Université Catholique de Louvain

Ecole Polytechnique de Louvain

Ozploding bozmbs

Project 2019 - LINGI1131 - Computer Language Concepts

Group 5

Brieuc de Voghel - 5910 1600
Séverine Mersch-Mersch - 3298 1600

Friday 3rd May, 2019

Contents

1 Introduction

For this project, we managed to implement all mandatory parts.

Side note : We implemented the specifications given in the project statement as well as we could, given our understanding of some ambiguous rules. Therefore, our game may behave differently from other implementations on some (non-critical) parts like player movement sequence, thinking delay, etc. However, this does not interfere with the proper working of the game but could only affect performances of some players during interoperability (further explanation in Section 4).

Of course, every decision was carefully thought about and can be explained during our interview.

In addition, we participated with Group 003 in the creation of a GitHub repository where all groups could share their compiled Player files in order to facilitate interoperability tests between groups[1].

In order to have a better game experience, we tried to make a more user friendly GUI : simple squares were replaced by images and buttons to easily start, quit and restart a game were added.

Finally, small extensions were added to the game : bonuses have an additional chance to drop a new life or a shield that protects from the next time the bombers gets hit by fire.

The bigger part of the time spent on the project was for developing a more advanced and smarter player, named Umberto. This was proven successful by another sharing platform[2].

2 Structure of Implementation

Our project is divided into different parts, all of them are implemented as port objects in order to communicate correctly with each other. What we mean by "correctly" is that, since the program uses concurrency, sent instructions will always be treated in the right order. Therefore, no dead/life-locks can exist (with a bit of care when implementing the port objects) and we get the expected behaviour. We have six main parts, the function Main that launches everything, the board (GUI) that solves all the graphical part, the Player Manager that generates the players, the Map Manager that remembers the layout of the map (where the players, the bonuses, the fire, etc. are) the Bomb Manager that takes care of the bombs and their explosions and, finaly, the Notification Maganer that gathers all the information and broadcasts it to the concerned managers.

The makefile contains the different mandatory parts. It also contains useful smaller rules (e.g. the `workInProgress` rule) that compile precise functors in order to avoid to compile everything when we do a little modification.

Let's have a look at the step by step execution of a new game :

- Main creates ports to communicate with the different Managers and the GUI.
- It then tells the GUI to build the game window and it creates the different bombers through the Player Manager in function of the variables within the Input file.
- Before letting the bombers play, it assigns a spawn position to all of them, he spawns them and then waits for the "Start" button to be pressed.
- Once the "Start" button pressed, the bombers are run turn by turn if the "isTurnByTurn" flag is set to true or a different thread for each player is created if the flag is set to false.
- All along the game, a "Quit" button is available to properly stop the game. There is also a "Relaunch" button to quit and then restart another game.
- If the game mode is simultaneous ("isTurnByTurn" set to false), each player is repeatedly asked to perform an action after a delay as specified in the Input file and bombs are managed separately in the Bomb Manager.
- If the game mode is turn by turn ("isTurnByTurn" set to true), all the players are asked to perform an action at (virtually) the same time, then bombs are detonated and a small delay is waited before iterating the next turn.
- The game ends if one of the three defined conditions are met. (As defined in the specifications, more than one bomber can end with the same number of points. There can thus be more than one winner, even if they don't finish alive in the board.)

[1] See GitHub repository : `https://github.com/Nicwalle/LINGI1131-Bomberman-Players`

[2] See sharing platform : `https://www.unimate.be/oz`

2.1 Managers' role

The manager that defines the graphical part of the game is the GUI. Small modifications were made to the given file. Three buttons were made (or replaced the original one) in order to start, end or relaunch the game properly. The colours have changed and images now replace some previous simple boxes for nicer graphics.

The Player Manager (given file) generates ports for the different bombers by launching their initialisation function. It was modified in order to be able to use the new bombers files.

The Notification Manager is the port where, by default, all instructions are sent to. It has a transmitting role : when it receives an instruction, it simply transmits it to the concerned Manager(s) and/or bombers.

The Map Manager remembers the state of the map. It begins by initialising the map. This means that it creates a cell for each square, and gives it a number to determine if it is a floor, a wall, a box that can transform into bonuses or points, etc. It also initialises the position of the players to negative points (out of the board).

