LINGI1131 - Project's Report - 2020

The Mysterious Dungeon

Caytan Antoine
de Moffarts d'Houchenée Gauthier

Last modification on : 29th April, 2020

Professor :
Peter Van Roy

Assistant :
Mathieu Jadin

1 Introduction

As part of the LINGI1131 course, we were asked to implement a customised version of the popular game, Captain Sonar. So, we completely redesigned the GUI to end up with *The Mysterious Dungeon*. This version of the game takes place in an old abandoned dungeon in the north of England. Famous for these evil beings, brave knights try to eliminate them to regain possession of the place ! But beware, this dungeon is sometimes likened to a deadly labyrinth with impassable spikes and bewitched chests ! Good luck...

2 Player's strategies

We have implemented two different players. We will call them "Random" and "smart". Their behaviours differ greatly.

Indeed, the random player makes arbitrarily choices while respecting the statement and avoiding to hurt himself. While the smart player takes in count the moves and injuries of the enemies. In addition to that, the smart player always start a game by launching a sonar and a drone to quickly localise his enemies. And this is only a small bit of his stategy. See the subsection 2.2 to see how smart it is !

Nevertheless, they react in the same way to a few messages without much impact on the strategy. Here is a complete list of features that are the same for both players :

- They both set their initial positions randomly and if the chosen position is on spikes, this position is shifted at the east (or at the west spot of either the next row or at the northernmost row)
- when the players receives the permission to dive, the player remembers it to launch an error if the main asks him to move when it shouldn't.
- When a mine or a missile explodes near them, they both bind the message respecting the statement without drawing any conclusions about the enemy's possible positions.
- Their reactions to the message "sayPassingDrone" and "sayPassingSonar" are identical since they only have to answer the question. If the map is such as there are more colums than rows then the answer to the question "sayPassingSonar" is his X coordinate and a lie in Y coordinate (and vice versa). We made this choice in order to give as little information as possible to the opponent.

As you can see, our both player only have a limited amount of identical "functions". Let's now review all players particularities ! You will find in the two following subsection a list for each player telling which behaviour is adopted in reaction to a each message.

2.1 "Random Player"

- move : This player chooses an arbitrary direction among the ones that will not lead to some spikes, walls or a previously visited spot (since the last surface). If the player can't move, he moves to the surface and the player remembers that it has lost his permission to dive.
- chargeItem : Choose arbitrarily an item. If this item is already fully loaded, it choses an other one. If it can not charge any item, then nothing is charged
- fireItem : Fire as soon as a weapon can be used without hurting the random player. The position of the weapons is chosen randomly among those that aren't spikes (or mines if the item is a mine).
- fireMine : If a mine is available the player blow it up if it doesn't hurt himself.
- the other messages are ignored, mentioned at the beginning of section 2 or rejected if the message is illegal.

2.2 "Smart Player"

- move : If the player has a limited ranged weapon, it heads towards the closest reachable position at the minimal range of this weapon from the player whose supposed position is the most accurate (mines take priority over missile). If none of these positions are reachable, the same procedure is applied with the positions of all enemies. If the player doesn't have any limited ranged weapon then it heads towards the closest position being out of range of the weapons. This is done while avoiding the spikes, walls or a previously visited spot (since the last surface). If the player can't move, he moves to the surface and it remembers that it has lost his permission to dive. This way of doing is supposed to protect the player and allow it to fire when it is needed. To move efficiently, a **BFS** (breadth first search) is computed.

- chargeItem : As this player always begin the game with a sonar and then a drone, these two Items are charge before any other at the beginning of the game. Then it is the missiles and mines that take priority ! The player will always prefer to charge a missile if they have a wider range and only need 1/4 more loads at worst than mines. Otherwise, mines are preferred. If the preferred weapon is already fully loaded then the item charged is chosen arbitrarily among the item that aren't fully loaded.
- fireItem : Try to fire the item on the player whose supposed position is the most accurate. To chose the position where to fire, the position with the highest probability of damaging the targeted player is chosen. The sonar and the drone have priority among the other weapons since we have to locate our enemies as quickly as possible and once it is done, the sonar and drone are almost never loaded. If the item is a drone, it chooses to drone over a row if there are more rows than columns and vice-versa otherwise.
- fireMine : If a mine is available the player blow it up if it doesn't hurt himself and damage a enemy.
- sayMove : the player takes a lot of advantage from this message as it can say where enemies are ! The player keep a matrix of one's and zero's to indicate the possibilities of position of the different enemies. Almost each time a enemy moves, we can eliminate some potential positions from the enemy's matrix since it cannot move on spikes or out of the map.
- sayAnswerDrone and sayPassingSonar : Unlike the random player, these functions are not ignored ! The smart player then removes the impossible positions from the matrix of the enemies.
- : sayDeath : if a enemy is dead then his matrix is removed from the "memory" of the smart player.
- sayDamageTaken : if the enemy has been touched by one of his missile then the smart player knows where the enemy is or at least localise him among 4 positions (if the enemy only took 1 damage).

