Scientifiy python
Project: grid car race

Ákos Mórocz, BBHBTK

May 12, 2022

1 Description of the game

The game description is based on the competition project of Introduction to Aritificial Intelligence course [1]. Short description:

- given track with several START and FINISH positions;
- goal: reach one FINISH point from the starting point;
- winner: reach one of the FINISH positions in the **fewest** steps;
- can be a multi-player game;
- steps of each player is one after the other in a fixed order (one round).

Steps:

- previous position: (x_{i-1}, y_{i-1});
- actual position: (x_i, y_i);
- next position: $(x_{i+1}, y_{i+1}) = (2x_i - x_{i-1} + k, (2y_i - y_{i-1} + l)$, where k, l $\in$ [-1,0,1];
- start position: $(x_{i-1}, y_{i-1}) = (x_i, y_i)$ chosen randomly;

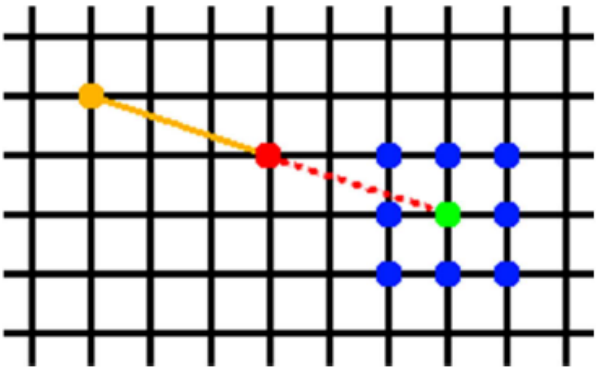

Figure 1: If the orange point is the previous position, the red one is the actual, then the player can move to the green point or to one of its blue neighbours [2].

- penalty for 5 rounds, respawn at the previous valid position as start position $[(x_{i-1}, y_{i-1}) = (x_i, y_i)]$ in case of
  - a step that leads out of the track
  - a step on a position occupied by other playes

2 GUI implementation

I designed and programmed the grid car race project in Python 3. Tkinter package is used for the GUI implementation. The workflow of the program is the following:

- Choose a map. The player can choose one map from the given maps by clicking on the image of the map.
- Add new player: many players can be added to the game by giving the name of the player. By "Add" button manual player, by "Add AI player" button automatic, AI player can be added to the game. The number of the players is limited by the number of starting positions of the track. This value can differ in case of different maps. After the player are added to the game, the game is started by pressing "Start" button.
- Controlling the manual player. The added players are shown in the map. The actual player is highlighted with filled, enlarged circle. The player can move one of the positions shown with circles around the player. The next positions are generated by the mentioned rule above. With the button on the right, the position can be chosen.

The program is divided into 3 subprograms:

- grid_race_main.py
- functions.py
- player.py

2.1 Subprogram: grid_race_main.py

The grid_race_main.py subprogram is responsible for the GUI implementation:

- load the track
- show the widgets
- handle interaction with the player

The loaded map is an image, which pixelvalues are transformed to 2D matrix, which store the map. Red pixel is represented by -1 as an out of trackt positions; blue ones are 1, as possible starting positions; green ones are 0, the valid positions of the tract; white ones are 100, the posibble FINISH positions, which have to be reached by the players.

The added players are postioned on one of the starting positions randomly (one of the blue positions).

The map is represented on a canvas, which also shows the positions of the players, their taken path, and their possible next positions. The canvas is updated after each movement of a player.

The next movement of the players are determined by their previous and current positions, shown in circles around the players. With the control button on the right, one of the 9 next possible values can be chosen.

The players follow each other one by one. The actual player is highlighted with enlarged circle. If a player takes an invalid movement (out of track, collision), it is banned for 5 rounds, it will skip the following 5 rounds, while the others can move.

