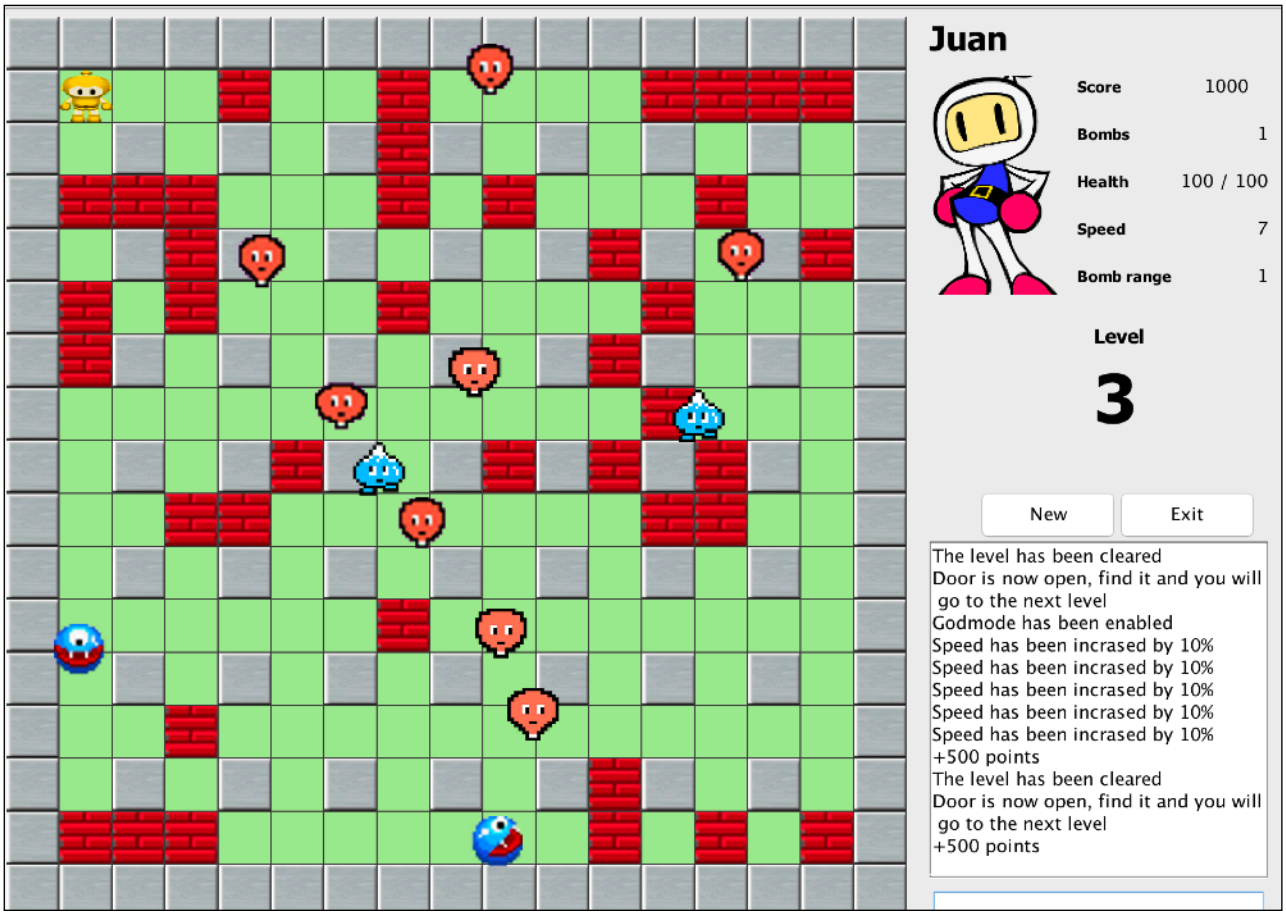
Bomberman

Luis Sánchez and Juan Sánchez

Final project - 18 December 2017



Index

1-Summary

This document explains briefly how did we code the bomberman game explaining the created classes, methods and algorithms and how they work. In addition, it also exposes which tasks have we accomplished and which ones have we added as an extra. Furthermore, it includes personal feedback, a conclusion and errors of the code

