UNIVERSITI TUNKU ABDUL RAHMAN (UTAR)

Lee Kong Chian Faculty of Engineering & Science (FES LKC)

Bachelor of (Honours) Software Engineering [SE]

Alex the Courage

Group Name	Name
Student Names & (IDs)	Hon Wen Xuan (2001171) Cheah Wen Huey (1804877) Harry Liow Ming Heng (2002435) Liang Yi Wei (2100506) Ong Ying Shuang (1802307)
Due Date	22 September 2023
Lecturer	Tai Liang Kwang
Marks	

Table of Contents

Project Overview

Game Overview

Alex the Courage is a single-player platform game designed to provide players with a thrilling and satisfying experience reminiscent of the popular title "Fall Guys." In this adventure, players control the fearless character, Alex, as they navigate through five exciting and challenging levels filled with obstacles, enemies, and checkpoints.

Story

In the mystical world of Ardentoria, where magic flowed like a river and courage was a rare and precious gem, there lived a young warrior named Alex. Alex was known far and wide as "Alex the Courage" for his unwavering bravery in the face of adversity. He had always been a force to be reckoned with on the battlefield, feared by his enemies and admired by his comrades.

But fate had a cruel twist in store for Alex. During a fierce battle against the dark sorcerer Malgrim, Alex's right hand was severely wounded by a malevolent spell. The battle was won, but victory came at a great cost. His hand, once a symbol of his strength, was now broken and withered, devoid of its power.

In Ardentoria, warriors drew their strength from the mystical tattoos that adorned their bodies, and the source of their power resided in their dominant hand. For Alex, this meant that his ability to harness magic and wield formidable weapons was forever lost. His once-mighty sword, Emberstrike, lay dormant, unable to respond to his diminished touch.

But Alex was not one to be defeated so easily. He refused to let his injury define him. With sheer determination and an unyielding spirit, he began a new journey to regain his strength and find a way to restore his hand's power. Along the way, he discovered the ancient art of evasion, a skill that allowed him to move with unmatched agility and grace.

Now, Alex the Courage must navigate treacherous terrains, avoid cunning enemies, and solve intricate puzzles with his newfound evasion abilities. He may not be able to attack his foes directly, but his agility and cunning make him a formidable adversary in his own right.

Prominent Game Features

UI Features:

1. Lively and Colorful Designs: The game's user interface features vibrant and eye-catching designs, with a playful and fun aesthetic. The color palette is filled with lively hues that reflect the game's cheerful atmosphere.

2. Orchestra Music: The game's soundtrack is composed of orchestral music, enhancing the overall atmosphere and immersing players in the fantasy world of Ardentoria. The music complements the game's tone, whether it's the excitement of exploration or the tension of evading enemies.

3. Main Menu Options:

- Start Game: Allows players to begin their journey. They can select specific maps or levels from here.
- Exit Game: Provides an option to exit the game entirely.
- Settings: Access to various game settings, including audio and screen settings.
- Select level: Provides options to navigate through different levels.

4. Pause Menu Options:

- Resume Game: Allows players to continue their current game.
- Restart Game: Allows player to restart the current level.
- Settings: Access to in-game settings, where players can make adjustments on the fly.
- Exit Game: Provides an option to leave the game and return to the main menu.

Physics:

Movement	Keybind
Upward	W
Backward	A
Crouch	S
Forward	D
Jump	Spacebar
Roll	E

AI Features:

1. Enemy AI:

- Enemies have distinct sight ranges, capable of spotting the player within a certain radius.
- Once an enemy detects the player, they will chase and initiate attacks.
- Different enemy types exhibit unique attack patterns, increasing the game's challenge and variety.

2. Obstacles AI:

- Interactive obstacles respond to the player's actions and trigger traps or other in-game events accordingly.
- These obstacles add an element of strategy and puzzle-solving to the gameplay.

Level Goal:

5 Maps: The game is divided into five distinct maps or levels, each with its own unique environment and challenges. These levels include:

Map / Level	Theme / Location	Description
1	Forgotten Island	This serves as the starting point for players, introducing them to the game mechanics and controls. Players must navigate through the island, overcoming obstacles to reach the map's endpoint.
2	Winter Forest	The second map presents a chilly, winter-themed environment with increased complexity. Players continue their journey, encountering new challenges and enemy types as they strive to reach the endpoint.
3	Winter Forest II	This map builds upon the themes and challenges introduced in Level 2, offering a deeper and more intricate gameplay experience within the winter forest setting.

4	Uhara Desert	The fourth map transports players to a scorching desert landscape. Here, they face a fresh set of obstacles and enemies while maintaining their goal of reaching the map's endpoint.
5	Uhara Desert II	The final map in the game intensifies the challenges and tests players' skills acquired throughout their journey. It serves as the ultimate test of their courage and determination to complete the game.

2. Procedurally Generated Terrain: The terrain in each map is procedurally generated, ensuring that each terrain offers a unique experience.

3. Objective: The primary goal in each map is consistent: the player must find a way to reach the endpoint of the level. To achieve this, they need to navigate through obstacles that block their path and avoid or overcome enemies.

Team Designation

Hon Wen Xuan – Audios, UI, integration of features, basically involve a bit on every aspect.

