Chad Martin
CSCI – 102L
Fall 2012
11/26/12

Homework 4 Design Document
Original Author of the Game: Spencer Moran

Requirements

- **Your application must have functional a menu-drive to allow the user to select which action(s) they would like to perform that meets all the requirements specified in the hw4 assignment description.**
- **You must use a C++ class to represent a user.**
- **You must validate data read from a database file and when a user is added according to the requirements specified in the hw4 assignment description.**
    - You must not allow duplicate user IDs.
    - You must not allow duplicate usernames.
    - If validation failed when a user is added, you should prompt the user again for a correct value (and you should give a reasonable error message to let the user know why the previous input was rejected).
    - Before you load a database file, you should clear out all the users information from your in-memory data structures. If validation failed when a database file is being loaded, you should clear out all the users information that you have processed from your in-memory data structures.
- **You must not allow a friendship to be formed to a non-existing user.**
- **You must provide the user a way to cleanly terminate your application.**
- **Shortest-path to the logged on user from another user.**
- This program must be playable by the user, and the user must be able to accomplish some task to achieve points.
- The user must control some cursor and be able to move it side to side.
- There must be different moving things represented by images on the screen that the player is able to interact with in some way to score points.
- The player must be able to win the game by accomplishing a victory condition, and must also be able to lose the game if some condition is not met.
- The player’s score, number of lives remaining, and current level must be shown in the game window.
- The game must be created using the Qt library.
- There must be at least 2 of "things" that shoot back at you.
    - The four different types of bosses all shoot back at your location via homing bullets.
- There must be at least 1 of "things" that follows you.
    - The 2 types of shields follow the player.
    - The 3 types of power ups that yield additional points will follow the player.
- There must be at least 12 different types of "things" that are implemented using a linked list.

    - There are 15 different types of power ups each representing a different bonus for the user some are positive bonuses and others are negative.
- There must be at least 3 different levels with different backgrounds and different (or at least some different) "things".
    - There are 12 different levels. As the levels increase, the hit points and chance to fire of the badguys increases. Throughout the last 3 levels, the movement speed and rate of fire of the badguys increases. Every 3 levels the movement algorithm changes.

Purpose/Overview

The purpose of this program is to create add features to an existing, playable game that utilizes the Qt library with a menu driven GUI to perform actions specified in the hw4 assignment description. Multiple users can be created, high scores can be compared, and information is saved in a database file. The player controls a spaceship and tries to shoot down the enemy spaceships, while also avoiding them. The player wins the game if he or she beats level 12, and has 3 lives with which to accomplish this objective. To beat a level, the player must destroy all enemies on the screen. Every time the player's spaceship is struck by an opponent's bullet or ship, the player will lose a life. The game ends when the player has 0 lives.

Instructions

In order to compile this program,

- extract files from ChadMart_HW4.zip
- navigate to the `/HW4/ directory
- enter "qmake -project" in terminal
- enter "qmake" in terminal
- enter "make" in terminal

In order to run this program,

- enter "./hw4" in terminal

Program library dependencies

- QT library
    - <QDesktopWidget>
    - <QApplication>
    - <QWidget>
    - <QKeyEvent>
    - <QImage>
    - <QRect>
    - <QPainter>
    - <QLineEdit>
    - <QFormLayout>

    - <QtCore>
    - <QtGui>
    - <QListView>
    - <QTableView>
    - <QTableWidget>
    - <QPushButton>
    - <QTextStream>
- C Standard General Utilities Library
    - <stdlib.h>
- STL Library
    - <vector>
    - <list>
    - <queue>
- Streams
    - <fstream>
    - <sstream>

Classes

The game has many different classes. The most important ones are Bullet, Spaceship, Badguy, PowerUps, Spacewars, and User. The Bullet class is the class that controls the bullets that both the player, the badguys, and the bosses shoot. There are three derived classes, GoodBullet, BadBullet, and BossBullet, which are used to control where the bullets are created and in what direction they move. The Bullet class has functions to move the bullet, determine whether or not the bullet is out of bounds, and to move the bullet down on the screen. It also has a QRect and QImage to draw it on the screen and control its size and location. The constructor of the GoodBullet, BadBullet, and BossBullet classes initialize the location of the bullets and the image for each bullet, and then the move functions move the bullet in the necessary direction.