When receiving instructions (mainly by the Notification Manager, except for get instructions) it reacts in consequence : it executes the moves of the bombers, tells them if they got a bonus or points, computes the victims by telling who got hit, etc. by sending the instructions to the concerned Managers through the Notification Manager.

In addition, it remembers some information like the number of boxes left and the number of lives of each bomber.

The Bomb Manager is in charge of planting bombs with their timer set according to the Input file. It then makes the bombs explode when the timer reaches 0 (decremented every turn). The positions that have to explode are calculated and the positions where to make fire appear are sent to the GUI. The timer of all the bombs are decremented at every turn, after the actions of all the bombers [3]. For the simultaneous part, the explosion delay is computed in a separate thread.

2.2 Use of cells

We allowed ourselves to use cells in the implementation as it was not restricted in the project statement[4]. Of course, they were wisely used in order not to have any concurrency problem. Indeed, the cells never exceed the scope of a precise function or the scope of a Manager (who has only got one instance). Thus, at no time a cell can be concurrently modified or accessed.

The use of cells allowed us to implement a more clear code. This led us to be able to concentrate more on the implementation itself than solving a pure coding problem. In this way, we were able to create a more complex bomber for example.

3 Players

After we understood the proper working of a bomber by re-implementing the given file of a random bomber, we decided to go with two players capable of reaching some objectives. Both our players can make use of our extensions : a life and a shield bonus.

3.1 Basic Player: Bomber005Tozzi

Our basic player, named Tozzi, does the minimum necessary things a player has to be able to do :

- He accepts the different messages and answers to it correctly, by giving his ID, his assigned spawn position, etc.
- He counts his own number of lives, bombs and points; knows if he's got a shield, and can share all this information
- If he gets hit, he updates his life counter if he has no shield, puts his state to off and informs of his death
- When asked for an action to execute, he tries to move as close to the centre of the map as possible, either by moving, either by placing a bomb to get rid of a box in it's way

[3] Again, this was our interpretation of the project statement; to be discussed during the interview

[4] When notified in the middle of the project that the use of cells is restricted, Hélène Verhaeghe allowed us to use them if their use was clearly justified.

- At the beginning, Tozzi initialises a map with all the information and he remembers the initial places of the players and boxes. But being basic, he doesn't update this information, except the box explosions

3.2 Advanced Player: Bomber005Umberto

Our more sophisticated player is called Umberto. In addition to all what Tozzi can do, he updates a lot of information :

- He remembers when a player spawn at some place,
- When a player moves, he keeps in mind where he moved and updates the map if a player took a bonus or point
- If a player dies, he kows he is of the board (at pt(x:-1 y:-1)
- If a bomb is planted, he writes it on the map to know that there is some danger in the zone around the bomb
- When a bomb explodes, he writes that there is no more danger on the map because of that bomb.
- If a box is removed, he knows that a bonus or a point appeared, so he adds this info on the map.
- When you ask him which action he wants to do, he executes the function *ExecuteAction* that gives him the best place to go for the objective we gave him :

ExecuteAction calculates the different possible moves that Umberto can do. Staying at his place and planting a bomb if he has one, or moving to one of the cardinal points if it is allowed (avoid boxes and walls). If there is only one possible move Umberto does that one, otherwise the function *Best* is executed to find the best thing to do. *Best* is the function that decides all the strategy of the player.

First of all it looks all around Umberto to see what is available in a distance of *MaximumDistance*. To do this, he propagates his movement in the four cardinal directions recursively and holds in memory the nearest interessting thing of each kind (box, bonus, points, safe place) and where he first has to move to reach it.

Afterwards he chooses the following option in this order of priority :

- If he is in danger, go to the the nearest safe place
- If he is at a distance 1 of a box, he plants a bomb.
- He moves towards a bonus then a point, then a boxBonus and then a boxPoint.
- Finally, if there are no other available and interesting things around, he moves randomly between the possible moves by avoiding dangerous places if possible.

4 Interoperability

All the players we did interoperability tests with were files uploaded to our sharing platform on GitHub (see Introduction).

Our tests consisted in running the players several times, as well with "isTurnByTurn" set to true as false. As not all players had the same extensions, the "useExtention" flag was most of the time set to false. We tried to sum up all the different characteristics we observed.

Remark : in our implementation, the thinking delay of the players is implemented in the main parts, while our players just answer directly the action that they want to do. We observed that other players achieve the thinking delay when we ask them what action they want to do. Hence it was sometimes an advantage for our player that could do more actions than the opponents in a period of time.

