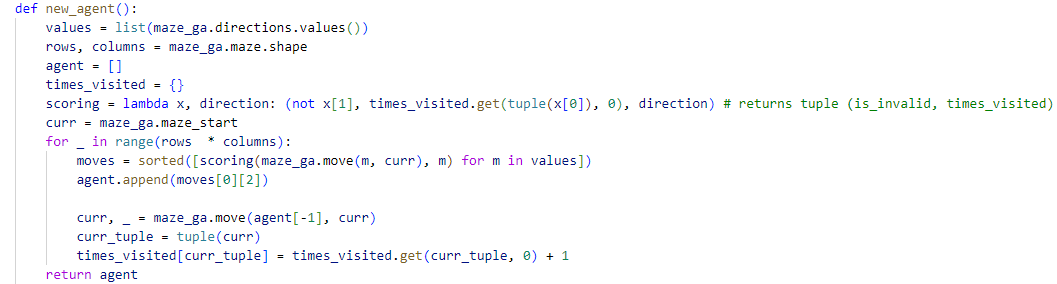
# Representation

We interpreted the maze as a 2D array of integers. The solution of the maze was represented by an array of integers.



# Initial population

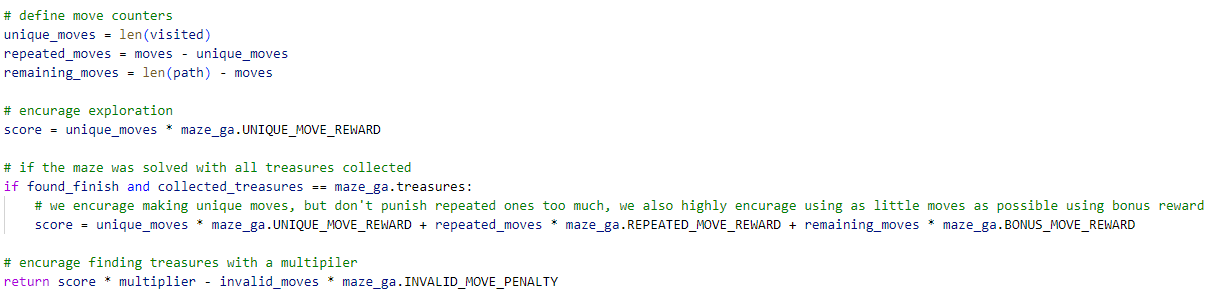
Each solution is of length equal to the maze size. We improved the random generation of solutions in such a way that when considering the next move the function only looks only at valid moves and prefers moving to a coordinate that is least visited.



# Fitness function

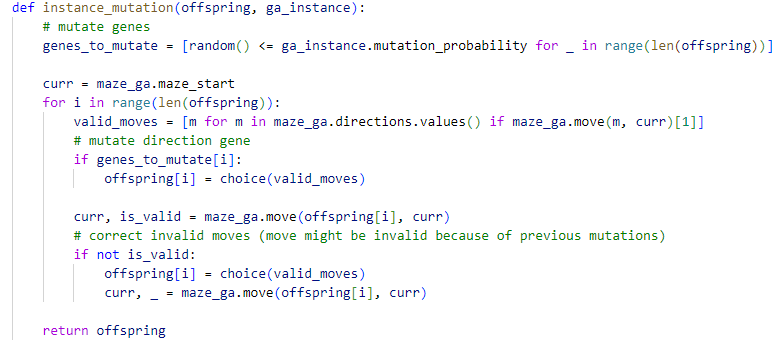
Our fitness function uses two different formulas depending on whether the solution collected all the treasures. If the solution does not collect all the treasures the score depends only on the unique moves made. In case the solution collects all the treasures and finds the exit we still reward making unique moves and additionally reward the more optimal path.

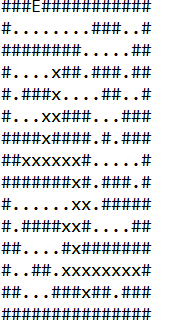
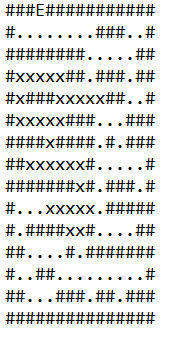
For each collected treasure or found exit we increase the score multiplier by one. The multiplier is applied to the final score and then we penalize invalid moves (goes out of bounds or hits a wall).



# Mutation function

The mutation function randomly selects genes to mutate and replaces them with a random valid gene, then the entire genome is checked and corrected such that the result is a valid path.

