SDD Major work

Circle Attack: A tower defence game with randomly generated maps

Student Name: Jaden Choi
Number: 35141405
Teacher: Mr Shirlaw
Course: Software Design and Development

Check 1

Problem statement and initial design specification

Problem Statement

Tower defence games are a lot of fun, however, they get old very quickly because of a lack of new maps; once you have completed all of the maps, there's not much else to do. New maps can come, but that relies on developers consistently updating their games and a good design team. And these updates can be very slow, leaving the game dead and boring for a long time. To fix this, I wanted to make a game that has a new map every single time, so it never gets stale.

Initial design specification

User specifications
- UX: Simple, intuitive and easy to use
- Simple, clean graphics
- Not hard to run
- UX is consistent
- Little to no bugs

Developer specifications
- Put appropriate datatypes to variables
- Structured approach
- Detailed, verbose documentation
- Little to no bugs

Modules:
- Tower placement
  - Tower placement overlays
- Random generation
  - Using noise function
- Main menu
  - Volume sliders
- Pause menu
  - Volume sliders
  - Exit game
- Rounds system
- Tower shooting
  - Damage
  - Balance
- Health systems
- Enemy movement systems

Gantt chart

Name	Begin	End date
Check 1	1/10/23	2/17/23
Problem Statement	1/12/23	1/20/23
Design Specifications	1/12/23	1/20/23
Gantt Chart	1/10/23	1/13/23
Screen design	1/23/23	1/27/23
Storyboard	1/26/23	1/31/23
Context Diagram	1/30/23	2/9/23
Discussion of...	2/13/23	2/17/23
Social and et...	2/13/23	2/17/23
Submission	2/20/23	2/20/23
Part A	2/20/23	3/6/23
Description o...	2/24/23	2/28/23
Pseudocode an...	2/20/23	2/24/23
IPO chart	2/20/23	2/24/23
Structure chart	2/20/23	2/24/23
DFD	2/20/23	2/24/23
Define files ...	2/24/23	2/27/23
Data dictionary	2/27/23	3/3/23
Platform/OS c...	2/27/23	3/3/23
Hardware implications	2/27/23	3/6/23
Submission	3/7/23	3/7/23
Check 2	3/14/23	5/19/23
Choice of UX ...	5/9/23	5/17/23
EBNF & railro...	5/1/23	5/12/23
Test data	4/3/23	4/28/23
Error checking	4/3/23	4/28/23
Desk check	4/3/23	4/28/23
Evidence of breakpoint	3/22/23	3/31/23
Readability of code	3/14/23	5/19/23
Discussion of errors	4/3/23	4/28/23
Submission	5/22/23	5/22/23
Part B	5/22/23	6/29/23
Efficiency an...	6/12/23	6/16/23
Refine UI	5/22/23	6/16/23
Suitable CASE tools	5/22/23	5/24/23
Catering for ...	5/25/23	5/30/23
How would you...	5/25/23	5/30/23
User manual	6/1/23	6/9/23
Installation manual	6/1/23	6/9/23
What have I learnt	6/19/23	6/29/23
Future directions	6/19/23	6/29/23
Comparison wi...	6/26/23	6/29/23
Submission	6/29/23	6/29/23
Bibliography	1/10/23	6/29/23
Logbook	1/10/23	6/29/23

Screen designs

Circle Attack

Play

Options

Exit

Options

Volume

Master

SFX

Music

Back

Life Money

Cost

Cost

Cost

Life Money

Game over

Score

New map

Retry

Menu

Cost

Cost

Cost
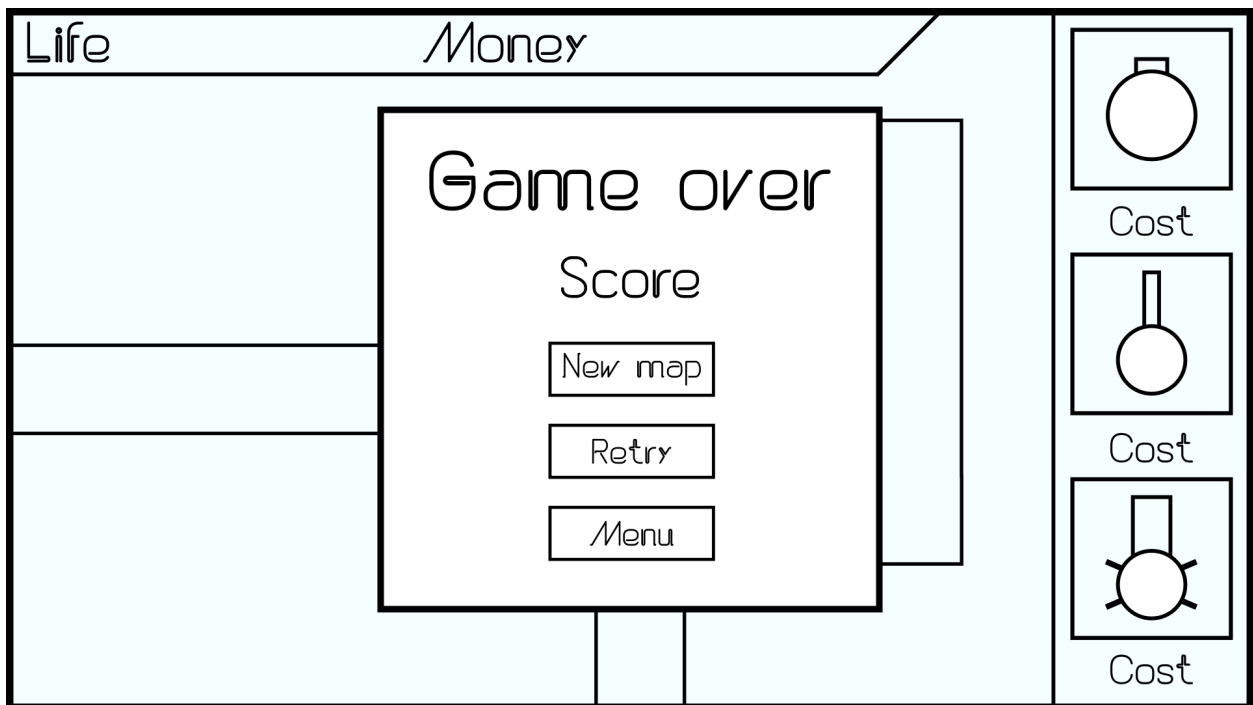
Storyboard
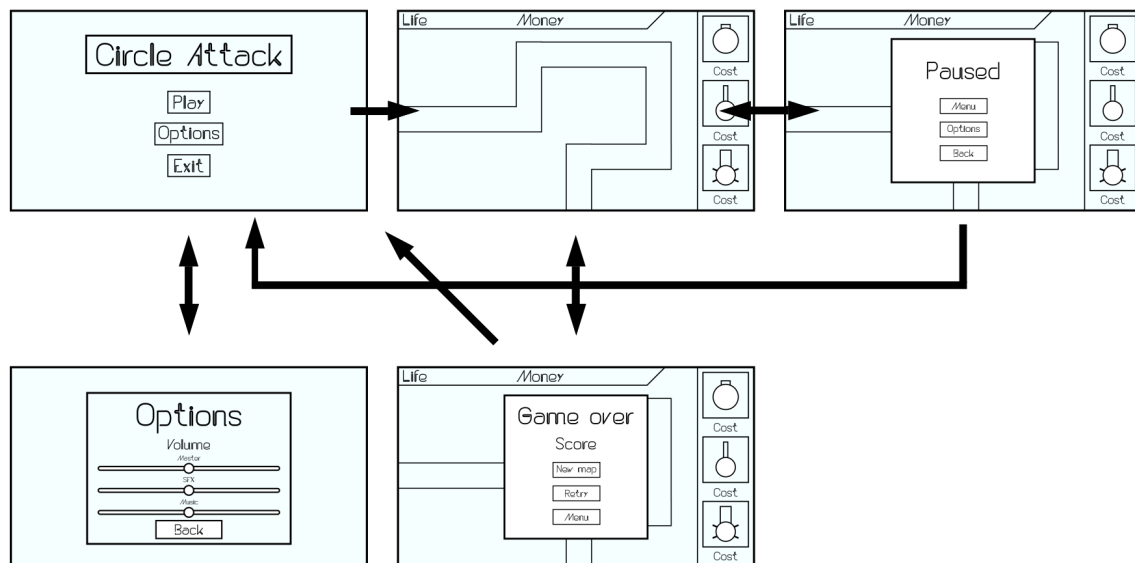
Context diagram
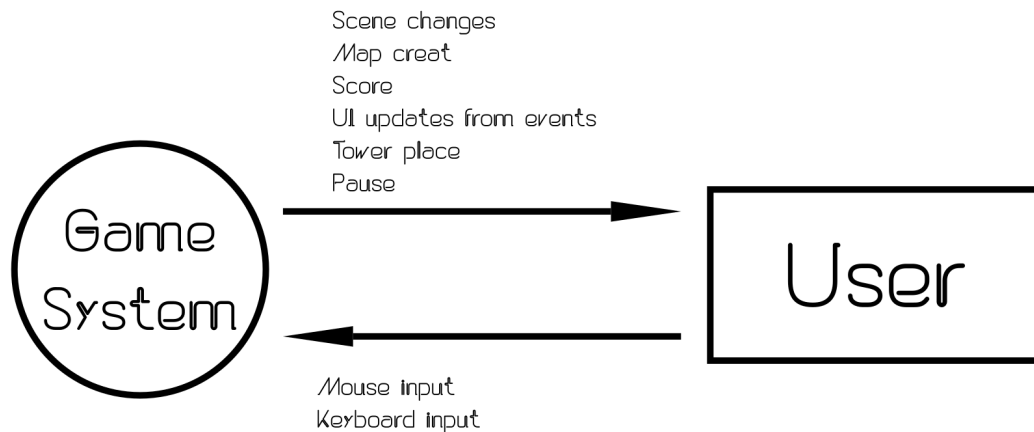
Discussion of selection of language to be used

I chose C# to program my game in. I chose C# because I am using the Unity game engine, which I have used in many other projects with C# and so I am the most familiar with it. There are many tutorials and resources available online and lots of materials dealing with my specific type of problems I will run into (Randomness & Generation). However, Unity also supports UnityScript, which is the JavaScript version of Unity. I avoided this as I am not making a webgame and I am also not too familiar with JavaScript, and many of the online tutorials use C# as opposed to UnityScript, so it would have been much harder for me.

As for the game engine, I chose Unity over other game engines because it is the most popular game engine with lots of resources and documentation, and it is great for small personal projects. Other game engines such as Godot are not as popular and as a result have less resources on it, and in general I am less familiar with it than I am with Unity.

Social and ethical considerations

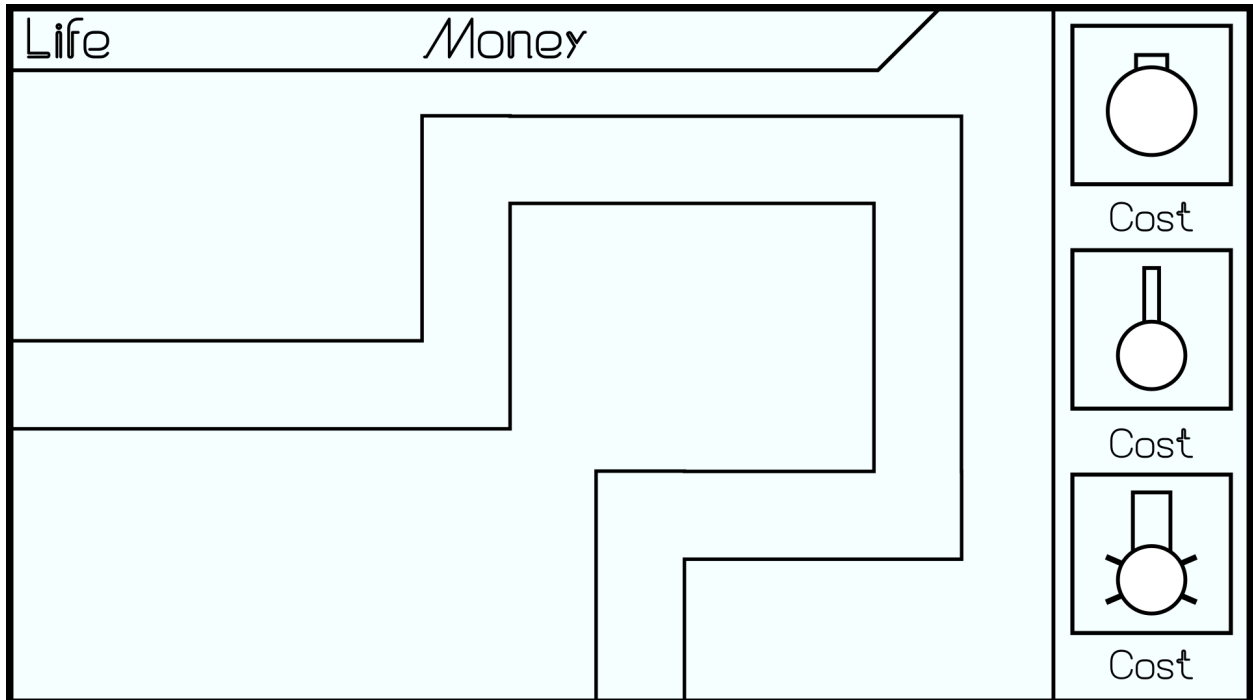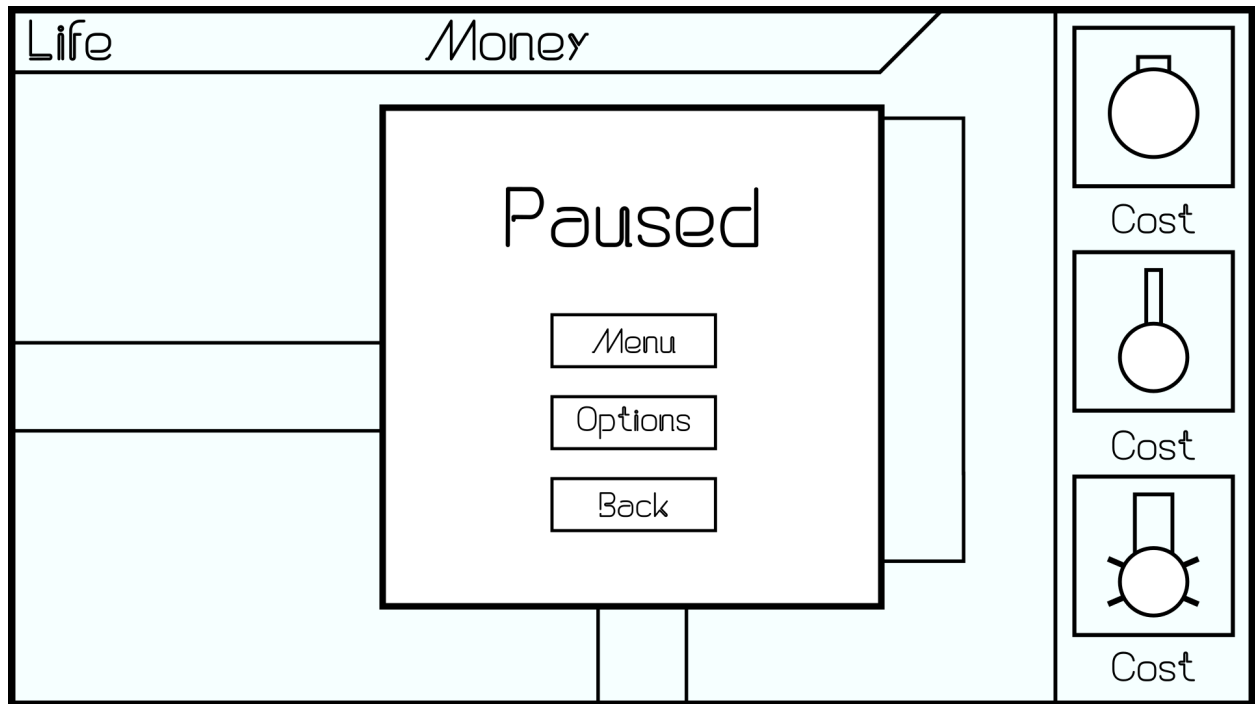Copyright - As is the case with many projects, a lot of assets are externally sourced. With these externally sourced assets, I must be careful to use assets that are clearly labeled with licences that grant permission of use, such as the CC (Creative Commons) licence.

Accessibility - People with disabilities or impairments regarding vision or hearing are not properly addressed as of right now, however in the future this may change with addition of a

high contrast mode. Most operating systems have a magnifier app that can magnify areas of the screen for the user, which could also help with vision impairments. My game should also be fairly easy to run on most computers used nowadays, as there is nothing that is heavily intensive on it.Unity can build the game for Mac, Linux, and Windows, so people can play the game regardless of OS.

Ergonomics - The game will have intuitive UX, with big buttons and very simple, easy to follow instructions, along with the inclusion of the user manual. Ergonomics should be a small concern with this game.

Part A

5 Functions or modules

Name	Description
Enemy (Module)	Contains necessary fields and methods for the functioning of the game, such as the DistanceLeft field which is used for the tower's targeting system in which it will target the frontmost enemy in its range (Minimum distance left)
Tower (Module)	Contains data such as the type of tower, range of the tower and price of the tower. This info will be made available to the user upon certain inputs such as hovering over a placed tower or visible on the screen at all times.
towerShoot	Script and function for the shooting mechanic of towers. Dependent on the tower type, so it will call the tower module and request the tower type from it.
EnemyInSight	Script and functions for determining the target for each tower, and rotating the tower + calling the shoot function
Menu (Module)	Handles the scene management and options from the main menu, options menu, and the main game. Also from the game over screen to the main menu

Pseudocode and system flowchart of a module

BEGIN EnemyInSight(Tower)

    VAR firstEnemyCollider

    LET ColliderArray() = Physics2D.OverlapCircle(
        Tower.Position
        Tower.Range
        EnemyLayer
)

    LET firstEnemyCollider = ColliderArray(1)

    FOR i = 2 to length of ColliderArray() STEP 1

```
            IF ColliderArray(i).GameObject.GetComponent<Enemy>().DistanceLeft <
    firstEnemyCollider.GameObject.GetComponent<Enemy>().DistanceLeft THEN
                firstEnemyCollider = ColliderArray(i)
        ENDIF
    NEXT i

    LET Tower.Rotation = atan(
            (FirstEnemy.position.x - Tower.position.x) / (FirstEnemy.position.y -
    Tower.position.y)
    )

    Tower.GetComponent<Shooting>().Shoot()

END EnemyInSight
```

```
BEGIN
EnemyInSight(Tower)
```

```
LET ColliderArray() =
Physics2D.OverlapCircle(
Tower.Position, Tower.Range
EnemyLayer)
```

```
LET firstEnemyCollider =
ColliderArray(1)
```

Is i less than or equal
to the length of the array?

no

yes

```
IF ColliderArray(i).GameObject.
GetComponent<Enemy>().DistanceLeft >
firstEnemyCollider.GameObject.
GetComponent<Enemy>().DistanceLeft THEN
```

yes

no

```
firstEnemyCollider =
ColliderArray(i)
```

```
Iterate
```

```
LET Tower.Rotation = atan(
(FirstEnemy.position.x -
Tower.position.x)
/
(FirstEnemy.position.y -
Tower.position.y)
```

```
Tower.GetComponent
<Shooting>().Shoot()
```

```
END
EnemyInSight(Tower)
```

IPO of main module (Game Client)

Input	Process	Output
Start game	Call generate map and generate path modules to create a random map	Randomised map, main game
Play round	Increment round counter by 1 Start sending enemies via the SendEnemy script	Round counter on screen goes up
Mouse input	Check if the mouse is hovering over a tower and handle accordingly; place towers down and call the shooting modules	Towers placed, UI update (Tower info)
Rounds played	Check if number of rounds is equal to the max round, if it is, end the game	Endgame screen, score, and menu buttons
Player Lives	If player lives is 0 or less, end the game	Game over screen

Structure chart of all modules

DFD

The diagram shows a Data Flow Diagram (DFD) with the following nodes and labelled data flows:

- **User** → (Mouse inputs) → **Game over**
- **User** → (Play game) → **Scene manager**
- **User** → (User options) → **store data**
- **User** → (Mouse inputs, Escape key) → **Game Client**
- **store data** → (User options) → **persistent data path**
- **Scene manager** → (New scene ID) → **Game Client**
- **Game over** ↔ (Replay? Menu?) **Game Client**
- **Game over** ↔ (Are all rounds over? Is health <=0?) **Game Client**
- **Generate map** → (Map data) → **Game Client**
- **Rounds** → (Round number) → **Send Enemies**
- **Game Client** ↔ (Did user play round?) **Rounds**
- **Rounds** → (Are all rounds over?) → **Game Client**
- **Game Client** ↔ (Tower position, Tower type) **Tower Placement**
- **Game Client** ↔ (Did enemy pass end?) **Tower Placement**
- **Tower Placement** ↔ (Angle towards enemy, First enemy) **Game Client**
- **Game Client** ↔ (Is health <=0?) **Health**
- **Tower Placement** → (Tower gameobject) → **Tower Shooting**
- **Tower Shooting** → (Angle towards enemy, First enemy) → **Tower Placement**
- **Tower Placement** ↔ (New player money) **Money Manager**
- **Money Manager** → (Tower cost, Player money) → **Tower Placement**

Files that will be used with respective data structures

My game will not use many files as I do not have a save game feature, the only files will be used for saving options. I have implemented these in a very very simple way. The only option I have that requires persistent data is the volume (Master, Music, and SFX)

Name	Type	Description
volumeData	.txt (The options are not stored in any special format)	Contains the different values for each volume slider on different lines E.g. 1 0.5 0.5 Meaning the master volume slider is max, the music slider

					is half, and the SFX slider is half		

Data Dictionary

Item	Type	Format	Bytes required for storage	Size for display	Description	Example	Validation
towerType	Int	N	1	1	An integer from 1-3 inclusive that defines the type of tower.	1	Between 1 and 3 inclusive
towerRange	Int	NNNN	2	4	Integer that defines the range that the tower can shoot towards	600	Has to be positive
healthPoints	Int	NNN	1	3	Tells the user how much health they have	100	
money	int	NNNNNN	4	6	Tells the user how much money they have	25565	Cannot be negative
round	int	NN	1	2	Tells the user what round of the game they are on	13	Between 1 and 30 inclusive
roundPlaying	boolean	X	1	1	Is used internally to store if the round is playing at the current point in time or not	true	
enemyType	int	N	1	1	Is used internally to check for the type of enemy and whether (There are 4 enemy types)	2	Has to be between 1-4
enemySpeed	int	NNN	2	3	Used to determine the speed that the enemies travel through the map	3	Cannot be negative
endScore	int	NNNNNN	3	6	Is determined by the amount of money spent and how many enemies were killed, along with the	501	

					type of enemy		
optionsData	string	XXXX XX	6	6	Appended to application.persistentData Path. Used for the persistent data storage of the values for volume	volume	

Platform/OS considerations & hardware implications

My software will not be very hard to run as there are not any intensive modules or methods being put into place, and the whole game itself is based off of a very simple concept. As for which operating systems my game will be available for, Unity supports building for windows, linux, macOS and Android, however Android requires additional steps such as scripts for floating joysticks and such.

However, my game won't need floating joysticks as there is no player movement involved, so Android support would be very easy to implement. I am planning to build for all 4 systems however I may consider not building for android later if the game is too hard to run on Android.

Minimum specs for Windows
- CPU: Intel Core i3 3210 | AMD A8 7600 APU or equivalent
- RAM: 4GB
- HDD: At least 128MB free
- GPU: Intel HD Graphics 4000 or AMD Radeon R5 series | NVIDIA GeForce 400 Series or AMD Radeon HD 7000 series
- Windows version: 64-bit Windows 7 or later
- Display: 1024x768 or better

Recommended specs for Windows
- CPU: Intel Core i5 4690 | AMD A10 7800 or equivalent
- RAM: 8GB
- HDD: At least 512MB free
- GPU: NVIDIA GeForce 700 Series | AMD Radeon Rx 200 Series
- Windows version: 64-bit Windows 10
- Display: 1280x720 or better

No network is needed since the game is not online

Check 2

Choice of UI items selected

For all navigation in the game, I have chosen to use simple buttons and sliders only. My game was simple and I knew this, so it was very easy to make an intuitive, easy to use UI.
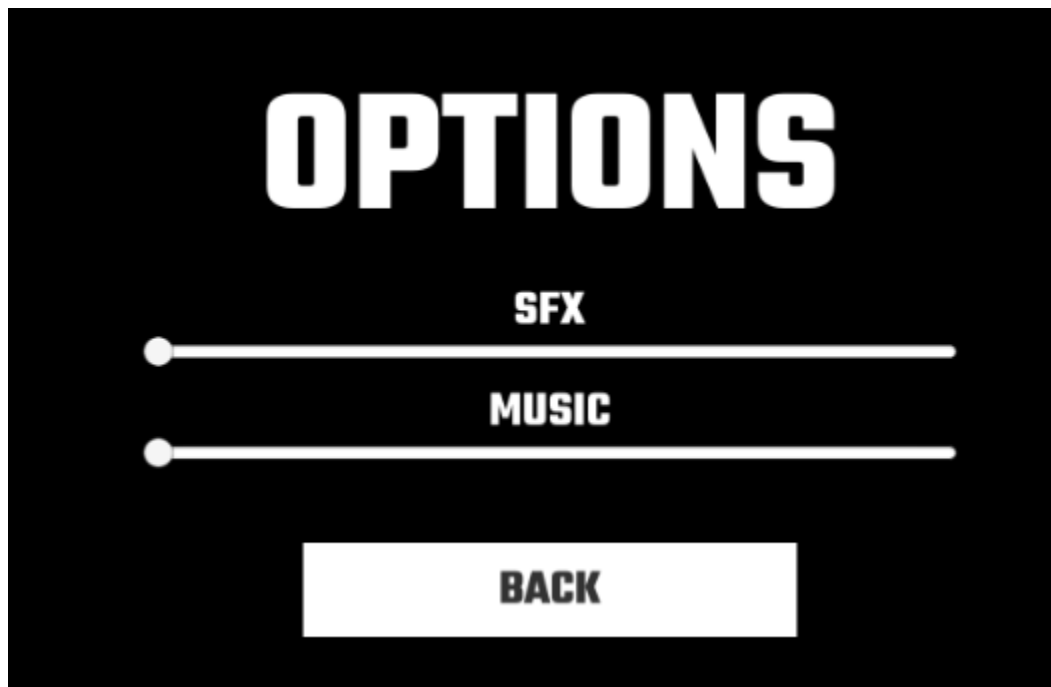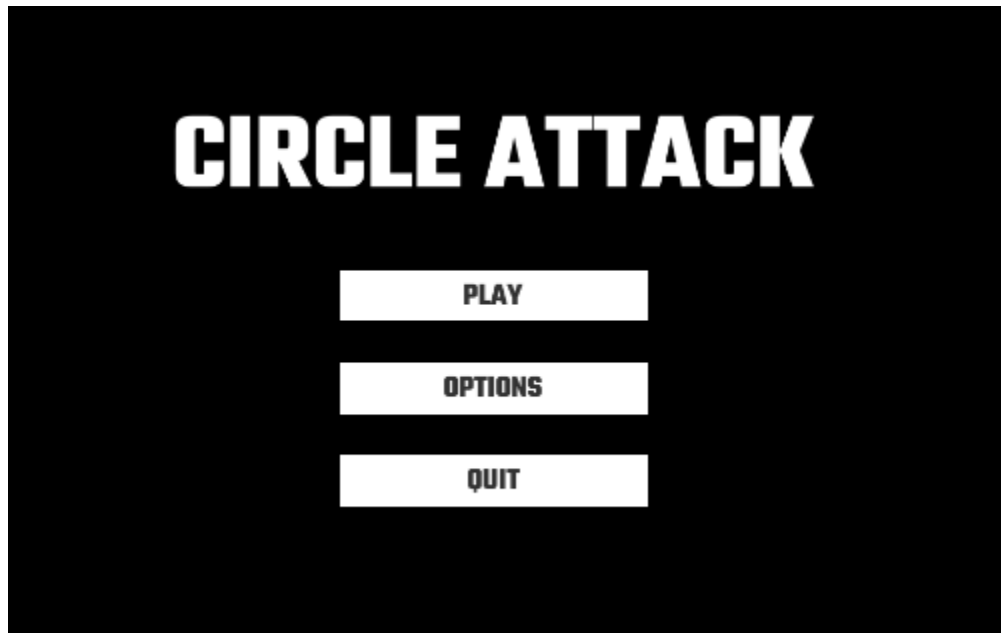
In the main menu, I use buttons to navigate throughout with easy to follow instructions, and the options menu have sliders that are very intuitive.

In the game, I used buttons for the tower placement and play round button, with sprite icons of each tower on the tower buying buttons. There is also a life and cash display in the top left.
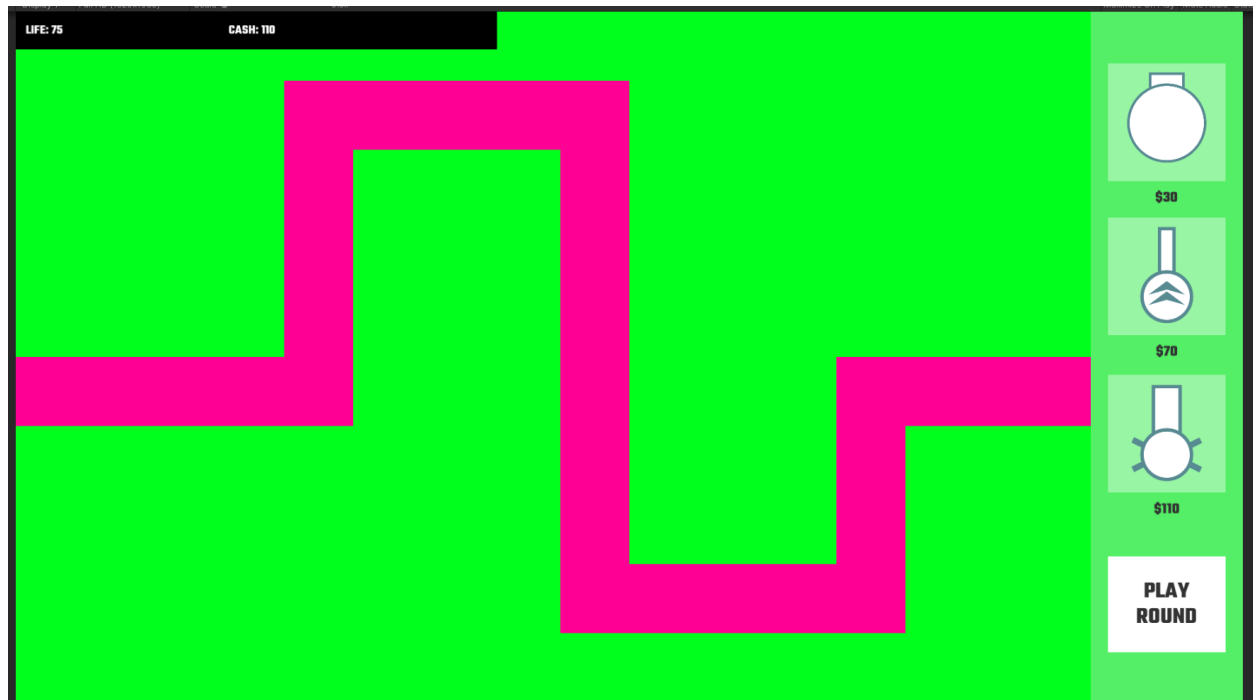
EBNF and railroad diagrams

while loop = while "(" <conditional statement> { &&|"||" <conditional statement> } ")"
"{"{<statement>;} "}"

conditional statement = <identifier>|<number> <comparison> <identifier>|<number>
comparison = == | != | <> | < | <= | > | >=

identifier = (<letter>|_) { <letter> | <digit> | _ }
number = <digit>{<digit>} | {<digit>}.<digit>{<digit>}f

digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
letter = a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

While loop

Conditional statement

Comparison

digit

0
1
2
3
4
5
6
7
8
9

letter

a
b
c
d
...
W
X
Y
Z

identifier

_

letter

letter

digit

_

number

digit

.

digit

f
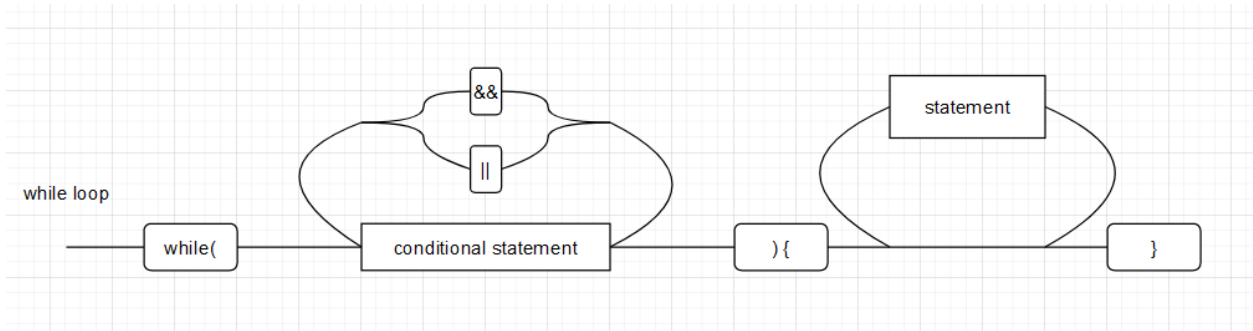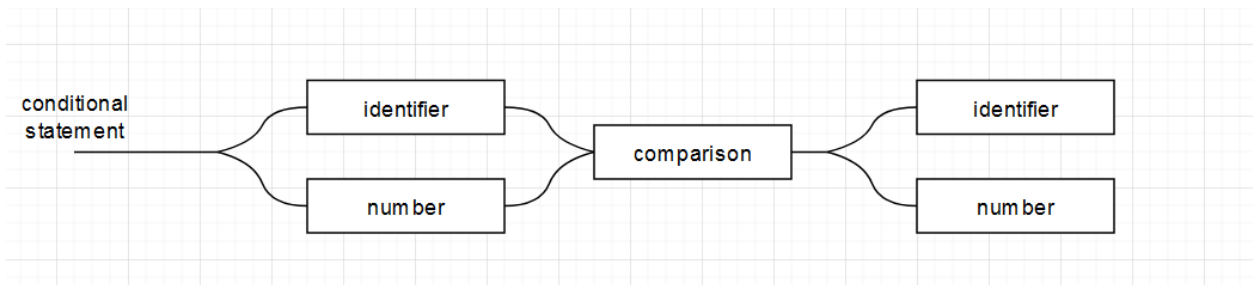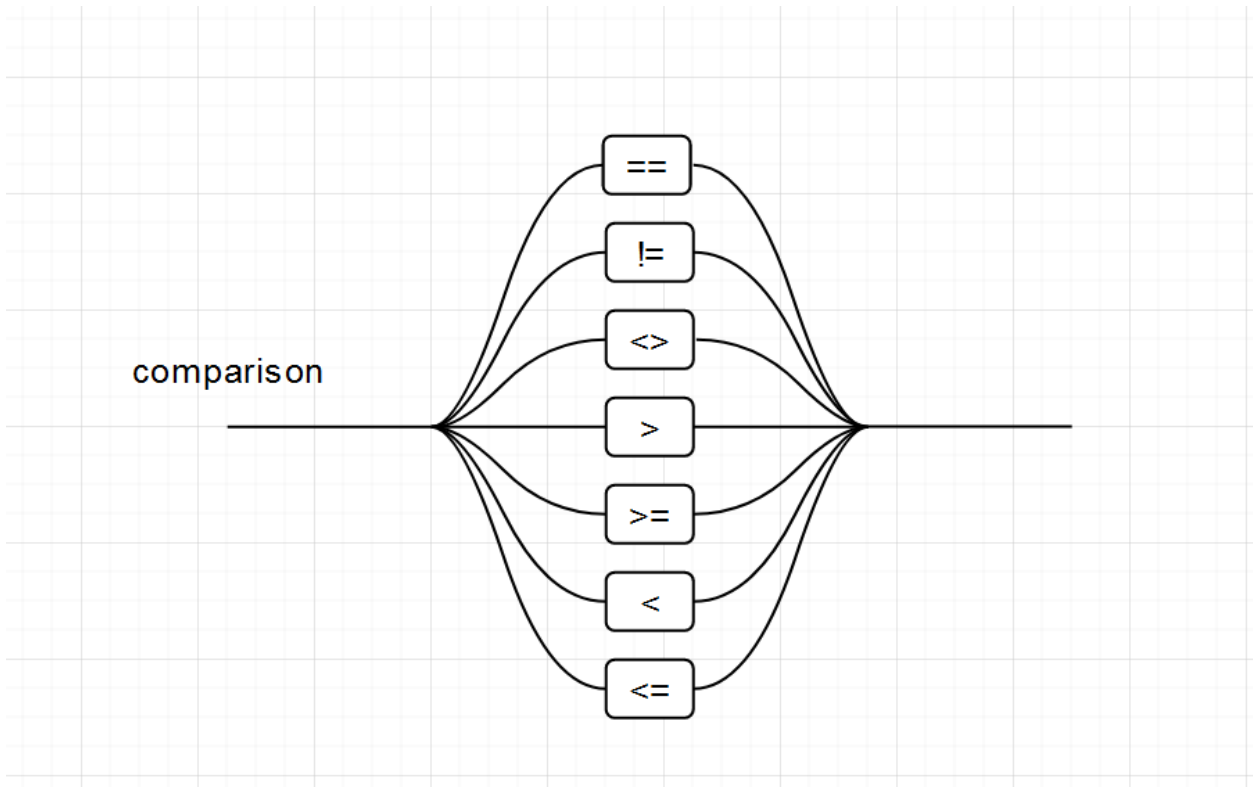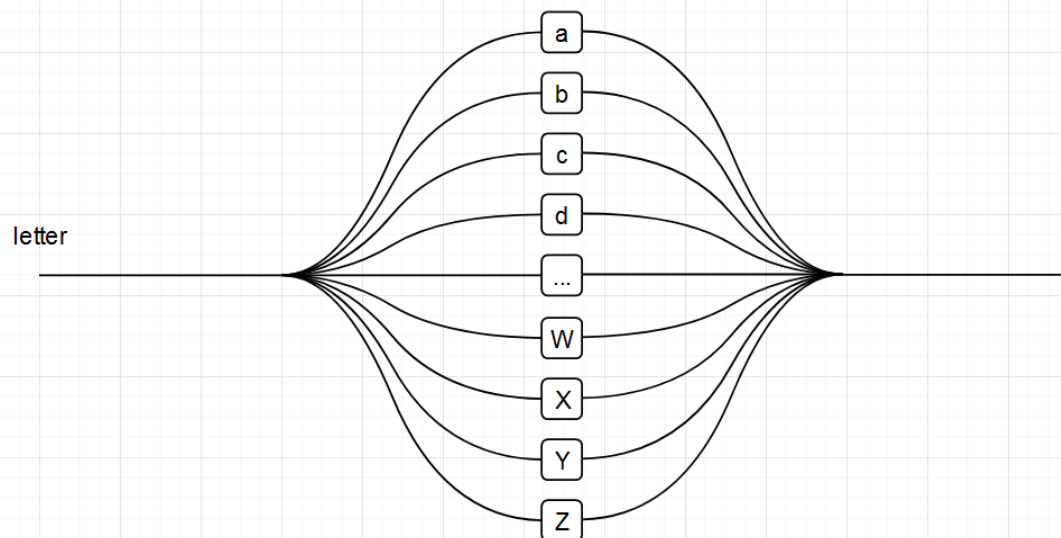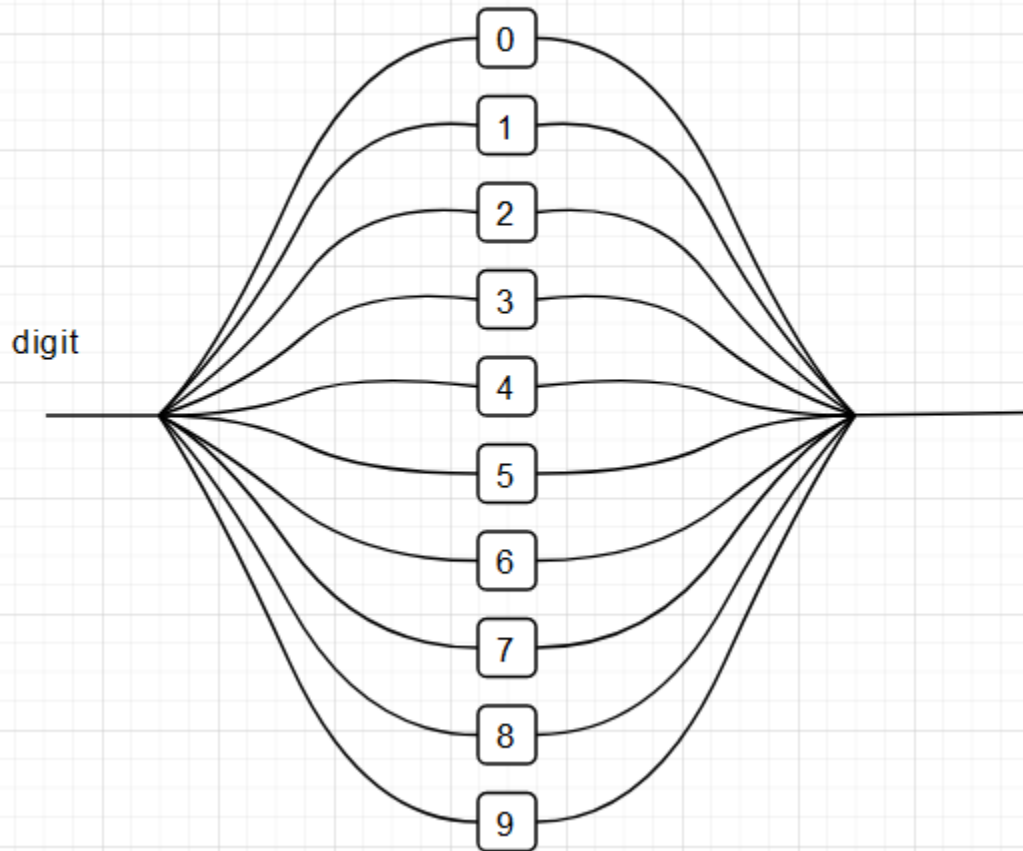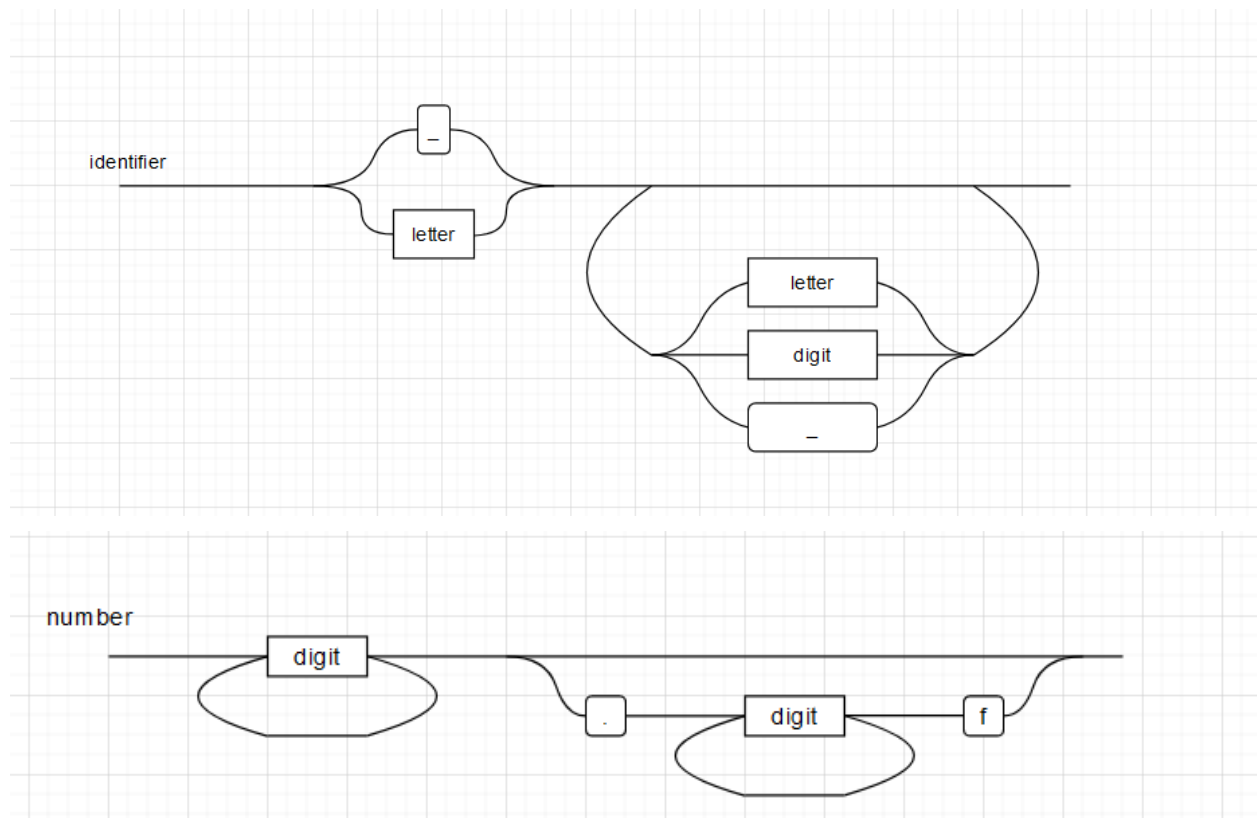
Test data

EnemyMovement.moveToPoints()

Point1	Point2	direction	Expectation	Observation	Match?
-9.5, -1	-8, -1	(1,0)	Moves 1.5 units right	Moved 1.5 units right	yes
-8, -1	-8, -4	(0, -1)	Moves 3 units down	Moved 3 units down	yes
-8, -4	3, -4	(1, 0)	Moves 11 units right	Moved 11 units right	yes
3, -4	3, -1	(0, 1)	Moves 3 units up	Moved 3 units up	yes
3, -1	-6, -1	(-1, 0)	Moves 9 units left	Moved 9 units left	yes

-6, -1	-6, 3	(0, 1)	Moves 4 units up	Moved 4 units up	yes
-6, 3	5, 3	(1, 0)	Moves 11 units right	Moved 11 units right	yes
5, 3	5, -1	(0, -1)	Moves 4 units down	Moved 4 units down	yes
5, -1	7, -1	(1, 0)	Moves 2 units right	Moved 2 units right	yes
7, -1	null	null	Game object destroys itself	Game object destroyed itself	yes

From this test data I am able to conclude that my enemy pathfinding script is able to navigate a complex map using a given set of points to turn at, and none of the points would be diagonal to each other so I did not need to check for this. However, there was a very rare occurrence where enemies skipped over a point, and this is due to the quantisation of the values, sometimes being large enough to skip over the boundaries for a point (.14 units), but this was very rare and I neglected it.

Error checking

Stubs

A stub is a dummy module that is called by another already written module. It allows the developer to know that the logic for that written module works correctly without having to wait until both modules are written. I used several stubs to test for logic errors in my program, such as when I was testing if my health script worked for the game over scene where it would access a method from another unwritten module named getFinalScore(), so I made a dummy module with the same name that logged "Code Reached" to the console, and using this I was able to tell if my logic was correct.

```
public void takeDamage(int damage) {

    if ( (health -= damage) <= 0 ) {

        lifeText.text = "LIFE: 0";

        gameObject.GetComponent<Score>().getFinalScore(false);
```

```
public void getFinalScore(bool won) {

    Debug.Log("Code Reached");

}
```

Flags

A flag is a boolean variable that gets toggled on and off and can be used within the module or in other modules to test if a certain condition is true or not. I used a flag in my tower placement script to solve and prevent errors within the script, and its main use is to see if a tower is actively being placed so it can make it follow the cursor, and implement the following logic on click and toggle the flag back for the next use.

```
if ( towerBeingPlaced ) {

    Debug.Log(removeZ(Camera.main.ScreenToWorldPoint(Input.mousePosition)));

    tower.transform.position = removeZ(Camera.main.ScreenToWorldPoint(Input.mousePosition));

    if ( Input.GetMouseButtonDown(0) ) {

        if ( checkIfValid(removeZ(Camera.main.ScreenToWorldPoint(Input.mousePosition))) ) {

            tower.GetComponent<TowerBehaviour>().enabled = true;
            tower.GetComponent<TowerClick>().enabled = true;
            playButton.enabled = true;
            towerBeingPlaced = false;
            tower = null;
            towerCollider = null;

        }

    } else if (Input.GetMouseButtonDown(1)) {

        gameObject.GetComponent<Cash>().updateCash(tower.GetComponent<TowerBehaviour>().value);
        Destroy(tower);
        towerBeingPlaced = false;
```
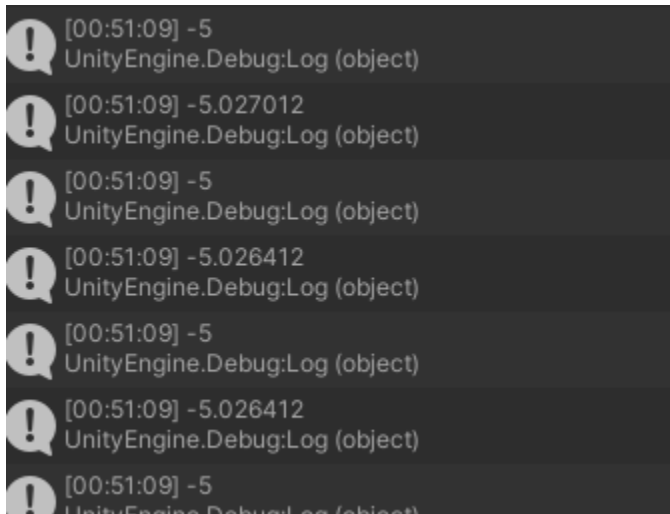
Debugging statements

I used debugging statements heavily throughout my code, mainly to expose the data of certain variables that I thought could be the cause of an error. Below is one of the examples where I exposed the point that the enemy would travel to and the enemy's position, to solve an issue where the enemy would not turn to the next point after passing over one.

```
void moveToPoints() {

    Debug.Log(transform.position.x);
    Debug.Log(roamingNodes[i].position.x);
```

```
[00:51:09] -5
UnityEngine.Debug:Log (object)

[00:51:09] -5.027012
UnityEngine.Debug:Log (object)

[00:51:09] -5
UnityEngine.Debug:Log (object)

[00:51:09] -5.026412
UnityEngine.Debug:Log (object)

[00:51:09] -5
UnityEngine.Debug:Log (object)

[00:51:09] -5.026412
UnityEngine.Debug:Log (object)

[00:51:09] -5
UnityEngine.Debug:Log (object)
```

Using this data I could see there was an imprecision in the value and implemented a bit of leniency on the point boundary rather than being exactly equal to each other.

```
if ( Mathf.Abs(roamingNodes[i].position.x - transform.position.x) < 0.07 && Mathf.Abs(roamingNodes[i].position.y - transform.position.y) < 0.0

    if ( i != roamingNodes.Count - 1) {

        i++;

        direction = new Vector2(roamingNodes[i].position.x - transform.position.x, roamingNodes[i].position.y - transform.position.y);
        direction = BetterNormalisedVector(direction.x, direction.y);
        rb.velocity = velocity * direction;

    } else {

        worldHandler.GetComponent<Life>().takeDamage(damage);
        Destroy(gameObject);

    }

}
```

Desk check

createNodes() function

Desk check 1: Trying to find out why an array out of bounds error occurred.

i	randomNessInt	wentVertical	randomDistance	nodeList[i] (x and y only)	nodeList.Count
0		true		(-1, 6)	1
	2				
			0		
				(-1, 6)	
0					1
	0				
		false	4		
				(3, 6)	
1					2
	0				
			3		
				(6, 6)	
2					3
	1				
		true	-5		
				(6, 1)	
3					4
	0				
		false	5		
				(11, 1)	
4					5

	2				
		true	1		
				(11, 2)	
5					6
	0				
		false	4		
				(15, 2)	
6					7
	2				
		true	0		
				(15, 2)	
7					8

*ArrayOutOfBounds error encountered

Nothing looks wrong with the output values of the desk check and everything makes sense aside from iteration 5-6, but I realised this was another logic error where the random integer could be equal to 0 as the Random.Range() function chooses from in between two integers which I had specified as -6 and 7. Seeing as nothing was wrong with the while loop, I then realised it had to be outside, and realised the issue was that the index I was using to access the last member of the list was greater than the boundary by 1 because the Count() returns the amount of elements which I would have to take 1 away from to access the last element.

Evidence of breakpoint, traces and single line stepping

Breakpoint

A breakpoint is where the program is stopped at a certain point in the code execution which allows the developer to analyse data at a specific time. I used breakpoints in my enemy movement script to see the value of the direction vector at certain points in the script, and discovered an error using this where the direction vector was wrong because the next point that the enemy was supposed to move to was later down the track and resulted in the enemy not turning where it should.

```
 93                         direction = new Vector2(roamingNodes[i].position.x - transform.position.x, roamingNodes[i].position.y
 94                         direction = BetterNormalisedVector(direction.x, direction.y);
 95                         rb.velocity = velocity * direction;
 96
 97          ⊟          } else {
 98
 99                         // Last node has been reached, destroy the enemy game object and damage the player
100                         worldHandler.GetComponent<Life>().takeDamage(damage);
101                         Destroy(gameObject);
```

100 %   ⊘ No issues found

Locals			▾ ⊓ ✕
Search (Ctrl+E) ⚲ ▾ ← → Search Depth: ▾			
Name	Value	Type	▲
▷ ▣ Active scene	{UnityEngine.SceneManagement.Scene}	UnityEngine.SceneManag...	
◢ ⬡ this	"Enemy1(Clone) (EnemyMove)"	EnemyMove	
▷ ⬡ base	"Enemy1(Clone) (EnemyMove)"	UnityEngine.MonoBehavi...	
⬡ damage	1	int	
▷ ⬡ direction	"(0.0, 1.0)"	UnityEngine.Vector2	
⬡ distanceLeft	31.10102	float	
⬡ i	2	int	
▷ ⬡ rb	"Enemy1(Clone) (UnityEngine.Rigidbody2D)"	UnityEngine.Rigidbody2D	

Call Stack

Search (Ctrl
Name
Void Ene
Void Ene

The value of the iterator was wrong because I had been looping through it every frame instead of checking every frame if the next point was reached. I solved this using an if statement each frame instead of the while loop.

Traces

Traces are tools that will track the values of variables during runtime and effectively let the developer see the update of the variable in real time. I used traces to track the position of the enemy and the position of the next node in the same script after I fixed the problem above using an if statement, however the enemy was still not turning. I had a thought that this was something to do with the imprecision of floating point numbers, and traced the variable to check.

Name	Value	Type
⬡ this.gameObject.transform.position.x	-4.060005	float
Add item to watch		

As I thought, there was a very small offset from the integer value created from floating point innaccuracy, and to solve this I implemented a bit of leniency so that it could be within a certain distance from the coordinate to start turning.

Single line stepping

Single line stepping is a tool that works when a breakpoint has been activated and allows the developer to go line-by-line through the code, helping to perform a pseudo desk check and understand where the logic takes the code and why.

```
 83
 84              // If the enemy position is within .07 of a node (accounting for floating point inaccuracy and/or quantisation)
 85              if ( Mathf.Abs(roamingNodes[i].position.x - transform.position.x) < 0.07 && Mathf.Abs(roamingNodes[i].position.y - transform.position.y) < 0.07 ) {
 86
 87                  // If there are still nodes left
 88                  if ( i != roamingNodes.Count - 1) {
 89
 90                      i++;
 91
 92                      // Increment i and get a new velocity vector
 93                      direction = new Vector2(roamingNodes[i].position.x - transform.position.x, roamingNodes[i].position.y - transform.position.y);
 94                      direction = BetterNormalisedVector(direction.x, direction.y);
 95                      rb.velocity = velocity * direction;
100 %      No issues found
```

Watch 1				Call Stack
Search (Ctrl+E)		Search Depth:		Search (Ctrl+E) ... View all Threads
Name	Value		Type	Name
this.gameObject.transform.position.x	-8.100001		float	Void EnemyMove:moveToPoints ()+0x1 at C:\Users\jaden\newest-revision\
Add item to watch				Void EnemyMove:Update ()+0x2 at C:\Users\jaden\newest-revision\cattack

I used single line stepping to make sure that the new if statement with the implemented leniency would actually evaluate to true when I needed it to, and I could see if it did by seeing what line it would skip to (as it would skip over lines because of the conditional).

Readability of code

Casing & naming

When naming variables in my code, I usually went with camelCase and tried to make the variable names as descriptive but concise as possible. Certain variable names became common and consistent throughout my program such as "worldManager" or "worldHandler" which was mainly used to access the singleton game object that had scripts that managed various aspects of the game.

```
public ContactFilter2D whatIsTower;
private CircleCollider2D towerCollider;
private Collider2D[] results = new Collider2D[10];
```

"whatIsX" is a common naming convention used for variables that can be used to filter for certain types of objects, in this case towers.

I continued to use camelCase for many of my methods too.

```
bool checkIfValid(Vector3 inputPos) {
```

```
public void sendNextRound(Button playButton) {
```

Whitespace

I spaced out certain sections within the same script as they felt like part of smaller sub-sections to me and should be separated in some way. This is particularly noticeable where I declare variables, as I declare variables and split them into groups where each group contains similarly purposed variables.

```
public float velocity;
public float distanceLeft;
public int damage;

public GameObject roamingNodesParent;
private List<Transform> roamingNodes = new List<Transform>();

private Rigidbody2D rb;
Vector2 direction;

public GameObject worldHandler;

int i;
```

```
public int round = -1;

public GameObject worldManager;

public GameObject panel;
public Button button1;
public Button button2;
public Button button3;
```

I also add many empty lines between statements in a similar fashion to how I group the variables; the similar statements would be in consecutive lines and when there is a new unrelated statement there would be an empty line.

```
void Awake() {

    rb = gameObject.GetComponent<Rigidbody2D>();

    roamingNodesParent = GameObject.Find("MapNodes");
    worldHandler = GameObject.Find("WorldHandler");

    for (int i = 0; i < roamingNodesParent.transform.childCount; i++) {

        roamingNodes.Add(roamingNodesParent.transform.GetChild(i));

    }

    for (int i = 0; i < roamingNodes.Count-1; i++) {

        distanceLeft += (roamingNodes[i+1].position - roamingNodes[i].position).magnitude;

    }

}
```

Indentation

Indentation is especially useful when it comes to seeing where pieces of code are meant to go, and if they are part of a function or a parameter. I consistently indented whenever I used if, for, or while statements, or when declaring functions.

```
bool checkIfValid(Vector3 inputPos) {

    Debug.Log(inputPos);

    Vector3 tempVector = new Vector3 ( inputPos.x + 9, inputPos.y + 5, 0 );
    bool placeable = true;

    if (towerCollider.OverlapCollider(whatIsTower, results) != 0) {

            placeable = false;

        } else {

            placeable = true;

        }

    if (tempVector.x >= 15) {

        placeable = false;

    }

    return placeable;

}
```

Comments

At certain points in my code I felt like the logic in my head was extremely hard to deduce from the code alone, and needed a comment alongside it to make sure if anyone was reading my code they would understand the thought process behind every part.

```
0    public Tilemap grass;
1    public Tilemap path;
2
3    // Flag variable and GameObject variable to handle the prefab and placement of towers
4    public bool towerBeingPlaced;
5    GameObject tower;
6
7    // Variables for the OverlapCollider() call
8    public ContactFilter2D whatIsTower; // Contains a ContactFilter2D that uses a mixed layermask of path and tower to filter for
9    private CircleCollider2D towerCollider;
0    private Collider2D[] results = new Collider2D[10];
1
2    // Setting the playbutton active or not
3    public Button playButton;
4
5
6    void Awake() {
```

Sometimes I felt that a larger explanation was needed.

```
Vector3 tempVector = new Vector3 ( inputPos.x + 9, inputPos.y + 5, 0 );
bool placeable = true;

if (towerCollider.OverlapCollider(whatIsTower, results) != 0) {

    /*/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
     *
     * The OverlapCollider() function returns an integer, specifically
     * the amount of colliders overlapping with the tower's collider
     * that follow the specified contact filter. In this case, the
     * contact filter filters for other towers or paths meaning if
     * there is a path or other tower colliding with the tower that is
     * about to be placed, render it unplaceable
     *
     / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /*/

    placeable = false;

} else {

    placeable = true;
```

Syntax, runtime and logic errors

Syntax errors

Syntax errors are errors where the syntax of the code does not match with the rules of the specified language. Syntax errors cause code to be unable to be compiled and are usually the result of something quite trivial such as a missing semicolon or parentheses. Because the code cannot be compiled with syntax errors, the developer will always be notified about them so it is easy to spot them, along with the error information that the compiler spits out which makes these kinds of errors very easy to solve most of the time.

[01:53:11] Assets\Scripts\MainGame\Management\TowerPlacement.cs(108,3): error CS1519: Invalid token 'return' in class, struct, or interface member declarati

[01:53:11] Assets\Scripts\MainGame\Management\TowerPlacement.cs(108,19): error CS1519: Invalid token ';' in class, struct, or interface member declaration

[01:53:11] Assets\Scripts\MainGame\Management\TowerPlacement.cs(182,1): error CS1022: Type or namespace definition, or end-of-file expected
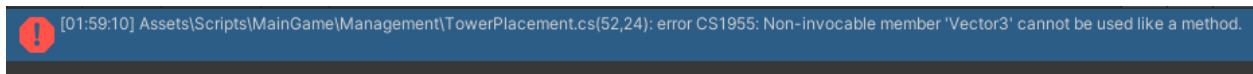
A cluster of syntax errors like this can often highlight that just one parentheses or semicolon is missing somewhere in the middle of the code, and it can sometimes be frustrating to find it.

```
if (tempVector.x >= 15)

    placeable = false;

}
```

The missing parentheses was in this if statement.

Runtime errors

Runtime errors are errors which stop the code during its execution stage. Common causes of runtime errors are impossible mathematical expressions (1/0, log(-1), sqrt(-1)), mixing up datatype assignments or array out of bounds errors. These kinds of errors also will produce error messages and stop the code from running, making them similarly easy to detect and again with the error info makes them handleable.

[01:59:10] Assets\Scripts\MainGame\Management\TowerPlacement.cs(52,24): error CS1955: Non-invocable member 'Vector3' cannot be used like a method.

```
Vector3 tempVector =  Vector3 ( inputPos.x + 9, inputPos.y + 5, 0 );
```

The vector was being constructed wrong and required the "new" keyword to denote a new instance of class Vector3.

```
Vector3 tempVector = new Vector3 ( inputPos.x + 9, inputPos.y + 5, 0 );
```

Logic errors

Out of the three errors, logic errors are the hardest to detect because they do not stop the code from being compiled and executed, rather they usually show up in the form of unexpected outcomes of the code and require various techniques and tools such as debugging statements and traces to solve.

A logic error I found in my program was with the tower placement method, where a placeable coordinate was marked as not placeable.
The first issue that I noticed was that the coordinate that was being checked was wrong because there was an offset of 9 and 5 due to the camera's origin being at the centre while the screen's origin being at the bottom left corner.
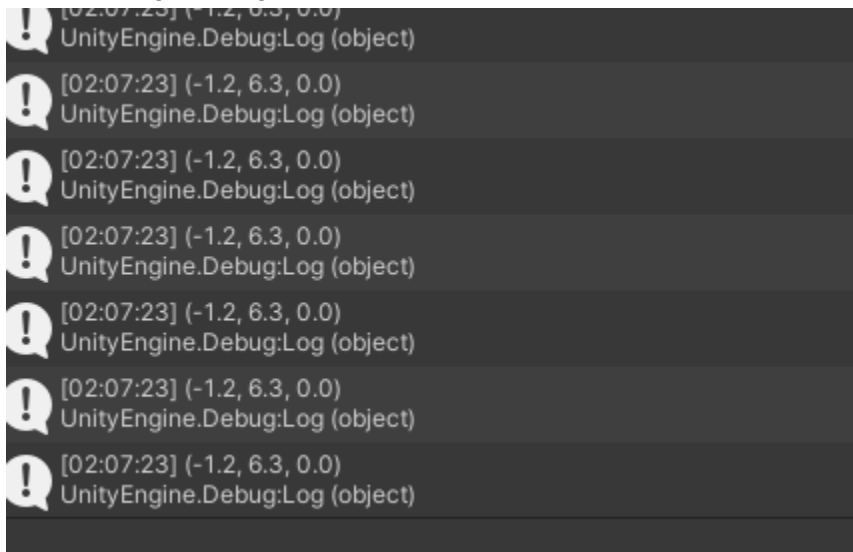
I found this by exposing the coordinate variable and checking its value and seeing it did not line up with the coordinate grid I was using.

The second issue I noticed was that the coordinate was not considered placeable only because one of the coordinates was not placeable (e.g. the x was within 1 unit but the y was far enough away; this would still be considered placeable). From this I realised I had to use a multiconditional AND rather than an OR.

Part B

Efficiency and elegance of code

I think my code started off very messy but as I progressed further into my project I found myself improving upon the efficiency of the code, finding faster ways to compute things and inbuilt methods that I could use.
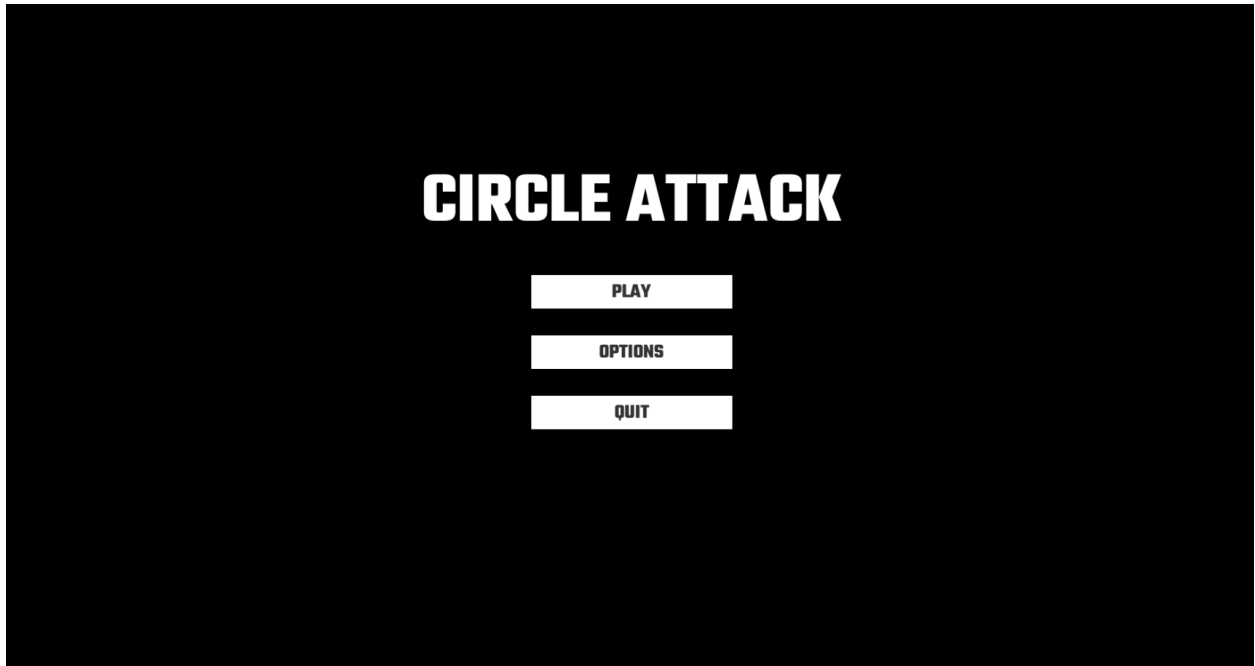
An example of this is where in my tower placement script, I needed to check if the tower was in a placeable coordinate to actually place it down. So originally, I would have to first create two lists which had the coordinates all "placeable" positions, and I created these lists using two for loops to go through the whole grid. I then had to take in the position of the mouse and convert it to a world point and check if this coordinate was within a certain boundary from the path and if it was, set it to not be placeable so the tower could not be placed on the path.

However, I had an error after this where I could place towers on top of each other. I knew how to fix this and started to work on it, but I realised that I could have used this new method that would give me an array or list with all the colliders that overlapped with the tower's collider, and all I needed to do was give it a filter (in this case a filter that detected collision between other towers and the path) and then check if the array had anything in it, if not, I knew there were no path tiles or other towers colliding with the tower being placed, and I could place the tower down, condensing a function that was maybe near one hundred lines to around 20~.
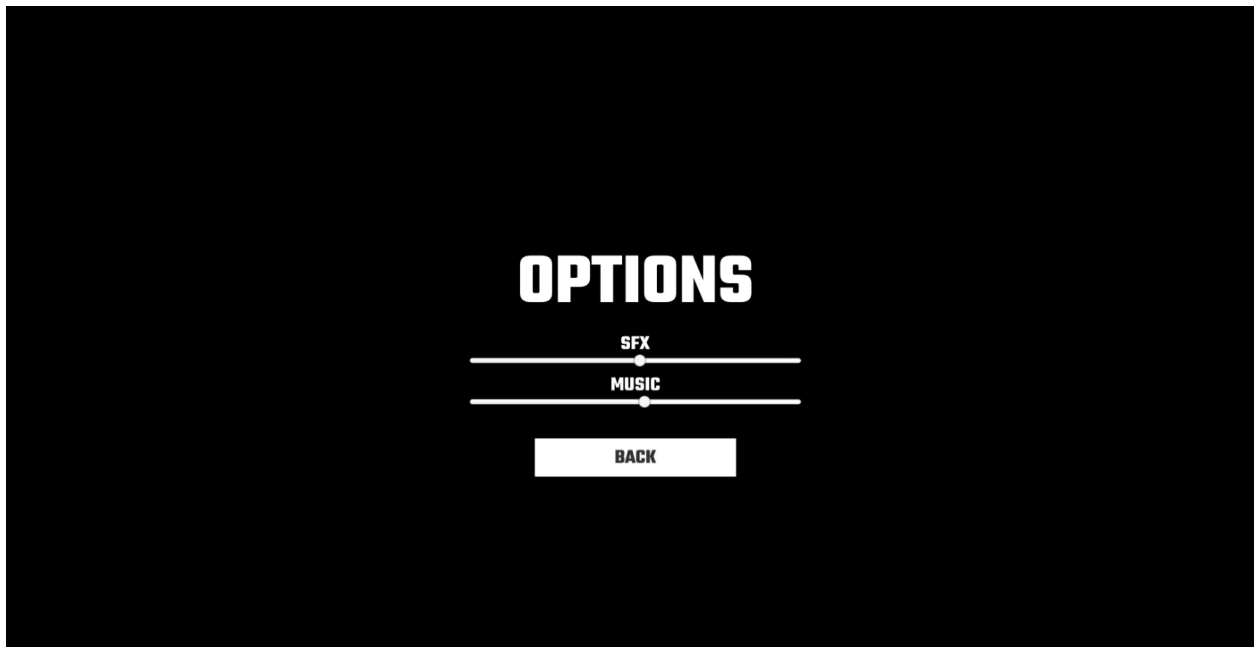
```
34      bool checkIfValid(Vector3 inputPos) {
35
36          Vector3 tempVector = new Vector3 ( inputPos.x + 9, inputPos.y + 5, 0 );
37          bool placeable = true;
38
39          if (towerCollider.OverlapCollider(whatIsTower, results) != 0) {
40
41              /*/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
42               *
43               * The OverlapCollider() function returns an integer, specifically
44               * the amount of colliders overlapping with the tower's collider
45               * that follow the specified contact filter. In this case, the
46               * contact filter filters for other towers or paths meaning if
47               * there is a path or other tower colliding with the tower that is
48               * about to be placed, render it unplaceable
49               *
50               / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /*/
51
52              placeable = false;
53
54          } else {
55
56              placeable = true;
57
58          }
59
60          if (tempVector.x >= 15) { // x >= 15 is covered by the tower placement UI, so this place should not be placeable
61
62              placeable = false;
63
64          }
65
66          return placeable;
67
68      }
```
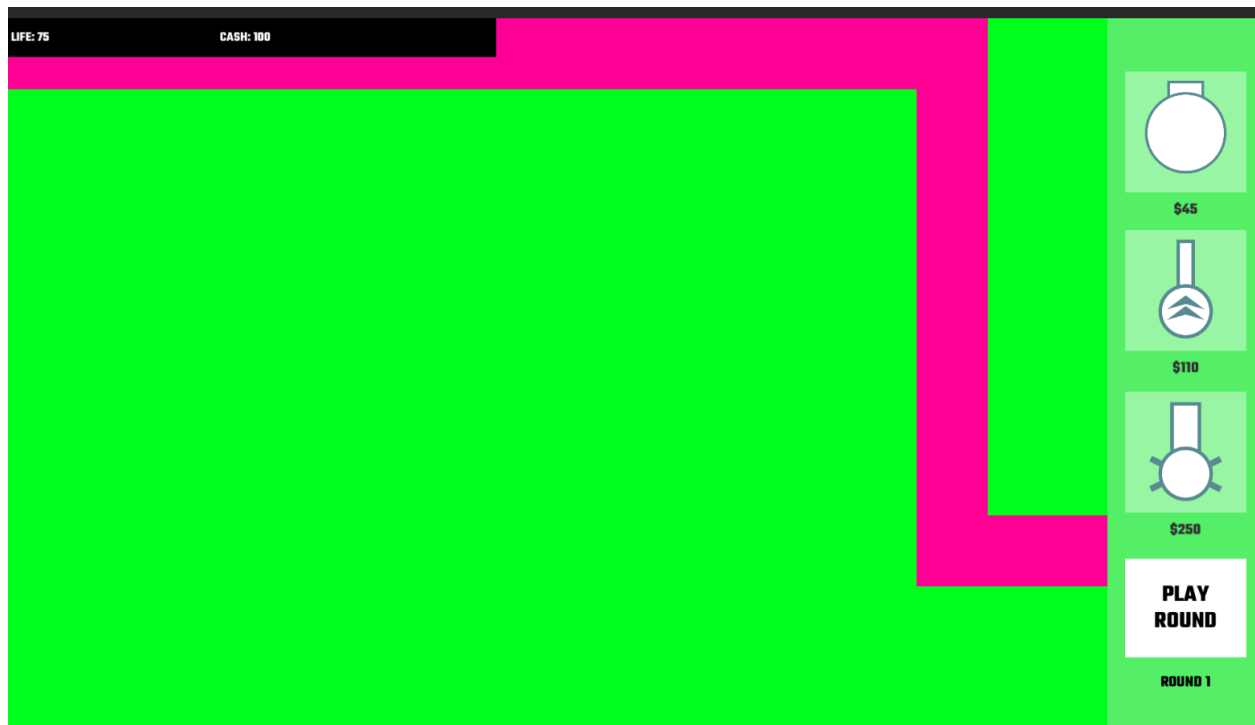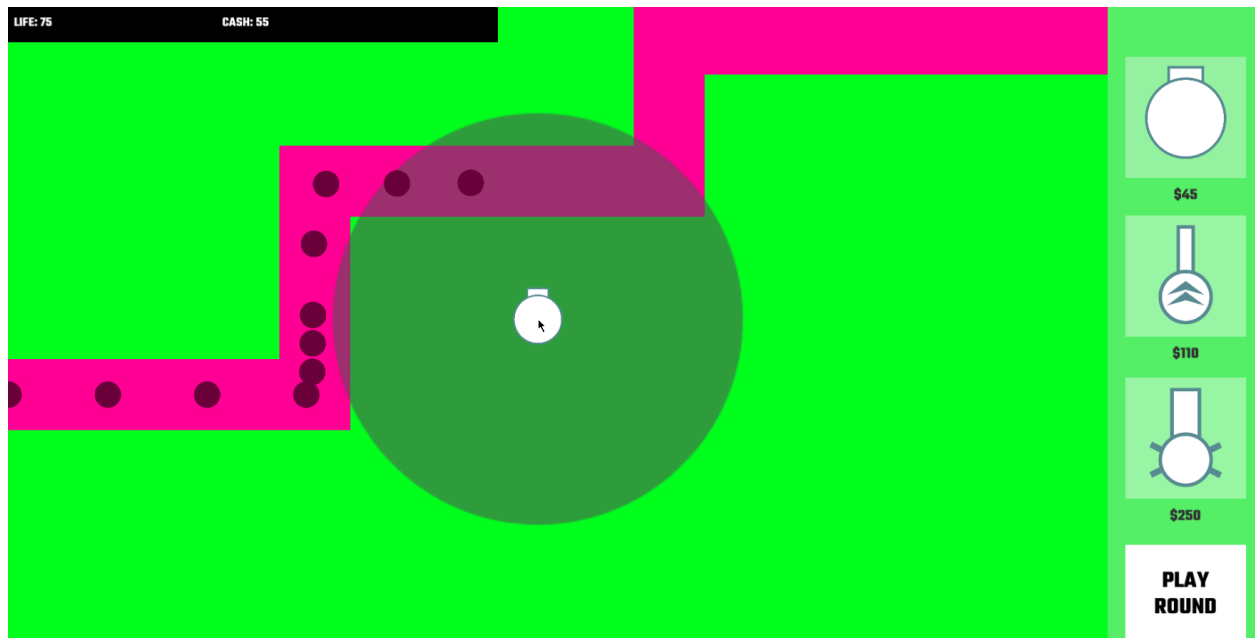
Refined UI
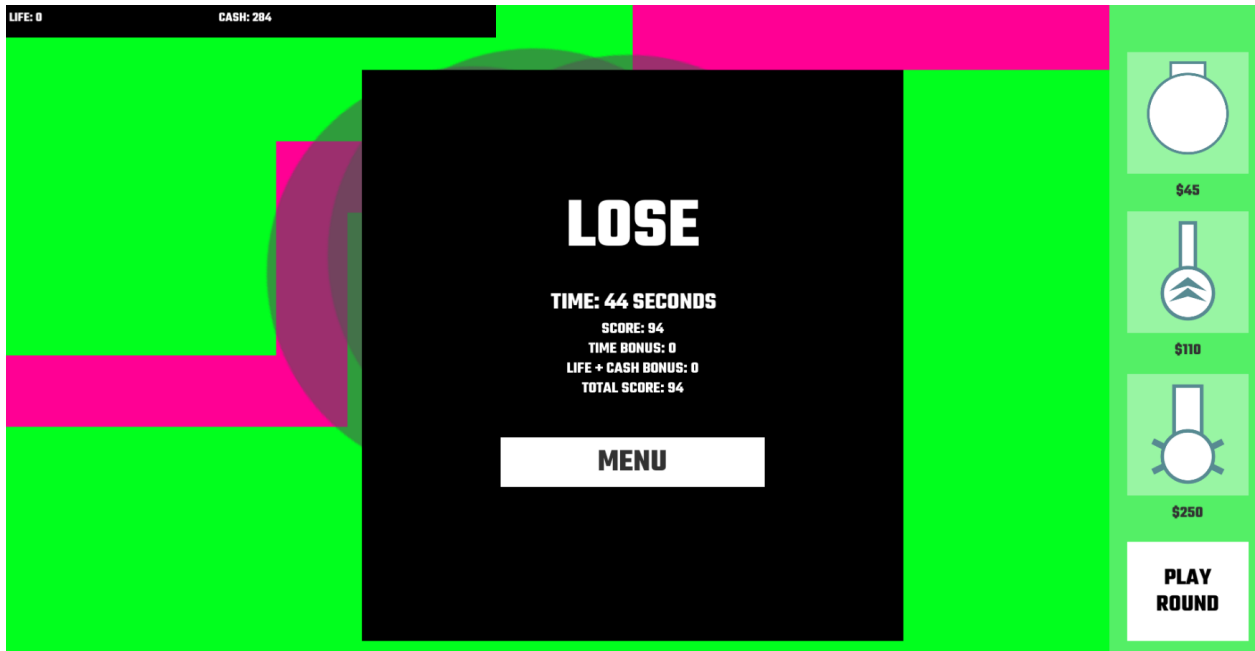
Main menu

Options menu

In game
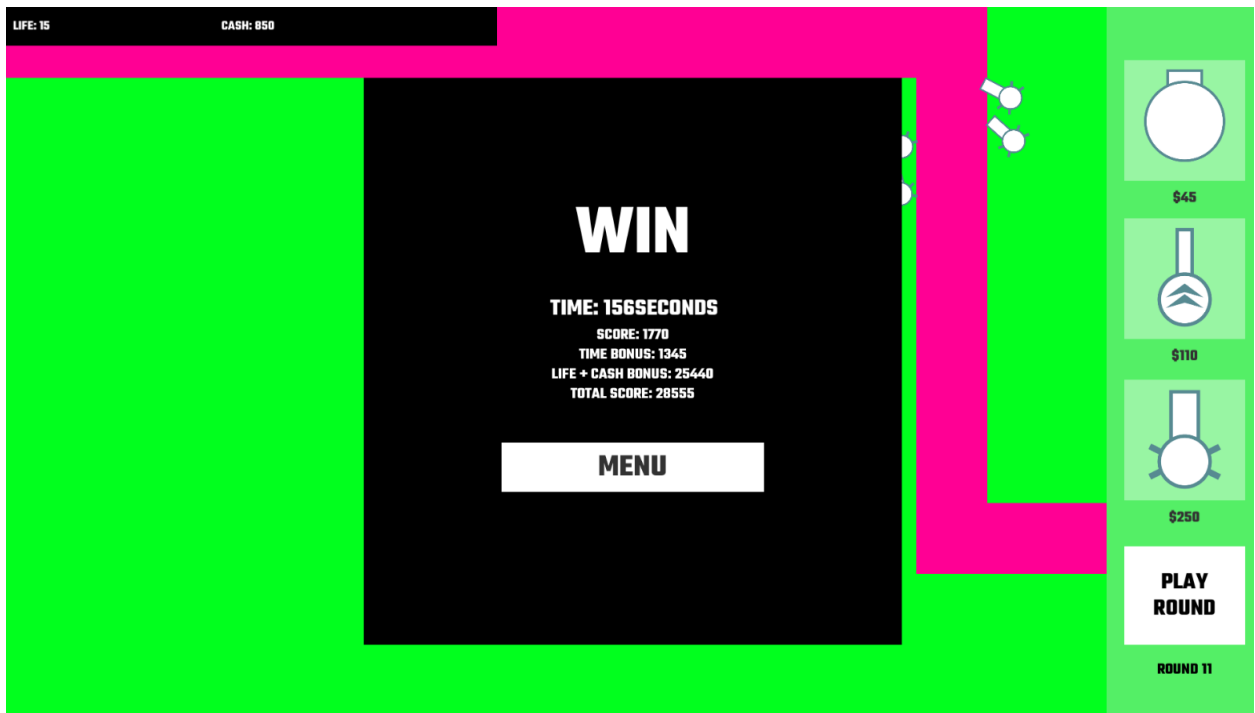

Tower being placed / range overlay

Game over screen

LIFE: 0   CASH: 284

LOSE

TIME: 44 SECONDS
SCORE: 94
TIME BONUS: 0
LIFE + CASH BONUS: 0
TOTAL SCORE: 94

MENU

$45

$110

$250

PLAY ROUND

Win

LIFE: 15   CASH: 850

WIN

TIME: 156SECONDS
SCORE: 1770
TIME BONUS: 1345
LIFE + CASH BONUS: 25440
TOTAL SCORE: 28555

MENU

$45

$110

$250

PLAY ROUND

ROUND 11

Suitable CASE tools

Git

Git is a distributed version control system and tracks changes in computer files, and is mainly used for working collaboratively in programming but can also be used as a version control system on a program for yourself. A distributed version control system is a form of version control that mirrors the complete codebase and its history on every developer's system. It uses a P2P approach to version control rather than a client-server approach.

The main advantage of Git is that it promotes a feature branch workflow, where you can create a new, isolated environment for each new feature that you or anyone else would want to implement into the main program, and merge the branches to the main branch later. This is a very common practice in large scale production.

In my project, I created a private repository with all my game assets and folders, and a public repository for my final submission. The private repository helped to keep my code up to date and let me work on it offline or online, and more importantly let me work across multiple computers, through commits to the repository and cloning on multiple devices.

GitHub

GitHub is a cloud-based platform and webservice for development using Git and allows developers to manage their code and also store it on a server and across devices. It extends Git, adding features such as access control, bug tracking and continuous integration.

I have two repositories, one private with all my Unity assets so I can work across computers and a separate public one for submission at https://github.com/collab112/circle-attack-2.

Catering for possible changes in user requirements

User requirements changes can come as a result of just time, as over the years new technology and new software will be developed, and occasionally users may have complaints or features that they would like implemented. As the developer I would have to take these suggestions into account and think about how to implement them into the program.

Because I am using GitHub, issues and commit requests can be sent directly to the developer, with a small description on what they were trying to achieve with the commit. Requests should be considered very carefully and branched workflow should be considered here.

User feedback could be given using GitHub issues, and as the developer I would need to research the issue, going through the SDLC of first defining and understanding the problem(s) through use of tools such as desk checks and CASE tools, planning a solution using mind maps and consulting my structure diagram and maybe DFD to find where exactly I would need to

change. I would then work on actually developing the code to solve this problem, followed by testing to make sure the issue was solved before deployment and maintenance.

User manual & Installation manual

Both of these are included in the GitHub repository

What have I learnt

Throughout my project I have learnt and improved on my knowledge of how to use various parts of the Unity game engine but also skills in project management, especially feasibility.

My first language that I learnt was C, and so object oriented programming was not my best area, but I did have some experience in it before this project. I felt I learnt a lot more about object oriented programming and the principles behind it after writing many many scripts in C#. The syntax is not too different, but coming from C where the main program was just in one big file, and suddenly having files everywhere was quite confusing and a little overwhelming to start off but I got used to it quite quickly.

This was the first time I had tried to generate a random map, and at first I tried to implement a set of rules as to where the coordinates could go but I constantly worried about a horrendous worst case scenario where the map would just go all over the place and be unplayable. I definitely had to learn to think up new solutions, and how to know when something is not working and requires a rethink.

I constantly thought that my program was quite big and a little out of scope for me, but I thought this would be counteracted by me just putting in a little extra effort. However, in the end, I gradually realised that my scope was actually not really too big or too small, in fact it was right where I wanted it to be, as it was relatively simple, but definitely not tiny, and had various challenges throughout.

Future directions

I think the end product has achieved what I wanted with my project.

Things I could definitely improve on is the balancing of the game, the assets and the UI. The texture assets are simple but a bit too simple, and lack any sort of flair. I also feel the game is a bit too easy where it is right now, but better balancing would require much more time with experimentation and testing. The UI is simple but the simplicity makes the game feel quite empty right now.

Perhaps in the future I may also look to enlarge the scope of the project as a whole, adding a lot more new towers, enemies, having a much larger pool of rounds to choose from and overall more things in general, because right now I feel the game is where I wanted it to be, but also kind of empty. Enlarging the scope would require me to rewrite some scripts as more of a template class that other objects can inherit from directly to make it easier to make new towers or enemy types.

Comparison with original specification

User

Specification	Met?
Simple and intuitive UX	Yes, there are intuitive and easy to follow buttons everywhere in the game, and controls follow common conventional control schemes
Simple clean graphics	To an extent, I feel like my graphics are very simple and clean, but also my graphics are very plain.
Not hard to run	Yes, the game is very easy to run
Consistent UX	Yes, UX is consistent throughout, using the same white and black scheme in most UI elements except for the tower panel. The text font stays the same throughout all scenes.
Little to no bugs	Yes, I feel like I eliminated most of the bugs within the game.

Developer

Specification	Met?
Appropriate datatypes	Yes, each variable has a meaningful and purposeful datatype and there are reasons why I chose specific types over others, such as lists over arrays because I do not need to initialise an array with a max index already.
Structured Approach	No, I did not use a structured approach
Detailed, verbose documentation	Yes, I put a lot of effort into both intrinsic, internal and written documentation on my code and the project as a whole

Logbook

Date of Entry	Description of task	Problems encountered	What I need to do	Status
10-Jan-2023	Created gantt chart	Deciding what software to use, and trying to use GanttProject for the first time, however in the end I got it working with a completed gantt chart.		Done
12-Jan-2023	Started documentation, created subheadings and started on problem and design specifications			Ongoing
20-Jan-2023	Started project and prototype/testing scene	I have not used the Unity 2D mode much, so I need to learn about the 2D datatypes as there are different datatypes for 3D and 2D worlds (Such as Vector2 for 2D and Vector3 for 3D). Also most 3D methods have a 2D counterpart.	Research the 2D methods and classes and become familiar with them.	Ongoing
22-Jan-2023	Researched random generation on tilemaps	As I have not dealt with this before, at first it looks very overwhelming.	I will need to do a lot more research and experimentation/testing with it to	Ongoing

		There is a lot of new material for me to look at.	get to know the methods, concepts and classes.	
23-Jan-2023	Looked through previous projects and online to find reusable code and modules	Will need to adapt code for new requirements	Make slight alterations in code so it matches with my project and not the previous one	Done
25-Jan-2023	Searched around on Unity script reference for tilemap	Not much	Implement the random generator with the SetTile method	Done
27-Jan-2023	Started work on tower and placement modules. Looked on YouTube to find random generation tutorials	A few bugs were present in the testing of the modules, such as the tower placement not displaying the overlay	Fix bugs through extensive testing and research on methods, finding exactly what they do and what is causing it to not function properly	Done
3-Feb-2023	Created git repo to manage source code of the project across machines	None	None	Done
5-Feb-2023	Tidied project folder	None	None	Done
6-Feb-2023	Tested all modules as a whole to see if they work with each other, e.g. tower placement and tower overlay.	Occasionally ran into a few errors between modules	Was not too much of a problem, fixed small issues in the code quite easily	Done
11-Feb-2023	Worked on tower modules - Made a template module which I can use for each	None right now, but there may be some fields or methods I will need to	Make sure to keep the class up to date	Ongoing

	tower	implement in the future.		
14-Feb-2023	Left test scene; created main menu and main game scene	None right now	Create scene flow - easily doable with Unity.SceneMag er	Done
15-Feb-2023	Started random generation module using tilemap	Need to either find an appropriate tileset to use or make my own Random generation module is not working as intended as of right now	Fix random generation and find placeholder tileset	Ongoing
17-Feb-2023	Refined documentation and various parts of it	None	Keep documentation up to date	Ongoing
20-Feb-2023	Moved project to different Unity version for compatibility with other machine	Had to create new git repo and push everything back into it again with new git files	Back up the old version and new version	Done
22-Feb-2023	Started shooting modules	Bug where the tower would only rotate towards the enemy once even though it was in the range throughout multiple frames. Tested using a stub piece of code for moving an enemy across a field	Make sure the shooting function is called periodically and at a fixed rate so it is consistently updating	Done
23-Feb-2023	Created more UI elements such	None	Work on map generation	Done

	as the top bar with health and money			
25-Feb-2023	Filled out parts of documentation	None	None	Done
26-Feb-2023	Started working on a prototype map for use with enemy movement testing	Enemies could not successfully follow the path, reason not known	Find out why enemies cannot follow the path and fix it	Ongoing
28-Feb-2023	Stopped with enemy movement and path following for now, moved onto prototype shooting with different tower types and projectiles	None	None	Done
2-Mar-2023	Refined documentation and updated various UI elements of the game, imported a few packages and assets for UI	None	None	Done
5-Mar-2023	Started health system, need to implement enemies taking away health and that requires me to implement an endpoint for the path	I do not have an endpoint system for the path yet, and it will be very troublesome to implement with the randomsied path	Find a way to implement an endpoint with the randomised path. Create a set of rules that will make sure that no bugs happen	Ongoing
7-Mar-2023	Filled out rest of documentation	None	None	Done
8-Mar-2023	Decided to try a new way of	Many errors within my script	Fix script	Ongoing

	implementing a random map where nodes are generated and can only move forward, down or up, making it easier to implement a random path that always starts on the left and easy to test for boundaries/impossible cases	and logic, requires lots of debugging		
16-Mar-2023	Stopped working on the randomness for a while and focused on the main game using a map that I had made manually so I could test modules such as enemy movement	Enemies would not turn when needed to and would walk off the path constantly	Figure out what was wrong with script, using debug statements	Ongoing
20-Mar-2023	Found the bug on the enemy movement script and fixed it	The enemy was running over the position specified and not matching it due to floating point and quantisation errors	Implement leniency into the check	Done
26-Mar-2023	Because of the new random path generator, the end point system would now be relatively simple once I fixed the map generation	None	Implement the ending point and what would happen should an enemy reach the end	Ongoing

6-Apr-2023	Wrote functions that would be called upon reaching the end of the path such as taking damage and updating cash	None	None	Done
7-Apr-2023	Created a pause menu script and UI for the pause menu. This was easy as I had done this multiple times before	None	None	Done
13-Apr-2023	Worked on making the sprites and buttons and tower prefabs in preparation for tower placement module	Many errors with tower placement, such as valid positions not being counted as valid	Find out why errors were happening and think of a solution	Ongoing
18-Apr-2023	Fixed many of the errors in tower placement, and did some testing on the module to make sure that it would work with the expected output	None	None	Done
20-Apr-2023	Found a much more effective and elegant way to check if a position was placeable and implemented it into my module	None	None	Done
26-Apr-2023	Started to work on the game as a whole more,	None	None	Done

	bringing modules together and such. Created a script for the final score of the player and started to weave modules together and call upon one other to progress the game			
1-May-2023	An issue where if there was no life left, the script was supposed to set the timeScale to 0 (pause everything in the game) but this didn't seem to run, and my lose function continued to get called repeatedly	The statement that was meant to stop everything in the scene did not run.	Think of a new way to stop the game over function from being called repeatedly	Ongoing
6-May-2023	I one time resized my window and realised certain UI elements broke at certain sizes.	At small sizes, UI elements would stretch and	Use the anchors to make the UI elements stretch in proportion to the monitor resolution	Done
10-May-2023	I decided to destroy all enemy gameobjects should the player's health get to 0 so that they could not cross the end and repeatedly call the lose function.	None	None	Done

15-May-2023	Created the main menu and scene flow - There were only two scenes, the game and the menu	None	None	Done
23-May-2023	During a regular playtest through my game, I realised that there was a bug with tower placement where you would be able to place the tower inside the tower buttons UI, however I knew how to fix this easily	None	None	Done
28-May-2023	Balanced certain aspects of the game more such as cost of towers, value of enemies, damage, fire rate of towers and value of towers.	None	None	Done
29-May-2023	Created an audiosource in the game scene and in the menu scene for sound effects and music, and implemented persistent data to both volumes which could be adjusted in the options menu.	None	None	Done
5-Jun-2023	Looked for	None	None	Done

	sounds to used and made a couple such as the laser gun sound and the winning music.			
11-Jun-2023	Started the final part which was the actual levels. I needed a script that had a timer that could delay between enemies sent, but I really did not have an idea on where to start on this.	Had not looked into cooldowns and other similar features of unity that implement cooldowns often so I did not have much experience with this	Find or think of a way to implement a cooldown on an enemy spawning	Done
15-Jun-2023	Found a method to implement this cooldown using a "coroutine" which would run and then the module would wait for the coroutine to finish running before continuing, which was perfect for me. I now had a template script for a level	None	None	Done
20-Jun-2023	Worked on and finalised all levels, and playtested the game various times, finding minor bugs and fixing them	None	None	Done

Bibliography

Svail (2022) *Unity Procedural Map Generation with tile maps*, *YouTube*. Available at: https://www.youtube.com/watch?v=SzkRdmB0mEQ (Accessed: 15 February 2023).

Unity Technologies. (no date) *Welcome to the Unity scripting reference!*, *Unity*. Available at: https://docs.unity3d.com/ScriptReference/ (Accessed: several times throughout the project).

Brummelen, J. van and Chen, B. (no date) *Procedural generation*. Available at: https://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html (Accessed: 13 March 2023).

Multiple authors & editors (no date) *Procedural generation* (2023) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Procedural_generation (Accessed: 16 March 2023).

Procedural generation (2023) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Procedural_generation (Accessed: 16 March 2023).

Unity Technologies (no date) *Coroutines*, *Unity - Manual: Coroutines*. Available at: https://docs.unity3d.com/Manual/Coroutines.html (Accessed: 05 May 2023).

Unity Technologies (no date) *Coroutines*, *Unity - Manual: Execution Order*. Available at: https://docs.unity3d.com/Manual/ExecutionOrder.html (Accessed: 20 May 2023).

Graphing calculator (no date) *Desmos*. Available at: https://www.desmos.com/calculator (Accessed: several times throughout the project).

inScope Studios (2016) *1.0 Unity Tower Defense Tutorial - placing tiles*, *YouTube*. Available at: https://www.youtube.com/watch?v=wS2LBCcnSQs (Accessed: 25 May 2023).

Swift, N. (2017) *Easy A* (star) pathfinding*, *Medium*. Available at: https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2 (Accessed: 01 June 2023).

GeeksforGeeks (2023) *C# program for Dijkstra's shortest path algorithm: Greedy Algo-7*, *GeeksforGeeks*. Edited by shubham_singh and _ani_s37. Available at: https://www.geeksforgeeks.org/csharp-program-for-dijkstras-shortest-path-algorithm-greedy-algo-7/ (Accessed: 07 June 2023).

AlexMorOR (2022) *Unity: What is a Coroutine and why is there an ienumerator*, *Habr*. Available at: https://habr.com/en/articles/684938/ (Accessed: 13 June 2023).

Zgeb, B. (2021) *Persistent data: How to save your game states and settings*, *Unity Blog*. Available at: https://blog.unity.com/games/persistent-data-how-to-save-your-game-states-and-settings (Accessed: 17 June 2023).