Harry Liow – Animation, a bit of Enemy AI and traps logic

Cheah Wen Huey, Liang Yi Wei – Enemy AI, traps logic, enemy features design

Ong Ying Shuang – Terrain, day night cycle

Project Technical Design Overview

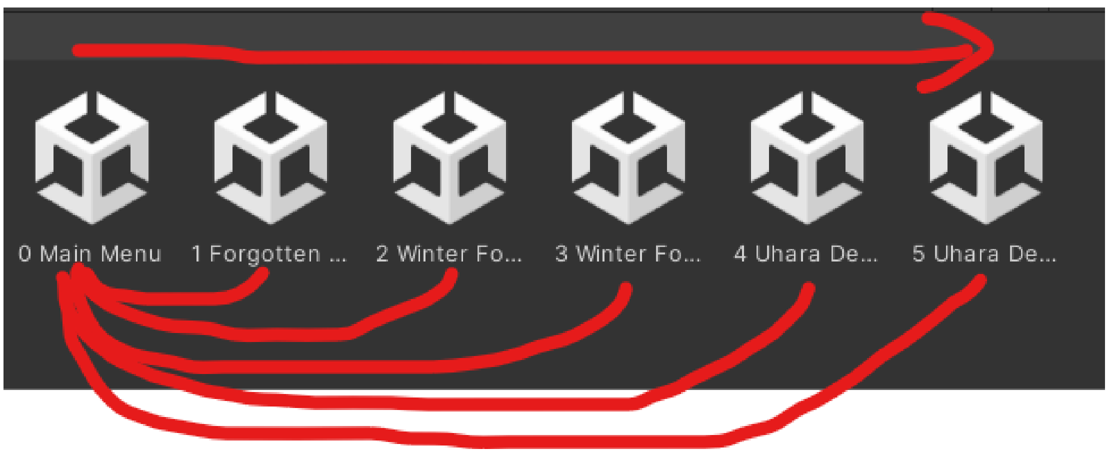
Project Game Development Setup

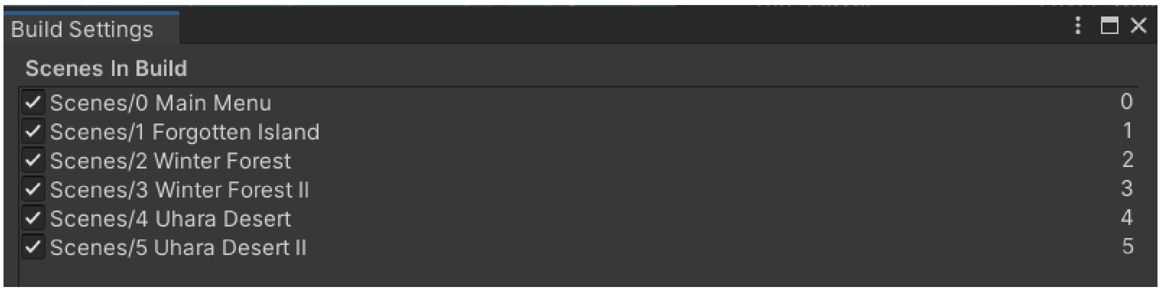
Project Organization Overview

The game assets are properly organized in the folders based on their uses. There is Animation folder that contains everything related to player's animations logic, Audio folder that stores the background music and sound effects, DayNightCycle folder that contains everything related to day night cycle, Decoration folder that stores assets for the level's environment, Enemy folder that stores enemy model and scripts that manage their AI, Materials folder that stores materials, Obstacles folder that contain every kind of obstacles and traps used in levels, Prefabs folder that contain prefabs that can put into levels that makes everything easier to implement, Scenes folder that contain scenes, Script folder that contain scripts, Terrain folder that stores assets of terrains, and UI folder that stores every UI elements.

Scene Overview

There are 6 scenes in the project. One scene being the main menu, while the other 5 being the different levels. The game starts in the main menu scene, and in there player can choose the level they want to play, or just click the play button which will send player to the first scene which is the first level of the game. Once player completes a level, player can choose to play next level which will send player to the next scene, or they can choose to return to the main menu. Completing the last level will only allow player to restart the scene or return to the main menu.




Build Settings

Scenes In Build

✓ Scenes/0 Main Menu	0
✓ Scenes/1 Forgotten Island	1
✓ Scenes/2 Winter Forest	2
✓ Scenes/3 Winter Forest II	3
✓ Scenes/4 Uhara Desert	4
✓ Scenes/5 Uhara Desert II	5

Technical developmental process

Planning and Design

Planning is the most crucial part of any software development project where decisions are made to set the course throughout the entire development process. To kick start our project, we had a brainstorming session to decide on the game genre, title, background as well as the main features and the overall flow of our game. Game physics for character movement and level goals designs are also included during the brainstorming session. To offer gameplay diversity and skill-challenge balance, a total of five unique levels are designed using different layouts of maps and terrains, obstacles and types of enemies, where each level will have progressively increasing difficulty as players have cleared a level. Lastly, each member was assigned with their respective tasks, and milestone planning was carried out to keep project deliverables on track.

Environment Setup

In setting up the development environment, Visual Studio Code is integrated with Unity as its code editor by configuring Visual Studio Code as the External Script Editor under Unity Preferences, External Tools. In Visual Studio Code, necessary extensions such as Unity for Visual Studio Code, C# and C# Dev Kit are installed from the Extension Marketplace. Basic editing features including syntax highlighting, smart code completions and code navigation are supported for highly productive source code editing as well as to boost the development process. Meanwhile in Unity, required assets, libraries and plugins from Unity Asset Store were imported into our project using Unity Package Manager. For better version control and team collaboration, GitHub was chosen as the platform for us to streamline the game development process and maintain a well-organized, documented project.

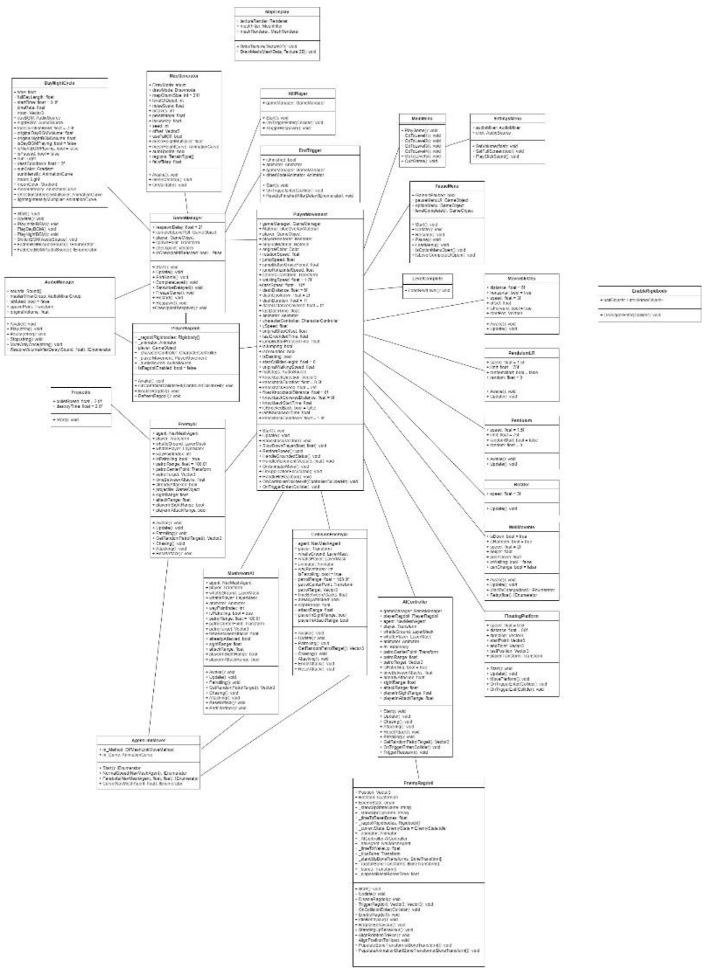
Development and Implementation

As a common and essential mechanic, player movements controls including idling, walking, jumping, and rolling are implemented to allow characters pass through obstacles and terrains in order to complete a level. The main camera was also attached to the tagged player object that will provide a third person view along player’s movement and navigation. Animation effects such as ragdoll animations, knockback and slow motion were added to make the player experience visually engaging. Audio is the key contributor to the overall gaming experience. Each level comes with its respective background audio that matches the atmosphere and ambiance to make virtual environments alive and immersive. To increase player engagement and skill development, different types of enemies that are capable in path finding, chasing, and attacking are created by utilising Unity’s NavMesh class. Meanwhile, the method of procedural terrain generation is used to automate the creation of landscapes whereas day-night cycle was applied to add a layer of realism to the visual environment.

Integration and Test

The integration of assets and game components from each member into a cohesive game project can be complex and time-consuming but with the utilisation of GitHub, this process was made easier and effective. When all level designs were completed, we conducted testing on each level design iteratively by simulating a player's experience in accomplishing the level goals and proceeding throughout the levels. As such, bugs and errors can be identified whereas the balance between difficulty and enjoyment can be evaluated to improve game quality and gaming experience.

