Aarushi Singh (17381)

Shina Sharma (17427)

Kali Gupta (1740)

|  |  |
| --- | --- |
| ZLATAn  An interactive game made by using pygame |  |

**Introduction:**

In the annals of gaming history, few titles hold as much reverence and nostalgia as the Wolfenstein series. Pioneering the first-person shooter (FPS) genre, Wolfenstein 3D introduced players to a world of intense action, labyrinthine levels, and memorable enemies, setting the stage for the evolution of modern gaming as we know it. Inspired by the legacy of Wolfenstein and fueled by a passion for game development, this report embarks on a journey into the heart of classic FPS gaming through the lens of Pygame, a versatile library for Python programming.

The allure of Wolfenstein lies not only in its groundbreaking gameplay but also in its ability to captivate players with its immersive world and relentless combat. With Pygame as our canvas and Python as our brush, we seek to pay homage to this legendary franchise while crafting a unique and engaging FPS experience that resonates with both veteran gamers and newcomers alike.

Through a meticulous exploration of game mechanics, level design, enemy AI, weapon systems, and more, we delve deep into the art and science of creating a Wolfenstein-inspired FPS game using Pygame. Our goal is to capture the essence of classic Wolfenstein titles while infusing our creation with modern elements and innovations that elevate the gaming experience to new heights.

As we embark on this Pygame odyssey, we invite you to join us on a quest to relive the glory days of FPS gaming, to challenge the odds, and to emerge victorious in the face of adversity. Together, let us unleash the spirit of Wolfenstein and forge a new legend in the ever-expanding realm of gaming.

Join us as we embark on a journey into the heart of classic FPS gaming, inspired by the legendary Wolfenstein series and brought to life through the power of Pygame.

**Background :**

The Wolfenstein series, developed by id Software, played a pivotal role in shaping the FPS genre. Known for its fast-paced action, challenging gameplay, and memorable characters, Wolfenstein has left an indelible mark on gaming history. Drawing inspiration from its rich legacy, our goal is to infuse our FPS game with elements that evoke the spirit of Wolfenstein while offering a fresh and innovative gameplay experience.Pygame is a popular library for Python programming language that facilitates game development. It provides modules and functions for handling graphics, sound, input devices, and other aspects crucial for game creation. By leveraging Pygame's capabilities, developers can focus on implementing game logic and mechanics without the need to deal with low-level details.

**Technology Stack:**

Python: Python serves as the primary programming language for developing the game logic, mechanics, and systems. Its simplicity and readability make it ideal for rapid prototyping and iteration.

Pygame: Pygame provides developers with tools and functionalities for handling graphics, sound, input devices, and more. It simplifies game development tasks and enables developers to focus on implementing gameplay features inspired by the Wolfenstein franchise.

Additional Libraries: Depending on the requirements of the game, developers may incorporate other Python libraries for tasks such as AI programming, physics simulation, or networking to enhance the game experience.

Getting Started with Pygame:

The first step in developing an FPS game with Pygame is setting up the development environment. This involves installing Python and Pygame, creating a new project directory, and organizing the project structure. Once the environment is set up, developers can begin coding the game's core functionalities. While the original Wolfenstein game did not use Pygame (as it was released long before Pygame existed), we can create a similar game using Pygame, which is a popular Python library for game development.

**Rendering Engine**: Pygame provides a simple and efficient way to handle graphics rendering. Wolfenstein's iconic 3D graphics could be implemented using Pygame's drawing functions or by utilizing more advanced techniques such as rendering textured polygons.

**Input Handling**: Pygame allows easy handling of keyboard and mouse input, which is crucial for controlling the player character and interacting with the game world in Wolfenstein.

**Collision Detection**: Pygame provides collision detection functions that can be used to handle interactions between the player character, enemies, and the environment in Wolfenstein's maze-like levels.

**Audio Support**: Pygame includes modules for handling sound effects and music, allowing developers to implement the iconic sound design of the Wolfenstein series.

**Game Loop**: Pygame's main loop structure is well-suited for running the game logic, updating the game state, and rendering frames, just like in the original Wolfenstein.

**Level Design and Asset Management**: Pygame facilitates loading and managing game assets such as textures, sprites, and level data, which would be essential for creating Wolfenstein's levels and environments.