4.1 Interoperability with Player000bomber

000Bomber is a basic player that does random movements 9 times out of 10 and plants a bomb otherwise.

Tested characteristics	Grade / Remark
Behaves like expected	Yes
Illegitimate actions	None
Extensions	None to test
Thinking delay (simultaneous)	In player
Match result vs. Umberto	Won 1/10
Other remarks	Over 100 actions, 10 of them were bombings (just like expected)

4.2 Interoperability with Player003John

John is a basic random player with no extension.

Tested characteristics	Grade / Remark
Behaves like expected	Yes
Illegitimate actions	None
Extensions	None to test
Thinking delay (simultaneous)	Expected to be in controller
Match result vs. Umberto	Won 0/10
Other remarks	Random player so nothing fancy, but does the job

4.3 Interoperability with Player087Bomber

087Bomber is an advanced player. He avoids explosions, he has the shield and life extension and he does nothing if it cannot avoid explosions.

Tested characteristics	Grade / Remark
Behaves like expected	Yes
Illegitimate actions	Stays on same tile if thread of explosion Sometimes just walks on a box At a certain point it even ran into a wall and then out of the board (led to "Stack failure")
Extensions	Impossible to test (uses unknown import)
Thinking delay (simultaneous)	Expected to be in controller
Match result vs. Umberto	Won 3/10
Other remarks	Smarter than average, unfortunate for the illegitimate actions Does not always take points, even if it would be it's best option to do

4.4 Interoperability with Player038Luigi

Luigi is an advanced player. He avoids explosions, tries to catch bonuses/points, drops a bomb if it stands next to a box. He also has shield and life bonus and life malus extensions.

Tested characteristics	Grade / Remark
Behaves like expected	Yes
Illegitimate actions	None
Extensions	Not tested
Thinking delay (simultaneous)	Expected to be in controller
Match result vs. Umberto	Won 4/10
Other remarks	Robust player

4.5 Interoperability with Player105Alice

Alice is an intermediate player. She dodges bombs and tries to get exploded boxes.

Tested characteristics	Grade / Remark
Behaves like expected	Yes
Illegitimate actions	Stays on same tile if thread of explosion
Extensions	None to test
Thinking delay (simultaneous)	Expected to be in controller
Match result vs. Umberto	Won 3/10
Other remarks	Smart player, unfortunate for the illegitimate actions

4.6 Interoperability with Player007James

James was impossible to test because it needs the file "Util.ozf" which was not given by it's creator. It is thus not interoperable.

5 Extensions

First of all, we put a lot of energy in creating a clever player Umberto.

In addition, we implemented some little extensions, the bonus boxes can drop new bonuses, like a shield and another life. The shield has the particularity to protect a player from the next bomb explosion that he gets. The life gives a new life to the player.

The problem that we faced was disappointment, in fact we began as soon as possible to work hard on this project to be on time because we knew we would have a lot of work and projects at the end of the semester. So less than one week later we got a functional code, running turn by turn or synchronous, except no advanced player. But unfortunately we got a precision about the project that we couldn't use explicit cells. We were a bit disappointed, and we decided not to spend more time about this project that took us already no negligible time.

6 Possible improvements

We had some ideas for further improvements that we could implement to get a funnier or more robust or more clever code :

- We don't take care about the possible cheaters, we could modify the map so that it looks if the requested move is acceptable (does not go on a box or wall, does not move more than one square every turn...)
- We could create a player that calculates the danger with some scale. Instead of knowing a place is safe or not, knowing how many turns left before it will explodes and so react better. We had an idea of how to implement it. Actually Umberto remembers every square of the map with danger by incrementing its value by 100 times the delay time of a bomb explosion. So reducing the value of the square by 100 every turn could let Umberto know the magnitude of the danger. This precise computation is only possible in the game mode turn-by-turn.
- Umberto could count the points of the opponents and the number of opponents alive, so that he could commit suicide if there only stays two players and that he has the best score (with some margin).
- We could show the winner on the screen instead of showing it on the browser.

7 Conclusion

To conclude, we were able to implement all mandatory parts and to create some extensions. We hope we understood the specification well. We are really proud of our work and our advanced player Umberto. He even tops the scoreboard of the sharing platform we talked about