Class diagram



Feature Implementation Summary

Game Object Design Setup

Each of the scene other than the main menu scene will have the following game objects in common: Game Manager, Audio Manager, Day Night Cycle, Player, UI (user interface), points, and terrain. Other game objects are depending on the level design, such as enemies, obstacles and environment.

Game manager containing a script that will handle the states of the game such as pause, restart, or resume the game, the player states such as respawn, mark down checkpoints location, and the completion of a level.

Audio Manager contains a script that create a list that can store multiple audio sources and have functions that can be called from other scripts to handles the audios.

Day Night cycle containing a sun, a moon, and a script that handles the rotation aspects, reflection intensity, and light intensity of the sun and the moon, as well as handle the background music to play at different time of a day.

Player game object containing the actual player object of the game and its respective camera that will follow the player.

UI containing pause menu that will show up when the game is paused, and an option menu nested inside the pause menu that will show up when the “options” button is pressed in the pause menu. UI also containing win screen that will only show up when player reach the finish point.

Points including the spawn point where the player will respawn when the checkpoints is not reached, the checkpoints which will mark the player’s new respawn point and get destroyed when player touches it, and the finish point which will end the game and show the win screen when the player reach it.

Terrain is randomly generated using a script, it has unequal heights, and it allows player to walk on it. The environments (or decorations), obstacles, and enemies are then put on the terrain to design levels.

Player Controls Implementation Overview

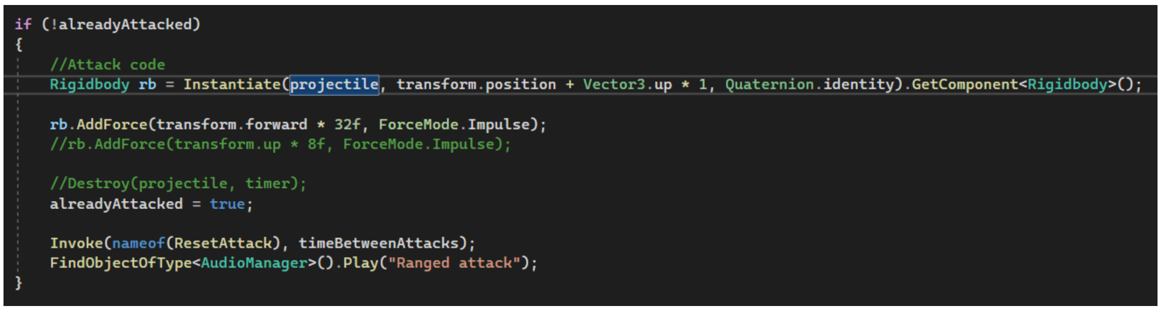
The player object can be control using WASD and spacebar to move around and jump, and pressing E can let player roll to the direction that the player is facing. User can use mouse to move the camera around, including looking up and down. Moving the camera while the player object is moving will change the direction that the player object is moving to.

Game Mechanics/Feature Implementation

This game is in platform genre, which player need to avoid all the obstacles and enemies and find path to reach the finish point. Since player can’t attack the enemies (designed as intended), player need to find ways to avoid the enemies, for example, hide behind buildings to dodge the attack from enemies. Some enemies and obstacles won’t be able to kill the player, but they will disturb player to reach the finish point. They will have the ability such as knockback and slow player. Some enemies and obstacles are fatal and will be able to kill the player. In a stage, player can also die by falling off the platforms. Player will respawn at the spawn point or the checkpoint after few seconds of their death. Player wins the game if they reached the finish point.

Game Physics Implementation Overview

- **Projectile physics**: Some enemies will shoot ball-like projectiles, or bullets when the player is in their attack range. The force will be applied to the bullets to the direction of the player, and it will fall to the ground gradually due to gravity. Below is the code to implement shooting bullets to the player.



```
if (!alreadyAttacked)
{
    //Attack code
    Rigidbody rb = Instantiate(projectile, transform.position + Vector3.up * 1, Quaternion.identity).GetComponent<Rigidbody>();

    rb.AddForce(transform.forward * 32f, ForceMode.Impulse);
    //rb.AddForce(transform.up * 8f, ForceMode.Impulse);

    //Destroy(projectile, timer);
    alreadyAttacked = true;

    Invoke(nameof(ResetAttack), timeBetweenAttacks);
    FindObjectOfType<AudioManager>().Play("Ranged attack");
}
```

- **Jump, roll, and fall physics**: Upward force will be applied to the player if the player jumps, and player will fall on the ground due to the gravity. The jump height is enough to jump over some obstacles. When the player is on the air, they won’t be able to jump again until they touch the ground. When there is roll command, the player will roll to the direction they are moving to, and the rolling speed is a bit faster than the walking speed. Player can roll under certain obstacles that are not able to go through when walking towards it, because when rolling, the player’s collider will become smaller. Player will fall down if they didn’t stand on any collider.

Animation Implementation Overview

1. The Character

Animator States:

IsDashing: This state is triggered when the player is in a dashing move.

IsFalling: This state is triggered when the player is falling, typically after a jump or when not grounded.

IsGrounded: This state is used to check if the player character is on the ground.

IsJumping: This state is triggered when the player is performing a jumping action.

IsMoving: This state is true when the character is moving and false when the character is stationary.

IsHit: This state is triggered when the player collides with an enemy or a bullet.

IsFinished: This state is triggered the victory animation when the player reached the finish line.

InputMagnitude: This is not a state but a parameter used to control the animations based on the magnitude of the player's input, representing how much the player is moving.

Animator Transition/Conditions:

The transitions between the states are typically controlled by the associated bool parameters mentioned above. For example:

If IsGrounded is true, the character will transition to the grounded state.

If IsJumping is true, the character will transition to the jumping state.

If IsDashing is true, the character will transition to the dashing state, and so on.

2. The Enemy AIs:

Animator States:

Idle: The default state where the enemy is stationary.

Move: Triggered when the enemy is in pursuit or moving around.

Attack: Activated when the enemy is in the range of the player and initiates an attack.

Hit: Triggered when the enemy gets hit by the player or an obstacle.

Ragdoll/Wake-up Animation: Activated when enemy experiences extreme forces or impacts, and then transitions back to previous states upon recovering.

Animator Transition/Conditions:

Speed Parameter: Controls the speed at which the enemy moves, affecting transitions between idle and move states.

IsAttacking Parameter: Boolean parameter to determine if the enemy is in an attacking state.

IsHit Parameter: Boolean parameter to determine if the enemy has been hit.

IsRagdoll Parameter: Boolean parameter to determine if the enemy is in ragdoll state.

The script implies a finite state machine within the Animator, where the character transitions between different states like Dashing, Falling, Grounded, Jumping, Moving, and Hit, based on player inputs and collisions with other game objects.

