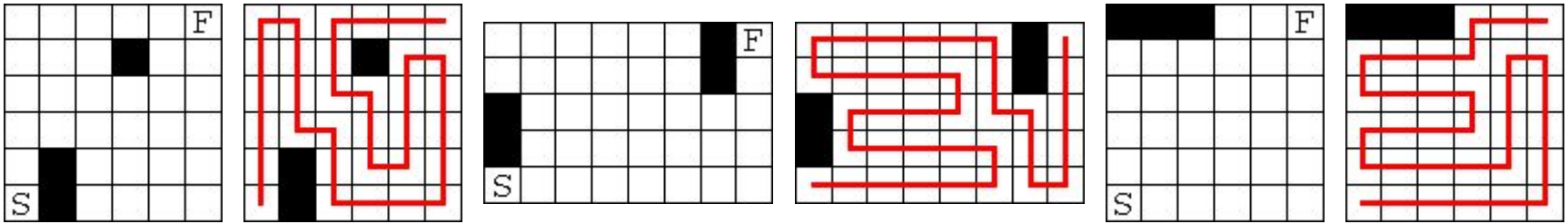
IART - Assignment 1

Checkpoint April 5th

Group 03_1A
Afonso Monteiro - up201907284@up.pt
Miguel Lopes - up201704590@up.pt
Vasco Alves - up201808031@up.pt

Work Specification

This assignment consists in the development of a program that will use heuristic search methods for a one player solitaire game, in our case the Unequal Length Mazes game. It consists of finding a path from the bottom left to the top right passing through every white square exactly once. The path must alternate between horizontal and vertical segments, and two consecutive segments can not be the same length. Each puzzle has a unique solution.



Examples of puzzles and their unique solution.

Problem Formulation

State Representation:

The board is represented through a Matrix with nxm dimension, where M[ni][mi]=k and k can be:

- **1** if there is a wall or if that square has been visited already
- **2** if that square can be visited (hasn't been visited yet)
- **0** represents the current position of the player

Additionally each game state holds the following variables:

- **moveCount**, a variable that indicates the number of moves made so far in the current direction.
- **previousMoveCount**, a variable that indicates the number of moves made in the previous direction and that limits the amount of moves that can be made in the current direction.
- **previousDirection**, a variable that indicates the previous direction (horizontal or vertical)
- **P=(x,y)**, a tuple with the position of the player to improve efficiency

Initial State

1	1	1	2	2	2
2	2	2	2	2	2
2	2	2	2	2	2
2	2	2	2	2	2
2	2	2	2	2	2
0	2	2	2	2	2

Final State (Solution)

1	1	1	1	1	0
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

Problem Formulation

Operators:

- MU(state), move up
- MD(state), move down
- MR(state), move right
- ML(state), move left

Evaluation Functions and Heuristics:

- Number of squares left to finish
- The Manhattan distance from the current square to the objective square

H = ManhattanDist(state, objective) + Count(not_visited) in state

Operator	Precondition	Effect
MU(state)	0 < y <= n - 1 and state.board[y-1][x] != 1 and ((state.previousDirection!=Up and state.moveCount!=state.previousMoveCount) or (state.previousDirection==Up))	newState.board[y-1][x]=0 and newState.board[y][x]=1 and P=(x,y-1) and newState.previousDirection=Up and (newState.moveCount+=1 or (newState.moveCount=1 and newState.previousMoveCount=state.moveCount))
MD(state)	0 <= y < n - 1 and state.board[y+1][x] != 1 and ((state.previousDirection!=Down and state.moveCount!=state.previousMoveCount) or (state.previousDirection==Down))	newState.board[y-1][x]=0 and newState.board[y][x]=1 and P=(x,y-1) and newState.previousDirection=Up and (newState.moveCount+=1 or (newState.moveCount=1 and newState.previousMoveCount=state.moveCount))
ML(state)	0 < x <= m - 1 and state.board[y][x-1] != 1 and ((state.previousDirection!=Left and state.moveCount!=state.previousMoveCount) or (state.previousDirection==Left))	newState.board[y][x-1]=0 and newState.board[y][x]=1 and P=(x-1, y) and newState.previousDirection=Left and (newState.moveCount+=1 or (newState.moveCount=1 and newState.previousMoveCount=state.moveCount))
MR(state)	0 <= x < m - 1 and state.board[y][x+1] != 1 and ((state.previousDirection!=Right and state.moveCount!=state.previousMoveCount) or (state.previousDirection==Right))	newState.board[y][x+1]=0 and newState.board[y][x]=1 and P=(x+1, y) and newState.previousDirection=Right and (newState.moveCount+=1 or (newState.moveCount=1 and newState.previousMoveCount=state.moveCount

Implementation

The chosen language to codify is Python

The data structures used include:

- A tree of nodes to hold the states
- A state class that has the board, the location of the player and the information of the previous move
- The board is represented using a list of lists
- The location is represented using a tuple of coordinates
- The previous move is stored using a string

What has been implemented:

- A State class
- A tree class (whose nodes will be consisted of State objects)
- The objective function
- Some methods to facilitate a player mode.