CSC 413 Final Project Documentation

Spring 2020

Greesan Gurumurthy

918737305

CSC413.Section03

https://github.com/csc413-03-spring2020/csc413-tankgame-Greesan.git

https://github.com/csc413-03-spring2020/csc413-secondgame-Greesan.git

Table of Contents

1 Introduction

1.1 Project Overview

In this project, we used our knowledge of Java encapsulation, class structuring, and JFrames to produce two different games. The first of these was a Tank Game, while the second was a version of Koalabr8. Both games are birds-eye view games that involve the animation of various sprites. Many of these sprites had various similarities, in terms of general location, movement patterns, and are associate with other types of sprites. For example, the heart sprites of the Tank game are positioned related to the tank sprite on the screen. To account for such similarities, we were able to create various abstract classes to pull similar methods from. This greatly increases both efficiency of code and memory usage of the game.

1.2 Introduction of the Tank Game (general idea)

The tank game is a player versus player game. Both players start as tanks on opposite sides of a map. The goal of the game is to destroy the opposing player's tank. By pressing preprogrammed keys (Player 1:WASD/space, Player2:arrow keys/enter), the tanks will move around. The keys associated with the up and down movement control the front and back motion of the tanks, while the keys associated with left and right movement rotate the tank to the left and to the right respectively. Each tank has a button that allows it to shoot missiles in the direction the tank is currently facing. There are various stationary objects that provide increased functionality in the game. For example, there are speed powerups, that boost the speed of the tank that collides with it, and there are walls, I also attempted to make a reflection based powerup, but it seems to change the direction of the bouncy missiles incorrectly, so I left it out of the final product. There are also breakable and unbreakable walls, which both hinder the tanks motion when collisions occur. The breakable walls can be destroyed by missiles but cause the colliding missile to explode and therefore disappear. There are also objects like lives and a health bar that are shown right below each tank. Each health bar can take three hits, after which a life is lost. As there are 3 lives per tank, this allows the tank to get shot at 9 times before the tank explodes and the game is lost. As the map is much larger than the screen, the screen is split in half, where the tank is rendered in the middle of each split screen, unless the tank is closer to the corners or edges of the map.

1.3 Introduction of the Second Game (general idea)

Koalabr8 is a player puzzle game where the player must control the koalas on screen in order to maneuver around various obstacles and reach the exit. These obstacles include moving sawblades and TNT. As the koalas are connected mentally, all koalas move at the same time, in the same direction as the key pressed (arrow keys). There are various objects that the koalas can interact with in order to solve the puzzle, including switches/detonators and exits. As of now, there is only one beginner level in the game that includes two koalas, two horizontal sawblades, multiple TNT / dynamite sticks, multiple walls, a switch/detonator, and two exits.

2 Development Environment

2.1 Version of Java Used

Both projects used Java 11.

2.2 IDE Used

I used IntelliJ Idea 2019.

2.3 Special Libraries/Resources

Resources used for tank game were found from images online and also from files provided to us by Professor Souza. The map file made was built using

3 How to Build/Import your Project

3.1 Importing the Games

Git clone from both repos.

3.2 Building the JAR

File-> Project Structure -> Artifacts -> + -> JAR -> Apply -> Build -> Build Artifacts -> Build -> go to out/artifacts. Right click and run the associated game's JAR file.

3.3 Running the Built JAR

Right-click the JAR file in IntelliJ or terminal and run in either the terminal or through IntelliJ.

4 How to Run the Games/Controls

Tank Game:

Open the JAR File to run the game.

