

Computer Games Development SE607

Technical Design Document

Year IV

[Seán Whelan]

[C00250016]

[26-04-2023]

**DECLARATION**

﻿

**Work submitted for assessment which does not include this declaration will not be assessed.**

* I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
* I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
* I have provided a complete bibliography of all works and sources used in the preparation of this submission.
* I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) SEÁN WHELAN

Student Number(s): C00250016

Signature(s): Seán Whelan

Date: 26 April 2023

--------------------------------------------------------------------------------------------------------

# Contents

**Technical Design**

**Architecture**

Overview Diagram

**Menu and Navigation**

Initial Menu

Path from high level design to low level changes

**Game Creation**

Game options

Game Size, Name and Background

Game Type

Toolbar

Rubber, Brush and Fill tools

Choicebar

Placement

Testing

**Storage and Saving**

Saving Game

Loading Game

Adding Game to database

Deleting Game

Downloading from database

**Gameplay**

Player

HUD

Clock, Dynamite Inventory, Health bar

Powerups

Nuclear Bomb, Invincibility, Invisibility

Enemies

Behavior

Seek, Attack, Wander

Objectives

Win case

Loss Case

Items

Dynamite

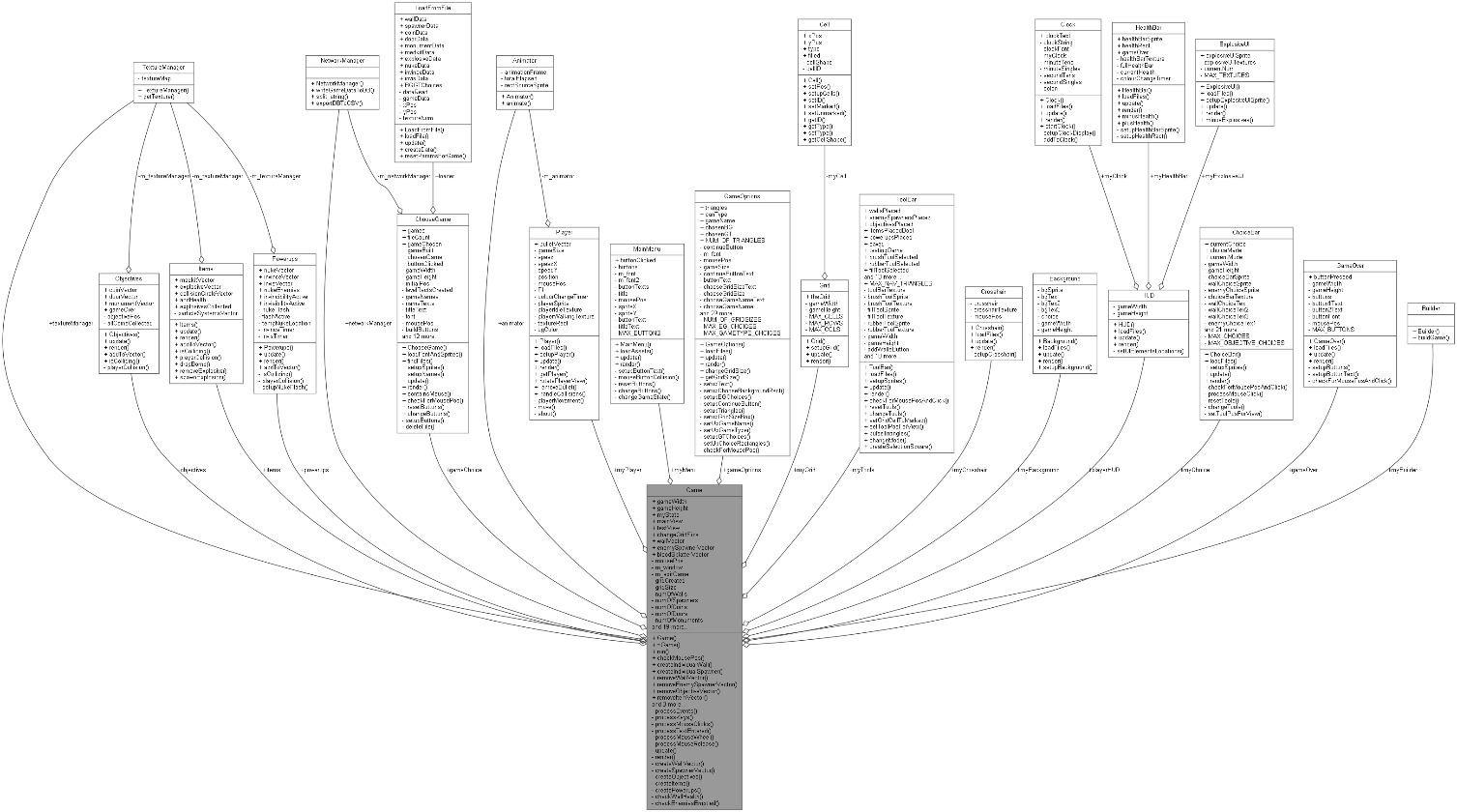
Particle System

Med kit

**Technical Design**

Architecture

Overview Diagram



**Menu and Navigation**

Initial Menu

The Menu of SlayerMaker is simple yet sleek making it intuitive. The Menu is 4 buttons consisting of “Create”, “Play”, “Options” and “Exit”. When the player hovers a mouse over one of the buttons both the text and the button itself increase in size. When the user moves the mouse away again the button goes back to normal. This is standard on almost all of the buttons in SlayerMaker. When the player clicks the button it brings them to that screen by changing the game state, an enum class. Depending on the different gamestate, the update and render methods update and render different objects. Everything clickable has some feedback to let the user know they are A) hovering over it and B) they have clicked it.

Path from high level design to low level changes

The creation of a game starts at a high-level view in which the menu consists of big decisions about how the game is going to play and look. As you continue through the creation process using the large, animated arrows to the top left and right of the screen the changes you are making to your game are smaller and smaller. As an example, you start with game type aka, what form your game will take, then you move to walls, your map creation fast forward, you end by placing down small amounts of powerups and items that the player of your game can find. This allows the flow of game creation to be easy to understand, as not all options are on screen at once, as well as intuitive

