CSC 413 Project Documentation
Fall 2020

Pramod Khatri

920831584

CSC 413-02

Tank Game: https://github.com/csc413-02-FA2020/csc413-tankgame-pramodkhatri10

Lazarus: https://github.com/csc413-02-FA2020/csc413-secondgame-pramodkhatri10

Table of Contents

1 Introduction

For the term project, we were assigned to develop two 2D games in JAVA. The first game was Tank Game which was non-negotiable, and we all had to do it. It was built from the scratch even though resources were given by the professor Anthony Souza. I chose Lazarus for the second game form the given options. Lazarus game has similarities from the tank game, and I have reused code from the tank game in Lazarus.

1.1 Project Overview

The objective of these two games development for the term project was to practice and apply Object Oriented Principle (OOP) as well as practice the reusability of the code. The idea is to develop the first game: Tank Game from the scratch and then depending on similarities the second game has with the Tank game, reusing good portion of code from the tank game to build second game.

1.2 Summary of Work Completed: Tank Game

Tank game is a 2D game which will have two players. Those two players can move left-right-back-forth. There is welcome screen that has three options: PLAY GAME, INSTRUCTIONS and QUIT GAME. Keys control are given on the instructions. Once the player starts moving tank and when they encounter or throw bullet at each other there will be explosion. There are two walls: breakable and unbreakable wall. Breakable wall breaks when bullet hits them but unbreakable wall which are the boundary of the game board do not break even with bullets explosion. Each tank has 3 lives and lives decrees if the tank gets hit with bullets from the other tank. When one tank runs out of its lives then the other tank will win the game and the winner is displayed in the screen saying the game is over.

1.3 Summary of Work Completed: Lazarus

Lazarus is another 2D game implemented for the term project. It is a one player game which is basically Lazarus that has been abducted blob Mob and the goal is to help Lazarus escaped from the abduction. Lazarus can move left and right and can jump one box, but different boxes; wood, stone, cardboard and metal, will keep coming from above. If the heavier box falls on the top of lighter box, then the lighter box disappears. Lazarus will have to build stairs from those falling boxes and try to reach stop sign which will help Lazarus to go to next level and then eventually escape. Lazarus has three lives and each time it gets hit by falling boxes, its lives decreases by one.

2 Background/Given Resources

For the first Tank game, we were given several resources. To start with, we were given the source code for tank rotation as well as several images and sound for the game. Also, for the second game, there was detail what really, we were building and the hints on how we could make the game board, use boxes, Lazarus. Sound and images were provided for the

Lazarus game as well. There were useful discussions in the slack channel as well where professor answered classmates' questions which were helpful in building games. Moreover, there were no restriction but clear guidelines to make this difficult task easier for us to build the game.

3 Development Environment

a. Version of Java Used: OpenJDK 14.0.2
b. IDE Used: IntelliJ IDEA Ultimate Version 2020.2.1

4 How to Build/Import Project

After cloning the GitHub repository in our computer folder.

1. Open IntelliJ IDEA
2. Select open or import icon
3. Navigate to the repository folder's name where we cloned the GitHub repository and Select the repository folder as the source root of the project
4. Click OK
5. After successful import of the project, we should be able to build project.

5 How to Run Project from JAR file

After jar file is created:

1st Way:
Either simply right click on jar file and do "run". OR go to terminal and go to jar directory of the project and do "java -jar csc413-tankgame-pramodkhatri10-master.jar".

2nd Way:
For both Tank Game and Lazarus, the driver class in "Gameworld". We can simply go to the driver class and run from the IntelliJ.

6 Assumption Made

For both the game, the details set of the professor are assumed to be implemented well in the games.

Briefly, for the Tank game: the tank will move in four direction: left, right, up and down. It has three lives, and those lives reduce after bullet explosion on the tank. Also, the breakable wall is destroyed once it gets hit by the bullet whereas the unbreakable wall does not.

For the Lazarus, Lazarus has three lives and two levels to complete in order to escape. It has to stay away from boxes and make stairs to get to stop point. It can only move left and right and jump boxes.

