A project report On C++ Rogue

Abstract

"C++ Rogue" is rogue-like dungeon crawler game which is a subgenre of role-playing games (RPG). The name comes from the genre of the game and the language used for developing it. It makes use of a C++ library called SFML (Simple and Fast Multimedia Library). It is a game with multiple levels where enemies keep spawning and the main objective is to survive and proceed to further stages of the game aiming to attain a higher score every time by collecting coins and killing enemies.

List of figures

List of tables

Table of Contents

1. Introduction

1.1 Project Introduction

C++ Rogue is a rogue-like RPG game. It is a typical roguelike game with progression elements like multiple levels, coin collecting and difficulty scaling. The game is set up in a dungeon with obstacles and enemies stopping us from progressing further into the game. It is a 2D game with simple graphical elements. It is developed using SFML (Simple and Fast Multimedia Library). This library helps with the multimedia aspect of the game. The game consists of a main character which spawns in a dungeon in the initial level who further gets to progress into other levels killing the enemies and unlocking further levels to get to the end stage which marks the completion of the game.

1.2 Problem statement

Since the starting phase of developing this game the following problems were encountered which needs to be addressed:

- Learning a new totally new library to start developing the game increased time limitations.
- As stated above we had limited time period of 6 months to complete the project due to which we had to limit the project features of the game.
- The game as in its current state doesn't fit with a true rogue-like game as we planned during the initial phase.

1.3 Objectives of the project:

- To make a game for entertainment and recreation purpose of the users.
- To develop a rouge-like game.

1.4 Features of the project:

- **Progression:** Progression is a key element in every game so our game offers a progression system where we progress into further levels.
- **Score Recording:** The game records the score of the player playing the game so that the players can check their highest score.
- **Easy controls & immersive gameplay:** The game offers easy controls which makes it easier to jump into the game immediately furthermore the game has immersive gameplay.
- **Pixel graphics-based interface:** The game has a graphical interface which helps the player to interact with the game.

1.5 Document Organization:

This document consists of five chapters and these chapters each have their own sub-chapters.

Chapter 1	Chapter 1: Introduction This chapter is about a brief introduction to the project and it consists of other five chapters and they are as follow: 1.1 Project Introduction 1.2 Problem Statement

	1.3 Objectives of the project 1.4 Project Features 1.5 Assignment of roles and responsibilities 1.6 Document Organization
Chapter 2	Chapter 2: System Analysis This chapter is about system analysis and it is further divided into other three sub-chapter and they are: 2.1 Requirement gathering process 2.2 Gantt chart
Chapter 3	Chapter 3: System design This chapter is about system design of the project and it is divided into four sub-chapter and they are: 3.1 Functional analysis 3.2 Algorithm 3.3 Flowchart
Chapter 4	Chapter 4: System development and Implementation This chapter is about System development and implementation of the project and it is divided into three sub-chapter and they are: 4.1 Programming platform 4.2 Test plan 4.3 Implementation and result analysis

Chapter 5	Chapter 5: Conclusion and Future Enhancement This chapter is about conclusion and future enhancement of the project and it is divided into three sub-chapter and they are: 5.1. Conclusion 5.2. Limitation 5.3. Future Enhancement

2. System Analysis

2.1 Requirement gathering process:

Requirement No.	Requirement Name	Requirement description
1	Language	It needs C++ for coding
2	Developing software	IDE or C++ compiler

3. System Design

3.1 Functional analysis

Function No.	Function name	Function Description
1	running()	This function checks if the game window is still running or open.
2	update()	This function updates all the running game events.
3	render()	This function renders all the game elements into the window.
4	initVariables()	This function initializes all the variable.
5	initWindow()	This function initializes the main game window with a specified window resolution.
6	initEnemies()	This function initialized the enemy pointer.

7	initObstacles ()	This function initialized the obstacle pointer.
8	initFonts()	This function initializes the font to be used int the UI.
9	initTexts()	This function the text to be used in the UI.
10	Game()	This is the constructor for the Game class.
11	loadMap ()	This function initializes the game map.
12	renderMap()	This function renders the map into the game window.
13	mapCollision ()	This function is used to define the boundaries of the map.
14	pollEvent()	This function is used to check the input or the events in the main game loop.
15	updateObstacle()	This function updates the position and movement of the obstacles in the game window.
16	updateEnemies()	This function updates the position and movement of the enemies in the game window.
17	updateTexts()	This function updates the texts being displayed in the UI.
18	renderGui()	This function renders the GUI into the game window.
19	getPosition()	This function generates random coordinates within the window.
20	spawnEnemy()	This function spawns enemy into the required postions.

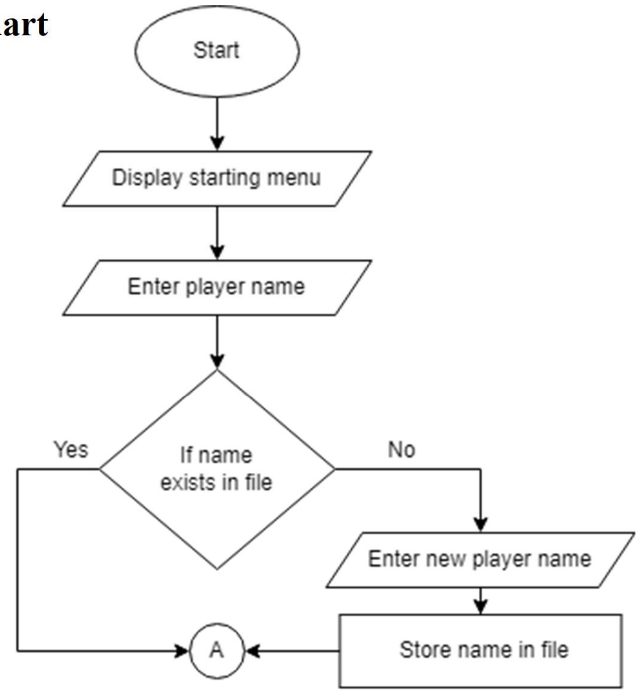
21	Enemy()	This is a constructor the Enemy class.
22	Animation()	This is a constructor for the Animation class.
23	updateAllFrames()	This function updates all the frames to create the animation.
24	pause()	This function holds the animation.
25	resume()	This function resumes the animation.
26	setSwitchTime()	This function sets the time to switch the animation frames.
27	updateUvRect()	This function updates the texture used for animating counting its rows and coulmns.
28	initTextures()	This function initializes the texture for the player
29	initSprites()	This function initializes the player Sprite (object).
30	HandleMovement()	This function handles the player movement when certain keys are pressed.