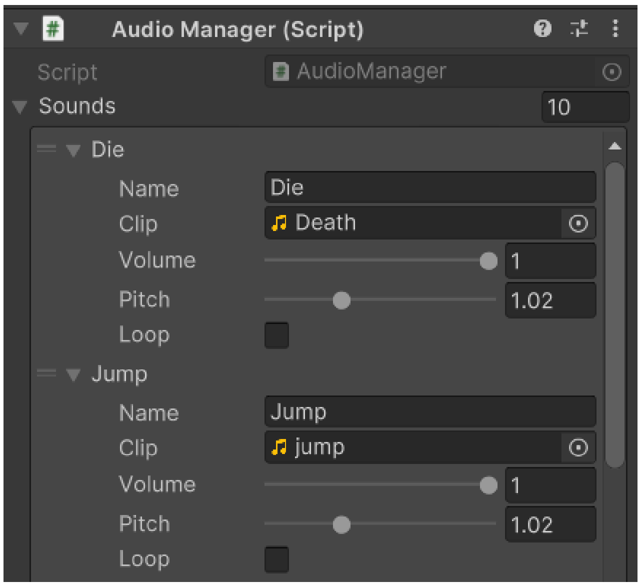
These states will be typically set up in the Animator window of Unity, with transitions and conditions based on the parameters mentioned. For example, the IsJumping state will have a transition to the IsFalling state when the ySpeed is less than 0, and similarly, other transitions will be defined based on the game's logic and requirements.

Audio Implementation Overview

The audio that we are using has 2 categories, which is background music and sound effect. In the Audio folder, there are 3 subfolders which is BGM, Sound Effects, and Scripts, the files inside the folders are self-explanatory. All audio files are in MP3 format.

Each of the level have their own background music, and the music will switch based on the in-game time. The background music switch logics are implemented inside the DayNightCycle script so the music will switch correctly based on the time. During the switch of the music, the current music will slowly fade out, and the next music will fade in seamlessly. When the game is paused, the background music’s volume will be lowered, and when the game is resumed, the music volume will be set back to normal.

The sound effects are any kind of sounds other than the music. It can be the clicking sound in the menus, player walking, jumping, and dying sound, checkpoints touching sound, enemies sound, and etc. Most of the general sound effects are store in a game object call Audio Manager, which has a script in it call AudioManager to manage all the sound effects. The AudioManager script will create a list that will use the Sound class that we created and it uses the object oriented approach to store the audio information such as its name, audio source, and volume. The AudioMananger class will have public methods such as Play() and Stop() to enable other game object to control the audio sources using scripts. In the unity editor, in the AudioManager script, just add a list and put the audio information in the inspector for it to store the audio sources.


Audio Manager (Script)

Script: AudioManager

Sounds: 10

Die
- Name: Die
- Clip: Death
- Volume: 1
- Pitch: 1.02
- Loop: ☐

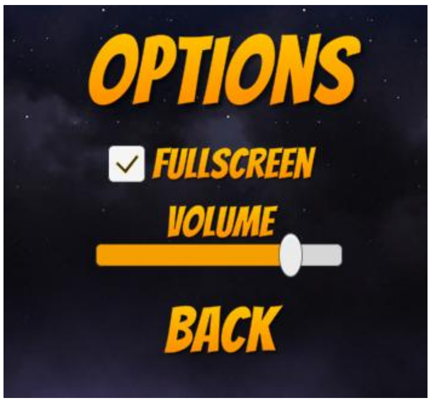
Jump
- Name: Jump
- Clip: jump
- Volume: 1
- Pitch: 1.02
- Loop: ☐

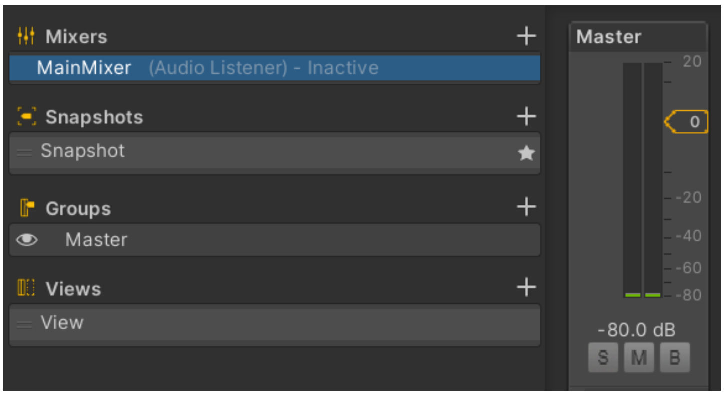
To play a sound, simply call the class and search the audio by name. Below is the example code to call the jump sound when the player jumps.

```
if (Input.GetButtonDown("Jump") && characterController.isGrounded) // Check if the player is grounded
{
    jumpButtonPressedTime = Time.time;

    // jump sound
    FindObjectOfType<AudioManager>().Play("Jump");
}
```

In the options menu, we can adjust the master sound volume of the game using the slider UI. The audio volume control is using the linear control, so that the user will hear that the volume is gradually decreasing when they lower the volume but not a sudden big drop of volume when decreasing the volume near the end, since the default audio control is logarithmic scaling. The slider volume controller is connected to the audio mixer so when the slider's value is changing, the audio mixer's volume will also change accordingly. All of the audio sources' output is then set to the audio mixer so the volume controller can control all audio volume while in game.





Game AI Overview

In this project, four distinct enemy AI entities will be implemented, each with unique characteristics tailored to their role in the game. These enemies are designed to engage with the player using a combination of patrol range, sight range, and attack range parameters. For example, when the player ventures into an enemy's sight range, it triggers an engaging pursuit as the enemy relentlessly chases the player until they fall within the attack range. At this juncture, the enemy transitions into an offensive stance, launching attacks. If the player manages to escape beyond the patrol range, the enemy will diligently return to its central patrol point, resuming its prescribed route. All four enemy types are equipped with the ability to both chase and attack the player, making for dynamic and engaging encounters. Furthermore, the attacks executed by these enemies will be diverse, falling into two main categories: projectile-based and melee attacks. This variety in attack styles will not only add depth to the gameplay but also require players to employ different strategies and tactics when facing these foes. The table below shows the enemy AI entities along with their descriptions.

Enemy	**Descriptions**
Beholder	Beholder is the enemy AI that serves a distinctive projectile-based attack mechanic. When a player is struck by one of these projectiles, the game immediately concludes, and the player will be prompted to respawn at predetermined checkpoint locations. Thus, to succeed in

	the game, players must skillfully evade these incoming projectiles, making dodging a crucial aspect of gameplay.
Chest Monster	Chest Monster features a formidable melee attack mechanic employed by certain enemies. When the enemy successfully lands a melee strike on the player, it has the effect of temporarily impeding the player's movement speed. Upon being struck by the enemy in close combat, players will experience a noticeable decrease in their movement speed for a brief duration. This strategic element adds an extra layer of challenge, requiring players to navigate encounters with these adversaries thoughtfully. It encourages players to be cautious and tactical in their approach to avoid getting hit and experiencing the associated movement penalty,
Zombie	Zombie enemy AI also implemented a melee attack feature, which shares similarities with the Chest Monster enemy but with a more severe consequence: upon being hit, the player faces instant demise, followed by a respawn at specific checkpoint locations. Therefore, player will also need to dodge from getting hit by the zombie enemy.
Mushroom	Mushroom enemy AI introduces acaptivating magic attack mechanism, which incorporates a knockback effect designed to disrupt the player's position by forcibly propelling them within a specified range. When subjected to the enemy's magic attack, players will experience a kinetic force that sends them hurtling away, a distance commensurate with the attack's characteristics.

Game Interactable Objects Design and Implementation

We have interactable objects such as checkpoints, obstacles, floating platforms, and finish point.

Checkpoints are triggers that mark the player's new respawn point and get destroyed when player walk towards and touches it. They are using the "Checkpoint" tag. The checkpoints can be place anywhere on the terrain, duplicate, and it can be customized so that when the player touches it, it will trigger some traps. For example, in level 1, there is a checkpoint which when the player touches it, it will enable the gravity of the floating giant balls and let them roll randomly so player need to avoid them to not get killed.

Obstacles are the objects that are scripted to keep moving to make the level harder. The example of the obstacles is pendulum that hanging a ball and keep swinging, the rotator that spins the sticks and player need to find timing to jump over it, and the moving walls that will move up and down and player can only go through it when it is downed.

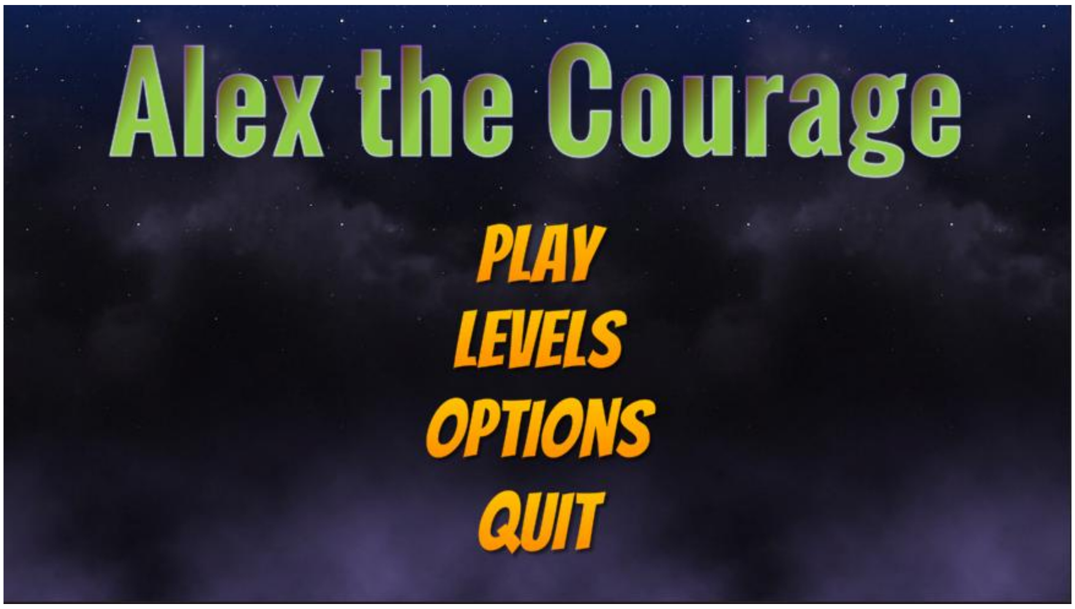
Floating platforms are only appeared in level 4, which it will move horizontally from one platform to the other platform repeatedly. Player needs to find timing to jump on it to be able to move to another platform. Fail to land on the floating platform will cause player to fall to death.

Finish points are a box collider that is set to trigger, so when the player touches it, it will trigger the win screen and the winning animation of the player and the princess.

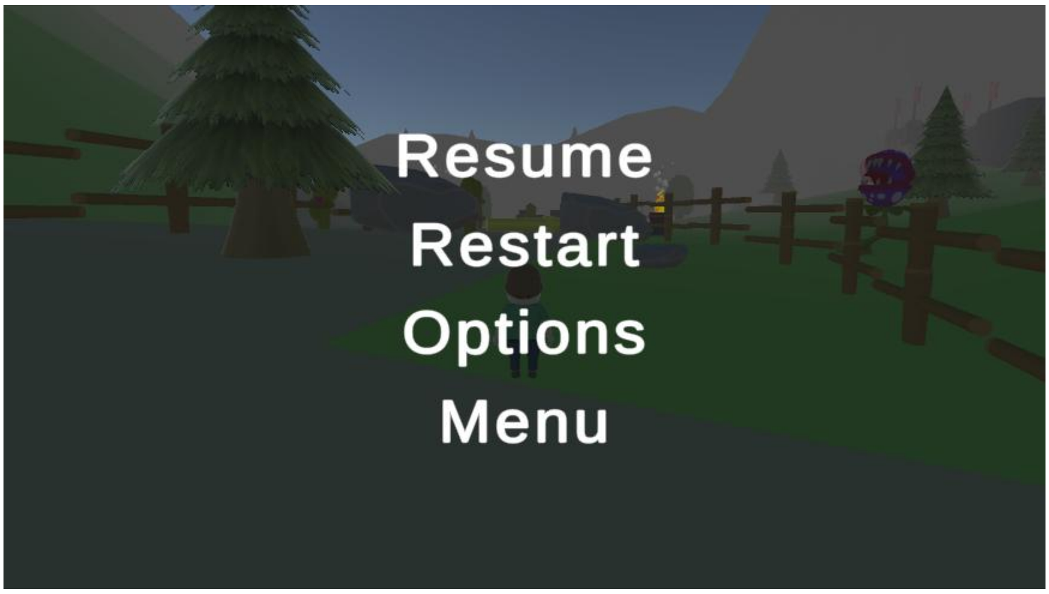
Game Interface Setup and Implementation

The camera is set to always follow the player at certain distance. When the player is in playing mode, the mouse cursor will lock on the game screen and will be set to invisible. When the game is paused, the cursor will reappear on the screen to let user interact with the UIs and when the game is resume, the cursor will lock on the screen again.

