Term Project Documentation

Tank and Galactic Mail Games

By Johnny Huynh & Adan Guanlao

Github Repository: https://github.com/sfsu-csc-413-spring-2018/term-project-team-g/

Introduction

Our goals for this project were to design and implement two games, namely Tank and Galactic Mail, while working as paired teams. The two games would teach us to be more mindful of object oriented coding and organization. The Tank game would serve as sort of a guideline in approaching the design of Galactic Mail, since many classes from Tank would be reused later on and be refactored into the commons package.

Technical Work

The most important aspect of designing our games was to be mindful of class reusability. We had to bear in mind that most of our code should not be rewritten when possible. Our reusable classes consisted of GameObject, CollidableObject, NonCollidableObject, GameClock, Explosion, RotatablePolygon, and Sprite. Once we were confident in those classes, we could continue building Tank game and its various components. A bulk of our work came from working with JPanel and retesting the tank's collision with indestructible walls.

Development Environment

NetBeans 8.2 and Eclipse Oxygen 4.7.2

Java 8 & 9

Command Line Instructions

We compiled and executed from ../term-project-team-g/src/

Tank Game Compilation & Execution

javac tankgame/TankGame.java

java tankgame.TankGame

Galactic Mail Compilation & Execution

javac galacticmail/GalacticMail.java

java galacticmail.GalacticMail

Scope of Work

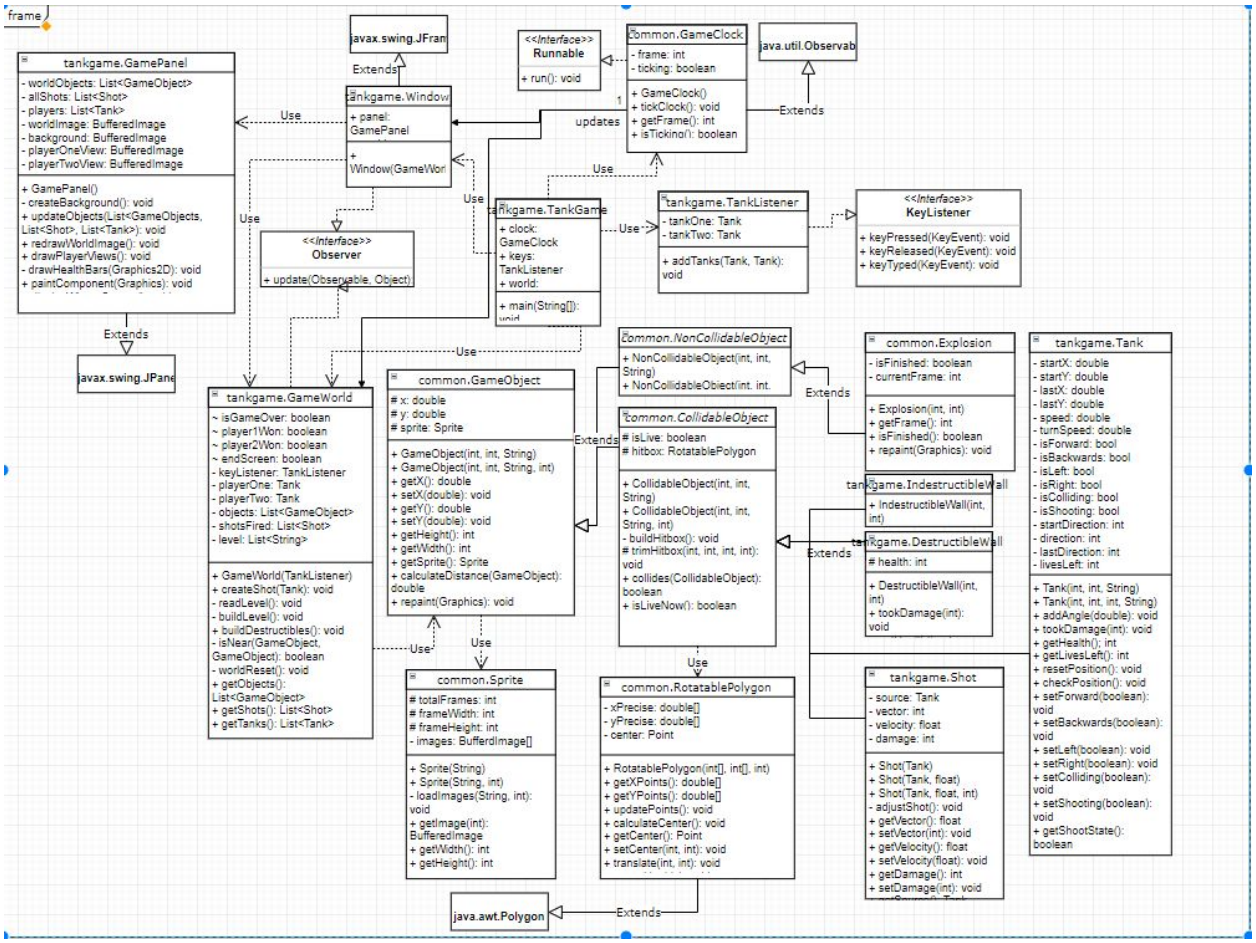
For the Tank game, we needed to design a world where objects could be created and destroyed depending on the circumstances, i.e. creating projectiles, destroying walls, emulating an explosion. To do so, GameObject and Sprite were made. We also needed to display dual perspectives of the players, so GamePanel was implemented. It might have taken the most time to build upon since we had trouble learning how to import images correctly and setting up split player perspectives. Completing the game's collision logic was also of great importance. We believe that we had applied collisions without the tanks nor projectiles looking weird when colliding with walls. The sprites of each object created/destroyed were working as expected, as well as the tanks turning accordingly. There were .mp3's of game sounds that should have been added, but we were unable to do so.