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 Badguy class is the class which manages the non-player controlled spaceships which attempt to shoot the player-controlled spaceship. There are a lot of classes which derive from the Badguy class, such as B1, B2, B3, B4, and Boss1. The Badguy class has three bool variables which are flags for whether or not it is active, whether or not it exploding, and whether or not it exploded. It also has a getter and setter function for each of these variables. It also has a function called offScreen which returns true if the badguys are out of bounds, and it has a virtual move function to control the movement of the badguys. The Badguy class also contains functions to get the image and rect, and it also has a QRect and QImage for painting into the window. Each derived class has a constructor which loads the correct image depending on which type of badguy is created, and a move function that moves the badguy accordingly. The B1, B2, B3, B4 subclasses that each have derived classes of their own like B1b and B4C. These subclasses have increased health and inherit the movement algorithm from their base class.

The PowerUps class is the class which manages the power ups that drop out of dead badguys and boss bullets. There are 15 classes which derive from the PowerUp class, such as P1, P2, P3, P4, and so on. The PowerUp class has one bool variable which is a flag for whether or not it is active. It also has a getter and setter function for this variable. It also has a function called offScreen which returns true if the PowerUps are out of bounds, and it has a virtual move function to control the movement of the PowerUps. The PowerUps class also contains functions to get the image and rect, and it also has a QRect and QImage for painting into the window. Each derived class has a constructor which loads the correct image depending on which type of PowerUp is created, and a move function that moves the PowerUp accordingly. The PowerUps are implemented using a linked list.

The User class is the class that represents a single user. The User class contains members that store information on the user's ID, user name, year at USC, friendships, and high score. Friendships are stored in an adjacency list based of a vector of other user's IDs. The User class has getter and setter functions for each of the member variables. There are no getter or setter functions for the friends vector so the Spacewars class must be a friend of the User class. The Spacewars class creates a vector of users to run the requested algorithms on.

The Spacewars class is the class that drives the program. It contains a number of checks, flags, counters, timers, signals, and slots to control the flow of the program through Qt. It also has three lists which contain the active bullets fired by the spaceship, the badguys, and the bosses, and it has a 2D array of badguys. The class also has many functions, most notably the paintEvent, timerEvent, keyPressEvent, and keyReleaseEvent 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, how fast the badguys explode etc.). The keyPressEvent and keyReleaseEvent functions determine what occurs when the player presses a button that is meant to perform an action. For instance, if the player presses ‘d’, the spaceship should move right on the screen. There are also five functions that control the movement of the types of bullets and powerups. There are also nine different functions which detect and report the occurrence of the nine 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 different screens are controlled by different display flags which are set through key presses and button signals. The vector of users is populated with the information from DB.txt at the start of the game. In the destructor the vector of users and their information is written out to DB.txt. The shortest path between 2 users is calculated with the BFS search function to assign levels to each user. When the users causes the high score screen to be displayed, the scores are sorted using a bubble sort algorithm. A style sheet and a QPallete has been added to the different widgets to match the aesthetic of the game.