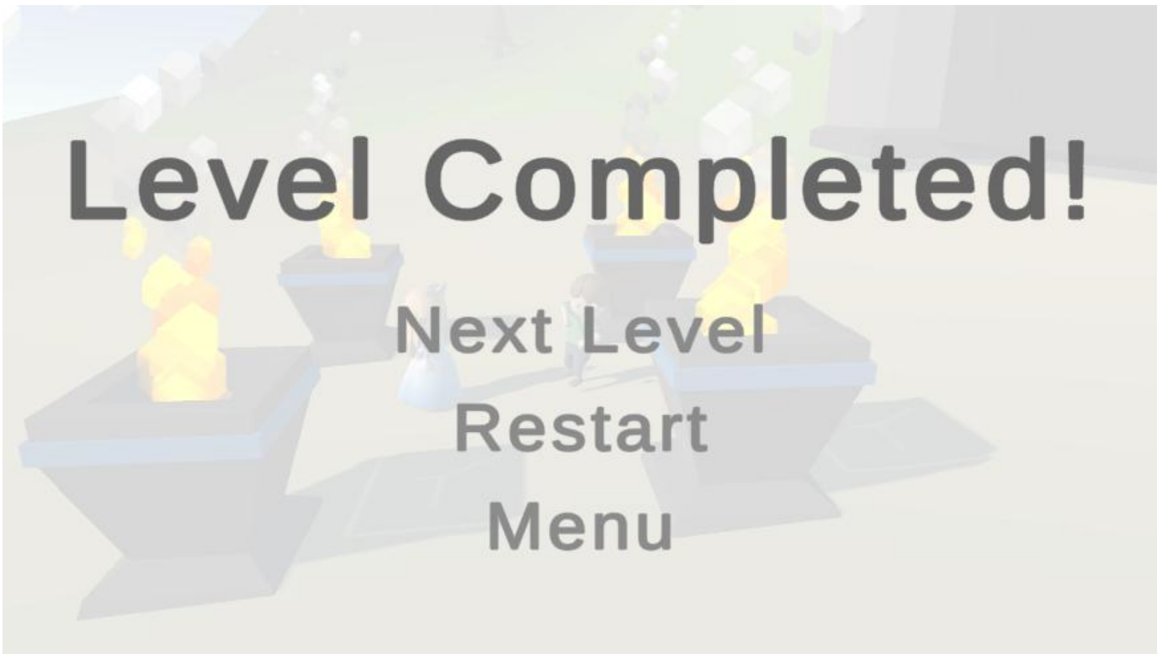
For the game GUI, we have start menu, pause menu, win screen, and options menu. All GUIs are made using canva. The start menu is the first thing player will see when launching the game. Player can choose “Play” to start from the first level, select levels, open the option menu, and quit the game in the start menu.



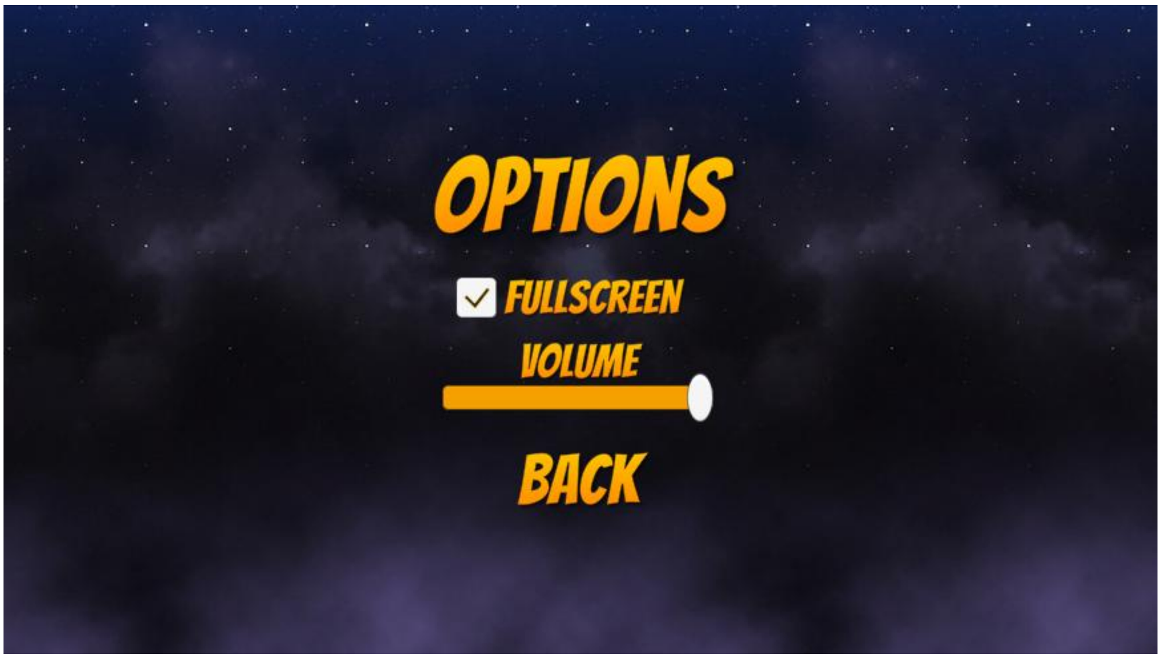
Pause menu will show up when the player pauses the game by pressing the ESC button. It has options to resume the game, restart the game, open the options menu, and return to main menu.



Win screen will appear when the player has reached the finish point. It has the options to go to the next level, restart the game, or go to the main menu.



The options menu can be open from the start menu and the pause menu. It has the options that can make the game go full screen and adjust the sound volume of the game.



Issues of the project and the development process

Introduction

Game development is a multidisciplinary process that encompasses a range of activities from coding and design to quality assurance. During the course of our project, we faced several challenges that affected both the performance and the development process. These challenges served as learning experiences that will guide future improvements.

1. Lack of Direction for the End User

The initial level design was broad and unguided, causing confusion among end users who were unsure of their objectives. This hindered the gameplay experience and end-user engagement. The lack of guidance can lead to a frustrating user experience, which could ultimately result in end users abandoning the game. By implementing in-game markers or a minimap will give end users a sense of direction. A brief tutorial or introductory cut-scene could also be helpful.

2. Camera Terrain Collision

The camera in certain sloped areas collided with the terrain, causing it to pass through the ground and affecting the visual experience of the game. This not only breaks immersion but also affects the player's ability to navigate through the game effectively. Implement collision detection algorithms to adapt the camera's position relative to the terrain. Techniques such as Raycasting could be useful in this regard.

3. Absence of Health System

The game lacks a health system, reducing the stakes and challenge. The player will die instantly when they touch or collide with the enemy. An absence of a health system can make gameplay less engaging and can trivialize combat or obstacle-avoidance mechanics. Introducing a health system with a UI display could add a layer of complexity and engagement to the gameplay.

4. Features Merge Conflicts

Merging conflicts in Unity files led to complications in the version control process. As we are using GitHub to collaborate with the development, when 2 people edit the same scene and commit their changes at the same time the scene will have error and can’t open. These conflicts disrupted the development flow and could lead to delays in the project timeline. The conflicts were resolved manually using text editors. For future projects, maybe a "lock" mechanism or branching strategy should be implemented to avoid such conflicts.

Conclusion

Throughout the development process of our Unity game, we have faced numerous challenges, particularly in the areas of performance and version control. These issues not only acted as bottlenecks in the development pipeline but also had the potential to affect the end-user experience. While some problems have been resolved, ongoing work is needed to improve the project further. The lessons learned from these challenges will serve as valuable insights for future game development projects.

Self-development report

Hon Wen Xuan

I am writing this self-development report as the leader of our game development group, where my role involves implementing audio, designing the game's GUI, integrating team members' work, and setting up a GitHub repository. In this report, I will reflect on my experiences, achievements, challenges, and the knowledge I've gained throughout the project.

Role and Responsibilities

Audio implementation

One of my primary responsibilities was to handle audio in the game. This included:

- Background Music: I implemented background music that sets the tone for the game. I researched and experimented with various tracks to find the perfect fit for our game's atmosphere. I integrated the background music into the DayNightCycle script so the different background music will play based on whether it is day or night in the game, and they can switch accordingly and seamlessly. I also made that so when the game is paused, the background music's volume will be lower, and the volume will set back to normal when the game is resumed.
- Sound Effects: I added sound effects for various in-game events such as player actions, enemy interactions, and environment interactions. This helped enhance the player's immersion in the game. I made an Audio Manger object attached with a script that can manage most of the sound effects in the game easily. I also set the enemies and obstacles sound into 3D sound.
- Sound Volume Control: I connect the audio mixer to the options menu so player can control the sound volume while in game. I make sure that all of the audio sources' output are set to that audio mixer so that all of the audios can be controlled in the options menu.

GUI Design

I was in charge of designing the game's graphical user interface (GUI) to ensure smooth navigation and control. This included:

- Main Menu: I created an intuitive main menu that welcomes players and provides options to start the game, select levels, access settings, or exit.
- Pause Menu: I designed a pause menu that allows players to pause the game, adjust settings, restart the game, or return to the main menu.
- Options Menu: I incorporated an options menu in main menu and pause menu where players can customize audio settings and screen settings.
- Win Screen: Upon completing the game, I created a win screen to celebrate the player's success. Player can choose to play the next level, restart the level, or return to the main menu.

Integration

As the leader, I was responsible for integrating various components developed by team members, including:

- Animation: I collaborated with the Harry to ensure smooth transitions and interactions between characters and objects.
- Enemy AI: I coordinated with the AI team to integrate enemy behaviours, ensuring they provided engaging challenges for players. I also need to know how they works to implement the sound effects.
- Player Control: I worked closely with Harry to refine player movements and interactions. I need to modify the player movement script to integrate the sound effects on player movements.
- Terrain: I learned how the terrain generation system worked, allowing me to incorporate it seamlessly into the game world, and check whether the environments, enemies, player, obstacles, and waypoints can work on it. I made spawn point, checkpoints, and finish points so that player can respawn accordingly and win the game.

- Obstacles: I integrated some obstacles with player and checkpoints so when the player reach the checkpoints, it triggers the traps. I also make help make the obstacles able to knockback and kill player.
- Prefabs and organization: I made prefabs so teammates can design the levels easily by just dragging the prefabs into their levels. I organized game objects and assets folder so teammates can find thing they want easily.

GitHub Repository Setup

To facilitate collaborative development and version control, I set up a GitHub repository for our project. This allowed us to:

- Share code and assets efficiently.
- Track changes and resolve conflicts.
- Maintain a structured project history.

I also learn what to do to lessen the conflict between each commit, such as avoiding changing the same scene at the same time.

Challenges and Learning

Throughout the project, I faced several challenges and learned valuable lessons:

- Time Management: Managing both leadership responsibilities and implementation tasks required effective time management. I learned to prioritize and allocate time wisely. Because the resources we found online is not the same as our project, we need to spend very long time to figure out on how to integrate them into our project.
- Communication: Clear communication was crucial. I improved my communication skills by regularly discussing progress with team members, addressing concerns, and providing feedback. There is also challenges in communication because when I faced a problem and need to ask my team members on how to solve it, I will need to wait a while for them to response and it makes the whole development process more time consuming, which is understandable since everyone as well as myself is not always working

on the game project all the time because we have other things to do in out life.

- Technical Knowledge: I gained a deeper understanding of audio implementation, GUI design, and integration. Learning about animations, AI, and terrain systems broadened my knowledge base. The most challenging part is to integrate scripts and components of these features.
- GitHub: Setting up and managing a GitHub repository was initially challenging but became a valuable tool for collaboration. I learned about branching, merging, and resolving conflicts. During the commitment to our GitHub repository, we faced several challenges such as merge conflicts, which we need to check if the merging process will not erase our changes. Some merge conflicts even seriously affect our project, which makes one of our scenes not able to open and we have to delete it.

Leading the game development project as an audio and GUI specialist was a rewarding experience. It allowed me to explore new skills, collaborate effectively with team members, and contribute to the creation of an engaging game. The challenges I faced strengthened my problem-solving abilities, and the knowledge gained has broadened my skill set for future projects.

As I reflect on this self-development report, I am proud of the accomplishments achieved in our game project and look forward to applying these skills and experiences to future endeavours in game development.

Cheah Wen Huey

In this project, my primary technical responsibility lies in the development of enemy AI systems, and I'm collaborating closely with my teammate, Yi Wei. Specifically, my focus is on creating AI enemies with melee attack capabilities, namely the Chest Monster and Mushroom enemies. While these two enemy types share similar overall behavior patterns, a key distinction lies in the implementation of melee attack. For instance, the Chest Monster will feature a distinctive "bite" animation when players enter its melee attack range, enhancing the immersion and engagement of the game. Upon attacking, the Chest Moster will cause a slow down effect on the player’s movement speed. However, for the Mushroom, it implements the knockback script, created by Wen Xuan, where player will be knock back to a certain distance when it get hits by the enemy.

Moreover, I contributed to the project by configuring the projectile mechanics, ensuring that they functioned as intended. When colliding with the player, these projectiles resulted in player character demise, while those that missed their mark disappeared from the game map after a brief duration.

Additionally, my role extends to the creation of waypoints, specifically the central point within the designated patrol range. This addition is instrumental in maintaining game balance by constraining enemy movements to a defined patrol area rather than allowing them to roam freely throughout the entire game map. This strategic decision ensures that the gameplay experience remains challenging and engaging, as enemies adhere to their designated patrol zones.

Other than that, in the context of the project report, I am also tasked with crafting the overall class diagram that provides an overarching view of the project's architecture. Whereas, in the Game AI Overview section, I collaborated with Yi Wei once again to provide detailed insights into the specific enemy AI implementations we've each worked on.

Throughout the development process, I encountered and overcame several challenges. One of the initial hurdles was animation integration, specifically related to the melee attack. A critical breakthrough was achieved by reconfiguring the Animator setup, resolving issues where the enemy stopped mid-attack sequence.

Besides that, I wrestled with the challenge of restricting enemy patrol ranges effectively. After exploring multiple approaches, the implementation of waypoints as central patrol points emerged as the optimal solution, ensuring enemies remained within their designated zones.

In essence, my contributions to this project spanned technical development, problem-solving, and documentation. The challenges encountered during development served as valuable learning experiences, enhancing our problem-solving skills and teamwork. Together, our efforts have culminated in an engaging and immersive gaming experience.

Harry Liow Ming Heng

As the primary developer responsible for player and enemy movement and animations, my role in this project was multi-faceted. I tackled a wide range of development tasks, from complex animations to physics-based interactive elements. This report explains the problems I met, how I solved them, what I did, and what I learned.

Player and Enemy Animation

1. Simple Movement of Character: Implemented basic movements, including walking, running, and idling.
2. Third Person Camera: Integrated a third-person camera that follows the player, providing an over-the-shoulder perspective.
3. Roll and Jump: Added roll and jump animations to the character for enhanced gameplay.
4. Ragdoll Mechanics: Incorporated ragdoll physics for both the player and enemy, enhancing realism.
5. Wake Up Animation: The enemy will disable the ragdoll, then come with a wake-up animation to continue patrolling or chasing player.
6. Knockback Effect: Created a knockback animation and effect when characters are hit by obstacles or enemies. It also added with the falling effect if the knockback effect triggered.
7. Slow Effect: Introduced a slowing effect when the player is attacked by an enemy.
8. Enemy State Transition: Designed animations for enemies transitioning from ragdoll state to a waking state.
9. NPC Victory Animation: Created a victory celebration animation that triggers upon successful completion of a level by the player.

Trap Logic and Level Design

1. Moving/Floating Platform: Implemented platforms that move along predetermined paths. Players are able to stand and move on the platform.
2. Ball Spawner: Designed a mechanism to randomly spawn balls at specified locations. Able to trigger ragdoll when player collides with the ball.

Issues Faced and Solutions Implemented

1. Falling Platform Behavior

Implementing the falling platform mechanics was a complex challenge, considering it had to fall only when the player stood on it for a specific duration.I have implemented Unity's OnControllerColliderHit method to detect when the player interacts with the platform. A Coroutine was set up to toggle the isKinematic property to false after a delay, allowing the platform to fall so far this fix the issues. But sometimes the player can stand on it but doesn't move with it, the reason is player model's children didn't inherit under the platform. The solution is I must remove the script component that applied on the platform and applied it again and it will solve the issue.

2. Missing Rigidbody Component

A runtime exception occurred because a GameObject was missing a Rigidbody, causing disruptions in the workflow. A Rigidbody component was attached to the GameObject, enabling it to interact within Unity's physics system, thus solving the problem.

3. Merge Conflicts in Unity Scenes

Version control complications arose, leading to merge conflicts in Unity's scene files. I manually resolved the conflicts by opening the .unity file in a text editor, streamlining the collaboration process and enabling the project to compile. These have headche me so much.

4. Enemy not colliding with player

When the enemy found and attacked the player it couldn't trigger the ragdoll of player. I have used the OnTriggerEnter() on the enemy script and OnControllerColliderHit() on the player, although it still lacks on what we expect, and these could only trigger the ragdoll when enemy's body collided with the player.

Conclusion

The journey of this project has been filled with learning experiences, challenges, and accomplishments. The scope of tasks I undertook has not only allowed me to contribute significantly to the game's features but also to deepen my understanding of various aspects of game development. Overcoming each obstacle, whether it was a technical issue or a design challenge, required research, testing, and iterations, all of which have contributed to my growth as a developer.

Liang Yi Wei

In this project, I worked with another member, Wen Huey, where both of us are responsible for designing and creating enemy AI that can enhance player experience by providing engaging, challenging, and dynamic opponents in our game.

As a beginner, I first conducted some research to learn about the fundamental concepts and characteristics of enemy AI in game development. Some of the basic features are state-based behavior, perception, pathfinding, and combat behaviors. Enemy AI with various states such as idle, patrolling, chasing, attack, and return were created using Unity's NavMeshAgent. The navigation mesh agent component is attached to an enemy's game object that allows it to navigate the scene using NavMesh Baking. Next, enemy AI can perceive player by defining two float variables: sight range and attack range which acts as a trigger for the enemy's state-based behavior. Tags and Layers were set up to enable enemy AI in identifying player's game object. Prefabs of different types of enemy

AI and game objects such as projectiles were created for reusability and consistency.

Throughout this project, one of the challenges that I encountered is the movement of enemy AI. The enemy AI tends to navigate the scene in a stationary state, and this was resolved by managing the state transitions in Animator Controller where the enemy will switch from ‘Patrol’ state to ‘Chase’ state as soon as it detects a player. Another issue that I dealt with is obstacle avoidance of enemy AI during patrolling. However, this can be tackled by using the NavMeshAgent component that allows configurations of settings for obstacle avoidance such as ‘radius’ that sets the radius of enemy’s collision capsule, ‘height’ that sets the height of the enemy capsule, avoidance priority and quality that determines the prioritisation and precision in avoiding obstacles.

In summary, I am committed to working on in-game Enemy AI creation and design. I have learned to create Enemey AI systems in Unity and it was a valuable experience in this course. For better time management, me and another group member, Wen Huey, have divided our stasks and a deadline was always set to ensure that we accomplish the delegated tasks on time. Lastly, the presence of teamwork, whether in contributing ideas and thoughts or solving encountered problems together, is the key element in attaining our project goals.

Ong Ying Shuang

During the development of our game, I took on the responsibility of learning and implementing procedural terrain generation. I began with extensive research on procedural terrain generation techniques. I studied various algorithms, including Perlin noise, simplex noise, and Fractal Brownian Motion. I collaborated closely with my teammates to understand the specific requirements of "Alex the Courage."

This included the need for dynamically changing landscapes across different maps and levels, each with its unique challenges.

Procedural terrain generation is a complex subject, and implementing it requires a solid understanding of mathematics and algorithms. I had to overcome the challenge of grasping these concepts, especially since they were new to me. Ensuring that the generated terrain was efficient and didn't impact the game's performance was crucial. I had to optimize the algorithms to generate terrain quickly while maintaining a high level of detail.

Procedural terrain generation provided great creative freedom for level designing. We could define high-level parameters, such as terrain type and difficulty, and let the system handle the details.

Learning and implementing procedural terrain generation in the development of "Alex the Courage" was challenging but seeing how much scripting can do in a game development process actually made me feel excited and even though our game wasn’t perfect I had a lot of fun throughout the course. Procedural terrain generation not only expanded my knowledge and skill set but also played a pivotal role in shaping the game's identity. The dynamically changing landscapes added depth and excitement to the gameplay, making it a more immersive and enjoyable experience for players.

Moving forward, I plan to continue exploring advanced procedural generation techniques and their applications in game development. I believe that this knowledge will be invaluable in creating engaging and dynamic gaming experiences in the future.

Assets

Characters

https://assetstore.unity.com/packages/3d/characters/newbie-friends-208112#description

Enemies

https://assetstore.unity.com/packages/3d/characters/creatures/rpg-monster-partners-pbr-polyart168251

https://assetstore.unity.com/packages/3d/characters/creatures/rpg-monster-buddy-pbr-polyart253961#content

https://assetstore.unity.com/packages/3d/characters/creatures/rpg-monster-duo-pbr-polyart157762

https://assetstore.unity.com/packages/3d/characters/creatures/free-animated-angry-log-revenge-ofthe-tree-150947#content

https://assetstore.unity.com/packages/3d/characters/creatures/character-cactus-32933

Environments

1) Forgotten Island

https://assetstore.unity.com/packages/3d/environments/various-environment-islands-lite254756#description

https://assetstore.unity.com/packages/3d/environments/stylized-pirate-asset-set-free-pack-195470

2) Winter Forest

https://assetstore.unity.com/packages/3d/environments/low-poly-nature-pack-rg236313#description

3) Ancient Village

https://assetstore.unity.com/packages/3d/environments/fantasy/polyart-ancient-village-pack166022

4) Uhara Desert

https://assetstore.unity.com/packages/3d/environments/landscapes/polydesert-107196#description

5) Forbidden Dungeon

https://assetstore.unity.com/packages/3d/environments/dungeons/ultimate-low-poly-dungeon143535

https://assetstore.unity.com/packages/3d/environments/dungeons/blue-dungeon-106912

6)Common Use

https://assetstore.unity.com/packages/3d/environments/little-low-poly-world-lite-srp-urp-119111