**User Interface**: Pygame can be used to create user interfaces for menus, HUD elements, and other in-game displays, allowing for a polished and immersive player experience.

**Networking (optional)**: While the original Wolfenstein was primarily a single-player game, modern versions might include multiplayer features. Pygame can be used to implement networking functionality for multiplayer modes.

Overall, Pygame provides a versatile and powerful framework for developing games like Wolfenstein, offering the necessary tools for graphics, input handling, audio, and more. With its simplicity and ease of use, Pygame could be an excellent choice for creating a modern homage to the classic Wolfenstein series.

**Basic information about game:**

The game consists of 5 main files which are

Main.py: All the files and functions operators are being called inside the main file.

Player.py: The player movements are controlled under this file.

Object\_renderer.py: All the inside functions are declared here.

Raycasting.py: To provide player with frontal view and watch view.

Map.py: To make a map for the given game.

**Player Movement:**

Player movement is fundamental to the FPS experience, allowing users to navigate the game world and interact with their surroundings. In Pygame, player movement can be implemented using keyboard input to control the player character's position and orientation. Techniques such as collision detection and velocity adjustment are employed to ensure smooth and responsive movement. Here we set some initial setting of the player and we will design the position of the player on the map.

The W,A,S,D keys are to be used for movement   
the angle of his direction

The speed of his movement

The speed of his rotation

The movement of the player is decided by calculating its movement directions by:  
dx = speed\*cos(a)

Dx= speed\*sin(a)

We will calculate so for every key

W: dx = speed\*cos(a)

Dx= speed\*sin(a)

A: dx = +speed\*cos(a)

Dx= -speed\*sin(a)

S: dx = -speed\*cos(a)

Dx= -speed\*sin(a)

D: dx = -speed\*cos(a)

Dx= +speed\*sin(a)

If we want the players movement speed to be independent of the frame rate then we need to get delts time value for each frame

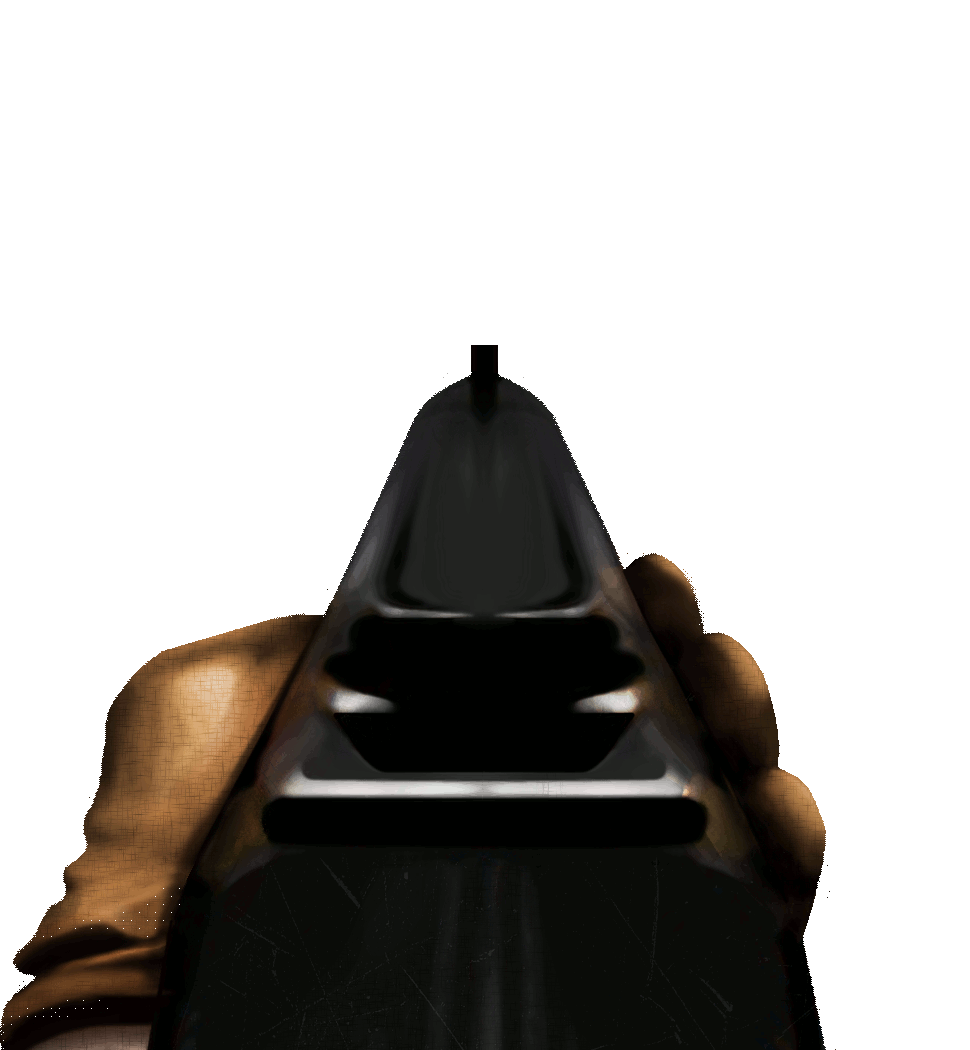
Delta time is the amount of time that has passed since the last frame

The direction of movement as a line and the player is in the form of circle

The player was passing from the walls so to prevent that we pass a function of collision

**Implementing Weapons:**

Weapons play a central role in FPS games, providing players with the means to engage enemies and overcome challenges. In Pygame, weapons can be implemented as objects with properties such as damage, firing rate, and ammunition capacity. Additionally, visual and auditory effects can be added to enhance the weapon's realism and impact. For our game we have chose a weapon which is then designed to reload on its own and work if clicked on left mouse button. The image is attached down below.



**Designing Enemy AI:**

Enemy artificial intelligence (AI) is crucial for creating engaging and challenging gameplay experiences in FPS games. In Pygame, enemy AI can be implemented using various techniques such as finite state machines, pathfinding algorithms, and behavior trees. By simulating intelligent enemy behaviors, developers can create dynamic and immersive combat scenarios. We firstly spawned the NPC and then applied logics for the movement after that we will add the death and other sounds when being hit by the player. We have further added the health of the NPC.

To be able to find the player we needed to add a file which was:

Pathfinding : We will use the algorithm where we will add all the tiles which are not visited in the queue and then make a dictionary out of it further till it reaches its goal point and then sort it in an order such that the shortest path can be found and then further implemented.

when more NPCs are added then they were getting combined at some point to tackle that problem we found the solution to get the value to be false so that two NPC won’t be at the same position.



**User Interface and Feedback:**

A well-designed user interface (UI) enhances the overall gaming experience by providing players with essential information and feedback. In Pygame, UI elements such as health bars, ammo counters, and interactive menus can be implemented using text rendering and sprite graphics. Effective UI design ensures that players remain informed and engaged throughout the game. Designing an effective user interface (UI) and implementing feedback systems are crucial aspects of creating a first-person shooting (FPS) game using the Pygame library in Python. The UI should provide players with essential information and feedback, while the feedback systems ensure that players are aware of their actions and the game state. Below, I'll outline how you can approach designing the UI and implementing feedback systems in your FPS game:

User Interface (UI) Design:

Health and Ammo Display: Include visual indicators for the player's health and ammo status. This could be represented by bars or numerical values displayed on the screen.

Weapon Inventory: Show the player's current weapon and available weapons in their inventory. This could be displayed as icons or text.

Objective Tracker: If your game includes objectives or missions, provide a tracker or HUD element to inform players of their current objectives and progress.

Minimap: Consider adding a minimap to help players navigate the game world and locate enemies, objectives, and points of interest.

Menu System: Implement menus for options, settings, and pausing the game. These menus should be intuitive to navigate and provide players with control over various game settings.

Feedback Systems:

Visual Feedback:

Hitmarkers: Display visual cues when the player successfully hits an enemy, indicating that their shots have landed.

Damage Indicators: Show indicators on the screen to inform players of the direction from which they're taking damage.

Animations: Use animations to convey important events, such as reloading weapons, picking up items, or completing objectives.

Auditory Feedback:

Weapon Sounds: Differentiate weapon sounds to provide auditory feedback on the type of weapon being used.

Enemy Alerts: Use sound cues to alert players to the presence of enemies, such as footsteps or enemy shouts.

Environmental Sounds: Implement ambient sounds to enhance immersion and provide feedback on the game world's atmosphere.

