CPT306 Individual Project
Coursework Submission Form

2022/23 Semester 2
Bachelor Degree – Year 4

Module Code	Module Leader	Module Title
CPT306	**Hai-Ning Liang** **Nan Xiang**	**Principles of Computer Games Design**

Section A: Your Details

To be completed by the student (in English using BLOCK CAPITALS)

Student's Name	DONGHENG LIN
Student ID	1929066

Section B: Assignment Details

To be completed by the student (in English using BLOCK CAPITALS)

Coursework Assignment Number	Assignment 4
Coursework Title	Creating a 3D Game
Method of Working	**Individual**
Date and Time of Submission	2023/5/18

Assignment details can be found in the assignment description.

Section C: Statement of Academic Honesty

To be completed by the student

By submitting this coursework for assessment, you are confirming that you have read and understood the University's policy on plagiarism and collusion and that the submitted work is your own.

(i) I confirm that I have read a copy of the current University's definitions of collusion and plagiarism on coursework and academic honesty, and that I fully understand the meaning of these terms.

(ii) I confirm that the submitted coursework has been created solely by me and that I have not been assisted, nor have copied part or all of somebody else's work, either with their explicit approval or without their knowledge or consent.

(iii) I confirm that this is my own work and that use of material from other sources, including the Internet, has been properly and fully acknowledged and referenced.

(iv) I confirm that the information I have given is correct to the best of my knowledge.

If this form is submitted electronically, please type your name in English (BLOCK CAPTIALS)

Student's signature	DONGHENG LIN	Date	2023/5/18

Design Report

Warzone: Scavenger

CPT306

Principles of Computer Games Design

Assignment 4 Individual Coursework

Dongheng Lin

Student ID: 1929066

Content

1. Overview 4
2. Story Synopsis 6
3. Gameplay Design 7
3.1 Game Genre 7
3.2 Gameplay Mechanics and Gameplay 7
3.3 Game Elements 7
3.4 Game Progression 9
3.5 User Interfaces 11
4. Implementation 13
4.1 General Structure 13
4.2 Class Implementation 14
5. Ethical and Social Issues 19
6. Limitations and Future Work 20

1. Overview

Warzone: Scavenger is a groundbreaking 3D survival game that immerses players in the midst of a war-ravaged kingdom, where resource collection and strategic thinking are as vital as quick reflexes. The game's unique concept hinges on the player operating a drone to navigate perilous warzones and salvage critically needed supplies amidst ongoing bombardment and airstrike.

[i]Fig. 1 Main menu of the game

The game's controls are an integral part of its design, using a combination of traditional WASD keys for movement and the SPACE bar for evasion maneuvers, such as jumping to evade explosions. The game's bombardment phases are conducted in waves, adding a rhythmic but unpredictable danger to the drone's mission. An onboard radar system provides timely alerts of imminent aerial threats, adding a layer of strategic decision-making to the gameplay. The drones are designed sensitive to environments such that signals for incoming explosives can be identified by player in both visual and sound information.

The primary objective is to secure a total of 100 supply units scattered across the landscape, adding an element of exploration and discovery. However, the survival of the drone itself is a continuous concern. If the drone sustains critical damage, the player can choose to expend 10 supply units to repair it, introducing a challenging trade-off between resource collection and survival.

Furthermore, the game offers an innovative mechanic of switching radio channels on the drone, controlled by numerical keys. This feature not only adds authenticity to the drone operating experience but also has potential implications for gameplay, such as receiving crucial information or instructions.

In essence, Warzone: Scavenger is a unique fusion of survival, strategy, and action, layered with a deeply engaging narrative. The game's mechanics demand both quick reflexes and strategic foresight from the player, making it a standout in the realm of 3D survival games.

2. Story Synopsis

Warzone: Scavenger weaves a compelling narrative of resilience, survival, and undying hope, set against the backdrop of a once-thriving kingdom, now reeling under the onslaught of an aggressive invader. The game unfolds in the heart of the kingdom, a picturesque meadow, once the flourishing agricultural hub, now transformed into a war-ravaged battleground. Surrounded by pristine waters and dotted with rolling hills, this landscape of natural beauty stands in stark contrast to the brutality of war.

The kingdom's inhabitants, who once reveled in prosperity and abundance, are now on the brink of survival, their lives disrupted amidst the ruins of their homeland. Yet, hope endures through the airborne supplies sent by empathetic allies and the innovative solutions from their own. The resilient engineers of the kingdom have constructed a beacon of hope - a high-tech drone designed to navigate the war-torn meadows and secure these vital resources.

Assuming the role of the drone operator, players are tasked with a mission that is as dangerous as it is essential. They must maneuver the drone through an onslaught of explosions, dodge enemy detection, and retrieve the precious supplies scattered across the scenic meadow. The drone becomes more than a machine; it symbolizes the kingdom's resilience, a glimmer of hope amidst despair, and their determination to reclaim their homeland.

The antagonist of the game is not an individual but a faceless, relentless invading force. This adversary is ruthless, its constant bombardment turning the serene meadow into a volatile warzone. Players are challenged to outwit and outmaneuver these forces, transforming the chaos of bombardment into opportunities for securing supplies.

The narrative culminates in a high-stakes event when a massive supply drop, critical for the kingdom's survival, is announced. This drop draws intense enemy defenses, turning the serene meadow into a storm of destruction. The player's ultimate test lies in guiding the drone through this tempest, navigating this perilous situation with skill, strategy, and resource management. The kingdom's future is precariously balanced on this final mission.

