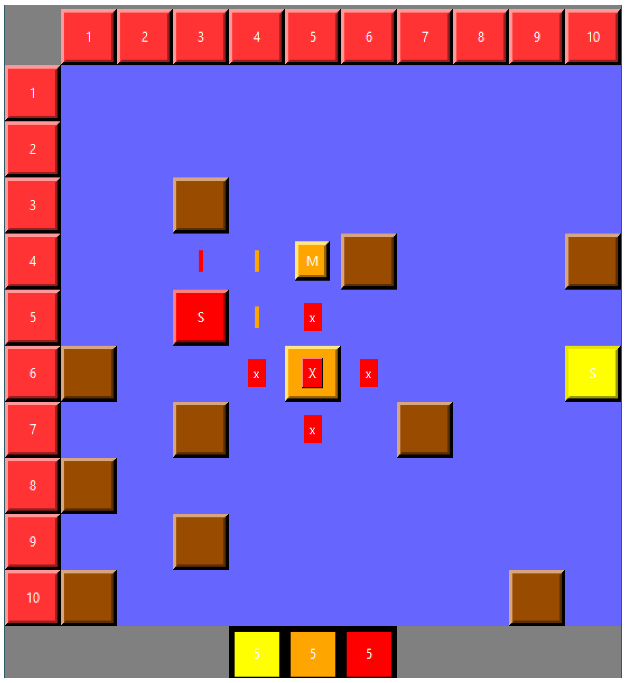
LINGI 1131 - Computer language concepts - 2020

Captain Sonoz Project



Group 18

Camberlin	Merlin	0944-17-00
Pisvin	Arthur	3711-17-00

Submitted on : May 6, 2020

Professor : P. Van Roy

1 Introduction

This report is a complement to the code that details information about the operator mode of the players implemented. The report is separated is several parts. The first explains the behaviours of both players implemented. The second aims to explain the different structures used abs the last one details which extensions have been implemented.

2 Strategies

2.1 The basic player (`Player018Basic.oz`)

The strategy followed by this player consists of adopting a random behavior.

Move Initial position and moves are selected randomly but the player cannot go outside the map or in a island. He has just as likely to choose any direction or surface.

Fire Item The player decides to charge either a mine or a missile . In fact, drone and sonar are never used since this player doesn't keep track the predicted positions of its enemies. The preferences order to fire an item is the following: missile, mine. Missiles are preferred since it can possibly damage an enemy further away (in general) and therefore avoid damages to the submarine itself.

Missiles Missiles are launch to a random position within reach in water. Launching a missile can damage the player itself.

Mines Mines are put randomly within reach in water. The player keep the list of its mines in memory. He randomly decides to explode it, or not. Exploding a mine can damage the player itself.

2.2 The hard player (`Player018Hard.oz`)

The strategy followed by this player consists of adopting a smarter behavior. The main idea is to store enemies' positions.

Moves Moves are selected randomly until a position of enemy is known. Once a enemy's position is known, the submarine heads towards him. This way, the enemy will finally be within reach of a missile.

Enemy's position Enemies' positions are memorised. They can be upgraded in function of these different messages.

- `SayMove` If the position of the enemy is known, this position is modified in function of the direction of the enemy. Else, this message is ignored.
- `SayAnswerSonar` The sonar gives 2 coordinates. The player keeps the two coordinates because he cannot know which is the wrong one.
- `SayAnswerDrone` The drone give either a column or a row. The player keep the information and put the other coordinate to 0.

By default, the enemy's position is set to (0,0). This indicates that no information concerning this enemy haven't yet been received.

Fire Item The player randomly decides to charge an item. This way, the submarine collects a variety of items. The preferences order to fire an item is the following: drone, sonar, missile, mine. Drone and sonar are preferred because they are mandatory to map enemies' positions. Sonar isn't the first since, it's impossible to determine which of the two coordinates answered is the correct one. Between missiles and mines, missiles are preferred.

Missiles Missiles are only launched if the enemy can be hit without damaging the sender itself and only if the predicted position of the enemy is approximately known. It means that if a coordinate x or y is in memory, the player sends a missile on the row or the column if the player can launch so far away.

Mines Mines are only exploded if there is a chance to damage an enemy (according to its predicated position) without damaging the owner of the mine itself.