3.2 Algorithm

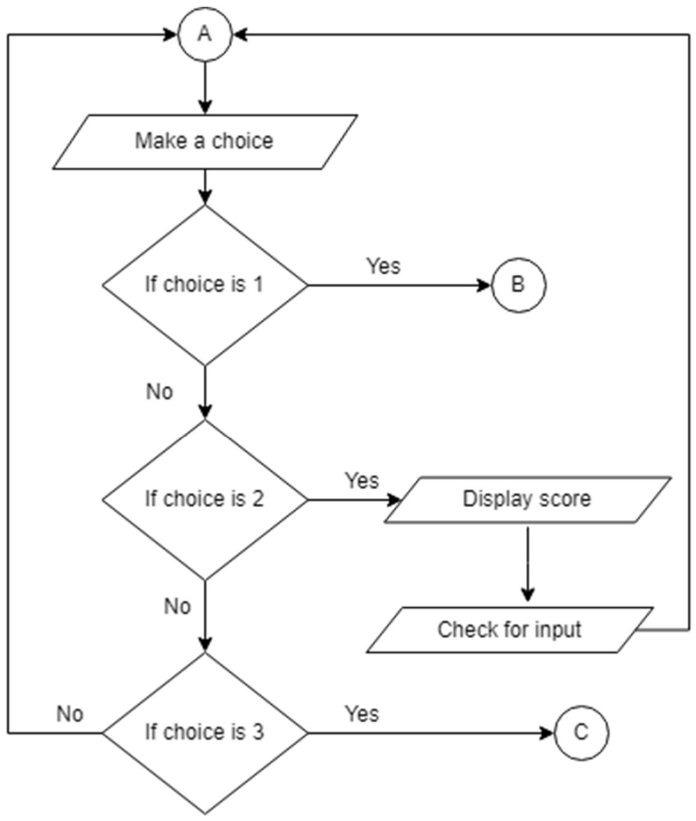
```
Step 1: Start
Step 2: Display Starting menu
Step 3: Enter player name
Step 4: If name exits in file
        Goto Step 5
         Otherwise
            Step 4.1: Enter new player name
            Step 4.2: Store player name in file
Step 5: Display Main menu
Step 6: Make a choice
Step 7: If choice is 1, 2 or 3
        Step 7.1: If choice is 1
                  Goto Step 8
        Step 7.2: If choice is 2
                        Step 7.2.1: Display Scores
                        Step 7.2.2: Wait for input
                        Goto Step 5


        Step 7.3: If choice is 3
                        Goto Step 9
        Otherwise
                Goto Step 5
Step 8: If game window is running
        Step 8.1: Check for event
        Step 8.2: Update current window
                Step 8.2.1: Update player
                        Step 8.2.1.1: Animate player with respect input
                        Step 8.2.1.2: Check player health
                        Step 8.2.1.3: If player health is zero
                                        Goto Step 9
            Step 8.2.1.4: Check kill count
            Step 8.2.1.5: If kill count exceeds for given level
                              Goto Step 8.2.4
        Step 8.2.2: Update enemy
            Step 8.2.2.1: Animate random enemies from array
            Step 8.2.2.2: Get player coordinates
            Step 8.2.2.2: Move enemy towards player
                          coordinates
            Step 8.2.2.3: If player coordinates intersect enemy
                          coordinates
                Step 8.2.2.3.1: If player attacks Then
                                    Remove enemy
                                    Increase player health
                                    Increase kill count
                                Otherwise
                                    Decrease player health

        Step 8.2.3: Update obstacles
            Step 8.2.3.1: Animate random obstacle from array
            Step 8.2.3.2: Get player coordinates
            Step 8.2.2.3: If player coordinates intersect obstacle
                          coordinates
                              Decrease player health
        Step 8.2.4: Load map
        Step 8.2.5: Check for map collision
            Step 8.2.5.1: Get player coordinates
            Step 8.2.5.2: If player coordinates match with the
                          map edge coordinates
                          Stop player movement
                          Otherwise
                              Goto step 8.2.1.1
    Step 8.3: Render current state
```

```
                Step 8.3.1: Clear current window
                Step 8.3.2: Render map
                Step 8.3.3: Render obstacle
                Step 8.3.4: Render enemies
                Step 8.3.5: Render player
                Step 8.3.6: Goto Step 8.1
                Otherwise
                     Continue
Step 9: Display end game menu
     Step 9.1: Check for input
     Step 9.2: If player wants to continue
                Goto Step 8
          Otherwise
                Goto Step 10
Step 10: Close window
Step 11: Stop
```