Warzone: Scavenger is more than a tale of war and survival; it is a testament to human resilience and tenacity. As players navigate the drone through the war-ravaged meadow, they witness the devastating impacts of war and the unyielding determination of a people to survive and rebuild. It is a narrative fraught with danger and despair, punctuated by moments of triumph and the enduring spirit of a kingdom striving to reclaim their once-prosperous land against all odds

3. Gameplay Design

Warzone: Scavenger is an innovative fusion of adventure, strategy, and survival game genres. It is a 3D third-person perspective game that plunges players into the heart of a devastated kingdom under a relentless onslaught. The mission: to operate a drone and navigate it through the hazardous, war-torn landscape to retrieve precious supplies, all while attempting to evade enemy bombardments.

3.1 Game Genre

Warzone: Scavenger amalgamates adventure, strategy, and survival elements to create a unique gaming experience. The explorative excitement of adventure games, the tactical nuances of strategy games, and the tension of survival games are intertwined, resulting in an immersive and dynamic gameplay experience.

3.2 Gameplay Mechanics and Gameplay

The gameplay mechanics in Warzone: Scavenger revolve around the strategic control of the drone, careful navigation of the terrain, efficient collection of supplies, and the tactical evasion of ongoing aerial bombardments.

Players guide the drone using the WASD keys for movement, with the SPACE key facilitating jumps to evade explosions. The perspective is controlled via the mouse, allowing players to strategically survey the terrain and plan their movements. Supplies can be collected by having the drone touch them. In the event of the drone's destruction, players can opt to use 10 supply units for repairs. Additionally, players can switch radio channels on the drone using the number keys for added immersion.
Outcome: The primary objective is to collect 100 supply units, which signifies successful completion of the mission. The game concludes when the drone falls into the surrounding waters or is irreparably destroyed by an explosion.

A game's score is determined by subtracting the number of destroyed ground blocks from a base value. This mechanism encourages efficient supply collection and strategic evasion of destruction, rewarding players for mitigating the impact of the enemy bombardment.

3.3 Game Elements

The enemy AI stages bombardments in waves, each wave generating 15 random bombs within a 64x64 block area. These bombs obliterate ground blocks within a certain radius and pose a lethal

threat to the player's drone. Utilizing immersive 3D graphics, Warzone: Scavenger depicts a strikingly realistic war-torn landscape. The drone, the ensuing destruction from the bombs, and the supply drops are animated to enhance the game's realism and immersion. Additionally, Sound cues are used to alert the player of incoming bombardment waves and imminent bomb drops. The feature to switch radio channels on the drone adds an extra layer of auditory engagement.

