Code Explanation: Main.cpp

The main.cpp script serves to outline the application’s core functionality. The header files for the component classes required to carry out the program’s tasks must first be referred to. These include GameConstants and GameResources, which define some of the more customisable aspects of the game such as the font and window icon. Operations concerning the physical appearance of the game are handled with cD3DXManager, cD3DXSpriteMgr, cD3DXTexture, and cSprite. More specific subclasses of cSprite include cStephen and cEnemy, which handle the player character and the obstacle characters, respectively. Lastly, cXAudio and cD3DXFont are incorporated in order to deal with the functions related to sound and score display. The standard namespace will be used for this program.

The next stage in the project’s implementation is to declare all of the global variables. To start, a series of Boolean variables is defined. Game tells the program whether or not the player is in one of the levels, while gameWater, gameLightning, etc. tell the program which level the player is currently in. A Boolean variable for music determines whether or not the background music is playing. Next, a global handle to hold the application when it runs is created, followed by a global variable to hold the window. The DirectX manager and sprite renderer must be explicitly referred to in order to make use of their functionality, otherwise we would not be able to see any of the game’s assets. Before the player character, Stephen, is first initialised, a floating point value of 0 is given for his rotation, ensuring that he will be upright upon spawning. His starting co-ordinates are contained within a vector, as well as the scaling of his sprite. The enemies’ position is also contained in a vector. The size of the window is in a rect container, so that the enemies can be created around the central point of the window without the need for precise calculation on the developer’s part. The enemies themselves are contained in a vector, with iterators George and Index set up for later use. Another Boolean is created to hold information on whether or not Stephen has destroyed an enemy, and his total kill count is held in an integer value for the program’s use, and a character value for display. Stephen himself is created as a list with the iterator Pete, rather than a vector as the changes that affect the enemies are more dynamic than any changes Stephen may possibly go through. Audio objects for the background music and kill sound effect are created as well, with a surface and a background image for the main menu completing the roster of global variables.

The window procedure method checks for available input messages from the queue, in order to determine if a user has pressed a key that corresponds to action in the game. If the player presses the left arrow key, the program checks if Stephen’s position on the x axis does not exceed the left-hand side of the window. If there is sufficient space, Stephen’s sprite will be moved ten units to the left and the loop will return a value of 0 to confirm that there are no errors within the loop. The same process is similar in the opposite direction for moving to the right, although the right-hand bound is increased by 90 so that the front of Stephen’s sprite is stopped against the side of the window, rather than his back. The game offers seven different backgrounds, so if the player presses a number from 1 to 7, the program will set every one of the level-related Booleans to false with the exception of the Boolean that corresponds to the chosen background. For example, if 1 is pressed, gameWater will be true along with the game Boolean that states that gameplay is underway to begin with. Every other scene Boolean (gameFire, gameLightning, etc), will be set to false. Pressing 8 and 9 do similar things, only instead of continuing gameplay, they shut off all of the Booleans except those for the game over and start screens, respectively. Next, if the space bar is pressed and the game Boolean is true, the iterator for the enemy vector will be sent to the beginning of the vector. As long as there is still at least one enemy left for the iterator George to point to, the program will check whether or not that enemy is intersecting with Stephen’s position when the space bar is pressed. If there is an intersection detected, the enemy at George’s position will be eliminated and the total number of killed enemies will be increased by one. This will be reflected in the score counter. If, however, there is no intersection, George will move on to the next enemy in the vector. Regardless of whether or not there was a hit, a sound clip will be played, representing Stephen Stills’ battle cry. Finally, if the window was closed or otherwise destroyed, the application will quit running. No matter what happens, the message will be returned to the default window procedure after it has been executed.