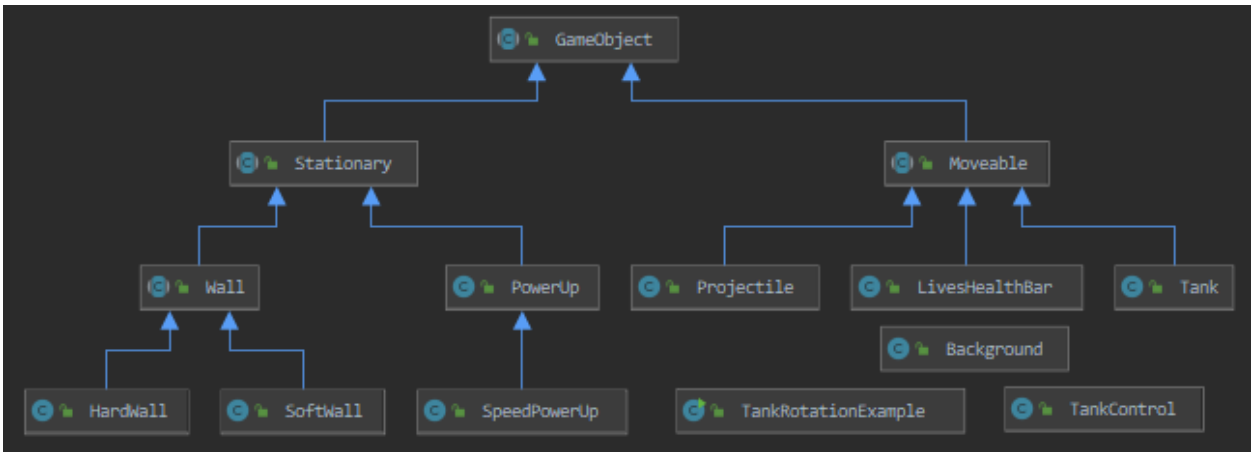
Player 1	Player 2	
Key	Key	Action
W	↑	Move Forward
A	←	Rotate Clockwise
S	↓	Move Backwards
D	→	Rotate Counter-Clockwise
Space	Enter	Shoot Projectiles

Koalabr8 Game:

Open the JAR File to run the game

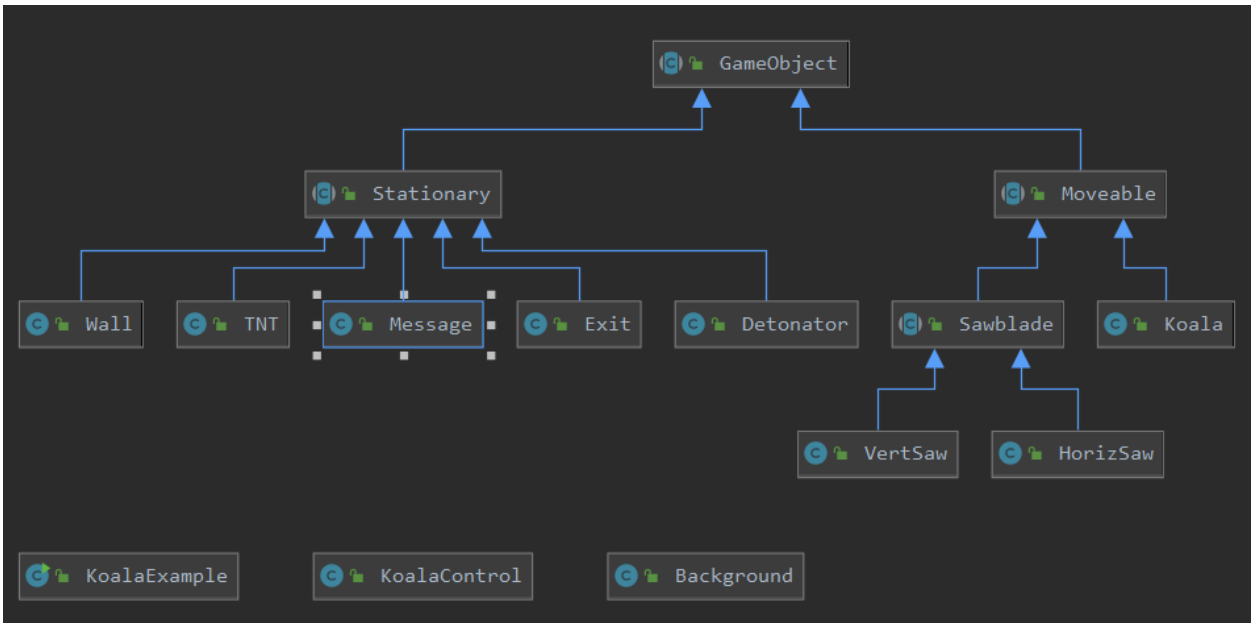
Key	Action
↑	Move Up
←	Move Left
↓	Move Down
→	Move Right

5 Tank Game Class Diagrams



As we can see, most objects in the game are GameObjects. These are split into Stationary objects and Moveable Objects. Walls and PowerUps comprise the Stationary GameObjects, while Projectiles, LivesHealthBar, and Tank make up the Moveable GameObjects. The TankRotationExample creates 2 Tanks, and ArrayLists of Walls and PowerUps based on the map file provided. The Background provides the DrawImage method to draw the background, while the TankControl provides the KeyListener and the methods to uses keyboard keys to control the Tanks.