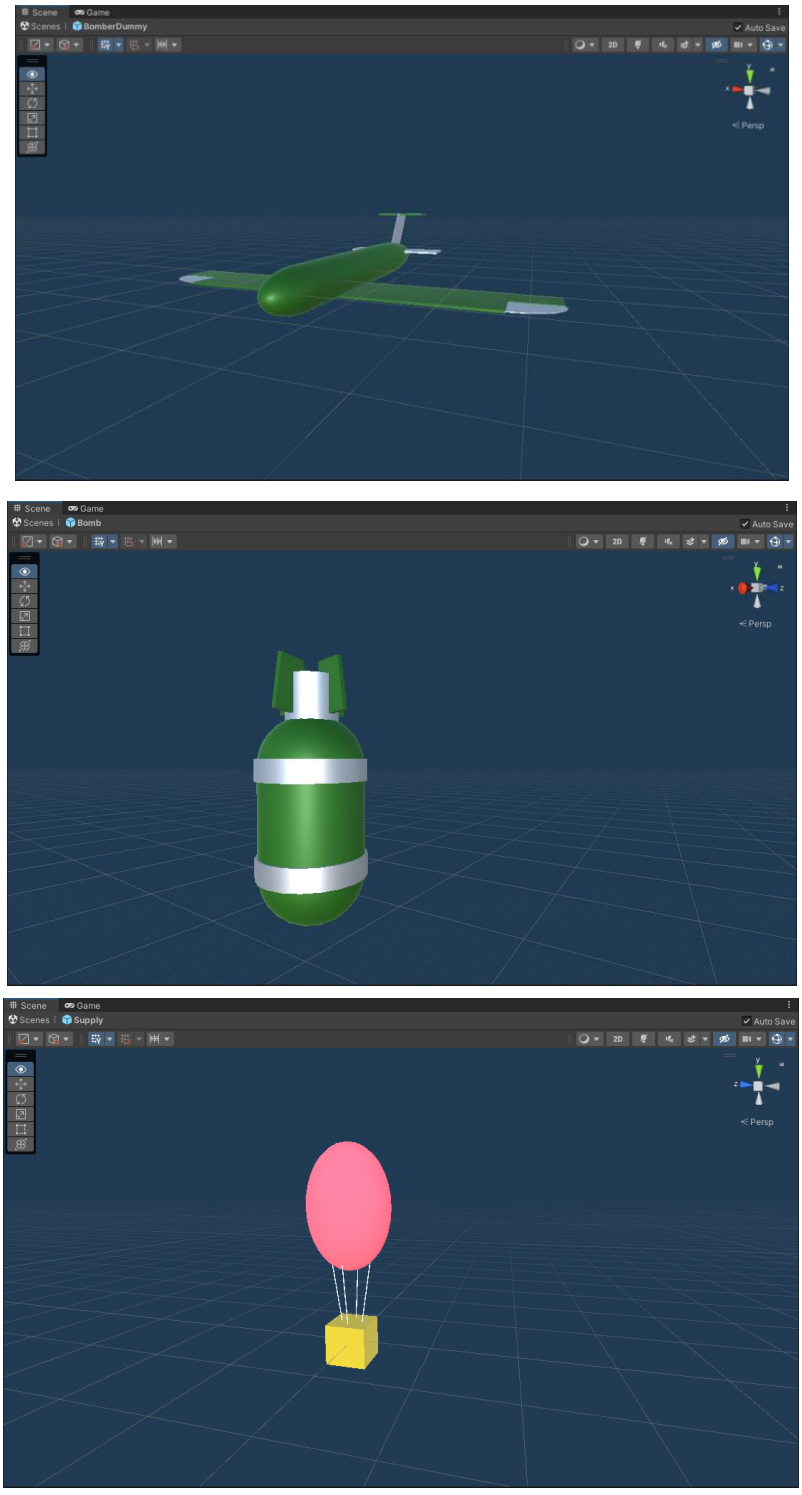

Fig. 2 Prefabs used for Objects

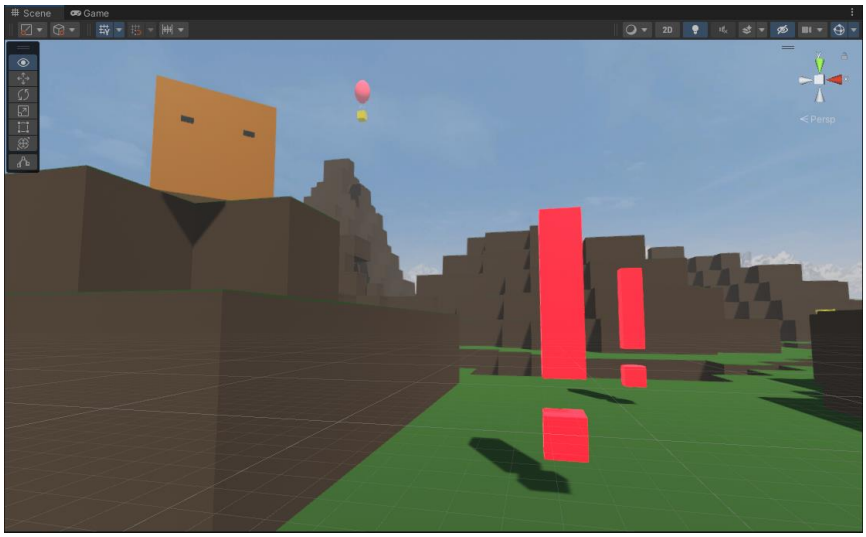

Fig. 3 Dangerous warnings for places under bombs

Fig. 4 Audio Assets Prepared for SFX and Radio

The terrain of the game is randomly generated for each session, contributing to the unpredictability and replayability of the game. Despite the constant of a scenic meadow surrounded by water, the terrain's random generation ensures that each gameplay experience is unique.

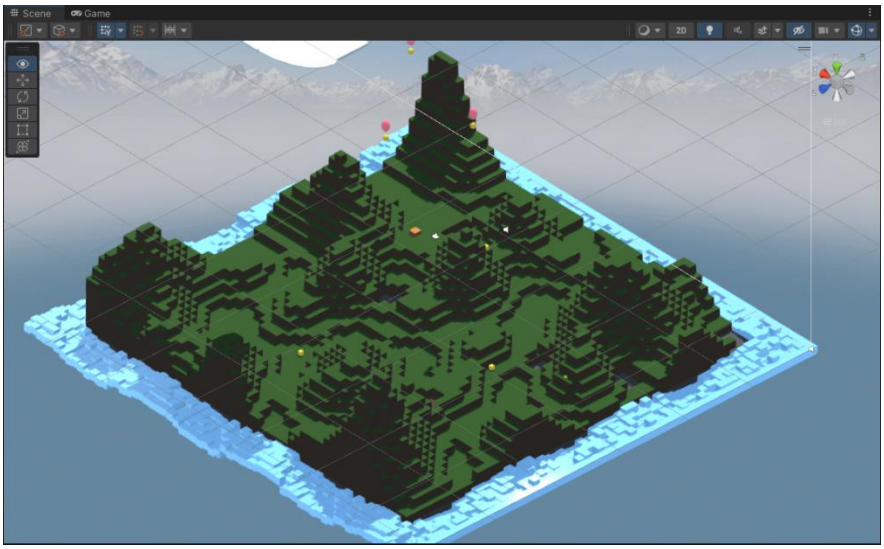

Fig. 5 Random Game Terrain

3.4 Game Progression

The game's progression is marked by increasingly challenging waves of bombardments, each wave posing a heightened threat with more destroyed landscape and greater destruction. Concurrently, the

number of supplies dropped also increases, providing the player with amplified opportunities to achieve the victory condition.