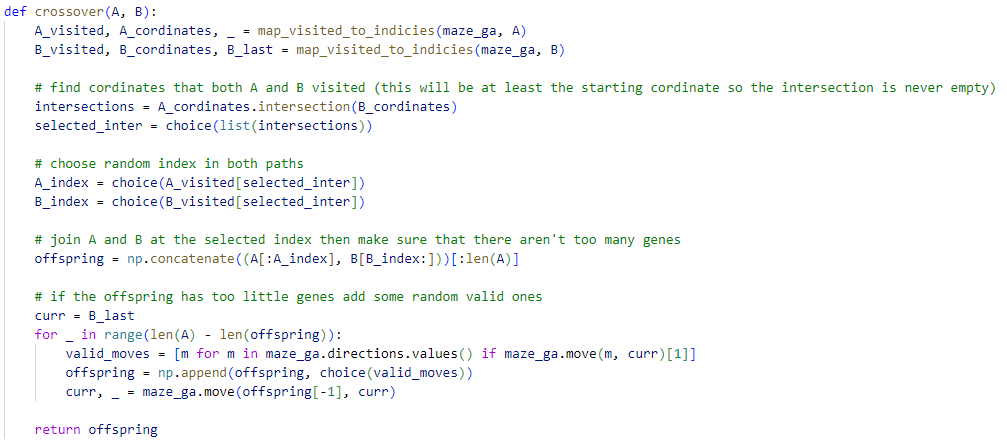


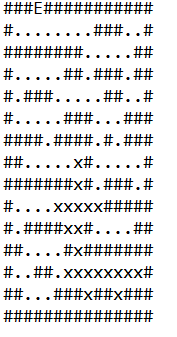
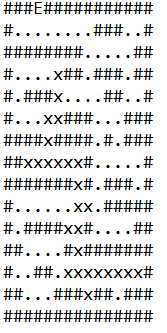
**MUTATION EXAMPLE**

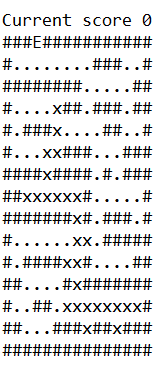
The image on the left represents the **original solution** and the image on the right represents the **mutated solution**. As we can see the bottom part of the solution is cropped and there is a new loop at the top left of the maze. These adjustments help diversify our population.

# Crossover function

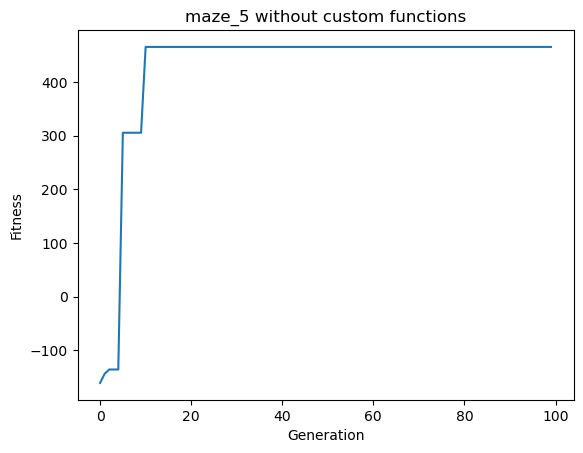
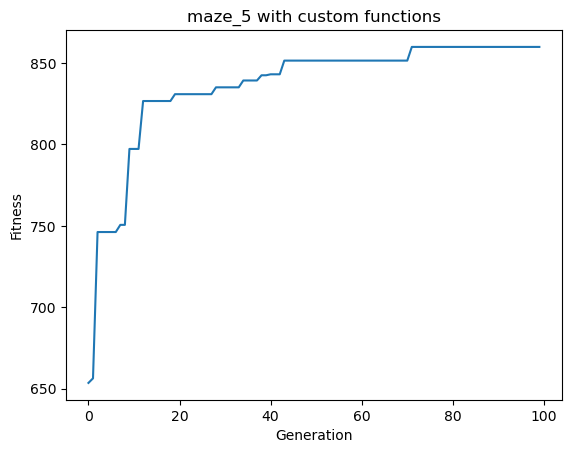
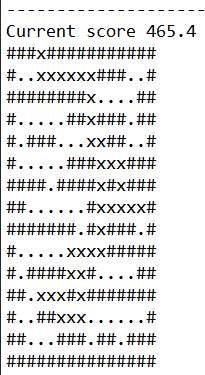
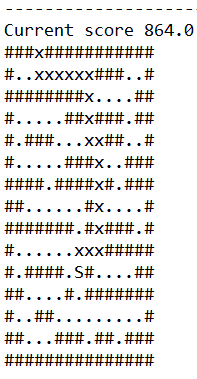
Crossover is customized so that it finds intersections of both paths and then randomly picks an intersection and splices them together (at least the intersection at the starting point is guaranteed to exist). If the resulting solution is too short, we pad it with random valid moves and if it is too long, we remove the extra moves.



