A

PROJECT REPORT

ON

Insanity – A 3D Horror Game

SUBMITTED BY

DHRUV JAIN	**SS21CO006**
FAWAZ KHAN	**SS21CO016**
MEET JADHAV	**SW21CO003**
TANISH MORE	**SS21CO026**
RHIAN KUTTIKADAN	**SS21CO012**

In The Partial Fulfillment for The Requirements of Diploma in Computer Engineering (Sandwich Pattern)

2023-2024

Under the guidance of

MS. JIJNASA S. PATIL

DEPARTMENT OF COMPUTER ENGINEERING

GOVERNMENT POLYTECHNIC, MUMBAI - 400051

DEPARTMENT OF COMPUTER ENGINEERING (SANDWICH PATTERN)

GOVERNMENT POLYTECHNIC, MUMBAI

MUMBAI – 400051

CERTIFICATE

This is to certify that following students have successfully and satisfactorily completed the project on **Insanity – A 3D Horror Game** presented their report in the partial fulfillment of requirement for Diploma in Computer Engineering from Government Polytechnic, Mumbai under the guidance of Ms. Jijnasa S. Patil in the academic year 2023-2024.

DHRUV JAIN	**SS21CO006**
FAWAZ KHAN	**SS21CO016**
MEET JADHAV	**SW21CO003**
TANISH MORE	**SS21CO026**
RHIAN KUTTIKADAN	**SS21CO012**

Project Guide	External Examiner
Head of Department	Principal

ACKNOWLEDGEMENT

This project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along the journey of our project. All that we have done is only due to such supervision and assistance and we would not forget to thank them.

We would first and foremost like to thank our principal Dr. N. N. Mhala and HOD Mrs. Neha H. Vachani, for supporting our project idea from the beginning and motivating us to work on our project without any worries. We owe our deep gratitude to our Guide, Ms. Jijnasa S. Patil, who helped us on our project work and guided us all along, by providing all the necessary information for developing a good system.

We respect her for providing us an opportunity to do the project work in Government Polytechnic Mumbai and giving us support and guidance which made us complete the project duly. We are extremely thankful to her for providing such a nice support and guidance.

We are thankful to and fortunate enough to get constant encouragement, support and guidance from all teaching staff of Computer Department, GPM who helped us in successfully progressing with our project work.

INDEX

SR. NO.	CONTENTS	PAGE NO.
1	**Introduction** 1.1 Objectives	7-9
2	**Feasibility Study** 2.1 Project Scope 2.2 Development Environment	10-12
3	**Project Planning** 3.1 Scope and Schedule 3.2 Resources and Roles 3.3 Change in Planning	13-16
4	**Requirement Analysis** 4.1 Functional Requirements 4.2 Non-Functional Requirements	17-19
5	**System Design** 5.1 Game Control 5.2 Game Flow	20-25
6	**Coding**	26-38
7	**Testing**	39-44
8	**Outputs/Screenshots**	45-50
9	**Future Scope and Conclusion**	51-53
10	**References**	54-56

TABLE INDEX

SR. NO.	CAPTION	PAGE NO.
3.1	Schedule of the project.	15
7.1	Functional Test Cases.	41
7.2	Performance Test Cases.	42
7.3	User Experience Test Cases.	43
7.4	Requirement Traceability Matrix.	44-45

FIGURE INDEX

SR. NO.	CAPTION	PAGE NO.
3.1	Unity engine diagram.	16
3.2	C# diagram.	17
5.1	Basic Game Model Flowchart.	23
5.2	Gameplay Flowchart.	24
5.3	Ghost AI Flowchart.	25
5.4	Sound Implementation Flowchart.	26
8.1	Menu Screen.	47
8.2	House on Terrain.	47
8.3	House Interior.	48
8.4	Random Wall Paintings.	48
8.5	Random Room.	49
8.6	Torch.	49
8.7	Ground Floor.	50
8.8	Cursed Dolls.	50
8.9	Doll Pickup.	51
8.10	Burning Dolls.	51

ABSTRACT

"Insanity" is a 3D first-person horror game that immerses players in a terrifying experience within a malevolent haunted house. The game is meticulously designed to provide players with an adrenaline-pumping horror experience, combining sinister environments, chilling audio, ghostly encounters, and strategically placed jump scares.

The game leverages cutting-edge technology to create an atmosphere of relentless suspense. High-quality 3D graphics and immersive sound design are used to craft an intricate haunted house filled with eerie details. Players will encounter creaking floorboards, whispered spectral messages, and ominous shadows, all intended to heighten their sense of immersion and dread.

Players will navigate through eerie rooms, corridors, and hallways, where the haunted house dynamically responds to their every move and decision. Ghostly encounters are strategically placed to contribute to an overarching narrative that unveils the dark history of the house.

The game incorporates jump scares executed with precision, using visual and auditory cues to keep players on edge. These moments are designed to induce shock and anxiety without compromising the overall gameplay experience.

"Insanity" is not merely about cheap thrills; it aims to provide a psychological and emotional journey that leaves a lasting impact. By pushing the boundaries of horror gaming and tapping into primal fears, "Insanity" seeks to redefine the horror gaming experience, providing a unique and unforgettable experience for players daring enough to enter its nightmarish world.

In summary, "Insanity" is poised to become a benchmark in the 3D first-person horror game genre, offering an unparalleled level of immersion, fear, and excitement for those who venture into its dark and chilling world.

CHAPTER 1

INTRODUCTION

Introduction:

The gaming industry is a dynamic landscape, continuously shaped by changing player preferences. One enduringly captivating genre is horror, recognized for its ability to deliver immersive, terrifying experiences. In response to the sustained demand for high-quality horror games, we introduce "Insanity," a 3D first-person horror game designed to transport players into a heart-pounding and hair-raising adventure.

1.1 Objectives:

The "Insanity" project has set a series of critical objectives, aiming to create a groundbreaking horror game that sets new standards in fear-inducing gameplay:

1.1.1 Immersive Environments: Our primary aim is to immerse players in a malevolent haunted house that is both visually stunning and psychologically chilling. By utilizing high-quality 3D graphics, we intend to meticulously detail every aspect of the eerie setting, from creaking floorboards to ghostly apparitions lurking in the shadows. This approach heightens the player's sense of fear and anticipation.

1.1.2 Terrifying Audio: Sound design plays a pivotal role in creating a gripping horror game. Our objective is to use audio that not only complements the visuals but also serves as a core component of the horror experience. Whispered spectral messages, ominous background sounds, and eerie music will envelop the player, ensuring that every step they take is fraught with tension and fear.

1.1.3 Dynamic Gameplay: "Insanity" aims to make the haunted house a living, malevolent entity. The objective is to create dynamic gameplay, with the house responding to the player's movements and choices, keeping players on edge and uncertain.

1.1.4 Well-Placed Jump Scares: We intend to strategically place jump scares throughout the game, utilizing visual and auditory cues that induce shock and anxiety, keeping players on edge without disrupting gameplay.

1.1.5 Emotional Resonance: Beyond immediate thrills, "Insanity" aims to create an emotionally resonant gaming experience. Our objective is for the game to leave a lasting impact, making players reflect on the horror they've experienced even after they've put down the controller.

This project report will delve into the details of how these objectives will be achieved, highlighting the technical and creative aspects of "Insanity" that make it a unique and unforgettable gaming experience.

CHAPTER 2
Feasibility Study

Feasibility Study

A feasibility study is an assessment conducted to determine the practicality and potential success of a project or business venture by evaluating various factors such as financial, technical, and operational aspects. It helps stakeholders make informed decisions about the project's viability.

2.1 Project Scope

The scope of "Insanity" encompasses the creation of a 3D first-person horror game that immerses players in a haunted house, delivering an atmosphere of fear and anticipation. What makes this project viable and efficient is our strategic use of resources from Unity Technologies, the creators of the Unity game engine. This includes leveraging assets from the Unity Asset Store, a treasure trove of high-quality 3D models and materials that align with our project objectives. These assets not only save development time but also enhance the visual fidelity of our game, ensuring that our vision is achievable within our allocated resources.

2.2 Development Environment

The approach to realizing "Insanity" is based on strategic resource utilization, meticulous planning, and creative innovation.

Unity Game Engine: Utilizing the Unity game engine provides us with a solid foundation for game development. It offers a user-friendly interface and extensive resources for 3D game creation.

Unity Asset Store: By accessing the Unity Asset Store for free models, we not only streamline development but also gain access to a vast library of assets that enhance the game's visual quality. This approach significantly accelerates the development process, making it feasible to meet project timelines.

Freesound.org: We will source sound effects from Freesound.org, ensuring high-quality audio components while adhering to budget constraints.

Mega Cloud Hosting: Collaborative development across teams will be made efficient and cost-effective through Mega cloud hosting. This approach facilitates real-time collaboration, version control, and project management, enabling our diverse team to work together seamlessly.

Resource Allocation: Resource allocation will be managed judiciously, ensuring that financial and human resources are effectively distributed to support the project's objectives.

2.3 Existing System

This section covers details about the existing games and movies we have referred for developing the story of game and adding elements inspired from it. There description are as follows:

1. Granny Horror Game:

The Granny Horror Game, developed by DVloper, served as a notable reference for our project. This survival horror game placed players in a suspenseful environment where they had to navigate a house while avoiding the pursuit of a hostile grandmother. We analyzed its innovative use of horror elements, immersive atmosphere, and strategic gameplay mechanics. The game's success was attributed to its ability to create tension through sound design, jump scares, and a dynamic AI system governing the antagonist. By understanding how Granny captivated players and maintained a sense of urgency, we aimed to incorporate similar psychological elements into our game to enhance the overall user experience.

2. Conjuring 2 (Movie):

The movie "Conjuring 2" was a significant source of inspiration for the thematic elements in our project. Directed by James Wan, this horror film was renowned for its gripping narrative, atmospheric cinematography, and effective use of supernatural elements. We studied how the movie built tension, employed visual and auditory cues to create scares, and weaved a compelling storyline within the horror genre. Our goal was to capture the essence of suspense and fear present in "Conjuring 2" and incorporate these elements into our game to provide players with a captivating and immersive horror experience. Drawing from the successful strategies employed in the movie, we integrated similar atmospheric elements into our game to enhance the overall horror ambiance of our project.

CHAPTER 3
Project Planning

Project Planning

Project planning is part of project management, which uses schedules and subsequently report progress within the project environment. Project planning can be done manually. The project plan clearly defines how the project is created, monitored and completed.

3.1 SCOPE AND SCHEDULE

Now, let's delve into the project's potential, its scale, and the expected timeframe for its completion. To achieve this, we will establish a well-structured timeline, specifying the start date and the projected completion date.

SCHEDULING OF THE PROJECT GIVEN BELOW:

Date	Day	Subject
11/07/2023	Tuesday	Introduction to Third Year Project
16/07/2023	Sunday	Group Selection
25/07/2023	Tuesday	Guide Allocation
17/08/2023	Thursday	Project selected
25/08/2023	Friday	Discussion with guide about project
27/08/2023	Sunday	Planning Story For the game
20/09/2023	Wednesday	Designing Environments and level
05/10/2023	Thursday	Implementing Functionalities in the game
15/10/2023	Sunday	Designing User Interface
30/10/2023	Monday	Game tested successfully
07/11/2023	Tuesday	Exported The Final Build

Table 3.1

3.2Resources and Roles

1.Unity:

Unity is a powerful and popular cross-platform game development engine and framework. It's used to create 2D, 3D, augmented reality (AR), and virtual reality (VR) applications. Unity allows developers to build interactive experiences for various platforms, including PC, mobile, consoles, and more.