HUD Elements:

Crosshair: Provide a crosshair in the center of the screen to help players aim accurately.

On-Screen Text: Display text notifications for important events, such as picking up items, completing objectives, or entering new areas.

Objective Updates: Update objective-related HUD elements to reflect changes in the player's progress and current objectives.

Customization and Accessibility:

Options Menu: Allow players to customize UI elements, controls, graphics settings, and audio preferences through an options menu.

Subtitles and Accessibility Options: Include options for subtitles, colorblind mode, and other accessibility features to accommodate a diverse range of players.

Testing and Iteration:

Playtesting: Conduct thorough playtesting sessions to gather feedback on the UI and feedback systems from players. Use this feedback to identify areas for improvement and iterate on the design.

User Experience (UX) Testing: Pay attention to how players interact with the UI and feedback systems, ensuring that they are intuitive, informative, and enhance the overall gameplay experience.

By carefully designing the UI and implementing effective feedback systems, you can enhance the immersive experience of your FPS game and provide players with the information they need to navigate the game world, engage in combat, and complete objectives effectively. Iterating on the design based on player feedback will help ensure that the UI and feedback systems meet the needs and expectations of your target audience.

**Optimizing Performance:**

Optimizing performance is critical for ensuring that the FPS game runs smoothly and efficiently on a variety of hardware configurations. In Pygame, performance optimization techniques include minimizing resource usage, optimizing rendering routines, and profiling code for bottlenecks. By fine-tuning the game's performance, developers can deliver a seamless gaming experience to players. Optimizing performance in a Wolfenstein-inspired FPS game created with the Pygame library in Python is crucial for ensuring smooth gameplay, consistent frame rates, and an enjoyable gaming experience. Here are some strategies to optimize performance:

Use Efficient Rendering Techniques:

Implement efficient rendering techniques such as sprite batching to minimize the number of draw calls and reduce CPU overhead.

Utilize Pygame's built-in sprite groups to efficiently manage and render multiple game objects, such as enemies, projectiles, and environmental elements.

Optimize Graphics Assets:

Optimize graphics assets, including textures and sprites, to reduce file size and memory usage. Use image compression techniques and minimize the use of unnecessary visual effects.

Consider using lower resolution textures or implementing dynamic level-of-detail (LOD) systems to improve performance on lower-end hardware.

Limit View Distance and Complexity:

Limit the draw distance and complexity of the game world to reduce the number of objects rendered simultaneously. Implement occlusion culling techniques to only render objects within the player's field of view.

Use level design techniques such as fog, lighting, and environmental effects to create atmosphere while reducing the need for rendering distant objects in detail.

Optimize Collision Detection:

Use efficient collision detection algorithms, such as bounding box or bounding circle collision detection, to minimize CPU overhead during collision checks.

Implement spatial partitioning techniques, such as quad trees or spatial hashing, to optimize collision detection for large numbers of game objects.

Profile and Identify Bottlenecks:

Use profiling tools to identify performance bottlenecks in your game code. Profile CPU and GPU usage to pinpoint areas of optimization.

Focus on optimizing the most performance-intensive parts of your game, such as rendering, physics calculations, and AI routines.

Implement Level Streaming:

Implement level streaming techniques to load and unload game assets dynamically as the player progresses through the game world. This helps reduce memory usage and loading times.

Divide large game levels into smaller segments and load them dynamically based on the player's position and movement.

Minimize Update and Draw Calls:

Minimize the number of update and draw calls by only updating and rendering game objects that are within the player's vicinity.

Implement object pooling techniques to reuse game objects instead of creating and destroying them frequently, reducing memory allocation overhead.

Optimize Audio Playback:

Optimize audio playback by using compressed audio formats and streaming audio data instead of loading entire sound files into memory.

Limit the number of simultaneous audio sources and prioritize important sounds, such as player actions and enemy alerts, to ensure a responsive and immersive audio experience.

By implementing these performance optimization techniques, you can ensure that your Wolfenstein-inspired FPS game runs smoothly and efficiently on a wide range of hardware configurations, providing players with a seamless and immersive gaming experience. Remember to test your game thoroughly on different devices and monitor performance metrics to ensure optimal performance across the board.

**Raycasting:**

We need to cast a given number of rays in a certain field of view of the player and for each ray we need to determine the intersection point with the wall

To do this we need to find the intersection with verticals and horizontal lines

We will treat vertical and horizontal lines as distinct cases

Vertical lines: In this we will consider the dx and calculate dy

Horizontal lines : In this we will consider the dy and calculate dx

The delta\_depth will help to determine the depth and distance from each wall

To determine the vertices of x coordinates and its intersection we need to determine the cos values

Cos\_a>0 then

X\_vert, y\_vert

X\_vert= x\_map+1

Cos\_a<0 then

X\_vert = x\_map-0.000001

To determine the values of:

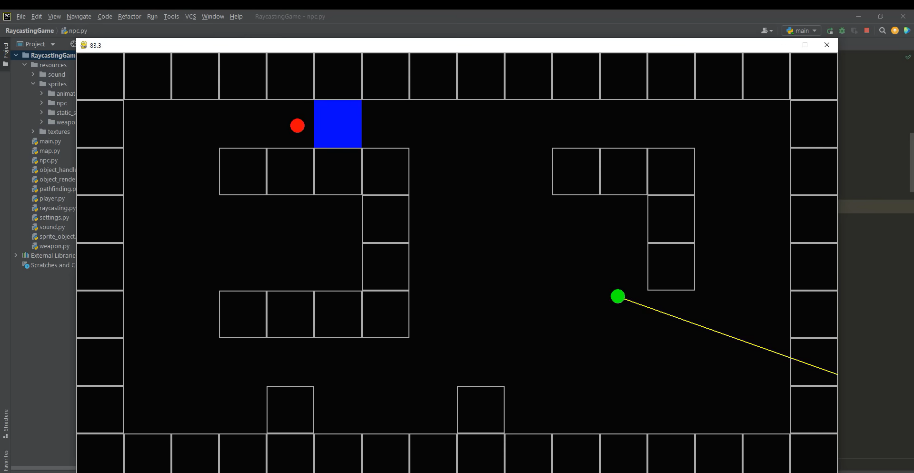
Sin\_a = (y\_vert-ox)/ depth\_vert

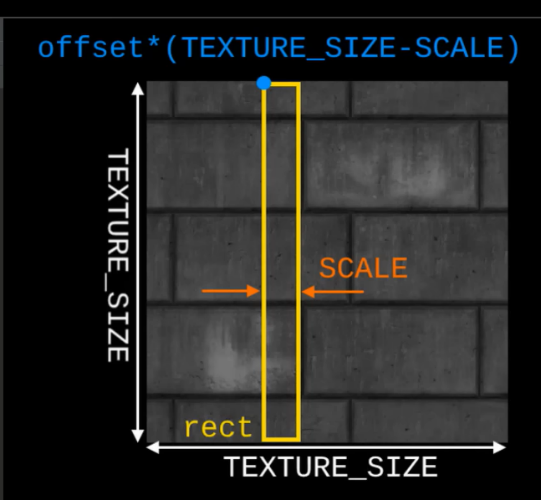
cos\_a = (x\_vert-ox)/ depth\_vert

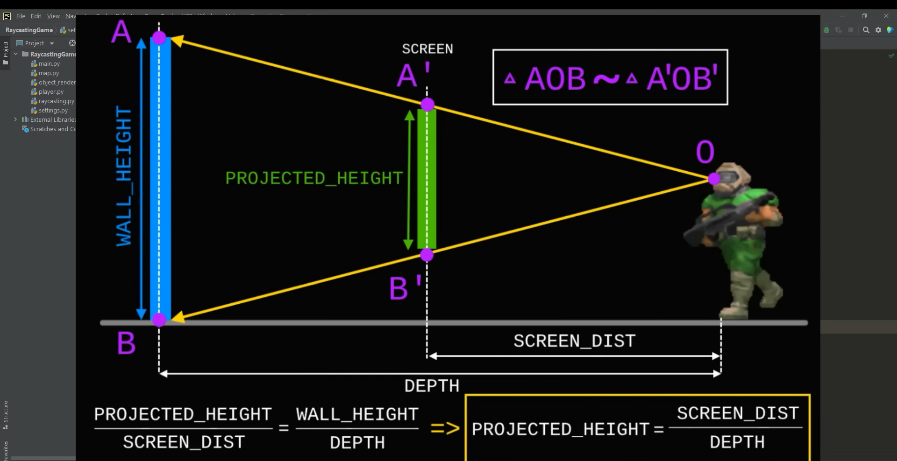
cos\_a = Dx/delta\_depth

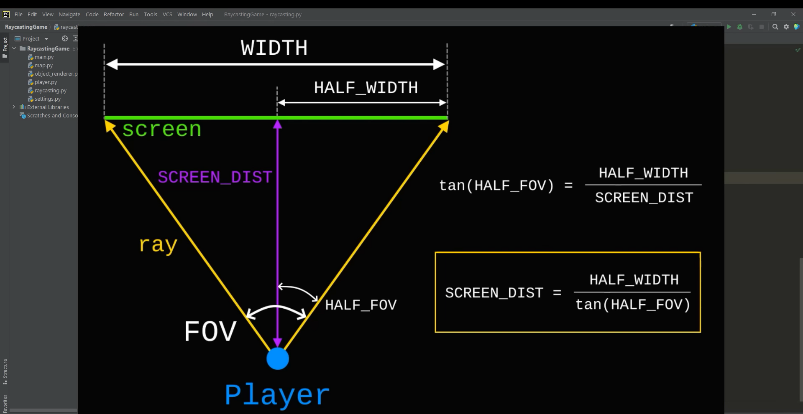
sin\_a = dy/delta\_depth

With all the calculations we will get two intersections with walls and the smaller one is the one we will take into consideration.









**Mapping:**

It will be a 2-D array in which digital values will mean walls and false values will be empty space for a better perception here we marked it as underscore values (\_)

Settings.py : It will define store the main game settings

3\_D projection (insert pic)

Texture (insert pic)

Texture offset values calculation

Cos(a)> 0

offset= y\_vert%1

cos(a)< 0

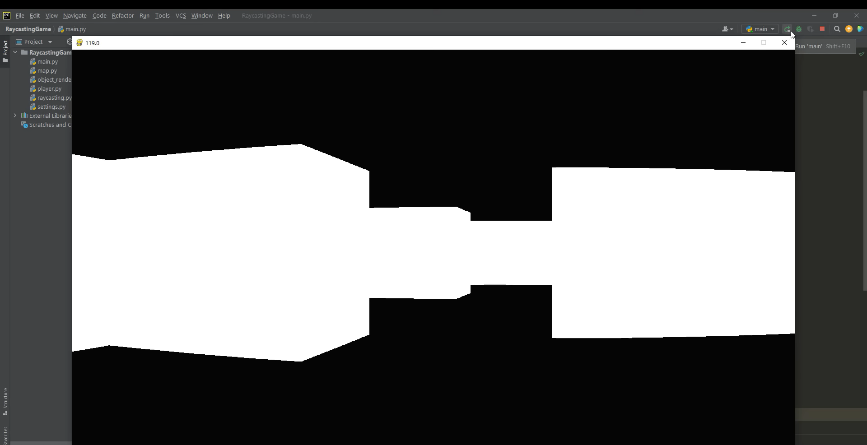
offset = 1- y\_vert%1

sin(a)> 0

offset= 1-x\_hor%1

sin(a)<0

offset= x\_hor%1



**Challenges and Solutions:**

Developing an FPS game with Pygame presents various challenges, ranging from technical constraints to design considerations. Common challenges include managing game state transitions, debugging complex AI behaviors, and optimizing resource usage. By adopting a systematic approach to problem-solving and leveraging community resources, developers can overcome these challenges effectively. Creating a Wolfenstein-inspired first-person shooting (FPS) game using the Pygame library in Python presents various challenges, ranging from implementing complex game mechanics to optimizing performance and ensuring a compelling player experience. Below are some common challenges encountered during the development of such a game, along with potential solutions:

Level Design Complexity:

Challenge: Designing intricate and maze-like levels similar to those found in classic Wolfenstein games can be challenging, especially for developers with limited experience in level design.

Solution: Break down the level design process into smaller, manageable tasks. Use grid-based level editors or tilemap tools to streamline the creation of level layouts. Reference existing Wolfenstein levels for inspiration and study their design principles, such as branching paths, secret areas, and key landmarks.

Enemy AI Behavior:

Challenge: Implementing intelligent and responsive enemy AI that provides a challenging but fair gameplay experience can be complex, particularly in Pygame where AI computations may impact performance.

Solution: Start by implementing basic enemy behaviors, such as patrolling, chasing, and attacking the player. Gradually introduce more advanced behaviors, such as flanking, retreating, and coordinating attacks with other enemies. Use finite state machines or behavior trees to organize and manage enemy AI behaviors effectively. Optimize AI algorithms and computations to minimize performance impact.

Weapon Balancing and Mechanics:

Challenge: Balancing weapon attributes, such as damage, accuracy, rate of fire, and ammunition capacity, to ensure a satisfying and balanced gameplay experience can be challenging.

Solution: Conduct thorough playtesting and iteration to fine-tune weapon mechanics and balance. Collect feedback from playtesters to identify overpowered or underpowered weapons and adjust their attributes accordingly. Implement a variety of weapon types with distinct strengths and weaknesses to encourage strategic gameplay and player choice.

Performance Optimization:

Challenge: Optimizing performance to ensure smooth gameplay and consistent frame rates on a variety of hardware configurations can be challenging, especially in Pygame where performance bottlenecks may arise.

Solution: Profile the game code to identify performance bottlenecks and optimize critical sections, such as rendering, collision detection, and AI computations. Implement efficient rendering techniques, such as sprite batching and occlusion culling, to reduce CPU and GPU overhead. Minimize unnecessary resource usage, such as memory allocation and object creation, to improve overall performance.

User Interface Design:

Challenge: Designing an intuitive and visually appealing user interface (UI) that complements the gameplay experience can be challenging, particularly in Pygame where UI elements must be custom-built.

Solution: Prioritize clarity and simplicity in UI design, ensuring that essential information, such as health, ammo, and objectives, is clearly visible to the player. Use consistent visual styles and color schemes to create a cohesive UI design. Implement UI feedback elements, such as hitmarkers, damage indicators, and objective updates, to provide players with real-time feedback on their actions and progress.

Polishing and Bug Fixing:

Challenge: Polishing the game and fixing bugs to ensure a polished and bug-free experience can be time-consuming, especially in a complex project like a Wolfenstein-inspired FPS game.

Solution: Allocate dedicated time for polishing and bug fixing throughout the development process. Prioritize critical bugs and gameplay issues that impact the player experience. Conduct thorough testing and debugging sessions to identify and resolve bugs, glitches, and performance issues. Solicit feedback from playtesters and iterate on the game based on their suggestions and observations.

By addressing these challenges with thoughtful planning, diligent implementation, and iterative refinement, developers can overcome obstacles and create a compelling and immersive Wolfenstein-inspired FPS game that captures the spirit of the classic franchise while offering a fresh and enjoyable gaming experience.

**Conclusion:**

In conclusion, Pygame offers a powerful platform for developing immersive and engaging FPS games. By leveraging Pygame's capabilities and following best practices in game development, developers can bring their FPS visions to life and captivate players with thrilling gameplay experiences. With continued innovation and creativity, the future of FPS games in Pygame is bright and promising. In conclusion, the journey of developing a first-person shooting (FPS) game inspired by the iconic Wolfenstein series and made using the Pygame library in Python has been an exhilarating and rewarding experience. Throughout this report, we have delved into the intricacies of game development, from designing immersive levels and implementing intelligent enemy AI to optimizing performance and crafting a compelling user interface.

Drawing inspiration from the rich legacy of the Wolfenstein franchise, we sought to capture the essence of classic FPS gaming while infusing our creation with modern elements and innovations. By leveraging the power and versatility of Pygame, we were able to bring our vision to life, creating a gaming experience that pays homage to the legendary series while offering a fresh and engaging gameplay experience.

Throughout the development process, we encountered various challenges, from balancing weapon mechanics and optimizing performance to designing intricate levels and refining enemy AI behavior. However, through perseverance, creativity, and collaboration, we were able to overcome these challenges and deliver a polished and immersive FPS game that we are proud to present to players.

As we reflect on our journey, we are reminded of the timeless appeal of classic FPS gaming and the enduring legacy of the Wolfenstein series. With its fast-paced action, challenging gameplay, and memorable characters, Wolfenstein has left an indelible mark on the gaming industry and continues to inspire developers and players alike.