**Game Creation**

Game Options

Game options is the first screen you see when you decide to create a game. Here you make most if not all of the main high-level decisions. These include the template game type, the size of the grid in which the game will be built, the name of the game and the background.

Game size

The game size is the size of the grid in which the placement of objects is done. Technically this dictates the overall maximum game objects you can place but also the amount of space your game will take up in the game world. The spread from minimum to maximum game size is 30 by 30 cells to 70 by 70 cells (900 – 4900 cells). Each cell is 30 by 30 pixels in size.

Name

The name of the game is entered via keyboard input by the user. The name of the game is final, and is also what the game is known as in the code after it as has been added. Additionally, when being saved to a .csv and then uploaded to the gamedata Database, the csv and table will be named after the games name. Lastly when choosing a game to play on the play screen, the games will show their name.

Background

The overall background of the game changes the ambiance of the level completely. This appears behind all of the game objects. This appears in the GUI when you choose your background and when testing or playing the game the background fills up all of the playable space and more.

Game type

To make the development of a game easy 3 game templates which you can deviate from are supplied. They change the objective of the game as well as the enemy behavior slightly.

1. Swarm Mode: In swarm mode the enemies will seek you in an attempt to kill you. The enemy spawners are now unlimited and it is inevitable you will die. In this mode the goal is to survive as long as you can. The HUD includes a clock for you to see this.
2. Protect and Serve: In this mode the enemies instead seek towards a monument which has a health bar of its own. The player will need to protect this be killing all of the zombies which get close to it or attack it. If the monument is destroyed you lose the game. However, in this mode the enemy spawners have a maximum enemy number they can spawn, if there is none left to spawn you win.
3. Collect and Run: In this mode the goal is to collect all coins and escape through one of the doors. The doors do not let you through until you have collected all coins. This game mode also has a limited number of enemies so you can either try rush before too many spawn and collect the coins quickly, finish all enemies first and saunter around or something in between

Toolbar

The toolbar is a UI element at the top of the screen during the game creation process. The toolbar, unlike in other applications is small consisting of just 3 tools for placing walls, spawners and game objects. One is to remove what’s been placed already, one is add a game object to the clicked cell and the last one is to add a lot of game objects at once.

Rubber

The rubber tool unlike the others doesn’t require you to also choose an item from the tool bar. The rubber tool allows you to remove game objects and walls from the creation grid that you have previously placed. This does a lot of things, firstly it removes the object from the vector of that object and deletes it. Additionally, it changes that cell back to transparent and sets it to empty to ensure a new game object can be placed in that cell in the future. Lastly it changes the amount of that specific object that has been placed to represent this.

Brush

The brush tool does require you to use the choice bar simultaneously. When you use the brush, you also choose one of the 3 options from the current choice bar. The chosen game objects are then instantiated in the cells in which you click. Additionally, any cells which you hover over while holding left click will also fill with game objects. The instantiation of the game objects in the cells occurs when left click is released.

Fill

The fill tool again requires simultaneous use of the choice bar. The fill tool creates a “Selection square” similar to clicking and highlighting an area on your desktop. With the fill tool when you click and hold the left button, a start point is added to the selection square, then when you move away from this the selection square grows to match the distance between both of the points (start and current mouse). When the mouse button is released, the cells which intersected with the selection square, which are not already filled with a game object are filled with the current choice. The selection square is deleted.

Choicebar

The choice bar is how you decide which of the objects you would like to place with the brush or fill tool. The choice bars 3 items change as you progress true the development of your game. Like most things the items in the choicebar are dynamic and will grow and shrink when you hover over them. The choice bar is just a UI element

Placement

Placement of objects is done by using the tool bar and choice bar simultaneously. Once you have a tool and choice made you can place objects. When you click a cell, its colour is set to something other than transparent and its type is set to a string e.g. “Wall1”. Then when the mouse release event is called a function which adds walls to those positions is added. The wall vector is dynamically added to at this point. The wall vector is a vector of unique pointers to objects of type wall. This means that the only limit to how many game objects the user can place is the overall grid size

Testing

Clicking an interface button allows the user to test their game. The game will play out similarly to how it would if they would have saved the game to a file. This way they can create their game. Test it, go back and change it to make it what they originally intended and then eventually save it once they are happy. This is done by simulating a separate scene with the same draw and update calls as a game scene would have.

**Storage and Saving**

Saving Game

Saving a game involves for the user just clicking a button. When they do that however a “saveToFile” function is called. This function creates a csv file in the ASSETS/GAMEDATA directory with the same name as the game, provided it doesn’t exist. Either way it opens that file and writes all of the information required to the file. This is split into 4 coloumns XPOS, YPOS, TYPE and OBJECT. Here the contents of the dynamic vectors of objects are saved as well as some of the game settings including game type and background.

Loading Game

When the game is constructed the levelLoader file opens the gameData directory and copies all the names of the game files to a vector. This is used to create the vectors of 4 different buttons for each game. One to play, one to build, one to upload and lastly one to delete. When you click play, that specific file is loaded in and all of the information is saved to various game data vectors. These vectors are just full of information like location etc. This information is then all used in a create Game function which uses all that information to create all the unique pointer vectors again. This ensures that when the objects are being added and deleted like this, the memory is getting released to be used again.

Adding Game to database

Adding a game to a database involves a couple of simple steps which are a little more difficult in code when using SQLite. Also error messages are output to the console if errors occur using std::cerr from the STL. Open the database, then you create the table, named after the file if it doesn’t exist, then you read all the data from the file and insert it line by line into the database table. This is done by splitting the data using a “,” as the deliminator and then inserting them underneath each of the respective column headers in the correct way. Lastly, it is important to close the database by calling sqlite3.close () and passing your database.

Deleting Game

Deleting the game is done by calling a function delete File and passing the filename. The file system creates a directory path, to the game data directory and a file path which is the directory path / (file name + file extension). If the file actually exists the filesystem function remove is called and passed the file path. The find files function needs to be recalled here, to make the user interface dynamic to what has occurred. Otherwise an app restart would be required.

Downloading from database

Downloading from the database is quite similar to uploading, except instead of taking a specific file to upload, all tables are downloaded at once. The steps involve opening the database, creating a list of tables that are inside the database, writing the column headers writing all the data rows one by one underneath this and then lastly closing the database.

**Gameplay**

Player

HUD

Clock, Dynamite Inventory, Health bar

Powerups

Nuclear Bomb, Invincibility, Invisibility

Enemies

Behavior