2. Unity Engine:

The Unity engine is the core software that provides the framework for creating games and interactive applications. It includes a wide range of features, such as graphics rendering, physics simulation, audio, scripting, and more. Unity is known for its user-friendly interface and the ability to export projects to multiple platforms.

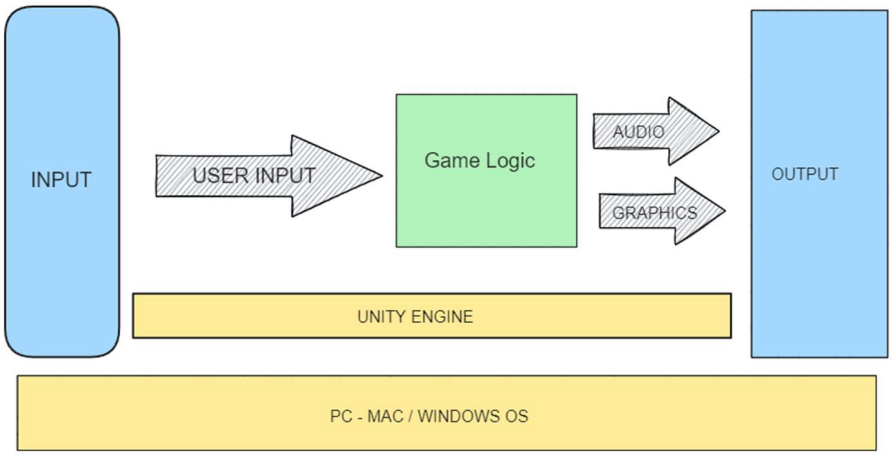

Figure 3.1

3. Unity Hub:

Unity Hub is a management tool that simplifies the process of working with multiple Unity projects and versions. It allows developers to create, open, and organize projects, install different versions of the Unity engine, and manage assets and packages. Unity Hub streamlines development workflows by providing a centralized hub for all Unity-related tasks.

4. C# (C Sharp):

C# is a high-level, object-oriented programming language developed by Microsoft. It's widely used in Unity for scripting game logic and creating interactive elements. C# is known for its ease of use, strong typing, and extensive libraries, making it a popular choice for Unity game development.

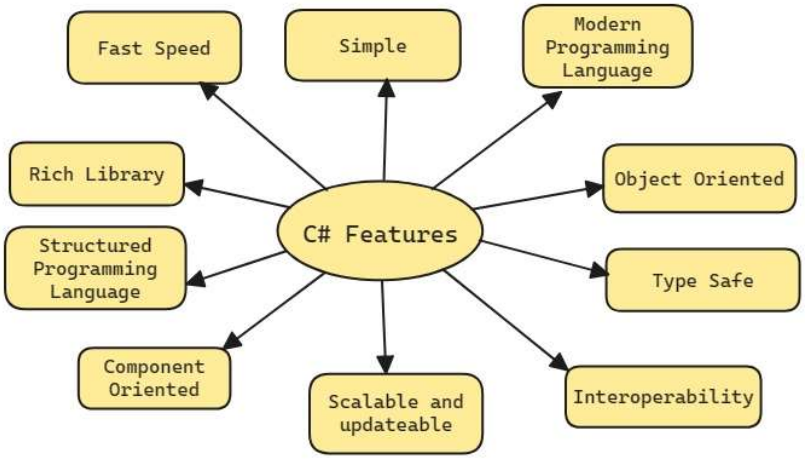

Figure 3.2

5. **Visual Studio:**

Microsoft Visual Studio is an integrated development environment (IDE) developed by Microsoft. It is used primarily for developing computer programs, web applications, and mobile apps. Visual Studio provides a range of features and tools for various programming languages, including C#, C++, F#, Visual Basic, and more.

3.3 Change in Planning

Change management was critical to adapt to evolving circumstances. As the project progressed, changes in scope, technology, or resources arose. We were committed to handle changes efficiently by evaluating their impact on the project timeline, costs, and objectives. We regularly reviewed and adapted to user feedback to ensure that "Insanity" remained aligned with the dynamic nature of the gaming industry and player expectations. Our iterative development approach was designed to incorporate changes seamlessly, providing flexibility without compromising project quality.

CHAPTER 4
Requirement Analysis

Requirement Analysis

"Insanity - A 3D Horror Game" stands out from others by offering meticulously designed environments, an emotionally resonant narrative, and dynamic gameplay that keeps players engaged and on edge, creating an unparalleled horror experience that lingers long after the game is over.

The requirement analysis phase of "Insanity - A 3D Horror Game" involves a thorough examination of both functional and non-functional requirements to ensure the successful development of the game.

Requirement Specifications:

- **Functional Requirements**
- **Non-Functional Requirements**

4.1 Functional Requirements

Functional requirements define specific features and interfaces crucial for the game's functionality:

User Interface (UI): The UI in "Insanity" must provide players with an intuitive and visually appealing means of navigating the game. It includes menus for game options, character customization, and inventory management. The UI should be aesthetically consistent with the horror theme, ensuring that players can easily access and adjust game settings, enhancing the overall user experience.

Hardware Interface: The game efficiently utilizes hardware interfaces to ensure smooth performance on a variety of hardware configurations. Optimization for different screen sizes, resolutions, and input devices, such as keyboard and mouse provided an optimal gaming experience for all players. Compatibility with a range of hardware setups was maintained to maximize accessibility.

Software Interfaces: "Insanity" seamlessly integrates with the Unity game engine, Unity Asset Store, and Freesound.org for asset utilization. This involves smooth asset importing, management, and integration to streamline the development process. Additionally, the game is compatible with popular gaming platforms, including PC, gaming consoles enabling easy distribution to a wide audience of players while adhering to platform-specific requirements and guidelines.

User Experience (UX): User experience encompasses the holistic interaction between players and the game. Beyond the UI and hardware interface, UX in "Insanity" emphasizes the emotional and psychological aspects of gameplay. It is about creating a sense of fear, tension, and immersion. The UX includes the design of dynamic horror elements, player choices, and the overall atmosphere that keeps players engaged, scared, and excited. It also considers responsiveness, efficient navigation, and an optimal balance between challenge and enjoyment to ensure a deeply engaging and satisfying experience for players.

4.2 Non-Functional Requirements

Non-functional requirements ensure the quality and performance of "Insanity":

Performance: The game runs smoothly and efficiently on various hardware configurations, including lower-end and high-end systems, ensuring an optimal player experience without significant lag, frame drops, or performance issues. High-quality 3D graphics and audio do not compromise performance.

Scalability: The game's architecture allows for potential updates and expansions, ensuring scalability for future improvements. The development team can add new content, features, and levels without causing a significant impact on the game's performance or stability, allowing for long-term growth and player engagement.

Maintainability: Maintenance and updates are manageable and cost-effective. The game is designed with clean and well-documented code, facilitating the efficient resolution of issues and the addition of new content or features. This keeps the game fresh and engaging for players while minimizing the development team's workload in the long term. Regular updates and bug fixes are conducted seamlessly to ensure a consistently high-quality player experience.

CHAPTER 5
System Design

System Design

Methodology

The methodology employed in the development of "Insanity - A 3D Horror Game" is structured to ensure efficient project management, asset utilization, and quality control. The following elements comprise the methodology:

- **Agile Development:** We have adopted an Agile development approach, emphasizing iterative cycles and frequent collaboration between team members. This methodology allows for flexibility, facilitating the accommodation of changing requirements and the integration of user feedback into the development process.

- **Collaboration Through Cloud Tools:** To enable seamless collaboration across geographically dispersed teams, we use cloud-based tools like Mega. This ensures real-time communication, version control, and centralized project management. It has proven to be instrumental in enhancing team coordination and productivity.

- **Quality Assurance:** Quality assurance and testing are integral to our methodology. We employ a rigorous testing process that includes functional, performance, and compatibility testing. This phase ensures that the game meets defined requirements, runs smoothly, and provides an engaging player experience.

- **User Feedback Integration:** Throughout the development process, we actively gather and integrate user feedback. This iterative approach ensures that the game aligns with player expectations and evolving industry standards. Player feedback guides refinements and improvements in subsequent development cycles.

- **Resource Allocation:** Resource allocation is managed judiciously, ensuring that financial and human resources are effectively distributed to support project objectives. This approach enables us to optimize costs while maintaining the desired quality.

Game Control

- W – Forward
- A – Left
- S – Backward
- D – Right
- X – Flashlight
- E – Inventory
- Space – Open door
- Esc - Menu
- LMB – Object Pick Up

Game Flow

Understanding the flow of the game is critical to delivering a suspenseful and engaging player experience:

Basic Game Model Flowchart:

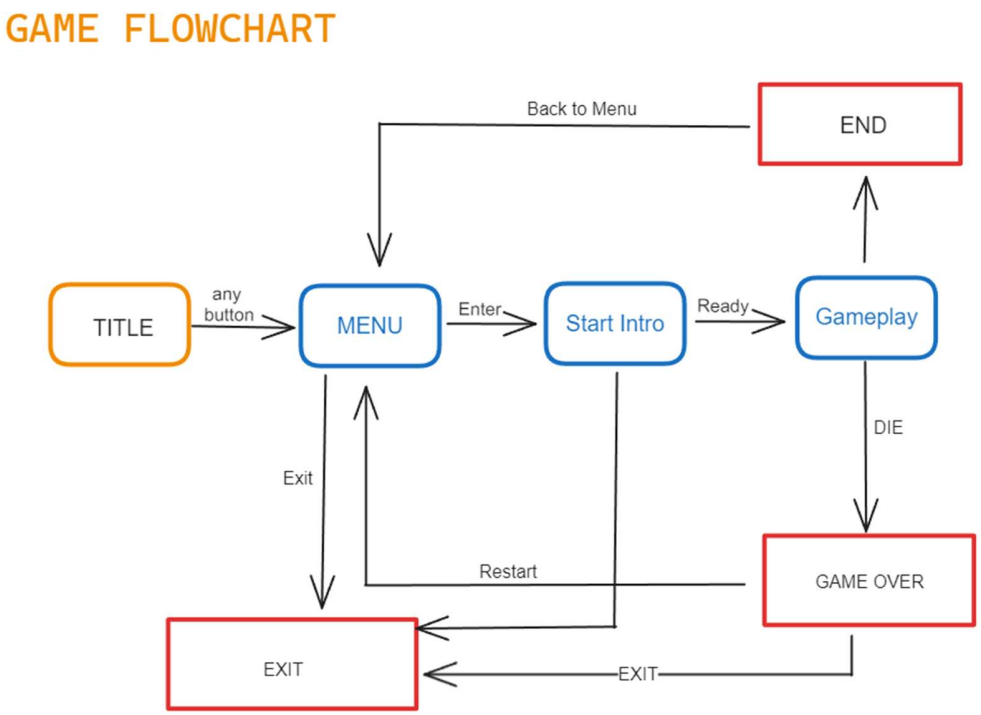

Figure 5.1

This flowchart is a schematic representation of the game's timeline. Once the player starts the game, he/she will be directed to the Title screen. By pressing any button there, then the player will be directed to the Menu screen.

