# Intelligent Systems - Milestone 1

Timothy Tadj -s5178358

## Problem formulation and software design

<u>Problems/Actions:</u>

a) Reading in the file, recreating the board and storing the solution.

   To solve this problem, we will create a class called game that will store all of the data that is contained in the file. When a game object is initialised, it will open the rh.txt and store the board and solution for each of the questions in the file. The best way to do this would be to make a dictionary with index of the problem number that stores a board object containing the solution and a 2d array of the initial board.

b) Printing to screen the initial board and proposed solution, for each board.

   To solve this problem a method inside the board class should be made that when called will print the 2D array stored in the board() object in a formatted manner using the print() function, and then it should print the solution stored in the board() object.

c) Implement a single state.

   To implement a state, we need to know the parameters for each car on the board, these parameters would include; the orientation of the car, the length of the car and the position of the car. Store this information the board object will include a cars{} dictionary with a key of the car type, the key will correspond to a car() object containing all of these necessary parameters.

d) Create an expand function from which you can create more states.

   The expand function is basically looking to see all the next possible moves from the current state. To expand we will look to see if there is an empty space next to each car. If there is then we will store it as a potential move and return a list of all possible moves.

## Data structures Used:

Class car(): //parameters for each car in problem

  Bool isVertical   //true if car is verical

  Bool isHorizontal //true if car is horizontal

  Int size //stores size of car

  Int dimension[2] //stores the location of the head of the car

Class board(car): //contains the board and solution

  string board[6][6] //stores problem read from file

  string solution //stores the solution read from file dictionary cars{} //stores a dictionary of all cars in board (as a key) with their respective parameters

class Game(board): //stores all boards

//on initialisation reads .txt file and stores all initial parameters for each problem

dictionary boards {} //stores all board objects with question number as the key

## Functions Used:

car.properties()

<u>Description</u>:

Prints the properties of a specific car (for bug testing).

board.stringToBoard()

<u>Description</u>:

Used for initialisation of data structure.

board.stringToSolution()

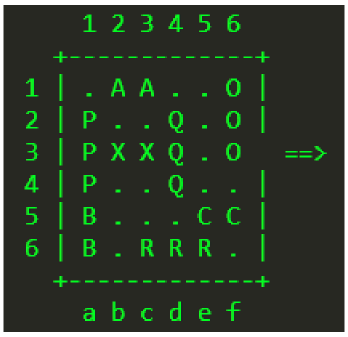
<u>Description</u>:

Used for initialisation of data structure.

board.printBoard()

<u>Description</u>:

Used to print the board in nice formatting. Looks like this:



```
     1 2 3 4 5 6
   +-------------+
 1 | . A A . . O |
 2 | P . . Q . O |
 3 | P X X Q . O   ==>
 4 | P . . Q . . |
 5 | B . . . C C |
 6 | B . R R R . |
   +-------------+
     a b c d e f
```

board.expand()

<u>Description</u>:

Returns a list of strings for all possible moves from the board by looking to see if there are empty spaces next to each car.

move(boardIn, mov)

<u>Description</u>:

Returns a copy of BoardIn (input board object) with the mov (instruction string) applied to it.

won(boardIn)

<u>Description</u>:

Returns True if a win is detected