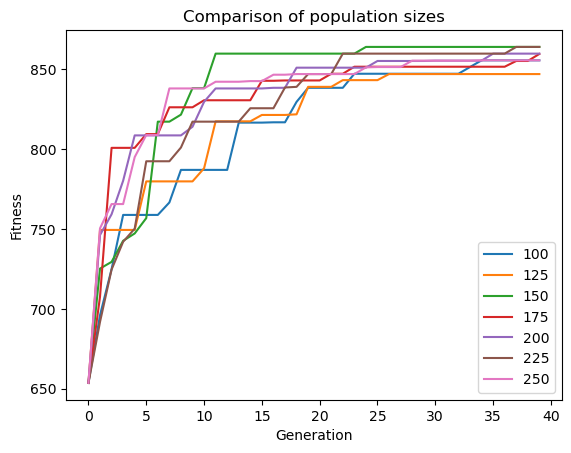
**CROSSOVER EXAMPLE**

The top two images represent the parents and the bottom image represents the child. As we can see the crossover happened around the starting point as the child instance has top features equivalent to the right parent and bottom features correspond to the right parent

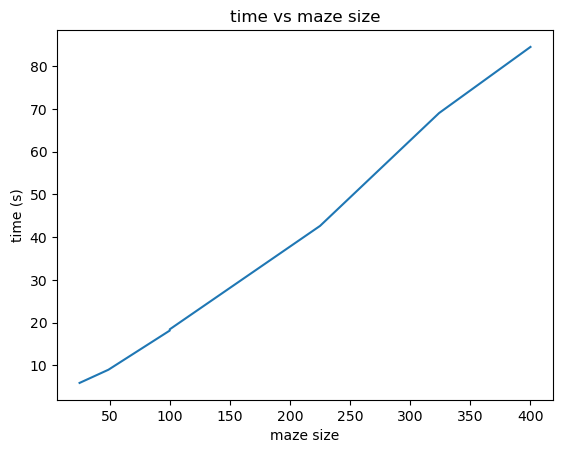
# Performance

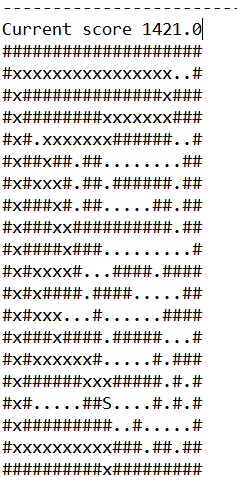
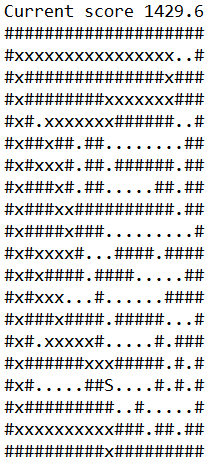
We ran the genetic algorithm with many different combinations of parameters and compared the performance. As we can see on the graphs below using custom functions greatly improves the fitness. While the run without custom functions only achieved fitness around ~**450**, the run with custom functions achieved **~850**. The run without custom functions starts with negative fitness as it includes invalid moves.

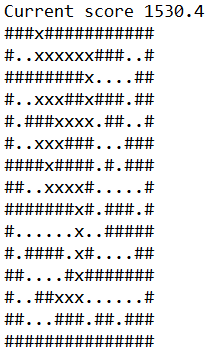
From the collected data we also extracted comparison among various initial population sizes. As we can observe from the graph increasing the population size improves the fitness to some extent, but at higher values the benefits of increasing population size are diminishing.



As our genome size is dependent on the maze size, we also looked at the increase of run time with larger maze sizes. On the graph below we can observe linear relation between genome size and time.



Our approach can solve most mazes with large enough population sizes, but struggles with collecting all the treasures in more complex mazes. Below is an example of a solution to maze\_7. The image on the left is the solution after 100 generations which includes some backtracking and visits some deviations from the optimal path. After additional 50 generations some of these deviations are mutated out, some minor ones still remain but would be removed when running the algorithm with more generations.

The example below illustrates the solution to maze\_treasures\_5 which collects all the treasures and finds the exit. The algorithm successfully found the most optimal solution without backtracking within 50 generations.