Joshua Garner

S1424311

Computer Games Software Development

I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award.

Signature

Jgarner

Introduction

This document will outline the code that is used to generate the game within visual studio. This will include the different classes required and the methods within those classes.

The majority of the code being explained will be within main.cpp although will make reference to code out with that file. The explanation will include all methods called and functions used, as well as the internal commentary found within the code itself.

Main.cpp Explanation

Files included within main.cpp

The main.cpp file includes the following header files; windows.h, windowOGL.h, GameConstants.h, cWNDManager.h, cInputMgr.h, cSprite.h, cBkGround.h, cFontMgr.h, mazeMaker.h, cFileHandler.h, cButton.h, main.h and cSound.h.

Initialization of the window

The first tasked handled by the program is the initialization of the window’s dimensions and the bits per pixel of the window, each of these values are stored as integer values. The program then creates an object for the window and creates managers for the input, sound and fonts. Then OpenGL is attached to the window object as well as the Input manager, the program then attempts to create the window used the given dimensions and properties that were assigned to it. If this is unsuccessful an error message is printed using a MessageBox method and the window manager resets the display and exits the program. Once the window is successfully created the OpenGL window is then created with the window dimensions provided earlier and is then attached to the program window.

Maps, Textures, Sounds and Fonts

The key buffers are cleared and then two maps are initialized as 2D arrays, map and winmap. Map is the basic map that will be displayed to players and will also be updated as the game progresses. Winmap is used to store a map that already has the win condition fulfilled and will be used for comparison later. After this an array is declared which holds the fonts for the game. Following this is an array that holds all the png files which represent each piece the player will be able to see and use during the game. There are two more arrays declared here for textures and music respectfully, these contain the music and background that will appear in the background of each scene. An array similar to the one used to hold the backgrounds in created in order to hold the textures for the game’s buttons.

Instantiation and Creation of Game Objects

The program then starts to instantiate objects for the textures that are stored in the array textureBkgList; it will set the sprite position using the setSpritePos method. It then assigns the texture to the object using setTexture and finally uses that texture to determine the dimensions of the object by using textureBkgList[3]->getTWidth() and textureBkgList[3]->getTHeight() respectively.

We then have to create an object for the tile map that will be displayed on screen. A input manager is attached to this object and its position is declared. This map will use the texturesToUse array to populate the map with sprites in the appropriate positions. The same is done for roadmap. We then do the same for winmap except no input manager is attached or positions given as this map isn’t interacted with by the player or displayed in the window. tileToDrag is also instantiated in the same way expect it is also assigned textures and texture dimensions.

The same processes used to instantiate the background textures are also used to instantiate the buttons that are used to navigate the game. These buttons include play, exit, load and save.

Handling Files

The program needs to call a file that holds the blank map that players will build upon. The file is called using cFileHandler theFile ("Data/blankmap.dat"). A string is the created to hold the map data. An array of strings is created and will be displayed during the appropriate game states. We then attempt to open the file we just called. If this fails an error message will be displayed and the program will be exited, if there is no error then mapData will be populated with the correct data so that once the game is run the appropriate sprites will be displayed. This process is repeated for the winmap so that it can be used to compare the two maps later on.

Menus, Game States and Gameplay

When the game is started the game is set to the MENU game state. The menu state informs players of the game’s name and controls, it also provides a play button, so that players may begin playing the game, there is also an exit button so that the player may end the program. The buttons positions are set using setSpritePos and they are then rendered using the render () method. In order for the play button to change game states the line: theGameState = playButton.update (theGameState, PLAYING) must be used.

Players that choose to play the game by pressing the play button, the game will move into the PLAYING game state. This is where the gameplay actually takes place. The program begins by rendering and updating the map and tileToDrag. It then renders and positions the save button, using saveButton.update ("Data/usermap.dat", tileMap.getMap () the file holding the player’s progress is updated and is called upon re-entering the game state. In order to determine if the player has won, an if statement is used to compare the players map and the winmap every frame. The maps are converted to strings and compared as two strings would be. If the two maps match then the player is awarded 1 score and the program changes the game state to WIN. It was attempts to erase the user data so that the game can be replayed. Again an exit button is provided to the player which will change the game state to END where the player may exit the game.

The WIN game state is entered once the player completes the puzzle correctly. It is comprised of a background, victory text and two buttons. The load button re-enters the PLAYING game state so that players may replay the level. The Exit button will exit the program when pressed.

The only other game state available to the player is END; this is accessed when the player presses the exit button during the game. In this state the player can either use the load button to return to the game with their progress intact. They may also use the Exit button to end the program.

Clean up

The OpenGL window is shut down and any resources are released and then the program window is destroyed.

References

All of the class and files associated of this were taken from the Lab work provide and then modified to suit my game.

All music was taken from Freesound.com.

All background images were taken from google image search, all appropriate credit goes to their original creators.