Here the player can start the game or quit. If the player chooses to play, then he will be shown the intro scene. Once the intro scene is over, then the gameplay starts. If the player dies during the game, then he will be redirected to the menu screen. If the player succeeds in defeating the ghost, then it will end the game and the desired ending is shown. Then he will be directed to the menu screen again, where he can quit or play the game again.

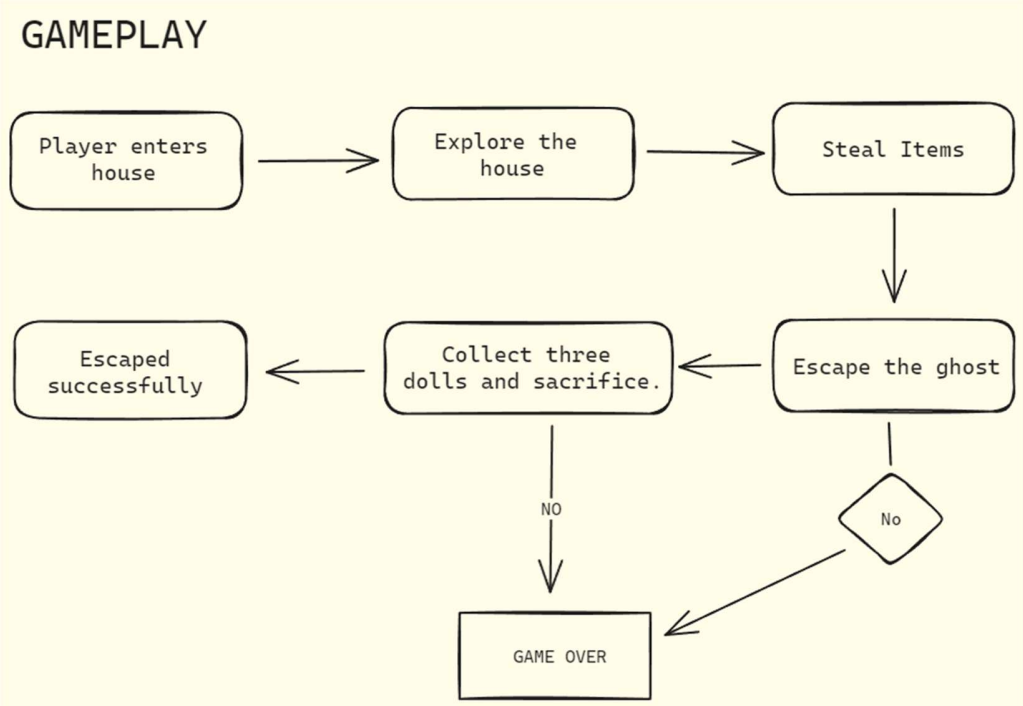

Figure 5.2

The gameplay starts by the player entering the main house. The player has to explore the house in search of the items to steal. Although the stealing of items will not aid in defeating the ghost, it is still an important mission and objective for the player. The items will include objects like keys to unlock doors. Once the player encounters the ghost, then it has to escape the ghost whilst collecting three dolls. The three dolls will aid in defeating the ghost, which ultimately help the player escape manor.

If the player cannot successfully escape the ghost or sacrifice the three dolls in time, then it would lead to the defeat of the player thus ending the game prematurely.

Once the player collects the three dolls, he has to go to a room to sacrifice them. If he sacrifices it – then he can escape from the house in time.

GHOST AI

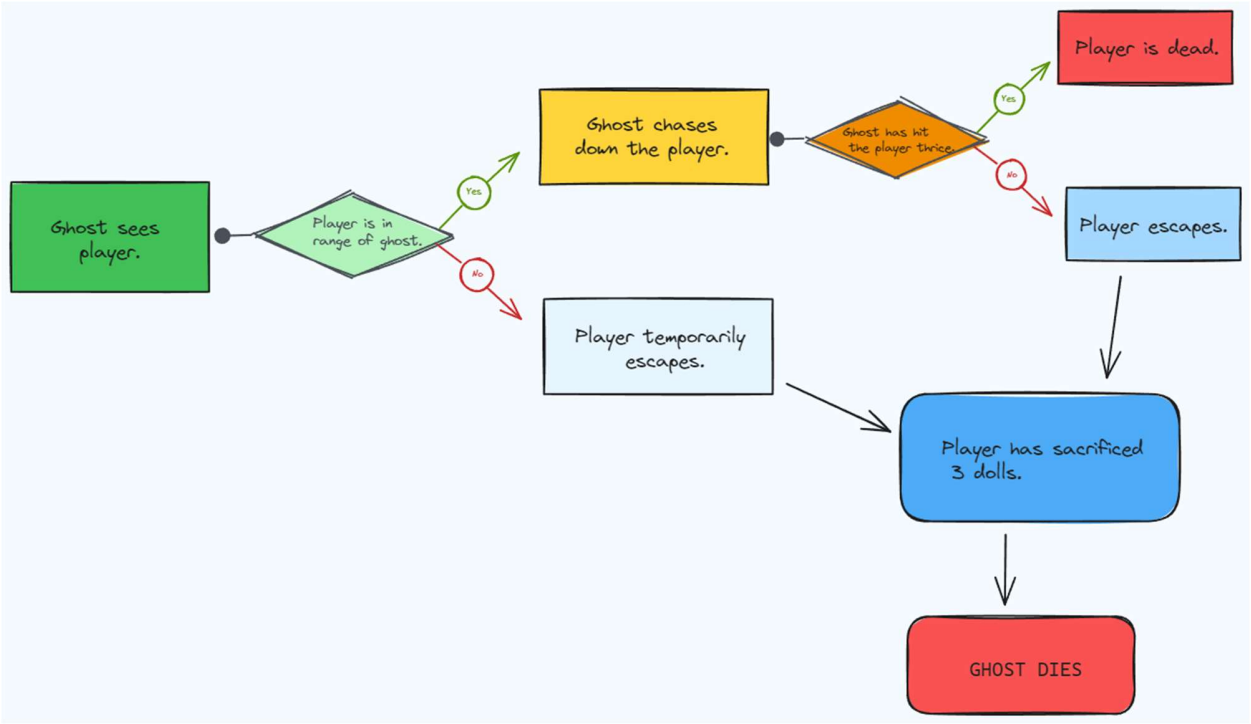

Figure 5.3

This is a flowchart for depicting the implementation of ghost ai.

If the ghost sees the player, (i.e. their colliders intersect) then the system will check if the player is in range of the ghost to start chasing. If the player is not in range, then he has limited time to move out of its range before it starts chasing.

If the player is in range, then the ghost starts to chase the player for some time. If the player gets hit thrice in total during that time, then the player dies.

If the player gets hit less than 3 times then the player should continue his main objective of finding the three dolls to sacrifice.

Once he sacrifices the three dolls, then the ghost dies.

SOUND

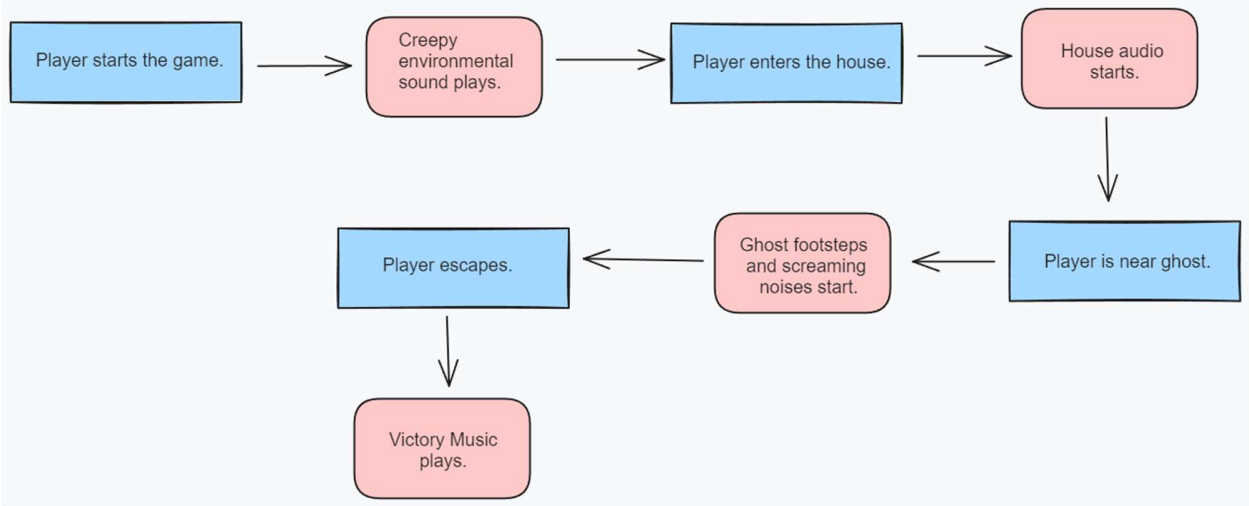

Figure 5.4

This is a flowchart depicting the sound system in the game.

Once the player starts the game, then the default creepy environmental sound will start playing. After entering the house, then a different music will start playing. This will be more intense and heart thumping music. Along with the music, there will be different sound effects like creaking noise for the doors, etc. If the player is near the ghost, then the sound of the ghost's footsteps and its blood-curdling screams will start. When the player kills the ghost, the game is almost over and the victory music will start.

CHAPTER 6
Coding

Source Code (C# Scripts)

PlayerMovement.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement :
MonoBehaviour{
   public CharacterController controller;
   public float speed = 12f;
   public float crouchSpeed = 5f;
   public float gravity = -9.81f;
   public Transform groundCheck;
   public float groundDistance = 0.4f;
   public LayerMask groundMask;

   private float initialControllerHeight;
   private Vector3 initialControllerCenter;
   private bool isCrouching = false;
   Vector3 velocity;
   bool isGrounded;
   private void Start(){
      initialControllerHeight =
controller.height;
      initialControllerCenter =
controller.center;
   }
   void Update(){
      isGrounded =
Physics.CheckSphere(groundCheck.position,
groundDistance, groundMask);
      if(isGrounded && velocity.y < 0){
         velocity.y = -3f;
      }
float moveSpeed = isCrouching ?
crouchSpeed : speed;
if (Input.GetKey(KeyCode.LeftShift)){
      if (!isCrouching){
            isCrouching = true;
controller.height = initialControllerHeight /
2f;
controller.center = new
Vector3(controller.center.x,
initialControllerCenter.y / 2f,
controller.center.z);
moveSpeed = crouchSpeed;
}}else    {
 if (isCrouching){
controller.height = initialControllerHeight;
controller.center = initialControllerCenter;
isCrouching = false;
        }}

 // Update moveSpeed based on crouch state
      moveSpeed = isCrouching ?
crouchSpeed : speed;

      float x = Input.GetAxis("Horizontal");
      float z = Input.GetAxis("Vertical");