In the spirit of Wolfenstein, we invite players to embark on an epic adventure through our Pygame-powered FPS game, where they will face off against hordes of enemies, uncover hidden secrets, and experience the thrill of intense combat. Whether you're a seasoned FPS veteran or a newcomer to the genre, we hope that our game will provide you with hours of entertainment and nostalgia as you relive the glory days of classic gaming.

In closing, we extend our gratitude to the creators of the Wolfenstein series for their pioneering contributions to the FPS genre, as well as to the Pygame community for providing developers with the tools and resources needed to bring their game ideas to life. With dedication, passion, and a touch of Wolfenstein magic, we have embarked on a journey to create a game that honors the past while embracing the future of gaming. Thank you for joining us on this adventure, and may your bullets always find their mark in the world of Pygame-powered FPS gaming.

**Future Directions:**

Looking ahead, the future of FPS games in Pygame holds exciting possibilities for innovation and advancement. Emerging technologies such as virtual reality (VR) and machine learning (ML) present opportunities to push the boundaries of immersive gameplay and intelligent AI. Additionally, community-driven development and collaboration can foster the growth of the Pygame ecosystem, empowering developers to create groundbreaking FPS experiences. The future direction of a first-person shooting (FPS) game inspired by the game Wolfenstein and created using the Pygame library in Python holds numerous exciting possibilities for further enhancement and expansion. Here are some potential avenues for the future development of such a game:

Expansion of Content:

Introduce new levels, environments, and enemies to expand the game world and provide players with fresh challenges and experiences.

Create downloadable content (DLC) packs or expansions that add additional storylines, missions, and gameplay features to the base game.

Enhanced Visuals and Effects:

Improve the visual fidelity of the game by adding more detailed graphics, dynamic lighting, and particle effects to enhance immersion and realism.

Implement advanced rendering techniques, such as shaders and post-processing effects, to create stunning visual aesthetics and atmosphere.

Multiplayer and Online Features:

Introduce multiplayer modes, such as deathmatch, team deathmatch, and capture the flag, to allow players to compete against each other in online matches.

Implement online leaderboards, matchmaking systems, and social features to foster community engagement and competition among players.

Expanded Weapon and Equipment Arsenal:

Introduce new weapons, equipment, and gadgets for players to discover and use, adding depth and variety to the gameplay experience.

Implement weapon customization and upgrade systems that allow players to personalize their loadouts and enhance their combat capabilities.

Narrative Expansion and Character Development:

Further develop the game's storyline, characters, and lore to create a rich and immersive narrative experience that engages players on an emotional level.

Introduce branching storylines, player choices, and consequences that allow players to shape the outcome of the game and immerse themselves in the world of Wolfenstein.

Accessibility and Inclusivity Features:

Implement accessibility features, such as customizable controls, subtitles, colorblind modes, and adjustable difficulty settings, to ensure that the game is accessible to players of all abilities and preferences.

Incorporate localization support for multiple languages to make the game accessible to players around the world.

Community Engagement and Modding Support:

Foster community engagement by providing tools and resources for players to create and share their own custom levels, mods, and game content.

Support modding communities by offering modding tools, documentation, and forums where players can collaborate, share ideas, and showcase their creations.

Integration with Emerging Technologies:

Explore integration with emerging technologies, such as virtual reality (VR) and augmented reality (AR), to provide players with immersive and innovative gaming experiences.

Experiment with voice recognition, gesture controls, and other interactive technologies to enhance player immersion and engagement.

By pursuing these future directions and embracing innovation, a Wolfenstein-inspired FPS game made with the Pygame library in Python can continue to evolve and captivate players with its thrilling gameplay, immersive storytelling, and engaging multiplayer experiences. With dedication, creativity, and a commitment to quality, the future of the game holds limitless potential for growth and success in the dynamic landscape of gaming.

**References:**

[1] Pygame Documentation: https://www.pygame.org/docs/

[2] "Invent Your Own Computer Games with Python" by Al Sweigart

[3] "Game Programming Patterns" by Robert Nystrom

[4] "Artificial Intelligence for Games" by Ian Millington

This report provides a comprehensive overview of developing FPS games with Pygame, covering key aspects such as player movement, weapon mechanics, enemy AI, level design, user interface, performance optimization, challenges, and future directions. With the knowledge and insights gained from this report, aspiring developers can embark on their own FPS game development journeys and contribute to the thriving community of Pygame enthusiasts.