Global Data/Functions

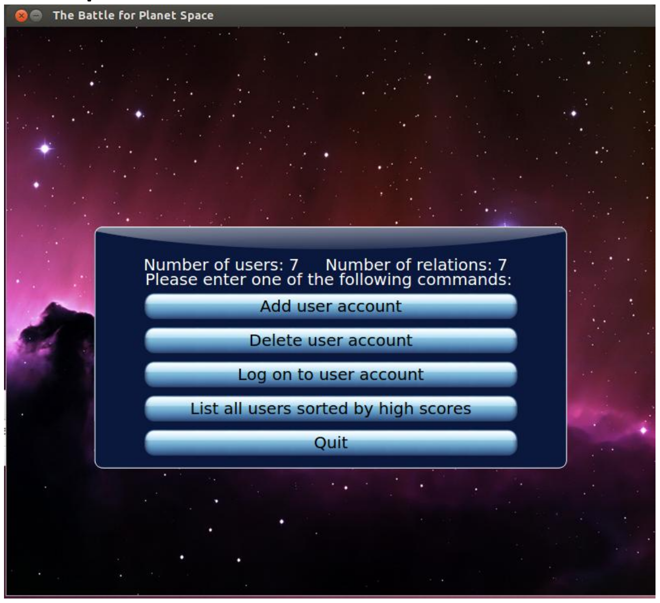
The program has many variables that are created and initialized within the scope of if statements or functions, but the program has no global variables or global functions. Originally, the program was created with an enumerated type to determine which type of badguy was going to be created, but then more options and movement patterns were added to the badguys, so it became more efficient to use inheritance and derived classes. This also held true when the new spawn locations for the badguys were added.

High-Level Architecture

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 and keyReleaseEvent functions determine what occurs when the player presses a key, including the pause function. Signals and slots are used to handle the button presses and list clicks. The different menus are controlled by Boolean flags that follow the naming convention display**Screen. The user database is stored in DB.txt. It is read in during the SpaceWars class constructor, and rewritten out during the destructor. When the database is read in, It is checked for errors, and if the data is valid a vector of users is populated.

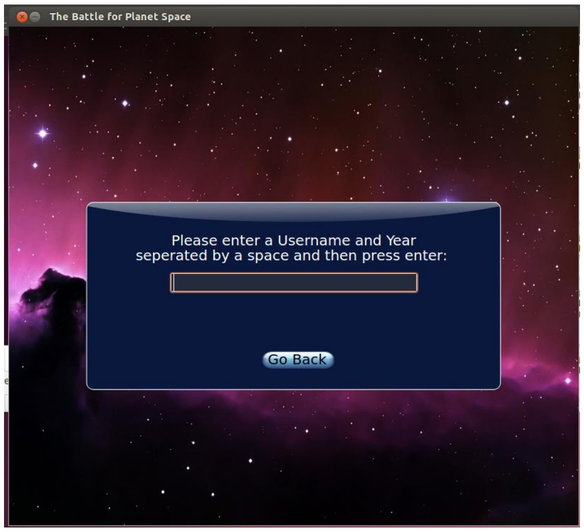
User Interface

- **Menu Interface**
    - **Start up Screen**


The Battle for Planet Space
Add user account
Delete user account
Log on to user account
List all users sorted by high scores
Quit

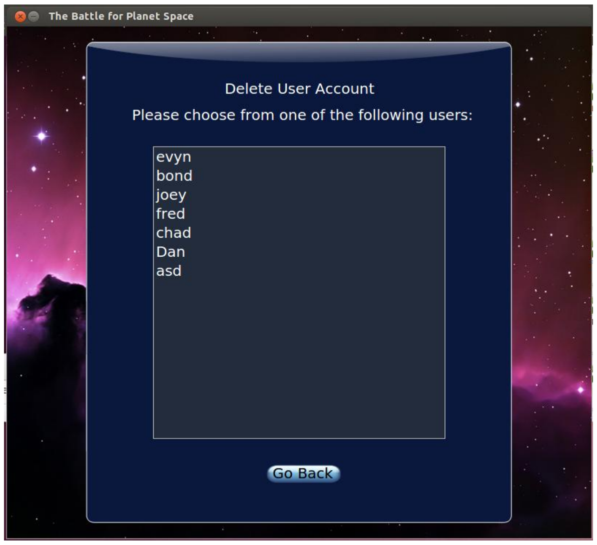
Number of users: 7 Number of relations: 7
Please enter one of the following commands:

- **Add User Screen**



Please enter a Username and Year
seperated by a space and then press enter:

- **Delete User Screen**

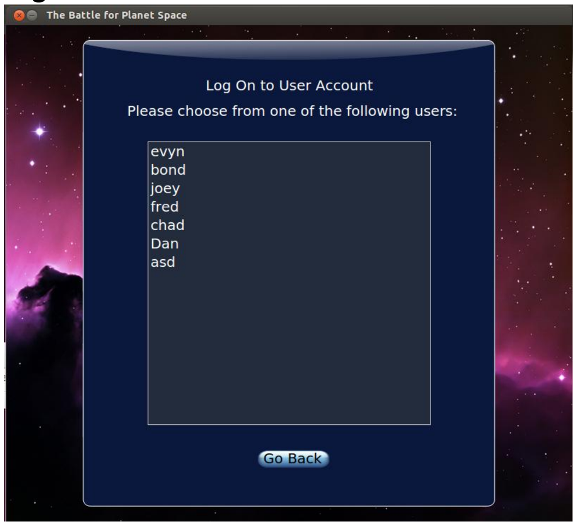


Delete User Account

Please choose from one of the following users:

```
evyn
bond
joey
fred
chad
Dan
asd
```

- **Log On User Screen**

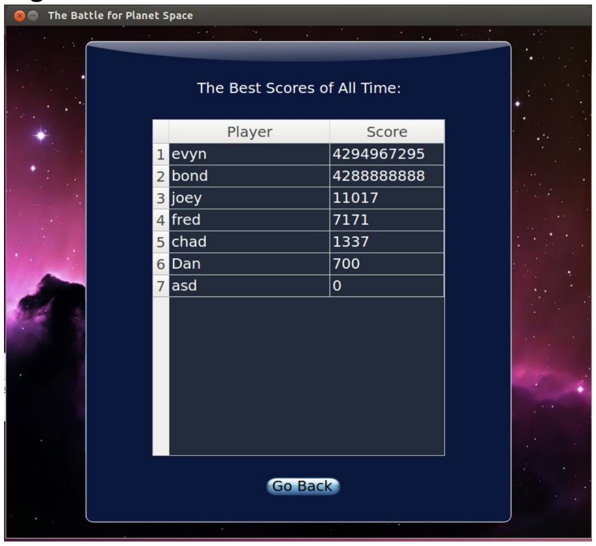


Log On to User Account

Please choose from one of the following users:

```
evyn
bond
joey
fred
chad
Dan
asd
```

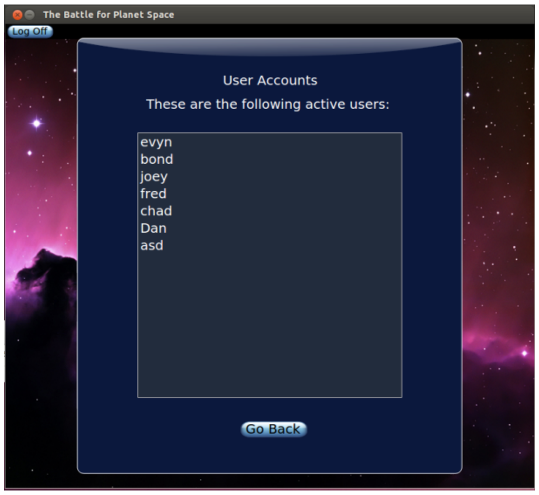
- **High Score Screen**



The Best Scores of All Time:

	Player	Score
1	evyn	4294967295
2	bond	4288888888
3	joey	11017
4	fred	7171
5	chad	1337
6	Dan	700
7	asd	0