      Vector3 move = transform.right * x +
transform.forward * z;

      controller.Move(move * moveSpeed *
Time.deltaTime);

      velocity.y += gravity * Time.deltaTime;

      controller.Move(velocity
*  Time.deltaTime);
   }
}
```

FlashlightToggle.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using
UnityEngine.Experimental.GlobalIlluminatio
n;

public class FlashlightToggle :
MonoBehaviour{
   public GameObject lightGO; //light
gameObject to work with
   private bool isOn = false; //is flashlight on
or off?
   public AudioClip buttonClickSound;
   public AudioSource torchAudioSource;
   // Use this for initialization
   void Start(){     //set default off
      lightGO.SetActive(isOn);
   }

   // Update is called once per frame
   void Update()
   {
      //toggle fla`shlight on key down
      if (Input.GetKeyDown(KeyCode.X)){
         //toggle light
         isOn = !isOn;
         //turn light on
         if (isOn){
            lightGO.SetActive(true);
         }
         //turn light off
         else {
            lightGO.SetActive(false);
         }
         torchAudioSource.PlayOneShot(butto
nClickSound);
      }
   }
}
```

torchMovement.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class torchMovement :
MonoBehaviour
{
   public Transform cameraObject;
}

   // Update is called once per frame
   void Update()
   {
      transform.localRotation =
Quaternion.Euler(cameraObject.transform.loc
alRotation.x+90,0,0);
   }
```

RenderText.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEditor.VersionControl;
using UnityEngine;
using UnityEngine.AI;

public class RenderText : MonoBehaviour{
// Start is called before the first frame update
   public TMP_Text messagebox;
   public TMP_Text instructionbox;
   void Start(){
      render("Welcome to this game",3f);
   }
   // Update is called once per frame
private IEnumerator
TypeandRenderMessage(string message ,
float delay){
      yield return new
WaitForSeconds(delay);
      for (int c=0; c<message.Length; c++){
         messagebox.text += message[c];
         yield return new
WaitForSeconds(0.05f);
         if(message.Length-1 == c){
            yield return new
WaitForSeconds(3f);
            messagebox.text = "";
         }}}
   private IEnumerator
TypeandRenderInstruction(string message){
      for (int c = 0; c < message.Length; c++){
         instructionbox.text += message[c];
         yield return new
WaitForSeconds(0.05f);
         if (message.Length - 1 == c){
            yield return new
WaitForSeconds(3f);
            instructionbox.text = "";
         }} }
   public  void render(string message, float
delayBeforeStart){
      StartCoroutine(TypeandRenderMessage(
message,delayBeforeStart));
   }
   public void renderInstruction(string
message){
      StartCoroutine(TypeandRenderInstructio
n(message));
   }
}
```

TriggerSEF.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TriggerSEF : MonoBehaviour{
public AudioSource playsound;
// This method will be called when the
player's collider enters the trigger zone.
   void OnTriggerEnter(Collider other)
   {
      // Check if the collider that entered the
trigger zone belongs to the player.
      if (other.CompareTag("Player")){
         // Play the sound when the player
collides with the object.
         playsound.Play();
      }
   }
```

EnterInHouse.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class EnterInHouse : MonoBehaviour
{
    public Transform door1;
    public Transform door2;
    public Transform player;
    public GameObject canvas;


    private bool nearDoor = false;
    public float rotateSpeed = 1f;
    public float rotation = 90f;
    private Coroutine AnimationCoroutine;
    private bool messageDisplayed = false;
    private Quaternion endRotation1;
    private Quaternion endRotation2;

    private void Update(){
        if (nearDoor){
            if
(Input.GetKeyDown(KeyCode.Space)){
                AnimationCoroutine =
StartCoroutine(doRotation());
            }
        }
    }

    private IEnumerator doRotation(){
        endRotation1 = Quaternion.Euler(new
Vector3(0,
door1.transform.rotation.eulerAngles.y -90,
door1.transform.rotation.eulerAngles.z));
        endRotation2 = Quaternion.Euler(new
Vector3(0,
door2.transform.rotation.eulerAngles.y + 90,
door2.transform.rotation.eulerAngles.z));

        float time = 0;
        while (time < 1){
            door1.transform.rotation =
Quaternion.Slerp(door1.transform.rotation,en
dRotation1, time);
            door2.transform.rotation =
Quaternion.Slerp(door2.transform.rotation,
endRotation2, time);

            yield return null;
            time += Time.deltaTime *
rotateSpeed;
        }
    }

    private void OnTriggerEnter(Collider
other){
        if (other.CompareTag("Player")){
            if (!messageDisplayed){
                canvas.GetComponent<RenderText
>().renderInstruction("click Space to open
door");
                messageDisplayed = true;
            }
            nearDoor = true;
        }
    }
    private void OnTriggerExit(Collider
other){
        if (other.CompareTag("Player")){
            nearDoor = false;
        }
    }
}
```

DoorOpen.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor.Experimental.GraphView;
using UnityEngine;

public class DoorOpen : MonoBehaviour{
   public GameObject Door;
   public GameObject Knoob;
   private bool nearDoor = false;
   public float rotateSpeed = 0.7f;
   public float rotation = 90f;
   public float direction = 1f;
   private Coroutine AnimationCoroutine;
   private Quaternion endRotation1;
   private bool isdoorclose = true;
   // Update is called once per frame
void Update(){
if (nearDoor){
if (Input.GetKeyDown(KeyCode.Space)){
     if(isdoorclose)
          StartCoroutine(doRotation());
         }
      }}
   private IEnumerator doRotation(){
      endRotation1 = Quaternion.Euler(new
Vector3(Door.transform.rotation.eulerAngles.
x, Door.transform.rotation.eulerAngles.y -
(90*direction),
Door.transform.rotation.eulerAngles.z));
      isdoorclose = false;
      float time = 0;
      while (time < 1){
         Door.transform.rotation =
Quaternion.Slerp(Door.transform.rotation,
endRotation1, time);
         yield return null;
         time += Time.deltaTime *
rotateSpeed;  }
      yield return new WaitForSeconds(2f);
      StartCoroutine(doRotationClose());
   }
   private IEnumerator doRotationClose(){
      endRotation1 = Quaternion.Euler(new
Vector3(Door.transform.rotation.eulerAngles.
x, Door.transform.rotation.eulerAngles.y +
(90*direction),
Door.transform.rotation.eulerAngles.z));

      float time = 0;
      while (time < 1){
         Door.transform.rotation =
Quaternion.Slerp(Door.transform.rotation,
endRotation1, time);
         yield return null;
         time += Time.deltaTime *
rotateSpeed;
      }
      isdoorclose = true;
   }

   private void OnTriggerEnter(Collider
other){
      if (other.CompareTag("Player"))    {
         Debug.Log(other.transform.position);
         nearDoor = true;
      }
   }
   private void OnTriggerExit(Collider
other){
      if (other.CompareTag("Player")){
         nearDoor = false;
      }
   }
}
```

MouseLook.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Mouselook : MonoBehaviour
{
   public float mouseSensitivity = 100f;

   public Transform playerBody;
   public Transform torch;
   float xRotation = 0f;
    void  Awake()
   {
      Cursor.lockState =
CursorLockMode.Locked;
   }

   // Update is called once per frame
    void Update()
   {
      float mouseX = Input.GetAxis("Mouse
X") * mouseSensitivity * Time.deltaTime;
      float mouseY = Input.GetAxis("Mouse
Y") * mouseSensitivity * Time.deltaTime;

      xRotation -= mouseY;
      xRotation = Mathf.Clamp(xRotation, -
90f, 90f);

      transform.localRotation =
Quaternion.Euler(xRotation, 0f, 0f);
      torch.transform.localRotation =
Quaternion.Euler(xRotation+90, 0f, 0f);
      playerBody.Rotate(Vector3.up *
mouseX);
   }
}
```

FlickerTrigger2.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FlickerTrigger2 :
MonoBehaviour{
   private bool playerInTriggerZone = false;
   public Light lampLight;
   public Color targetColor;
   public float targetIntensity;
   private Color originalColor; // Store the
original color of the light.
   private float totalFlickerCount = 4;
   public AudioClip flickerSound1;
   public AudioClip flickerSound2;
   public AudioClip flickerSound3;

   public AudioSource audioSource;
   private bool playsound = true;
   // Start is called before the first frame
update
   void Start(){
      originalColor = lampLight.color;
   }
   // Update is called once per frame
   void Update(){
      if (playerInTriggerZone &&
totalFlickerCount > 0){
```

```
            StartCoroutine(FlickerLight());
        }
    }
    void OnTriggerEnter(Collider other){
if (other.CompareTag("Player")){
            playerInTriggerZone = true;
        }    }
void OnTriggerExit(Collider other){
        if (other.CompareTag("Player")){
            playerInTriggerZone = false;
            lampLight.enabled = true;
        } }

IEnumerator FlickerLight(){
        float flickerDuration = 1.3f; // Duration
of flickering in seconds.
        float flickerInterval = 0.5f; // Time
between flickers.
        float elapsedTime = 0f;
        bool lightOn = true; // To keep track of
whether the light is on or off.
        while (elapsedTime < flickerDuration)  {
            // Toggle the light on/off.
            // Wait for the flicker interval.
            yield return new
WaitForSeconds(0.5f);
lampLight.enabled = lightOn;
lightOn = !lightOn;
// Update the elapsed time.
elapsedTime += flickerInterval;
        }

        // Ensure the light is back on when
flickering ends.
        lampLight.enabled = true;
        totalFlickerCount--;
        if (playsound)
        {
            float random = Random.Range(1, 3);
            Debug.Log(random);
            if (random == 1)
            {
                audioSource.PlayOneShot(flickerS
ound1, 0.1f);
            }
            else if(random == 2)
            {
                audioSource.PlayOneShot(flickerS
ound2, 0.1f);
            }
            }else if(random == 3)
            {
                audioSource.PlayOneShot(flickerS
ound3, 0.1f);
            }
            playsound = false;
        }
    }
}
```

Ghost2.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using UnityEditor.Experimental.GraphView;
using UnityEditor;
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.SceneManagement;
using Random = UnityEngine.Random;

public class Ghost2 : MonoBehaviour{
    public NavMeshAgent ai;
    public List<Transform> destinations;
    public Animator aiAnim;
    public float walkSpeed, chaseSpeed,
minIdleTime, maxIdleTime, idleTime,
catchDistance, chaseTime, minChaseTime,
maxChaseTime, jumpscareTime;
    public bool walking, chasing;
    public Transform player;
    Transform currentDest;
    public string deathScene;
    public GameObject hideText,
stopHideText;
    Vector3 dest;
    public GhostPatrol ghostPatrol;

    void Start(){
        walking = true;
        /*currentDest =
destinations[UnityEngine.Random.Range(0,
destinations.Count)];*/
    }
    // OnTriggerEnter is called when the player
enters the ghost's trigger collider.
    void OnTriggerEnter(Collider other){
        if (other.CompareTag("Player")){
            walking = false;
            StopCoroutine("stayIdle");
            StopCoroutine("chaseRoutine");
            StartCoroutine("chaseRoutine");
            chasing = true;

            ghostPatrol.enabled = false;
        }
    }

    // OnTriggerExit is called when the player
exits the ghost's trigger collider.
    void OnTriggerExit(Collider other){
        if (other.CompareTag("Player")){
            stopChase();
            ghostPatrol.enabled = false;
        }
    }
    void Update(){
        if (chasing){
            dest = player.position;
            ai.destination = dest;
            ai.speed = chaseSpeed;
            aiAnim.ResetTrigger("walk");
            aiAnim.ResetTrigger("idle");
            aiAnim.SetTrigger("sprint");
            if (ai.remainingDistance <=
catchDistance)
            {
                player.gameObject.SetActive(false)
;
                aiAnim.ResetTrigger("walk");
                aiAnim.ResetTrigger("idle");
                hideText.SetActive(false);
                stopHideText.SetActive(false);
                aiAnim.ResetTrigger("sprint");
                aiAnim.SetTrigger("jumpscare");
                StartCoroutine(deathRoutine());
```

```
            chasing = false;
        }
    }

    if (walking){
        /*dest = player.position;
        ai.destination = dest;
        ai.speed = walkSpeed;
        aiAnim.ResetTrigger("sprint");
        aiAnim.ResetTrigger("idle");
        aiAnim.SetTrigger("walk");
        if (ai.remainingDistance <=
ai.stoppingDistance)
        {
            aiAnim.ResetTrigger("sprint");
            aiAnim.ResetTrigger("walk");
            aiAnim.SetTrigger("idle");
            ai.speed = 0;
            StopCoroutine("stayIdle");
            StartCoroutine("stayIdle");
            walking = false;
        }*/
    }
  }

  public void stopChase(){
    walking = true;
    chasing = false;
    StopCoroutine("chaseRoutine");
    /*currentDest =
destinations[Random.Range(0,
destinations.Count)];*/
  }

  IEnumerator stayIdle(){
    idleTime =
Random.Range(minIdleTime, maxIdleTime);
    yield return new
WaitForSeconds(idleTime);
    walking = true;
    /*currentDest =
destinations[Random.Range(0,
destinations.Count)];*/
  }

  IEnumerator chaseRoutine(){
    chaseTime =
Random.Range(minChaseTime,
maxChaseTime);
    yield return new
WaitForSeconds(chaseTime);
    stopChase();
  }
  IEnumerator deathRoutine(){
    yield return new
WaitForSeconds(jumpscareTime);
    SceneManager.LoadScene(deathScene);
  }
}
```

