CMSC 421, Spring 2010: Project 2

Last updated on March 30, 2010

- Due date/time: Noon on Thursday, April 15.
- Late date/time (5-point penalty): Noon on Saturday, April 17.

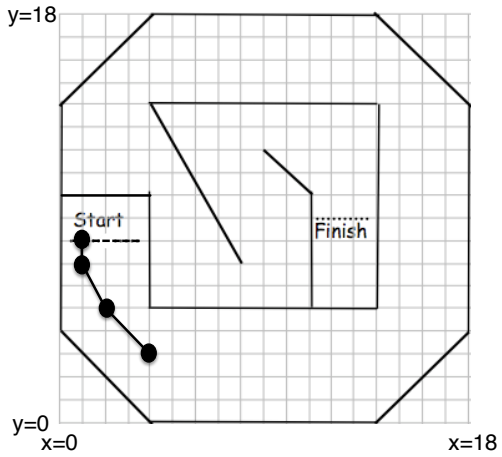

Figure 1: Example of the first few moves in a Racetrack game.

Racetrack is a game that was popular during the late 1960s and early 1970s. It is played on a piece of graph paper, on which someone has drawn an outline of a racetrack (e.g., see Figure 1). Each player's car starts at a point (x_0, y_0) on the starting line, with velocity vector $(0, 0)$. The players take turns moving. Each move has two parts: choosing a new velocity for the car, and moving the car to its new location:

- Suppose (v, w) is the car's old velocity. Then for its new velocity, you may choose any vector (v', w') such that $v' \in \{v - 1, v, v + 1\}$ and $w' \in \{w - 1, w, w + 1\}$.
- Let $(x', y') = (x + v', y + w')$, where (x, y) is the car's old location and (v', w') is the velocity that you chose above. If the edge (i.e., the line segment) from (x, y) to (x', y') intersects the racetrack's boundary, then the car crashes against a wall and you lose the game. Otherwise, the car's new location is (x', y').

If there are two or more players, the objective is to cross over the finish line before the other players do. If there is just one player, then the objective is to cross over the finish line in the least number of moves. For this project, you will implement functions that compute an optimal sequence of moves for the one-player version of the game.

Comment your code, indent it appropriately, and use good programming style (e.g., as in Norvig's *Tutorial on Good Lisp Programming Style*).

Data formats

In the input to your algorithm, I'll use the following data formats.

Edges and points. Each edge will be represented by a list of its endpoints, and each endpoint will be represented as a list of its (x, y) coordinates. The edge's endpoints may be given in either order. For example, the finish line in Figure 1 may be represented by either `((11 9) (14 9))` or `((14 9) (11 9))`.

Racetrack boundary. The boundary of the racetrack will be a set of edges, represented as a list. Since the list represents a set, the edges may be in any order. For example, the racetrack in Figure 1 can be represented by the following list:

```
(((4.1 0) (14 0))  ((14 0) (18 3.9)) ((18 3.9) (18 14))
 ((18 14) (14 18)) ((14 18)(4.2 18)) ((4.2 18) (0 14))
 ((0 14) (0 4))    ((0 4) (4.1 0))
 ((0 10) (4 10))   ((4 10) (4 5))    ((4 5) (14 5))
 ((11 5) (11 10))  ((11 10) (9 11))  ((14 5) (14 14))
 ((14 14)(4.1 14)) ((4.1 14) (8 7)))
```

States. A state of the world s will be represented by a list

$$((x\ \ y)\ \ (v\ \ w))$$

that gives the racecar's (x, y) coordinates and its velocity (v, w). For example, in Figure 1, the car's starting state is `((1 8) (0 0))`. The numbers in s will always be integers.

Paths. A path for the racecar is a list of states of the world $(s_0\ \ s_1\ \ \ldots\ \ s_n)$ such that each state is a feasible successor of the previous one. In other words, for every pair of consecutive states $s_i = ((x_i\ \ y_i)\ \ (v_i\ \ w_i))$ and $s_{i+1} = ((x_{i+1}\ \ y_{i+1})\ \ (v_{i+1}\ \ w_{i+1}))$, the following conditions must hold:

$$v_{i+1} \in \{v_i - 1, v_i, v_i + 1\};$$

$$w_{i+1} \in \{w_i - 1, w_i, w_i + 1\};$$

$$x_{i+1} = x_i + v_{i+1};$$

$$y_{i+1} = y_i + w_{i+1};$$

the edge $((x_i\ \ y_i)\ \ (x_{i+1}\ \ y_{i+1}))$ doesn't crash,
i.e., it doesn't intersect the racetrack boundary b.

Functions to write

1. Write a predicate `(intersectp` e e'`)` that returns `T` if the edges e and e' have any points in common, and `NIL` otherwise. For example:

```
(intersectp '((0 0) (1 1)) '((0 1) (1 0))       ⟹  T
(intersectp '((0 0.0) (0 1)) '((1 0) (1 1))     ⟹  NIL
(intersectp '((0 0) (2.5 0)) '((0 0) (0 5.3)))  ⟹  T
(intersectp '((1 3) (1 6)) '((1 4) (1 7)))      ⟹  T
```

HINT: Suppose $e =$ `(`$(x_1\ \ y_1)\ \ (x_2\ \ y_2)$`)`. Then for every point `(`$x\ \ y$`)` in e, there is a number $0 \leq t \leq 1$ such that

$$x = x_1 + t(x_2 - x_1);$$
$$y = y_1 + t(y_2 - y_1).$$

Similarly, suppose $e' =$ `(`$(x'_1\ \ y'_1)\ \ (x'_2\ \ y'_2)$`)`. Then for every point `(`$x'\ \ y'$`)` in e', there is a number $0 \leq t' \leq 1$ such that

$$x' = x'_1 + t'(x'_2 - x'_1);$$
$$y' = y'_1 + t'(y'_2 - y'_1).$$

Thus e and e' intersect iff there are numbers $0 \leq t \leq 1$ and $0 \leq t' \leq 1$ such that

$$x_1 + t(x_2 - x_1) = x'_1 + t'(x'_2 - x'_1); \quad (1)$$
$$y_1 + t(y_2 - y_1) = y'_1 + t'(y'_2 - y'_1). \quad (2)$$

From Equations 1 and 2, you should be able to derive formulas for t and t'.

2. Write a predicate `(crashp` e b`)` that returns `T` if the edge e intersects the racetrack boundary b (a list of edges as discussed earlier), and `NIL` otherwise. For example, suppose `b` is the list of edges given earlier. Then

```
(crashp '((2 4) (3 6.5)) b) ⟹  NIL
(crashp '((1 4) (1 2)) b)   ⟹  T
(crashp '((1 5) (4 5)) b)   ⟹  T
(crashp '((1 5) (4 6)) b)   ⟹  T
(crashp '((1 1) (1 2)) b)   ⟹  NIL
```

3. Write a function `(successors` s b`)` that returns the set of all feasible successors of the state s, i.e., the set of all states to which we can move from s without crashing. More

specifically, if s is the state `((`x y`)` `(`v w`))`, then `(successors` s b`)` should return a list of all states `((`x' y'`)` `(`v' w'`))` that satisfy the following properties:

$v' \in \{v-1, v, v+1\}$;
$w' \in \{w-1, w, w+1\}$;
$x' = x + v'$;
$y' = y + w'$;
the edge `((`x y`)` `(`x' y'`))` doesn't intersect the racetrack boundary b.

For example, if `b` and `s0` are the racetrack boundary and starting state shown in Figure 1, then `(successors s0 b)` may return the following list (with the edges possibly in a different order):

```
(((1 8) (0 0)) ((1 9) (0 1))  ((2 9) (1 1))
 ((2 8) (1 0)) ((2 7) (1 -1)) ((1 7) (0 -1)))
```

4. Write a function `(find-path` s_0 b l h`)` that uses the A* algorithm to find a path for the racecar. The starting state is s_0, the racetrack's boundary is b, the finish line is l, and the heuristic function is h. Your function should return a path $p =$`(`s_0 s_1 ... s_{n-1} s_n`)` such that the edge `((`x_{n-1} y_{n-1}`)` `(`x_n y_n`))` intersects the finish line. The path should take the shortest possible number of moves (i.e., n should be as small as possible).