




Assignment 5 CSC413

Software Reusability across two games

Dusan Zika Cvetkovic

Student id: 916428960



Tale of Contents

1.	Introduction.....	1
2.	Tank game	1
2.1.	Class diagram	1
2.2.	Specific implementations	2
2.2.1.	Collision detection.....	2
2.2.2.	Battlefield creation.....	3
2.2.3.	Animated background	3
2.2.4.	Input controller	4
3.	Lazarus Game	5
3.1.	Class diagram.....	5
3.2.	Implementation	5
3.3.	Movement implementation	5
4.	Android Tank game.....	6
4.1.	Porting code from Java to android framework	6
4.2.	Server part and client modifications.....	6
5.	Conclusion	7

1. Introduction

This project is assignment 5 for CSC 413 class. Main goals:

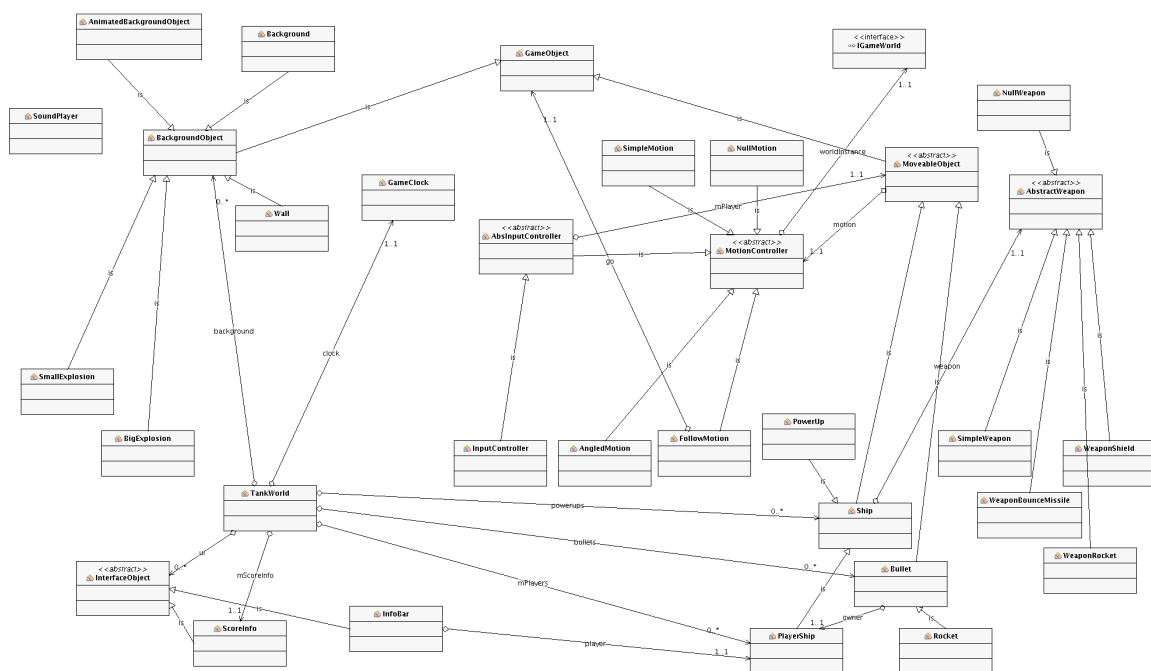
- learning how to think about and make reusable code,
- developing game in Java from scratch
- porting code to Android platform
- developing server side for Android game

In this paper I am going to try to explain how I reused my code across to games and how I ported one of my games to android platform.

2. Tank game

Tank game is just one part of the project. Other two parts of the Java project are game engine and Lazarus game. This is also the separation between packages in my project. I have three main packages gamengine, tankgame and lazarusgame.

2.1. Class diagram



Upper part of the diagram contains entire gameengine (GE) package. Lower part of the diagram is tankgame implementation.

Most heavily used class from GE is GameObject class. It is extended by BackgroundObject and MoveableObject. Background objects are static objects on the scene like Wall, SmallExplosion and BigExplosion, while MoveableObject are objects that can move on the scene like Bullets or Ships (Tank extends Ship).

On the tankgame side of the diagram main class is TankWorld. It keeps track of all the object drawn on the scene like players (Tanks), bullets, walls and powerups.

2.2. Specific implementations

In next couple paragraphs I will try to explain some of the interesting implementations that I did.

2.2.1. Collision detection

Function collision is in charge of detecting any collision that can happen between two instances of GameObject class.

```
public boolean collision(GameObject otherObject) {
    Area a = getCollisionArea();
    a.intersect(otherObject.getCollisionArea());
    return !a.isEmpty();
}
```

As it can be seen from the snippet getCollisionArea() method is very method that provides interface for all the subclasses to define specific Area which is defined as collision area for that specific subclass. For example, in PlayerShip (Tank), we have:

```
@Override
public Area getCollisionArea() {
    Rectangle rec;
    if (down == 1) {
        rec = new Rectangle(location.x + 7, location.y + 9, 27, 45);
    } else if (up == 1) {
        rec = new Rectangle(location.x + 7 + 27, location.y + 9, 27, 45);
    } else {
        rec = new Rectangle(location.x + 7, location.y + 9, 53, 45);
    }

    AffineTransform af = new AffineTransform();
    Area a = new Area(rec);
    af.rotate(heading, this.location.getCenterX(), this.location.getCenterY());
    Area ra = a.createTransformedArea(af);
    return ra;
}
```

Don't mind looking at specific numbers, these are simply taken from looking at the png provided for Tank object. Since that picture has border that we do not want to take into account when we collide tank with other objects on the scene, the Area that we want to collide with other objects is defined here without that transparent border. Another interesting thing in this implementation are down and up values. These values are set when user presses down and up keys respectively. When user is pressing up key we don't want back part of the tank to be checked for collision (with wall) so we specify just the front part of the tank to be checked for collision. Similar thing is done for front part and down key as well.

2.2.2. Battlefield creation

Creating of battlefield is done using config file:

```
//size of the battlefield --- "bf width height tileSize"
//bf 640 480 32
bf 960 768 32
//define walls (non breakable) --- "w width height locationx locationy"
w 960 32 0 0
w 960 32 0 23
w 32 768 0 0
w 32 768 29 0
w 96 32 1 11
w 96 32 1 3
w 64 160 9 5
//define breakable walls --- "bw width height locationx locationy"
bw 32 160 14 1
bw 32 160 14 9
```

Static method from GE Utils class is used to create HashMap from input file:

```
public static HashMap<String, ArrayList<Integer[]>>
getBattlefieldDataFromConfigFile(String battlefieldconfig, Class<?> aClass )
```

When the hash map (configfile in next example) is initialized with data from file read we proceed with drawing when we initialize the scene. Example:

```
for (Integer[] vals : configFile.get("w")) {
    addBackground(new Wall(vals[0], vals[1], new Point(vals[2] * tileSize,
vals[3] * tileSize), wall1, false));
}
```

2.2.3. Animated background

All the magic happens in AnimatedBackgroundObject class. Basically what this class does is update itself (meaning go to next sprite element) on every 5 frames. It implements Observer class so it gets notified as any other object when it needs to be updated (on every frame controlled by GameClock). Number of frames for controlling how fast animation could be could be easily externalized.

All that each object who wants to be Animated needs to do is extend this class and define constructor which receives location point as a parameter. Another thing it should do is call super constructor and provide sprite and number of images in sprites. Everything else is done by super class.

```
public class SmallExplosion extends AnimatedBackgroundObject {
    static BufferedImage sprite =
ImageUtils.toBufferedImage(TankWorld.sprites.get("Explosion_small_strip6"));

    public SmallExplosion(Point location) {
        super(location, sprite, 6);
    }
}
```

2.2.4. Input controller

Handling user's input is part of every game so I tried to make some `AbsInputController` in `gameengine` which could be extended to handle specific requests for every game made. All that each of the subclasses need to do is implement these methods:

```
@Override
    public abstract void keyReleased(KeyEvent e);
    @Override
    public abstract void keyPressed(KeyEvent e);
```

Example:

```
@Override
    public void keyPressed(KeyEvent e) {
        int code = e.getKeyCode();
        // left
        if (code == keys[0]) {
            this.setMove("left");
        } // up
        else if (code == keys[1]) {
            this.setMove("up");
        } // right
        else if (code == keys[2]) {
            this.setMove("right");
        } // down
        else if (code == keys[3]) {
            this.setMove("down");
        } // fire
        else if (code == keys[4]) {
            this.setFire();
        }
        setChanged();
        this.notifyObservers();
    }
```

`setMove` is method defined in super class (GE) and it assumes that Object controlled by this input controller has aforementioned variables (left, right...). `SetFire` method is specific for Tank Game and is define in `InputController` class.

Controls that are available for game being implemented can be specified via constructor:

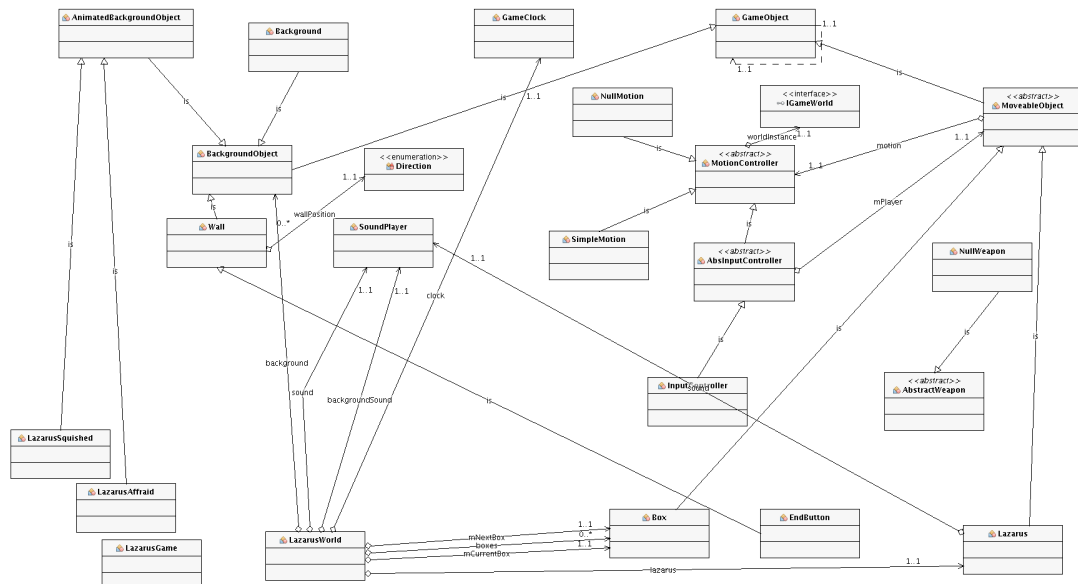
```
public InputController(PlayerShip player) {
    this(player, new int[]{KeyEvent.VK_LEFT, KeyEvent.VK_UP, KeyEvent.VK_RIGHT,
    KeyEvent.VK_DOWN, KeyEvent.VK_SPACE});
}

public InputController(PlayerShip player, int[] keys) {
    super(player, TankWorld.getInstance(), PlayerShip.class, keys);
}
```

3. Lazarus Game

In this chapter after showing the class diagram I will point out the main components being reused from GE package. Then I will go over the implementation of lazarus movement since that is the only somewhat difficult part in this game.

3.1. Class diagram



3.2. Implementation

Same like with Tank Game in Lazarus Game (LG) we can see in upper part classes from GE and in lower part classes from LG. On the LG side of the diagram main class is LazarusWorld. It keeps track of all the object drawn on the scene like player (Lazarus), boxes, walls...

As we can see I used many classes from GE to make my life easier. Classes extended are `AnimatedBackgroundObject` (`LazarusSquished` and `LazarusAffraid`), `AbsInputController` (`InputController`), `MoveableObject` (`Boxes`, `Lazarus`).

3.3. Movement implementation

I am not too proud of this implementation, because I think it could be done better. Any way I will explain my reasoning in next couple sentences. All the code I will be explaining here is the code from Lazarus class.

Method `update()` is getting called on every frame update. In this method we check if `lazarus` is currently in the move and if it is we just call the same movement function until its done. If the movement function is initiated for the first time (meaning first time that we detect the request for movement, e.x. key pressed), we compare level of `lazarus` and the level of the desired movement side. Based on the result we move `lazarus` to next position, one level up, moving on same level, or moving desired side and then making him fall until he hits

something. One helper array that we make use of in every frame is possible moves array which is simply an indicator of Lazarus possible moves.

4. Android Tank game

In this chapter I will just make a quick overview if what I have learned is needed in order to port the game from desktop game written in Java to Android platform.

4.1. Porting code from Java to android framework

While importing code to Android Studio most of the errors were due to next couple required transitions:

- JPanel paint method in Java is used to handle updates and drawing each frame, control the frame rate... In order to be able to use pretty much same functions in Android framework I used Surface View combined with separate thread. In thread I draw scene on surface view in infinite loop with small sleep time between frames.
- Majority of small errors which were tedious and made me spend most of my time were transition from Area -> Region, Rectangle -> Rect... The transition itself is not that hard, but finding the exact class which can provide same functionality like the ones from Java was bit of a challenge.

After all the transitioning was done and app started working same like in the desktop environment. But it was not playable on one device since it's multiplayer game. Then I decided to make server side of the game.

4.2. Server part and client modifications

I had already working sample code provided from the professor. I just made couple requests and responses on the server side. On client side basically in every request I need to send request size, request code, followed by data that server needs to handle in some way. I needed to add new request for every modification required to be made on the scene. This means:

- Adding/removing each player to the scene
- Adding/removing each bullet
- Update tank current data (position, heading)
- Increasing/decreasing health
- ...

5. Conclusion

This project is very helpful for realizing what it takes to be done in order to develop game. Dealing with collisions and porting the application to Android framework was extremely interesting to me and I had so much fun working on this project. It was at the same time pleasant and challenging experience.

There is obviously much more that I did not explained, especially for Android part in this paper, but I think the code is pretty well written for both server side and client as well. There is lot of room for improvement as in every other project, but considering the time frame and the work done, I am satisfied with the work done and with my overall experience with the project.