7 Game Key Controls

8.1 Tank Game

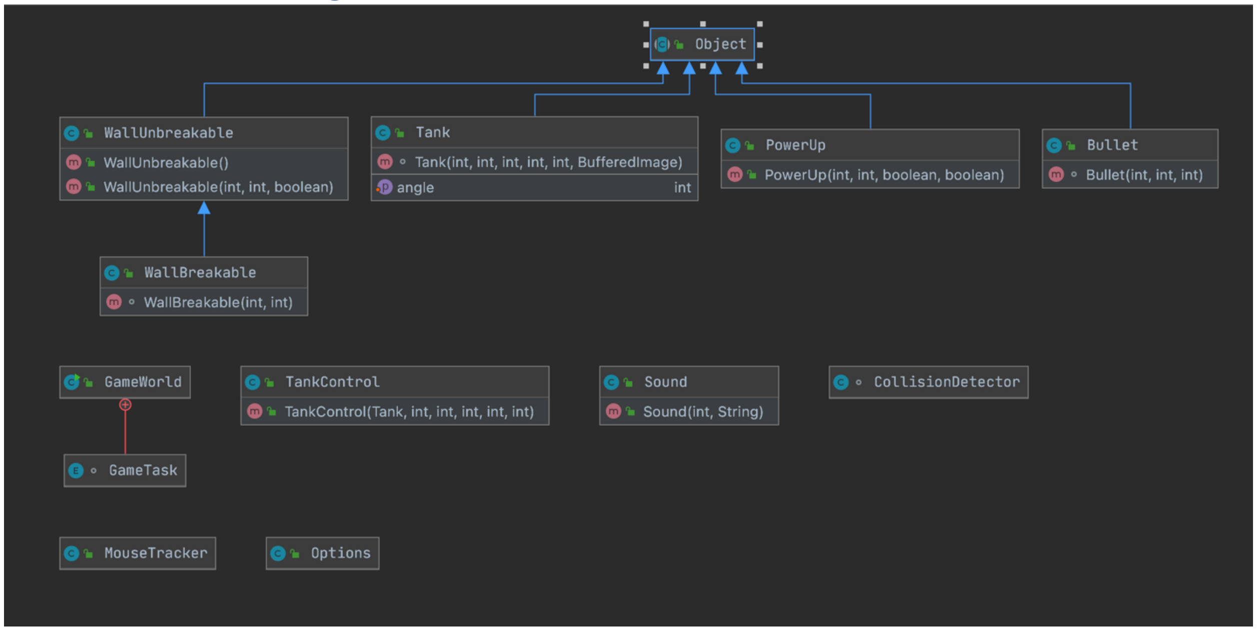
Movement	**Player 1**	**Player 2**
Forward	W	Up Arrow
Backward	S	Down Arrow
Rotate Left	A	Left Arrow
Rotate Right	D	Right Arrow
Shoot	Q	Enter

8.2 Lazarus Game

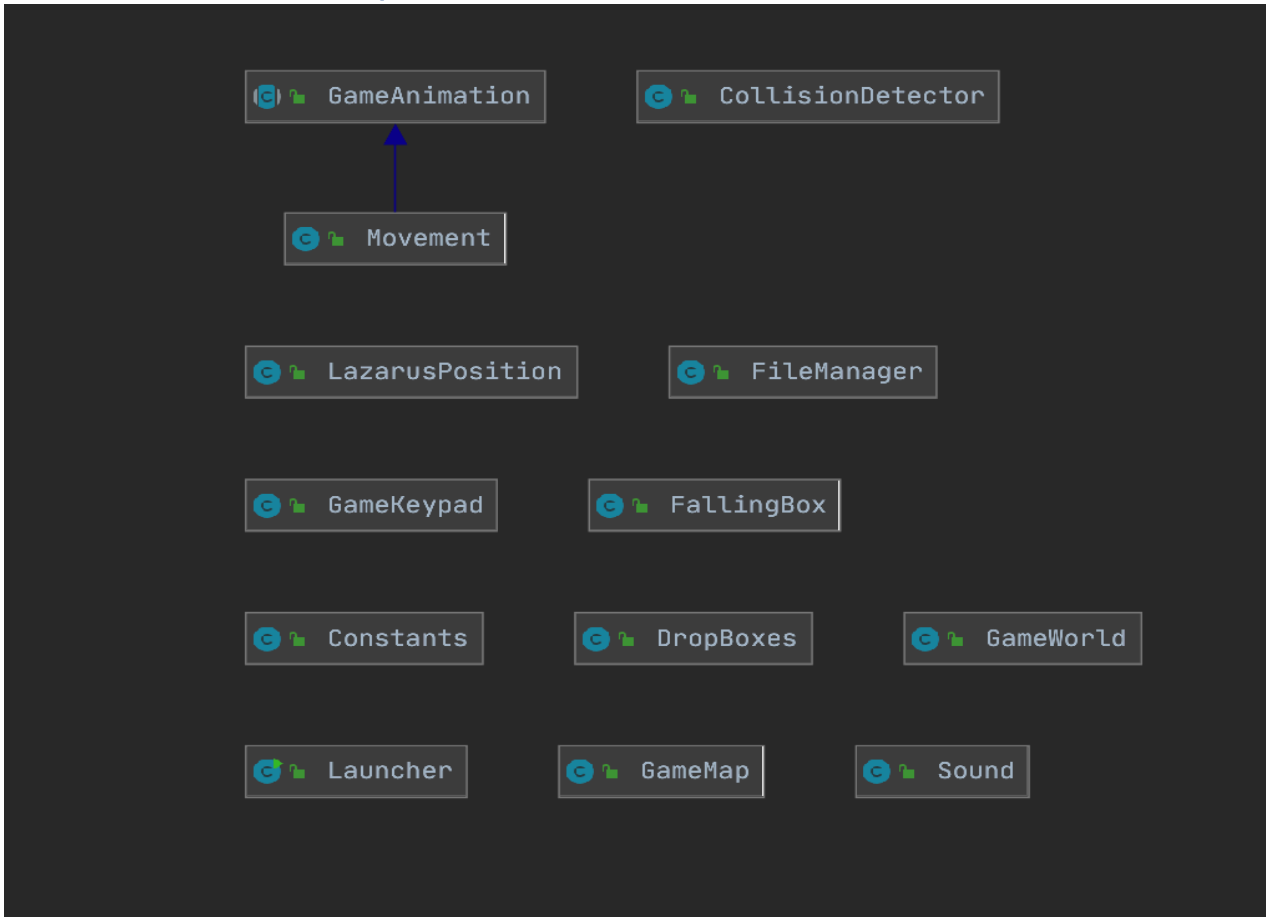
Use **ENTER** key to start the game and **ESCAPE** key to quit the game when it's over and **SPACE** key to exit game when you win