GhostPatrol.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GhostPatrol : MonoBehaviour
{
  public Transform[] points;
  public int current = 0;
  public float speed = 2.0f; // Adjust the
speed as needed
  public float waypointRadius = 0.1f; //
Adjust the radius for waypoint proximity

  void Update()  {
    // Calculate the distance to the current
waypoint
```

```
        float distanceToWaypoint =
Vector3.Distance(transform.position,
points[current].position);

        // Check if the ghost is close enough to
the waypoint
        if (distanceToWaypoint <
waypointRadius){
            // Move to the next waypoint
            IncreaseTargetInt();
        }
        // Move the ghost toward the current
waypoint
        MoveToWaypoint();
    }

    void MoveToWaypoint(){
        Vector3 direction =
(points[current].position -
transform.position).normalized;
        transform.position += direction * speed
* Time.deltaTime;
    }

    void IncreaseTargetInt(){
        current++;
        if (current >= points.Length){
            current = 0;     }}}
```

MainMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class MainMenu : MonoBehaviour
{
    public GameObject optionMenu;
    public GameObject title;
    public GameObject mainMenu;
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManag
er.GetActiveScene().buildIndex+1);
    }

    public void ShowOptions()
    {
        optionMenu.SetActive(true);
        title.SetActive(false);
        mainMenu.SetActive(false);
    }

    public void QuitGame()
    {
        Debug.Log("quit");
        Application.Quit();
    }
}
```

OnSliderChange.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class OnSliderChange :
MonoBehaviour
{
   public Slider slider;
   public TMP_Text value;

   public void HandleChange()
   {
      value.text = slider.value.ToString();
   }}
```

OptionMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewBehaviourScript :
MonoBehaviour
{
   public GameObject optionMenu;
   public GameObject mainMenu;
   public GameObject title;

   public void Back()
   {
      optionMenu.SetActive(false);
      mainMenu.SetActive(true);
      title.SetActive(true);
   }
}
```

PickUpObject.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PickUpObject : MonoBehaviour
{
   private bool isSeeingObject = false;
   private GameObject pick;
   public GameObject canvas;

   private void Update()
   {
      if (Input.GetMouseButtonDown(0) &&
isSeeingObject)
      {
         canvas.GetComponent<HandleInvent
ory>().AppendList(pick.name);
         pick.SetActive(false);
      }
   }
   private void OnTriggerEnter(Collider
other)
   {
      if (other.CompareTag("pickable")){
         isSeeingObject = true;
         pick = other.gameObject;
      }
   }
   private void OnTriggerExit(Collider
other){
      if (other.CompareTag("pickable")){
         isSeeingObject = false;
         pick = null;
      }
   }
}
```

HandleInventory.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class HandleInventory :
MonoBehaviour
{
    public GameObject inventory;
    private bool isInvOn = false;
    private string[] listofItems = new
string[10];
    private int numOfElementsInList = 0;
    public TMP_Text inventoryTextbox;
    // Update is called once per frame
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.E)) {
            if(isInvOn)
            {
                isInvOn = false;
                inventory.SetActive(false);
            }
            else
            {
                isInvOn = true;
                inventory.SetActive(true);
            }
        }
    }

    public void AppendList(string item)
    {
        listofItems[numOfElementsInList] =
item;
        numOfElementsInList++;
        UpdateList();
    }
    public void UpdateList()
    {
        string listString = "";
        for (int i =0; i<numOfElementsInList;
i++) {
            listString += listofItems[i];
            listString += "\n";
        }
        inventoryTextbox.text = listString;
    }

}
```

CHAPTER 7
Testing

Functional Test Cases:

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result
TC-011	Inventory Management	Functional	Player has items in the inventory	Add an item to the inventory	The item is added to the inventory	The item is added to the inventory
TC-012	Character Interaction	Functional	Character encounters an NPC	Initiate a conversation with an NPC	Dialogue options are displayed, and conversations progress logically	Dialogue options are displayed, and conversations progress logically
TC-013	Object Physics	Functional	Player interacts with physics-based objects	Push, throw, or manipulate physics objects	Objects respond to interactions realistically	Objects respond to interactions realistically
TC-014	Game Save Load	Functional	Player is in the game and has previously saved progress	Save the game, exit, and then load the saved game	The game is saved and loaded accurately, and the player resumes from the saved state	The game is saved and loaded accurately, and the player resumes from the saved state
TC-015	Puzzle Solving	Functional	Player encounters a puzzle	Attempt to solve the puzzle using the correct approach	Solving the puzzle correctly progresses the game	Solving the puzzle correctly progresses the game

Table 7.1

Performance Test Cases:

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result
TC-016	Frame Rate Stability	Performance	Game is running	Play the game, monitor frame rates, and check for any noticeable drops	The game maintains a stable frame rate without significant drops	The game maintains a stable frame rate without significant drops
TC-017	Loading Time	Performance	Loading a new game level	Measure the time it takes to load a level	Loading times are within acceptable limits	Loading times are within acceptable limits
TC-018	Memory Usage	Performance	Game is running	Monitor memory usage during gameplay	Memory usage remains within acceptable limits	Memory usage remains within acceptable limits
TC-019	Network Performance	Performance	Game includes online features	Test online gameplay, check for lag or disconnections	Online gameplay is smooth with minimal lag or disconnections	Online gameplay is smooth with minimal lag or disconnections
TC-020	Resource Utilization	Performance	Game is running	Monitor CPU and GPU utilization	CPU and GPU usage are within acceptable limits	CPU and GPU usage are within acceptable limits

Table 7.2

User Experience (UX) Test Cases:

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result
TC-021	User Interface Clarity	User Experience	Player is in the game	Navigate through the game menu and UI	Game menus and UI elements are clear and intuitive	Game menus and UI elements are clear and intuitive
TC-022	Controls Intuitiveness	User Experience	Player starts the game	Attempt to move, interact, and use in-game controls	Controls are intuitive and easy to use	Controls are intuitive and easy to use
TC-023	HUD Information	User Experience	Player in gameplay	Observe the heads-up display (HUD)	Relevant information on the HUD is easily readable and helpful	Relevant information on the HUD is easily readable and helpful
TC-024	Audio Balance	User Experience	Game is launched	Play the game with different audio settings	Audio levels are balanced, allowing for clear dialogue and sound effects	Audio levels are balanced, allowing for clear dialogue and sound effects
TC-025	Navigation Guidance	User Experience	Player exploring the house	Try to find the way to the next objective	Clear visual or audio cues guide the player to the next objective	Clear visual or audio cues guide the player to the next objective

Table 7.3

Requirement Traceability Matrix

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result	Status
TC-001	Player Movement	Functional	Player starts the game	Move the player character forward	Player character moves forward smoothly	Player character moves forward smoothly	Passed
TC-002	Graphics Quality	Performance	Game is launched	Adjust graphics settings to the highest quality	High-quality graphics with no lag	High-quality graphics with no lag	Passed
TC-003	Audio Effects	Functional	Game is launched	Test game audio with headphones	Immersive and chilling audio experience	Immersive and chilling audio experience	Passed
TC-004	Jump Scares	User Experience	Player exploring the house	Enter a room with a jump scare event	Player experiences a jump scare	Player experiences a jump scare	Passed
TC-005	Lighting Effects	Visual	Game is launched	Enter a dark room	Realistic and eerie lighting effects	Realistic and eerie lighting effects	Passed
TC-006	Ghost Encounters	Functional	Player exploring the house	Encounter a ghost in a specific location	Player encounters a ghostly apparition	Player encounters a ghostly apparition	Passed
TC-007	Saving Progress	Functional	Player in the middle of gameplay	Use the save feature	Game saves the progress correctly	Game saves the progress correctly	Passed
TC-008	Game Performance	Performance	Game is running	Play the game for an extended period	Game doesn't crash or lag	Game doesn't crash or lag	Passed
TC-009	Puzzles and Objectives	Functional	Player exploring the house	Solve a game puzzle or objective	Objective is completed with correct input	Objective is completed with correct input	Passed

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result	Status
TC-010	Storyline Progression	Functional	Player completes certain objectives	Continue with the storyline	Game progresses to the next part of the story	Game progresses to the next part of the story	Passed
TC-021	User Interface Clarity	User Experience	Player is in the game	Navigate through the game menu and UI	Game menus and UI elements are clear and intuitive	Game menus and UI elements are clear and intuitive	Passed
TC-022	Controls Intuitiveness	User Experience	Player starts the game	Attempt to move, interact, and use in-game controls	Controls are intuitive and easy to use	Controls are intuitive and easy to use	Passed
TC-023	HUD Information	User Experience	Player in gameplay	Observe the heads-up display (HUD)	Relevant information on the HUD is easily readable and helpful	Relevant information on the HUD is easily readable and helpful	Passed
TC-024	Audio Balance	User Experience	Game is launched	Play the game with different audio settings	Audio levels are balanced, allowing for clear dialogue and sound effects	Audio levels are balanced, allowing for clear dialogue and sound effects	Passed
TC-025	Navigation Guidance	User Experience	Player exploring the house	Try to find the way to the next objective	Clear visual or audio cues guide the player to the next objective	Clear visual or audio cues guide the player to the next objective	Passed

Table 7.4

CHAPTER 8
Output/Screenshots

Menu Screen:

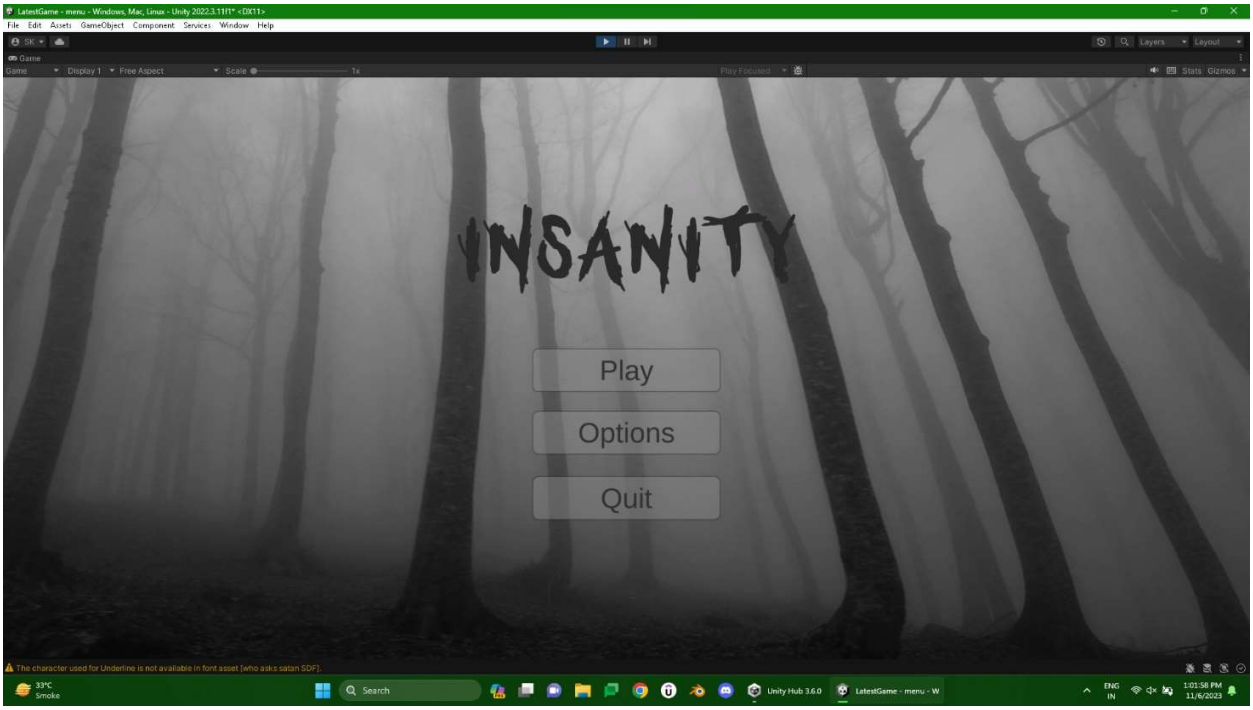

Figure 8.1

House on Terrain:

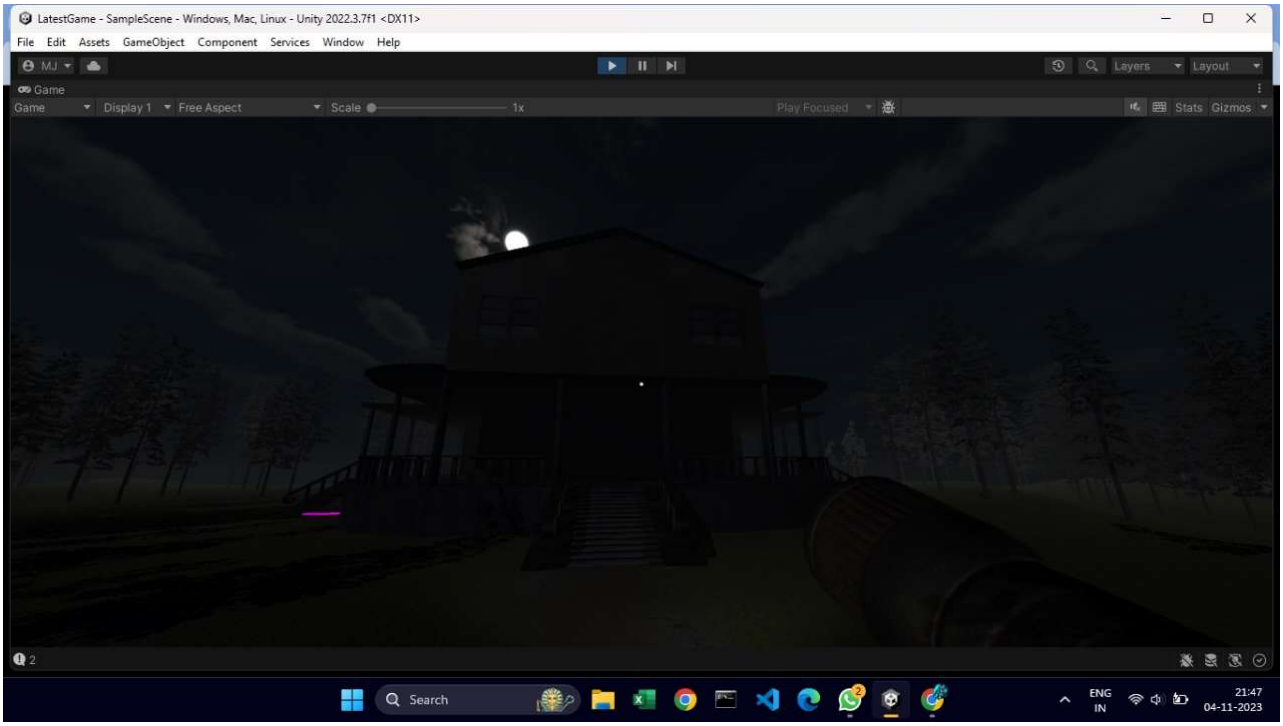

Figure 8.2

House Interior:

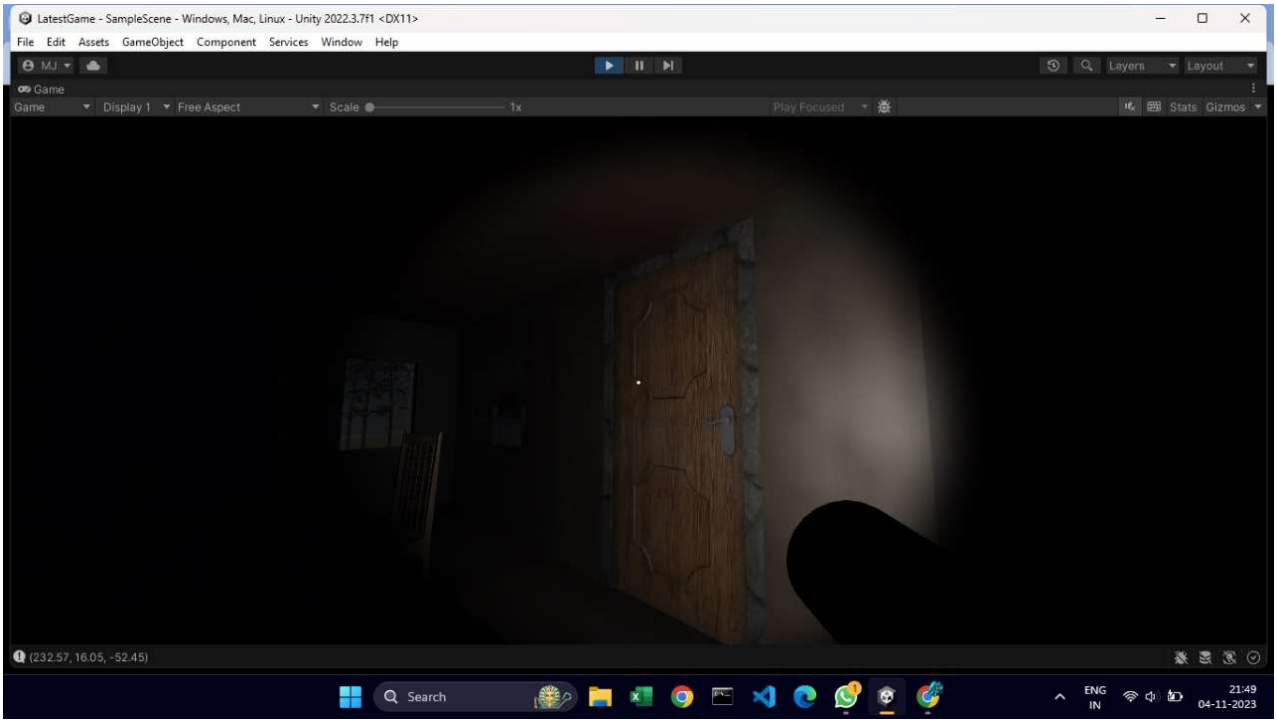

Figure 8.3

Random Wall Paintings:

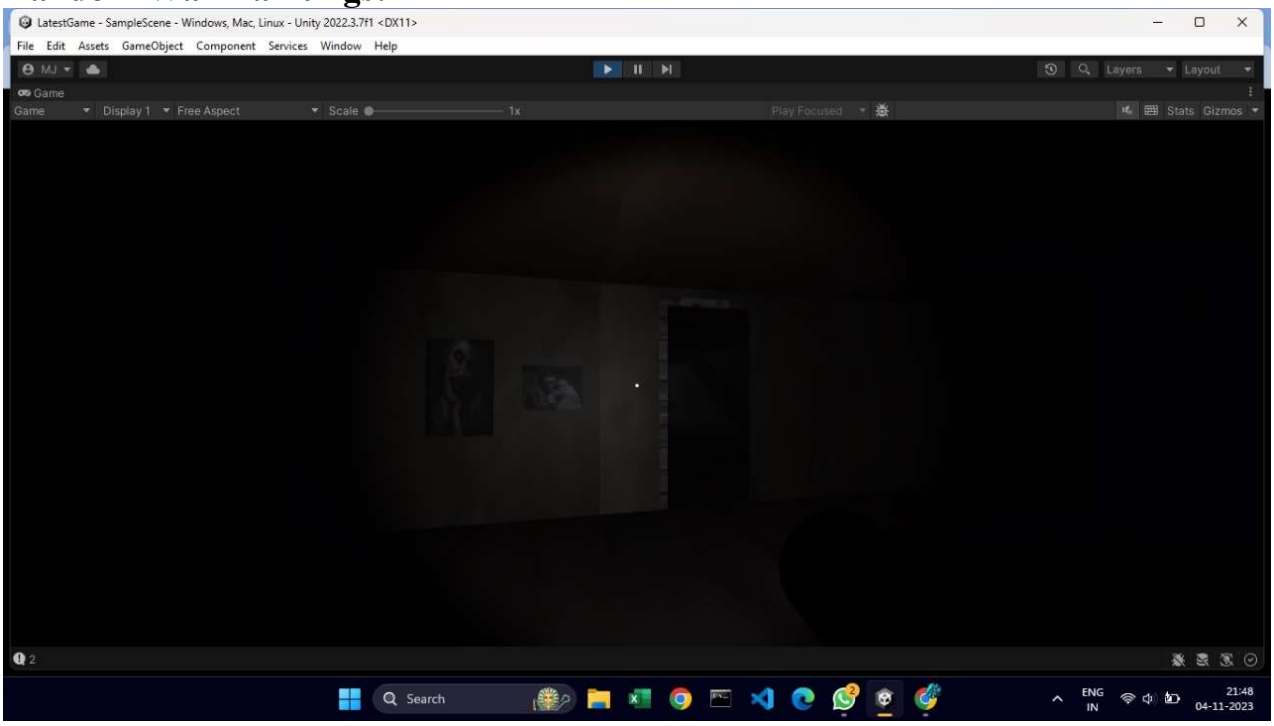

Figure 8.4

Random Room

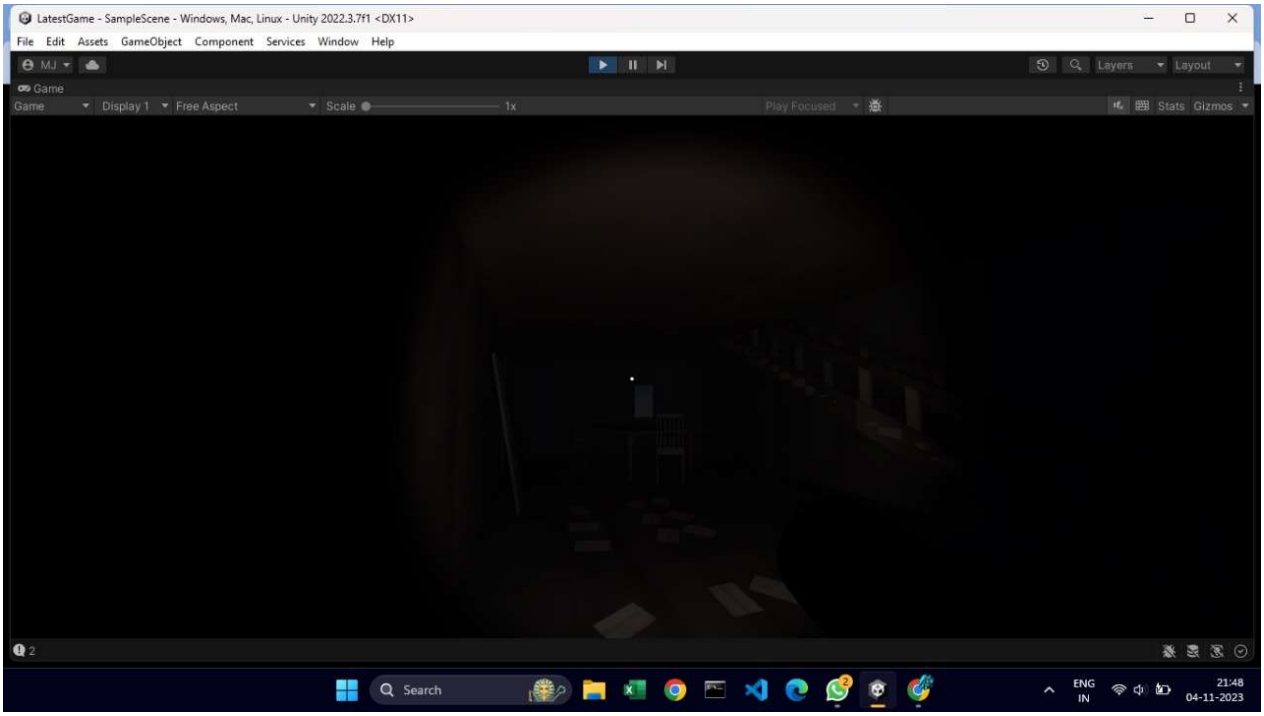

Figure 8.5

Torch:

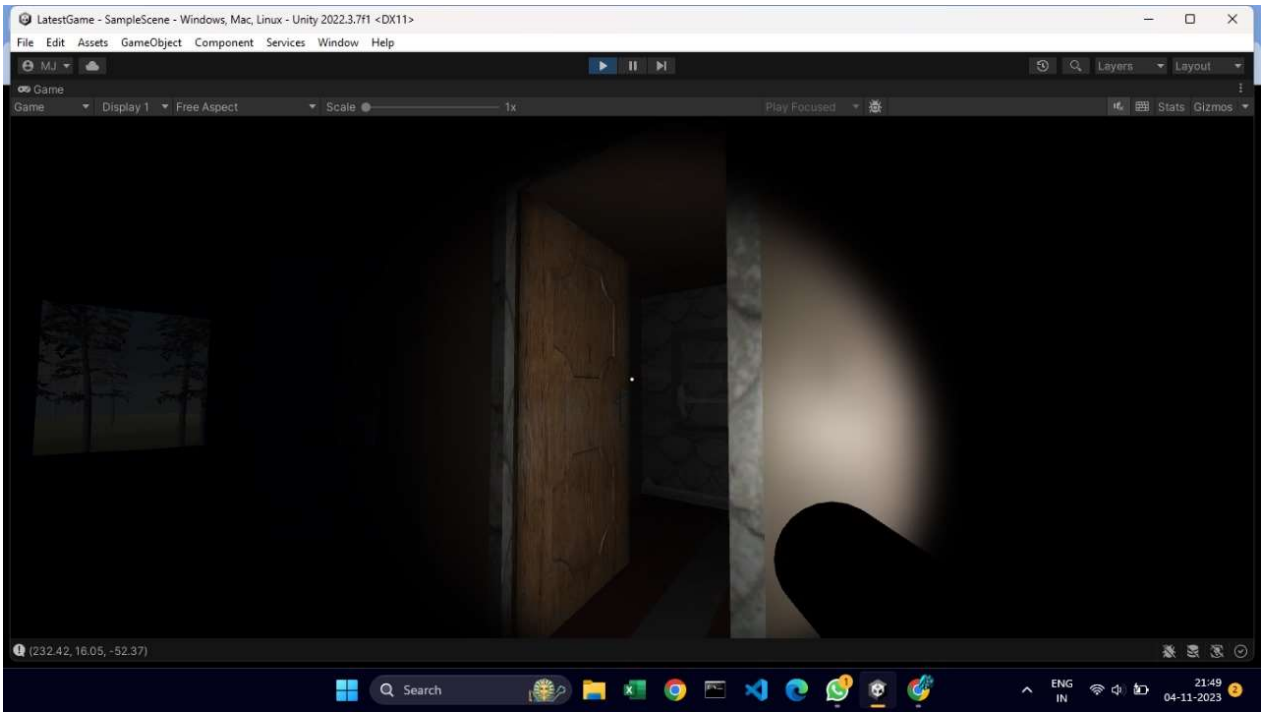

Figure 8.6

Ground Floor:

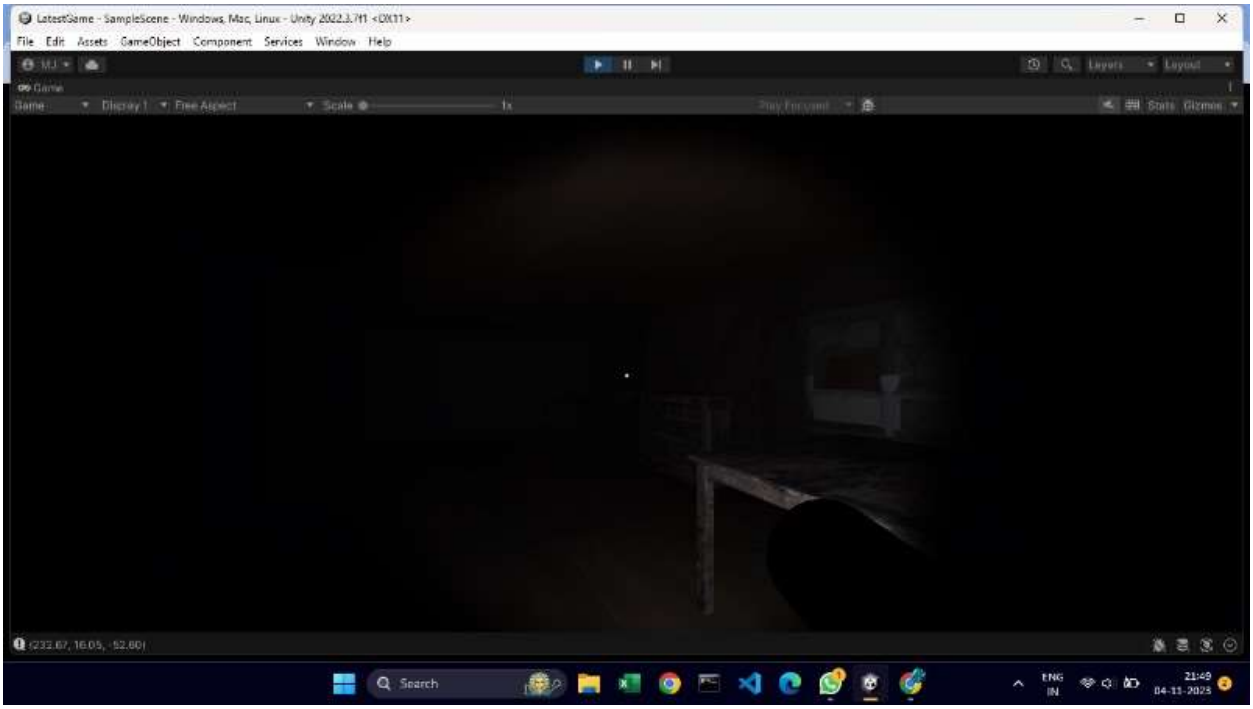

Figure 8.7

Cursed Dolls

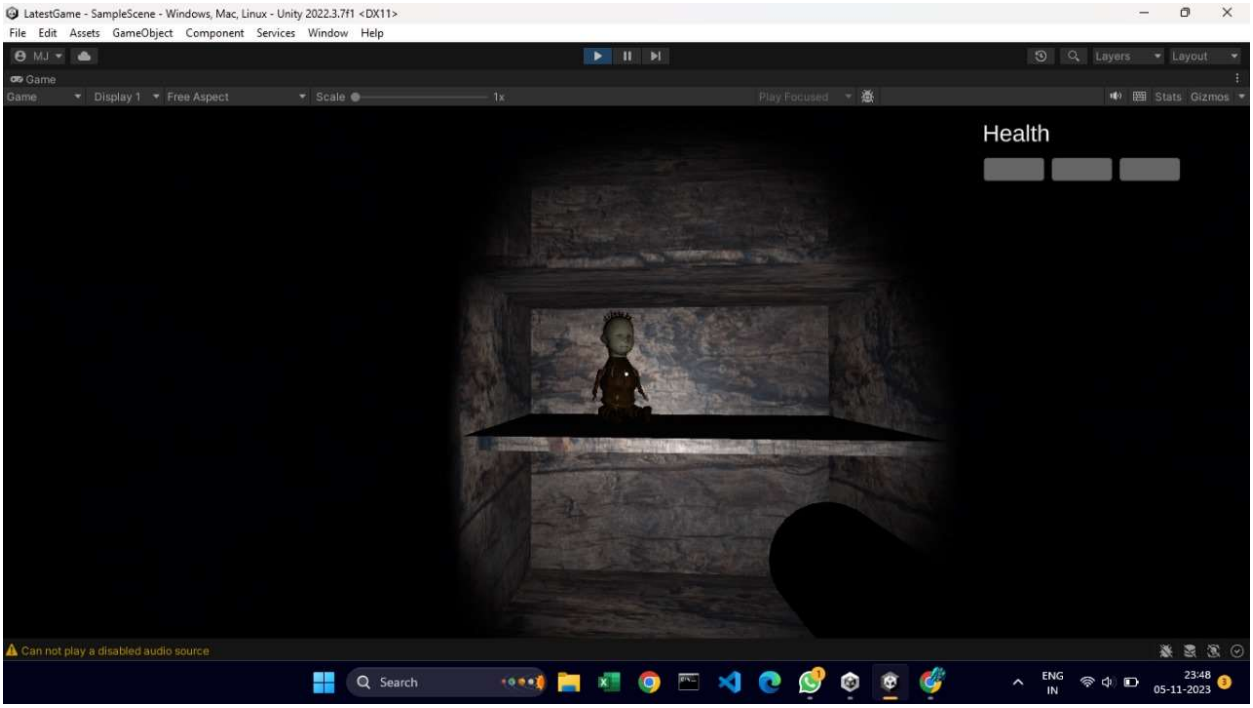

Figure 8.8

Doll Pickup

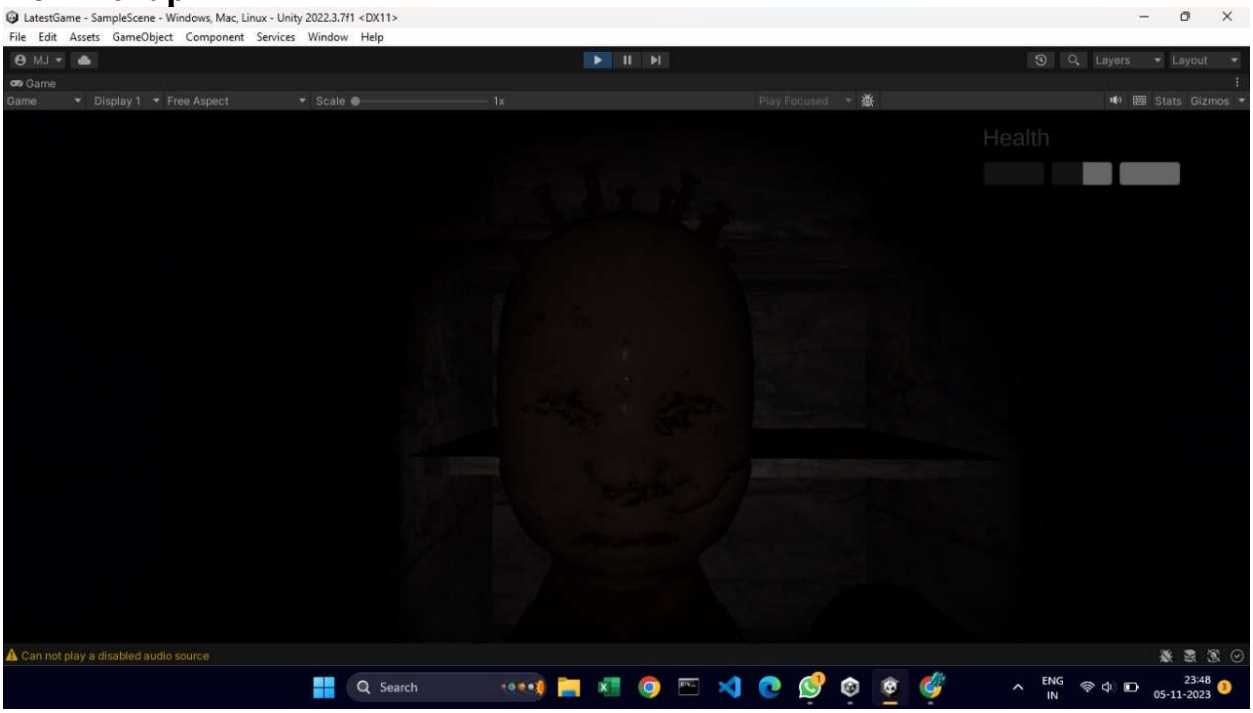

Figure 8.9

Burning Dolls

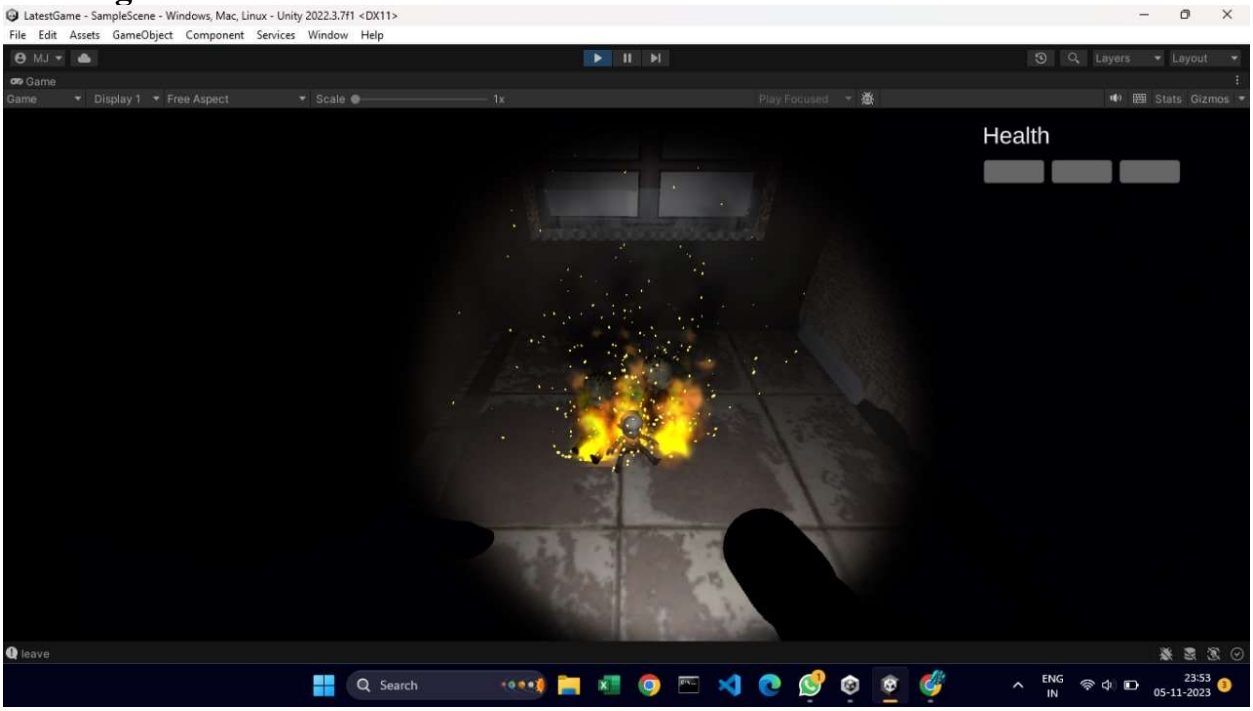

Figure 8.10

CHAPTER 9
Future Scope and Conclusion

Future Scope

As "Insanity - A 3D Horror Game" nears its completion, it's essential to consider its future scope and potential avenues for development and enhancement. The following areas offer exciting possibilities for the game's future:

Additional Game Content: "Insanity" can be expanded with the addition of new levels, haunted environments, and characters. These additions can further challenge players and keep the game fresh for existing and new players.

Multiplayer Model: The inclusion of a multiplayer mode could create opportunities for cooperative or competitive gameplay, adding social dynamics to the horror experience. Players could explore the haunted house together, facing terrifying challenges as a team.

Enhanced Storytelling: Expanding the narrative depth and complexity of the game would further engage players. Deepening the lore and backstories of the haunted house's inhabitants and mysteries could provide a more immersive and emotionally resonant experience.

Mobile Adaptation: Adapting "Insanity" for mobile platforms could make it more accessible to a broader audience. A carefully designed mobile version could retain the core horror elements while optimizing for touch-screen controls and shorter gameplay sessions.

Conclusion:

In conclusion, "Insanity - A 3D Horror Game" has been a thrilling and challenging project that has pushed the boundaries of horror gaming. From meticulously designed environments and high-quality graphics to dynamic gameplay and immersive audio, the game offers an adrenaline-pumping horror experience that keeps players engaged, scared, and excited.

The successful implementation of our methodology, including Agile development and Scrum practices, has allowed us to manage the project effectively and respond to changing requirements and player feedback. Collaboration through cloud tools has enhanced team communication and project coordination.

The game's system features, including a first-person perspective, realistic graphics, and

responsive controls, have created a deeply immersive experience. The narrative-driven exploration, dynamic horror elements, and player choices ensure that players remain engrossed in the game's suspenseful world.

Looking to the future, "Insanity" holds exciting potential for expansion, including additional game content, a multiplayer mode, enhanced storytelling, and adaptation for mobile platforms. These developments can further engage players and extend the life of the game.

In summary, "Insanity - A 3D Horror Game" stands as a testament to the possibilities of horror gaming, providing an unforgettable experience for players while holding the promise of continued innovation and growth in the future.

CHAPTER 10
REFERENCES

REFERENCES

-Unity Game Engine:

- Unity Technologies. (2023). *Unity Real-Time Development Platform* [Online]. Available: https://unity3d.com

- Unity Technologies. (2023). *Real-Time 3D Development Software for Games & More* [Online]. Available: https://unity3d.com/unity

C# Scripting:

- Unity Technologies. (2023-11-01). *Unity – Manual: Scripting* [Online]. Available: https://docs.unity3d.com/Manual/ScriptingSection.html

- Vishnu Sivan. (2022-04-30). *Unity 3D C# scripting cheatsheet for beginners* [Online]. Available: https://blog.devgenius.io/unity-3d-c-scripting-cheatsheet-for-beginners-be6030b5a9ed

Websites:

- Unity Technologies. (2023-11-01). *Unity – Manual: Working in Unity* [Online]. Available: https://docs.unity3d.com/Manual/UnityOverview.html

- Unity Technologies. (2023-11-01). *Unity – Manual: Physics* [Online]. Available: https://docs.unity3d.com/Manual/PhysicsSection.html

- Unity Technologies. (2023-11-01). *Unity – Manual: Audio* [Online]. Available: https://docs.unity3d.com/Manual/Audio.html

- Unity Technologies. (2023-11-01). *Unity – Manual: Animation* [Online]. Available: https://docs.unity3d.com/Manual/AnimationSection.html

- Unity Technologies. (2023-11-01). *Unity – Manual: User Interface* [Online]. Available: https://docs.unity3d.com/Manual/UIToolkits.html

- Unity Technologies. (2023-11-01). *Unity – Manual: Unity's Asset Store* [Online]. Available: https://docs.unity3d.com/Manual/AssetStore.html

- Unity Technologies. (2023). *Resources* [Online]. Available: https://resources.unity.com

Unity Asset Store:

- Unity Technologies. (2023). *Unity Asset Store - The Best Assets for Game Making* [Online]. Available: https://assetstore.unity.com

- SpaceZeta. (2023-08-29). *Street Lamps 2 | 3D Exterior | Unity Asset Store* [Online]. Available: https://assetstore.unity.com/packages/3d/props/exterior/street-lamps-2-260395

- Pieter Ferreira. (2023-01-16). *Modular Kit (Horror House) 104 pieces* [Online]. Available: https://sketchfab.com/3d-models/modular-kit-horror-house-104-pieces-c8194ff536db49fa811b06b498ebb7d1

- Kajaman. (2018-02-18). *Kajaman's Roads - Free - 3D Roadways - Unity Asset Store* [Online]. Available: https://assetstore.unity.com/packages/3d/environments/roadways/kajaman-s-roads-free-52628

- Brittany Bolick. (2018-09-06). *Rusty Flashlight _ 3D Tools _ Unity Asset Store* [Online]. Available: https://assetstore.unity.com/packages/3d/props/tools/rusty-flashlight-122403