6 Second Game Class Diagrams



Similar to the Tank Game, most objects in the game are GameObjects. These are split into Stationary objects and Moveable Objects. Walls, TNTs, Messages, Exits, and Detonators comprise the Stationary GameObjects, while Sawblades and Koalas make up the Moveable GameObjects. The KoalaExample

creates ArrayLists of Koalas based on coordinates I provided, and ArrayLists of Walls, TNTs, Saws, and Exits based on the map file provided. The Background provides the DrawImage method to draw the background, while the KoalaControl provides the KeyListener and the methods to uses keyboard keys to control the Koalas.

7 Description of Shared Classes

GameObject (abstract): Classic object of the game. This class implements a ready-to-go drawImage method in the Koalabr8 game, and has variables for object position, visibility in both games. In the koalabr8 game, it also instantiates a hitbox Rectangle object, as most game elements, except for buttons and messages, require some form of collision check. Only the hitboxes that move need to be updates, so the update function is placed in Moveable.

Moveable (abstract): This class extends GameObject. The update function updates hitbox based on the child object's current x and y.

Stationary (abstract): This class extends GameObject. I realized this is a placeholding abstract class in Koalabr8.

Background: This class is its own separate class in both games. This serves as a concrete class that provides the game with a background.

Wall (abstract/also not abstract): This class is abstract for the tank game, but concrete for KoalaBr8. This class extends Stationary. This is a placeholding class as the walls require nothing but hitboxes for collision checking, which is already provided in GameObject. While my implementation of this class in the tank game was not well done, I implemented it better in the Koalabr8 game.

Quick note: While making the Tank Game, I did not organize the drawImage method properly, and therefore implemented it more than once. I learned from this and made the drawImage method a concrete method in gameObject, ensuring that exists in all child classes.

8 Description of Unique Tank Game Classes

TankRotationExample: This class extends JPanel, allowing for the BufferedImages to be seen, and this is the class that is run. It has an init function to set up all GameObjects, and constantly updates all GameObjects throughout its main method. The init function creates ArrayLists for Walls and Powerups. This code also reads from a map file, in order to populate various coordinates with their intended objects, as this is easier to revise, as by changing the map file, we can change the stage layout. It also has a paintComponent method to update the JPanel to display all objects as they move.

Tank: This class extends Moveable and can be used to create a Tank object. It provides collision checking methods for the koala with various other objects. It also allows for movement of the Tank, in accordance with the direction key pressed. This is more complicated than our code for the Koala class, as this also spawns new Projectile objects (stored in an ArrayList) that fires with initial velocity in the direction the shooting Tank is currently facing. To limit the number of Projectile's spawned, I use two Date objects to

ensure that one bullet spawns a certain time after the previous bullet. Also, the tank creates a LivesHealthBar object which is used to change and monitor the Tank's lives and health.

Methods:

Toggle_Pressed / Untoggle_Pressed: This is used by the TankControl class to interact with the KeyListener, to see if an arrow-key is pressed

MoveLeft/Right/Up/Down: This method revises the x and y according to the direction button pressed.

Update: I decided to override GameObject's update, this method checks if a direction key is pressed. If yes, then it calls the associated moveLeft/Right/Up/Down function. If there is no direction press, the koala image to the KoalaStartImage.

CollisionCheck (Obj object): This method checks for collisions with various other objects and performs the correct consequence of collision. If a wall is hit, then stop the motion of the tank in that direction. If another tank is hit, stop the motion of the hitting tank in that direction. If a projectile collides with the tank, the tank loses one health or life. If the tank hits a SpeedPowerUp the speed of the tank and the launch speed of the projectiles increases.

TankControl: This class implements KeyListener, and maps arrow-key presses into tank movement.

Projectile: This class has a CollisionCheck method to check for collisions with SoftWalls. If this collision occurs, the SoftWall and the Projectile disappear, as the SoftWall was broken by the Projectile. It has an update function that changes its x and y based off of its initial launch direction and velocity.