3 Implementation and design choices

3.1 Implementation

In the implementation, several record has been defined to store different information. There are at the number of three :

1. `state()`: used to store information about the current state of the submarine.
2. `playerState()`: used to store information about the current state of the player during the game.
3. `gameState()`: used to store information about the current state of the game (only used for turn by turn mode).

3.1.1 `state()`

The record `state` is used in the player file to know the state of the player. It has the following features:

- `<id>` = id of the submarine
- `<position>` = current position of the submarine
- `<lastPositions> ::= nil | (<position> '|' <lastPositions>)` = previous positions visited by the submarine since the last surface.
- `<damage> ::= 0 | 1 | 2 | ...` = current number of damages taken by the submarine.
- `<loads> ::= loads(mine:<i> missile:<i> drone:<i> sonar:<i>)` = record of the current loaded items.
  where `<i> ::= 0 | 1 | 2 | ...`
- `<weapons> ::= weapons(mine:<i> missile:<i> drone:<i> sonar:<i>)` = record of available weapons.
- `<mines> ::= nil | (<mine> '|' <mines>)` = list of previous placed mines.

  where `<i> ::= 0 | 1 | 2 | ...`

In the `medium player` and `hard player`, information about enemies' position are stored and therefore, the following feature is added:

- `<enemies> ::= nil | (<enemy> '|' <enemies>)` = list of enemies to attack.
  where `<enemy> ::= enemy( id:<idNum> position:<position>)` = record to represent an enemy with its id and its predicted position

3.1.2 `playerState()`

The record `playerState` is used in the `main.oz` file to send information to the player . It has the following features:

- `<port>` = port created for the player
- `alive ::= true | false` = boolean value to indicate if the player is still alive or not.
- `turnAtSurface: 0 | 1 | 2 | ...` = represent the number of turns spent at surface.

3.1.3 `gameState()`

The record `gameState` is used to keep track of the state during the game : how many players are still alive and the order to play. This record is specially useful for the turn by turn mode.

It has the following features:

- `nbPlayersAlive::= 0 | 1 | 2 | ...` = number of players still alive
- `playersState: nil | (playerState '|' playersState)` = a list of `playerState` (one for each player)

3.2 Design choices

The `main.oz` is split in two main parts to distinguish the turn by turn and simultaneous modes.

- In the turn by turn mode, the `gameState` record keeps track of the order to enable players to play in turns. The game is over when only one player alive remains.
- In the simultaneous mode, each player has its own thread. There is no longer any question of order and turns. In this case, no more `gameState` is therefore needed. The game is over when there is one player left.

Another point that need to be emphasize it the reset of enemy's location. Once a missile is launched or a mine exploded, the position of the enemy targeted is voluntary reset. Since enemies' positions are (hardly) never precisely known , resetting the position enables the player to get a new estimation of its position.

4 Extensions implemented

4.1 A third player (Player018Medium.oz)

A third player has been implemented. Its behavior is exactly the same as the `hard player` except that his moves are random. This player has been implemented due to observations. In fact, according to situations, it's something more useful to move randomly. Actually, the problem with the `hard player` is that he heads toward an enemy so much that he becomes within reach of an enemy missile. By moving randomly, the number of times, the submarine is within reach of enemy missiles should be less.

4.2 Map generator

A map generator has been implemented as a function taking the number of row and column specified in the `Input` file. The `MapGenerator()` function randomly places islands on the map. It has a local variable called `Ratio` representing the approximate ratio of islands on the map. The ratio is defined as :

$$\text{Ratio} = \frac{\text{Number of island}}{\text{Number of columns} \cdot \text{Number of rows}}$$

By default, the ratio is set to 10%.

4.3 `explosion(ID Position)` and `removeMine(ID Position)`

`explosion(ID Position)` and `removeMine(ID Position)` have been respectively implemented and modified to emphasize location where a potential enemy can be damaged. When these messages are received on the `GUI`, small squares are temporarily drawn on the map. The squares are colored according to the color of the submarine that sends the missile or that explode the mine.

5 Conclusion

To summarize, three players have been implemented. The first with a random behavior that doesn't keep track of enemies positions. On the contrary the two others keep track of them and send missiles or explode mines when it can damage the enemy. The differences between the `hard player` and the `medium player` state in the moves chosen.