The made actions are shown in a scrolled text box at the bottom. It contains the name of next player to move, the skipped steps in case of invalid movement, if it reached the FINISH position and the finish of the game (all players are finished). The number of steps are shown in each actions of the players.

2.2 Subprogram: functions.py

This subprogram contains some functions which can be used to check valid positions and to create a zombie array and an algorith to determine the next step of the AI player (check section *Implementation of AI players*).

2.3 Subprogram: player.py

This subprogram contains the class of the players. Some basic parameters: name, previous, current positions, number of taken steps, if it is an AI player. This class also contains the next movement of the player in case of manual and AI player as well.

3 Implementation of AI players

First, we have to explore the whole racetrack. In the framework, the track is given in a 2D matrix. We have to find the START and FINISH positions, represented in the 2D matrix with value 1 and 100, respectively. In order to determine the distance of each valid position from the FINISH position, I will use the *Zombie* algorithm.

The *Zombie* algorithm [**1**] is an iterative algorithm. This algorithm does the following: given a 2D grid, the racetrack. In case of valid positions (which value is not -1) each cell is either a zombie or human. Zombies can turn adjacent horizontal or verticalal (up/down/left/right) human beings into zombies every step. The first zombies are the FINISH positions. The task is to determine the distance from the FINISH position by infecting all humans. Increase the distance value by 1 in each step during infection.

- Initially push all the zombies (FINISH) positions into the queue.
- While the queue is not empty try to visit the all the four directions for each zombie because the effect of each zombie will be in all the four directions.
- After each set of zombies increment the distance by 1.

The return value of this function is a 2D matrix that contains distance values from the FINISH positions. An example of it is shown in fig. 2, where -1 is an invalid position represented with red, 100 is the FINISH point represented with white, the valid positions with the distance values are represented with green. The other white position is the START position.

As we know the racetrack in advance, I will use informed search method, as it results more efficient searching. Greedy search algorithm takes into account only the heuristic. The heuristic is the generated zombie array.

The *Zombie* algorithm provides the cost from the current position to one of the FINISH positions. Before each step, the search is executed, and the best path (which results the smallest value) will be chosen for the next step. The step is only executed if the line between the current and next position is valid (*validLine* function), otherwise the second, third and so on best option will be chosen. Because of the penalty, invalid positions will get high cost values depending on the neighbouring cost values, so they will not appear in the result of the searching.

The game can be played with multiple agents. The position of them is changed round by round, so the cost values of them are dynamically changing. If a position is occupied by another agent and this position is not a FINISH position, then it will not be added to the searching results.

References

[1] `https://www.geeksforgeeks.org/minimum-hours-taken-by-zombies-to-infect-all-humans-by-infecting-up-left-down-and-right-only/`

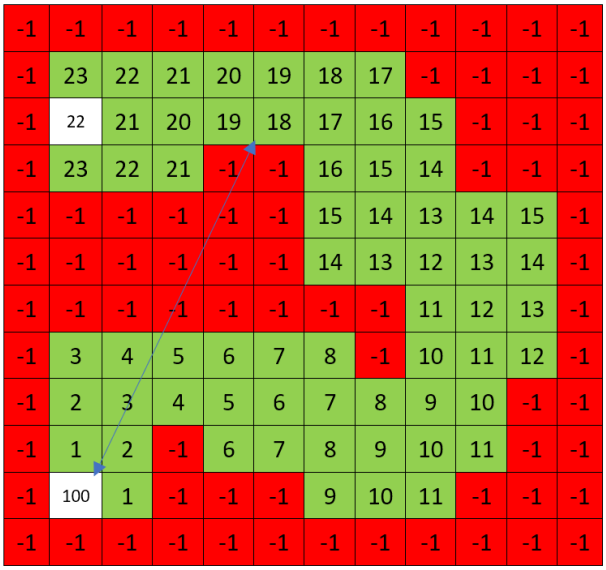

Figure 2: The result of the generation of zombie array.

[2] `http://users.itk.ppke.hu/~karacs/AI/`