Movement	**Lazarus Movement Key**
Left	Left Arrow
Right	Right Arrow

8 Tank Class Diagram

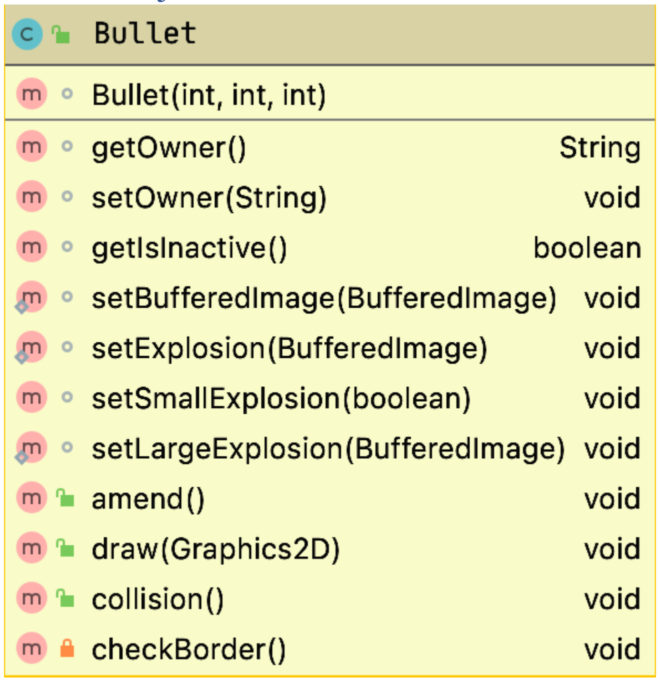


9 Lazarus Class Diagram



10 Implementation Details of Tank Game Classes Package: game

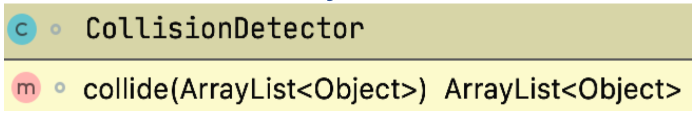
10.1 Bullet.java



Bullet	
Bullet(int, int, int)	
getOwner()	String
setOwner(String)	void
getIsInactive()	boolean
setBufferedImage(BufferedImage)	void
setExplosion(BufferedImage)	void
setSmallExplosion(boolean)	void
setLargeExplosion(BufferedImage)	void
amend()	void
draw(Graphics2D)	void
collision()	void
checkBorder()	void

Bullet.java class extends the Object class and implements the functionality of the abstract class. It works on representing the colliding bullets to tanks.

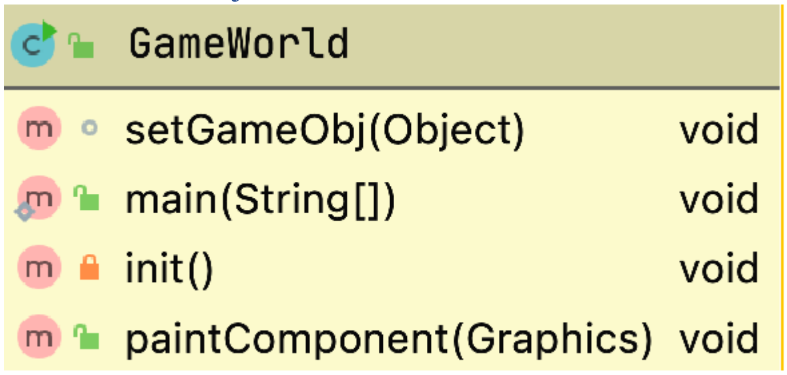
10.2 CollisionDetector.java



CollisionDetector	
collide(ArrayList<Object>)	ArrayList<Object>

CollisionDetector.java is responsible to determine and validate the collision of bullets with tanks and walls, and also determines how the lives is affected by the collision.

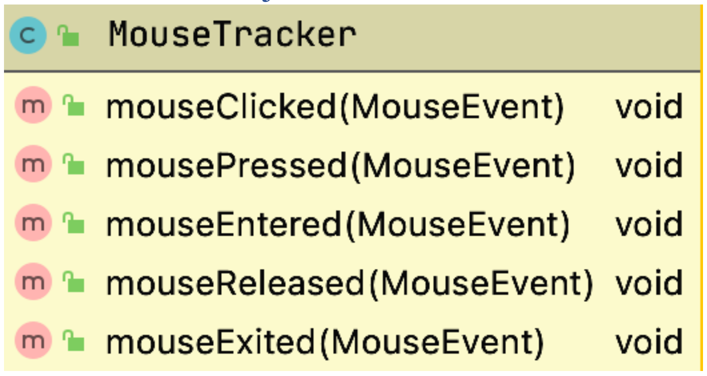
10.3 GameWorld.java



GameWorld	
setGameObj(Object)	void
main(String[])	void
init()	void
paintComponent(Graphics)	void

GameWorld.java is the driver class of the tank game, it contains the main () method for the game as well as the game window for the game.

