CSC 413 Term Project Documentation

Fall 2018

Anmol Gondara

CSC 413.01, Fall 2018

https://github.com/csc413-01-fa18/csc413-tankgame-anmol2727

https://github.com/csc413-01-fa18/csc413-secondgame-anmol2727

Table of Contents

1 Introduction

a. Project Overview:

The term project was composed of two, two-dimensional games that were written in Java. The first game was a two player tank game called Tank Wars, and the second was a one player game called Super Rainbow Reef. The objective of this project was to practice and apply good object oriented programming (OOP). The ultimate goal was to design and implement the tank game using OOP and then reuse a good segment of the code in the second game.

b. Intro to the Tank Game:

The tank game – formally called Tank Wars – is a two player game in which each player controls a single tank. Both tanks can move forwards, backwards, left, and right. The game accounts for collisions between the tanks and the destructible & indestructible walls. The pair of tanks have the ability to shoot bullets that can destroy each other and or the destructible walls when they collide. Other features of the game include: split screen; mini-map; and health bar, lives, and score count for each tank. The end goal is to destroy the opposing tank until all its lives run out.

c. Intro to the Super Rainbow Reef Game:

The Super Rainbow Reef game is a one player game where the user controls a character known as Katch who can only move left and right. Katch has to bounce his friend Pop, who is another character in the game, off his shell and help him destroy (by way of colliding) the enemy character Biglegs. Similar to the tank game, this game also accounts for collisions – in this case between the three characters and the destructible and indestructible bricks. Supplementary features include: multiple levels, powerups to increase lives remaining, score and life count. The end goal is to destroy all of the Biglegs in every level.

2 Development Environment

a. Version of Java Used:

   1.8.0_101

b. IDE Used:

   IntelliJ IDEA 2018.2.5 (Ultimate Edition)

c. Special Libraries or Special Resources Used:

   - javax.swing.*;
   - java.awt.*; java.awt.image.BufferedImage; java.awt.image.ImageObserver; java.awt.event.KeyEvent; java.awt.event.KeyListener;
   - javax.imageio.ImageIO; java.io.BufferedReader; java.io.File; java.io.FileReader;
   - java.util.Objects; java.util.logging.Level.*; java.util.logging.Logger.*;

3 How to Build/Import the Games

a. How to Import Game:

First I accepted the invitation to create my repository by clicking on the link provided on iLearn.

I opened the Terminal application on my computer and typed 'cd ~/Desktop' to change my directory to my desktop. Then I typed 'git clone <link>' where <link> was my GitHub repository link with '.git' attached to the end. This step cloned my repository to my desktop.

To begin importing the project to my IDE, I opened IntelliJ and clicked 'Import Project' on the startup screen, followed by selecting the 'csc413-tankgame-anmol2727' folder, which I cloned to my desktop in previous step. Next, I clicked

‘Create project from existing sources’, and chose ‘Next.’ To finish the process, I continued to click ‘Next’ until I was prompted with the ‘Finish’ option.

To successfully complete the importing procedure, I clicked File -> Project Structure -> Project -> and chose my ‘Project SDK’ as 1.8. I then changed the ‘Project language level’ to 8. To save my changes, I clicked ‘Apply’ and ‘OK.’

Lastly, I marked the csc413-tankgame-anmol2727 folder as the sources root.

I repeated these steps for the Super Rainbow Reef game.

b. <u>How to Build the JARs</u>:

First I put my Resources folder inside the jar folder that was already given to us.

Then I clicked File -> Project Structure -> Artifacts -> and chose the ‘+’ button and then selected JAR -> From modules with dependencies. In the ‘Module’ box I chose ‘csc413-tankgame-anmol2727’ and in the ‘Main Class’ I chose ‘Window.java’ (for the Super Rainbow Reef game I selected ‘Console.java’). I then picked the option to ‘extract to the target JAR’ and picked the jar folder as the ‘Directory for META-INF/MANIFEST.MF’ and confirmed with ‘OK.’ Lastly I chose the jar folder as the ‘Output directory’ and clicked ‘Apply’ and ‘OK.’