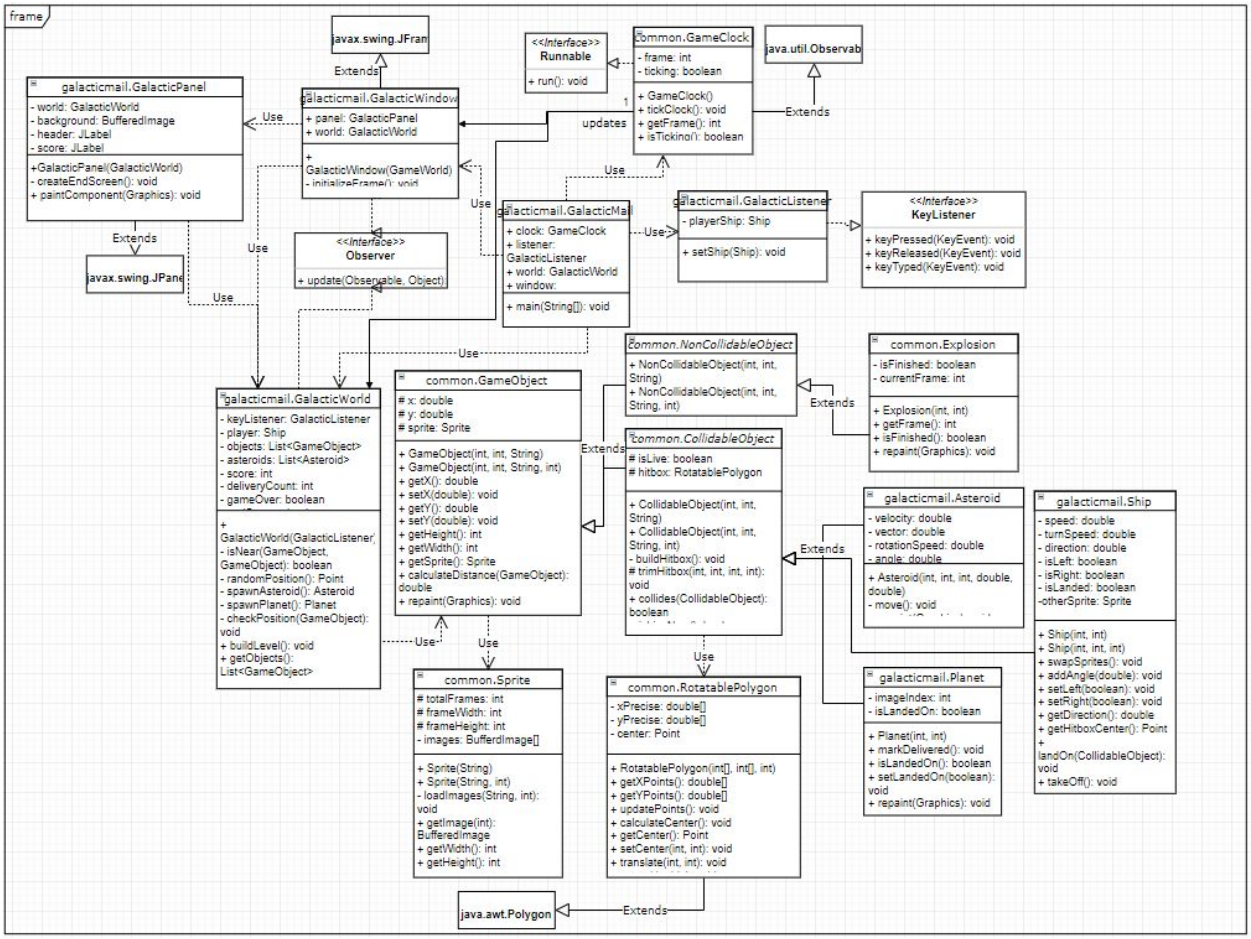
As for the Galactic Mail game, we needed to implement a wrap around effect for the spaceship and asteroids. The ship would behave similarly to the tank, but be unable to move backwards, would launch forward in a direction by pressing spacebar, and appear on the opposite side of the screen after going heading offscreen.