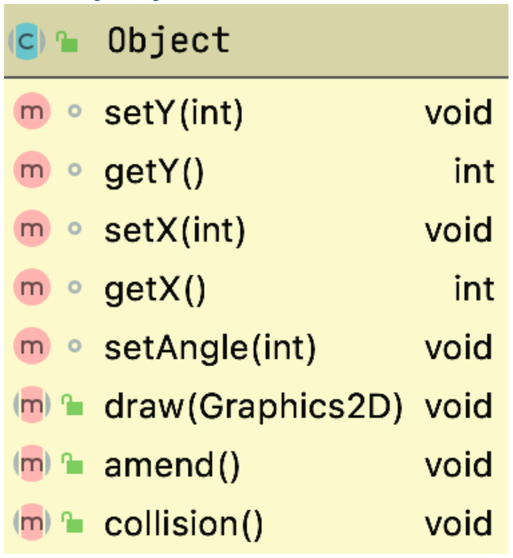
10.4 MouseTracker.java



MouseTracker	
mouseClicked(MouseEvent)	void
mousePressed(MouseEvent)	void
mouseEntered(MouseEvent)	void
mouseReleased(MouseEvent)	void
mouseExited(MouseEvent)	void

MouseTracker.java validates the mouse events: mouse click, mouse release and mouse exit which scans clickable region for allowing to user to either play the game, see the instructions or exit the game.

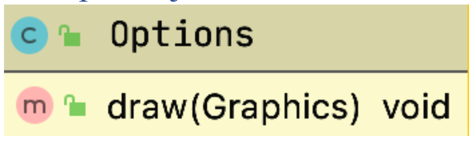
10.5 Object.java



Object	
setY(int)	void
getY()	int
setX(int)	void
getX()	int
setAngle(int)	void
draw(Graphics2D)	void
amend()	void
collision()	void

Object.java is an abstract class that holds x and y positions and the velocities. It has getters and setters for the positions and has rectangle shape which comes useful when handling the collision. It has abstracts methods that are used by the game objects in the game.

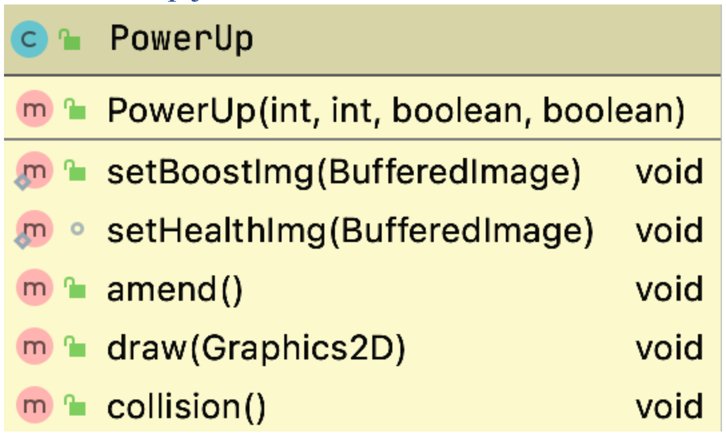
10.6 Options.java



```
Options
draw(Graphics)  void
```

Options.java has the properties for the game board and the screen like the fonts, color, width and height as well as the strings setup for the display.

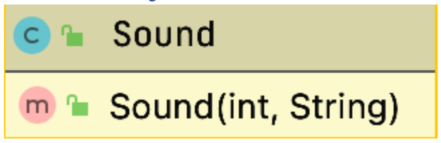
10.7 PowerUp.java



```
PowerUp
PowerUp(int, int, boolean, boolean)
setBoostImg(BufferedImage)    void
setHealthImg(BufferedImage)   void
amend()                       void
draw(Graphics2D)              void
collision()                   void
```

PowerUP.java is responsible to handle and update the lives of the tank as well as the power boost up for the tanks as the Collison occurs.

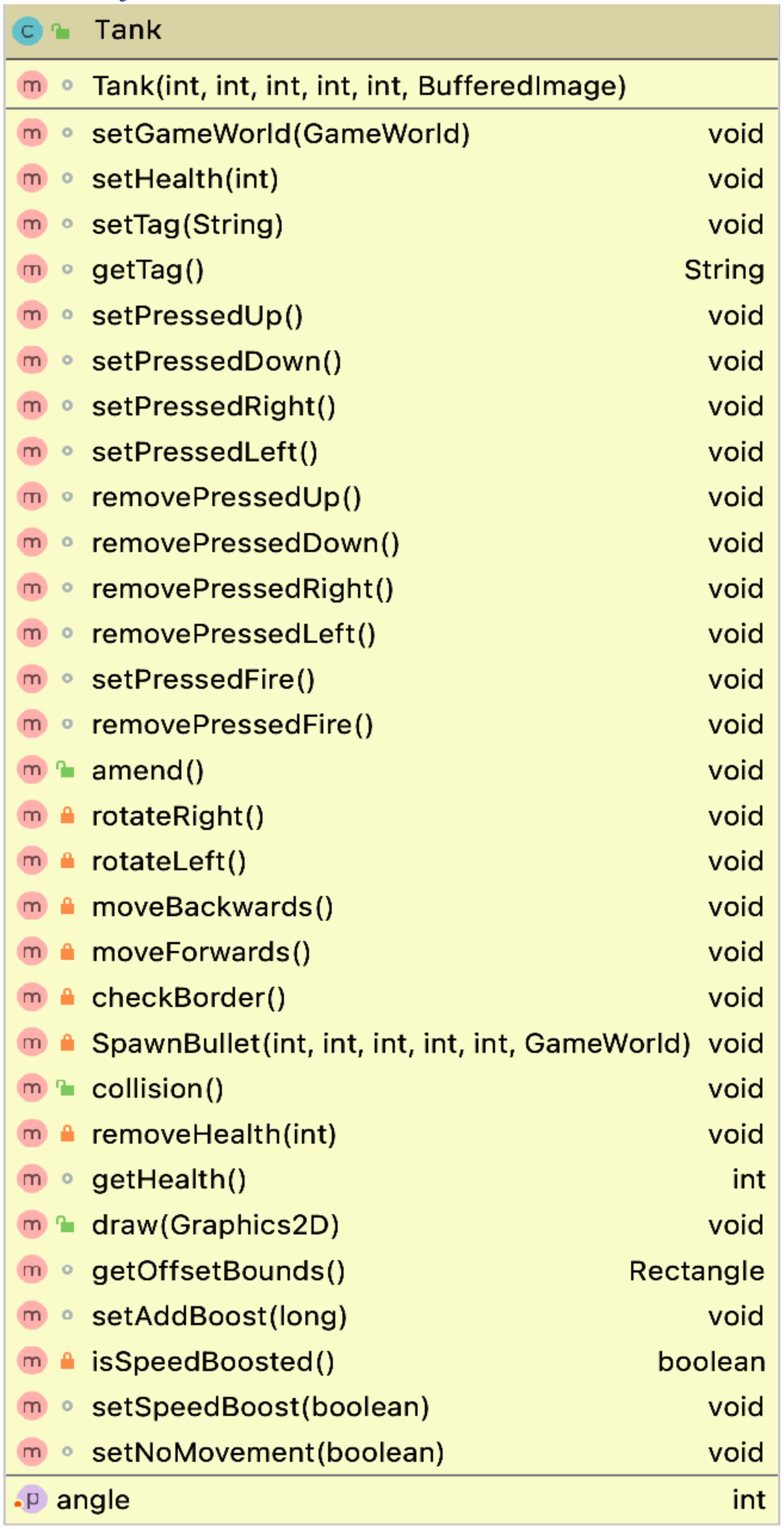
10.8 Sound.java



```
Sound
Sound(int, String)
```