2-Main classes

For structuring the code we created the following classes:

❖ **Cell:** We created the cell class thinking about each cell the board of the GUI (the visible part for the player). Inside this class we included all the information that we thought it was important to know about a cell of the board

- **Byte id:** (for recognising whether it was a walkable block, a brick, a wall, the place where a bomb had been placed…)
- **Short RGB variables:** for the RGB scale we used a green red and blue short variables that we later erased as they are only used for "the floor" (green walkable blocks) so they are not longer in the code.
- **Boolean walkable:** as its name indicates this variable is used for knowing whether or not a block can be trespassed by the player. We could have used the cellID variable for checking if a block can be trespassed but using a boolean makes the easier to read and code.
- **Boolean item:** it contains whether there is an item or not, as we also have a variable called item id we could have just used that, but using a boolean for easily checking whether there is an item or not on a cell is rather preferable than checking all the ids because it makes the code cleaner.
- **Byte itemID:** it contains the id of the item held on the cell, be aware that we also considered the door (both closed and opened) as an item so it was easier to store all together as boosters. By default has value -1 as 0 was used for closed door.

Main methods:

- **public void setCellID(byte cellID):** Depending on the type of cell (0 wall, 1 floor, 2 brick, 3 placed bomb, 4 spawn of enemy, 5 fire, 6 player position) it will change the properties. Despite being all cells walkable by default we still need to specify that floor is walkable as we will use this method when destroying bricks. Some ids do not need any further modification as the cell they will change has the same properties than needed (i.e. enemy spawn), they instead will be used to check different things of the code (as for example if an enemy has spawned in a cell, do not spawn other in the same cell.
- **public boolean isEnemyWalkable():** it will check if it is walkable for some enemies; EnemyWalkable if cell is walkable and cellID!=3 (if a bomb is not in that cell)
- **public String getItemImage():** it returns item image according to the cell. Checking if cellID==7 (opened door) or if cellID==0 (closed door) separately is not necessary as they both have the same image but it made it easier for us to read.

❖ **Player:** The player class was created because the player will have to held many information such as its speed, health, bombs, score… So we decided that creating an outer class would be a great idea for organising the code. It has the following variables:

- **Byte speed:** It contains the speed value of the player's movement, it will take from 1 to 10 as the movement provided by gb_moveSpriteCoord can take values for a cell from 1 to 10, meaning that with 1 it will move 1/10 of a cell. By default it's set to 2. This decision will be explain while talking about enemies movement.
- **Bomb[] bomb:** It contains the information about the bombs of the player. We created this variable inside the player class instead of the main class because we though it would make more sense if ever turned the game to multiplayer; multiple players, multiple Bomb variables one for each player. Nevertheless as we code we realised it may have been better to declare it in the main class as we would only have to write bomb[] instead of player.getBomb()[] which would have made some lines shorter and by consequence bit easier to read, but we remained with our original idea.
- **Int score:** It stores player's score through the game.
- **Short x, y:** It stores player x and y position, being initial position x=15 y=19. The values are 15 and 19 instead of 5 and 9 so we don't have problems with the walls situated at the top and left of the board (setting 5 and 9 we would enter negative numbers). For x the initial value that we'll give it is 15 as the player spawns in the middle of a cell and we consider X1 to be the initial part of the cell (belonging to that cell) and X9 to be the final part of the cell (belonging to that cell). X5 will be the middle of a cell. About y the initial value is 19 as the player spawns touching the end of a cell. Setting this values made for us an easier coding but as this is not the way gb_moveSpriteCoord is thought we will subtract 5 and 9 respectively to our coordinates for moving (more about movement and coordinates will be explain through the report)
- **String image:** it stores the current image of the player.
- **Byte animationCounter:** a counter used for animations.
- **Boolean godmode:** it stores whether god mode is active or not (more about it in the command section of the report)
- **String [][] animation:** it contains all the images of the player's movement, rows of this matrix will refer to which direction is the player moving (up, down right, or left) and columns to different animations for a direction movement .

