TANKS

Team Exception

CSc 335

Introduction

This tanks project was performed by Ryan Fitzgerald, Liu Liu, Andreko Lennartz, and Christian. The goal of this project was to design a feast paced action game that implements key and mouse features in order to provide a more full set of controls for the user to interact with the game. Additionally, this Tanks game will include a campaign option in order that a single user plays an AI enemy tank or a PVP option in which the user may connect to another computer and play against another user. Tanks will include as wide array of obstacles that the tank may encounter including mobile and immobile barriers. There will also be multiple terrain and, depending on the terrain, will influence the speed at which the tank will accelerate. These are just a few of the features that this Tanks mini game will contain.

Design

Early on in the design stages of this game, Team Exception had desired to use a similar layout to that of the Javaboy program as had been done in class in the sense that the world and/or battlefield would be a grid of tiles. However, there was one big issue that had to be resolved: collisions. If we would have decided to stick with the grid system as in Javaboy, we would have to calculate each and every possible way each and every object could intersect another object. After careful thought and advice, it was decided to use Java’s geometry class which allows us to run a “sub GUI” underneath the GUI visible to the user and keeps track of all the objects seen to the user as a series of geometrical shapes. Therefore, each object (that had a visible form on the GUI seen by the user) had a corresponding rectangle class object that would represent the object for collisions. When these shapes intersect or collide, we can make the proper method calls in order what to do next rather than calculate its location on the map and then make appropriate method calls. This was the most significant design change in the entire project. This was chosen for its simplicity and speed. If we would have tried to calculate everything out manually, then the lag of the game would have probably been horrendous.

The maps that were made for the campaigns or PVP were essentially pre designed maps that were separate classes. When single player mode was opened, the “Level1” class map was used as an argument to construct the map for the level 1 stage. When this level was complete the next pre made map class (“Level2”) would be used as argument to create the next stage. All these levels extend a super class called “Map”. This design was used because creating random maps for one might produce maps that are unplayable in the sense that there are too many obstacles to maneuver around to get to the enemy tank and therefore the level can’t be completed. Secondly, having controlled pre-designed levels allow for the campaign to become increasing more difficult (to some degree of course since difficulty sometimes is subjective but the point still stands).

The moving and aiming for the tank were performed by key and mouse listeners. The movement of the tank was done by pressing keys. In this case up, down, left, and right movements of the tank are done by pressing the w, s, a, and d keys respectively! The tank will visibly change direction on the GUI as well. The shooting was done by implementing the mouse listener in Java. The basic concept behind the calculations was be determining the location where the mouse was clicked and how far away the mouse was clicked from where the tank itself was that is doing the shooting. By calculating the distance, a best fit trajectory path for the bullet to follow (which is resolved into x and y coordinates) can be calculated. When the animation for the bullet is to start, that thread can used these trajectory coordinates to move across the screen in the appropriate direction. Additionally, only one bullet may be fired at a time so there is an observer that determines whether the current bullet has been removed from the field or not in order that a second bullet is fired. These listeners were used essentially because they were required but also allow the user of the game more flexibility in controls rather than using only the keypad.

The obstacles in this game extend an obstacle class; the obstacles that were included were the crate, spike pit, TNT, fire ring, and immovable object (so terrain barrier). The crate is simple an obstacle that can be moved and destroyed. The immovable object is simply an object that can neither be moved nor destroyed. The other three obstacles, the spike pit, TNT, and fire ring, are all for extra credit. The spike pit basically will destroy the tank if it moves across it. The TNT basically when shot will explode and deal damage to a surrounding radius, essentially killing any tanks nearby. The fire ring is a roaming troll obstacle that will destroy any tanks that it will collide into. If it comes into contact with a crate or TNT box, the fire ring’s AI will prevent it from moving into that object. So essentially the fire ring is smart enough to not blow itself up or destroy everything on the map and will only move into a space it is empty or occupied by only the tank. The factory design pattern that was used for the items when constructed upon placement on the map is straightforward and allows for many reproductions of the similar items without crashing Eclipse or slowing it down.

The terrain was a relatively simple extra credit aspect of the project. Essentially, there are three different terrains: ice, grass, and sand. Each is its own class and contains only one property which is the speed the tank will accelerate over such terrain. In our Tanks, ice moves the fastest while sand is the slowest. When the map is created it will also tell the tank what it’ speed and therefore how fast it will be moving on that level. The only exception is the ice block on the ice levels. When the tank hits one of these ice blocks, its speed will temporarily be reduced to 0 for a short period of time. This was a simple design that added another interesting element to the game.

In campaign mode, the user will have three lives to complete five levels. On the right side of the screen, the user can see the lives he has currently, the current level he is on, and the items that are visible on the field. This was an additional add on to the Tanks game.

In the Lan view, when host starts to create a new game, the screen is going to lag until an other player is connected. We designed it this way because when trying to use threading, Stream corrupt exceptions were thrown. The design for hitting ready – start button is to make sure both client and host are ready for the game. (of course the host has to start hosting first). So when both sides are connected, the client is able to hit the ready button, and then the host's start button will be enabled. Once both players agree to start the game, the game view will be changed to the networkTankView.

In networkTankView, the user will have unlimited lives to compete with a remote Internet user. However, Whoever first get to the 5 kills will win the game, and the game will return to the title view and reset the connection for both players. One thing to note is that for both players, Green is the player himself, and red is the enemy. The shooting update for enemytank is updated by receiving a simpler class with 4 integers that creats the projectile. The shooting update for itself is by networkcontroller where it controls all the collisions.

Updating of enemy move is done by sending String of movements(e.g “up”, “down”..) to the other computer because Strings are easier to be transmitted, and that it makes the multiplayer game running smoothly. The map design is based on campain mode level 1, without the items. The panel on the right of the screen is based on the campaign mode, but instead, has a score board on it. As stated, whoever gets the 5 kills first will win the game.

We took a long time trying to figure out what the enemy tank should be. We ended up implementing it locally from PlayerTank, but remotely from EnemyTank. We did not make enemy tank to take damage from spikes or TNTs so we used a switch to turn the AI feature on and off – by adding a identification integer to the enemy tank to specify if its an AI or not.

Exception Handling: we check for disconnections from either client or host , so whenever the client disconnected or the host disconnected the program will report and jump back to the LAN view.