DUNGEON DUALITY

1. Quick Overview **3**
- Project Motivation 3
- Goal of Project 3
- Use Cases and Features 4
- User stories 4

2. Features **4**
- Feature 1: Multiplayer Lobby System 5
- Feature 2: Networking Architecture and Synchronisation 6
- Feature 3: Player Movement and Combat System 7
- Feature 4: Flexible UI & Screen Sizes 8
- Feature 5: Enemy AI and Targeting System 9
- Feature 6: Health and Damage Systems (HUD Bar) 10
- Feature 7: Coin Collection and Scoring System 11
- Feature 8: Visual Effects and Animation System 11
- Feature 9: Audio Management System (Background Music & Sound Effects) 12
- Feature 10: Multi-Level Progression 13

4. Installation Instructions **14**
- System Requirements 14
- Testing Instructions 15

5. Testing **16**
- System Testing 16
- User Testing & Feedback 17

6. Software Engineering Principles **19**
- DRY (Don't Repeat Yourself) 19
- Separation of Concerns 19
- Composition Over Inheritance 19

7. Further work **20**
- Character upgrading 20
- Character Skills 20
- Infinite Space 20
- Player Cooperation 21
- Environmental Challenges 21
- Synchronised game pause and offline mechanics 21
- Game bosses 21
- Achievements 21
- Powerups 22
- In-game communication 22

8. Acknowledgements **22**

1. Quick Overview

The motivation behind Dungeon Duality stems from our desire to recreate the thrill and strategic camaraderie of dungeon-crawling games, infused with the addictiveness of roguelike mechanics and multiplayer cooperation.

Project Motivation

The inspiration for Dungeon Duality comes from our shared love of games that bring people together through challenge and cooperation. Whether it was growing up playing co-op with friends on the couch, or jumping online after a long day to tackle a roguelike dungeon, there was always something unforgettable about the mix of tension, laughter, and teamwork that games can create.

When we began talking about this project, we both wanted to recreate that thrill of working together to overcome tough challenges, even with people you've just met. Games have a unique way of building connection, trust, and excitement, turning strangers into allies as players strategize and adapt in the heat of battle.

At its core, Dungeon Duality is about capturing that spirit, and making it easy to hop into a dungeon with someone, communicate and coordinate, and come out the other side with shared stories of close calls and hard-won victories. We wanted to build a game where the experience of teaming up is as seamless as possible, allowing people to focus on the fun and the strategy rather than dealing with unnecessary friction.

Drawing inspiration from titles like Survivor.io, Archero, and Dungeons & Dragons, our aim is to create a networked co-op experience that challenges players while allowing character progression and customization.

Goal of Project

The goal is to develop a polished 2D multiplayer dungeon crawler where two players collaborate to survive through 3 escalating levels filled with waves of enemies, as the goal is to destroy the enemy tower base.

Use Cases and Features

Core Use Cases:

- Multiplayer dungeon crawling with progression
- Shared and synced environment for co-op gameplay

User stories

Core user stories:

1. As a user, I want to select a unique character before entering a level, so that I can play according to my preferred playstyle and avoid visual confusion with my teammate.
2. As a user, I want to play a game that becomes progressively more challenging with each level, so that I stay engaged and feel a sense of accomplishment as I improve.
3. As a user, I want to experience the game with a friend in a co-op setting, so that we can enjoy the thrill of teamwork, support each other, and strategise together.

Extensions:

1. As a user, I want to chat with my teammate in the lobby using temporary speech bubbles, so that we can coordinate roles or character selections before the level begins.
2. As a user, I want to be rewarded with achievements and loot based on specific actions, so that I feel recognised for my unique gameplay choices.

