Allen Dorris

Assignment: Project Report

Due date: April 24, 2017 (Monday) by 11:30 AM

1)

The application replicates an artificial agent, referred to as the ‘character’, attempting to escape a randomly generated maze full of monsters, traps, and food. It successfully escapes the maze when it has reached the ‘goal’. The character has three ‘hearts’ of health. If it moves onto a trap or attacks a monster, it loses one heart of health, and if it eats food, it gains one heart of health. If the character reaches 0 hearts of health, it dies and is unable to search for the exit further.

Problem defined in terms of the five components:

**Initial State:**

The maze is randomly generated; as such, any state where two objects in the maze do not share a location is a valid initial state.

**Actions:**

The character can perform the following actions:

Move(x): Moves the character from its current location to adjacent location x, so long as x is valid.

Attack(x): Character attacks monster x sharing the same position.

Eat(x): Character eats food x sharing the same position.

Die(): Character dies and is unable to continue.

**Transition State:**

RESULT(In(CurrentLocation), Move(NeighboringLocation)) = In(NeighboringLocation)

**Goal Test:**

The character successfully reaches the location of the goal.

RESULT(In(CurrentLocation), Move(GoalLocation)) = In(GoalLocation)

**Path Cost:**

Every action has a path cost of 1.

2) The project is written with Java as its language. It uses an external grid library, edu.kzoo.grid, to visually represent the problem. One issue is because the maze is completely generated, some mazes may be unsolvable (i.e. the goal state is unreachable). A timeout method has been implemented to cause it to fail should too much time pass, though the timer is large enough that the algorithm should not be affected if it is possible to actually solve the problem.

2i)

The heurisitic function used for the genetic algorithm is a slightly modified Euclidean distance function, allowing only 1.50 times more slack in the travel cost. This is to account for the fact that there may be walls blocking the path of the character. A chromosome is randomly selected out of the population for use, and crossover/mutation are randomly performed on the selected chromosome. The only issue with this representation is that the genetic algorithm is slightly bugged. While the algorithm implementation is mostly correct, there is a minor bug that occasionally causes it to skip multiple spaces at once (ex. Going from (4,5) to (6,6), which is normally illegal as it can only move one square at a time). It is not clear what is causing this bug.

The chromosomes are selected in pairs from the chromosome, and act as:

00 = Up, 01 = Down, 10 = Left, 11 = Right

Ex.

[0011110111] = Up => Right => Right => Down => Right

2ii)

Simulated Annealing uses a pure Euclidean distance function to get the best results. The temperature function uses K and K\_Max, where K is the iteration count and K\_Max is 127, the Byte’s maximum value. The temperature’s value will continue to change and when the character reaches the goal state signaling the end of the algorithm, the temperature will be 0. Occasionally, the simulated annealing implementation will become stuck on a wall. While it will eventually correct itself, it can drastically increase the path cost or time as it will move to and from the same two squares many, many times.

The objective function is using the character location and goal location.

Ex.

Character at (4,5), Goal at (7,4):

√ ((7 – 4)2 + (4 – 5)2)) = √10

3i)

Fitness Function: -1 \* Cost(X, Y) – (MaximumCost(Z, Y) / 100.0f)

X represents the ChromosomePath, Y represents the goal location, Z represents the character location.

Cost: √ ((Y1–X1)2 + (Y2 – X2)2)

Maximum Cost: √ ((Y1 – Z1)2 + (Y2 – Z2)2)) + 0.50 \* √ ((Y1 – Z1)2 + (Y2 – Z2)2))

Total Fitness Function: -1 \* √ ((Y1–X1)2 + (Y2 – X2)2) - √ ((Y1 – Z1)2 + (Y2 – Z2)2)) + 0.50 \* √ ((Y1 – Z1)2 + (Y2 – Z2)2))

3ii)

Objective Function: √ ((Y1 – Z1)2 + (Y2 – Z2)2))

4i) Some of the accuracy numbers may be slightly skewed, as a maze may be generated that is impossible to complete (i.e. the goal state is impossible to reach). The larger the grid, the less likely this is to occur.

Genetic ran over 50 iterations on 15x15 grid:

Number right: 44

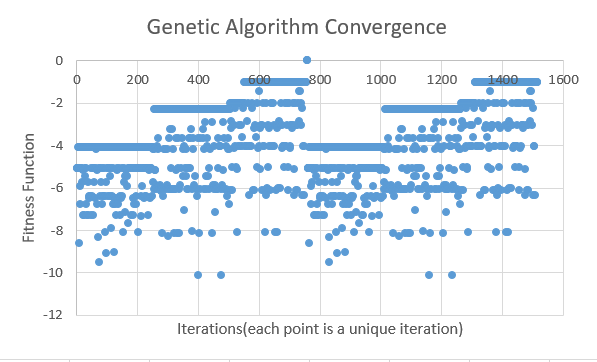
SA ran over 50:

Number right: 42

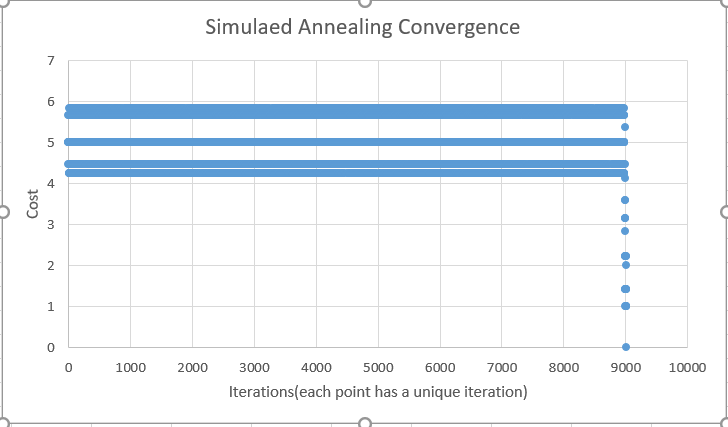
In general, it seems like the Genetic Algorithm has a slightly better accuracy.

4ii)

Genetic Algorithm run time over 50 iterations on 15x15 grid: 35 seconds



Simulated Annealing run time over 50 iterations:55 seconds



Genetic Algorithm converges at around 1900 iterations (approaching 0). Simulated Annealing converges at around 9000 iterations (approaching 0).

The Genetic Algorithm performs far better than Simulated Annealing in both accuracy and time, and converges far faster.