BROGLE Boris, GERMAIN Clément, PONCET Yann, PONTHON Rodolphe

LP24: Project report:

Lemmings

Table of contents

BROGLE Boris, GERMAIN Clément, PONCET Yann, PONTHON Rodolphe

Introduction

In the context of the UV LP24 at the UTBM, we had to develop a Java game.

The goal of this project was to provide a java version of the Lemmings, a 2-dimensional puzzle-platformer video game, but with the use of Students instead of Lemmings.

I. Developed features

1) Skills

We chose to implement 6 skills: the Basher, the Blocker, the Climber, the Digger, the Exploder and the Floater.

Like in the original Lemmings game, those skills can be applied zero or several times, depending of the level.

We tried to implement the behaviour of each skills like in the original game. Furthermore, each skills have a different graphical sprite so it is easy to identify them.

2) Levels

The Levels are divided in two main categories:

First, we have the campaign levels, which are non-customizable, and have a rising difficulty. We chose to develop 8 campaign levels, with multiple block textures. Every campaign levels uses different skills combination.

Then, we have the custom levels, which can be created and modified by the user by using the Level Editor (see below). 16 slots are reserved to them.

3) In Game settings

Several settings can be modified while playing.

First, we can change the spawn rate with two buttons (+ and -), there is a maximum and a minimum value for that.

We can also increase the speed of the game (by a factor of 4) by pressing the >> button.

Then, we have the “kill all” button, which applies the skill exploder to all the Students currently on the map.

Finally, we can put the game in pause, restart the level or return to the menu with 3 additional buttons.

We also print all the useful information to the user.

4) Menu

The main menu is showing the different levels of the game, there are separated between the campaign and the custom levels. Every level is represented by a thumbnail.

5) Level Editor

We wanted to implement a level editor because once the user finishes the campaign mode, he generally never (or rarely) come back on the game. We wanted him to have more imagination, more fun and think about levels that could be instantly playable.

It was also useful to do our tests and to design the campaign map, because we finished the level editor before having all the campaign worlds ready.

In the level editor, we can chose all the parameters of a level and design a map. Every custom levels can be edited or deleted at any moment.

II. UML diagram

The complete UML diagram of the project can be found in *annex*.

III. Decisions taken

1) Students and skills

To implement the different skills, we first did an abstract *Student* class in order to implement the commons methods to all the students, like the general *move()* and the *changeClass()* method.

In the *move()* method, we chose to make our students “teleport” from one block to another

(currently 50px wide), and not moving continuously.

The *changeClass()* method was done by creating a new student with the same attributes as the current student and telling the world to remove the old one and adding the new one.

Each skill is then implemented by its own class, with its own attributes and move() method if needed.

We did not chose to implement an abstract *MovingObject* or others classes above the *Student* class because it seemed useless when we conceived our UML diagram. But for future updates in the game, it would have been better.

2) Blocks and level

The map is (currently) composed of 24 blocks wide and 16 blocks height, every block has its own type with its own texture:

- 9 for the exit
- 8 for the start of the level
- 4 for an unbreakable block
- 3 for a complete breakable block
- 2 for an 1/3 destroyed block
- 1 for a 2/3 destroyed block
- 0 for air

A block can also be blocked or not. The blocked state is currently used by the blocker, once it is blocked, no students can come in the block from the left or the right side.

The levels are stored in a .txt file with first the useful informations like the time, the number of student to save or to spawn, the skills of the level… and then, the blocks of the map organised like a 2 dimensional table, each one represented by its blockType.

3) World and game while playing

The *World* class is here to store the students in a *CopyOnWriteArrayList* and the level. It also contains all the necessary attributes used in the game.

This class has a run() method which launches the game. This method is composed by a *do while* statement: every time we pass in that loop, we apply the *move()* method to the students and inform the GUI that the game has changed using the observer.

4) GUI

For simplicity reasons, we chose to send directly the *World* object to the GUI in order to have full access to all of its attributes and methods. We know that it is not a true MVC pattern (we shall have used string arrayList), but we saw that late in the TP sessions and it was way simpler by passing the world.