Main methods:

-public void addBombNum(): Basically it creates an array of length equal of bomb array plus one, it copies the explosion ratio of bomb[0] (all will have the same ratio and bomb[0] will always exist) and apply it for all the array. Then we make null the value of bomb and later assigned the copy array to bomb so we archive our objective.

-public void setScore(byte id): Depending on the id introduced it adds points to the score (1 will be for balloon, 2 for blue stalker, 3 for blue pac-man, 4 for getting a bonus, 5 for going to the next level)

-public void setHealth(byte enemyID): It works similarly to the previous one, depending on the id it will subtract health to the player (0 for fire, 1 for balloon, 2 for blue stalker and 3 for blue pac-man), it also makes sure health is non-negative, if it is it will set health to 0 .

-public void setRemoteControl(): It is used for commands, if remote control is true it will set it to false and vice versa. public void setGodMode() works the same way

-public void setMovement(String key, Cell[][] myBoard): This methods allows us to move the player according to the set rules. We first check that the block he will be going to is walkable by

adding to the player positions the movement times the velocity (basically the number of part of a cell he is going to move) and divided by ten (myBoard is an smaller array and by dividing by ten we check for the desired cell). If this is true we will add one to the animation counter and we would check that the animation counter is not greater than the animation[0].length minus one; if it is, it will be set to 0. Then we will add to the coordinate of the movement 1 times the speed, and set image to animation[the corresponding number according to the movement coordinate][animation counter].

-public String getDieAnimation(): Using the animation counter it will return a certain image from player dying animation, we could have used an array String to store this images and use the animation counter to return a certain image but as they only were 4 images we decided not to do it.

- **Bomb:** This class is focused on the bomb information. It has the following variables:
  - **Short x, y:** it stores bomb x and y coordinates. They will be player's coordinates in the moment of planting the bomb divided by 10 so they can be easily used in relations to the cells.
  - **Byte radio :** it contains the explosion ratio.
  - **Byte animationCounter:** a counter used for animations.
  - **-String[] placementAnimation:** contains placement animation images
  - **-String[][][] blowingAnimation:** contains blowing animation images for ratio 1.
  - **-String[][] animationBiggerRatio:** contains the extra images needed for the animation of explosions greater than 1.
  - **Boolean available :** it says whether a bomb is available or not
  - **Int timer :** is the timer used for the explosion.

Main methods:

-public String getPlacementAnimation(): using the animation counter it returns one of the two images, if the counter is equal to one it sets it to zero, otherwise it adds one to the counter

-public String getBlowingAnimation(int rows, int columns): depending on the explosion ratio it makes use of blowingAnimation and (if ratio is greater than one) animationBiggerRatio returning the corresponding image according to rows and columns (its values will go from minus ratio to ratio)

- **Enemy:** It is an abstract class as it will have different classes deviated from this one inheriting all it's variables. We have used protected instead of private as then on each enemy class this variables would be easier to manipulate. This are its variables:
  - **Short x, y:** it will follow same rules as player x and y.
  - **Byte counter:** will be used for enemies movement and animation (further explanation on how it works on each enemy movement method explanation)
  - **Byte movement:** used for enemies movement (further explanation on how it works on each enemy movement method explanation)
  - **String image:** it stores the current image of the enemy
  - **String [][] animation:** it contains all the images of the enemy's movement.

Main methods:

-public abstract void movement(Cell myBoard[][]): It will be used by each enemy, each enemy having each movement code. It will allow for an easier coding on the main.