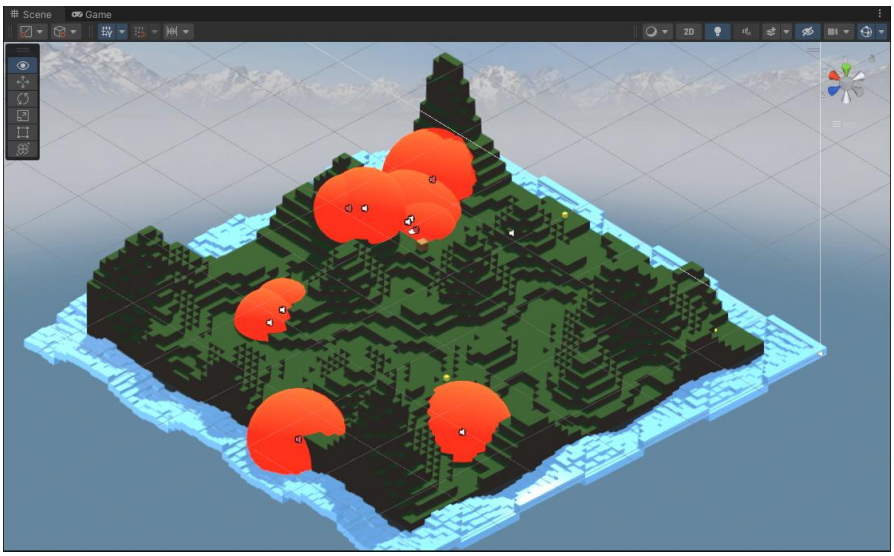

Fig. 6 Explosion Effect

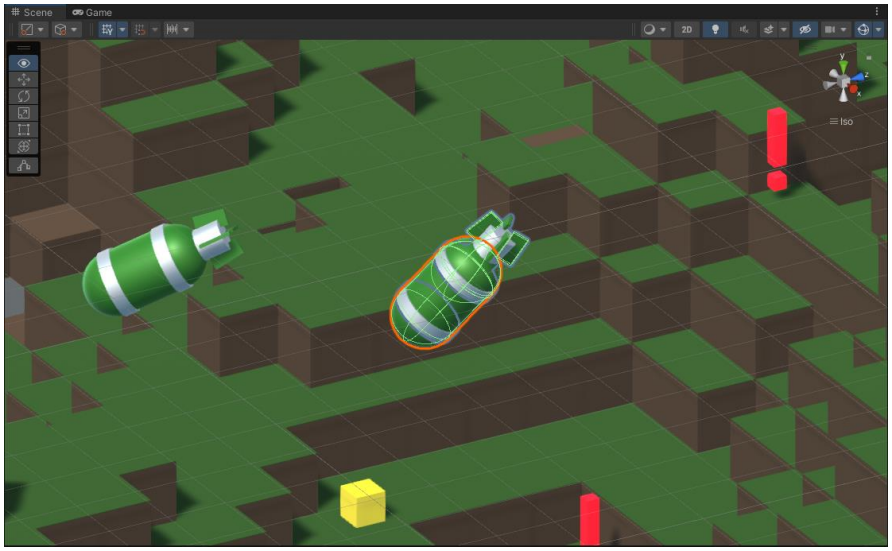

Fig. 7 Bombs

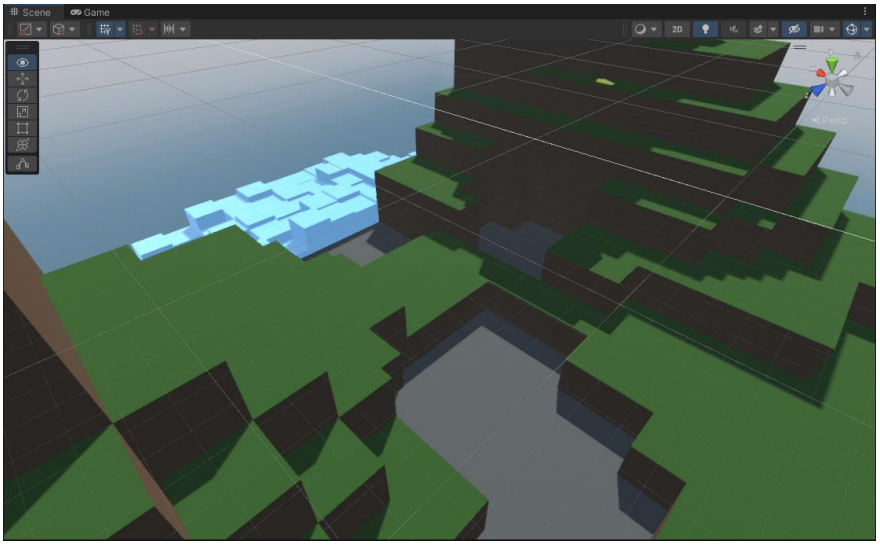

Fig. 8 Craters after Explosion

3.5 User Interfaces

The user interface is meticulously designed to provide vital information to the player efficiently. It comprises a radar system that alerts players of incoming bombs, and a supply count that keeps track of the collected supply units. It also displays the drone's health status, the current radio channel, and importantly, the time elapsed since the game's start. Each interaction with the user interface is accompanied by a smooth visual or sound transition, contributing to a seamless and immersive gaming experience.

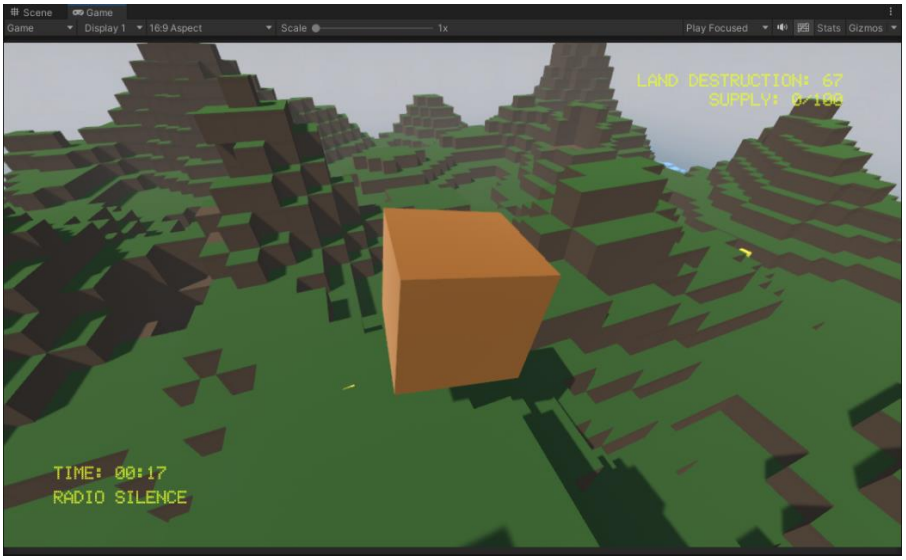

Fig. 9 In game graphics

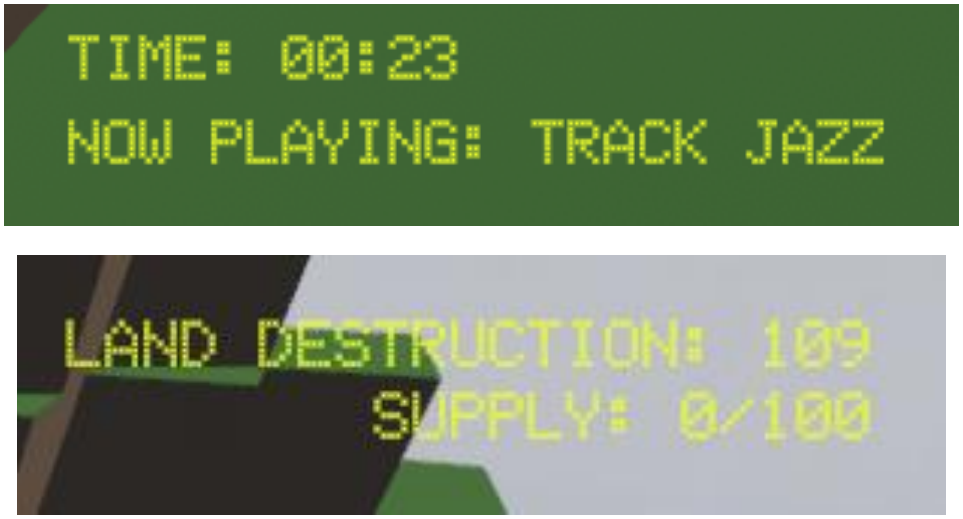

Fig. 10 HUD

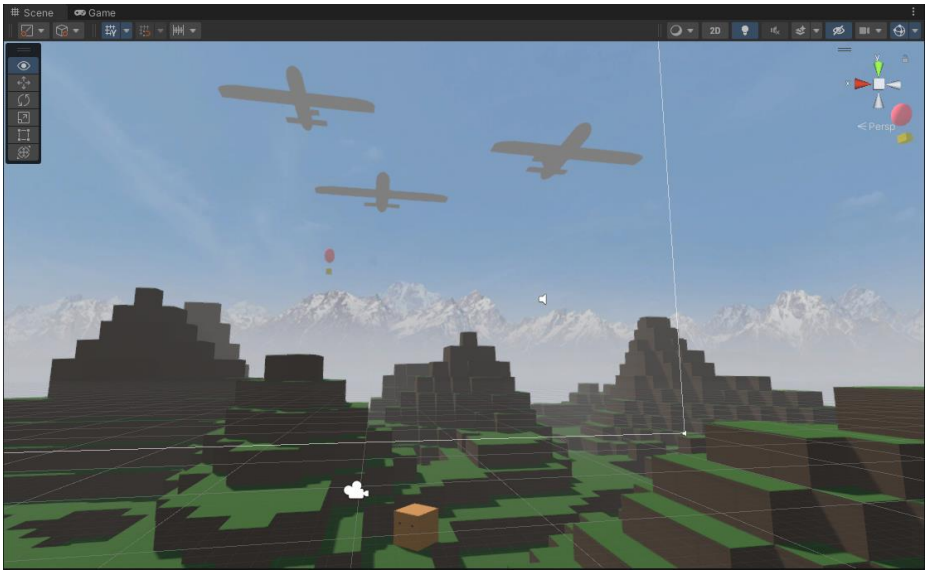

Fig. 11 Ongoing Bombardment is noticed to user by UI

4. Implementation

4.1 General Structure

Implementing Warzone: Scavenger required a strategic blend of various technical skills, meticulous planning, and efficient execution. The development process involved numerous stages, each contributing towards crafting a game that provides a seamless and immersive gaming experience.

The initial step involved mapping out the game's structure through a Unified Modeling Language (UML) diagram. UML diagrams are pivotal in visualizing the game's structure, detailing the relationship between different components, and understanding the flow of control and data through the system. It served as a blueprint for the subsequent development phases, ensuring that every component was carefully designed and integrated.

The UML diagram comprised several core components, each with a distinct role in the game's functionality. There are 2 scenes as shown in Fig. 12 and the scenes are composed of high-level key-components including, MenuManager, GameManager, MainCamera, InGameUI, Player, Supply and Bomb. In the following section, the class-level description will be given to illustrate the implementation of the component level functionalities.

