

## Final project

### Bomberman

#### Programming

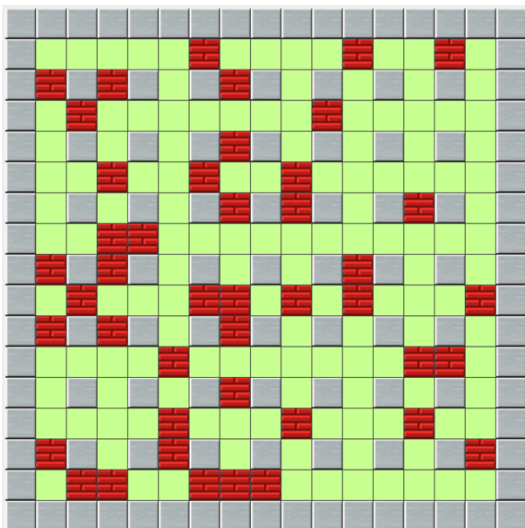
#### Bachelor's Degree in Computer Science and Engineering

2017-18

The final project consists on developing a version of Bomberman, a maze-based video game where the player uses delayed blast bombs to kill enemies and reach the exit to next level. The main purpose of the game is to reach the last level.

### 1 Basic rules

Each level of the maze is represented using a NxN cells board (the provided demo has a 17x17 board), where each cell belongs to one of the following three types:



- **Regular cells (green cells):** both player and enemies can move inside them. Notice that by default player and enemies move 1/10 of the cell each time, or in other words the player needs to press 10 times the same movement key to go from the current cell to the next one placed on the same direction as the movement key.

- **Brick cells (red cells):** they cannot be trespassed neither by the player nor by the enemies, but can be destroyed by the effect of a bomb. When destroyed, they become regular cells. After the explosion, some of them will contain bonus that the player can get. Only one of the available brick cells per level will hide the door to the next level. Each level contains X brick cells (50 in the demo game), randomly placed. Bonuses and

next level door are also randomly hidden under bricks. At most one object will be hidden per brick.

- **Wall cells (grey cells):** cells that cannot be occupied nor destroyed with a bomb. They shape the perimeter of the board and appear also in alternate positions inside it. As a result, two wall cells cannot appear next to each other.


The player uses delayed blast bombs to break through the brick walls and to kill enemies. Bombs destroy the bricks found in adjacent cells to a given distance, but do not destroy walls. If a blasting bomb hits an enemy, it will be killed and the user will get some points. Once all the enemies are killed, the player will be able to proceed to the next level through the door if it has been uncovered.



At the beginning of the game, the player is just able to use a single bomb simultaneously; another bomb cannot be placed until the current one blasts. Bonuses found under bricks will allow for more simultaneous bombs. Other bonuses imply increments on the movement speed (see section 1.2).

Player and enemies move vertically and horizontally (initially 1/10 of cell each time), but they cannot move diagonally. Speed bonuses will allow the player to move 2/10, 3/10, etc.

## 1.1 The player

The player  has the following features:

- **Score:** it will be increased any time an enemy is killed.
- **Bombs:** The simultaneous number of bombs that the player can place in the board. Every time a bomb is placed, this number will be decreased. When the bomb blows up, the counter will be increased again.
- **Health:** The current health of the player. The player loses some health when gets hit by an enemy. When a bomb hits the player instead, the health is set to zero. When the health reaches zero the game ends.
- **Maximum health:** the initial health of the player.
- **Explosion range:** The number of adjacent cells affected by a bomb explosion. Initially the range is 2; the cell where the bomb is placed and all the adjacent ones are affected by it. Bonuses increase the explosion range.
- **Speed:** The number of tenth of cells the player moves. As said, initially the speed will be 1/10 and can be increased any time a speed bonus is found, with a maximum of 10/10.

## 1.2 Bonuses

Some bricks reveal bonuses when destroyed by a blast. To get the bonus the player needs to pass through the cell containing it. The following items can be found under brick cells:



**Bomb:** Several of them can be found at every level. It increases by one the number of simultaneous bombs.



**Fire:** There is one per level. It increases the explosion range by one unit, with a maximum of 5 units.



**Special fire:** There is one every 5 levels. It increases the explosion range to its maximum (5 cells).



**Remote control:** There is one every 10 levels. It allows for the simultaneous and immediate explosion of all the bombs in the board using the "tab" key.



**Roller skate:** It can randomly appear (50% chance) in even levels. It increases the player speed by 1/10 of cell, up to a maximum of 10/10.



**Geta:** It can randomly appear (20% chance) on each level. It reduces the player speed to its minimum (1/10 of cell)

## 1.3 Enemies

Different kinds of enemies can be found in the maze. When an enemy touches the player, its health will be decreased. The list of enemies is as follows:




**Balloon:** It moves randomly with a 1/10 of cell speed. They are not too smart but it is difficult to calculate where they will be to kill them with a bomb. Each level contains a random number between 1 and 10 of balloons. Killing them increases the score by 100.



**Drop:** It tries to chase the player with a 1/10 of cell speed, going directly towards it. If an obstacle is found it stays put until the player moves to a reachable position. It first appears at the second level and another one will be added every four levels. Killing them increases the score by 250.

## 1.4 Levels

Randomly placed under a brick there will be a door  to the next level. This door will remain closed until all the enemies have been killed. When the door is opened, the player can use it to go to the next level, appearing at the same position on that new level. There is no way to come back to a previous level.

