Spencer Moran

12/2/12

Homework 4 Design Document

**Purpose/Overview**

This program will implement a menu to control the flow of the SpaceWars game. From the menu, a user account can be created and deleted, users can be sorted by high score, and the user can also login to access the game itself and to perform a number of other functions. The program will load the database from a default filename, and then when the program is exited, it will print the results back into the same file.

**Requirements**

The game must have a functional menu in some type of user interface. It must load properly from a database file, and write properly to a file. A user must be able to log on, and the program must correctly list all usernames. A user must be able to be added to the user list, and the deletion of users must also function properly (including the deletion of dangling friends). Similarly, the game must add and delete friends correctly. It must be able to sort both all users and all of a user’s friends by their high score. The game must also properly implement the shortest path algorithm from one user to another.

**Instructions**

All the files are contained in one .zip folder. Extract the folder to a target location. In a UNIX shell, go to the directory /homework3 which you extracted. Type these three commands, in this order: qmake –project, qmake, and then make. To run the file, enter ./homework3 and the game will open. There are many includes needed to make this game work. Among them are stdlib.h, QPainter, QApplication, QDesktopWidget, time.h, QRect, QImage, and list.

**Classes**

I used two different classes to implement the menu driven side of the program. The first is a User class, which contains all the variables associated with a User. This class has a constructor and a destructor, but no functions. The Database class contains the list of users for the program. It also has variables to track the maximum number of users, the highest id found when reading the file, and the number of relationships between the users of the game. It has many functions to control the list of users. It has functions to add and delete users, to add and delete friends, to sort by high score, to sort a user’s friend by high score, to write to the file, to print the list of user names to the screen, and to find the minimum path from one user to another. It also has functions to read from the data file and validate all the information in the file (including that all relationships are bidirectional), as well as functions to update the high score for a user. It also has a number of getters and setters.

This game has many different classes. The non-derived classes are SpaceWars, Bullet, Badguy, Spaceship, PowerUp, LinkedList, and Node. There are also many derived classes. SpaceWars is the class that controls the function of the game. It contains a number of functions, flags, counters, and timers to control the flow of the program through Qt. It also has two lists which contain the active bullets fired by both the spaceship and the badguys, and another LinkedList of badguys. The class also has many functions, most notably the paintEvent, timerEvent, and keyPressEvent which are the three functions that control the flow of the program and paint the images onto the screen. The paintEvent function controls what gets drawn on the screen. The timerEvent function contains all of the timers that govern when certain activities occur (i.e. when a badguy shoots a bullet, when a new set of badguys spawn, etc.). The keyPressEvent function determines what occurs when the player presses a button that is meant to perform an action. For instance, if the player presses ‘w’, the spaceship should move up on the screen. There are also functions that control the movement of both types of bullets. There are also different functions which detect and report the occurrence of the many different collision types in the game. The class also contains functions to start, pause, and restart the game, as well as a victory function that alerts the player when the victory condition has been met. The game ends in the paintEvent function if the player runs out of allotted lives.

The bullet class is the abstract class that controls the bullets that are shot by the badguys and the player controlled spaceship. The bullet class has eight derived classes: GoodBullet, BadBullet, BossBullet1-5, and AimBullet. The GoodBullet class controls the bullets that are shot from the player’s spaceship, and the BadBullet controls the bullets that are shot from the badguys. The AimBullet class controls the bullets shot by the boss of the first level, and the BossBullet1-5 classes control the 5 bullets that the end boss shoots. Each of these classes has a constructor, destructor, and a move function. The constructor initializes the appropriate image for the class, and the move function moves the bullet in the appropriate direction on the screen.

The badguy class is the abstract class that controls the different types of enemies that appear onscreen. The badguy class has two overloaded create functions. One creates the initial badguy and takes no parameters, and the second one takes an integer parameter (the type of the first badguy) and then creates the remaining badguys. Both functions return a pointer to a derived class. The badguy class also has a number of other functions, and getters and setters. The badguy class also has eight derived classes: B1-B5, and Boss-Boss3. The B1-B5 classes control the spaceships that the player fights during each level on the way to the boss. The Boss-Boss3 classes are responsible for handling the end bosses of each level. Each of the classes B1-B5 each has a move, setLocation, and setFirstLocation function. The setFirstLocation function determines the starting coordinates of the first badguy to spawn. The setLocation function then uses those coordinates to spawn additional badguys in the appropriate places. The move function controls the movement of each badguy, depending on both its type and its spawn location. The Boss-Boss3 classes all have a move, setLocation, setFirstLocation, and changeImage function. The setLocation function determines where the boss will spawn, and the move function moves it in the desired directions. The changeImage function is used to control the images of the boss, to simulate the health bar above the boss’s image. This function gets called whenever the boss is hit by a bullet or bomb, and the appropriate image is then loaded.

The Spaceship class is used to create and manage the player-controlled spaceship. It has a resetState function, to move the spaceship to a constant, initial location, and a move function for moving the spaceship up, down, left, and right. It also has a QRect and QImage to paint the spaceship onto the screen. In the constructor, the image for the spaceship is loaded and then resetState is called.

The PowerUp class is the abstract class that controls the powerups that scroll across the screen and help or hurt the player. The PowerUp class has many functions that are used to control the PowerUps, including one virtual function and one pure virtual function, move and trigger respectively. It also has a create function similar to the Badguys create function which controls the creation of a derived type by returning a pointer to a randomly selected derived class. The PowerUp class has six derived classes: ExtraLife, LostLife, Bomb, ScoreBonus, ScoreMinus, and RandomUp. Each of these classes is a different type of PowerUp. These classes all have a constructor, a move, and a trigger function. The constructor uses a clever way to set a string to control which image is displayed. The move function moves all the powerups on the screen, and the trigger function controls the outcome when the player collides with a powerup. Each type has a different trigger.

The last two classes in the program control the linkedlist which holds all the badguys and bosses for the program. The first one, Node, holds the data and has a pointer to both the previous and next Nodes in the linked list. The second class, LinkedList, has many functions, including getFirst, getNext, and remove. The getFirst function returns the first Node in the list, and the getNext function returns the next node in the list from the one passed as a parameter. There are two overloads of the remove function, one to remove the first node in the list, and the other to remove the node that is passed to the function as a parameter.

**Global Data/Functions**

The program has many variables that are created and initialized within the scope of if statements or functions, but the program has no global variables or global functions. The only functions that are outside one of the classes are the functions in the Qt.cpp file where the main function is. The purpose of these functions is to handle the initializing of the window that the game is then run inside.

**High-Level Architecture**

The menu will be written in the main function, and will be displayed in the console. The main will need to pass the high score to the game window so that the high score can be determined and returned. There will be an option in the menu to play the game, and so the window must be initialized and started from that option in the case statement. There will be another menu inside the top level menu when a user is logged on.

Qt will loop in an infinite loop, and the timers in the SpaceWars class will control the program’s flow. The timerEvent function in the SpaceWars class will determine what events occur for each of the different timers, and then the paintEvent function will draw the appropriate images onto the screen. The keyPressEvent function determines what occurs when the player presses a key, including the pause function and the restart after the player loses a life. Within the SpaceWars class, polymorphism and inheritance is used in the creation and management of all the badguys, powerups, and bullets.

**User Interface**

The menu will be displayed in the console, and the game will be run from there. The interface uses the Qt library to draw the images onto the screen. The player’s spaceship will start in the lower center area of the window, and once the game begins, the player will control where the spaceship moves in the 800 by 700 window. The player cannot move his or her spaceship outside the edges of the window. The badguys can exist in a small area outside of the window, to create a more realistic feel when the badguys fly onto the screen rather than just appearing in the window. The player uses the ‘w’, ‘a’, ‘s’, and ‘d’ keys to move, and the space bar key to shoot. The game will stop when a level change is reached, and the player will have to press the ‘m’ key to begin the new level. In each level, there will be a different background image that the player is fighting in front of.

**Test Cases**

I tested the menu and the reading from file extensively, testing every possibility and flaw I could find. In each case, the validation worked and correctly printed an error message to the console. I also verified that the high score was being retrieved correctly from the game, and I made sure that the game recorded the high score properly even if the user reset the game and the score got reset to zero as a result.

I tested my game thoroughly through a number of test cases. All the collision types were tested, as well as all winning and losing conditions. I tested all the spawn locations for the badguys, and all the movement conditions for each as well. All of the bullets moved in the intended paths as well. In all cases, the expected outcome occurred.