HardWall: This class extends and is a concrete version of Wall.

SoftWall: This class extends and is a concrete version of Wall. The Projectile class has a collision check with SoftWalls in order to break such wall (Wall.visible = false).

LivesHealthBar: This class extends Moveable. It stores health and lives data and organizes the images of lives and the healthbar in an organized fashion.

PowerUp: (Abstract): This class extents Stationary. This is an abstract class that provides methods and variables used by SpeedPowerUp.

SpeedPowerUp: This class extends PowerUp. This is a concrete version of PowerUp.

9 Description of Unique Second Game Classes

TankRotationExample: This class extends JPanel, allowing for the BufferedImages to be seen, and this is the class that is run. It has an init function to set up all GameObjects, and constantly updates all GameObjects throughout its main method. The init function creates ArrayLists for Koalas, TNTs, Sawblades, Walls, and Exits. This code also reads from a map file, in order to populate various coordinates with their intended objects, as this is easier to revise, as by changing the map file, we can

change the stage layout. It also has a paintComponent method to update the JPanel to display all objects as they move.

Koala: This class extends Moveable and can be used to create a Koala object. It provides collision checking methods for the koala with various other objects. It also allows for movement of the Koala, in accordance with the direction key pressed. To create a stuttering walk (could use some improvement), I use two Date objects to ensure that one bullet spawns a certain time after the previous bullet.

Methods:

Toggle_Pressed / Untoggle_Pressed: This is used by the KoalaControl class to interact with the KeyListener, to see if an arrow-key is pressed

MoveLeft/Right/Up/Down: This method changes image of koala and revises x and y according to the direction button pressed.

Update: I decided to override GameObject's update, this method checks if a direction key is pressed. If yes, then it calls the associated moveLeft/Right/Up/Down function. If there is no direction press, the koala image to the KoalaStartImage.

CollisionCheck (Obj object): This method checks for collisions with various other objects and performs the correct consequence of collision. If a wall is hit, then stop the motion of the koala in that direction. If a sawblade is hit by the koala, koala becomes invisible (dies). If an exit is touched by a koala, the koala disappears (escaped).

CollisionCheck(Obj object, ArrayList <TNT> tnts): This is a specific collide check method for detonator. The detonator must detonate the TNT, so we pass in the tnt to allow that specific gameObject's visibility to be set to false, thereby detonating the TNT.

KoalaControl: This class implements KeyListener, and maps arrow-key presses into koala movement.

Message: This class extends Stationary. This is a concrete version of the stationary abstraction and is used to show restart, title and end images.

Detonator: This class extends Stationary. This is a concrete version of the stationary abstraction and is used to explode the TNT when a Koala object collides with the detonator.

TNT: This object extends Stationary. This is a concrete version of stationary abstraction.

Sawblade: This abstract class extends Moveable. This is an abstract class that provides methods and variables used by Vertblade and Horizblade.

VertSaw: This class extends Sawblade. The parent class's update version is overridden. It still calls the parent class's update function, but also forces the motion of the saw vertically based on the object's input parameters (v, maxY, and dir).

HorizSaw: This class extends Sawblade. The parent class's update version is overridden. It still calls the parent class's update function, but also forces the motion of the saw horizontally based on the object's input parameters (v, maxX, and dir).

Exit: This class extends Stationary. This is a concrete version of the Stationary abstraction.

10 Reflection on Development Process

I liked this project. I liked how the first project set up the basis for the second project, and that we got control over the second one. When I started this class, I was doubtful of my ability to organize my code more efficiently, but because of this project, I feel like this is something I have become much better at. The videos provided were an immense help on this project, That being said, I got stuck on creating the JAR file both times around for a matter of hours. After doing more research on manifests and the jar format, I found my mistakes. Considering the free reign over the second project, I was able to play around more with the code, and felt like I created some impressive features, like the moving sawblades, and the detonator that activates the TNT to explode.

11 Project Conclusion

I would say this was a koala-ty project, pun intended. I was victorious in creating two different games. When I started, I had a hard time finding out how to organize this project, but the videos were very helpful to provide a basis to start creating the general outline of my code. I felt like I understood the organization of my code a lot better for my second game and am very proud of the result of both games. As a result of this project, I was able to learn a lot more about class organization, and how to write efficient code, while creating a few fun games.