- **List all User Screen**

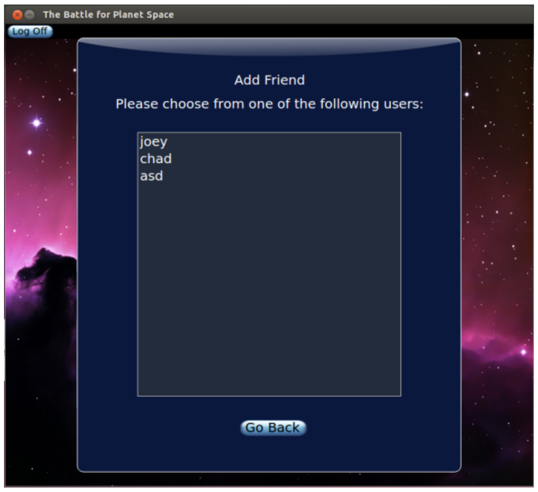


User Accounts

These are the following active users:

evyn
bond
joey
fred
chad
Dan
asd

- **Add Friend Screen**

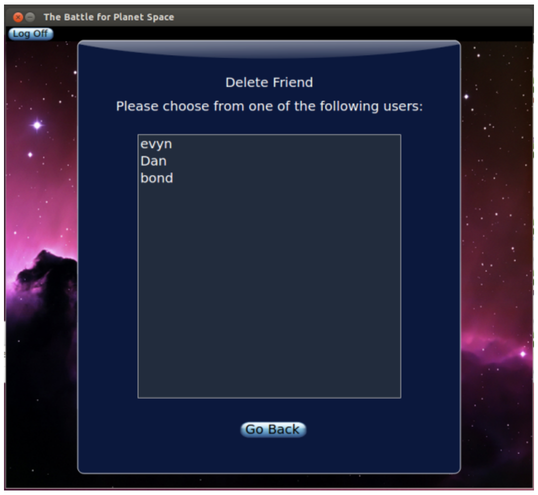


Add Friend

Please choose from one of the following users:

joey
chad
asd

- **Delete Friend Screen**

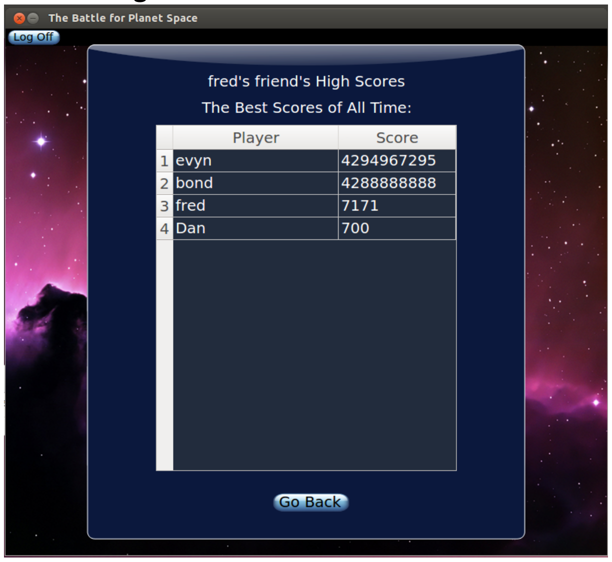


Delete Friend

Please choose from one of the following users:

evyn
Dan
bond

- **Friend’s High Score Screen**

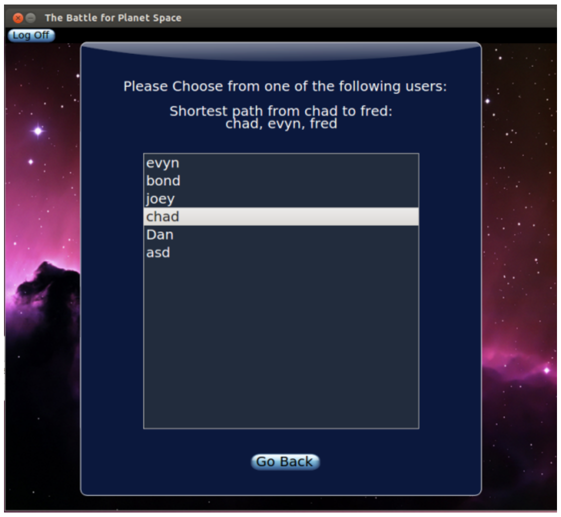


fred's friend's High Scores

The Best Scores of All Time:

	Player	Score
1	evyn	4294967295
2	bond	4288888888
3	fred	7171
4	Dan	700

- **Shortest Path Screen**

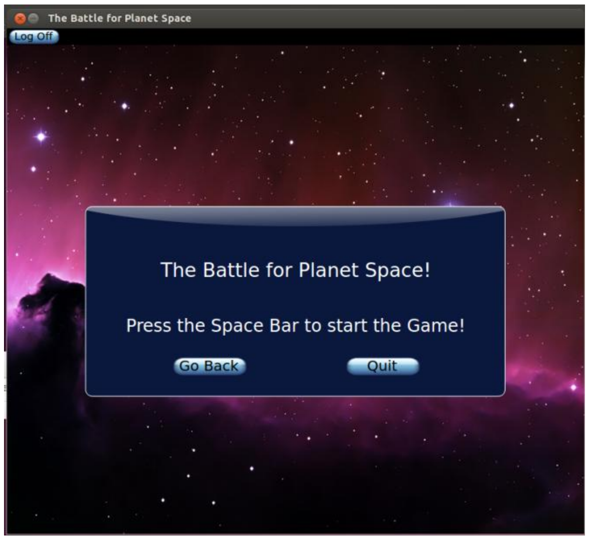


Please Choose from one of the following users:

Shortest path from chad to fred:
chad, evyn, fred

evyn
bond
joey
chad
Dan
asd

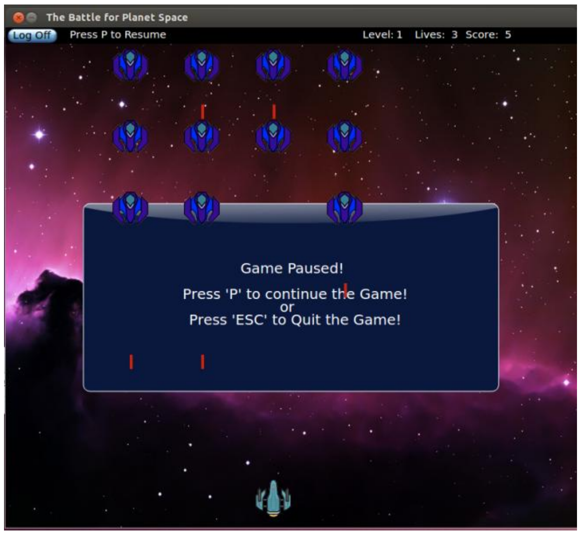
- **Intro Screen**



The Battle for Planet Space!

Press the Space Bar to start the Game!

- **Paused Screen**

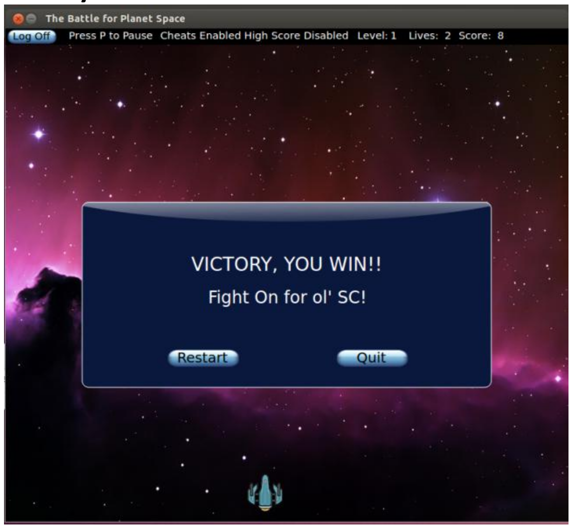


Press P to Resume Level: 1 Lives: 3 Score: 5

Game Paused!