Sounds.java is responsible for handling the audio system in the game i.e. load, play and loop the game sound in the game play.

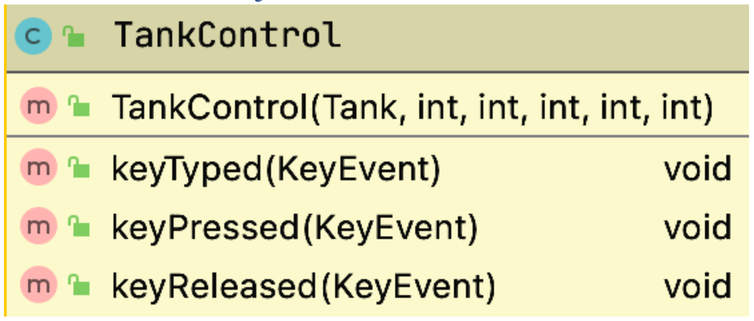
10.9 Tank.java



Tank	
Tank(int, int, int, int, int, BufferedImage)	
setGameWorld(GameWorld)	void
setHealth(int)	void
setTag(String)	void
getTag()	String
setPressedUp()	void
setPressedDown()	void
setPressedRight()	void
setPressedLeft()	void
removePressedUp()	void
removePressedDown()	void
removePressedRight()	void
removePressedLeft()	void
setPressedFire()	void
removePressedFire()	void
amend()	void
rotateRight()	void
rotateLeft()	void
moveBackwards()	void
moveForwards()	void
checkBorder()	void
SpawnBullet(int, int, int, int, int, GameWorld)	void
collision()	void
removeHealth(int)	void
getHealth()	int
draw(Graphics2D)	void
getOffsetBounds()	Rectangle
setAddBoost(long)	void
isSpeedBoosted()	boolean
setSpeedBoost(boolean)	void
setNoMovement(boolean)	void
angle	int

Tank.java adds many functionalities to our tank game. It extends the Object class and uses Boolean for tank movement control. It also allows to bullet spawn and holds methods for rotations and movement of tank in various directions.

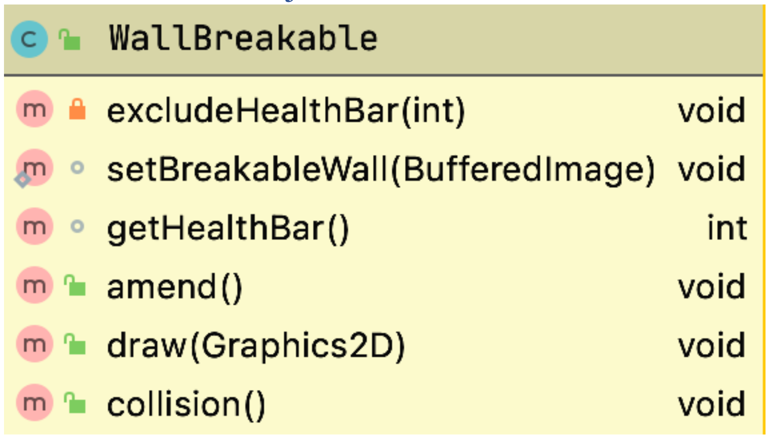
10.10 TankControl.java



TankControl	
TankControl(Tank, int, int, int, int, int)	
keyTyped(KeyEvent)	void
keyPressed(KeyEvent)	void
keyReleased(KeyEvent)	void

TankControl.java is responsible for handling the control of tank on basis of scanning the game keypad whether they are clicked to direct the movement or been kept pressed and if they have been released to abrupt the movement.