To manage the blocks graphically, we chose to directly print the images corresponding to the textures on the screen. We also chose to do our buttons the same way (see *encountered difficulties*).

In order to apply the skills on the students or to detect when the user clicked on a button, we chose to implement the *MouseListener* interface to the board. Then, using the *mousePressed* method, we catch the coordinates of the click and call the corresponding methods of the world. By doing that, we don't use a *Controller*: the interactions with the world are directly done in the *View*.

5) Level Editor

We wanted the level editor to be able to create a fully customizable level, to insure the freedom of the user imagination. That's why in the level editor, each parameters can be changed from the number of spells to the textures of the level.

However, we limited the user to one spawn and exit, and putted limits to the number of skills (99) and other parameters.

The level editor itself works the same way as the GUI of the game, except that it reads and save the .txt files.

6) Ressources organisation

We organised our resources by using 7 different folders:

- *backgrounds* : it stores each background of each levels (including customized levels)
- *blocks* : it stores each textures of blocks of each campaign levels, and we gave the choice to the user between 4 other textures (already used in campaign levels) that are in four other folders.
- *levels* : it stores text levels files that contains the data of a level
- *messages* : it stores messages that the game prints to the user (for example the ending screen of a level)

- *student_sprites* : it stores the sprites of students ; their power's sprites, their different "walk state"
- *thumbnails* : it stores the icons printed in the main menu of each levels (campaign and custom)
- *user_interace* : it stores all the buttons, selectors of the graphical interface which allows the user to interact with the game

IV. Encountered difficulties

We had no really big difficulties during the project but we had some difficulties to start the graphical part because we had no TP sessions dealing with it behind us when we started this part.

It was the same when we had to deal with the buttons, we started to use images instead of button object but we didn't had the time to change them after we saw it in the last two lessons.