3 Implementation

3.1 Design and Choices

We divided our code in small pieces to make it easily readable and editable. The variable names are either well explained or have been chosen to be understandable at first glance. Each non-trivial function is accompanied by it's description.

In the `Main.oz`, a choice was made to keep 3 arrays of informations throughout the execution : `PlayerPortList`, `SurfaceJoueursList` and `VieJoueursList`, which contains the ports, life and the "surface-state" of each player, respectively. This was done by placing these 3 values as arguments of the functions that execute the game in turn by turn or simultaneously.

3.2 Limitations

- If the map is totally full of spikes, our function reacting to the message `initPosition` will loop. We decided to keep this limitation even if it is easy to fix because the input.oz file is considered as correct.
- The sound extensions (see section 4.3) are only executable on Windows. They have been implemented by the `{OS.System}` instruction which use a command specific to the Windows terminal.

4 Extensions

4.1 Layout

As we said in the introduction, we completely redesigned the layout of Captain Sonoz to an "old school" theme, *the dungeons*. First, we created the map, by replacing water by a brick floor, the island by spikes and the border by walls. For the gameplay, we recontextualize some actions. Instead of sending a drone, a player will invoke a fairy that will unobtrusively spy the other player on a given row or column. Moreover, when a player want to erase his path, he disappears in a magic hole. Finally, The mines are replaced by trapped chests that explodes [1].

1. Obviously, we love explosions

FIGURE 1 – Graphical Elements - Map

After that, every player have now, a different skin related to his color.

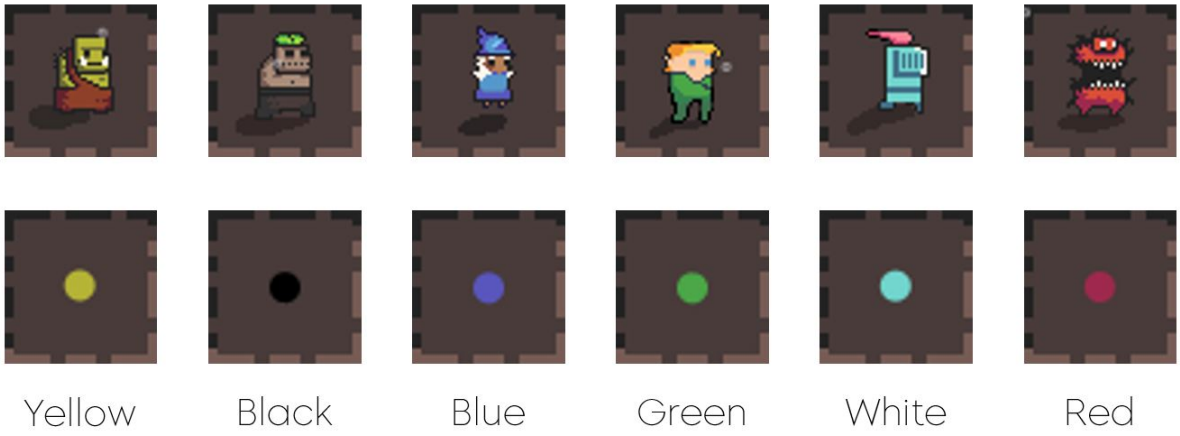

FIGURE 2 – Graphical Elements - Players

4.2 Animations

3 animations have been implemented in the game :

- The spy fairy : She flies over the row or column chosen by the player who summons her.
- The escape : When a player want to erase his path, he disappear in a magic hole under his feet.
- The explosion : when a trapped chest is engaged or when a player throws a fireball.

4.3 Sounds

To complete the animations and to give some energy to the game we've add different sounds.

1. *Start* : when the game starts.
2. *Play* : music played during the whole game.
3. *Ouch* : when damage is taken by a player.
4. *AAAaaah* : when a player dies.
5. *Surface* : when a player decides to erase his path and disappears in a magic hole.
6. *Explosion* : when an item explodes.
7. *Drone* : when the spy fairy is invoke.
8. *Victory* : when the game ends.

The `Input.oz` and `GUI.oz` files were not submitted on the basic INGInious task to avoid errors due to automatic correction. These files are present in the .zip submitted on the "Additionnal files" task.

The layout was inspired by the open-source work of *ox72*[1]. As for the sounds, they are available on the Youtube channel *Sound Library* [2]. At last, the author of the game music is *8 bit Universe* [3].

Références

[1] https ://0x72.itch.io/

[2] https ://www.youtube.com/channel/UCXm0O2N7IWtjSGv9tblq34g

[3] https ://www.youtube.com/user/8BitUniverseMusic