❖ **Balloon:** This class extends Enemy class, so it inherits all its fields. If a balloon is created we start by changing the animation array to the desired one with the corresponding image. As no further information about the images was included we supposed images "enemy111.png","enemy112.png" and "enemy113.png" for going left and up and images "enemy121.png","enemy122.png" and "enemy123.png" for going right and down and so we use them.

Note: Balloon will move randomly but in the same direction for some time so it actually leaves his spawn cell. Otherwise most of them would never leave their surrounding cells.

Main methods:

-public void movement(Cell myBoard[][]): If the counter is 0 or 8 movement variable will be set for a random value between 1 and 4, depending on this value the enemy will try to move for a direction or for another. If it can move into that direction (enemy walkable) one will be added to the counter, it will move one on this direction and a certain image from animation will be assigned to image, animation[the corresponding number according to the movement][counter/3]. Counter is divided by 3 so it's value is always 0, 1 or 2 (length of animation[the corresponding number according to the movement] minus one). It will keep going to that direction and going inside that part of the code till the counter reaches 8 or it can't move, in that last case counter will be set to 0 so movement value will be changed on the next cycle.

❖ **Blue Stalker:** This class extends Enemy class, so it inherits all its fields. If a Blue Stalker is created we start by changing the animation array to the desired one with the corresponding images. As no further information about the images was included we supposed images "enemy211.png","enemy212.png" and "enemy213.png" for going left and up and images "enemy221.png","enemy222.png" and "enemy223.png" for going right and down and so we use them.

Note: Blue Starker will move according to the player position, it will try to approach player coordinates but if a wall, brick or bomb is on his way he will remain there until the player moves to another position. It is not really cleaver but can be pretty annoying, its damage is greater than balloon one.

Main methods:

-public void movement(Cell myBoard[][]): First it will check that it is not on a cellID 6 (more about cellID 6 and player coordinates later) , if so it will try to find the player (will find it always unless player is on a bomb, more about this condition later) and approach his coordinates to the player ones by checking whether his coordinates are greater or smaller and whether the cell it is trying to go is enemy walkable or not. In addition it will reset the cellID with the player position to 1 so there are not multiple player positions

❖ **Blue Pac-Man:** This class extends Enemy class, so it inherits all its fields. If a Blue Stalker is created we start by changing the animation array to the desired one with the corresponding images.

Note: Blue Pac-Man is an added enemy, we have provided along with the code and report his images. This enemy idea has come from Super Bomberman 5 game and so have came its images. This enemy moves into one direction till it touches a wall or brick where it is able to change its direction, it can only move in the middle parts of a cell. It can eat bombs so the player will have to carefully calculate where to place the bomb so it explodes at the exact moment. The enemy should be close enough but not in the bomb cell before blowing or the bomb will disappear and will not explode.

Main methods:

-public void movement(Cell myBoard[][]): When the counter is -1 it will set his movement variable to a random number between one and four, then if the counter is -1 or 3 it will be set for 0 (animation reasons). Later it will check its movement number and according to it and whether the block he wants to walkable (notice not enemy walkable) or not and add one to its coordinates or not, then image will be set to animation[the corresponding number according to the movement] [counter], and we will add to the counter, if it has not be able to move or movement was equal to 0 (at the beginning) the counter will be set to 0 so next cycle the movement variable will get a new value.

❖ **MainProgram:** In this class is where everything happens and works. We have declare some public static variables as we will be using them through different static methods in this class. The most important ones are:

- GameBoardGUI board: it is given and is how we will access the graphical interface

- Cell[][][] myBoard: it contains the the number of levels (1 dimension) plus all the information of cells of the board (2 dimensions).

