Computer Language Concepts : Project

Gustin Simon 11711400
Pinon Brieuc 68411400

April 2017

1 Introduction

The project consists of making an implemention of the board game "Captain Sonar". From an object point of view, we have players that interact on a game board and a graphical interface that represent the current situation. A game controller controls the synchronization of the players and the graphical interface.

2 Implementation choices

Two modes had to be implemented, a sequential mode *turn by turn*, where only one player plays at every moment one after the other, and a concurrent mode, *simultaneous* where each player plays at different speeds and concurrently. In the each mode, players are modelized by ports objects, they answer questions and update their states with information given, but they do not spontaneously act. It is the controller in *main.oz* that makes the game take place. But the controller does not store directly information about the state of the game, but rather the players that do by handling their currents states.

The controller needs two pieces of information in order to do his job : the life state of the players and the surface information about each player. The life state is asked to the player by checking if the player bind *ID* to *null*. In *turn by turn*, the surface information is handled by two determinitic lists (is at surface, and for how much time) that are updated by recursive call in the arguments; in the *simultaneous* case a simple delay is set and no information has to be stored.

2.1 Turn By Turn

In *turn by turn* mode, figure 2.1, we only have one controller since the actions are sequentials. The paradigm used is synchronous message passing.

2.2 Simultaneous

[!h] In *simultaneous* mode, figure 2.2, we have one controller by player. Each controller is roughly the same than in *turn by turn* mode but with delays.

3 Players

We made two different artificial intelligence. The first one has a behavior that is completely random. We called it "Random AI". The second one is more reflexive, but still has some random behaviors. We called it "Basic AI".

Both AI have a similar architecture : the AI is a port object with a state, for which a thread is running, calling recursively a function that treats the messages sent to the port when they arrive. This function that is called everytime a message arrives on the port is defined as follows : `fun {Behavior Msg PlayerID State}`. The parameter `Msg` is the message received by the port, `PlayerID` is the ID

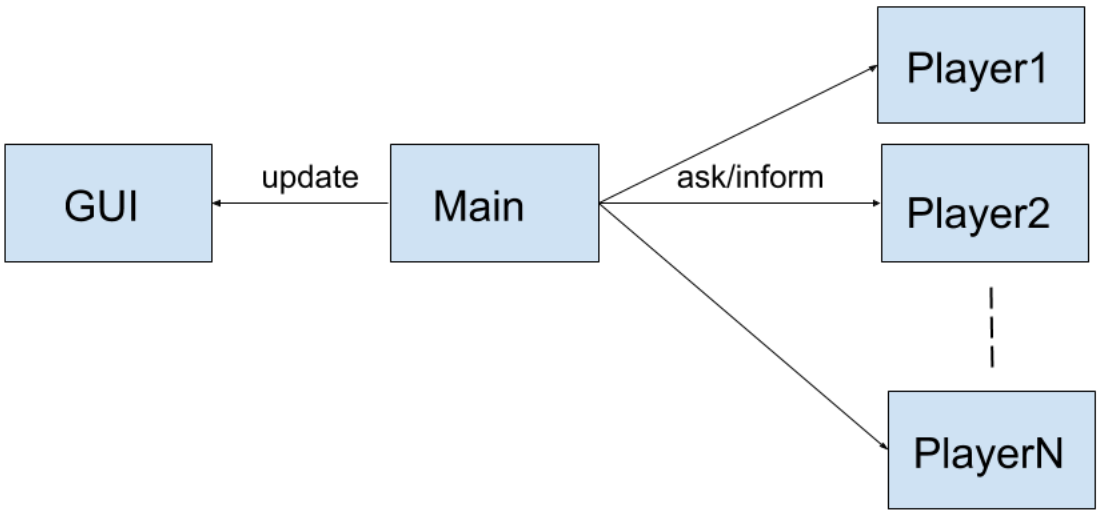

Figure 1: Structure in *turn by turn* mode

of the AI (which it will never change) and `State` is the current state of the port object. The function returns a new state, that will be given as the parameter `State` on the next call to the function (when the next message arrives).