Assumptions

Everything stated in the prompt was straight forward. We assumed that anything not specified by the project's prompts were up for us to decide to implement or not. So, in Galactic Mail, we adjusted the game's difficulty to increase as more landings were secured. A certain number of landings would spawn more asteroids that the ship needed to avoid.

Implementation Discussion





Implementation Decisions

The Observer/Observable pattern was used in both games with the GameClock being observed by the game world and game window. On the clock tick, the world is notified to update it's object positions and to check for collisions. Additionally, the window is notified to update the screen by redrawing the game objects.

We knew most of our objects needed to be created and presented visually in the game world, so we purposely attached Sprite to GameObject. Objects were distinguished between objects with hitboxes and collision effects (CollidableObject) and objects without collision (NonCollidableObject). GameObject and its immediate subclasses were designed with reusability in mind.

Each game's panel class either had access to its game's world object or to the world's list of objects to allow for drawing the objects to the panel.

Code Organization

We simply divided up our packages into three categories of common, tank, and galacticmail, based on which classes were utilised in which game or both. Our most important class for reuse was GameObject, as more than half of our project's classes stemmed from it.

Conclusion

We seemed to have quite a few complications with the Eclipse IDE. There were times when everything supposively synced correctly, yet Eclipse would rename directories or set to build incorrect paths. This caused a lot of confusion and lost time spent on working on fixing these issues. We learned that utilising the same environment would be much more ideal for future projects to avoid such implications, as well as properly planning and hitting milestones on time. In the Tank game, we found the tank's collision to be tricky to deal with, since there were times that it was expected to stop when confronted by wall, yet sometimes pass through anyways. To fix it, we needed to change the way its flags were set and behaved when the player was holding down a direction key. As for Galactic Mail, maintaining an object's collision box while wrapping around the screen proved to be a challenge. Most of the time the boxes would not follow the object as expected at all. It was tricky to create objects without them getting too close to already present objects. Overall, a troubling aspect of the project was being unable to make all the classes reusable.