Design Document HW 4

Philip Lee

Purpose/Overview

The purpose of this homework assignment is to implement a game using Qt. The point of the game is to get the user character, a frog, past a river of fireballs. Each time the frog successfully crosses the river, a new fireball is added for difficulty. The player has three lives to attempt to get as high a score as possible. Movement is taken care of by the WASD keys. ESC can be used to exit the game at any time.

Requirements

This project has to be created with QT based off of homework assignment 3A. Images have to be used to represent “things” in the game.

Classes

Game – Class that holds the setup and main components of the game. Contains the code for the graphics, menus, and interactions. Has checks for logic and game rules.

1. Game(QWidget) – Takes in parent widget from main(). Sets up the basic variables including lives, score, side the user is on, title of window, and GUI components (title, buttons, and score/lives counter).
2. createEnemy() – Called upon to create and implement an enemy sprite and attach it to vector of enemy units in use. Uses functions provided in enemy.cpp in order to do so.
3. startClicked() – Called when a new game is starting. Ensures initial score and life count are 0 and 3 respectively. Hides menu components and brings back life/score counters. Resets the character to the starting position as well as reset the enemy vector to empty and re-add the basic starting five. Also, calls startTimer() for QTimerEvents.
4. restartMenu() – Called after game ends. Hides the GUI for the actual game and shows the buttons for end or reset game.
5. checkEvent() – Main logic component of the game. Contains collision check between user and enemy sprites. If a collision is detected, the function reduces the life count by one, resets the character to starting position, and makes the character invincible for 2 seconds to account for lag. If no collision is detected, the second portion of the function checks if the character has successfully crossed the river and accordingly adjusts the score.
6. timerEvent(QTimerEvent) – Function in charge of timer events. Every 20 milliseconds, the function is called. Auto moves the enemy sprites in a simple path. Calls on checkEvent() to check for collision in that period. Also calls on repaint() which sends a paint event.
7. paintEvent(QPaintEvent) – Function that is called each time timerEvent() calls repaint(). Checks if the game is over. If so, paints the restart menu instead of next frame of game. If game is not over, draws enemy and user characters on screen.
8. keyPressEvent(QKeyEvent) – Function that is called each time user presses the keyboard. Checks to see if WASD movement keys are pressed. ESC key can also be used to exit the game.

Char – Class that takes care of the user character. Contains coordinate variables for the user. Also contains functions to manipulate the coordinates.

1. Char() - When first called, creates a QRect with the character image of a frog and sets it off screen to hide it. Coordinates are set at starting location.
2. reset() – Function to reset and move the character sprite to starting location.
3. move() – When called, the user sprite is moved to stored coordinates.
4. getXDir() – Returns an int containing the x coordinate of the user.
5. getYDir() – Returns an int containing the y coordinate of the user.
6. setXDir(int) – Takes in an int, verifies the int is within boundaries, and sets the x coordinate to the given int.
7. setYDir(int) - Takes in an int, verifies the int is within boundaries, and sets the y coordinate to the given int.
8. getRect() – Returns the QRect used as a placeholder for the character sprite.
9. getImage() – Returns the QImage used to store the sprite designated for the user character.

Enemy – Class used to take care of the enemy characters. Contains coordinates and direction the sprites are in/moving in.

1. Enemy(int, int) – Creates an enemy object with given x and y int coordinates. Loads the fireball picture and sets it offscreen for now. Also provides a random direction (left/right) that the fireball will move in.
2. move() – When called, the enemy sprite is moved to stored coordinates.
3. hide() – Hides the enemy sprite off screen for future use.
4. getXDir(), getYDir() – Same as Char class. Returns x, y coordinates as form of int.
5. setXDir(int), setYDir(int) – Same as Char class function. Sets x, y coordinates to give int with boundary checks.
6. getDirection() – Returns int for direction the enemy sprite will move in. (0 = left, 1 = right)
7. getRect() – Same as Char class function.
8. getImage() – Same as Char class function.

Global Data

The QWidget created in the main() function will be provided to the Game class for use. There are no other global variables.

High-Level Architecture

The main.cpp will create a QApplication and QWidget to house the game. The Game class will be called and given the QWidget. The Game class will then control most of the application till the Game class exits. The Game class will first reset the game to the initial menu. The user is provided with short instructions and a start button. When the button is clicked, startClicked() will be called and the actual game will be put into motion. Enemy and user sprites will be placed on screen. Controls with the keyboard as well as the game timer are started. When the game is over, Game class will call upon the end game screen for reset or close.

User Interface

The user interface is a GUI provided by QT through QWidget. The graphics are small 20x20 .png’s loaded into the game. Only keyboard is used besides when mouse is used to click on buttons.

Test Cases

1. Confirmed the game started with correct setup of score, lives, and enemy/user characters when I hit the start button on first run.
2. Test the movements of the user sprite for collisions in top left/right and bottom left/right corners with other enemy sprites.
3. Test the boundaries to make sure nothing will escape the viewable screen.
4. Test creation of new enemy sprites when user scores and confirm newly created enemy sprites interact with user character.
5. Verify game over is detected correctly and new game can be started.

Problems:

1. Unable to determine why the game creates 3 times as many enemy sprites at unusual speeds when the reset is called after the initial game over.