Tech Stack

- Game Engine: Unity
- Networking: Unity Netcode for GameObjects (Multiplayer HLAPI)
- Version Control: GitHub

2. Features

Here are the features currently implemented in our prototype

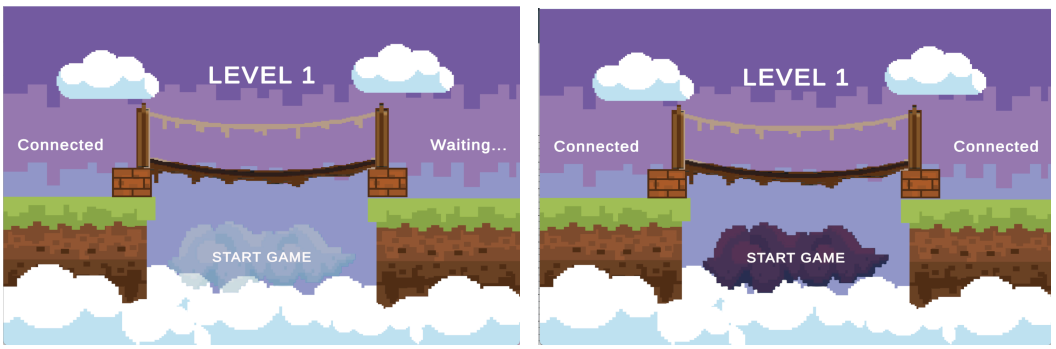
Feature 1: Multiplayer Lobby System

Our game includes a custom-designed multiplayer lobby system, enabling players to host and join cooperative sessions over a network. The lobby serves as the pre-game staging area where players configure settings, choose their character (in the future when it is implemented), and prepare for gameplay.

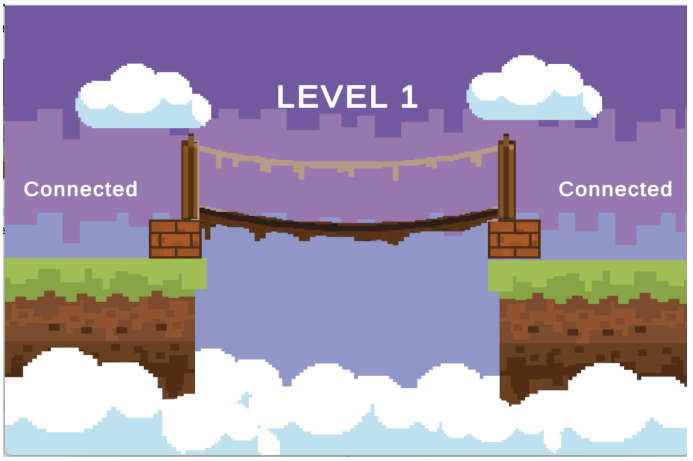
Key points:

- Two-player support with validation: The lobby permits exactly two players, ensuring consistent game balance.
- Start Game Control: Only the host can start the game once both players are present. This will be later extended to game settings as well
- Level Selection Binding: Level selection in the menu is propagated to the lobby so that all players join the same game environment.
- Scene Persistence: The NetworkManager is marked as persistent across scenes, maintaining connection state through transitions.

The multiplayer cooperative system is central to the dungeon crawler, allowing two players to explore and fight together in a synchronized, collaborative experience. Using Unity's Netcode for GameObjects, the system employs a hybrid authority model explained more in Feature 2.



Disabled on host when only one player is connected.



Client does not have a start game button

Feature 2: Networking Architecture and Synchronisation

The networking architecture represents a sophisticated implementation of Unity's Netcode for GameObjects, designed to provide smooth multiplayer experiences while maintaining game state consistency. The system uses a hybrid authority model that balances responsiveness with security, ensuring that players experience minimal input lag while preventing common multiplayer issues like desynchronisation and cheating.

The client-authoritative components, including ClientNetworkTransform and OwnerNetworkAnimator, handle aspects of the game that require immediate responsiveness. These components allow clients to make local changes to position and animation states that are then synchronised across the network. This approach eliminates the round-trip delay that would occur if these updates required server

approval, resulting in smooth, responsive gameplay that feels natural to players.

Server-authoritative systems manage critical game state elements that require consistency and security. Health values, damage calculations, and game progression events are processed exclusively on the server to ensure that all clients maintain synchronized game states. The RPC system facilitates communication between clients and server, with [ServerRpc] methods handling client requests and [ClientRpc] methods broadcasting server decisions to all connected clients.

Feature 3: Player Movement and Combat System

The player movement system provides intuitive and responsive character control that forms the foundation of the gameplay experience. Players use arrow keys to navigate through the dungeon environment. The movement implementation prioritises smooth, physics-based motion that feels natural and responsive while maintaining network synchronisation across all connected clients.

Player input is processed exclusively by the local owner to avoid input conflicts in multiplayer. Directional inputs are translated into normalized movement vectors that drive smooth, physics-based movement using Unity's Rigidbody2D. This approach ensures consistent velocity regardless of frame rate and provides natural collision responses, allowing players to collide against enemies or bounce off walls realistically.

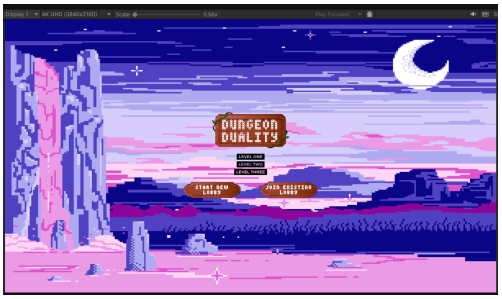
The custom struct is built by us for easier synchronisation across all of the different GameObjects and scripts for all types of movements.

```
public struct MovementDirection
{
    public float v;
    public float h;
}
```

The combat system integrates tightly with movement by maintaining the player's last facing direction, tracked as one of four cardinal directions. This facing direction governs both weapon positioning and attack orientation. When the attack input (spacebar) is received, the weapon dynamically adjusts its position and rotation using 2D Euler angles to align precisely with the player's current orientation. This ensures that melee attacks always originate from and face the expected direction, even if the player is stationary.

This system delivers responsive and intuitive melee combat that feels connected to player movement. Visual and audio feedback is synchronized with attacks to reinforce impact and timing, enhancing player immersion. The combination of physics-driven movement, collision handling, and directionally-aware combat contributes to fluid gameplay and strategic depth, encouraging players to position and time attacks thoughtfully during cooperative play.

Feature 4: Flexible UI & Screen Sizes



The UI is designed to dynamically adapt to various screen sizes and resolutions. During development and testing, enabling free size scaling allowed all UI elements to fit seamlessly within the same screen space. As a result, the interface maintains a consistent, polished appearance across different devices and aspect ratios, ensuring a smooth user experience regardless of screen dimensions.

Feature 5: Enemy AI and Targeting System

The enemy AI system creates challenging and engaging opponents that actively pursue players and create dynamic combat scenarios. Each enemy operates independently while maintaining awareness of player

positions and reacting appropriately to player actions. The AI system balances simplicity with effectiveness, ensuring that enemies provide meaningful challenge without overwhelming players or creating frustrating gameplay experiences.

The enemy targeting system implements a basic but effective player-seeking behavior where enemies automatically identify and pursue the nearest player character, ensuring that enemies remain engaged with players throughout the gameplay session and adapt to changing battlefield conditions.

Movement AI is implemented through a combination of distance calculation and vector mathematics to create smooth, natural-looking pursuit behavior. The CalculateMovementVector() method determines the direction toward the target player and normalizes the resulting vector to ensure consistent movement speed. The system includes a maximum engagement distance check, preventing enemies from pursuing players across the entire map and creating opportunities for strategic retreat and positioning.

The collision and knockback system adds tactical depth to enemy encounters by implementing realistic physics responses when enemies contact players. When an enemy collides with a player, the system calculates the collision point and applies an appropriate impulse force to push the enemy away from the player. This mechanism prevents enemies from becoming stuck against players and creates dynamic positioning opportunities during combat. The collision timeout system ensures that enemies don't immediately re-engage after being knocked back, providing players with brief recovery windows.

Feature 6: Health and Damage Systems (HUD Bar)

The health and damage systems provide the core mechanical framework for combat encounters and player progression. These systems ensure that combat has meaningful consequences while maintaining game balance and providing clear feedback to players about their current status.

The health system for both players and enemies uses NetworkVariables to ensure that health values remain synchronized across all clients. The system implements value change callbacks through the OnValueChanged event, allowing for immediate UI updates and effect triggers when health values change.

The damage application system uses server-authoritative Remote Procedure Calls (RPCs) to ensure that damage calculations are consistent and cannot be manipulated by clients. When a player or enemy takes damage, the TakeDamageServerRpc() method is called, which executes only on the server and updates the authoritative health value. This approach prevents cheating while ensuring that all clients receive consistent damage information. The system includes bounds checking to prevent health values from becoming negative and handles player death by triggering appropriate game state changes.

Feature 7: Coin Collection and Scoring System

The coin collection system provides immediate rewards for combat success and creates a foundation for resource management and progression mechanics. Players earn coins by defeating enemies, with the collection process providing satisfying audio-visual feedback and contributing to their overall score. This system encourages active engagement with enemies and provides measurable progress metrics for player achievement.

The coin collection mechanism uses Unity's trigger collision system to detect when players interact with coin objects in the game world.Upon collection, the system increments the player's coin count, destroys the coin object, and updates the user interface to reflect the new total. This process provides immediate feedback and reinforces the reward cycle for successful combat encounters.

Feature 8: Visual Effects and Animation System

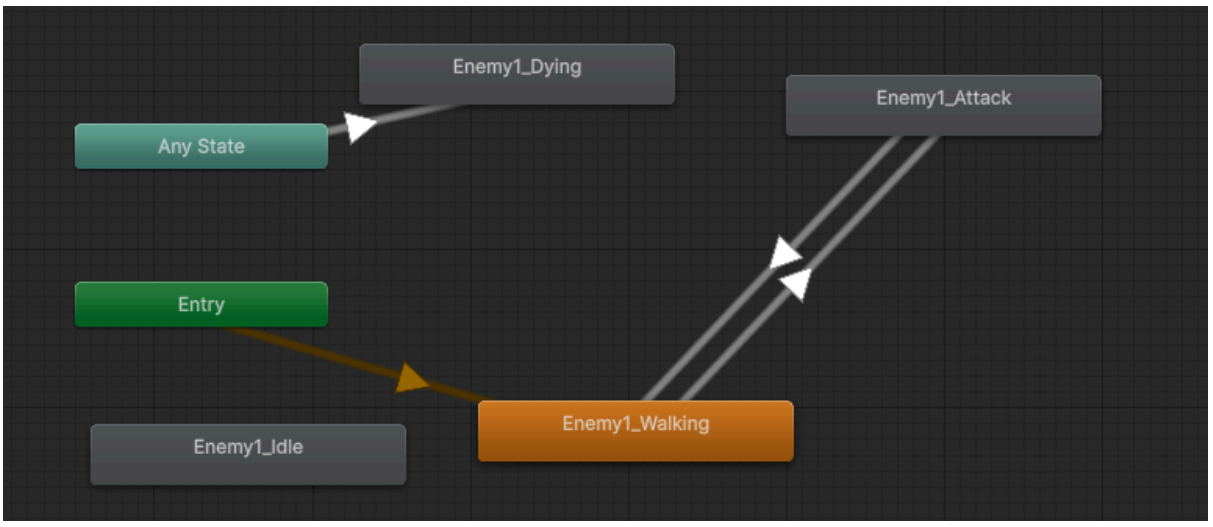
The visual effects and animation system enhances the gameplay experience by providing immediate feedback for player actions and game events. The system includes damage indicators, weapon animations, and character movement animations that create a cohesive and engaging visual experience. These effects are synchronised across all network clients to ensure that all players see consistent visual representations of game events.

The damage visualisation system provides immediate feedback when players or enemies take damage through temporary blood effects. The ShowInjuredBlood() method activates blood sprite overlays for a brief duration, creating visual impact that reinforces the combat system. The timing system uses Unity's Invoke() method to automatically hide these effects after a predetermined duration, ensuring that the visual effects don't become overwhelming or permanent.

The weapon animation system creates dynamic attack visualisations that correspond to the player's current facing direction. When players attack, the weapon sprite appears and rotates through a swing animation that provides satisfying visual feedback. The system adjusts weapon positioning and rotation based on the attack direction, creating convincing animations for attacks in all four cardinal directions. The animation includes collision detection through child objects, ensuring that the visual weapon swing corresponds to actual damage application.

Character animation is handled through Unity's Animator component, with the PlayerAnimator class managing animation state transitions based on player input. The system translates movement input into appropriate animation states, ensuring that character sprites accurately reflect the player's current actions. The animation system includes idle, walking, and directional states that provide visual variety and help players understand their current movement state and facing direction.

For enemies, we used triggers in Unity's Animator, that is called from the Enemy script.



Feature 9: Audio Management System (Background Music & Sound Effects)

The Audio Management System enriches the immersive experience of the dungeon crawler by dynamically delivering both ambient background

music and impactful sound effects based on the game context. This system centralises all audio functionalities into a singleton AudioManager component, ensuring persistent and globally accessible sound control across all game scenes and states. By separating music and sound effects into two dedicated AudioSource instances, musicSource and sfxSource. the system enables fine-grained control over audio playback, volume levels, and transitions between musical themes.

The system initialises during the Awake() phase and persists across scene loads using Unity's DontDestroyOnLoad, ensuring that audio playback remains seamless from lobby to gameplay and back. It provides intuitive public methods to play, pause, resume, and stop background music, with contextual transitions such as PlayLobbyMusic() and PlayGameplayMusic() triggered by game state events. To prevent redundant playback, the PlayMusic() method checks for already playing clips and skips reloading them if unnecessary.

Sound effects, such as weapon attacks, are handled via the PlaySFX() method, which uses AudioSource.PlayOneShot() for instant feedback without interrupting ongoing music. The system also includes muting controls and runtime volume adjustment interfaces through SetMusicVolume() and SetSFXVolume() to support player preferences or adaptive audio balancing. Overall, this system adds auditory depth to the player experience while maintaining modularity and ease of integration with game events like combat or scene transitions.

Feature 10: Multi-Level Progression

The game features multiple levels that progressively increase in difficulty and variety. Each level introduces distinct enemy sprites to visually communicate differences in enemy strength and damage potential, helping players quickly identify threats. Alongside visual changes, levels vary key gameplay parameters such as enemy spawn rates, health points, attack damage, and the number and locations of spawn points.

These variations create a dynamic challenge curve, requiring players to adapt their strategies as they advance. Increased spawn rates and additional spawn points in later levels intensify combat, while tougher enemies with higher HP and damage values demand more coordinated teamwork and tactical play. This layered design not only enhances gameplay depth but also reinforces a sense of progression and accomplishment.

Level One	Level Two	Level Three
Basic introduction to game mechanics	Increased spawn locations, rate, enemy health	Increased spawn rate, enemy health

After each level, either the player finished, or died.

4. Installation Instructions

Our project is accessible at this link, OR download and extract the zip file named ‘Dungeon Duality.zip’ encased within the folder of 7477.zip (our code).
https://drive.google.com/drive/folders/1TrQNcTG7e1FmNorMJmFrq7qIGnRknaC3?usp=sharing

System Requirements

This is only currently tested on Windows 11.

Testing Instructions

Since the game requires multiplayer, you will have to simulate the multiplayer on your own.

1. First, open two identical windows. You may resize the windows to fit two on the same screen if you do not have a monitor.

On the first window,

2. You will arrive at the main menu. Here, select which level you wish to play by clicking it. Currently all the levels are unlocked for you to try. The selected level is highlighted in red.
3. Click ‘Start new game’. If you did not select a level, the game will automatically assume you wish to enter the first level.
4. You will then enter a lobby that has ‘Connected’ on the left and ‘Waiting…’ on the right, showing that it is waiting for the other player since this is a 2 player game.

On the second window,

5. Choose the same level and click ‘Join existing game’. This will act as the second player joining the first player’s lobby.

On the first window,

6. The button to start the game, which was previously disabled, will be enabled. It looks like a cloud. Click the button to begin the level.

Each window will then have identical controls.

- WASD to move around
- Space bar to attack

You may test some features such as

- Dying when touching the pothole
- Synced movement in 8 directions
- Collision with the walls and other player and enemies
- Attacking the enemies in 4 directions
- Health dropping as enemies attack you
- Death from lack of health

- Attacking the enemy tower and destroying it

A game over screen or a game won screen will appear for both players if both players die, or the tower is destroyed respectively.

Have fun!

5. Testing

System Testing

System testing was conducted to verify the overall functionality and stability of the game before releasing it to users. This phase involved rigorous internal testing by the development team, focusing on identifying and fixing bugs, ensuring game mechanics worked as intended, and verifying network stability for multiplayer features.

Key testing activities included:

- Functionality Testing: Ensured all game features such as player movement, combat, enemy spawning, level progression, and UI responsiveness operated correctly without crashes or glitches.
- Network Testing: Validated synchronization across clients in multiplayer sessions, checked latency handling, and ensured consistent game state during typical gameplay and edge cases like player disconnections.
- Compatibility Testing: Tested the game on various screen sizes and resolutions to confirm the flexible UI scaled properly and remained usable.
- Regression Testing: Repeated tests after fixes to ensure that new changes did not break existing features.

The system testing phase provided the foundation of a stable and playable game, reducing the risk of major issues during user testing.

User Testing & Feedback

We tested the game with a few of our friends and families, and here were some of the responses we got.

QUESTION	ANSWER
Was the objective of the game clear from the beginning?	Yes, quite clear, although it took me a while at first to realise I needed to knock down the white pillar.
Did the pacing of the game feel appropriate across the 3 levels?	Yes, because it got gradually harder to complete after each level.
How satisfying did you find the core game loop (exploration, combat, coin collection)?	The later challenges were challenging enough that it was possible to die, so beating those levels felt quite rewarding.
Were you ever unsure what to do next or how to progress in the level?	No, as the objective remained constant throughout the levels.
Was it easy to connect with another player and start a session?	Yes.
How well did the two-player mechanics (e.g., shared goals, coordination) work?	Well, we kind of both just spammed the pillar....so not much coordination there.
Was it easy to stay synchronized with your teammate (e.g., no major lag/desync issues)?	Yes, no lag issues.
Did the co-op experience feel meaningful and encourage teamwork?	Yes, I used my teammate as a meat shield which was fun and encouraged teamwork.
How intuitive were the basic controls (movement, attacking, etc.)?	Quite intuitive once I found the controls, but took me a while to find them (instructions would be good).
Did the combat feel responsive and impactful?	Yes, animations, sound effects and health bars helped me to gauge the timing and impact of my attacks.
Was there any input lag or unresponsiveness, especially in multiplayer?	Nope, no input lag.
Did enemies pose a fair challenge?	Yes, it got decently challenging in the later stages.
Did the enemy AI behavior feel dynamic or too predictable?	They kind of just rush you with attacks which is predictable, but at least the spawn locations are random which is good.
Did enemy variety keep the gameplay interesting across levels?	Sadly no, as they all more or less function the same way across levels, just with

	varying attack damage.
Did knockback and other feedback effects enhance the combat experience?	Yes, it felt like I was being forced into a corner by the enemies at times which was quite fun.
Did the level design offer enough visual variety to prevent confusion?	Yes, the different enemies between levels was a good touch.
Was it easy to monitor your health and other key stats?	Yes, it was very clear easy to monitor due to its size
How clean or cluttered did the HUD feel?	Placement of the health bars felt a bit random, the one for the pillar seems too high and the one for the player is not near any of the edges.
Would tooltips or item descriptions be helpful during gameplay?	Instructions and descriptions would make it feel more immersive.
Did you experience any bugs, glitches, or unexpected behavior?	Sometimes I was bumped off the screen by enemies and unable to reenter the screen. I also occasionally had some trouble starting games after finishing a level (even when both players were connected).
Did the game handle death, game over, and victory transitions properly?	Yes, there were clear UI interactions and graphics to show them`
Would you be interested in character upgrades or different classes?	Yes, I would like to see different kinds of attacks for the players and the enemies to spice things up.
Would randomly generated environments enhance replayability for you?	Yes, rogue-likes are quite enjoyable although they also need to have interesting variations - a common way to achieve this is to enable all kinds of interactions (between items, enemies, playable characters).
How appealing are power-ups and environmental hazards as gameplay additions?	Very, I want to have something stopping me from simply dashing to the pillar to spam attacks before I die.
What was the most enjoyable part of the game?	Killing the enemies was satisfying as heck, although there was no great incentive for me to do so - coins don't do anything, and in levels 2 and 3 I got swarmed so much that I had to focus on hitting the pillar before I died.
What was the most frustrating part of the game?	Navigating back to the levels I wanted to replay, and having to press many buttons/restart game to do so.
What suggestions do you have for	The co-op is quite solid, I would focus on

improving the co-op experience?	improving level design and in-game interactions. But for co-op specifically, you could have enemies that only 1 player can see but only the other can attack.
Would you recommend this game to a friend? Why or why not?	I would, as it is fun to move around in the arena together. However, as a game it is quite simplistic at the moment, although still fun.

6. Software Engineering Principles

DRY (Don't Repeat Yourself)

We made a conscious effort to avoid duplicating code. A clear example is the enemy logic, which reused a single Enemy.cs script across all levels.

Although the enemy prefabs varied due to different animations, the underlying script remained the same, which helped us manage shared behavior efficiently.

Separation of Concerns

We had basic separation between gameplay (Player.cs, Enemy.cs), networking (ClientNetworkTransform.cs, NetworkManager), and UI (contained within the UI/ folder).

However, due to time constraints, some scripts like Player.cs and GameManager.cs started becoming bloated, taking on more responsibilities than they ideally should have.

Composition Over Inheritance

Our behaviors were mostly built using composition. For example:

Attack behaviors were encapsulated in Weapon.cs and WeaponCollider.cs instead of subclassing from Player.cs.

Animation syncing was handled separately via PlayerAnimator.cs.

This approach helped us keep scripts reusable and easy to debug.

7. Further work

There is potential to develop a polished 2D multiplayer dungeon crawler where two players collaborate to survive through 20 escalating levels filled with waves of enemies, culminating in a final boss fight. Along the way, players will collect loot, upgrade their characters, and make strategic decisions that affect both short-term survival and long-term progression.

Character upgrading

We plan to expand this project into having multiple characters with multiple attack types and different stats (movement speed, attack speed, attack range, etc.). The player should be able to raise his stats with coins and special items collected.

Character Skills

The characters all come with a special skill. For example, the healer can heal the lowest health member every specific time interval. The giant can create shockwaves that damage an AOE surrounding him with every specific number of attacks.

Infinite Space

We plan to generate a repeating tilemap with randomly generated areas (environments such as ice, grass, volcanic, etc.) and random environmental items such as rocks, houses, campfires (where characters can heal at).

Player Cooperation

Further to the multiple characters, we also plan to have players cooperate more by being able to revive each other within 20 seconds of being downed, but will require an inactivity of 5 seconds.

Environmental Challenges

We wanted to integrate different environmental challenges (e.g. thorns that prevent movement and causes damage over time, lightning that strikes the an area on the ground, earthquakes that stun the player, etc.). However, unfortunately, we didn't have much time to do these.

Synchronised game pause and offline mechanics

The multiplayer game right now is not optimised for co-op as the player is reliant on the other player not going offline in the middle of the game. Game pauses by the host should be enabled, and the game ending when one player goes afk should also be implemented.

Game bosses

At the end of every level, after the tower is destroyed, a unique game boss should be placed, angry about his tower being destroyed. Then the boss mechanics will be different from level to level, and the player has to fully beat this boss in order to move onto the next level.

Achievements

Achievements can be unlocked that give special items that can be equipped such as a helmet or a coat. This will give stat boosts or special skills. For example, a medic achievement which can be unlocked by reviving a player 3 times, will give the passive skill of +5 to health every 1 minute to both players.

Powerups

During the level, small powerup items will be placed that can give a time-limited boost to the characters. For example, a health potion will add some health, a rage potion will increase attack damage, movement speed and attack speed for 10 seconds, etc.

In-game communication

Chat bubbles, preset chat messages and emotes will be added as a part of improving the user experience as well.

8. Acknowledgements

Assets

We used free assets from CraftPik for sprites, Vecteezy and FreePik for backgrounds.