Press 'P' to continue the Game!
or
Press 'ESC' to Quit the Game!

- **Victory Screen**

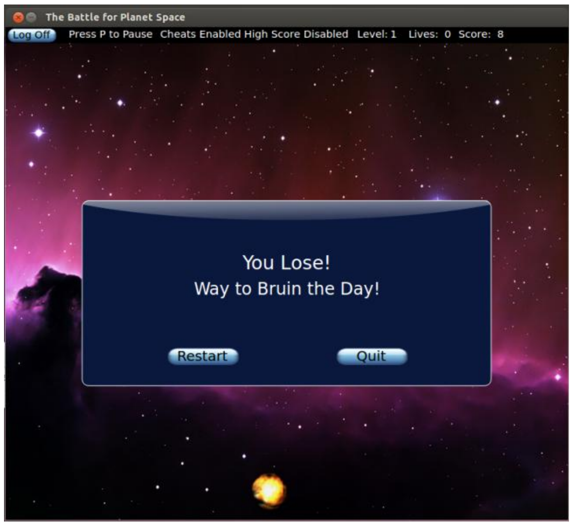


Press P to Pause Cheats Enabled High Score Disabled Level: 1 Lives: 2 Score: 8

VICTORY, YOU WIN!!

Fight On for ol' SC!

- **Game Over Screen**


The Battle for Planet Space

Press P to Pause Cheats Enabled High Score Disabled Level: 1 Lives: 0 Score: 8

You Lose!
Way to Bruin the Day!

- **In Game Interface**

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 'a', left arrow, right arrow, and 'd' keys to move, and the space bar key to shoot.

- **Controls**
  - Press Space Bar – Shoot
  - Press Left Arrow – Move Left
  - Press Right Arrow – Move Right
  - Press 'R" – Restart Game
  - Press Space Bar or 'M' to Start Game
  - Press ';' to enable cheat codes
  - Press 'T' – toggle God Mode
  - Start Up Screen
    - Press '+' go to add user screen
    - Press '-' go to delete user screen
    - Press 'L' go to Log in screen
    - Press 'H' go to High Scores screen
    - Press 'Q' quit application

- Users Screen
  - Press ‘A’ go to list all users screen
  - Press ‘F’ go to add friendship screen
  - Press ‘D’ go to delete friendship screen
  - Press ‘S’ go to Friends high scores user screen
  - Press ‘H’ go to High Scores screen
  - Press ‘B’ go to shortest path screen
  - Press ‘P’ go to Game intro screen

- **Power Ups**

Extra Life	3 Extra Lives	-1 Life	Double Total Points	Lose All Points
Victory	+10 Points	+100 points	Temp Freeze Enemies	Level Up
Level Down	Game Over	Temp God Mode	Temp Shield	Triple Shot

- **Cheats**
  - Press ‘;’ to enable cheat codes
  - Press ‘T’ – toggle God Mode
  - Press ‘1’ – spawn extra life
  - Press ‘2’ – spawn 3 extra lives
  - Press ‘3’ – spawn deathshroom
  - Press ‘4’ – spawn double points
  - Press ‘5’ – spawn lose all points
  - Press ‘6’ – spawn victory POW
  - Press ‘7’ – spawn +10 points
  - Press ‘8’ – spawn +100 points
  - Press ‘9’ – spawn Freeze
  - Press ‘0’ – spawn level up
  - Press Minus – spawn level down
  - Press Equal – spawn game over Skull
  - Press Backspace – spawn Temp God Mode
  - Press Backslash – spawn Temp Shield
  - Press BracketRight – spawn Triple Shot

Test Cases

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. The menu was thoroughly tested by trying every button combination. The propagation of the high score tables and user lists were tested before and after adding/deleting users/friendships. The user counts and relationship counts were both tested before and after adding/deleting users/friendships. The shortest path algorithm was tested using a variety of user graphs. In all cases, the expected outcome occurred.