An internal counter will measure the time the player needs to reach the next level. The player will receive points depending on that time. If a given time is reached, the player will not receive those extra points.

## 2 Demo game

A demo of the Bomberman game is provided so the student can analyze it and understand the game.

The demo game provides an extended interface including a history and a console, where the following commands can be used:

- **explore:** shows the objects hidden below the bricks.
- **level<number>:** teleports the player to the desired level.
- **new:** generates a new maze.
- **god:** switches on/off the mode where the player does not receive any damage.
- **noclip:** switches on/off the collision with the walls; when off the player can traverse the walls and bricks.
- **detonate:** detonates all the placed bombs.

## 3 Work to be done

The students are required to develop the whole game, except for the graphical user interface (GUI), which is already programmed. **It is not a requirement** that the students' implementation behaves exactly as the demo game, but all the Section 1 rules must be followed.

To implement the game, **a good classes' design** is essential. Before writing any code, students must make themselves sure their classes' design is correct by speaking with their lab professor. Don't start any implementation until your class design has been approved by your professor.

Once the classes are defined, implementing the game will follow an incremental path, increasing the game functionalities step by step. The following steps are recommended (but students are free to follow their own path):

- **Sprint 1: maze and basic player**
  1. Create the maze structure, with bricks and walls. Create the different levels. Do not consider bonus objects or enemies at this step.
  2. Locate the player at the first level and make it move at 1/10 cell speed. Check collisions with bricks and walls. Do not implement movement animation, do it in the next sprint.
  3. Endow the player with its initial features: speed, number of bombs, health, maximum health, etc.

- **Sprint 2: bombs and player animation**

1. Animate the player movement by changing its sprite, choosing the right one depending on the direction it is moving.
2. Place the bombs when the space bar is pressed. They must blast after some run cycles.
3. Generate the animation for the explosions and their outcomes taking into account their reachability: they will destroy bricks but will stop if there is a wall.
4. Optional: show in the console where bombs have been placed.

- **Sprint 3: bonus**

1. Randomly place bonus elements, hidden below bricks. When a brick is destroyed, the bonus it has below, if any, will be shown. Implement the behavior of the bonuses when the player walks above them.
2. Randomly place a closed door hidden below a brick.
3. Optional: show in the console what a bonus is doing when collected.

- **Sprint 4: basic enemies and levels**

1. Place non-moving balloons in the game. Remove them when killed by bombs and reduce player's health when both are on the same cell.
2. Open the door when all the enemies are killed and generate a new level.

- **Sprint 5 (Optional): Enemies and console commands**

1. Include at least two types of enemies and make them move as specified by the rules.
2. Include new enemies with different behaviors.
3. Implement console commands as desired.

## 4 Using the GUI

The GUI provides an API (Application Programming Interface) with some methods that allow interacting with it. These methods, including their inputs and outputs, are described in a Javadoc document provided with the GUI. Using the API, basic operations on the interface, as switching the color of a cell, changing a text, removing or adding images, etc. can be performed.

Also, the interface provides the last action the player has performed by means of the `gui.gb_getLastAction()` method. The action is returned as a `String` with the following values:

- "left": the player pressed the left arrow key.
- "right": the player pressed the right arrow key.
- "up": the player pressed the up arrow key.
- "down": the player pressed the down arrow key.
- "space": the space bar was pressed.
- "tab": the player pressed the tab key.
- "new game <player name>": the N button has been pressed.
- "command <thecommand>": a command has been introduced.

## 5 Delivery rules

The following rules are **compulsory**. Not following them can result in the practical exercise **not being considered for evaluation**.

The final project must be performed in groups of two students. Each group must upload to Aula Global a zip file containing a report and the source code before **20<sup>th</sup> December 2017, 11:55 pm**. The name of the zip file

must be “bomberman-name-initials1-name-initials2.zip” (for example if the students are Lucía Pérez Gómez and Juan García Jiménez, the file will be named bomberman-lpg-jgj.zip).

- The report, in **PDF** format, will be **10** pages long as maximum. It will include at least:
  - Front page, table of contents and brief summary of the document.
  - Description of the classes’ design, including most relevant fields and methods.
  - General description of the most relevant algorithms used.
  - Description of the work performed, functionality included, parts not performed and/or extra functionalities provided.
  - Conclusions:
    - Final summary of the work performed.
    - Main problems found when implementing the game.
    - Personal comments (feedback) including critics to the final project.
- Just the src folder containing the code must be delivered. Include also any other folder needed to run the program in case extra functionalities are provided.

## 6 Evaluation

Final project will be evaluated from 0 to 10 points. To get 5 points at least the basic part, corresponding to Sprints 1 and 2 of Section 3, must be performed. To get the maximum score, at least Sprints 1 to 4 must be correctly implemented. The optional part will allow for 1 extra point, but the maximum score for the final project will be always 10.

The following items will be considered in order to mark the final project:

- Appropriate class design.
- The game runs correctly and follows the directives stated by this document.
- Quality of the implemented code.
- Exclusive use of the syllabus of the course.
- Inclusion of code comments.
- General description of the implemented code in the documentation (Section 5).
- Adherence to the delivery rules (Section 5).

**Copies will be seriously penalized:** both copied and copying groups will be excluded from continuous evaluation in addition of any other administrative measure that the University could take. Different final projects will be considered a copy even if the similarity between them is restricted to a single method. Automatic plagiarism detecting tools will be used to check plagiarism.