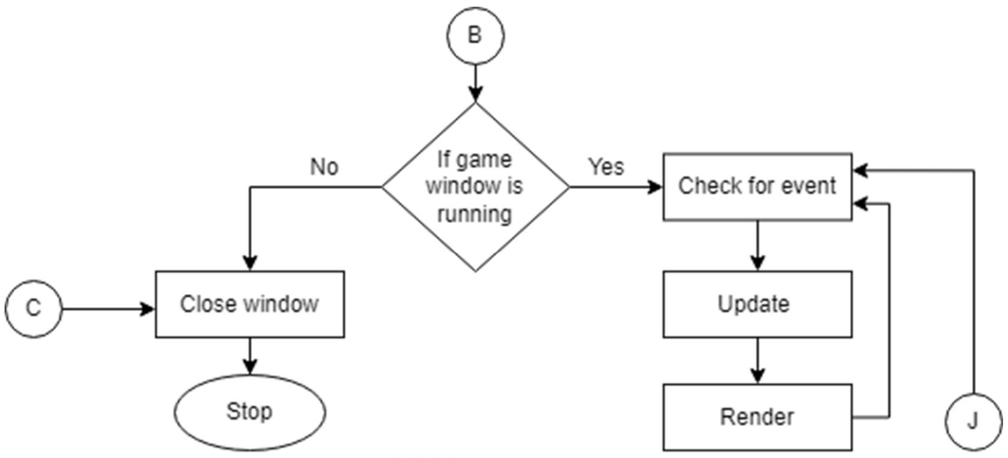
3.4 Flowchart



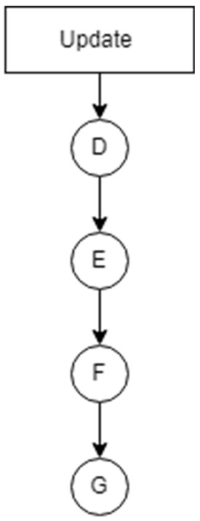
1. Starting Menu



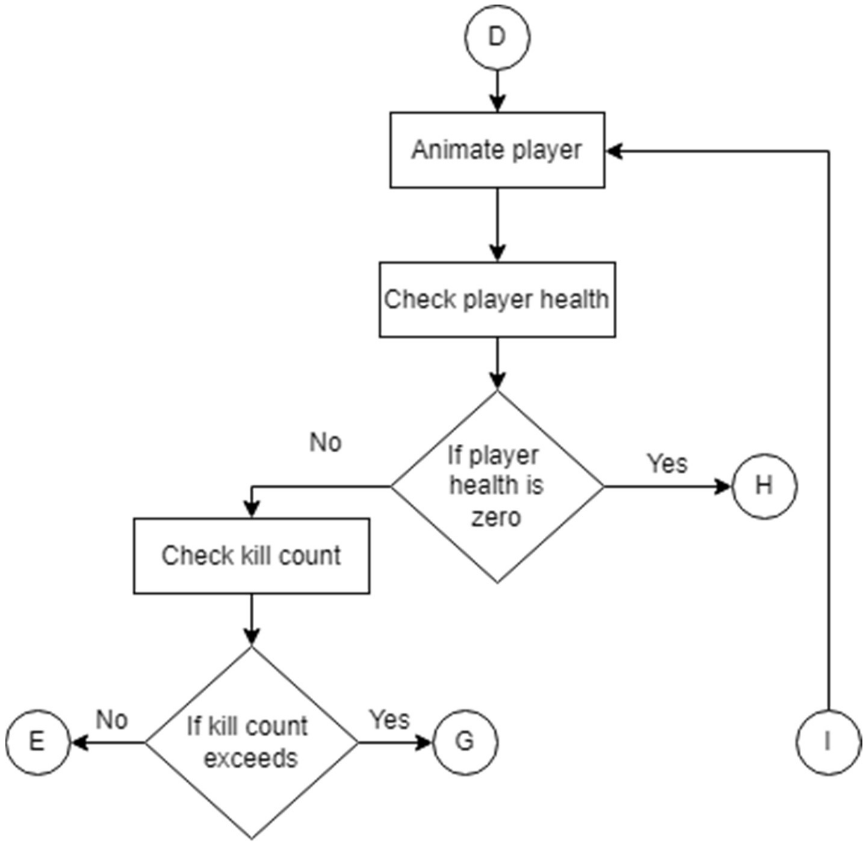
2. Main menu



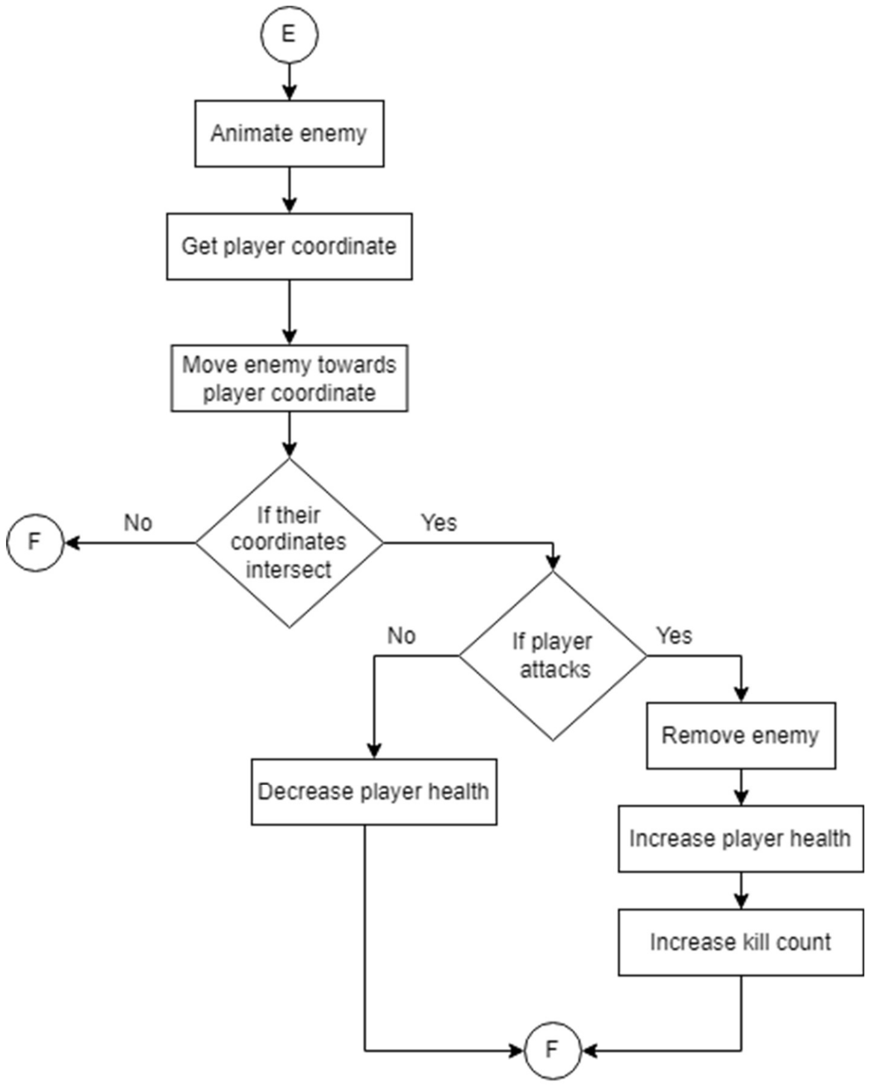
3. Main game loop



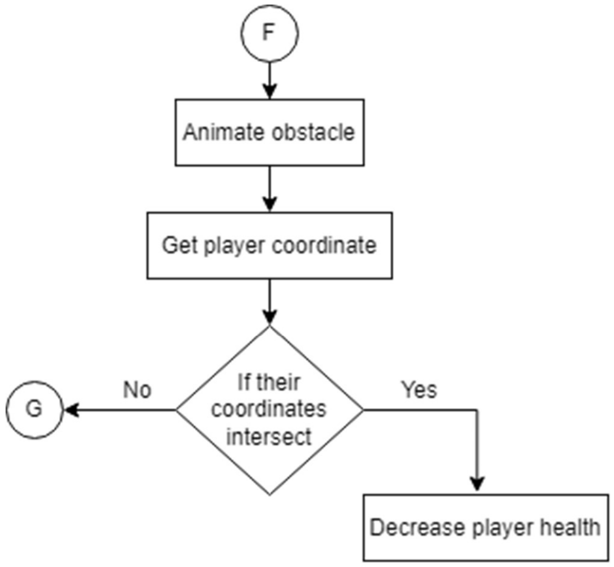
4. Update



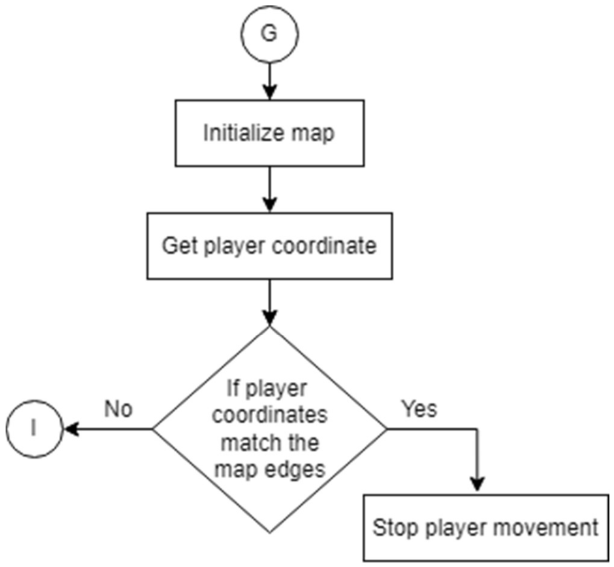
5. Update player



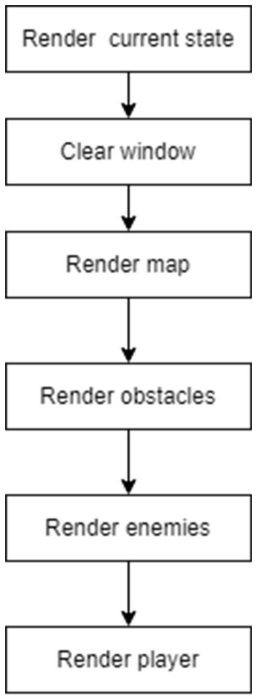
6. Update enemy



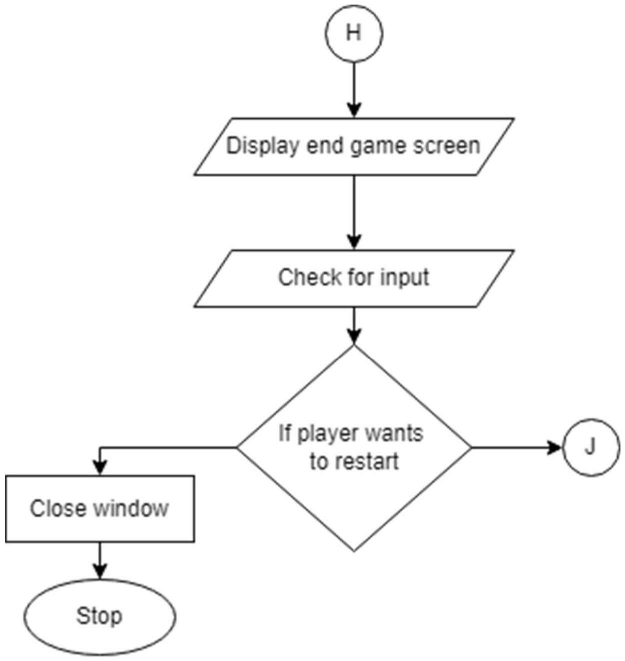
7. Update obstacle



8. Update map



9. Render



10.End game menu

4. System Development and Implementation

4.1 Programming platform:

Microsoft Visual Studio has been used as the programming platform for this project. C++ programming language together with an external library called SFML (Simple and Fast Multimedia Library).

4.2 Test plan

Test case	1
Test objective	To check the player moment.
Expected result	The player should be moving to the respected directions when the keys are pressed.
Test result	The player moved accordingly i.e. left with A, right with D, upwards with W and downwards with S.
Conclusion	Expected result matches the test result.

Test case	2
Test objective	To check the map collision.
Expected result	The player should be moving only inside the map boundaries.
Test result	The player moved inside the map boundaries.
Conclusion	Expected result matches the test result.

Test case	3
Test objective	To check enemy collision.
Expected result	The player health should decrease if player makes contact with enemy. The player health should increase if player kills the enemy and increase kill count by 1.
Test result	The player health decreased when player made contact with enemy. The player health increased when player kills the enemy and increase kill count by 1.
Conclusion	Expected result matches the test result

Test case	4
Test objective	To check the obstacle collision.
Expected result	The player health should decrease if player makes contact with obstacle.
Test result	The player health decrease when player made contact with obstacle.