To build the JAR, I clicked Build -> Build Artifacts -> and then chose ‘Build’ under ‘Action.’ This created the JAR file inside the jar folder.

I repeated these steps for the Super Rainbow Reef game.

c. <u>Commands to Run the JARs</u>:

```
cd ~/Desktop/csc413-tankgame-anmol2727/jar
java -jar csc413-tankgame-anmol2727.jar
```

```
cd ~/Desktop/csc413-secondgame-anmol2727/jar
java -jar csc413-secondgame-anmol2727.jar
```

4 How to Run the Games/Rules & Controls

Running the Tank Game/Rules & Controls:

To run the Tank Game, you can open the Window.java class and click on the green play button on either line six or twenty and choose 'Run Window.main().' Another option is to right click the Window class in the left panel and choose 'Run Window.main().'

The rules for the Tank game are simple. There are two players who each control a single tank and are trying to destroy each other with bullets that shoot out of the said tanks. There are walls scattered throughout the map, the red ones are indestructible while the orange ones can be destroyed with the bullets. The tanks may only go past the walls by destroying them, otherwise they will collide. Each tank has a total of two lives. The tank that kills the other tank two times first wins the game.

The controls for the game are as followed:

Tank1(positioned on the left side) – **wKey**(moves the tank **up**), **dKey**(moves the tank **right**), **sKey**(moves the tank **down**), **aKey**(moves the tank **left**), **fKey**(**shoots bullet** from tank)

Tank2(positioned on the right side) – **upKey**(moves the tank **up**), **rightKey**(moves the tank **right**), **downKey**(moves the tank **down**), **leftKey**(moves the tank **left**), **enterKey**(**shoot bullet** from tank)

Running the Super Rainbow Reef Game/Rules & Controls:

To run the Super Rainbow Reef Game, you can open the Console.java class and click on the green play button on either line fourteen or thirty-four and choose 'Run Console.main().' Another option is to right click the Window class in the left panel and choose 'Run Console.main().'

The rules for the Super Rainbow Reef game are straightforward. You control the character called Katch, which is the shell. Your goal is to bounce the character called Pop, which is the starfish, off of Katch and hit all of the Biglegs, which are the squids. There are rainbow colored bricks that stand in the way of Pop and the Biglegs, which you can destroy by making contact with them. There are also bricks that have a heart in them that upgrade your life count. In total, there are three levels and to move onto the next level, you must destroy all the Bigelegs present (by making contact with them) on the screen. To win the game you have to finish level three with one life or more remaining.

The controls for the game are as followed:

Katch – **leftKey**(moves Katch **left**), **rightKey**(moves Katch **right**), **enterKey**(**shoots Pop** off of Katch)

5 Assumptions Made

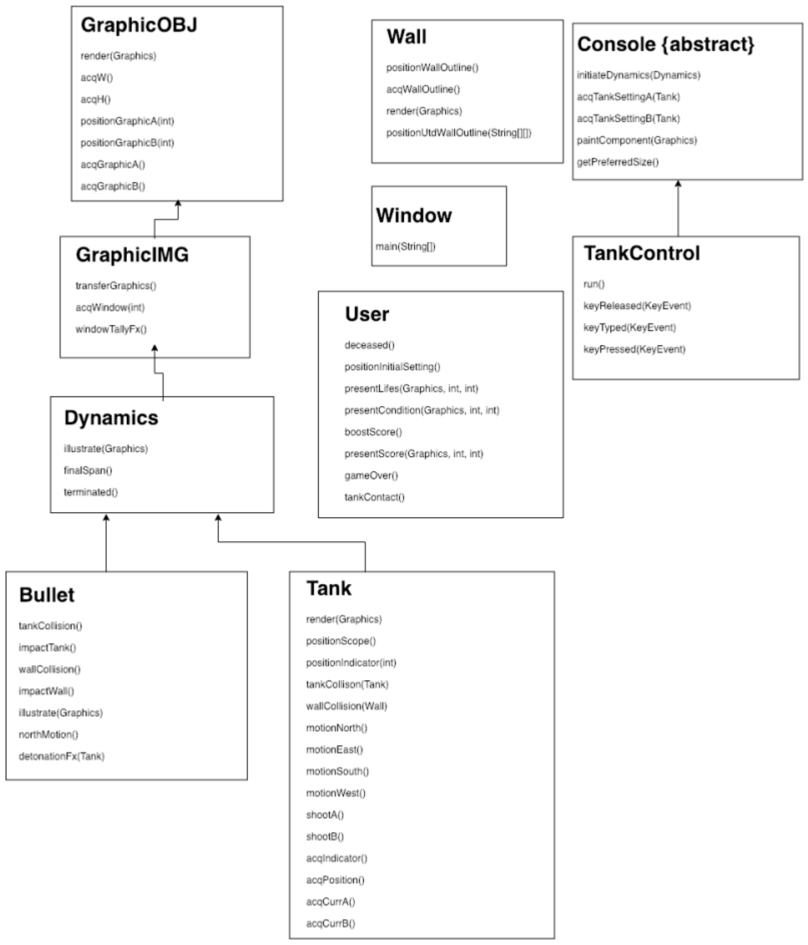
Assumption 1:

When beginning the Tank game, I assumed that I could learn about Swing and the Abstract Window Toolkit (AWT) as I developed the game. This was a big mistake, which ended up slowing my progress down substantially. I have no one to blame for this but myself because I knew that working with graphical user interfaces (GUIs) was one of my weak points as a programmer. I actually realized that with the first assignment (calculator) and I even discussed how the GUI was troubling for me in the documentation. Obviously this project dealt largely with building a complex graphical user interface so I should have spent more time reading about Swing and AWT before beginning the coding process.

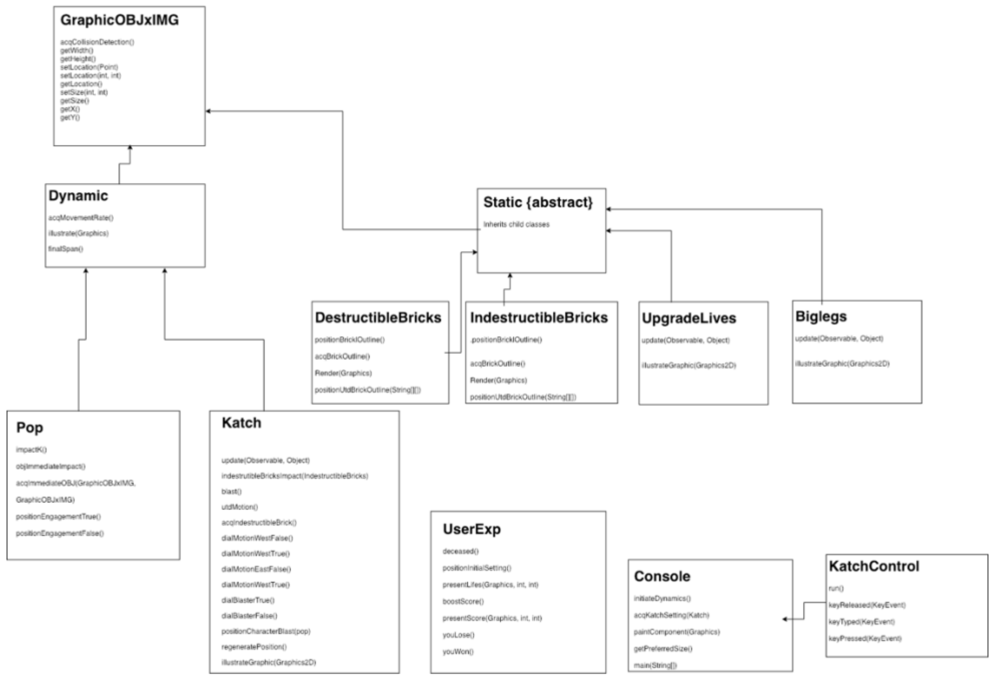
Assumption 2:

When developing the Super Rainbow Reef game, I assumed that it would be better if rather than reusing my classes from the Tank game exactly, I redesign the structure of the game to make it more organized. Although my second game came out to have a better design, it took me much longer than it really should have. I would have been better off directly reusing the same classes instead of mixing around the methods. This lead to more methods having to be implemented, and when I realized my mistake, it was too late to start over, as I was worried I would not have enough time to finish. This all happened in large part to my first assumption because if I had done a better job learning about Swing and AWT, my first game's design would have been much better, and translated to my second game in a smoother fashion.

6 Tank Game Class Diagram



7 Super Rainbow Reef Game Class Diagram



8 Class Descriptions of All Classes Shared Among Both Games

Between both of my games, I had a total of twenty-one classes and I used all but five of the classes among both games. I would like to note that some of the classes I mention in this section are not *exactly* the same, as I changed around a few of the methods between them to improve the overall design of the second game, but looking at them from an overview, they accomplish the same objectives.

<u>GraphicOBJ/GraphicIMG and GraphicOBJxIMG classes</u>:

For the tank game I created two classes, one called GraphicOBJ and the other GraphicIMG. When I was designing the second game, I wanted to make combine these two classes so it would be more concise and organized.

In essence, the classes are used to create the games objects and then render the graphics for the user to see. The child classes that are inherited by these parent classes are either dynamic and or static. These include the tanks, bullets, Katch, Pop, walls, bricks, life upgrades, and Biglegs.

<u>Console classes</u>:

The Console classes are used to set the game world. They set the display dimensions, provide the resource paths, Constructors used to process the graphics of the game objects described above through GraphicOBJxIMG, and call various methods to ultimately execute the game in the main method. These classes extend the subclass JFrame and implements Runnable. For the tank game, this is where I implemented the split screen and mini-map. For the Super Rainbow Reef game, I created the maps for the game through an array in this class.

Tank and Katch classes:

The Tank and Katch classes are child classes of the parent class GraphicOBJ and are mainly used to detect collisions between the two objects and other game objects such as tanks, walls, Pop, Biglegs, bricks, and life upgrades. Other features of these classes are the way each object moves in terms of angles. Additionally, both classes acquire the positions of the respective game objects. Some of the resources used in these classes include Math and BufferedImage. Math is used to handle angles as mentioned before and BufferedImage is used to render resources.

TankControl and KatchControl classes:

The TankControl and KatchControl classes are used to control the tanks in the tank game and Katch in the Super Rainbow Reef Game. The classes extend Console and implement the subclass KeyListener. The only changes I made in these two classes are that in Tank Control I decided to do a switch statement and in KatchControl I decided to do if statements. This is in large part due to the tank game having more controls as there are two players.

Bullet and Pop classes:

Bullet and Pop classes are similar to the Tank and Katch classes, as they are both child classes of the parent class GraphicOBJ and are used to detect collisions between the two objects and tanks, Katch, Biglegs, walls, bricks, and life upgrades. Motion is also taken care of for the objects through the Math resources. For the Bullet class, detonation animations are also implemented. The Objects resource is used for both classes as well.

User and UserExp classes:

The User and UserExp classes are all about user specific features such as the user's scores, health conditions, life counts, and level handling (only for Super Rainbow Reef game). Furthermore, both classes handle when the a player has lost a game and when player has won a game, displaying graphics based on the situation.

Wall and IndestrcutibleBricks classes:

The Wall and IndestructibleBricks classes both account for collisions, but the main difference between the two is that the Wall class also removes the destructible walls when a collision is made. I decided to change this up for the second game because I wanted to have better organization. Another key difference in this class is the fact that I created the map outline for the tank game through a text file, whereas I did not to that for the Super Rainbow Reef game.