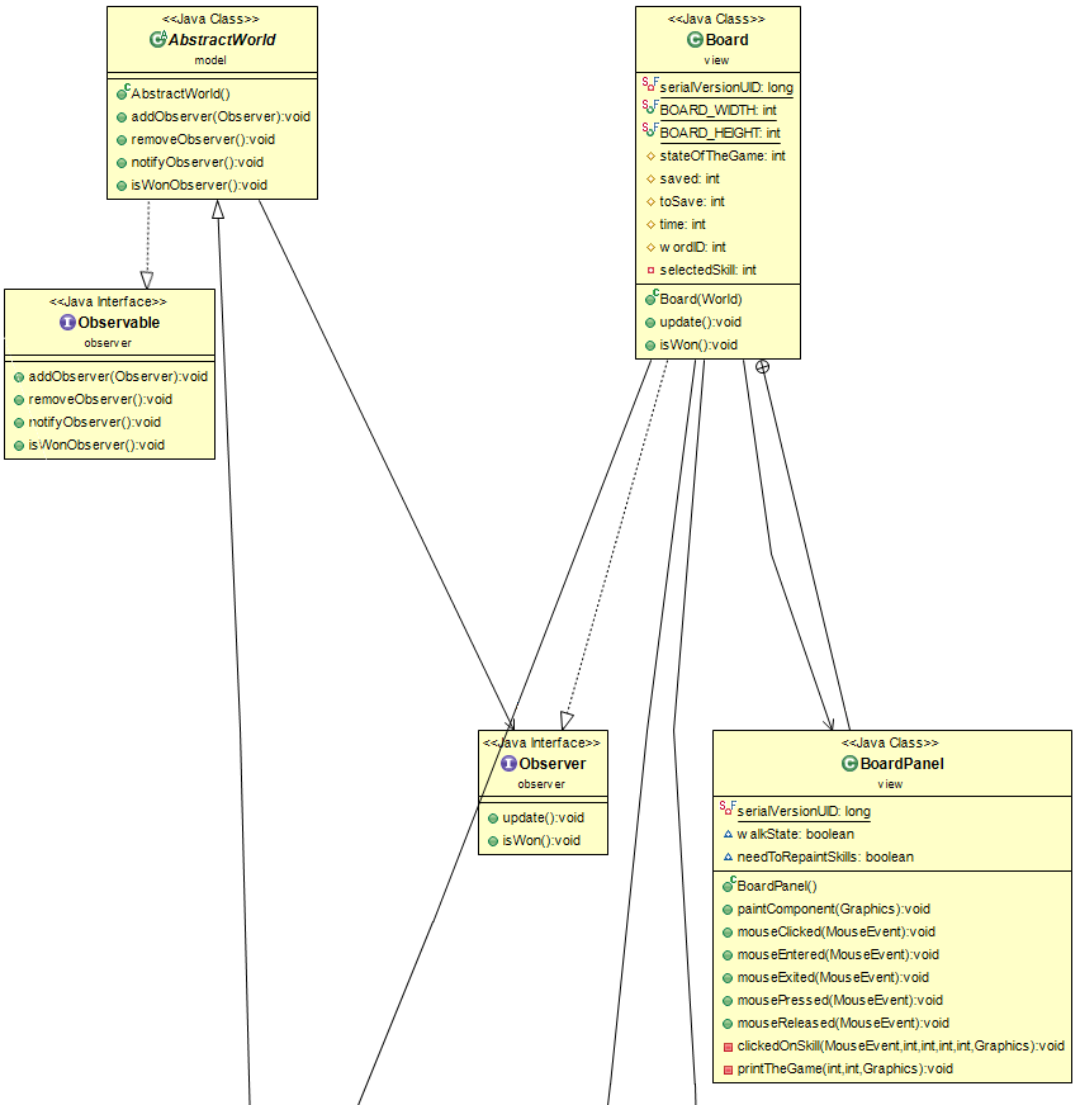
V. Conclusion

This project has enabled us to work in group and to train us with the object oriented programming. We worked regularly on it and helped us each other. It was also quite interesting to see the game working with a graphical interface.

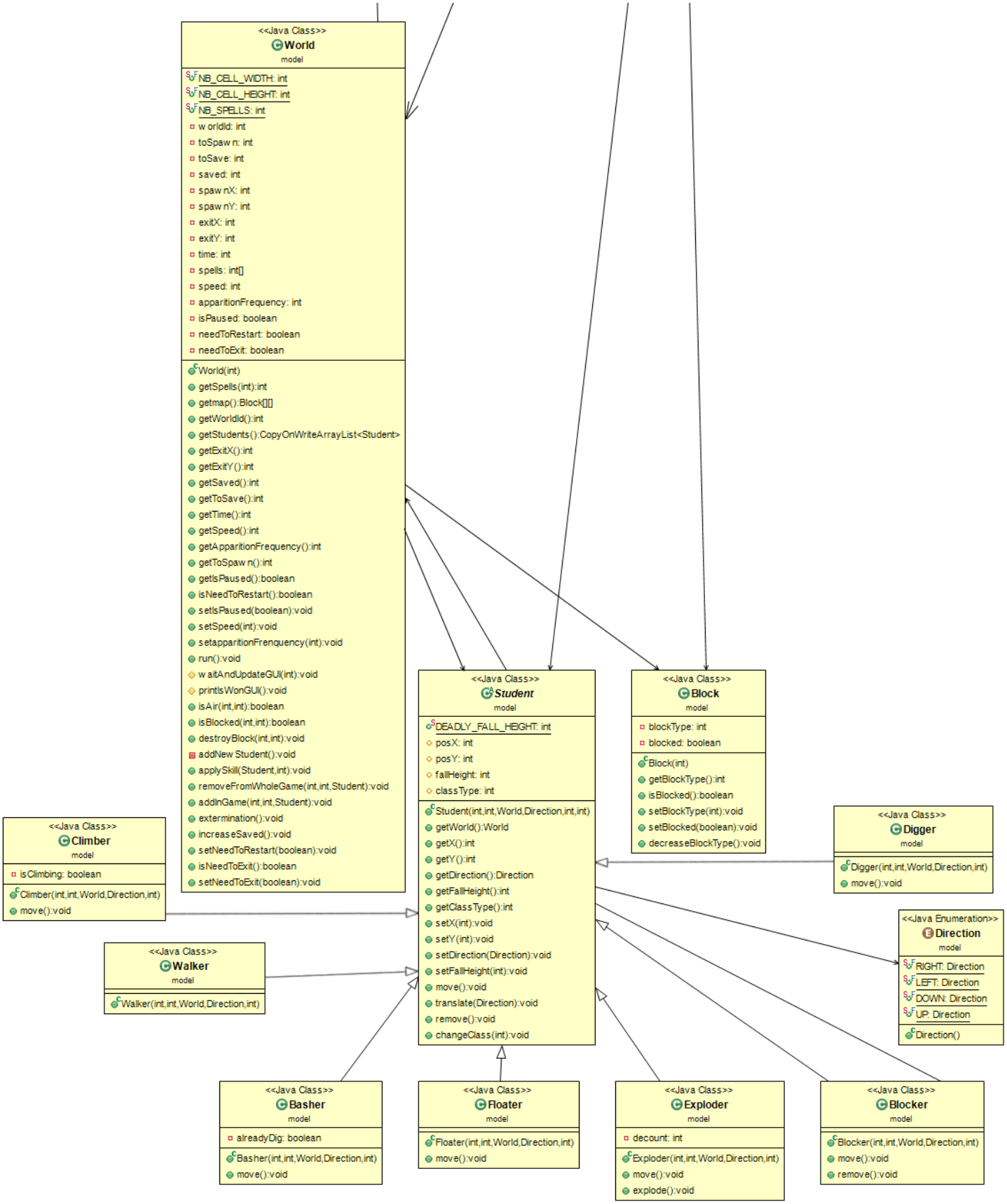
But we didn't had time to do all what we wanted: separate the controller part, do more skills, do a better graphical part, add different types of blocks like moving ones, traps or even blocks that can be controlled by the user...