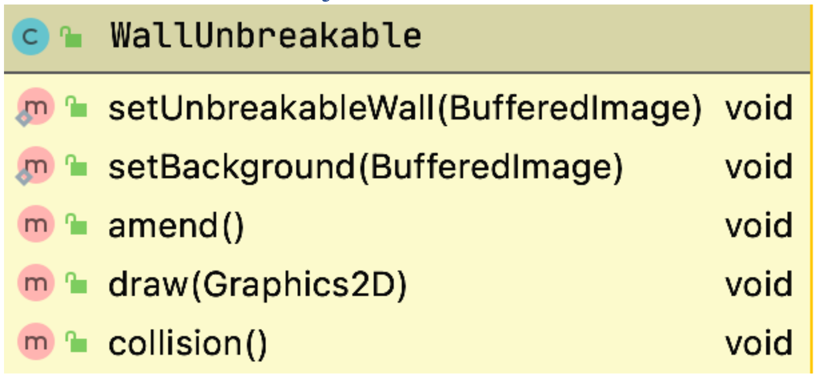
10.11 WallBreakable.java



WallBreakable	
excludeHealthBar(int)	void
setBreakableWall(BufferedImage)	void
getHealthBar()	int
amend()	void
draw(Graphics2D)	void
collision()	void

WallBreakable.java is responsible for setting up the breakable wall in the game board, then exclude as well as get the health bar and then validate collision with the wall.

10.12 WallUnbreakable.java



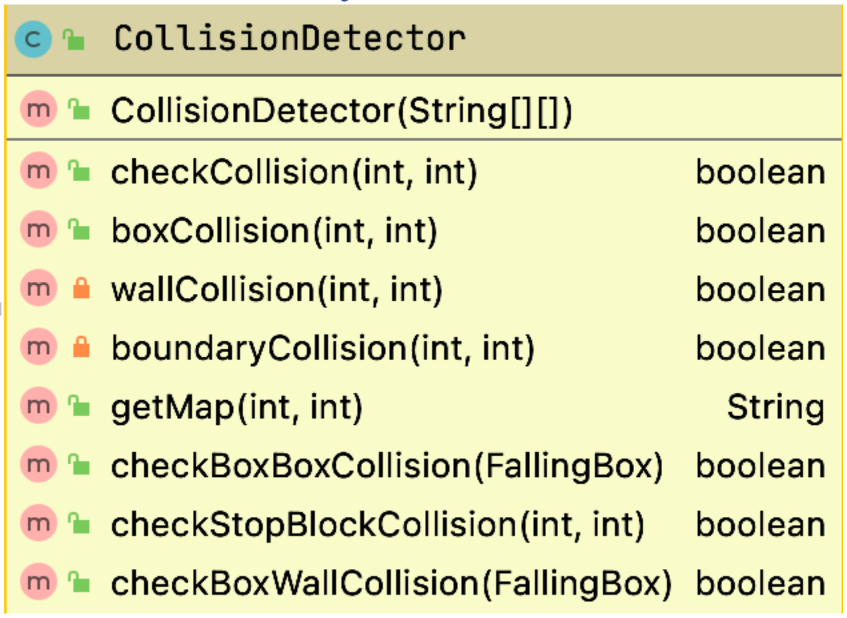
WallUnbreakable	
setUnbreakableWall(BufferedImage)	void
setBackground(BufferedImage)	void
amend()	void
draw(Graphics2D)	void
collision()	void

WallUnbreakable.java is responsible for keeping the wall stable regardless of Collison from bullets; it sets the background for the wall as well as set unbreakable functionality and then updates to the game board with the Collison validation.

11 Implementation Details of Lazarus Game Classes

Package: GameUnit

11.1 CollisionDetector.java



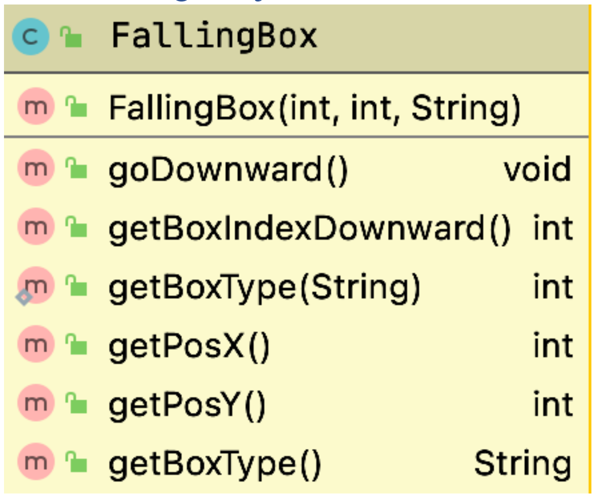
CollisionDetector	
CollisionDetector(String[][])	
checkCollision(int, int)	boolean
boxCollision(int, int)	boolean
wallCollision(int, int)	boolean
boundaryCollision(int, int)	boolean
getMap(int, int)	String
checkBoxBoxCollision(FallingBox)	boolean
checkStopBlockCollision(int, int)	boolean
checkBoxWallCollision(FallingBox)	boolean

CollisionDetector.java checks and validates the collision of Lazarus with other gam objects such as box, wall, boundary.

11.2 Constants.java

Constants.java has the constants values defined for the game objects.

