Team 22
John Mendoza & Naylin Min
CSC 413: Software Development
T, TH: 7-8:15
Spring 2018 Semester

Tank Game: https://github.com/csc413-02-sp18/csc413-tankgame-Team22
Super Rainbow Reef: https://github.com/csc413-02-sp18/csc413-secondgame-Team22

Introduction:

For our semester project, we are to code two games from scratch. For the first game, we are to implement a 2-player Tank Game. For our second game we get to choose the game to implement and we choose to do Super Rainbow Reef, a brick breaker game.

For the Tank Game, the idea is for two tanks to try and destroy the other tank and get rid of all their lives. The requirements are to have 2 players, each controlling their own tank. The two tanks should be able to move up and down and to rotate left and right. The movement and the rotation of the tanks should be smooth. The tanks should be able to shoot bullets which would do damage to the other tank. The game should also include a split screen for the players as well as a mini-map. Both players should have health bars and life count, both should decrease properly. The map include breakable walls and unbreakable walls as well as a power up of our choosing. The collision between tanks, walls, powerups, and bullets are to be handled accordingly.

For our second game, we choose Super Rainbow Reef. In Super Rainbow Reef, *BigLegs* had invaded Rainbow Reed and the duo of *Pop* and *Katch* must stop them to save their home. The idea of Super Rainbow Reef is for *Pop* to defeat the *Big Legs* on the map with the help of *Katch*. However, the *BigLegs* are often hiding behind the coral defenses. Much like a typical brick breaker game, *Katch* is to catch *Pop* as he bounce off the coral defenses(bricks) and the *BigLegs* on the map. If *Katch* fail to catch *Pop*, he would fall out of the level and lose a life. The requirements of Super Rainbow Reef are that *Pop* would move realistically. If *Pop* bounces off the left-end of *Katch*, *Pop* should bounce toward left side, while bouncing off the right-side would send Pop right. Left and right arrow keys would move *Katch* horizontally to catch *Pop*. There should also be "gravity" in the game as well, where *Pop* speeds up each time he collide with Katch. The game is to have several levels, containing a number of *BigLegs* to defeat. Breaking the coral defenses (bricks) as well as defeating the *BigLegs* will accumulate points. There are to be special blocks that give the player extra rewards but these special blocks and the bricks don't have to be destroyed in order to finish a level. If *Pop* and *Katch* defeat the *BigLegs* in all the levels or if *Pop* falls off the map with no lives left, then the game would end and high scores would be displayed.

For both our projects, we used the IDE NetBean 8.2 and JDK 1.8.

//Need to mention how to build or import your game in the IDE you used and how to run the game. Also assumptions made.

Rules & Controls:

Tank Game:

Each player get 3 lives to begin the game. The goal is to defeat the other player by shooting at tank until their health bar is at zero. If a player lose all 3 lifes, the game will end. Player 1 controls: Up - move up, Down - move down, Left - rotate left, Right - rotate right, and Space - shoot. Player 2 controls: W - move up, S - move down, A - rotate left, D - rotate right, and Q - Shoot.

Super Rainbow Reef:

The main goal is for *Pop* and *Katch* to clear the coral defenses and defeat the *BigLegs* in all the levels. Secondary goal is to get the high score for the game. *Katch* controls: Left - move *Katch* left and Right - move *Katch* right.

Class Diagram:

Tank Game Class Diagram:

Super Rainbow Reef Class Diagram:

Shared Classes:

Even though the two games are fundamentally different in game play, controls, and rules, there are a lot of overlapping ideas and classes that are used in both games. The classes that were used in both games are Background class, GameEvents class, GameObject class, Controls class, Game class, and Window class. **Background class** handles the background image on both classes. **GameEvents class** extends Observable and handles all the synchronizing of Observer objects. **GameObject class** is the base class used in both games. It implements Observer so that all the subclasses from GameObject get "observed" by our GameEvents class. GameObject class takes in x-y coordinates and bufferedimage, since all the objects will have an image associated with it and their x-y coordinates of the object. **Control class** handles key presses for both games and determined the controls of the game. **Game class** is our main class that put all the pieces together. In this class, the game loop is handled as well as any threads that we might use, such as for soundplayer. **Window**

class is used to organized the codes that is used for our JFrame. In this class all the upkeep for the game window is contained.

Tank Game Classes:

Classes that are specific to the Tank Game are Tank class, Bullets class, Wall class, and Powerup class. **Tank class** handles both player tanks. It takes in x-y coordinates, a bufferedimage, and tank ID in the constructor. The tank class also extend GameObject class and implement Observer. In this class, the movement of the tank, firing bullets, collision with other tank, power up effects, and live and health checking is handled. In our **bullets class**, the constructor takes in x-y velocity, damage of the bullet, and the angle of the tank, on top of x-y position coordinates and bufferedimage. In this class, collision between bullet and tank is handled. Our **wall class** constructor takes in x-y coordinates and a bufferedimage. This class handles both breakable and unbreakable walls. This class also handle wall collision with tanks and bullets. **Powerup class** is similar to our breakable wall from our wall class. It handles collision with tanks and the respawn rate of the power up once it is collected.

Super Rainbow Reef Classes:

Classes that are specific to the Super Rainbow Reef game are Player class, Pop class, BigLeg class, Bricks class, SoundPlayer class, MainMenu class, MouseInput class, and TiledMap class. **Player class** handles *Katch's* movement. **Pop class** handles *Pop's* movement from collisions. Players have no real control of *Pop's* movement, instead, the movements are determined by collision between *Pop* and other game objects such as bricks, *BigLegs,* or *Katch*. **BigLeg class** is for creating BigLeg objects. The class constructor take in x-y coordinate and a bufferedimage. **Bricks class** handles all the breakable bricks in the game. We used a hashmap to hold all the bufferedimages, and depending on which key we used to call the bufferedimage of the brick from the constructor, they would be assigned different brick IDs. These brick IDs determine the score of the brick or effect of the brick. **SoundPlayer class** is used for our background music. **MainMenu** and **MouseInput classes**, work together to get our title screen working. MainMenu class display the title page with options while MouseInput class handles the mouse clicks to pick the options. **TiledMap class** is used to read out .txt file of our levels. We used a .txt file with symbols to build out levels. TiledMap class would take in the file name and build the map based on the symbols in the text file.

Team-reflection:

This was one of the toughest projects to get things rolling because it was our first time coding a big project from scratch. One of the hardest and most time consuming part of this project is understanding Java graphics component and how to use them. It took us about 2-3 weeks just to get a tank on a window that moves. We started out with Canvas to get the tank displayed and moving. However, we ran into a bunch of issues with it. One of which was the tank would leave a trail of the image as it moves. From the advice given in class, we decided to move on from Canva and switch to JPanel. Getting the tank to move smoothly and the understanding behind the how was also difficult at the start. Once we got the tank displayed and moving properly, the project progress a lot smoother. Collisions were handled using rectangle at first. However, we found that rotating rectangles, such as the tanks, and using the build in method .intersects() were returning strange results. We decided to switch the hitbox for tank into a circle to make rotations more smoother. Another issue we ran into was split screen/mini map. We ended up using the strategy of building one bufferedimage with all the game component in it and using subimages and scaling to get the split screens and minimap working. Another time consuming part about our first project was playing with the x-y coordinates and trying to display the objects in the correct locations.

For our second game, it was a lot less learning and trying to figure out the Java graphic library since we had already did the tank game. We were able to re use a lot of the code in our first game. We used the same strategy for border checking, collision, and overall class diagram. However, there were still issues that had to face. One of which was collision between *Pop* and the bricks. Instead of breaking one brick and bouncing off it, *Pop* would keep going and breaking all the bricks in the column.

Summary of work:

Naylin: I help code collisions logic and power ups for the tank game. Helped debugged collision glitches on both games. I also help implement bullets and handling of it. Helped implement "zones" for *Katch* for our second game but we ended up not using it. I also helped with high-score keeping and display logic. We met up about 2-3 times a week to work together and talk over issues and how to fix them.

Project Conclusion: