Amey Vaze

CSC 413.02, Fall 2019

Introduction:

The purpose of this project was to demonstrate competence in object-oriented programming by developing games in the Java programming language.

The task at hand was developing a simple game in which two players control a tank each, and attempt to destroy the other player in a classical last-man-standing format, by either shooting them down with bullets, or by ramming into each other. Each player must do so in a finite arena, littered with obstacles that are to be avoided, as well as upgrades that can either rejuvenate the players’ tanks or increase the damage and speed of their bullets. Players can keep track of these objects, as well their own health and life counts in the center of the window; at the top is a shared minimap, and below that, a status screen showing the number of lives left per player, as well as a rough estimate of the health left per life (shown as a healthbar, rather than showing the number of health points directly).

Development Environment:

The Java version used for this project was Java 15.0.1, and the IDE used was IntelliJ IDEA Ultimate 2019.3. The project was built off the provided TankRotationExample project on iLearn; as such, credit goes to Anthony Souza for creating the Tank, TankControl, TRE, EndGamePanel, StartMenuPanel, Game Constants and Launcher classes which were later modified, as well as for sourcing the tank1 png. No other special libraries or resources were used.

Importing and Building Instructions:

Since the project is to be turned in using a zip file, and not a GitHub repository, importing the project is as simple as unzipping the folder, extracting the project folder to the end-user’s local directory/folder of choice, opening IntelliJ, going to the top-left and clicking File>Open, and then navigating to the directory the project folder was placed in.

In order to build the JAR, while the project is loaded in IntelliJ, click File>Project Structure, then once the Project Structure pop-up menu is opened, click Artifacts, the “+” symbol near the top-left, JAR>From modules with dependencies (note that JAR is simply hovered over and not clicked). Once that is clicked, the Create JAR from Modules menu will be opened; in Main Class, type TankGame\_ASV.Launcher, while in Directory for META-INF/MANIFEST.MF, make sure what is written in the text box is the current location of the src folder of the project. Please note that if the META-INF folder and MANIFEST.MF file are already present, they will need to be deleted before another JAR can be created. Furthermore, if a JAR structure is already present in the Artifacts menu in IntelliJ, it is recommended to delete the structure, for simplicity’s sake. Once that is done, click OK in the bottom-right, and the menu will close. Go back to the top, click Build>Build Artifacts, hover over the JAR structure that was just created (if the above instructions were followed, it should be the only structure in the menu), and click Rebuild. The newly built JAR will be in the Compliation\_Outputs/artifacts/TankGame\_ASV\_jar directory.

Finally, to run the game, as noted in the readme.txt, the JAR and layout1.txt files can be placed in whichever local directory the end-user wants (as long as they can access that directory through the command-line). Once the files are in the directory of choice, open the command-line, navigate to the directory, and enter the command:

java -jar TankGame\_ASV.jar

From there, the game will open.

Rules & Controls:

This is a two-player game, where each player controls a tank that can shoot bullets. Each player starts with 3 lives, and 10 health bars per live. Player 1 uses WASD to move/turn their tank around and SPACE to shoot; Player 2 uses the arrow keys and ENTER to do the same for their tank. Health packs and bullet upgrades are scattered across the map that can be used to restore lost health (but NOT lives) and to enhance bullets, respectively. Furthermore, walls outline the perimeter and are scattered across the interior of the map; if you collide with them, you lose health points. However, some walls are more fragile than others. Collide with or shoot these weaker walls enough, and they will break, allowing you to pass through where they once stood. Finally, the locations of all these objects can be altered to the end-user’s choosing by editing layout1.txt. More information on how to do so can be found in the readme.txt file.

The objective of the game is simple; shoot the other player until they are out of health and lives, and be the last player standing.

Assumptions Made in Design and Implementation:

When it came to game design, it was assumed that any future players would be able to WASD and arrow keyboard controls to move, and to be able to understand the last-man-standing goal of the game. In implementation, it was assumed players would have some level of file system and command-line knowledge, to be able to access the correct directory, and be able to input a command to run the game.

Tank Game Class Diagram:

Original:



Final:

A picture containing text

Description automatically generated

A stand-alone version of this diagram can be found in the same folder.

Class Descriptions of Shared Classes:

Camera: Provides a birds-eye of the player. Each Camera centers on a specified player and moves alongside the player.

CollisionDetector: Determines if a collision has occurred between two GameObject instances (for example, a Tank and a Bullet).

GameObject: A superclass from which all derivative GameObject classes inherit from (Tanks, Bullets, Walls, PowerUps).

Minimap: Provides a miniaturized display of the locations of all significant GameObjects in a given level (for example, the positions of Tanks, Walls and Powerups, but not Bullets).

PowerUp: Interactable GameObjects that augment the player character (weapon upgrades, health packs etc.).

Score: Displays characteristics relevant to the player (health, lives etc.).

Bullet: GameObject subclass; damage-dealing, linear-trajectory projectile object that is spawned from Tank objects whenever the corresponding TankControl object receives a “shoot” input. 2 variants shown ingame.

Tank: GameObject subclass that is controlled by the player(s). Each instance has 3 lives, 100 health per life, and various

TankControl: Listener class that used by TRE to allow players to control Tank objects.

TRE: Game loop class. Creates all GameObject subclass instances, and all instances necessary to play the game (Minimap, CollisionDetector, Score, Cameras etc.). Determines when the game has ended, and when it has, jumps to EndGamePanel. If “Restart Game” was clicked on EndGamePanel, then TRE resets all internal parameters to what they were at initial creation.

Wall: Stationary GameObject subclass. Acts as a barrier that other GameObject classes can’t pass through.

EndGamePanel: Class that only displays when one player loses all their lives and health. Offers 2 options: Restart game or close game.

StartMenuPanel: Acts as a “Title Screen”. Allows players to start the game when the button is clicked.

GameConstants: Contains dimension values for game window size. Can be expanded upon to include other values currently in other classes should future editors deem them to appropriate for the class.

Launcher: Main Class. Creates StartGamePanel, EndGamePanel, and TRE object instances using values derived from GameConstants class. The TRE instance in particular is displayed through a “splitscreen” JPanel, with a CardLayout to organize the placement of Cameras, Minimap and Score.

Self-Reflection:

It goes without saying that I highly underestimated the effort needed to get this project to work. While time was lost due to unforeseen circumstances, time was equally lost in my overall lack of motivation in finishing the project itself; however, in the end, I was finally able to complete the game, and soon after complete the second, using the Tank Game as a template. At the very least, completing the project did give me some assurance in my ability to see such a project the whole way through.

Conclusion:

In creating the Tank Game, a certain competency in OOP is displayed, using the Java programming language as a medium. The game developed, on its surface, appears no different to the many browser games resembling it available on the World Wide Web, if somewhat toned down in their scope, due to assumed constraints in knowledge and time. The game contains the necessary components for local multiplayer gameplay, indicators for game knowledge that are vital for players, and is in general, not overly complicated to launch and play the game.