# Simple 3D FPS GAME (Zomvoid)

*A*

*Project*

*Report*

Submitted in partial fulfillment of the requirement for the award of degree of

**Bachelor of Computer Application (BCA)**

of

**Kavikulguru Kalidas Sanskrit University.**

*Submitted by*

**Yash Soni**

*Under the guidance of*

**Prof. Sneha Shashikant Lokhande.**

Kavikulguru Kalidas Sanskrit University's

**Bakliwal Foundation Collage of Arts, Commerce & Science**

Vashi.

**BATCH: 2022-2025**

BAKLIWAL FOUNDATION
College of Arts, Commerce & Science

Kavikulaguru Kalidas Sanskrit Vishwavidyalaya's
**Bakliwal Foundation College of Arts, Commerce & Science**
Vashi.

## CERTIFICATE

This is to certify that the project entitled Simple 3D FPS Game  undertaken at the PCP Center: Bakliwal Foundation of Arts, Commerce & Science, Vashi, New Mumbai by Mr. Yash Soni holding PRN No. 202218100096991, Studying Bachelors of Computer Applications Semester – VI has been satisfactorily completed as prescribed by the Kavikulaguru Kalidas Sanskrit University, during the year 2024-2025.

**Project In-charge** **Co-Ordinator**

**External Examiner**

**Internal Examiner** **Principal**

## DECLARATION

I hereby declare that the project entitled, “Simple 3D Fps Game” done at Bakliwal Foundation College of Arts, Commerce & Science, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university. The project is done in partial fulfillment of the requirements for the award of degree of BACHELOR OF COMPUTER APPLICATIONS to be submitted as final semester project as part of our curriculum.

Yash Soni

## ACKNOWLEDGEMENT

We greatfully acknowledge the support and guidance of my teachers H.O.D Prof Sneha Shashikant Lokhande, Prof Kalyani Akshay Kulkarni, Prof Divya Patil, Prof Shaikh Mohammed Umar and Mr Ankit Shirwastav. Also would like to acknowledge the open source community whose tools and resources made this project possible.I extend my sincere thanks to our principal Dr. Sharadkumar Shah.

## ABSTRACT

This project presents the development of a simple 3D fps (first person shooter) game , designed to demonstrate core gameplay mechanics such a player movement , shooting , basic AI behaviour and User Interface. Built using open source tools including Godot game engine , Blender and Gimp.

# Chapter 1: Introduction

## 1.1 Background:

Zomvoid is a 3D first-person horror game set in an abstract place called the ”Void”. The player takes on the role of a nomad navigating this unique environment. The game is being developed using entirely open-source tools, including Blender for 3D modeling, GIMP for texture and image editing, and Godot with GDScript for game engine and scripting. This commitment to open-source tools aims to demonstrate the capabilities of freely available software in creating high-quality game experiences.

## 1.2 Objectives:

The primary objective of Zomvoid is to create an engaging and immersive FPS experience, despite being developed with only open-source tools. Specific objectives include:

Design and develop a visually striking and cohesive game world using Blender and GIMP.

Implement smooth and responsive FPS mechanics within the Godot game

engine.. Create AI for enemies.

Optimize game performance for a smooth player experience.

Showcase the power and potential of open-source game development tools.

## 1.3 Purpose, Scope, and Applicability:

### 1.3.1 Purpose:

The purpose of Zomvoid is to provide players with an entertaining and unique gaming experience. It also serves as a demonstration of the viability of open-source tools in game development, potentially encouraging other developers to explore these resources. The game aims to deliver a complete gameplay loop, from exploration to combat.

### 1.3.2 Scope:

The scope of Zomvoid includes the creation of a single-player experience with the following key features:

A explorable game world with distinct locations and

environments. NPC interaction, with dialogue.

Combat mechanics, featuring a limited selection of weapons and enemy types.

### 1.3.3 Applicability:

Zomvoid is applicable to players who enjoy surreal FPS games, exploration, and basic combat gameplay. Its cold, minimalist survival aesthetic and apocalyptic harsh setting may appeal to players seeking unique and visually distinct gaming experiences. Additionally, the game can be used to showcase the capabilities of open-source game development tools in educational settings or industry demonstrations.

## 1.4 Achievements:

The project aims to achieve the following:

A fully playable game demo with the core mechanics

implemented. A compelling game world with a distinct

visual style.

Functional AI for enemies, providing a challenge to the

player. Optimization of the game to ensure smooth

performance.

A well-documented development process, highlighting the use of open-source tools.

## 1.5 Organization of Report:

The remainder of this blackbook is organized as follows:

Chapter 2 provides a survey of the technologies used in the project. Chapter 3 details the requirements and analysis of the game.

Chapter 4 outlines the system design.

Chapter 5 covers the implementation and testing phases. Chapter 6 presents the results and discussion.

Chapter 7 concludes the blackbook and discusses future scope. References, a glossary, and appendices are included at the end.

# Chapter 2: Survey of Technologies

Blender: Blender is a free and open-source 3D creation suite. It was used for all 3D modeling, animation, and rigging.

GIMP (GNU Image Manipulation Program): GIMP is a free and open-source image editor. It was used for creating and editing textures for 3D models, as well as for any 2D image assets.

Godot: Godot is a free and open-source game engine. It was used as the primary engine for game development, including scene management, physics, and scripting.

GDScript: GDScript is a scripting language used within the Godot engine. It was used to implement game logic, player controls, AI behavior, and interactions.

# Chapter 3: Requirements and Analysis

## 3.1 Problem Definition:

Zomvoid addresses the challenge of creating a compelling and complete FPS horror game experience using only open-source tools. The game aims to provide a unique setting, engaging gameplay, and a compelling world, demonstrating that high-quality games can be developed without relying on expensive, proprietary software.

## 3.2 Requirements Specification:

Functional Requirements:

Player movement and camera

control. Weapon system.

Enemy AI with pathfinding and combat

behavior. NPC interaction with dialogue.

User interface (UI) for displaying information and

menus. Sound effects and music.

Non-Functional Requirements:

Smooth and consistent frame

rate. Responsive controls.

Intuitive user interface.

Visually appealing

graphics. Stable

performance.

## 3.3 Planning and Scheduling:

Setp-1 : creating assest and textures using blender

and gimp Step-2 : writing game logic and

mechanisms

Step-3: putting it all together and testing

## 3.4 Software and Hardware Requirements:

**Hardware Requirements;**

I) i3-12th Generation or Above

II) GPU That Supports Vulkan 3.0

III) 4GB Ram

**Software Requirements:**

Windows 10 or Above / Linux

## 3.5 Preliminary Product Description:

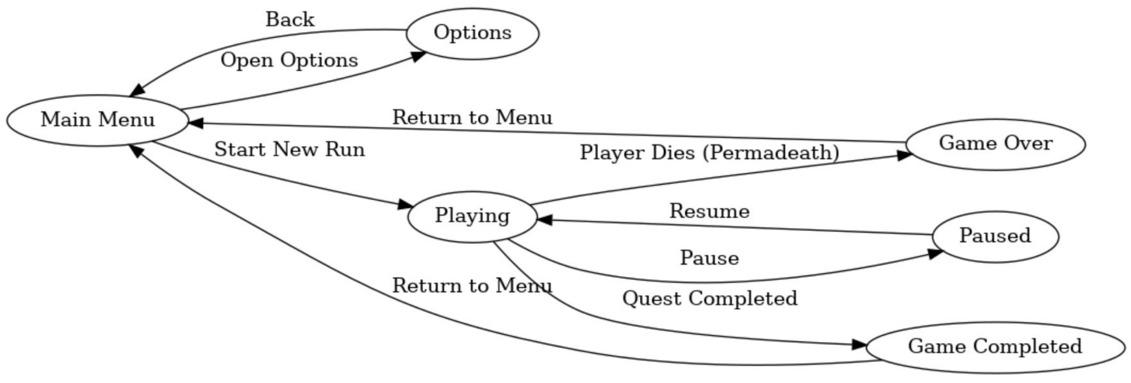
Zomvoid is a single-player Horror FPS game where the player explores the void. The player kills voidwalkers and collect 2 artifacts. The game features a psx low-poly apocalyptic harsh survival aesthetic, a focus on exploration, and a mix of combat and survival gameplay.

## 3.6 Conceptual Models:

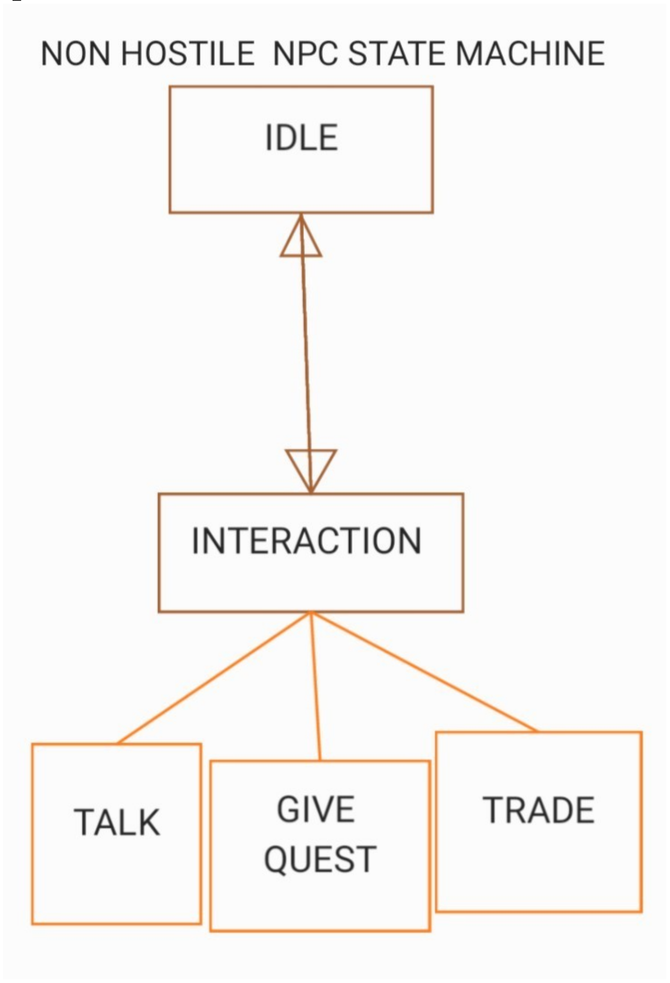
State

Diagrams:

game states:



npc states:



# Chapter 4: System Design

## 4.1 Basic Modules:

The game is structured into the following modules:

Player Module: Handles player movement, camera control, and

player stats. Weapon Module: Manages weapon selection, firing,

and reloading.

Enemy Module: Controls enemy AI, pathfinding, and combat

behavior. NPC Module: Handles NPC interaction, dialogue.

UI Module: Displays game information, menus, and HUD elements.

## 4.2 Data Design:

### 4.2.1 Schema Design:

The game's data will be stored in a structured format,using Godot's built-in resource system. This will include data for:

Player: Position, health, ammo,

map. NPCs: Dialogue.

World: Level data, spawn points, environmental information.

### 4.2.2 Data Integrity and Constraints:

Data integrity will be ensured through:

Data validation to prevent invalid values.

Relationships between data tables to maintain

consistency. Save data integrity checks to prevent

corruption.

## 4.3 Procedural Design:

### 4.3.1 Logic Diagrams:

Control flow charts and state diagrams will be used to illustrate the flow of logic in key systems, such as:

Enemy AI behavior.

Player-NPC
interaction.

### 4.3.2 Data Structures:

Appropriate data structures will be used to efficiently store and manage game data, including:

Arrays and lists for storing collections of

objects. Dictionaries for mapping data.

Custom data structures for specific game elements.

### 4.3.3 Algorithms

**Design:** Algorithms will be designed for key game

mechanics, such as: Enemy pathfinding (e.g., A*

algorithm).

Collision
detection.

## 4.4 User Interface Design:

The user interface will be designed to be intuitive and informative, providing the player with essential game information without being intrusive. Key elements include:

HUD (Heads-Up Display): Displays health, and other relevant

information. Menus: For navigation, settings, and pause /

resume.

Dialogue boxes: For displaying NPC dialogue.

## 4.5 Security Issues:

As a single-player game, security concerns are minimal. However, measures will be taken to:

Prevent save data corruption.

Ensure the integrity of game

files.

## 4.6 Test Cases Design:

Test cases will be designed to cover all aspects of the game,

including: Player movement and controls.

Weapon
functionality.

Enemy AI
behavior.

NPC interaction .

Resume/Pasue

functionality. UI

elements.

# Chapter 5: Implementation and Testing

## 5.1 Implementation Approaches:

The game will be implemented using an iterative development process, with a focus on building and testing core mechanics early on. Code will be written in GDScript, following a modular and organized structure.

## 5.2 Coding Details and Code Efficiency:

GDScript code will be written with clear comments and organized into functions and classes. Optimization techniques will be employed to ensure smooth performance. The program code will contain comments explaining the work a piece of code does, why it does it, or why it does it a particular way. The function of the code will be explained with a shot of the output screen of that program code. The efficiency of the code and how code optimization is handled will also be explained.

## 5.3 Testing Approach:

Testing will be conducted according to the scheme presented in the system design chapter and will follow a suitable model (e.g., category partition, state machine-based). Both functional testing and user-acceptance testing will be employed.

### 5.3.1 Manual Testing

The game was tested manually throughout the development process to ensure stability and playability. Each feature—ranging from core movement and interaction systems to UI elements and enemy behavior—was manually verified through repeated in-game sessions. This hands-on approach allowed the developers to catch bugs, tune difficulty, and improve the overall player experience iteratively during implementation.

### 5.3.2 Closed Alpha Testing:

A closed beta testing phase was conducted with a small group of friends and family to gather feedback on gameplay, performance, and overall user experience. Testers were encouraged to play the game in various ways and report any bugs, confusion, or suggestions. This feedback loop helped shape the final adjustments before project completion.

# Chapter 6: Results and Discussion

## 6.1 Test Reports:

1: there were bugs with collision shape

2: movement felt a little odd due to head bob  (now removed)

## 6.2 User Documentation:

W - move forward

A - move left

S - move back

D – move right

Space – to jump

Shift – to sprint

E – to interact

# Chapter 7: Conclusions

## 7.1 Conclusion:

Zomvoid is a 3D horror FPS game developed using only open-source tools, demonstrating the capabilities of these tools in creating a complete and engaging game experience. The game features a unique setting, compelling gameplay, and a cold, minimalist aesthetic.

## 7.2 Limitations of the System:

1: smaller map

2: basic enemy model

3: lack of animations due to time constraints

4: limited weapons

## 7.3 Future Scope of the Project:

Future development could include:

Expanding the game world with new locations and

survival tasks. Adding more weapons and enemy types.

Implementing additional game mechanics..

## REFERENCES

Games: Stalker series, Metro Trilogy, Halo Trilology and The Long Dark

Other Media : Snowpiercer.

## GLOSSARY

State Machines : a Game AI Behavoiur

alogorithm 3D modeling : process of

creating 3d models textures : Images

projected on 3d models

APPENDICES

player movement code:

```
extends CharacterBody3D

@export var move_speed = 5.0
@export var sprint_multiplier =
2.0 @export var jump_velocity =
4.5 @export var
mouse_sensitivity = 0.003

var move_velocity = Vector3.ZERO # Renamed variable
var gravity =
ProjectSettings.get_setting("physics/3d/default_gravity") var
is_sprinting = false

@onready var head = get_node("head")
@onready var camera = get_node("head/Camera3D")

func _ready():
	Input.mouse_mode = Input.MOUSE_MODE_CAPTURED

func _unhandled_input(event):
        if event is InputEventMouseMotion:
                rotate_y(-event.relative.x *
                mouse_sensitivity) head.rotate_x(-
                event.relative.y * mouse_sensitivity)

                head.rotation.x = clamp(head.rotation.x,
                deg_to_rad(-90), deg_to_rad(90))


func
        _physics_process(
        delta): # Apply
        gravity
        if not is_on_floor():
                move_velocity.y -= gravity * delta # Use the renamed variable
        else:
          move_velocity.y = 0.0 # Reset vertical velocity when on the ground

        # Handle jump
        if is_on_floor() and Input.is_action_just_pressed("ui_accept"):
                move_velocity.y = jump_velocity # Use the renamed
                variable

        # Handle movement input
        var direction = Vector3.ZERO
        if

                Input.is_action_pressed("

                forward"): direction.z -= 1

        if

                Input.is_action_pressed("b

                ackward"): direction.z += 1

        if

                Input.is_action_press

                ed("left"): direction.x

                -= 1

        if

                Input.is_action_presse

                d("right"): direction.x

                += 1


        # Normalize and move in the camera's forward direction
        if direction.length() > 0:

                direction = direction.normalized().rotated(Vector3.UP,

                rotation.y) if Input.is_action_pressed("sprint"):

                    is_sprinting = true

                else:

                    is_sprinting = false

                move_velocity.x  =  direction.x  *  move_speed  *
(sprint_multiplier  if is_sprinting else 1.0) # Use the renamed variable
```

```
            move_velocity.z = direction.z * move_speed *
(sprint_multiplier if is_sprinting else 1.0) # Use the renamed variable

        else:

            move_velocity.x = lerp(move_velocity.x, 0.0, 0.2) # Use the renamed
variable

            move_velocity.z = lerp(move_velocity.z, 0.0, 0.2) # Use the renamed
variable


        velocity = move_velocity # Assign our calculated velocity to
            the CharacterBody3D's built-in velocity

        move_and_slide()
```

Grass Shader To Make It Sway :-

```
shader_type spatial;

render_mode  blend_mix,   depth_draw_opaque,cull_disabled,
             diffuse_burley, specular_schlick_ggx;


uniform vec4 albedo : source_color;

uniform sampler2D texture_albedo : source_color, filter_linear_mipmap,

repeat_enable; uniform ivec2 albedo_texture_size;

uniform float point_size : hint_range(0.1, 128.0, 0.1);


uniform float roughness : hint_range(0.0, 1.0);

uniform sampler2D texture_metallic    :
        hint_default_white,         filter_linear_mipmap, repeat_enable;

uniform vec4 metallic_texture_channel;

uniform sampler2D texture_roughness   :               hint_roughness_r,
        filter_linear_mipmap, repeat_enable;


uniform float specular : hint_range(0.0, 1.0,

0.01); uniform float metallic :

hint_range(0.0, 1.0, 0.01);


uniform   sampler2D   texture_emission  :     source_color,
          hint_default_black, filter_linear_mipmap, repeat_enable;

uniform vec4 emission : source_color;
uniform float emission_energy : hint_range(0.0, 100.0, 0.01);


uniform sampler2D texture_noise : hint_default_white;
uniform vec3 uv1_scale; uniform vec3 uv1_offset; uniform vec3 uv2_scale; uniform vec3
uv2_offset;


uniform vec2 wind_direction = vec2(1, -0.5);

uniform float wind_speed = 1.0;

uniform float wind_strength =

2.0; uniform float noise_scale =

20.0;


instance uniform float camera_bend_strength : hint_range(0.0, 3.0) = 0.2;


varying float color;

varying float

height;


void vertex() {

        height = VERTEX.y;

        float influence = smoothstep(0, 1, height / 2.0);

        vec4 world_pos = MODEL_MATRIX * vec4(VERTEX,

        1.0); vec2 uv = world_pos.xz / (noise_scale + 1e-2);

        vec2 panning_uv = uv + fract(TIME * wind_direction *

        wind_speed); float wind = texture(texture_noise,
        panning_uv).r * 2.0 - 0.4;

        color = texture(texture_noise, uv).r;
        vec2 wind_offset = -wind_direction * wind_strength * influence * wind;
        world_pos.xz += wind_offset;
        world_pos.y -= wind * influence * smoothstep(0.0, height, wind_strength);


        //Push the top vertex away from the camera to bend the grass clump

        float    ndotv    =    1.0    -    dot(vec3(0.0,    1.0,
                 0.0), normalize(INV_VIEW_MATRIX[1].xyz));

        world_pos.xz += INV_VIEW_MATRIX[1].xz * camera_bend_strength * height
    * ndotv;


        vec4 local_pos = inverse(MODEL_MATRIX) * world_pos;

        local_pos.x += wind_strength * influence * cos(TIME * 1.0)

        / 8.0; local_pos.z += wind_strength * influence * sin(TIME

        * 1.5) / 8.0;


        VERTEX = local_pos.xyz;

        //NORMAL = vec3(0.0, 1.0, 0.0);

}


void fragment() {

        vec2 base_uv = UV;


        vec4 albedo_tex = texture(texture_albedo,

        base_uv); ALBEDO = albedo.rgb * albedo_tex.rgb;
        float metallic_tex = dot(texture(texture_metallic,
                 base_uv), metallic_texture_channel);
        METALLIC = metallic_tex * metallic; SPECULAR = specular;


        vec4 roughness_texture_channel = vec4(1.0, 0.0, 0.0, 0.0);

        float roughness_tex = dot(texture(texture_roughness,
                 base_uv), roughness_texture_channel);

        ROUGHNESS = roughness_tex * roughness;



        // Emission: Enabled

        vec3 emission_tex = texture(texture_emission, base_uv).rgb;

        // Emission Operator: Add

        EMISSION = (emission.rgb + emission_tex) *

        emission_energy; ALPHA *= albedo.a * albedo_tex.a;

}
```

Github: https://github.com/SmartApe251198/zomvoid

Game Screenshots: