FINAL INTERNSHIP REPORT BY STUDENT

Student Name: Ho Li Lian

Admin No: 212174D

Diploma: Game Development & Technology

Period: 1

Module: Internship Programme (12 weeks)

Company Name: CREEK & RIVER Co., Ltd.
(OX Engineer Studio)

ACKNOWLEDGEMENTS		
1	**Mr. Chua Han Xiang** (School Industry Mentor)	**Senior Lecturer** School of Design & Media
2	**Ms. Lee Katrina Johanne Pe** (Industry Internship Mentor)	**Engineer** Division 1 OX Engineer Studio
3	**Mr. Yoshihiro Machida** (Industry Internship Supervisor)	**Manager** OX Engineer Studio
4	**Mr. Ryotaro Haga** (Industry Internship Supervisor)	**Leader** Division 1 OX Engineer Studio
5	**Mr. Shunta Nomura** (Industry Internship Supervisor)	**Global Promoter** OX Engineer Studio & Coyote 3DCG Studio
6	**Ms. Mio Hirata** (Industry Internship Supervisor)	**Agent** Division 1, Section 1 Digital Contents Group 1

CONTENTS

1 ORGANISATION BACKGROUND

1.1 Organisation Information

Name: クリーク・アンド・リバー社 CREEK & RIVER Co. Ltd.

Address:	SHINTORA-DORI CORE., 4-1-1, Shimbashi, Minato-ku, Tokyo 〒105-0004
Country:	Japan

1.2 Nature of Business

CREEK & RIVER Co., Ltd. is an agency which matches freelancers or short-term self-employed individuals with firms seeking to outsource certain jobs. They provide an environment where professionals can maximise their potential through job provision, project composition, intellectual property monetization, and educational opportunities. It operates in 18 different fields, such as the game programming, and advertising industries. CREEK & RIVER Co., Ltd. also offers a variety of courses, of which includes website design, and digital content creation.

1.3 Platform and Technology

Platform	Windows 11 Android
Hardware	Windows Desktop Android Test Phones
Software: Development Tools	Unity 2021.3.18f1 DxLib 3.24b PlantUML Microsoft Visual Studio 2022 Microsoft Visual Studio Code 1.76.2
Software: Version Control	GitLab Git Bash 2.31.7
Software: Communications	Slack Microsoft Teams
Software: External Tools	GIMP Blender Spine

2 INTERNSHIP BACKGROUND

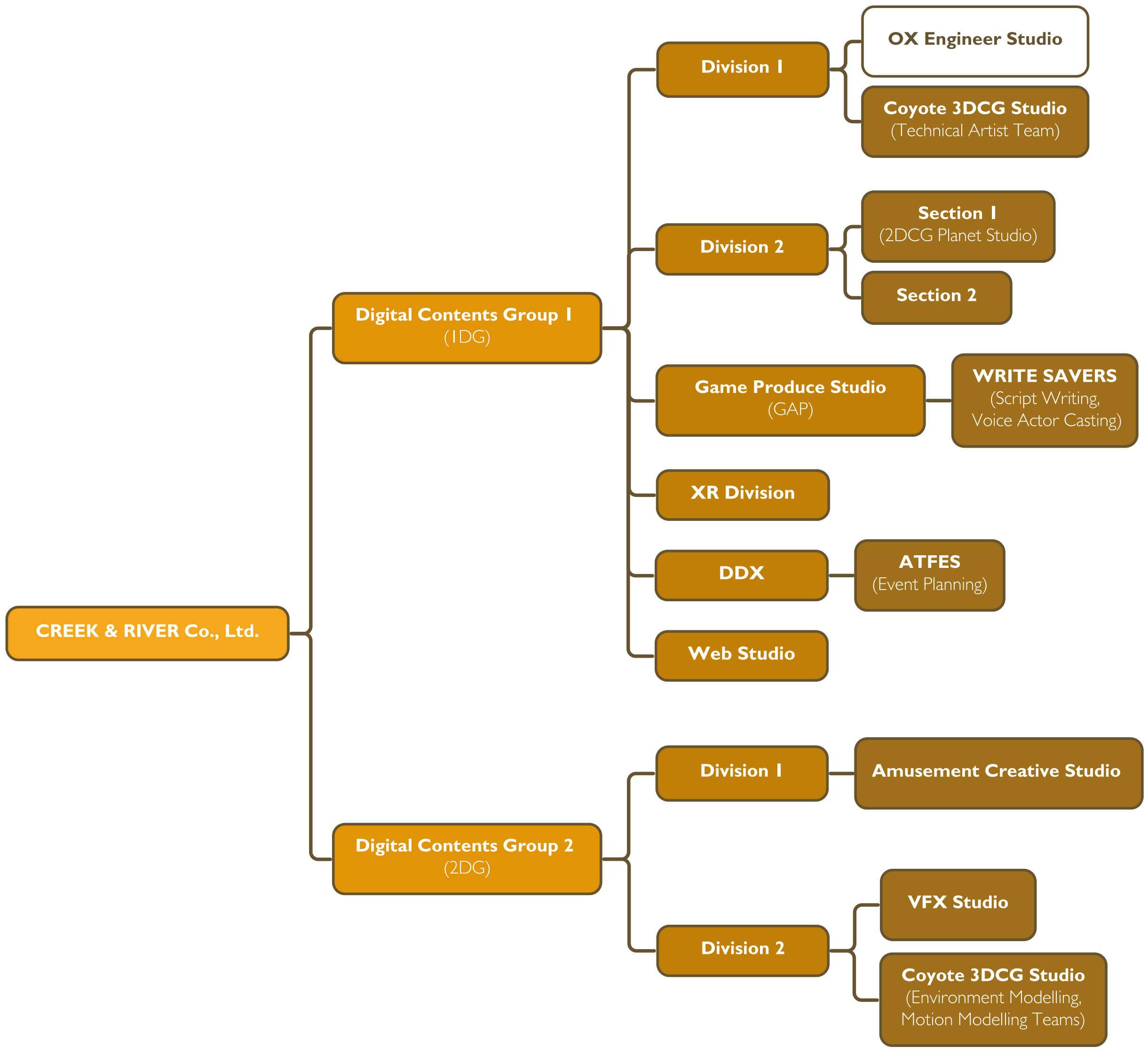
2.1 Department Description

OX Engineer Studio is a subsidiary of CREEK & RIVER Co., Ltd. They specialise in the development of consumer games, PC games, and social games across a diverse range of genres, from action-packed titles to immersive online experiences. Additionally, they have actively contributed to the creation of AAA games and undertaken large-scale projects. Notable examples of their portfolio include "PSO2 New Genesis" and "SHAMAN King ふんばりクロニクル". The developers of the studio use game engines such as Unreal Engine 4 and Unity. One of OX Engineering Studio's goal is to improve the lifetime value of creators. To this end, the company has implemented a variety of training programmes designed to foster the rapid growth of their developers.

2.2 Department Role

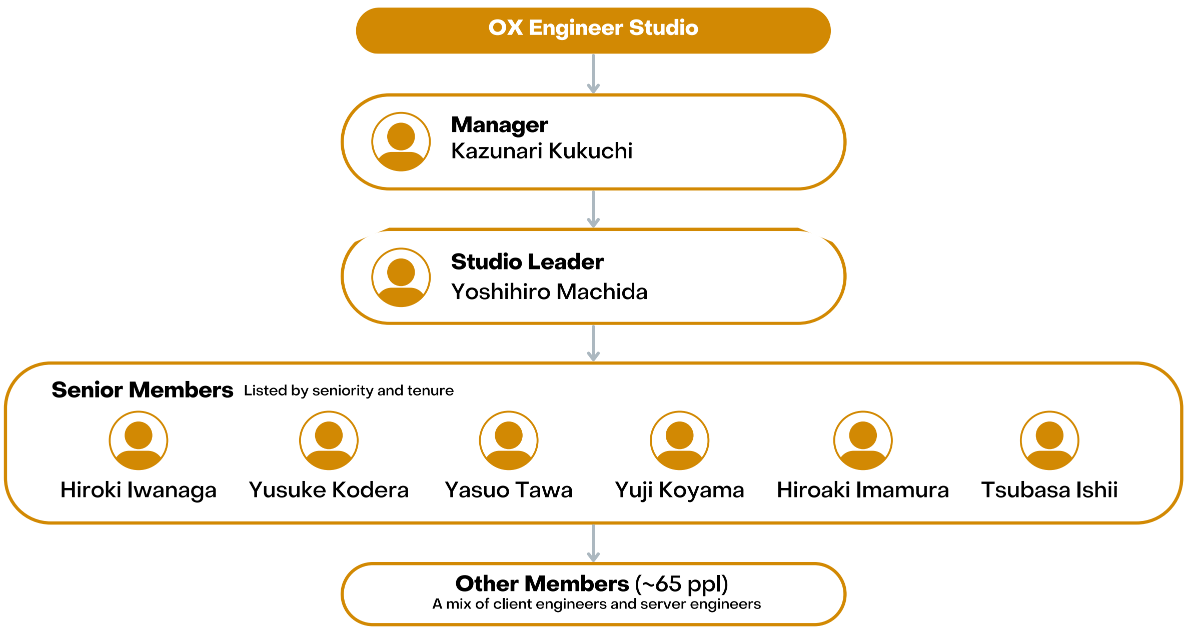
The role of OX Engineer Studio is to realise and maximise the careers of creators in the programming field, while also helping to grow their connections with clients in the game development industry. An integral part of their mission is to elevate the overall standards of the industry by nurturing and training their team members, facilitating the exchange of valuable know-how, and offering various opportunities for career advancement.

2.3 CREEK & RIVER Co. Ltd. Organisation Chart

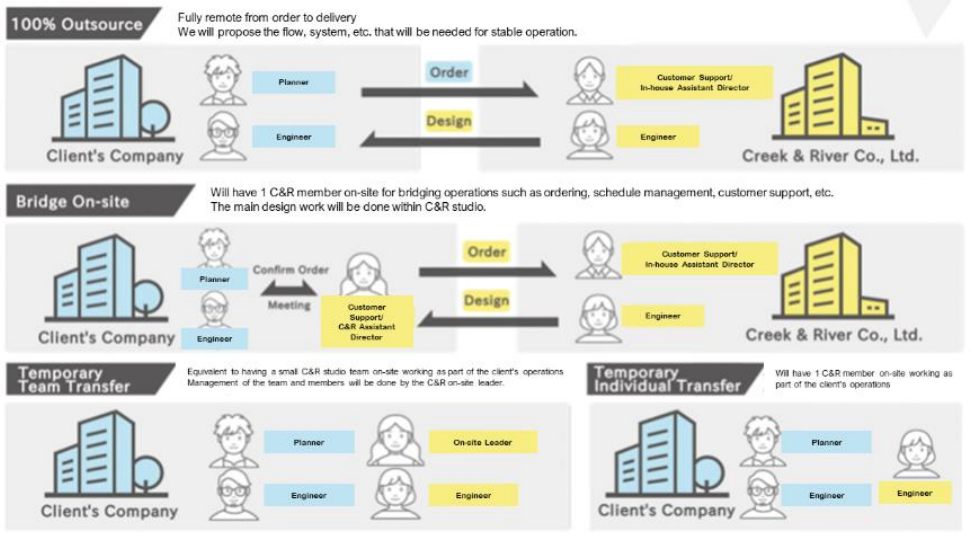


CREEK & RIVER Co., Ltd.
Digital Contents Group 1 (1DG)
Division 1
OX Engineer Studio
Coyote 3DCG Studio (Technical Artist Team)
Division 2
Section 1 (2DCG Planet Studio)
Section 2
Game Produce Studio (GAP)
WRITE SAVERS (Script Writing, Voice Actor Casting)
XR Division
DDX
ATFES (Event Planning)
Web Studio
Digital Contents Group 2 (2DG)
Division 1
Amusement Creative Studio
Division 2
VFX Studio
Coyote 3DCG Studio (Environment Modelling, Motion Modelling Teams)

2.4 Department Organisation Chart



The other 65 engineers work with CREEK & RIVER's clients or partners, following any one of the four work setups described below:


100% Outsource
Fully remote from order to delivery
We will propose the flow, system, etc. that will be needed for stable operation.
Planner
Engineer
Client's Company
Order
Design
Customer Support/ In-house Assistant Director
Engineer
Creek & River Co., Ltd.
Bridge On-site
Will have 1 C&R member on-site for bridging operations such as ordering, schedule management, customer support, etc.
The main design work will be done within C&R studio.
Planner
Engineer
Client's Company
Confirm Order
Meeting
Customer Support/ C&R Assistant Director
Order
Design
Customer Support/ In-house Assistant Director
Engineer
Creek & River Co., Ltd.
Temporary Team Transfer
Equivalent to having a small C&R studio team on-site working as part of the client's operations
Management of the team and members will be done by the C&R on-site leader.
Planner
Engineer
On-site Leader
Engineer
Client's Company
Temporary Individual Transfer
Will have 1 C&R member on-site working as part of the client's operations
Planner
Engineer
Engineer
Client's Company

2.5 Name and Designation of Industry Internship Mentor

Ms. Lee Katrina Johanne Pe, Engineer of Division 1 in OX Engineer Studio

2.6 Name and Designation of Industry Supervisor

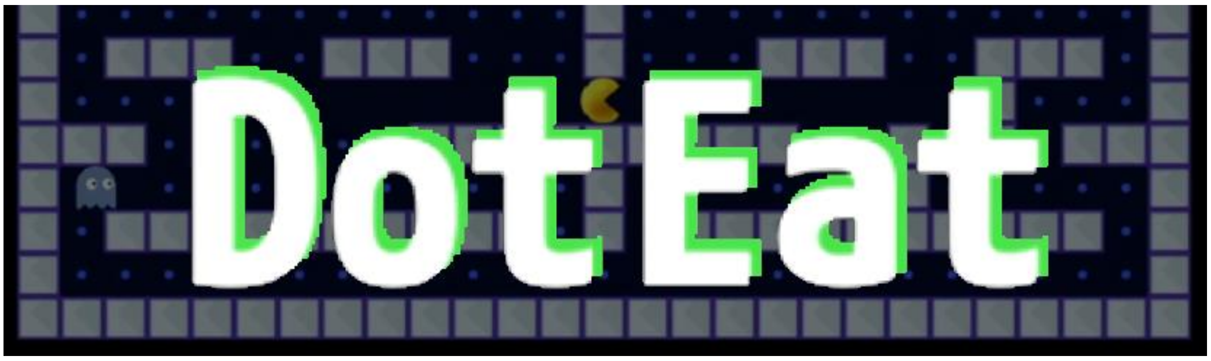
Mr. Ryotaro Haga, Leader of Division 1 in OX Engineer Studio

2.7 Name and Designation of School Industry Mentor

Mr. Chua Han Xiang, Senior Lecturer from School of Design and Media in Nanyang Polytechnic

2.8 Student's Role during the Internship

My role during the internship was to recreate and improve upon existing client projects. Given the original project and its framework, I was tasked with improving the code architecture and creating my own rendition of each project from scratch. I was also given the freedom to revamp the theme and add enhancements to both the aesthetics and gameplay, so long as the core mechanics remained the same.



3 DOTEAT: PROJECT BACKGROUND

3.1 Project Name

DotEat

3.2 Project Background/History

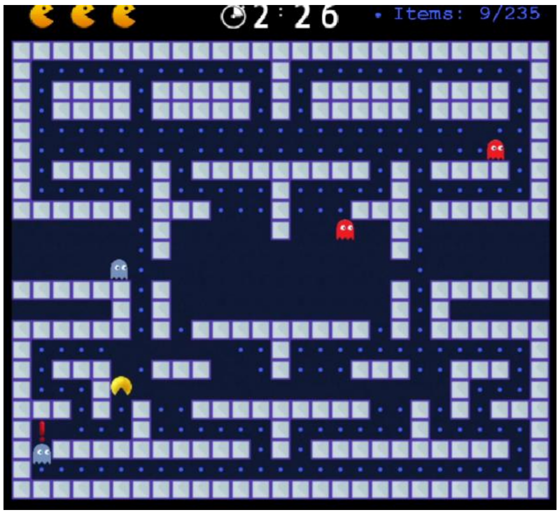
DotEat is a project excerpted from the curriculum of Game Engineer Academy, an in-house training programme produced by CREEK and RIVER. The programme trains people completely new to game development and equips them with crucial skills like C++ and the usage of Unreal Engine 4 over the course of five months. An added bonus is that the course is offered completely free of charge, with interested applicants only needing to pass an interview to enroll.
DotEat's syllabus was translated into English and then provided to us via pages hosted on GitLab. We were then tasked with following the instructions, which progresses from being quite guided with code already provided, to eventually letting us figure things out on our own with small hints of what to implement.

3.3 Project Scope

- Implement coding conventions from Google's C++ Style Guide
- Implement good game architecture
- Implement design patterns like event listeners, and state machines
- Implement version control with GitLab using the terminal
- Build upon existing game by adding additional feature using excess time

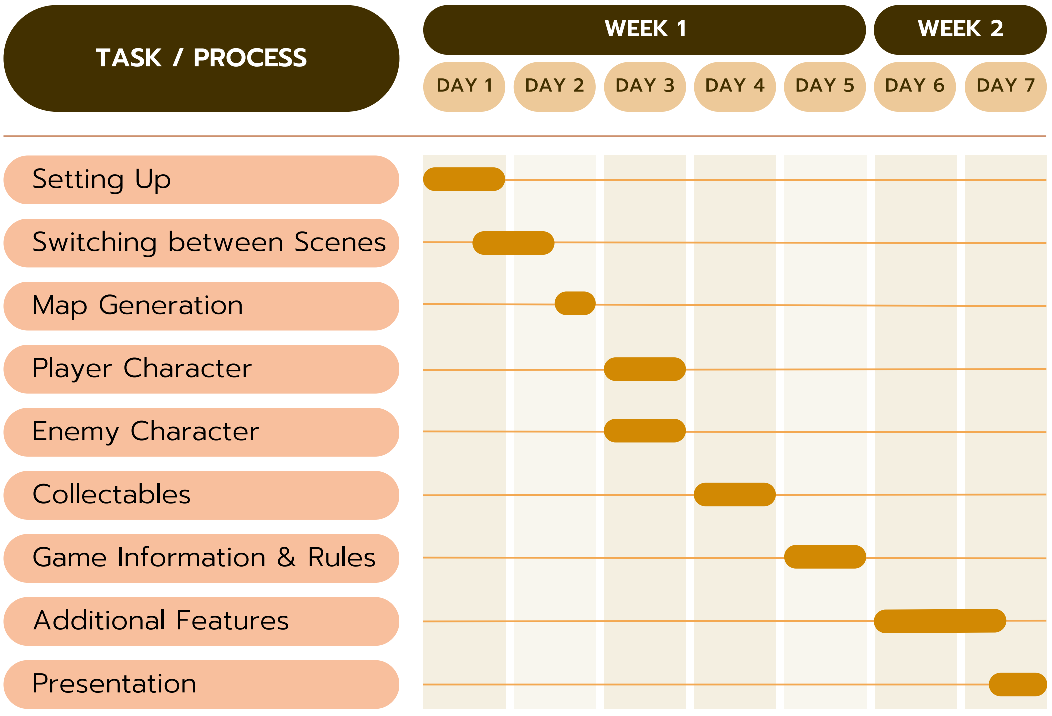
3.4 Project Description

DotEat is a simple arcade game similar to PacMan. The player controls a circle with a mouth through an enclosed maze using the up, down, left, and right keys. The objective of the game is to eat all of the dots placed in the maze while avoiding ghosts. Colliding with a ghost costs one life and causes the player to respawn at the start position. Once all three lives are used up, the player loses. The game is developed in C++ using a Japanese game library called DxLib.



4 DOTEAT: PROJECT SCHEDULE

4.1 Task Schedule



DotEat Gantt Chart

4.2 Task List

Week 1

- Setting Up
- Switching between scenes
- Map generation
- Player character
- Enemy characters
- Collectables

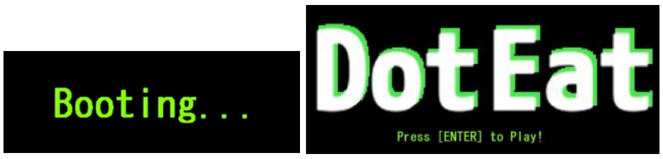
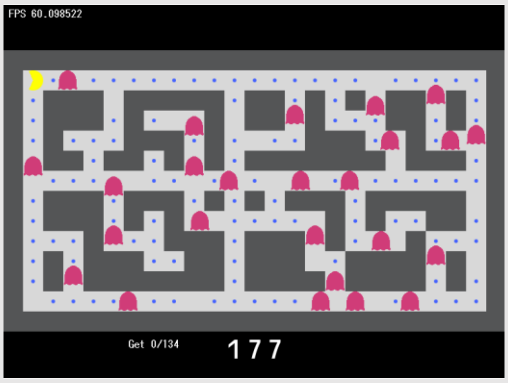
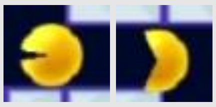
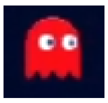
Week 2

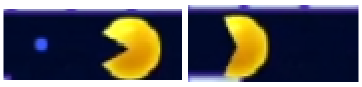
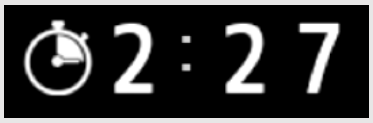
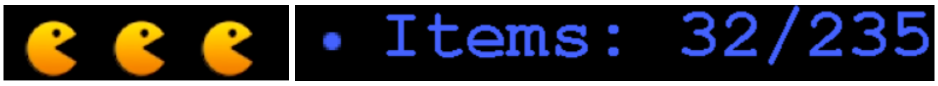
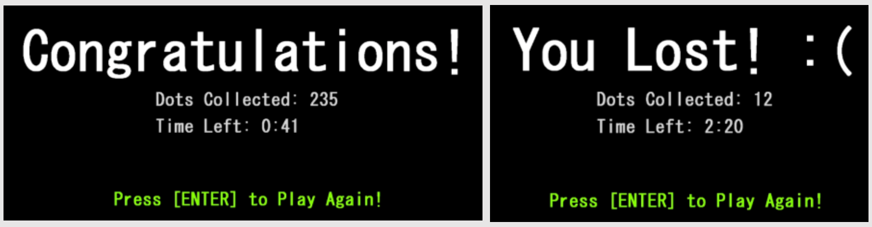
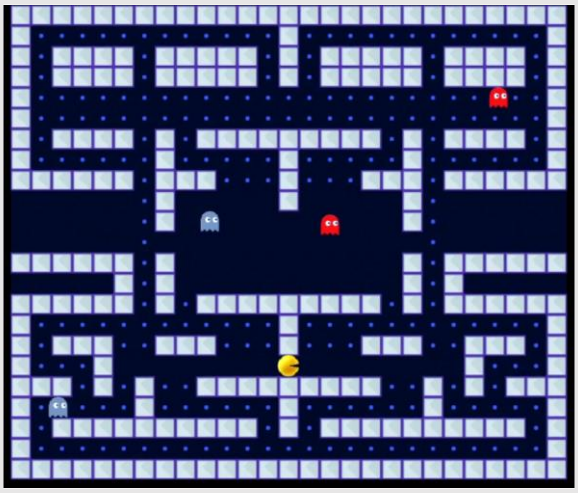
- Game information and win/lose states
- Additional features
- Presentation

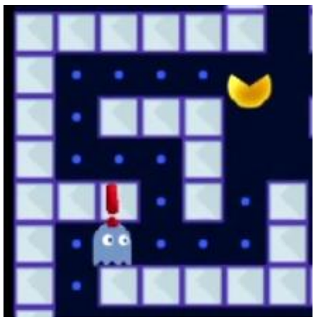
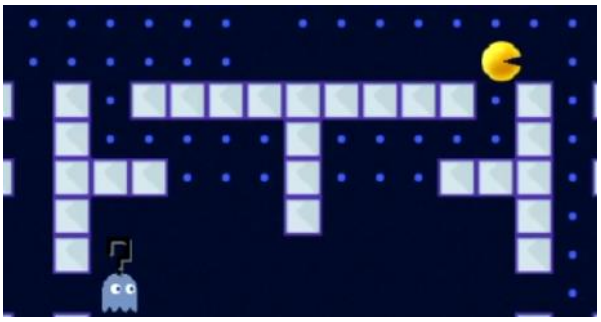
5 DOTEAT: PROJECT ACCOMPLISHMENTS

5.1 Accomplishments

Setting Up	Set up development environment, installing required applications like Visual Studio 2022 and GitBash
	Setting up main message loop for DxLib
	Implementing a Task Manager using an Observer Pattern
	Creation of a Controller class that allows for input keys to be easily remapped
	Implementation of event listeners that raise an event when an input key is pressed
	Push base project to company's repository in GitLab using commands executed through GitBash
Scenes	Implementation of base Scene class with elements of a state machine

	Implementation of SceneChanger class that releases the current level and loads the next level. Executes different functions according to the current phase. For example, if the class is in the Initialization phase, it will initialize the current level, then switch to a Processing phase.
	Creation of Booting, Title, and Gameplay scenes. The Booting scene launches for 3 seconds before automatically switching to the Title scene, where pressing [ENTER] would switch the scene to the Gameplay scene 
Map	Addition of different map tile types (Wall, Road, Dots, Player Spawn, Enemy Spawn)
	Designing and creation of map using a two-dimensional array containing the different tile types
	Loading and displaying of map based on array 
	Checking of possible walkable tiles
Player Character	Creation of base Character class that manages the display of the character sprites and storing the current character's type and position in the map
	Creation of player class that inherits from Character class and extended to detect key presses of movement keys
	Movement of player character based on key presses
	Collision detection based on map
	Implementation of player's open and closed mouth animation. Done essentially through calculations of a "pie-chart" where the rendered circle sprite is partly cut off based on the given percentage, giving the illusion of a mouth. 
	Implementation of smooth movement by linearly interpolating the change in positions when moving from one grid to another
Enemy Characters	Creation of base enemy class that inherits from Character class
	Creation of EnemyA that inherits from base enemy, and extended to include a state machine 
	Implementation of EnemyA's AI. Every second, it checks for a walkable tiles near it in the map, then randomly picks one from the available tiles and moves to it.
	Implementation of player's death using a hit point and attack point system. When the player collides with a ghost and vice versa, the player reduces their hit points

	by the ghost's attack points (currently both are set to 1), and respawns at their spawn point.
Collectables	Creation of Dot item class that inherits from Character class
	On collision with the player, the dot gets gets replaced with a normal Road tile on the map 
Game Information & Rules	Implemented dynamic scaling of interface and map based on the current window size, which scales accordingly to the displaying monitor's size
	Creation of Game Mode class using a Singleton design pattern. This class stores the time left, the amount of dots collected so far, and the total amount of enemies and dots in the level
	Implemented a countdown based on information from the Game Mode class. Once the timer reaches 0, the game automatically loses.  2:27
	Implementation of the main win/lose states. When the player collects all of the dots, the game wins. When the player uses up all of their lives, the game loses.  Items: 32/235
	Creation of a Results scene. Winning or losing the game would transition to this scene, which shows the relevant texts based on whether the player won or lost, the total dots collected, and the time remaining.  Congratulations! Dots Collected: 235 Time Left: 0:41 Press [ENTER] to Play Again! You Lost! :(Dots Collected: 12 Time Left: 2:20 Press [ENTER] to Play Again!
Additional Features	Improved the user interface to be more aesthetically pleasing compared to the examples given in the tutorial
	Overhauled the map and expanded on it to be 1-to-1 with the original PacMan game 
	Creation of a second Enemy type, B. This enemy is more intelligent than the previous one. It utilizes a state machine to switch between an Aggravated state and a Patrol state. Under its patrol state, it would behave like Enemy A. However, once the player gets close to the enemy, an exclamation mark (!) would show up above

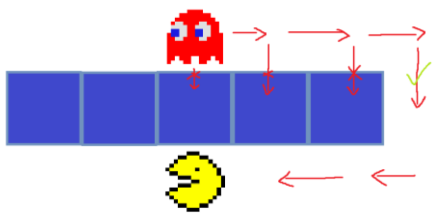
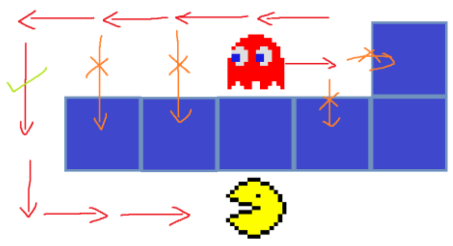
	its head and it would switch states to Aggravated. Under this state, the enemy would use psuedo-pathfinding to chase the player, and its movement speed increases as well. However, once the player goes out of range, the enemy would show a question mark (?) above its head, and goes back to Patrol.  
Presentation	Presented the game along with what was changed from the original tutorial to our supervisors and colleagues

5.2 Concepts Learnt

- Usage of DxLib
- Practicing good coding conventions
- Class Inheritance
- State Machines
- Event Listeners
- Singletons
- Virtual Controllers
- Version Control (Usage of GitLab, terminal commands in GitBash, and naming conventions)
- Forward Declaration
- NPC Artificial Intelligence
- Presenting in two languages

5.3 Challenges Faced

Challenge	Solution
I was unfamiliar with using the new library, DxLib. Additionally, it is Japanese based, so the documentation and help forums were also in Japanese. As such, it was sometimes quite difficult to figure out what some functions in the library do.	Even though this was a completely new library, I tapped into what I had learned in my studies about OpenGL, a graphics library, and applied it. Through this, I realized that there are quite a few similar functions and methods between the two libraries, albeit the syntax was different. For example, both DxLib and OpenGL required the user to draw a window to begin rendering things to the screen. Additionally, the way images are rendered are through graphic handles that store a relative path to the image asset that you wish to be rendered. Through this method, I was able to pick up the library quite fast and learn the various different functions by essentially interpreting them in OpenGL terms. To add on, when searching for help about the library, I translated what I wanted to find into Japanese before searching it on Google. This helped me find a lot of useful forum posts that detail how to solve my issues, and I was able to navigate the forums even though I was not too familiar with terminologies in Japanese.
It was difficult to implement established pathfinding algorithms due to the current game's	I designed my own simple algorithm that works within the constraints of the code architecture. While it is not the most optimized algorithm, I believe that this addition makes the enemy look much smarter. The enemy would try to move towards the player. However, if the next desired tile is a wall, it would move in the opposite axis while constantly checking if it could continue moving in the desired direction.

implementation of the map.	For a clearer illustration of this, I would provide an example case where the enemy wishes to go down towards the player but is blocked by a wall:  In this case, the player would first try to move right, then check if it can move down, if it cannot, it will move right again. It will repeat until it can successfully move down. But what if there is another wall at the right? Then my algorithm would make the enemy move in the opposite direction. Given the following example:  The enemy tries to move down, but is unable to. It then continuously moves right while checking if he can move down, but is eventually blocked by a wall. In this case, the algorithm would swap the direction and make the enemy continuously move left until it can move down and continue back on its path.

5.4 Possible Improvements

- A more optimized and smarter pathfinding algorithm
    - Even though it is hard to implement, A* Pathfinding might be possible if I was given more time to figure it out.

- Design more levels
    - The current game only has one level, which gets quite dull since there is nothing to do after completing it.

- Implement music and sound effects
    - The original Pac-Man game had iconic sound design that add to the experience, without it, the game does not have the same lively effect.

6 DUNGEON CUBER: PROJECT BACKGROUND

6.1 Project Name

Dungeon Cuber

6.2 Project Background/History

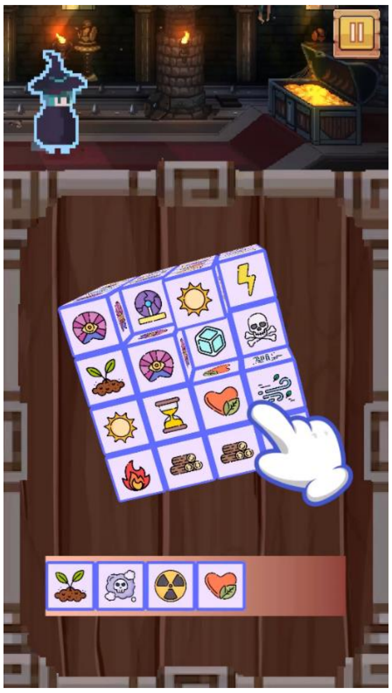
The original implementation of this project was a game commissioned by a client. The client mentioned that his main inspiration for this game came from a game called Tap Away 3D. It is a puzzle game where you tap blocks on the cube to make them fly in the direction indicated by their design. However, you cannot move a block if its direction is blocked by another block. The goal of the game is to clear the entire cube by tapping the blocks in the correct order. To date, Tap Away 3D has amassed over 1 million downloads.

6.3 Project Scope

- Analyze implementation and code architecture of existing project
- Create own rendition of project from scratch
- Improve code architecture and readability
- Improve and build on gameplay elements and mechanics
- Improve general aesthetics of the game
- Improve UI elements

6.4 Project Description

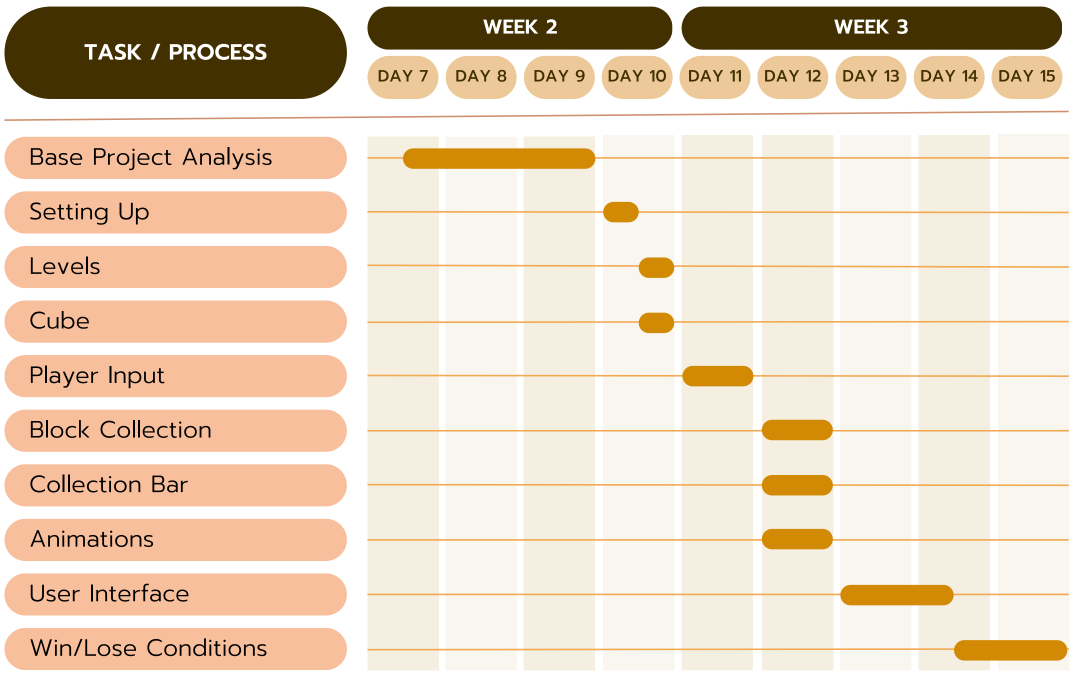
The original project is a mobile puzzle game where the player can rotate a 3x3x3 cube made of individual blocks. Tapping on a block would collect it and add it to a bar at the bottom of the screen, where collected blocks are automatically sorted according to their type. When the player collects three of the same type of block, it forms a match, and the match gets cleared from the bar. The player needs to be careful and not fill up the bar all the way with different collected block types; otherwise, they will lose the game. However, if they successfully clear all the blocks in the cube, they will clear the level and move on to the next.



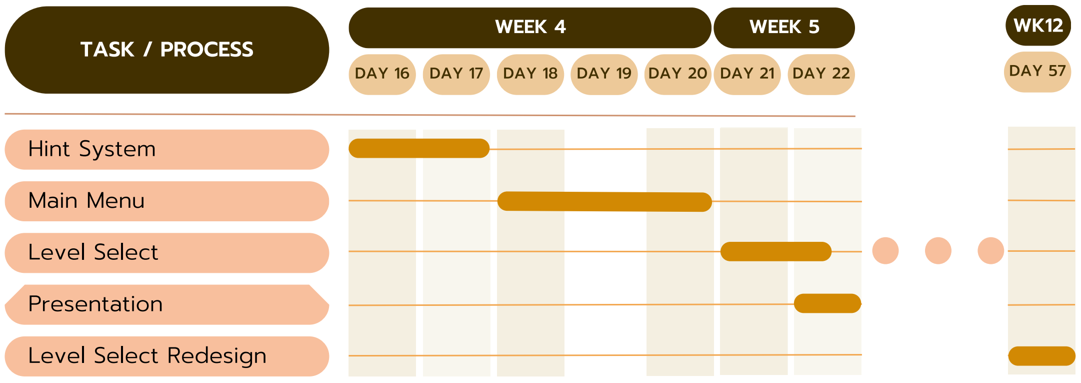
For my recreation of this project, I have decided to make the game dungeon crawler-themed. The player is a witch who is trying to go through a dungeon to obtain the ultimate treasure. To do so, she needs to cast teleportation spells to move through the different sections of the dungeon, and the incantation gets increasingly difficult as the security gets heavier. All the block types would have different elements. The clearing of a cube signifies that the teleportation spell has succeeded, and the witch can then proceed to the next area.

7 DUNGEON CUBER: PROJECT SCHEDULE

7.1 Task Schedule



Dungeon Cuber Gantt Chart Pt 1



Dungeon Cuber Gantt Chart Pt 2

7.2 Task List

Week 2

- Analysis of original project
- Setting up
- Levels
- Cube

Week 3

- Player Input
- Collection of blocks
- Collection Bar
- Animations using DOTween
- User Interface
- Win/Lose Conditions

Week 4

- Hint System
- Main Menu

Week 5

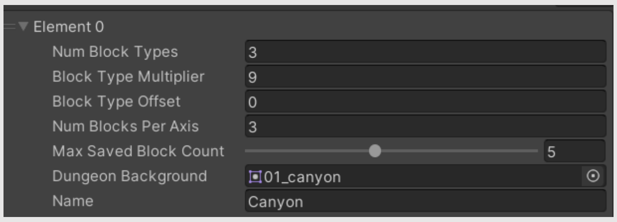
- Level Select
- Presentation

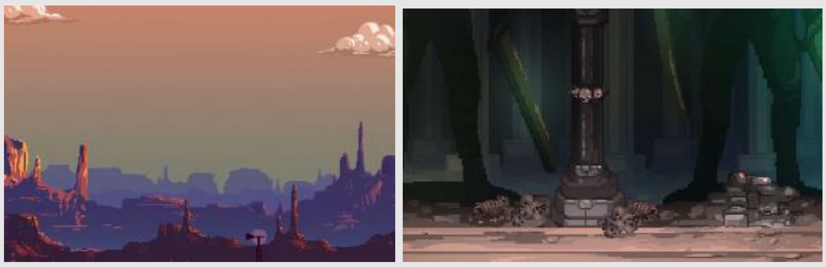
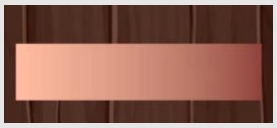
Week 12

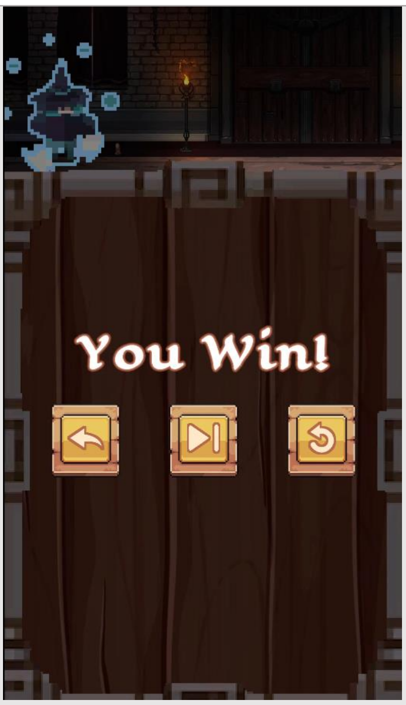
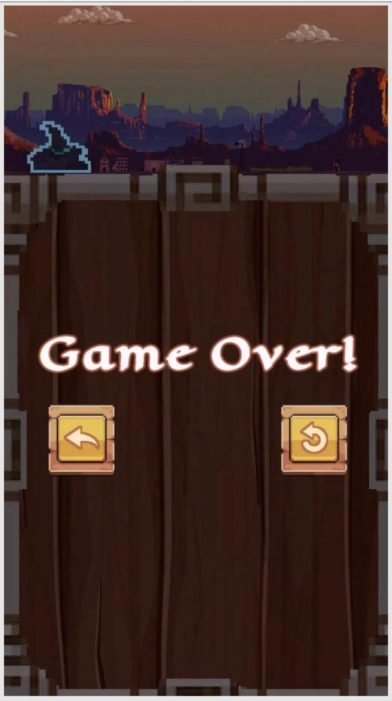
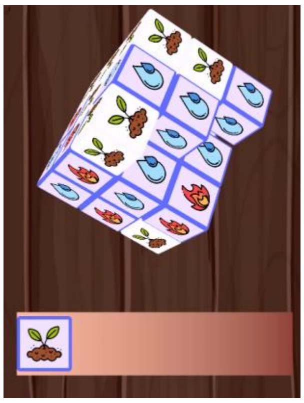
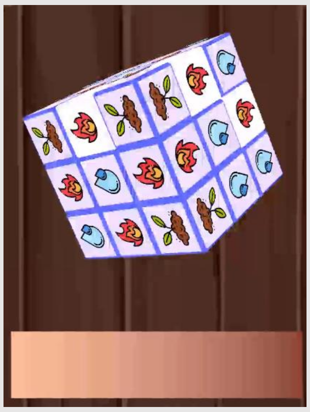
- Level Select Redesign

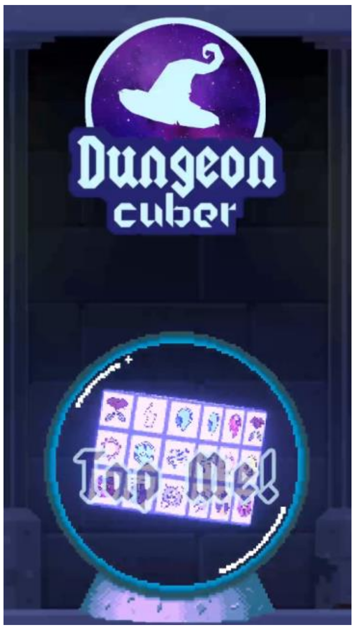
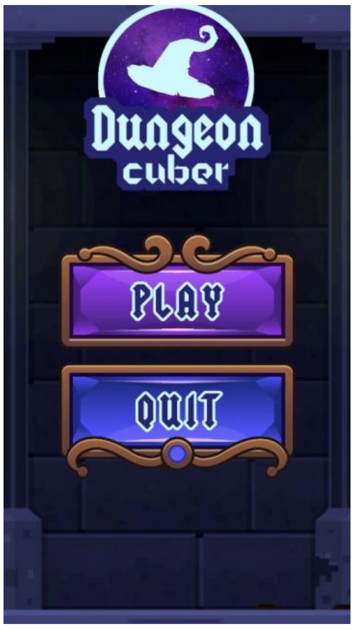
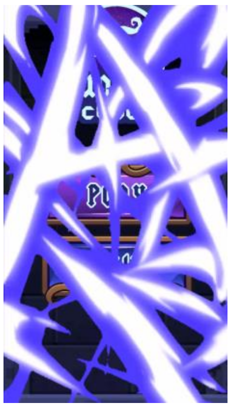
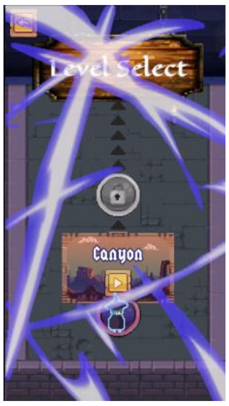
8 DUNGEON CUBER: PROJECT ACCOMPLISHMENTS

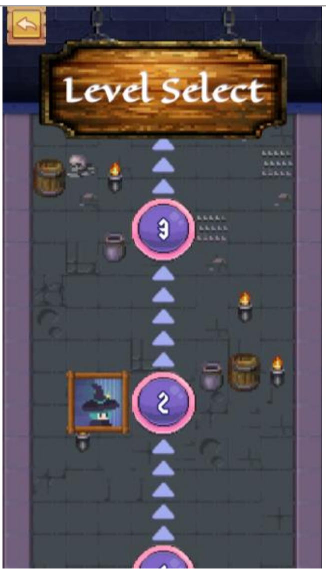
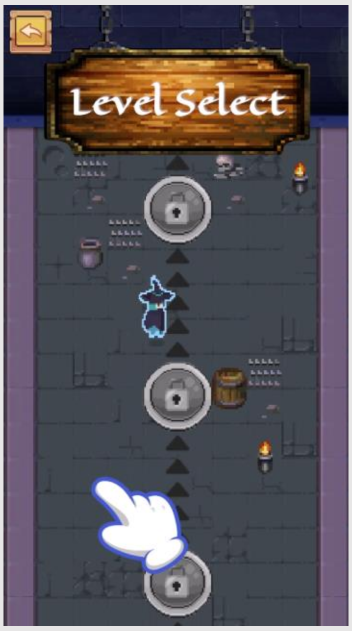
8.1 Accomplishments

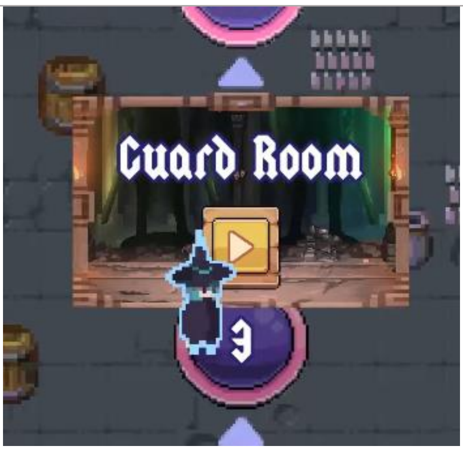
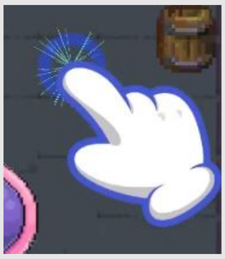
Base Project Analysis	Reviewing and breaking down the code of the original project
	Drawing and cleaning up layout of UML class diagram using PlantUML
	Drawing and cleaning up layout of UML sequence diagram using PlantUML
Setting Up	Installing Unity, Visual Studio Code, PlantUML and dependencies
	Cloning new project from Git
Level	Creation of base serializable class Level that controls various level-specific variables that affect difficulty  Element 0 Num Block Types: 3 Block Type Multiplier: 9 Block Type Offset: 0 Num Blocks Per Axis: 3 Max Saved Block Count: 5 Dungeon Background: 01_canyon Name: Canyon
	Creation of LevelManager class that stores a list of Levels and controls the loading and switching of levels
	Setup 5 levels with variables that increase in difficulty
Cube	Dynamic spawning of cube based on current level's variables
	Randomization of block types on cube
Player Input	Rotation of cube based on input vector
	Pointer sprite showing up whenever player taps on the screen
Block Collection	Destruction of blocks when the player taps on them
	Implemented moving of blocks to collected bar after tap
Collection Bar	Dynamic scaling of bar width and height based on screen resolution
	Display of bar with collected blocks through rendering a second camera
	Custom insertion sort algorithm for a more optimized sorting of blocks

	Checking of a match in bar depending on the number of blocks per axis specified in the level (eg: for a 3x3x3 cube, 3 blocks of the same type is required for a match, but for a 4x4x4 cube, 4 blocks are required)
DOTween Animation	Created custom grow and shrink animation for tapped block
	Created custom shake animation for blocks appearing in collection bar
	Created custom block shifting animation
	Created custom spin and shrink animation for cleared blocks
	Animation of cube exploding upon lose
Win/Lose Conditions	Checking whether entire cube has been matched, if so, level is cleared
	Checking whether entire collection bar has been filled up, if so, fail the level
	Resetting of levels when retrying
	Implemented going to next level on level clear
User Interface	Added backgrounds to the game that changes across levels 
	Added witch sprite with different states and 4 different animations
	Custom gradient shader for collection bar 
	Collection bar fading out on game end
	UI overlay fading in on game end and playing the corresponding witch animation

	 
Hint System	Displaying of hint after 3 seconds of idle time
	Changing and resetting of materials for highlighted block
	Highlighting remaining block(s) of same type in the cube if the player has collected some blocks. Highlighted type is checked based on how close the player is to a match, for example, a player has a collection bar of "Plant, Water, Water". In this case, highlighting water blocks would take priority over Plant. 
	Highlighting a random match in the cube if there are no collected blocks. Unlike the previous hint type, this will only highlight the required amount to make a match. 

Main Menu	Created custom pixelated RenderTexture for menu cube's camera to fit with the 8-bit aesthetic
	Created custom glow post-processing and material for the cube
	Created assets and animated the main menu screen. Main menu starts off with fading in the logo, and sliding in a crystal ball with the cube floating and spinning inside. When the player taps on the crystal ball, it would continuously rock back and forth until the cube explodes. The menu buttons would then slide in afterwards.  
	Implemented ability to skip animations by tapping on the screen while the current animation has not yet finished
Level Select	Created a scene transition animation 
	Created tiles and a tilemap for level select
	Programmatically generated tilemaps. Essentially, I get the middle point of the screen and the player's current level. Then, based off of the middle, I begin drawing a node at the center of the screen based on the player's level. Then, based on the number of levels in level manager, I expand downwards and upwards, drawing more nodes and paths until the number of levels is satisfied. After this, I draw the walls of the dungeon by getting the outermost edges of the screen's resolution and draw them based as much The ground is then filled in based off of the surrounding walls drawn. Finally, instantiate a photo of a witch next to the current level such that players can track their progress.

	
	Movement of camera by dragging screen, takes into account the constraints of the dungeon
	Creation of custom ScriptableTile class that can store information of the node, such as the level number of the node.
	Implemented node tapping functionality. Tapping on a node would change its sprite to a darker "pressed" version, and releasing would transition the witch to that specific level
	Addition of décor in the map. The values can be tweaked easily in the inspector, but essentially the more the witch progresses in the dungeon, the more décor and dangerous items spawn. These décor ranges from spikes to skeletons to barrels.
Presentation	Created presentation slides that contain Japanese translations
	Presented the game along with what was changed from the original tutorial to our supervisors and colleagues
Level Select Redesign	Instead of instantiating a static photo, I modified the code such that it instantiates a controllable witch entity. It has idle and walking animations, and the player can control the witch by tapping anywhere on the screen, and the witch would try to go to that position. If the position is blocked by a wall, the witch would stop. 
	Removed functionality to tap on nodes. Now, the player needs to move the witch on top of a node to display the dungeon information, then click the play button on the popup

	
	Addition of particle effects when tapping in level select 

8.2 Concepts Learnt

- PlantUML
- DOTween
- Linq
- Class and Sequence Diagrams
- Serializable fields and classes
- Dictionaries
- Sorting algorithms
- Coroutines and callbacks
- Camera RenderTexture
- Unity Sprite Editor & Slicing
- Unity Shaders
- Unity Tile Map system
- Scriptable Tiles

8.3 Challenges Faced

Challenge	Solution
I wanted to fade the collection bar out on game over, but it is being rendered via a camera, not a UI element.	After some research online, I found a new feature in Unity called RenderTextures. Essentially, these act as textures that can be appended to raw image UI elements that gets updated in runtime based on the camera you attach it to. Through this, I am able to tweak the alpha component of the raw image to get the entire camera render to fade out smoothly. However, I also faced another issue where static was appearing on some mobile phones in the place of the render texture. I solved this by simply clearing the render textures on initialization.
First time using the DOTween library, was difficult to figure out how to get the animation look I wanted	DOTween is an extremely useful library that helps to programmatically generate animations by tweening between values like transformation and alpha. As such, I spent quite a lot of time in a separate test project just messing around with the different features available to get myself familiar with the library.

Wanted to programmatically generate my level select page, but instantiated the same sprite that many times is quite resource intensive	I decided to opt for a tilemap system. While it was slightly more complicated to implement due to needing to work with 3 different coordinate systems (Cell, Screen, World), I am quite happy with the final result. Using the tilemap system also enabled me to quickly implement my desired feature of moving a witch around during the final week of my internship. This is because since everything was in place already, I only needed to add a collider to the wall tilemap, and was able to not change that much of the code.

8.4 Possible Improvements

- Add enemies
    - The original idea for my project was actually to make the matches cast spells to attack enemies in the dungeon. However, due to insufficient time, I was unfortunately unable to implement this feature and opted for a simple teleportation spell instead. However, adding this feature would make the game much more interesting and give it a better difficulty curve.
    - Essentially, there would be enemies that march towards the player from a base on the right side of the screen.
    - The player needs to complete matches quickly to cast spells and kill enemies before they come too close to her.
    - There is also an additional combo system where the player's combo goes up if they complete multiple matches in a row. For example, if the player collects the blocks of type "A, A, B, B", then collects another block A, forming a match, and collects another block B immediately afterwards, forming another match. This would give the player a combo of 2. The higher the combo, the more powerful the spell.
    - The base mechanics are still the same, where the clearing of the cube would immediately win the game and the witch would destroy the base of the enemies.
    - However, there is an added score system now thanks to this new system, where at the end of the level it will display the number of enemies the player has killed. The players would then be incentivized to play riskier to get a better score.

9 PIZZA DASH: PROJECT BACKGROUND

9.1 Project Name

Pizza Dash

9.2 Project Background/History

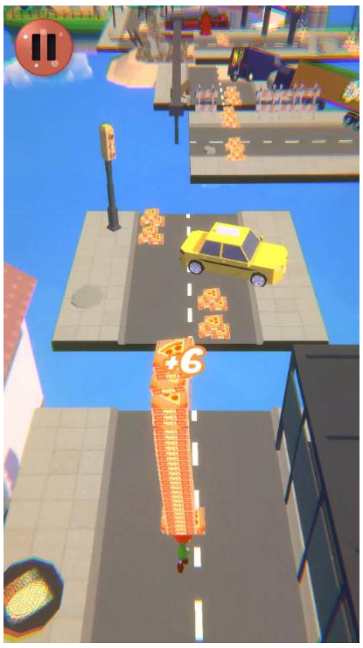
The original implementation of this project was another game commissioned by the same client. The client mentioned that his main inspiration for this game came from a game called Shortcut Run. It is a multiplayer game where you collect floorboards along a track and use them to build bridges to use as shortcuts to cut across sections in the level. The goal of the game is to reach the finish line before everyone else. To date, Shortcut Run has amassed over 50 million downloads.

9.3 Project Scope

- Analyze implementation and code architecture of existing project
- Create own rendition of project from scratch
- Improve code architecture and readability
- Improve and build on gameplay elements and mechanics
- Improve general aesthetics of the game
- Improve UI elements

9.4 Project Description

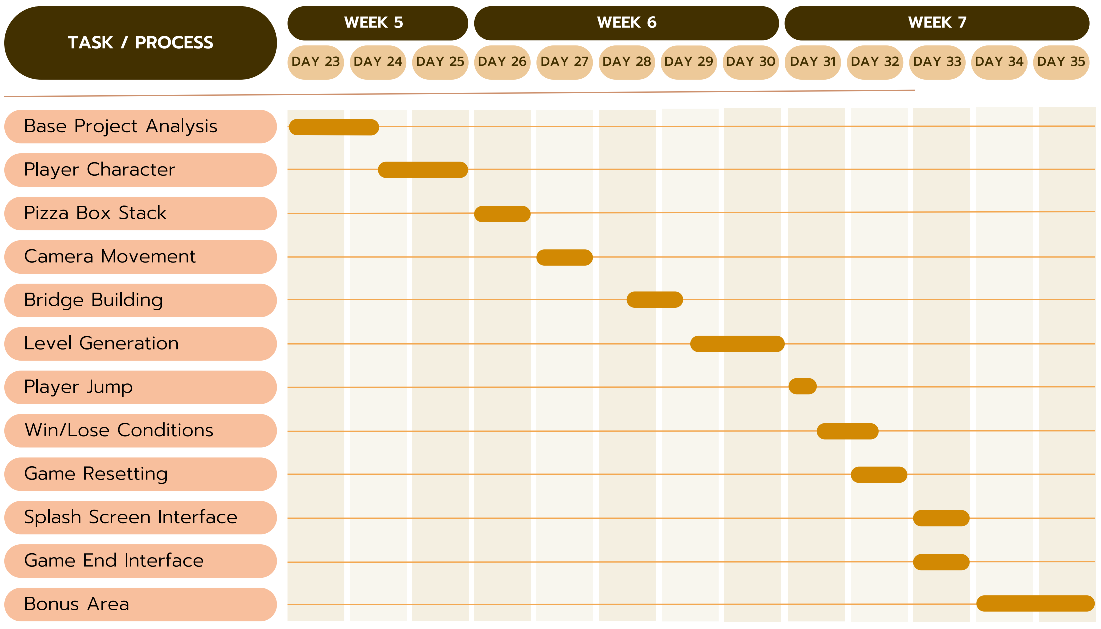
The original project is a fast-paced mobile runner game where the player has to collect planks scattered throughout the level. The player is able to use these planks to build a bridge to across gaps between blocks in the level. Going over an edge with no planks left would cause the player to automatically jump a short distance. The goal of the game is to successfully reach the end of the level without falling through the gaps. There is a bonus section at the end of the level where the player uses all their remaining planks to build a long bridge, and finally jump off from it to land on their final score.



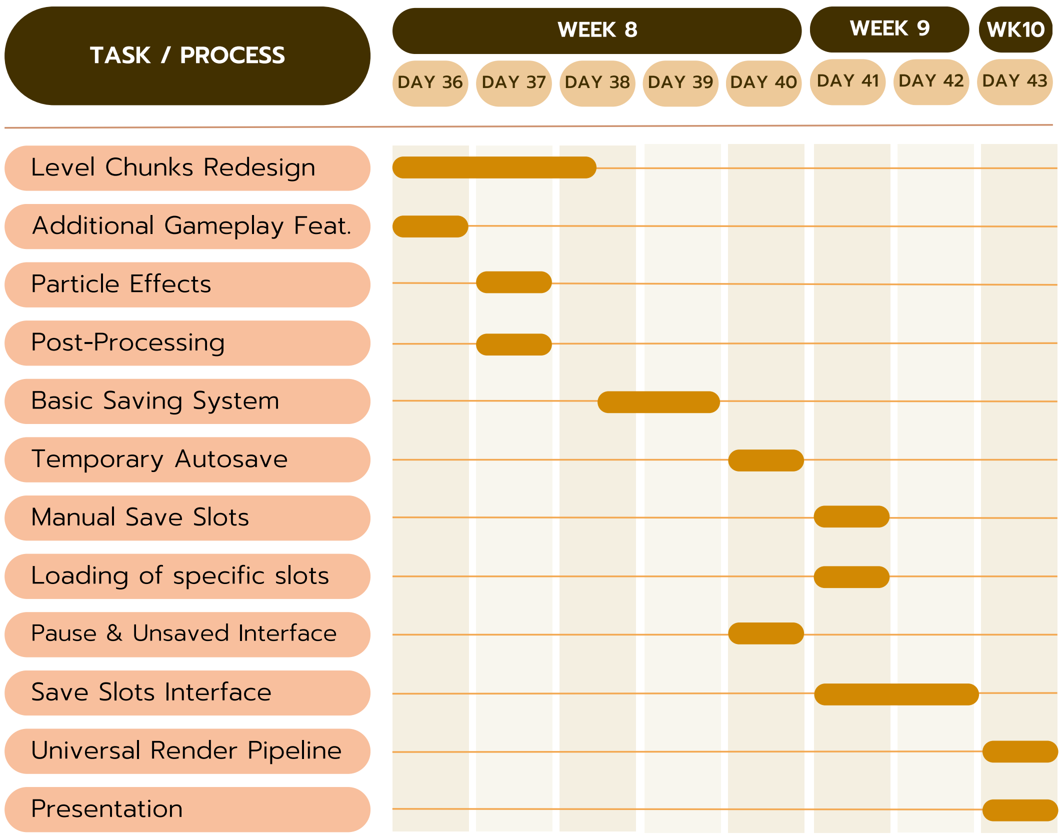
For my own implementation, I decided to give this project the theme of delivering pizzas amidst an earthquake. Instead of planks, the player would have to collect pizza boxes scattered throughout a crumbling city and use the pizza boxes as makeshift bridges to get across the gaps in the level. There is also debris, fallen structures, and vehicles that the player would have to avoid. At the end, the player reaches a village, where they deliver their remaining pizzas to hungry villagers. The final score would be based on how many pizzas the player could successfully deliver.

10 PIZZA DASH: PROJECT SCHEDULE

10.1 Task Schedule



Pizza Dash Gantt Chart Pt 1



Pizza Dash Gantt Chart Pt 2

10.2 Task List

Week 6

- Analysis of original project
- Player character
- Stacking of pizza boxes
- Camera movement
- Bridge building
- Dynamic level generation

Week 7

- Player jump
- Bonus area at the end of level
- Win/Lose conditions
- Game resetting
- User Interface
  - Splash sceen
  - Game end screen

Week 8

- Level chunks redesign
- Additional Gameplay Elements
- Particle Effects
- Post-Processing
- Save and Load system
  - Basic saving
  - Temporary autosave

Week 9

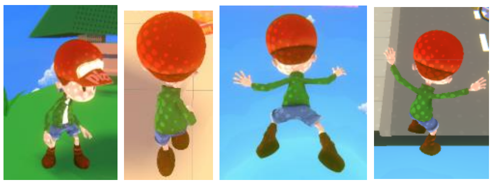
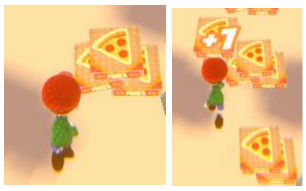
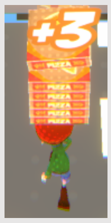
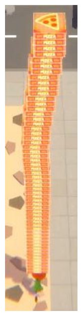
- Save and Load system
  - Manual save slots
  - Loading of specific save slots
- User Interface
  - Pause screen & Unsaved Progress screen
  - Save Slots screen

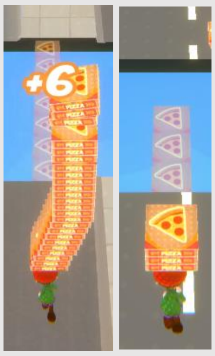
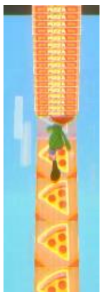
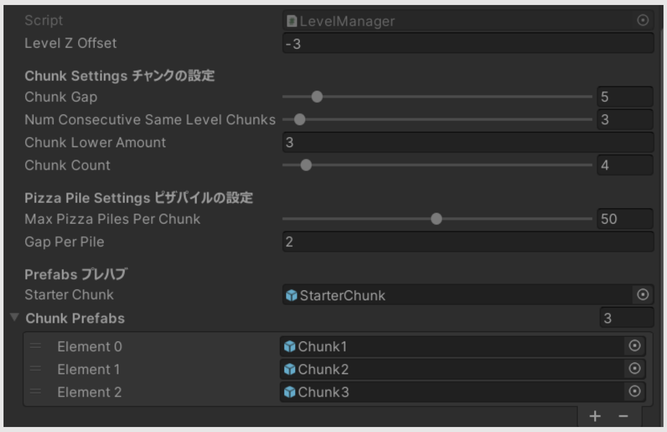
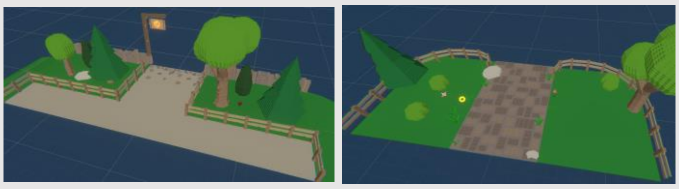
Week 10

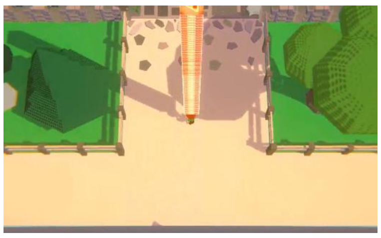
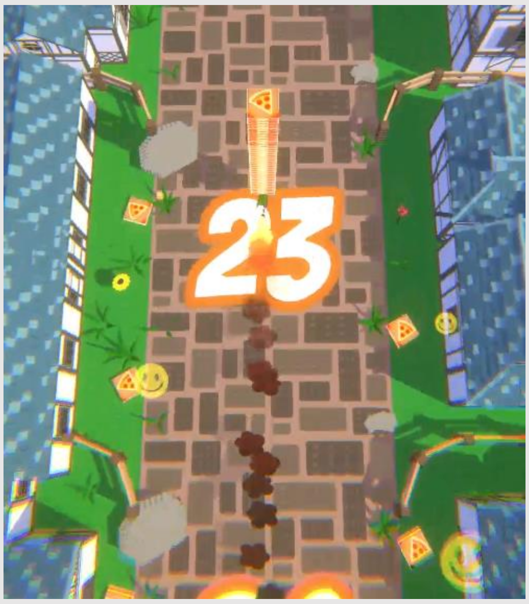
- Changing render pipeline to Universal Render Pipeline
- Presentation

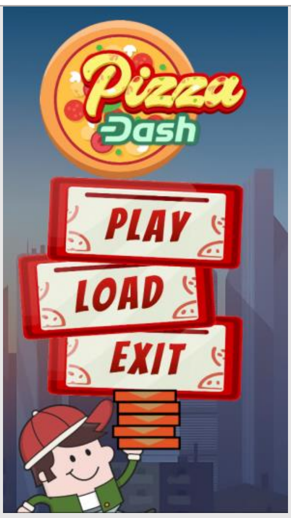
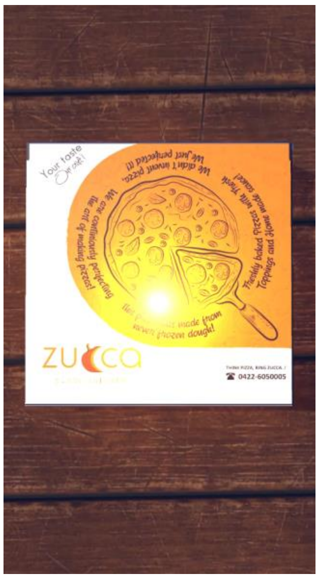
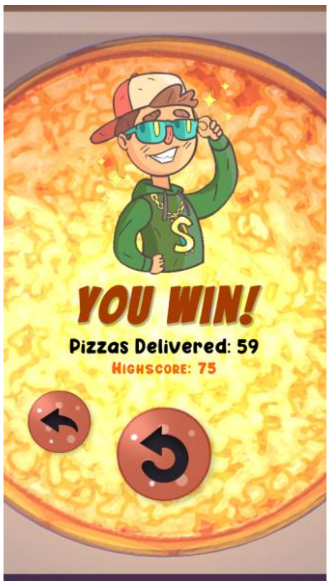
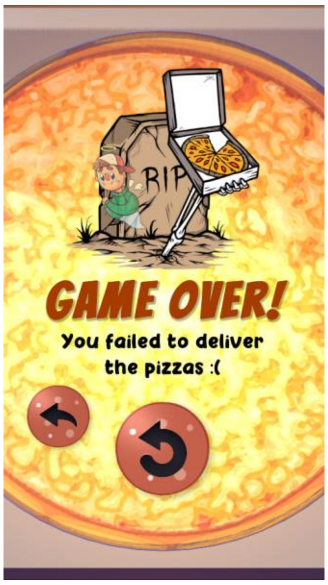
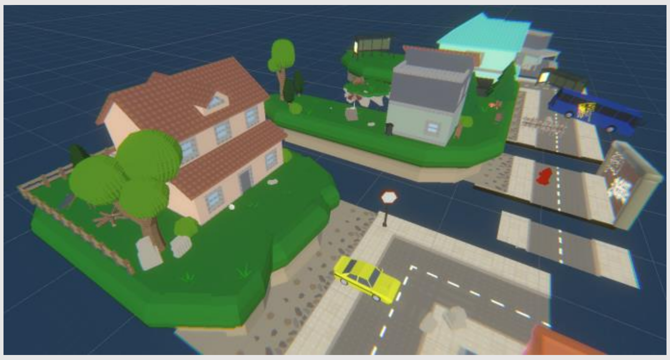
11 PIZZA DASH: PROJECT ACCOMPLISHMENTS

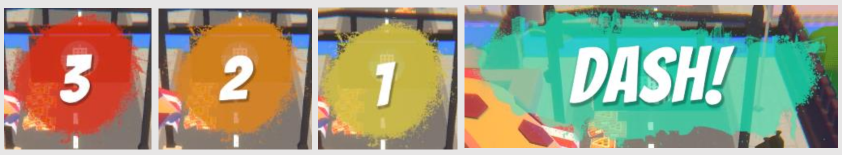
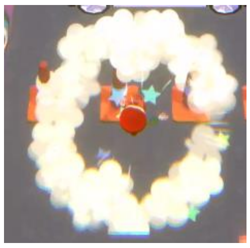
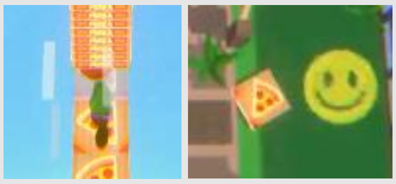
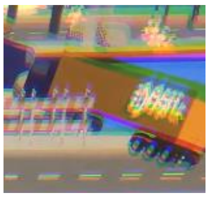
11.1 Accomplishments

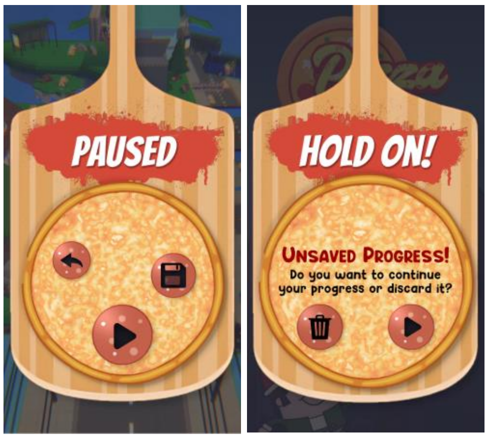
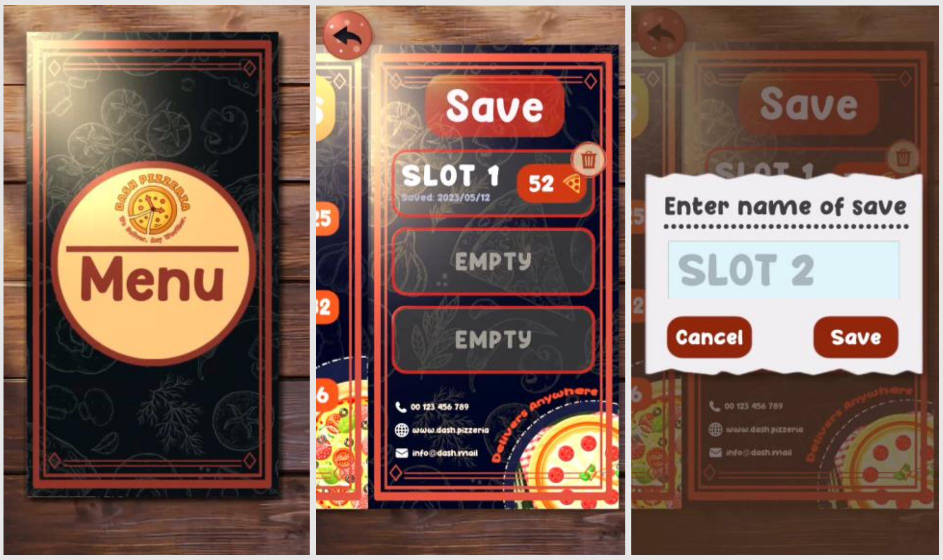
Base Project Analysis	Reviewing and breaking down the code of the original project
	Drawing and cleaning up layout of UML class diagram using PlantUML
	Drawing and cleaning up layout of UML sequence diagram using PlantUML
Player Character	Importing 3D character model, animations, and modifying textures
	Player state machine
	Player movement based on swiping left and right. Movement speed and controlability changes according to current state of player (eg: when bridging, the player runs forward faster but the player is unable to move left or right).
	Player animation based on current state 
Pizza Box Stack	Object pooling of pizza boxes and stack items to reduce instantiation calls during runtime, thus reducing lag
	Collection of individual pizza boxes when the player runs over them 
	Transferring pizza boxes onto a stack on player, with a text indicator showing the amount collected over a 1 second interval before fading out and resetting the count 
	Animation of stack entirely done through calculations. Stack moves based on the direction of the player's movement, with the movement being more drastic higher up in the stack. Then, the stack moves in the opposite direction repeatedly with diminishing magnitudes until they eventually move back to their original position. 
Camera Movement	Following of player while keeping initial camera offset
	Camera zooming out with each pizza box collected
	Easing of camera zoom through animation curves
	Detection of edges in the level using boxcasts

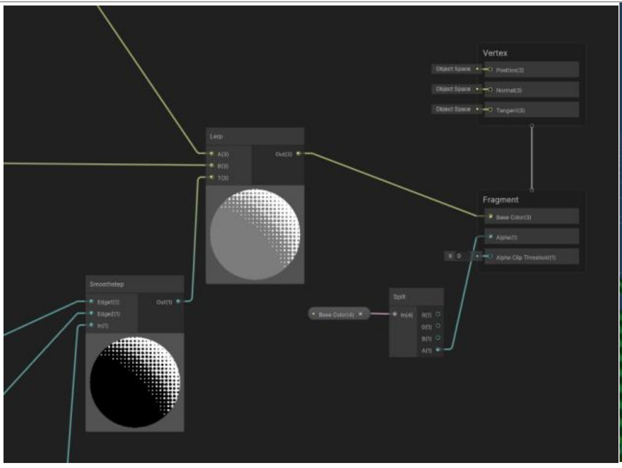
Bridge Building	Rendering of ghost of bridge to be built when player is near an edge. If the player has sufficient pizza boxes, would display a bridge that stretches until the next edge. If not, would just display a bridge that is the length of all the remaining pizza boxes the player currently has 
	Building of bridge once player crosses bridge ghost, while transferring the used pizza boxes away from the stack 
	Deactivation of bridge items once they exit camera view for optimization
Level Generation	Separating level into different chunk prefabs
	Randomized loading of chunks based on various soft-coded variables  Script LevelManager Level Z Offset -3 **Chunk Settings チャンクの設定** Chunk Gap 5 Num Consecutive Same Level Chunks 3 Chunk Lower Amount 3 Chunk Count 4 **Pizza Pile Settings ピザパイルの設定** Max Pizza Piles Per Chunk 50 Gap Per Pile 2 **Prefabs プレハブ** Starter Chunk StarterChunk **Chunk Prefabs** 3 Element 0 Chunk1 Element 1 Chunk2 Element 2 Chunk3
	Dynamic generation of pizza box piles based on the boundaries defined via box colliders in each chunk prefab
Player Jump	Implementation of automatic jump whenever player reaches an edge (regardless of being on a bridge or chunk). Adds an upward force to the player but slows down the forward movement
	Updating player states and animations to have jumping
Bonus Area	Designing the bonus start and bonus section prefabs using free models and an external editor 

	Creation of bonus start's beginning and end colliders that switches the state of the player accordingly. Bonus start also moves the horizontal position of the player towards the middle while easing the camera to a top-down view. 
	Implementation of bonus section functionality. On spawn, the bonus setion would randomize 4 houses along designated positions. There are also 4 pizza boxes in the prefab set to inactive, with 4 large colliders. When the player enters a collider, it would set the corresponding pizza box to active while removing one item from the player's current stack, to show that the player delivered that pizza. While this happens, update the score and display it accordingly on the floor as well. 
Win/Lose Conditions	Added fail zones to the bottom of each chunk that causes the player to lose the game as soon as he falls through them
	Created callback when all pizzas have been delivered at the bonus stage, at which the player would win the game
Game Reset	Called the Reset function of all tasks, which reset the necessary variables for a new game
User Interface	Created assets and animated the splash screen. Pizza rolls in, then the text for the logo fade in along with a pizza boy dashing in from the left side of the screen. Finally, the buttons of the menu drop on top of the hand of the pizza boy one after the other, simulating the pizza boxes that the player will have to collect when they play the game.

	
	Created assets and animated the game end screen. Used a secondary camera with a render texture that fades in the display. The second camera looks down at table. Then, a pizza box slides in from the top, shakes a bit, then opens to reveal the results in a plain cheese pizza. The buttons are then styled as pepperonis in the pizza.  
Level Chunks Redesign	Designed 3 new chunks using free models and an external editor. These chunks are less blocky compared to the original project, and have edges with variation to give the level a more organic feel. Additionally, I have added pizza-related graffiti to tie in with the theme better. 
	Designed a starting chunk that the player always start off at. This chunk is directly outside a pizza shop, so it helps to add to the visual storytelling of the game.

Additional Gameplay Elements	Added a countdown at the start of the level so that players could prepare after pressing the start button 
	Added a new way to lose in the game. Since there are a lot of obstacles in the chunks I designed, the player is able to lose by bumping into them as well. 
Effects	Added particle effects for building bridges, delivering pizzas, and bumping into obstacles 
	Added post-processing bloom and chromatic aberration to fit with comic book style 
Save/Load System	Pause menu that freezes all physics, animations, and particle effects
	Saving and loading of various game data including: • Player's position and state • Exact animation frame of player • Retaining velocity of player, ensuring that saving mid-jump is possible • Randomized level chunks and collected pizza piles • Collected pizza box stack on player • Camera position

	• Built bridges
	Automatic temporary saving of data whenever player exits the app or goes back to the main menu in the middle of a level. Temporary data automatically gets discarded when the player wins or loses.
	Implemented AES Encryption for saved data
	Implemented 3 different save slots which also saves a custom save slot name and the date the slot was saved
	Deletion of save slots
	Loading of specific save slot
User Interface (Cont.)	Created assets and animated the pause and unsaved progress screens. A paddle with a plain cheese pizza slides in from the top, with the buttons and information fading in afterwards  PAUSED HOLD ON! UNSAVED PROGRESS! Do you want to continue your progress or discard it?
	Created assets and animated the save slot progress screen. Designed 3 pages of a pizza diner menu, and placed the save slots on the final page. The menu slides in from the top, and opens up to reveal the save slots. The input box for entering a custom save slot name is styled after a receipt.  Menu Save SLOT 1 52 Saved: 2023/05/12 EMPTY EMPTY 00 123 456 789 www.dash.pizzeria info@dash.mail Delivers anywhere Enter name of save SLOT 2 Cancel Save
Universal Render Pipeline (URP)	Updated all materials in project to use URP
	Created custom sub-graph asset to obtain main light in scene
	Created custom half-tone shader graph and applied them to materials to give the game a comic book look

	
Presentation	Created presentation slides that contain Japanese translations
	Presented the game along with what was changed from the original tutorial to our supervisors and colleagues

11.2 Concepts Learnt

- PlantUML
- DOTween
- Linq
- Data structures (Stacks and Lists)
- Object pooling
- Renderer bounds
- Encapsulating bounds of multiple renderers
- C# "get" and "set" properties
- CeilToInt function
- OnBecameInvisible callback
- AES Encryption
- Unity's Physics Module (Raycast, Boxcast, OverlapBox)
- Unity Animator Controllers and Animations
- Unity Animation Curves
- Unity Particle System
- Unity JSONUtility
- Usage of an External Editor (Modular Asset Staging Tool)

11.3 Challenges Faced

Challenge	Solution
The code for the original project was quite messy with almost no comments provided.	I went through the entire code line-by-line, commenting things out to figure out what each piece of code does. Finally, I went extremely in-depth in my UML diagrams that act as my own personalized comments for the project. In the end, I was able to get a clear picture of how the original project was implemented, and even learn a couple new features from the code.
The game was quite laggy on low-end mobile devices.	Taking advantage of the fact that I switched my project over to use the Universal Render Pipeline, the additional control allowed me to tweak a lot of minor graphics settings such as lowering the shadow resolution and which lights should cast shadows. I have also removed a lot of unneccessary post-processing. While there are still some lag spikes, the game now runs much smoother on low-end devices.

11.4 Possible Improvements

- Cutscene
    - Currently, there is no proper explanation for why there are pizza boxes everywhere. Originally, I wanted to create a short cutscene where the pizza delivery truck did not close its trunk properly, so all the pizza boxes fell out. As a result, the pizza boy begins to chase after the truck and then the game begins. However, due to the lack of time, I was unable to implement this.

- Collapsing sections of roads
    - To give the game an even more immersive experience, it would be nice to implement sections of the roads that collapse in front of the player when they get near. This makes the environment more dynamic and makes the player feel like they are really in the middle of a natural disaster.

12 GALATIC GUARDIAN: PROJECT BACKGROUND

12.1 Project Name

Galatic Guardian

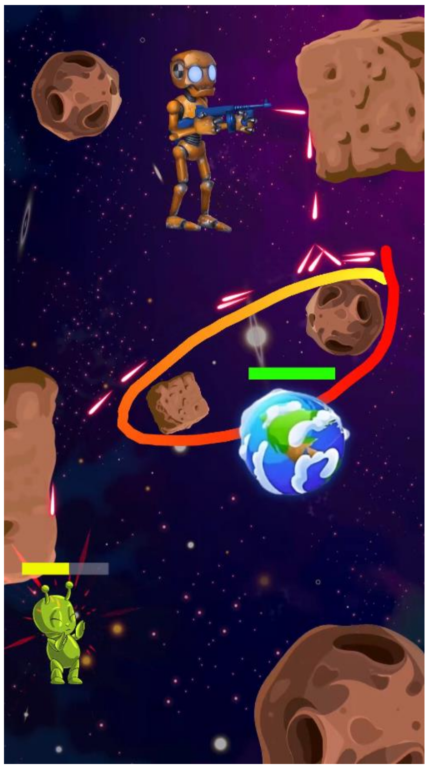
12.2 Project Background/History

The original implementation of this project was another game commissioned by the same client. The client mentioned that his main inspiration for this game came from a game called Save the Doge. It is a puzzle game where you draw shapes to protect the Doge from being hurt by enemies like bees and crocodiles. To date, Save the Doge has amassed over 50 million downloads as well.

12.3 Project Scope

- Analyze implementation and code architecture of existing project
- Create own rendition of project from scratch
- Improve code architecture and readability
- Improve and build on gameplay elements and mechanics
- Improve general aesthetics of the game
- Improve UI elements

12.4 Project Description

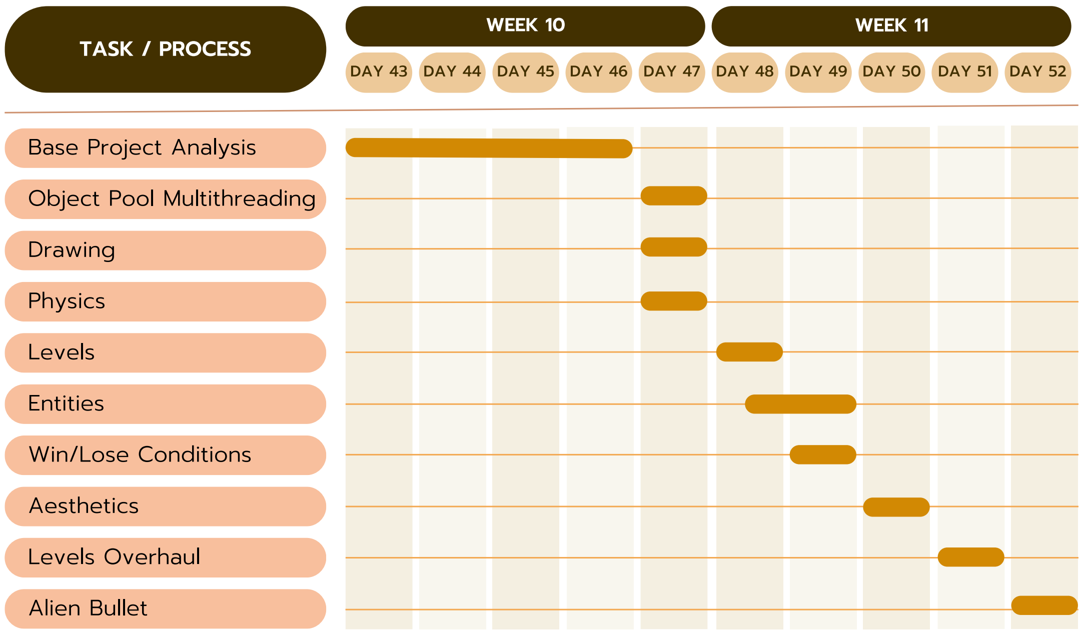


The original project is a mobile puzzle game where the player is able to tap and hold to draw a line in the level. Upon releasing their finger, the drawn line would be turned into a collider that gets affected by gravity. At the same time, the player character would begin shooting a limited number of heart objects. The goal of the game is to draw a line to direct the required amount of hearts towards a girl, making her fall in love with the player. There are also various gimmicks across different levels. For example, there are portals that teleport the hearts to their partner portals. Additionally, there is another girl who instantly causes the player to lose the game upon receiving enough hearts.

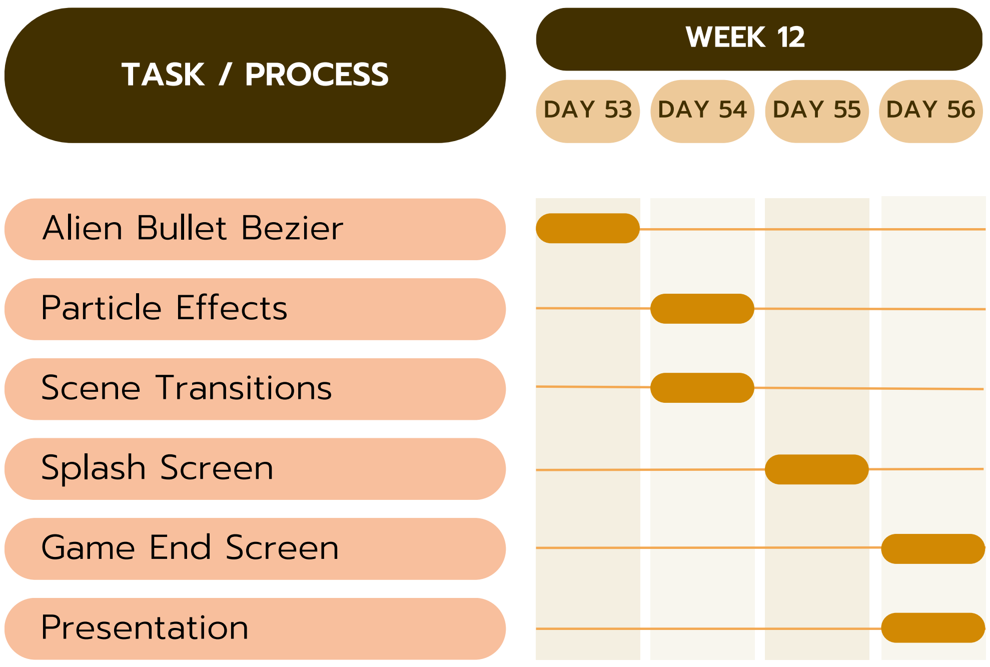
For my implementation of this game, I decided to go with a sci-fi theme. The player plays as a robot that is the galaxy's last hope of standing against an alien invasion. Instead of hearts, the player shoots plasma bullets that should be directed towards aliens to damage them and eventually defeat them. As for the gimmicks, the portals would be replaced with blackholes. Additionally, there would be planets that the player should avoid hitting with his bullets, and destroying them would mean an instant game over.

13 GALATIC GUARDIAN: PROJECT SCHEDULE

13.1 Task Schedule



Galatic Guardian Gantt Chart Pt 1



Galatic Guardian Gantt Chart Pt 2

13.2 Task List

Week 10

- Analysis of original project
- Object Pooling Multithreading
- Drawing
- Physics

Week 11

- Levels
- Entities

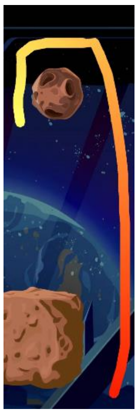
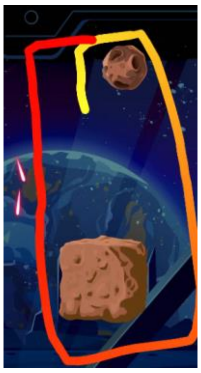
- Win/Lose Conditions
- Aesthetics
- Overhaul of Levels
- Alien shooting bullet

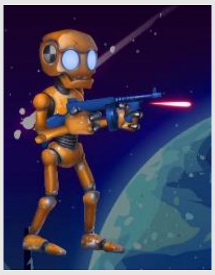
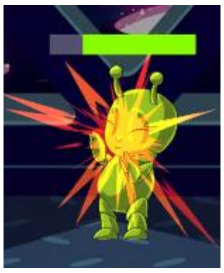
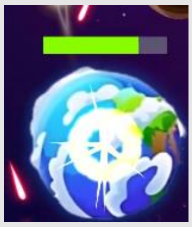
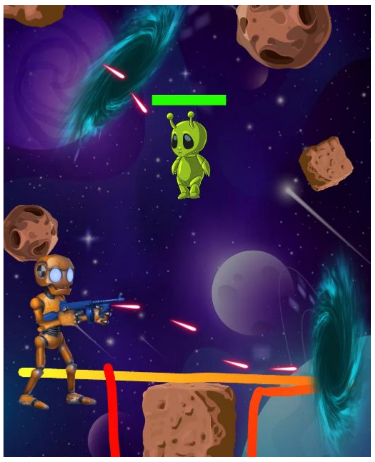
Week 12

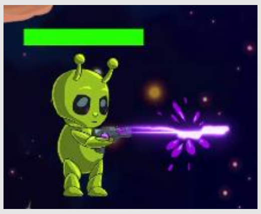
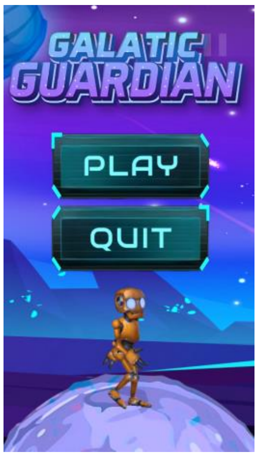
- Adding bezier curve to shot bullet
- Particle Effects
- User Interface
    - Transition Animation
    - Splash Screen
    - Game End Screen
- Presentation

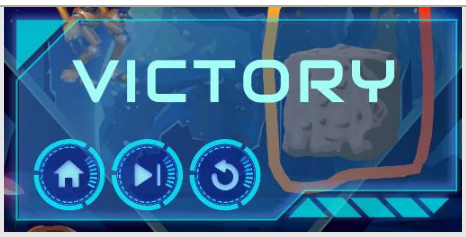
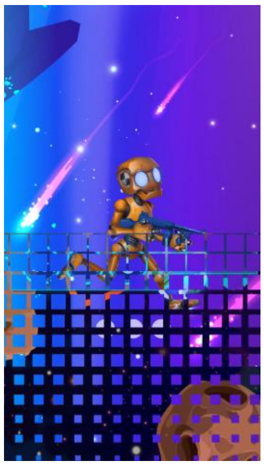
14 GALATIC GUARDIAN: PROJECT ACCOMPLISHMENTS

14.1 Accomplishments

Base Project Analysis	Reviewing and breaking down the code of the original project
	Drawing and cleaning up layout of UML class diagram using PlantUML
	Drawing and cleaning up layout of UML sequence diagram using PlantUML
Object Pooling Multithreading	Usage of UniRX to run multithreading in the background while the game switches to a loading animation
Drawing	Input management using UniRX Subscribers
	Instantiating of line object and adding points to line based on tap position
	Implemented custom gradient material 
Physics	Generation of edge collider based on line
	Enabling of physics and gravity on tap release 
	Recalculation of center of mass based on line drawn
	Line drawn gets destroyed when out of camera view
Levels	Level instantiating based on list of prefabs given

	Dynamic scaling of levels based on screen resolution
Entities	Creation of player entity that shoots out bullets 
	Creation of alien entity with a health bar 
	Creation of planet entity that inherit from the same base class as the alien 
	Creation of black hole entity that teleports bullets to its partner. Modified to allow for teleportation through both sides, and propelling the bullet based on the direction of rotation of the black hole. 
Win/Lose Conditions	Subscribe to a list of observables containing whether all aliens have died, if so, call the win event
	Subscribe to a list of observables containing whether all bullets from all players have been shot out, if so, call the lose event
	Subscribe to all planet's death state, as soon as one of them gets destroyed, call the lose event
Aesthetics	Created custom animations for players and aliens to simulate floating in space using Spine
	Linked animations to events in-game
	Made characters float by translating a random amount up and down slowly
	Made bullets rotate according to their velocity's direction

Levels Overhaul	Creation of 5 levels with progressive difficulty increments and visual storytelling. Player starts off in a spaceship with just a single alien. Then, the player goes outerspace and gets introduced to planet entities while trying to defeat another alien. Then, the next level introduces the player to black holes After which, the player enters a mirror dimension where there are double of every entity (Players, Aliens, Planets) and learns that they need to defeat all aliens to clear a level Finally, the final level puts everything the player has learned to the test as this is the alien's homeworld, and the player has to kill 3 aliens with the help of a blackhole while avoiding hitting a planet.
Alien Bullet	Implementation of special alien bullet that all aliens would fire when the game is lost. The aliens automatically targetting the remaining players or planets alive in the level that have not been targetted yet. It prioritizes destroying planets first, so it would search for all alive planets that have not been targetted by other aliens. If none are found, it would do the same for alive players. If there are still none found, will just continue shooting in a straight line. Upon collision with the alien's bullets, the players and planets would be immediately destroyed. 
	Implementation of a bezier curve such that the bullets feel more natural. Also made it such that the bullets rotate based on the curve.
Particle Effects	Added particle effects for the destruction of entities along with camera shake
	Added particle effects for entities getting hurt
	Added particle effects for player and planet when the game is won
User Interface	Created assets and animated the splash screen. A UFO zips around and pulls up the game logo, while the background pans down. This reveals the robot player walking on the alien's homeworld, with smaller UFOs floating in the background. 
	Created assets and animated the game end screen, which is a holographic display of the result accompanied with relevant buttons.

	
	Created transition animations between scenes 
Presentation	Created presentation slides that contain Japanese translations

14.2 Concepts Learnt

- PlantUML
- DOTween
- UniRX and Reactive Programming
- Multithreading
- Linq
- Generic types
- Gradient Evaluation
- Edge Colliders
- Subscriber Lifecycle Management
- Bezier Curve Calculation
- Animating using Spine

14.3 Challenges Faced

Challenge	Solution
First time learning about reactive programming and using UniRX, so it was quite difficult to get used to it	I read through the documents and researched on various examples during my free time, such is while commuting. Additionally, I made minor notes based on what I learned such that I could refer to them. Thanks to this, I was able to apply UniRX to many aspects of my project, even though it was my first time encountering it.

14.4 Possible Improvements

- 2D Light Shadergraph
  - If given the time, I would like to test what I had learned in the previous project about the Universal Render Pipeline and try to create a shader graph for 2D light. That way, my bullets could glow nad it adds to the atmosphere

- Pause Screen

- Due to the lack of time, unlike the other projects, this project does not have a pause menu. I believe that the player should be given the freedom to replay the level or go back to the splash screen whenever possible, so this is quite an important feature that I missed.

END