Dynamics and Dynamic classes:

The Dynamics and Dynamic classes were used as a stepping point towards moving game objects such as the tanks, bullets, Katch, and Pop. The classes illustrated the graphics for the listed objects and passed onto the game objects classes.

9 Class Descriptions of Classes Specific to Tank Game

For the tank game, I only have one class that is specific to only the tank game, whereas I reused the rest of my classes – some directly and some indirectly – in the Super Rainbow Reef game.

Window class:

In the Window class, I simply created a subclass of JFrame by using the Swing Java Foundation Class. This class was solely for creating a window and adding functionality such as window name, close functionality, position, thread loop to update the window, etc. Furthermore, I placed my main method in this class to ultimately run the game. I did not reuse this class in my second game because I wanted to be more efficient so I decided to do the same thing but rather in the Console class (as explained above).

10 Class Descriptions of Classes Specific to Super Rainbow Reef Game

For the Super Rainbow Reef game, I tried to design the game a little differently than the tank game, so this resulted in a few extra classes that I had to create. These classes all extend the game object class, but the difference was that I created a Static abstract class that would inherit objects that can all be destroyed.

Static class:

The Static was simply used as an abstract class that inherited all of the static objects: DestructibleBricks, UpgradeLives, and Biglegs. I created this in the second game because it made the overall organization of my game much better. Inside the class a I declared a constructor that takes into consideration the position of the graphics as x and y coordinates, the height and width of said graphics, and a BufferedImage graphic.

DestructibleBricks class:

The DestructibleBricks class extends Static and creates a constructor that accounts for every destructible bricks collision and how many points the user scores when Pop collides with the bricks.

UpgradeLives class:

The UpgradeLives class extends Static and creates a constructor that accounts for every life brick collision and how many points the user scores when Pop collides with the life.

Biglegs class:

The Biglegs class extends Static and creates a constructor that accounts for every bigleg collision and how many points the user scores when Pop collides with the bigleg.

11 Self-Reflection on Development Process

Working on the term project had its highs and lows for me in various aspects. I can say with conviction that this project was by far the most difficult for me to do throughout my academic career. As I stated earlier, I had never worked with graphical user interfaces before this semester and obviously this project focused largely on Swing and AWT. Although I'm not proficient with these Java Foundation Classes (JFCs), I still learned a lot throughout these past eight weeks. I made a lot of mistakes in the development process that cost me a lot of time, and I hope to learn from these faults in future work. As always, one of the biggest issues was time management, something that I never seem to get right. I ended up focusing a lot on certain aspects of the games such as graphical design (animations) rather than spreading my time out wisely and focusing on other mechanics such as collisions. My hope is to better my time management in the future so I'm not stressing myself out as the deadline approaches. I'd also like to say that my organization was not the best in my first game, so I wanted to better that in my second game by redesigning the classes. This turned out to be a mistake because I ended up spending almost double the time than what I had originally hoped for. If I had designed a better first game, my second game would have been implemented in a more fluid way. Overall, I learned that I should not try to digest something completely new as I code, and rather do my research beforehand and then start to develop my project (in reference to Swing and AWT). This would have helped in my time management and in result in my overall design of the games.

12 Conclusion

In the end, through all my ups and downs with the term project, I was able to complete both games. For the tank game, I was able to meet all the requirements besides creating a working power up. Once again, the reason I was not able to implement the power up was due to my poor time management. I had implemented the class for it but it crashed my program and I could not figure out why. I ended up keeping the class and implemented it successfully in my Super Rainbow Reef game. Out of both the games, my second game was better overall as it had a better design and met all the requirements.

All in all, the term project was a great learning curve for me. I came away with a lot of knowledge that I previously did not have. Even though I did not get to become any sort of expert on game design, I still believe I became a better programmer than I was before this assignment. One of my regrets for this assignment was the fact that I did not have the time to implement a sound player class, something that I had hoped I could accomplish. Despite this fact however, I enjoyed working on the two games and the past eight weeks were a tough challenge but a great experience nonetheless.