Grid car race

Description of the game

A racetrack is given with a path. The path has several start and finish positions. The game can be played by more racers, who start from one of the start positions. The winner is the one who reaches the finish line in the fewest steps (the distance is not taken into account). Players take steps one after the other in a fixed order.

Each player is allowed to choose a step in the following manner. If the previous position of the player is (x_{i-1}, y_{i-1}), and the actual position is (x_i, y_i), as the next movement, the player can choose from the positions:

$$(x_{i+1}, y_{i+1}) = (x_i, y_i) + ((x_i, y_i) - (x_{i-1}, y_{i-1})) + (k, l) = (2x_i - x_{i-1} + k, 2y_i - y_{i-1} + l)$$
$$\text{where } k, l \in [-1, 0, 1].$$

At the beginning the previous and the actual positions are the same.

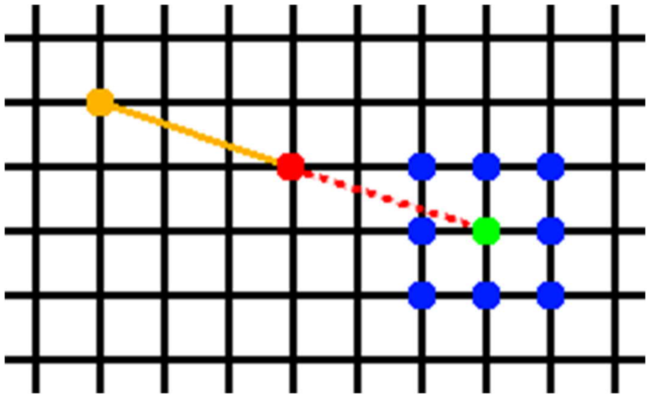

Figure 1. If the orange point is the previous position, the red one is the actual, then the player can move to the green point or to one of its blue neighbors.

The initial positions of the players are chosen randomly. If a player makes a step that leads out of the track, or to a position occupied by another player, it is blocked for five rounds, and it is restarted from its last valid position with zero speed. The player innocent in the accident can continue without any penalty.

The framework

The framework in which the player agents are tested can be accessed at the following URL: http://users.itk.ppke.hu/~karacs/AI/competition/env1.html. The framework allows two playing modes: a manual mode, in which inputs can be entered interactively, step by step; and a code play mode that allows agent codes to be run.

The aim of the competition is to implement a racer agent that plays on a racetrack, takes into account the borders of the path and the other players as well.

The competition will have three rounds with increasingly complex environments. After each round test runs will be carried out on non-disclosed tracks.

After loading the framework, first the map of track to be used and the playing mode have to be selected, and then the players have to be added one by one.

The HTML file realizing the framework can be downloaded and modified, including the JavaScript codes. However, the evaluation runs will take place in the framework provided as a reference.

Manual mode

After choosing the manual play, add one or more players. On the racetrack players appear at their initial position along with the nine next possible target locations. The next step can be selected with nine UI buttons. The name of the next player can be read in the status text area with some additional information. The manual mode is provided for understanding the environment.

Code play mode

Choose the code play mode and add at least one player. You have to give the name and the code of the player. The code has to be a class that contains the following two functions:

`init(c, playerData, selfIdx)`

- `c`: the matrix representing the racetrack. `c[i][j]` can take the following values:
    - negative number (e.g. -1): field is located out of the track, if a player moves there it receives a penalty
    - positive number or zero: regular fields of the track
    - 1: start positions
    - 100: finish positions
- `playerData`: information about the players. `playerData[i]` contains
    - `playerData[i].oldpos`: the players previous positions (coordinate)
    - `playerData[i].pos`: the players actual position (coordinate)
    - `playerData[i].penalty`: the number of penalty rounds of the player
- `selfIdx`: the index of the actual player in the `playerData` array and in the `playerAt(p)` function.
- The function must return within 10 seconds. Should the `init` function of a player not return within 10 seconds, the player will be disqualified.