Test case	5
Test objective	To check kill count and health.
Expected result	If player kills the enemy increase kill count and increase health points by 1
Test result	The kill count and health increased when player killed an enemy.
Conclusion	Expected result matches the test result.

4.3 Implementation and result analysis

S.N	Test Objective	Result
1	To check the player moment	Successful
2	To check the map collision	Successful
3	To check enemy collision	Successful
4	To check the obstacle collision	Successful
5	To check kill count and health	Successful

5. Conclusion and Future Enhancements:

5.1 Conclusion

"C++ Rogue" in its final state contains most of our planned features for the game. Some of the initial planned features like inventory system and multiple levels were not achieved. The game can be further polished into a true rogue like game with certain enhancements. By the completion of this project we have achieved most of our goals to develop a rogue like game.

5.2 Limitations:

Like every other program this program also has its own limitations and they are stated below:

- Playing multiplayer an online is not possible.
- Limited levels.
- No inventory system.

5.3 Future Enhancements:

The future enhancements that could be done in this project are as follows:

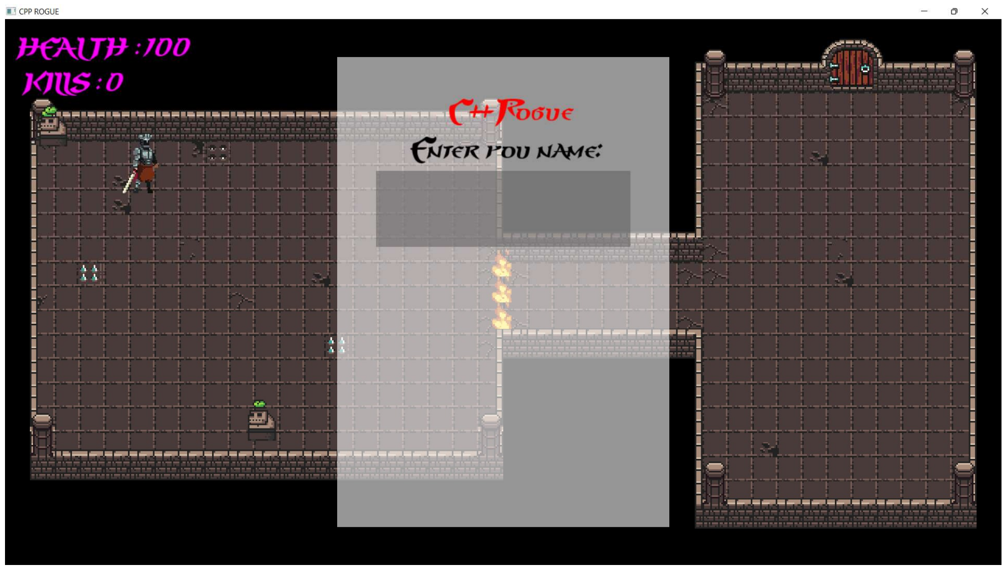
- More levels can be added.
- Weapon loot and inventory system can be added.
- Player customization can be added.
- Online multiplayer mode can be added.

References:

- Balagurusamy, E. (2011). *Object Oriented Programming with C++* (4th ed.). The McGraw-Hill Companies.
- Pupius, R. (2017). *Mastering SFML Game Development* [E-book]. Packt Publishing.
- Shekar, S. (2019). *C++ Game Development By Example: Learn to build games and graphics with SFML, OpenGL, and Vulkan using C++ programming* [E-book]. Packt Publishing.
- Pupius, R. (2015). *SFML Game Development By Example*. Packt Publishing.
- *[C++ & SFML - Simple 2D Games] - Introduction video*. (2019, August 4). YouTube. Retrieved April 15, 2022, from https://www.youtube.com/watch?v=BySDfVNljG8&list=PL6xSOsbVA1eb_QqMTTcql_3PdOiE928up

Appendices:

Appendix-1: (Output screen of the program)


CPP ROGUE
HEALTH :100
KILLS :0
C++Rogue
Enter you name: