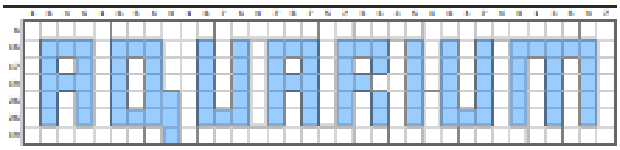
Checkpoint Assignment 1

Group 27



Nuno Oliveira up201806525

Luís Miguel Pinto up201806206

Marcelo Reis up201809566

1- Specification of the work to be performed

The chosen problem is the Aquarium from Topic 1 – “Heuristic Search for One Player Solitaire Games”.

The rules of Aquarium are simple:

- The puzzle is played on a rectangular grid divided into blocks called "aquariums".
- The objective is to "fill" the aquariums up to a certain level or leave it empty.
- The water level in each aquarium is one and the same across its full width.
- The numbers outside the grid show the number of cells that must be filled horizontally and vertically.

2- Related work references

For the purposes of this project only the slides from theorical classes were used and the website with the original game (https://www.puzzle-aquarium.com).

3- Formulation of the problem as a search problem

State Definition: A state is the same rectangular board (a bi-dimensional array of squares) where the aquariums may or may not be filled up to a certain level (>=0) and still obey the restrictions put on by the horizontal and vertical numbers outside the grid. A board representation where a level in an aquarium is only partially filled is

not valid state similarly to ones that have higher levels filled and leave lower ones unfilled.

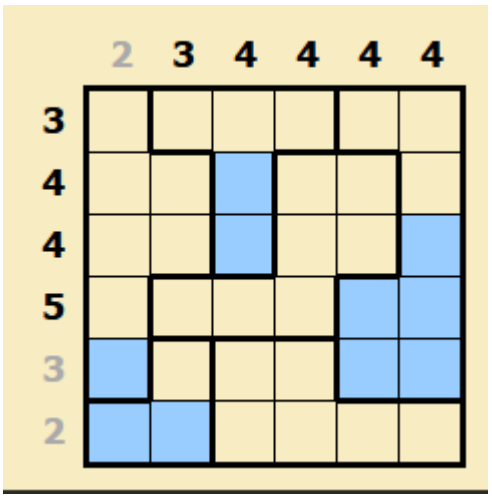


Figure 1 - Valid State

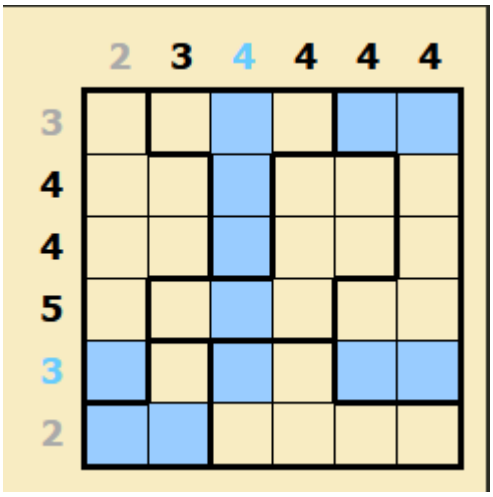


Figure 2 - Invalid State

Initial State: The initial state will consist of the empty board.

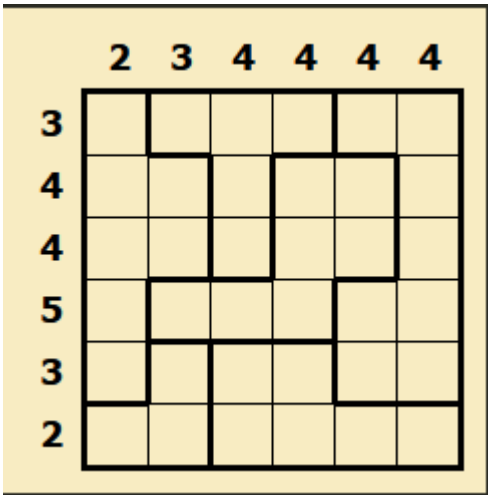


Figure 3 - Initial State

Objective Test: The objective test must first check if the state is valid and then for each row and column verify the amount of filled squares against the specified.

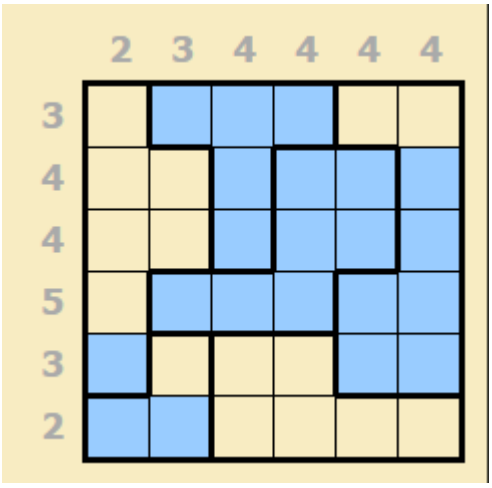


Figure 4 - Final State

Operators:

Name - Fill square.

Preconditions - The square must be unfilled first.

Effect - The hole level it is on will be filled and so will the one under it.

Cost - 1.

Heuristics:

G(n) – Number of moves up to the current state.

H(n) – The number squares yet unfilled.

4- Implementation work already carried out

We chose to use Java as the programming language with Gradle for build automation and dependency management. For the graphic part we will use Swing's API

The board, as already stated is a bi-dimensional list. This specific list is not used by the AI because we implemented an abstraction of it to minimize the number of possible actions and thus also optimizing the search methods.

```
List<List<Square>> matrix = new ArrayList<>();
```

The vertical and horizontal numbers:

```
List<Integer> horizontalCount,verticalCount;
```

A square will belong to a level so when it is filled, the game will automatically fill the aquarium up to and including that same level. This is possible because the levels have references to level immediately under it (if it exists).

```
public class Level {
    private List<Square> squares = new ArrayList<>();
    private boolean painted = false;
    private Level nextLevel;
```

This way, an aquarium has a list of levels instead of the squares it holds.

```
public class Aquarium {
    private List<Level> levels = new ArrayList<>();
```

The Board class is the one that contains a map of Aquariums.

```
private Map<Integer,Aquarium> aquariums= new HashMap<>();
```

On another note, all the search methods will use the same graph. The only difference will be the way the queue is prioritized.

As for the file structure, the classes that handle any drawing, game logic and search methods will have their own package in “UI”, “board” and “graph” respectively.