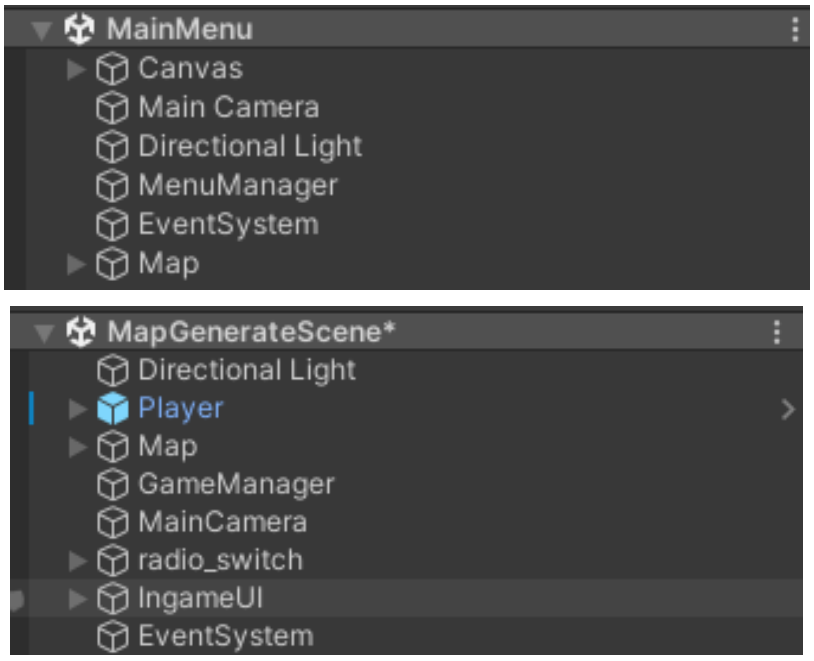

Fig. 12 Scenes and Game Objects

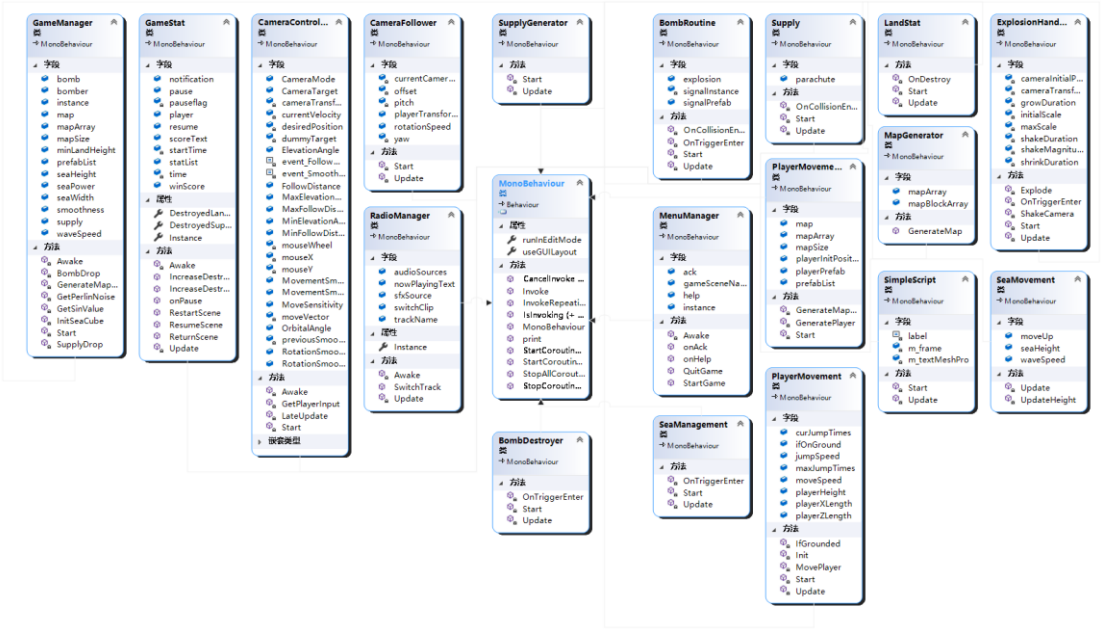

Fig. 13 Class UML Diagram

4.2 Class Implementation

The **GameManager** class is the main controller of the game, managing various aspects such as the map generation, sea cube initialization, and the dropping of supplies and bombs.

Class Variables hold references to various GameObjects and prefabs including the **map, supply, bomb, and bomber.** Including variables defining the **map size and properties**, such as the sea width, sea height, sea power, wave speed, minimum land height, and smoothness. A 3D integer array mapArray that represents the generated map.

The **GameManager** class serves as the main orchestrator of the game, setting up the environment, and managing the essential game mechanics like supply drops and bomb drops. It leverages Perlin noise to create a natural-looking terrain and uses coroutines to manage repeated and time-based actions. Therefore, there are also Methods within supports many major functionality of the game, which are:

1. **GetPerlinNoise(int x, int z):** Returns the Perlin noise value for a given x, z coordinate, ensuring the returned height is within a certain range.
2. **GenerateMapArray():** Initializes the mapArray and generates the map's terrain using Perlin noise for a more natural appearance.
3. **GetSinValue(float x, float z):** Returns a value used for simulating the sea's wave movement.
4. **InitSeaCube():** Initializes the sea cubes' starting positions and sets their movement direction.
5. **SupplyDrop(GameObject supply):** A coroutine that spawns supply packages at random positions in the game world every 5 seconds.

6. **BombDrop(GameObject bomb, GameObject Bomber, float v, float h):** A coroutine that spawns bomber and bomb objects, and moves the bomber towards a target position before spawning the bombs.
7. **Awake():** Enforces the singleton pattern for GameManager, generates the map array, generates the map itself using MapGenerator, and initializes the sea cubes.
8. **Start():** Starts the SupplyDrop and BombDrop coroutines multiple times, creating a series of supply drops and bomb drops in the game world.

The **GameStat** class manages the game's statistics, such as the count of destroyed lands and destroyed supplies, and handles the game's pause functionality. **DestroyedLandCount** and **DestroyedSuppliesCount** track the number of destroyed land blocks and collected supplies, respectively.

statList is an array of **TextMeshProUGUI** objects used to display game statistics on the screen.

The **GameStat** class serves as a central point for managing and displaying game statistics and handling the game's pause functionality. It also provides methods for restarting and resuming the game. Specifically, The **GameStat** class supports game statistics and state transition by implementing following methods:

1. **Awake():** Enforces the singleton pattern, initializes the pause screen and the supplies count text, and sets the start time.
2. **IncreaseDestroyedLandCount():** Increases the count of destroyed land blocks and updates the corresponding text display.
3. **IncreaseDestroyedSuppliesCount():** Increases the count of destroyed supplies and updates the corresponding text display. Also checks if the player has reached the win score and pauses the game if so.
4. **Update():** Computes and displays the elapsed game time, and listens for the Escape key to pause the game.
5. **onPause(int pauseCode):** Handles the game pause. Depending on the pauseCode, it toggles the pause state, displays the win notification and score, or displays the loss notification and the resume button.
6. **RestartScene():** Restarts the current scene, effectively restarting the game.
7. **ReturnScene():** Returns to the main menu scene.
8. **ResumeScene():** Resumes the game after a loss if the player has enough supplies, reduces the supplies count, and moves the player to a safe position.

As for the **explosion** component, the illustration may involve several cooperating classes.

LandStat class is used for tracking the status of the land objects in the game. Currently, the class is relatively bare with only a **Start()** and **Update**() method that don't perform any actions, and an

OnDestroy() method. The **OnDestroy()** method is a Unity-specific method that gets called when the **GameObject** this script is attached to is destroyed. Here, when a land object is destroyed, it calls the **IncreaseDestroyedLandCount**() method of the **GameStat** singleton. This class is important for keeping track of how many land objects have been destroyed during gameplay, which might be a crucial gameplay metric.

The **BombRoutine** class is attached to the bomb objects in the game. It handles the behavior of the bombs, specifically their interaction with the ground and their destruction. The **Start**() method initializes the **signalInstance** variable as null. The **Update**() method is called once per frame and checks if the bomb's position has dropped below a certain threshold, in which case it destroys the bomb and the signal instance if it exists. It also casts a ray downwards from the bomb, and if it hits an object tagged as "ground", it either creates a new **signal instance** at the hit point or moves the existing signal instance to the hit point.

The **player control implementation** is adjusted a little bit compared with its original design to better integrate with the game. In the original script, the player's movement was based on the world's coordinate system. The script uses the WASD keys to directly move the player along the world's axes, and when a key is pressed, it sets the player's rotation to a fixed direction relative to the world's coordinate system. This means that the player's controls are **fixed relative to the world.** If the player is facing east (rotation (0, 90, 0)) and the player presses 'W', they will move north, not in the direction they're facing.

The updated script changes the player's movement to be relative to the camera's orientation, not the world's coordinate system. This is a common setup in third-person games, where the player's forward direction is usually the same as the camera's forward direction. The script uses the WASD keys to move the player relative to the camera's forward and right vectors such that:

1. 'W' moves the player in the direction the camera is facing.
2. 'S' moves the player opposite to the direction the camera is facing.
3. 'A' moves the player to the left of the camera.
4. 'D' moves the player to the right of the camera.

The script also makes the player **always face the direction of movement** by updating the player's rotation to look in the direction of movement. This means that the player's controls will be intuitive from the player's perspective, not fixed relative to the world. If the player is facing east and they press 'W', they will move east, not north.

In the **OnCollisionEnter**() and **OnTriggerEnter**() methods, the bomb is destroyed if it collides with the ground or enters an explosion, respectively. It also instantiates an explosion **GameObject** at the

bomb's position.

The **BombDestroyer** class is designed to handle the destruction of ground objects in the game. The **OnTriggerEnter**() method within is automatically called by Unity when the explosion collider (set as a trigger) enters another collider. If the other collider is tagged as "ground", it destroys the ground object. This script is likely attached to an explosion **GameObject**, and when the explosion happens, it destroys any ground in its radius.

ExplosionHandler Class is attached to explosion **GameObjects** and controls the growth and shrinkage of the explosion, simulating an explosion effect. It also shakes the camera for a specified duration and magnitude during the explosion. In the **Start**() method, it initializes the **initialScale** of the explosion and the **cameraTransform** and **cameraInitialPosition** for camera shaking. And in the **Explode**() coroutine, it gradually increases the size of the explosion to a maximum scale over a **growDuration**, waits for a brief period, then gradually shrinks the explosion back to its initial scale over a **shrinkDuration**. During the growth phase of the explosion, it starts the **ShakeCamera**() coroutine, which randomly moves the camera within a specified magnitude for a specified duration, creating a camera shake effect. The **OnTriggerEnter**() method checks if the explosion collides with a GameObject tagged as "Player", and if so, it calls the **onPause**() function in the **GameStat** singleton with parameter 2, indicating a loss condition for the player.

The **Supply** class is an essential component of the game that handles the behavior of the supply packages that are dropped into the game world. The **Start()** method is called at the beginning of the script's lifecycle. It appears to be commented out but was initially designed to get the **parachute GameObject** from the supply package's children. Update() function is called every frame. It checks whether the supply package has fallen below the 'ground level' ($y < 0$). If it has, the supply package is destroyed, ensuring that dropped packages do not fall indefinitely.

OnCollisionEnter(Collision other) function is called when this GameObject starts colliding with another GameObject. It checks for various conditions: If the supply package collides with a GameObject tagged as "ground", "bottom", or "Supply", it finds the parachute child GameObject and destroys it. This simulates the parachute being discarded once the package has landed.

If the supply package collides with a GameObject tagged as "Player", it calls a method **IncreaseDestroyedSuppliesCount()** from the **GameStat singleton instance**, which presumably increases the player's collected supply count. After this, it destroys the supply package, indicating that the player has collected it. In essence, the Supply class is responsible for the lifecycle and interactions of the supply packages in the game, from their descent to their collection by the player.

The **CameraFollower** class is a part of the game that handles the camera's behavior, specifically making it follow the player character smoothly and react to mouse movements. The class is designed to make the camera's movements appear natural and intuitive, enhancing the player's immersion in

the game.

The game's development was an iterative process, with each version tested rigorously to ensure optimal performance. The implementation process not only focused on the game's functionality but also paid significant attention to the user experience, ensuring the game was both enjoyable and engaging. This class ensures that the player always has the best possible view of the game environment and their character, despite any obstructions or movement, enhancing the gameplay experience.

RadioManager class manages audio in the game, particularly music tracks. It appears to simulate a radio, with the ability to switch between different tracks. It has a singleton pattern implemented, similar to **GameStat**, which allows other classes to easily access its public methods and properties. The **audioSources** array stores different audio tracks that the player can switch between. The **sfxSource** is used to play sound effects, in this case, the sound of switching tracks.

Specifically, in the **Update**() method, it listens for number key inputs (1, 2, 3, etc.). If a number key is pressed, it calls **SwitchTrack(i)**, where i is the number key pressed minus 1 (since array indices start at 0). This allows the player to switch between tracks by pressing the corresponding number key. In the **SwitchTrack(int trackIndex)** method, it first checks if the provided **trackIndex** is valid (i.e., within the bounds of the **audioSources** array). If it is, it plays the switch sound effect, then iterates over all audio sources. If the index of the current audio source matches **trackIndex**, it checks if that audio source is already playing. If not, it starts playing that audio source and updates the **nowPlayingText** UI element to display the name of the now-playing track. If the index doesn't match **trackIndex**, it checks if that audio source is playing, and if so, it pauses it. This ensures that only the selected track is playing.

The **SeaManagement** class is relatively simple. As its name implies, it is likely responsible for managing the sea or water-related interactions in the game. The **OnTriggerEnter(Collider other)** method is the key part of this class. It is called when another collider enters the trigger zone of the object this script is attached to (likely the sea or some water body in the game). If the entering object has the tag "Player", it calls the **onPause(2)** method from the **GameStat instance**. This could indicate that the player's character has fallen into the sea or a similar event, which should pause the game and possibly display a loss message.

In the end, the implementation of Warzone: Scavenger was a complex orchestration of various game development elements. It required a deep understanding of the game's structure, a careful calibration of its components, and an unwavering commitment to providing an immersive and engaging gaming experience. Despite the challenges, the process was a rewarding journey of turning a creative vision into a captivating reality.

5. Ethical and Social Issues

Warzone: Scavenger, despite being a survival strategy game, deals with themes of war, destruction, survival, and hope, which are profound and can lead to several ethical and social considerations.

The game revolves around navigating a war-ravaged kingdom, and thus contains depictions of violence and war. This might not be suitable for younger audiences or people who are sensitive to such themes. Developers should consider appropriate content warnings and an age rating system to ensure the game is played by a suitable audience.

Although the antagonist is a faceless invading force, care should be taken to avoid potential stereotyping or negative representation of real-world cultures, nationalities, or conflicts. The game should avoid promoting harmful narratives or biases.

The game's intense themes and high-stakes gameplay could potentially lead to stress or anxiety in some players. It's essential to balance challenging gameplay with a positive and encouraging game environment. Players should be reminded that it's a game, and failure is a part of the learning process.

The game has a chance to promote messages of peace, resilience, and the costs of war. It can emphasize the importance of diplomacy, cooperation, and conflict resolution, potentially opening discussions about the impacts of war on societies and individuals.

As with any game that may collect user data (for leaderboards, game saves, etc.), it's crucial to ensure players' data is stored securely and ethically, adhering to all relevant data protection laws and regulations.

In terms of demographics, the game seems best suited for mature teenagers and adults due to its strategic elements and themes of war. Geographically, it should appeal broadly as long as care is taken to avoid any culturally insensitive elements. Psychographically, it would appeal to players who enjoy strategy, action, survival games, and those who appreciate a rich narrative backdrop to their gaming experience.

6. Limitations and Future Work

While the novel concept and captivating gameplay of Warzone: Scavenger are undeniably engaging, a critical examination reveals certain limitations inherent in the current iteration of the game. Furthermore, this scrutiny yields various opportunities for future growth and development.

Currently, the drone's capacity for interacting with the game environment is primarily restricted to the collection of supplies. Despite the war-torn landscape being replete with detailed narratives, it arguably provides a limited scope for dynamic gameplay. A potential remedy for this could involve the introduction of destructible or interactive environmental elements that could be strategically incorporated into gameplay.

In addition, the wave-based bombardment and supply collection mechanisms do provide an appreciable degree of challenge. However, there is a risk of these repetitive tasks diminishing the player's engagement over time. To mitigate this, the introduction of new challenges or mechanics in subsequent levels, such as hostile drones, diversified types of supplies, or weather conditions affecting drone operation, could be considered.

What’s more, the current version of the game offers a solitary player experience. The incorporation of a multiplayer mode could potentially heighten its replayability and broaden its appeal. This could allow players to collaborate in supply collection, compete against one another, or even assume the role of the invading forces.

By the way, the narrative of Warzone: Scavenger is compelling in its own right. However, it is predominantly conveyed through textual means and indirectly implied through gameplay. The integration of cutscenes, voice-overs, or other narrative techniques could augment player engagement with the storyline and characters.

The game presently employs traditional keyboard controls. Future enhancements could involve the development of support for gamepad or even motion controls, potentially offering a more immersive drone piloting experience.

As illustrated, the faceless enemy forces in the game operate based on a relatively simplistic AI. An enhancement of the AI, rendering it more unpredictable and strategic, could elevate the game's challenge and captivation.

Therefore, There is scope for the future implementation of additional game modes, such as a survival mode, where the player must endure continuous bombardment as long as possible, or a time-trial

mode, requiring the player to amass a specified quantity of supplies within a predetermined timeframe.

In conclusion, Warzone: Scavenger, as it currently stands, offers a distinctive and captivating gaming experience. However, there exists significant potential for further development and refinement. By addressing the current limitations and augmenting its features, we can enhance its appeal and longevity in the competitive gaming industry.