LINGI1131
Computer Language Concepts
Captain Sonar

LAMOTTE Pierre - 65441500
Ecole Polytechnique de Louvain
UCLouvain
p.lamotte@student.uclouvain.be

***Résumé*—This document describes the *Captain Sonar* coded in *Oz*. Implementation strategies will be explained for the *GUI*, *Input*, *Main* and player files.**

I. Players

Players have been implemented with records and basically do the same actions. They ara implemented with the following keys :

id ID of the player (ID number, color and name) ;

life Total number of lives the players starts with (initialized to the maximum damage the player can support *-refered in Input-*) ;

mines Number of mines the player can place ;

minesPlaced List of the unexploded mines the player placed ;

missiles Number of missiles the player can launch ;

sonars Number of sonars the player can launch ;

drones Number of drones the player can launch ;

path List of the positions the player has taken since he last dove ;

dive Gives an information about if the player is at the surface or not ;

pos Current position of the player.

A. Thor

Based on the nordic god **Thor**, this player has been implemented as a pure brutal player. He does not think much and mainly attacks.

Thor is a bit dumb and sometimes decides not to charge any item *-on average 1 in 8-* and mainly charges missiles. This player always places the mine on its previous position and fires them randomly. This player does not know what to do with information he can collect through sonars or drones becaus **Thor** is not a good strategist.

B. Loki

This player adds the **enemyPos** key to his description record. That key refers all the positions the other players anounce and helps **Loki** define its strategy.

Based on the nordic god **Loki**, this submarine is deceitful. This player remembers enemies' announced positions and checks if one of its placed mines is closed to one of the registered positions before he fires it. If nothing interesting has been found, **Loki** exploses a random mine in his list.

II. Main

I decided to implement the game manager using the object-oriented programming paradigm. I thought cells could help me implement players' basic changes.

After having spawned the players *-as objects-* at their initial position, the manager asks what kind of game has to run *-turn by turn or simultaneous-*.

In turn by turn, a list *-Living-* of the living submarines is initialised and filtered at each turn. This allows the manager to define the end of the game by checking the list at each turn to know if there's more than one player alive. If more than one player can coninue playing, each one makes a couple of actions *-as described in the instructions file-* at its turn.

In simultaneous mode, a cell is created and is incremented each time a player dies. This allows the manager to end the game once there's only one player left. The counter is checked every time a player finishes a turn.

III. Extensions

A. GUI

Images have replaced water, islands, mines, submarines and pahts and the grid axes have been reshaped *-background, shape and foureground-* the same way the life pannel has been.

Figure 1. New images

B. Input

A random map generator has been implemented. This creates a map with a random ratio of water/islands *-between 4 and 6 times the number of players-* over a map randomly sized *-between 4 and 10 times the number of players-*.