The state of each object is a record containing all the information that the AI could need to treat any message sent to its port. In a sense, it can be seen as the state of the attributes of an object (in an OOP paradigm). The exact format of these states will be explain later on.

The treatment of a message is simply done by doing computations on the information provided in the message and the current state of the port object. An updated state is then created and returned by the function `Behavior`.

It is worth mentioning that as it was asked to make a defensive code, we added lots of `else` clauses and an `ERR` procedure so that an error message is printed onto the oz browser when a record or a variable doesn't have the right value or format. We also added a `WARN` procedure and a `Fct` procedure that would respectively print warnings and debugging information for debugging purposes.

3.1 Random AI

3.1.1 State

As we stated, AIs use a record with a certain format as the state of the port object.

The format of the state this AI uses is as follows : `stateRandomAI(life:_ locationState:_ weaponsState:_)`. The field `life` simply contains the remaining life of the submarine, while the two other fields contain another record that represent a state. We chose to do this that way so we wouldn't have to much variables to write on every `case` statement on the state of the port object.

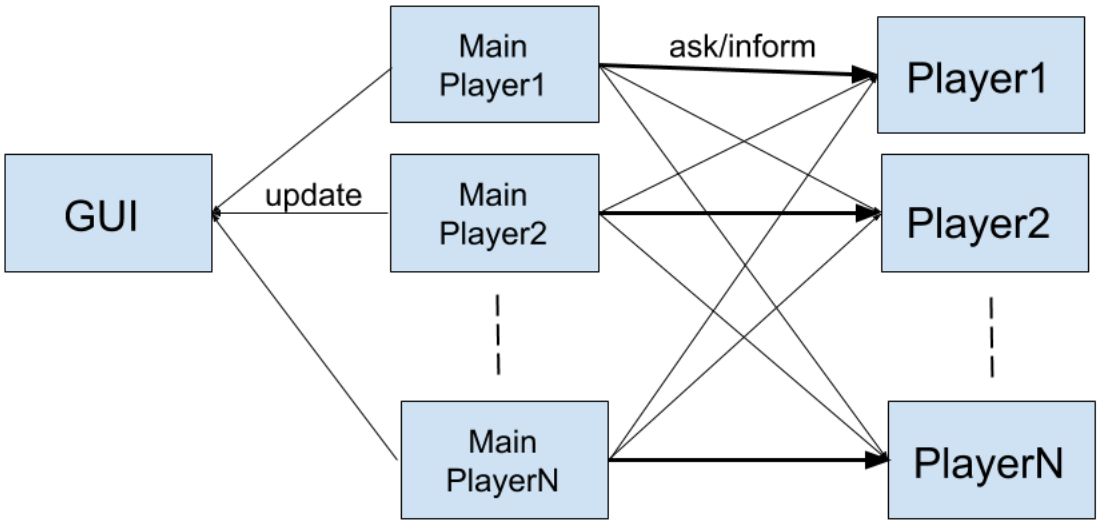

Figure 2: Structure in *simultaneous* mode

After that, the field `locationState` contains a record of type `stateLocation (pos:_ dir:_ visited:_)`. Its field `pos` contains the position of the submarine (as a record of type defined in the instructions), the field `dir` contains the last direction travelled by the submarine or `surface` if it is at the surface and the field `visited` is simply a list of all the positions visited by the submarine on this diving phase.