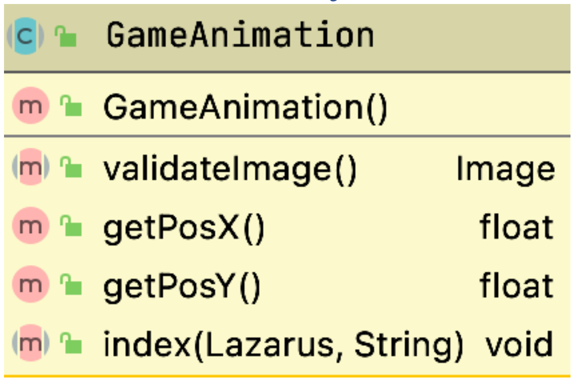
11.3 FallingBox.java



FallingBox	
FallingBox(int, int, String)	
goDownward()	void
getBoxIndexDownward()	int
getBoxType(String)	int
getPosX()	int
getPosY()	int
getBoxType()	String

FallingBox.java is responsible for handling the different types of boxes depending on their weights falling down the game board with their respective positions.

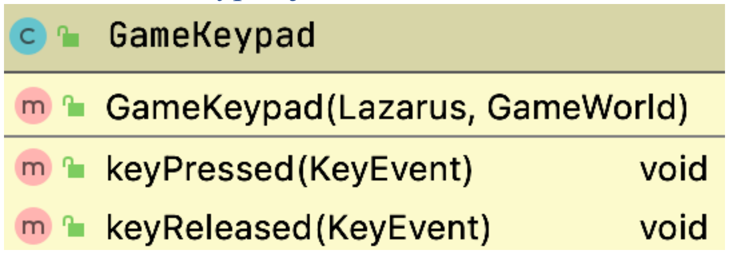
11.4 GameAnimation.java



GameAnimation	
GameAnimation()	
validateImage()	Image
getPosX()	float
getPosY()	float
index(Lazarus, String)	void

GameAnimation.java is responsible for handling the animation of movement in the game; validates the image and checks for position of Lazarus.

11.5 GameKeypad.java



GameKeypad

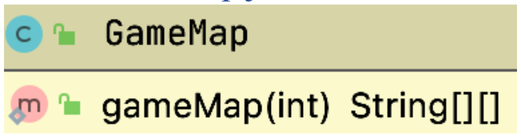
GameKeypad(Lazarus, GameWorld)

keyPressed(KeyEvent) void

keyReleased(KeyEvent) void

GameKeyPad.java extends abstract KeyAdapter that extends KeyListener and EventListener and is responsible for determining the game state as well as the keypad being pressed and released during the game state.

11.6 GameMap.java

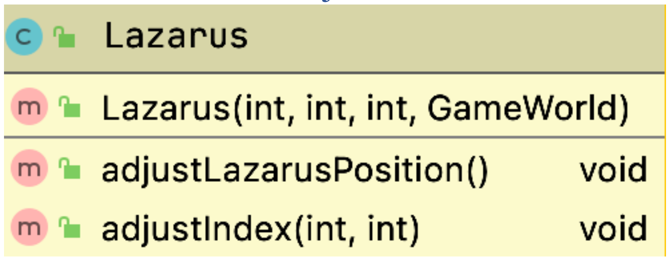


GameMap

gameMap(int) String[][]

GameMap.java is responsible for setting up the game board map for blocks, Lazarus and wall.

11.7 LazarusPosition.java



Lazarus

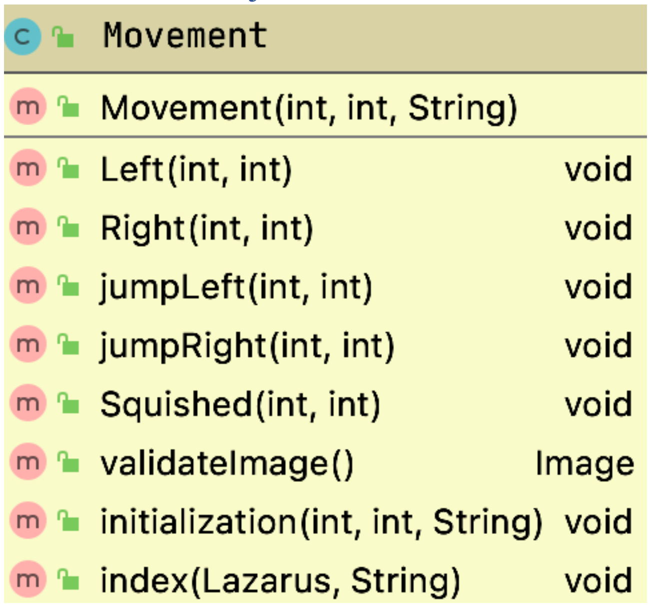
Lazarus(int, int, int, GameWorld)

adjustLazarusPosition() void

adjustIndex(int, int) void

Lazarus.java is responsible for handling the resetting and adjusting the position of Lazarus in the game board.

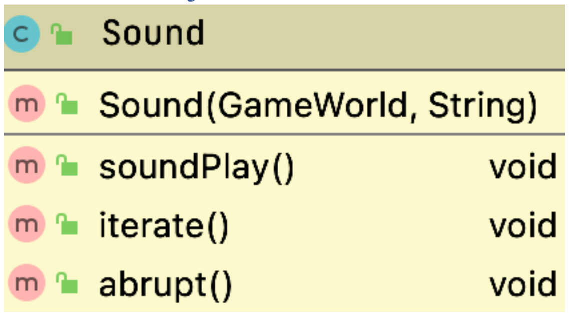
11.8 Movement.java



Movement	
Movement(int, int, String)	
Left(int, int)	void
Right(int, int)	void
jumpLeft(int, int)	void
jumpRight(int, int)	void
Squished(int, int)	void
validateImage()	Image
initialization(int, int, String)	void
index(Lazarus, String)	void

Movement.java is responsible for handling the movement of Lazarus in the game board; how it goes left, right, how it jumps to either direction and also validate the images depending on the moves Lazarus makes and then update the index.

11.9 Sound.java

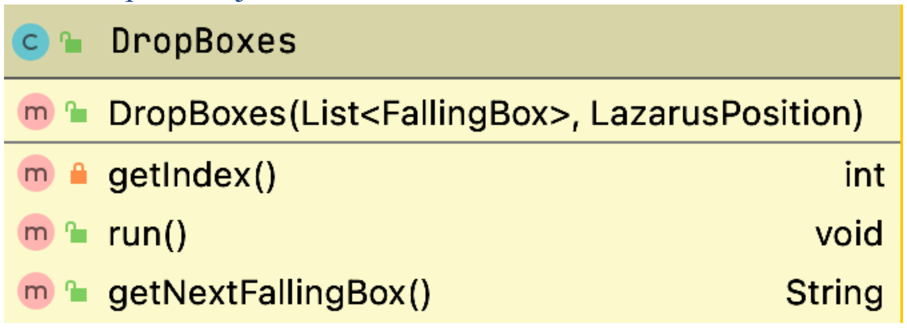


Sound	
Sound(GameWorld, String)	
soundPlay()	void
iterate()	void
abrupt()	void

Sounds.java is responsible for handling the game sound in the game like when to play it and loop the audio and when to stop audio play.

Package: GameWorld

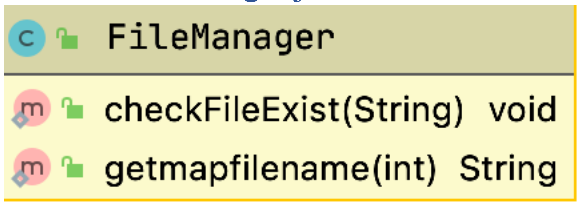
11.10 DropBoxes.java



DropBoxes	
DropBoxes(List<FallingBox>, LazarusPosition)	
getIndex()	int
run()	void
getNextFallingBox()	String

DropBoxes.java is responsible for handling the index of falling boxes as well as dropping boxes on the basis of their types with timer. It extends TimerTask.

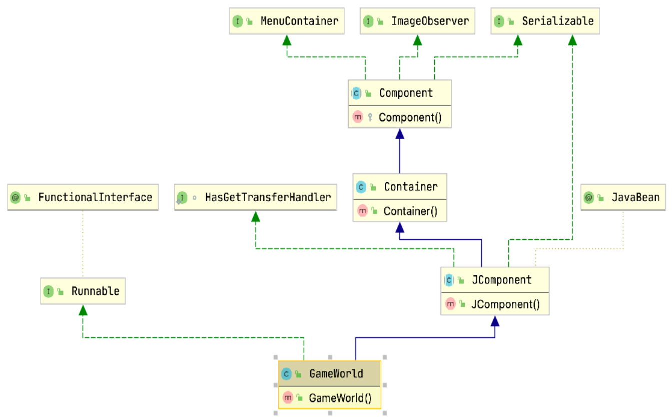
11.11 FileManager.java



FileManager	
checkFileExist(String)	void
getmapfilename(int)	String

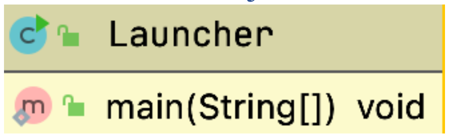
FileManager.java is responsible for validating file existence and handling the errors in case of file no existence.

11.12 GameWorld.java



GameWorld.java is the main class of the Lazarus game and it connects all the objects of the game and allows to execute runnable interface for the project. It is responsible for handling the runnable interface of the game; start the game with time as well as allow boxes to drop. Draw 2D graphics paint of the game board and handle the game state whether it is running or won or over. It handles the movement of Lazarus and boxes in the game condition and validates the Collison in the game, resets the game level if necessary.

11.13 Launcher.java



Launcher.java is our main driver class to run the program.

12 OOP and Code Reuse Discussion

Even though both games are not entirely same, they have similarity in the design and implementation. Both the games have objects like tank, wall, players, bullets, blocks, Lazarus. Moreover, the components for the game like sound, file reader, Collison detector, map, movement handler, launcher are used in both the game. Therefore, after making the tank game, it was easier to work on the Lazarus game with the idea of implementing OOP concepts in project and reusing the ideas code and classes from the first game. Object Oriented Programming principles have been applied in both tank game and Lazarus. The project and classes are designed in such a way that it is easy to use and maintain.

13 Project Reflection

This was my first-time building games in JAVA, and I loved it for the reason it taught me how to think in bigger picture when making a software or an application that is useful in real world. At first, it was tough for me to get the idea on how and where I start to build a game that has graphics on it. The guidelines provided by the instructor were very helpful in approaching the project as well as the slack channel discussion. After building two games for the term project, I feel I have good idea on OOP principle as well as solid design principle. Moreover, I realized how important is it to have well designed project and code for the future reference that we can use to develop another program.

14 Project Conclusion

To summarize this term project, it was fun and needed hard work to make it work. Both the game works fine as required; the movement in the direction, game state (ready, over, won), validating collision, audio player and binding the game objects in the game window and implementing in the runnable interface. I feel that this project has taught me how to step up to be a better programmer and write effective code as well as try to avoid and correct mistakes in developing a software. It was fun and I learned a lot during this game development project.