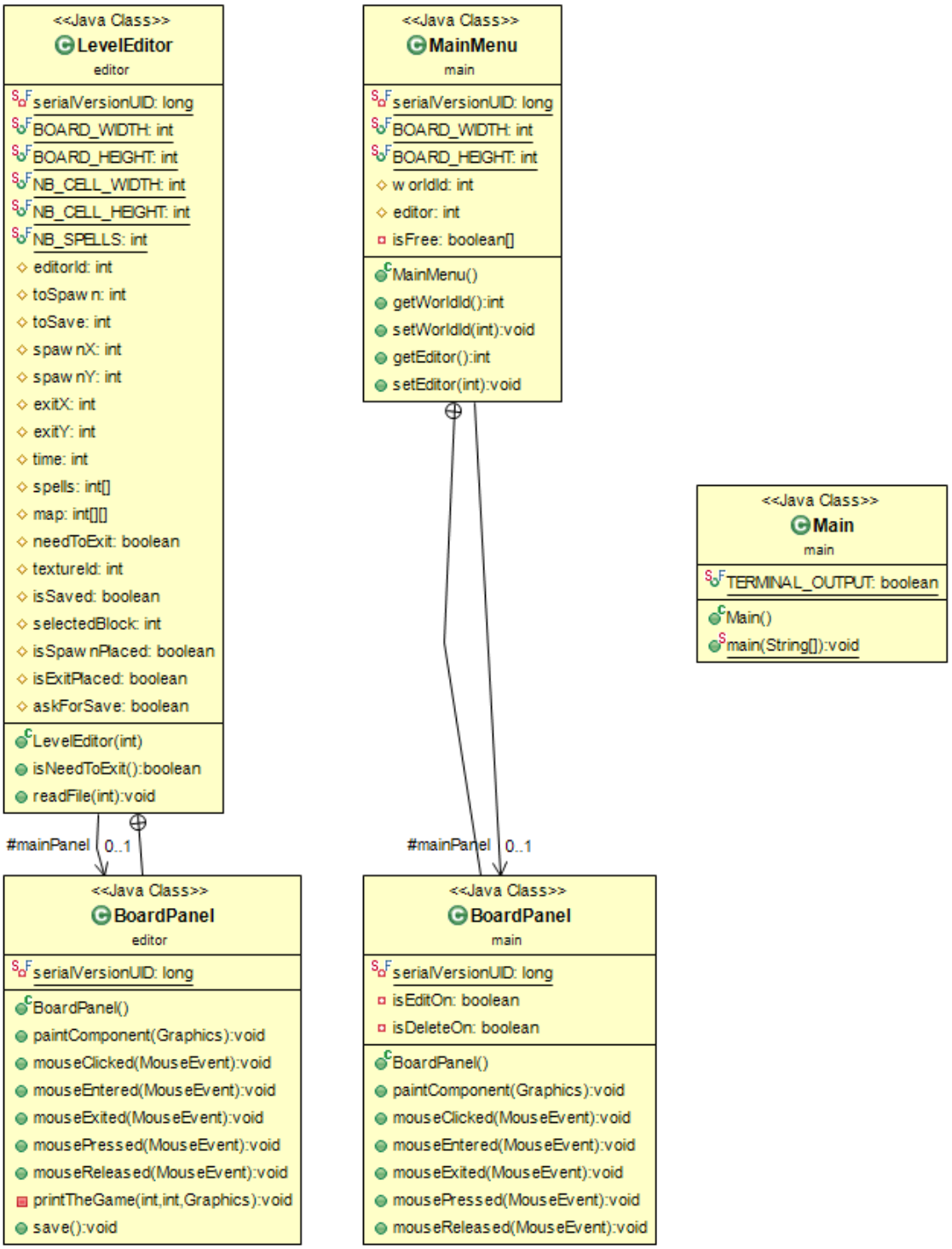
We also tried to make our code as optimized as we could, but there is certainly a better approach for some parts of our code.