- Enemy[][] enemies: each row of this array is the enemy type (0 balloon, 1 Blue Stalker, 2 Blue Pac Man) and the number of columns the number of enemies (each row may vary)

We start the main method setting exitGame to false that will be set to true if the player wants to exit the game, while he/she does not we will clear all sprites and establish the scoreboard. Then while the player is alive we will do a for loop for each level the player goes through. First we set the walls, then the floor and then the bricks, secondly we set bonuses (see setbonuses method for further information) and enemies (see enemies method for further information) followed by printing the board and setting player coordinates and visible. After this we enter a while loop with conditions player alive and door closed, inside this loop we will check player health each time we enter and add one to the player timer that will measure the time we spend in a level (if we spend too much time we won't be given any points for leaving a level). Then it will check if an item is on the floor and print its image if thats the case, it will also check if the door is closed by using the boost method (see boost method for further information). After this if god mode (see commands for further information) not active it will check for damage, then it will move enemies using moveEnemies method (see moveEnemies method for further information) and enter a switch depending on the user behaviour.

Inside this switch if the user:

- Press movement key (up, down, right or left key): it will invoke the player movement method and send it the pressed key and myBoard[current level] (previously explained) and then will set the player image along with it's coordinates
- Press space key: it will check there are available bombs and that a bomb has not been place in the same place (cellID 3) and will place the bomb there and set the bomb not available
- Press tab: If player has remote control active and a bomb placed it will set its timer to 70 what will make the bomb detonate (see bomb method for further information)
- Else: if the user has introduced a command (even if it is incorrect) it will go to the command method (see for further information), and if the user has pressed exit game it will set alive to false (so it leaves the levels loop) and exitGame true so it exits the game

After the switch we have set a Three.sleep(); so it goes slower than what could really go (otherwise enemies would instantly move and so would the player). It will keep in this loop until the user finds and crosses the door or dies.

Outside this loop if the player is not alive and has not chosen to exit the game we will perform player dying animation using player.getDieAnimation (previously explained), then we will hide the player and print game over. Else if the player is alive (it will mean it has found the door) if the player's timer is smaller than 1800 (5 minutes) he/she will get a +500 points bonus.

After this, outside the loop alive and level<15 if the player hasn't chosen to leave it will be keep in a loop till he or she chooses to either exit or start a new game.

Outside this, and closing the main method there is System.exit(0) that will close the interface

Main methods:

-public static void setBonuses(Cell myBoard[][]): First it creates an array of bonuses (length 7, bonuses + door) and sets the value for each bonus (depending on the level, randomness and item). Then they are distribute through the board making sure no other item is already on that cell and that the cell is a brick

-public static void bomb(int ii): (notice ii is the bomb number in the array) If the remote control is not active it will add one to the timer each time it goes through the method. Then it will set cellID of the place where the bomb is planted to 3, and while the bomb timer is smaller than 70 it will perform the placement animation, else if bomb timer is equal to 70 the animation counter will be set to 0 and if remote control is active it will add one to the bomb timer (otherwise with remote control the bomb would not leave that part of the code as timer is not running). If the timer is in between 70 and 90 it will add one to the counter if remote control is active (non remote control already added one to the counter) and it will set and animate the fire if bomb timer smaller than 90 and destroy the fire and reseting the cellID to 1 if bomb timer equals 90. When the timer is greater than 90 we will make the bomb available and set both counters to 0.

Note: In this code bombs are treated as images rather than sprites so sometimes the image of the bomb is on top of the player. As we aren't given any method to change that and we have been told leaving it like that is fine by the professor so we did; despite it we have achieved that most of the time player is printed over the bomb.

Note: The bombs coordinates are printed positive and taking the origin at the top left corner, being 1 1 the spawn and 1 2 the cell under the spawn cell.