`moveFunction(c, playerData, selfIdx)`

- parameters `c`, `playerData` and `selfIdx` are the same as in the case of the `init` function
- returns a point with fields `x` and `y` as integer coordinates in the set [-1, 0, 1]
- The function must return within 1 second. Should the `moveFunction` of a player not return within 1 second or return with an invalid value, it will be banned from 5 rounds.

The `init` function of the players are called only once at the beginning of the race, whereas the `moveFunction` is called in every step to obtain the next move of each player. In addition to these two functions the player code may define other functions and variables.

The structure point is a variable that has fields x and y. In JavaScript you can define a point easily:
```
p={x: 3, y: 4}
```

The player can use several other functions that are implemented already:

- `lc.validLine(p1,p2)` – returns true if the line between positions p1 and p2 is valid (including the two positions themselves), i.e. it lies within the track; otherwise returns false
- `lc.playerAt(p)` – if there is a player at position p, then returns the index of that player; otherwise returns -1
- `lc.equalPoints(p1,p2)` – returns true if both coordinates of the two input arguments are equal

At the beginning of the code play mode, player names and codes have to be added. Only the inner part of the function corresponding to agents has to be copied into the text field, as shown in the following example:

```
var customAgent = function(){
      /* copy from here
      var someVariable;
      this.init = function(c, playerData, selfIdx){
            // ....
      }

      this.moveFunction = function(c, playerData, selfIdx){
            // ....
      }
      */ // to here
}
```

The framework has a reference agent implementation (`randomAgent`) that steps randomly. This can be useful as a basis when building new agents.

Codes have to be uploaded to Moodle, as a JavaScript .js file, where the function name has to be the Neptun ID of the player.

Auxiliary information

About the JavaScript variables

JavaScript variables are typeless.

```
j=2;
l=3;
k='4'
l=l+j; // l = 5
l=l+k; // l = '54' as the concatenation of the two characters
l=+j+l; // l = 56, because the + sign changes defines the type as a
number
```

The scope of the variables is quite unique in JavaScript. If defined without var, the interpreter will search for the value, where it was defined with the keyword var. If it cannot be found, it is considered as a global variable. This behavior can be dangerous, so use the keyword var. The scope of the variable defined with var is the function (so a variable defined in a loop will be valid without the loop but inside the outer function as well).

Classes, objects

In JavaScript, almost everything is an object, so functions as well.
Classes can be defined as follows:

```
var ourClass = function(){
      var variable;
      this.fv1=function(){}; // public functions
      var fv2=function(){}; // private function
}
var ourObject = new ourClass();
```

JSON

JavaScript object notation, a simple description of an object:

```
var band={guitarist: 'john', singer: 'mary', drummer: 'zotyó'}; // the
structure will have three fields
band.drummer; // zotyó
band['drummer']; // zotyó
```

Arrays:

```
var bands=['Bon Jovi', 'Kis Grófo', 'Spice Girls', 'Justin'];
bands[1]; // Kis Grófo
```

Arrays may contain several elements with different types.

Objects can be nested:

```
bands=[
           jovilagvan: {
                guitarist: 'john',
                singer: 'mary',
                bass: 'máté',
                vilolin: 'pali',
                fans: ['peter', 'assad', 'kate', 'timi']
           },
           Justin: {
                singer: 'Justin',
                guitarist: 'ester',
                drummer: 'gizi',
                fans: null
           }
      ];
```

Thus

```
bands[0].fans[2] // kate.
```

Useful JavaScript functions

Array operations

```
var myArray=[]; // create an empty array
myArray.push(item); // add item at the end of myArray
k = myArray.pop(); // get and remove the last element of myArray
myArray.unshift(item); // add item at the beginning of the array myArray
k = myArray.shift(); // get and remove the first element of myArray
o = myArray.length; // length of myArray (not a function!)
```

Miscellaneous mathematical operators

```
l = Math.random(); // random value between 0 and 1.
y = Math.sqrt(x); // square root of x (+)
x = Math.sqr(y); // square of y
u = Math.abs(r); // absolute value of r
```