Annexes


<<Java Class>>
AbstractWorld
model
AbstractWorld()
addObserver(Observer):void
removeObserver():void
notifyObserver():void
isWonObserver():void
<<Java Class>>
Board
view
serialVersionUID: long
BOARD_WIDTH: int
BOARD_HEIGHT: int
stateOfTheGame: int
saved: int
toSave: int
time: int
wordID: int
selectedSkill: int
Board(World)
update():void
isWon():void
<<Java Interface>>
Observable
observer
addObserver(Observer):void
removeObserver():void
notifyObserver():void
isWonObserver():void
<<Java Interface>>
Observer
observer
update():void
isWon():void
<<Java Class>>
BoardPanel
view
serialVersionUID: long
walkState: boolean
needToRepaintSkills: boolean
BoardPanel()
paintComponent(Graphics):void
mouseClicked(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
clickedOnSkill(MouseEvent,int,int,int,int,Graphics):void
printTheGame(int,int,Graphics):void

BROGLE Boris, GERMAIN Clément, PONCET Yann, PONTHON Rodolphe




<<Java Class>>
LevelEditor
editor
serialVersionUID: long
BOARD_WIDTH: int
BOARD_HEIGHT: int
NB_CELL_WIDTH: int
NB_CELL_HEIGHT: int
NB_SPELLS: int
editorId: int
toSpawn: int
toSave: int
spawnX: int
spawnY: int
exitX: int
exitY: int
time: int
spells: int[]
map: int[][]
needToExit: boolean
textureId: int
isSaved: boolean
selectedBlock: int
isSpawnPlaced: boolean
isExitPlaced: boolean
askForSave: boolean
LevelEditor(int)
isNeedToExit():boolean
readFile(int):void
<<Java Class>>
MainMenu
main
serialVersionUID: long
BOARD_WIDTH: int
BOARD_HEIGHT: int
worldId: int
editor: int
isFree: boolean[]
MainMenu()
getWorldId():int
setWorldId(int):void
getEditor():int
setEditor(int):void
<<Java Class>>
Main
main
TERMINAL_OUTPUT: boolean
Main()
main(String[]):void
#mainPanel
0..1
#mainPanel
0..1
<<Java Class>>
BoardPanel
editor
serialVersionUID: long
BoardPanel()
paintComponent(Graphics):void
mouseClicked(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
printTheGame(int,int,Graphics):void
save():void
<<Java Class>>
BoardPanel
main
serialVersionUID: long
isEditOn: boolean
isDeleteOn: boolean
BoardPanel()
paintComponent(Graphics):void
mouseClicked(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void