-public static void enemies(): set the number of enemies according to the level number and randomness, int id is used for not losing track of the sprite id number, it starts by one as 0 is the player. Then it goes through the array setting the enemies on a cell and changing its id to 4 so we know an enemy has already been placed there

-public static void moveEnemies(): This method is used for both checking there are enemies alive and to move the remaining enemies. If we are checking for Blue Stalkers if the player is not over a bomb we will transform the cell the player is on to cellID=6 so the Blue Stalker can know where is the player and track it. We don't change it always because it will change cells with id 3 (place bombs) and some mechanics as Blue Pac-Man eating bombs wouldn't work. If checking for Blue Pac-Mans we will check it is not on a cell with a bomb and if it is we will set its timer to 91 so it disappears without blowing. Then we check an enemy is not over fire and if it is we kill it and add score to the player. Also if there aren't any enemies alive we will change itemID 0 to 7 (from closed door to opened) and display a message letting the player know.

Note: Player default speed is set to 2 as otherwise enemies would move much faster (enemies don't depend on gb_getLastAction()). Setting default player speed to two not only allows the player to go almost at the same speed than enemies do but also makes playing much more enjoyable.

-public static void scoreboard(): This method sets the scoreboard each time we go check the player is alive and the door has not be found

-public static void damage(): Checks player is not over fire neither an enemy and using player.setHealth(id) method subtracts health if necessary.

-public static boolean boost(): Checks if the player is over a boost if so it erases the boost image, sets cell item to false and applies the effect to the player, if the boost is not itemID 7 returns true (it will be used for checking if door is opened and player is over it so it can be sent to the next level) else (any other booster) false.

-public static void command(String key): Checks for the introduced command and if non of them matches prints a help sentence for the user to know what to do or how to access the commands. Else depending on what the user has introduced it will do one thing or another. Here I will explain the less obvious ones:

- /godmode: Player can't die, it gets on and off with this command
- /remotecontrol: remote control gets on or off with this command
- /destroy: destroys all the bricks
- /show: shows boosters
- /kill: kills all enemies
- /clear: removes enemies and bricks for the level
- /resetstats: resets all player variables to default
- /?: it displays all commands that the user can use, a message suggesting the player to use /? Is displayed if an unknown command is introduced

3-Work performed

This bomberman follows all the rules and parts suggested on the paper (including the optional ones) adding an new enemy with new textures and behaviour.

4-Conclusions

Finally summary: With some difficulties we have came up with a really good bomber man able to move, get boosters, go through levels by crossing the door and being able to freely use plenty of commands so we make the player experience better.

Main problems: During the development we had many problems, the first major one came while implementing the movement, it took us many time to understand moveSpriteCoord and how it worked. Then for the bomb animation and some times the bomb was animated and some other times it wasn't. In the end it was due to an error that make the counter not always add one. Also with the bomb getting the proper animation when the ratio was greater than one took us many tries until we finally made it work disregarding the explosion ratio. Implementing movement for all the enemies was really hard specially for the Blue Pac-Man as it had to be in the middle parts of the cell, also getting the skins and making them work took longer than expected.

Feedback: I believe after all the effort we have ended up we a really good bomberman with commands, an extra enemy and really fun to play. On the other hand we still get some errors that we haven't figurate out, still everything seems to be working

```
ERROR: A file in the images folder could not be loaded as an image:
images/.DS_Store
```

In addition sometimes (most of the times) we get this error while running the game but if checking with debug mode step by step it won't happen

```
ERROR using gb_setSpriteImage: Image null could not be loaded correctly, or
sprite id does not exist.
```

Also sometimes the game is a bit laggy, if picking up extra bomb boosters if a bomb is placed will make the placed bomb to don't do anything but the image still remains and new and exit buttons have some time failed but we believe it to be fixed.

Setting the portrait image (given) and using the given command the figure gets cut up and down.

I think that somethings of the GUI should be better explained, how to exit the interface wasn't explained anywhere (at least I didn't find it), also I believe the moveSpriteCoord works really weirdly, the centre of cell 1 1, at least from my point of view, should be 10 10. Also the professor should check somehow that both partners work or it may happen that one works really hard and the other one no so much and lastly I think creating a better moveSprite method would be really nice for those they want a really nice game as the movement using moveSpriteCoord doesn't feel good.