The last field of the state of the AI contains a record of the type `stateWeapons (nbMines:_ minesLoading:_ minesPlaced:_ nbMissiles:_ missilesLoading:_ nbDrones:_ dronesLoading:_ nbSonars:_ sonarsLoading:_)`. Each field named `XLoading` contains the number of charges of the weapon of type X while each field named `nbX` contains the number of weapon of the type X that have been charged and are available to the submarine. Necessarily, the value contained in `XLoading` is always smaller than the number of charges needed to load a weapon of type X. The field `minesPlaced` contains a list of the mines that have been placed by this AI but haven't yet exploded. Hence, when the AI decides to explode a mine, it chooses randomly a mine in this list, remove it from it and explode it.

3.1.2 Behavior

As stated before, this AI behaves randomly but still respect the rules of the game. It chooses its initial position randomly (but still as a valid position in the water on the grid), it moves randomly (but never goes twice on the same position in one dinving phase), it chooses to load weapons and which weapon to use at random (if at least one weapon of this type is available), or can choose not to use a weapon at all.

More closely about its moving behavior, this AI doesn't move if it is at the surface. If it isn't, it chooses randomly to get to the surface. If it decides to stay underwater, it tries to move in a random direction. If the new position is invalid (out of the grid or an island) or if it has already been visited, it tries again to move in a random direction until it finds a direction that it can travel to. Obviously, if there is no position around the submarine that is valid, it is forced to get to the surface.

Randomness is still an important component in the answering process of sonar probing. Since the AI can choose to fake one of its coordinates when answering a sonar probing. Thus, it randomly chooses which coordinate to fake and randomly chooses the value it will answer for it. Therefore, it is possible (but improbable) to return both the right coordinates.

It doesn't keep track of any other player and thus ignores the information about them provided by the game controller as well as the answers of the sonars and drones that it fired.

3.2 Basic AI

3.2.1 State

The state of this AI has a format that is close to that of the random AI but includes slight differences. The format is defined as follows : `stateBasicAI (life:_ locationState:_ weaponsState:_ tracking:_)` where the fields `life` and `locationState` are the same as the ones in `stateRandomAI`.

The field `weaponsState` still contains the state of the weapons but is slightly different : `stateWeapons (minesLoading:_ minesPlaced:_ missilesLoading:_ dronesLoading:_ lastDroneFired:_ sonarsLoading:_)`. The field `minesPlaced` still contains a list of the unexploded mines placed by the AI. We chose here to remove the fields named `nbX` to keep only the fields named `XLoading` which contain the total number of charges loaded by the submarine and can then have values greater than the one needed to load a weapon of the type X. The number of weapon of this type available is then computed when needed. We made this choice simply because it requires less code when using `case` statements on this state and our code is then less prone to include mistakes. The downside is of course that it requires a little more computation to be made. One additional field is present in `weaponsState` : `lastDroneFired` which obviously contains the last drone fired by this submarine. It is needed so that the AI knows what row or column was probed by the drone when it receives a message that a drone answered. Indeed, the drone answer messages are only a boolean for each player, stating if it was detected by the drone.

The last field in `stateBasicAI`, `tracking`, contains all the information gathered by the AI about the positions of the other submarines. Therefore, it contains a list of records (at most one for each opponent) that include the current information that this AI has about the position of each other player. These records are defined as `trackingInfo(id:_ surface:_ x:_ y:_)`. The field `id` contains the ID

of the submarine that this record contains information about. The field `surface` contains either a boolean stating if this submarine is at the surface, or the value `unknown` if this information is not known by this AI. The fields `x` and `y` each contain a record that is composed of information about the current position of the player along each axis and the certitude about this information. These record can have three different formats : `unknwon` if the coordinate is unknown, `supposed(_)` if the coordinate has been gathered using a sonar and can then be wrong (the coordinate is then stored between the parentheses and finally, `certain(_)` if the coordinate is undoubtedly the one stored between the parentheses. This information is updated everytime the AI gets a message about the movement of another AI or gets an answer from a drone or a sonar it fired. Of course, the list is also updated to remove information about opponents that have died since this information is no longer needed.

3.2.2 Behavior

This AI has a more complex behavior.

As the random AI, it still chooses its initial position in a random way. Choosing always one of the corners is another strategy we considered as it is usually a safer place than the center of the grid. Nevertheless, this is not always the case because of islands, and using this strategy would have implied to change the movement behavior explained hereafter.

Concerning its movement behavior, it is the same as the one of the random AI as long as it has no target. When a player is targeted, it goes in its direction to be able to shoot, but stays away enough to not get damages from its own weapons. A target is acquired when the AI knows undoubtedly both the coordinates of an opponent. To get to the target, we decided simply to make the submarine go in the direction of its target. There is no need to make a much more complex behavior since getting stuck is improbable as the target is always moving. Yet, to be sure not to get stuck, we added some randomness to the movement even when the AI has a target : sometimes, the AI will decide not to go in the direction of its target.

The AI decides to get to the surface only if it has no other choice.

To get the positions of the other players, the AI uses sonars and drones. First, it fires a sonar to get an idea of the positions of every player. It puts the positions answered as supposed positions in the tracking information. Then, it checks which coordinate is right and which isn't by sending drones to those rows and columns. If the answers of the drones imply that a coordinate marked as *supposed* is right, we mark it as *certain*; on the contrary, if the answers impy that a *supposed* coordinate is wrong, we mark it as *unknown*. Then, a sonar probing will be needed if the AI only has information about one coordinate of a player.

The choice of which weapon to load and to fire is driven by the most precise information we have about a player. Hence, as soon as both the coordinates of any player are known by the AI, it loads either mines or missiles and moves to a position which it can shoot it from. If the most precise information it has requires drone probing to precise the information, the AI will decide to load and fire drones. This is respectively the same thing for sonar probing.

The choice of using missiles rather than mines, as both are attack weapons, is made using an arbitrary formula that uses the range of permitted placement of the weapon, the maximum distance of placement and the cost (i.e. the number of charges to get a weapon of this type). If both are considered as powerful, the AI chooses randomly everytime between one or the other. This being said, it also takes account of which one can be available on the next turn.

Nevertheless, the behavior regarding mines is a little different than the one concerning missiles since mines can be exploded whenever the AI wants. We just programmed this AI so that everytime it gets asked if it wants to explode a mine, it checks for each mine if it knows with certainty if another submarine is in the range of it. It explode a mine only if it can make damages to another player and if it is away enough not to hurt the current AI.

The last behavior that is worth mentioning is the faking of one coordinate when answering to sonar probing. When a sonar asks the position of a player, this player has the permission to fake one of its coordinates. The behavior of our AI is the same as the one of our random AI : it randomly chooses which coordinate to fake and randomly chooses the value it will answer for it.

4 Extensions

4.1 GUI

A small message and sound are respectively displayed and played when a player die. To hear the sound the `mp3` codec has to be decodable (`libsox-fmt-mp3` for example).

4.2 Map generator

A random map generator is implemented, each box has an independent probability to be island or water, the probability of the island case over the water case can be adjusted.

5 Problems encountered

While this project didn't seem very difficult at first sight, it was not that easy to fulfill.

The first problem that we encountered was to have a code that was clear and improvable. Indeed, since it is a group project, we needed to be able to read each other's code.

Next, we had a problem of a bad implementation choice. We thought of making more port objects in our artificial intelligences. Namely, we thought of making one that would keep track of the location of the player, one that would keep track of its weapons, etc. and just send messages to these port objects in order to change their state and exchange information with it. We would have put those port object either in the state records of the `Behavior` functions or as parameters of this function. Nonetheless, we decided when we began the coding process to use only one port object for each player because we thought it would make the code easier to read and to maintain. It turned out it was not really the case : using more port objects could have made our code more easily readable and maintainable.

Last, we couldn't find a problem in the code of our basic AI that make the program wait after a few turns. Fortunately, this bug only happens when there is a certain number of players in the game. Indeed, it works perfectly for a game with a random AI and a basic AI, for instance.