

Computer Games Development SE607

Technical Design Document

Year IV

Danial Nor Azman

C00253517

[26/04/2023]

**DECLARATION**

****

Muhammad Danial Hakim Nor Azman

26/4/2023

Danial Nor Azman

C00253517

Contents

Contents

[2 Accessibility in game for disabled people 3](#_Toc133245022)

[3 Summary of the game 3](#_Toc133245023)

[4 Bullet Elements 3](#_Toc133245024)

[5 Gun Types 4](#_Toc133245025)

[6 Enemy 5](#_Toc133245026)

[6.1 States 5](#_Toc133245027)

[6.2 Decision Making - Fuzzy Logic 7](#_Toc133245028)

[6.3 Adaptive system 9](#_Toc133245029)

[6.4 Vision Cone 10](#_Toc133245030)

[6.5 Pathfinding 10](#_Toc133245031)

[7 Procedural level generation 11](#_Toc133245032)

[8 Accessibility settings 11](#_Toc133245033)

[8.1 Magnifier 13](#_Toc133245034)

[8.2 Rebind Controls 14](#_Toc133245035)

[8.3 Visual cues as alternate to Audio cue 15](#_Toc133245036)

# Accessibility in game for disabled people

The purpose for this Technical Design Document is to outline the design and development for a game called LabEscape that focus on testing the accessibility features for people with disabilities.

# Summary of the game

LabEscape is first person rogue like shooter game in which the level layout and obstacle are procedurally generated at run time so that player can have a different experience every time. The in-game objective is cleared out a number of rooms depending on the size of the lab room generated. Once the number of required rooms cleared has been reached, a door will spawn in the starting area where player must make their way back without dying as there is no healing option. Once player pass through the door. A message will appear declaring they have won. They can use different types of weapons and ammo type against enemy that have adaptive system to the guns the player is using.

# Bullet Elements

Logo

Description automatically generated with medium confidenceA picture containing bubble chart

Description automatically generatedA picture containing icon

Description automatically generatedA picture containing bubble chart

Description automatically generated

There are 4 different types of elements and different effects.

* Ice – Freeze the enemy movement for 2 seconds.
* Fire – Set the enemy on, deal damage over time for 3 seconds.
* Electric – Deal area of effect damage.
* Water – Slow the enemy movement for 4 seconds.

Player can change their bullet types by interacting with these objects in game. The game used Enum class called elements and when player interact one of these, the bullet will update its element. To set the effect of the bullet, I use 2 dictionary. The first dictionary set the element of the bullet while the other set the effect of the bullet.

Codes and detailed explanation

Text

Description automatically generated

When player interact with one of item as shown above, it will update the element based on the colour of the item. The bullet class then updates its element by looking through the dictionary to check if the element for that colour exist. For example, if the item player interact with is red, it will be fire element.

Text

Description automatically generatedWhen the bullet collides with the enemy, it will set one of these effects based on the element the bullet is.

# Gun Types

A picture containing text

Description automatically generatedA picture containing text, sign

Description automatically generatedA picture containing text, sign

Description automatically generated

There are 3 different types of guns in this game.

Assault – Allow player to hold the fire button and keep firing until it run out of ammo or reload.

Burst – Fire 3 bullets with faster rate of fire with 1 click but required the fire button to be clicked again.

Shotgun – Fire 5 bullets at shorter range which random spread

This is achieved due to the custom gun script which control how the gun works. Some of settings are listed below:

* Time between shots – The duration for the next bullet to fires after the previous bullet.
* Bullets per tap – Bullets fires per clicked
* Spread – The higher this value, the less likely the weapon to hit the centre
* Magazine size – How much ammo the gun have before reload is required
* Allow button hold – Whether the weapon will continue to fire if player hold the fire button
* Shoot Force – How fast the bullet will go
* Time Between Shooting – Reload time

Changing the value for some of these settings would allow the gun to behave differently.

Text

Description automatically generated

Example:

Assault Gun

Table

Description automatically generated with medium confidence

Burst Gun

A picture containing graphical user interface

Description automatically generated

In both examples, upward force was going to be used as recoil which I end up did not use in the final version of the game.

The burst gun has higher shoot force, which lets the bullet to go faster compared to the assault gun. It also releases 3 bullets at once with shorter duration between shots.

# Enemy

## States

I make a BaseState class that the other state will inherit from so that I can continue to add more state in the future if I want to.

The enemy uses fuzzy logic (which will be explain in the next section) to make a decision and also how fast the enemy should be moving. The enemy have 3 main states which is listed below.

Patrolling: Enemy will patrol around the room, avoiding the obstacle for both player and enemy. There are multiple points in the room where the enemy must go to, the points are randomly chosen once the enemy has reach one of the points. This will go in a loop so enemy will always patrol unless an event occurs.

Chasing / Attacking: Enemy will chase the player if player is found. Player is considered found if enemy can see the player with their vision cone. Player will be attacking once player is in attacking range of the enemy.

Retreating: Enemy will heal up if the enemy health is below a certain threshold unless an event occurs.

Codes for BaseState

Text

Description automatically generated

Example RetreatState

Text

Description automatically generated

When the enemy enter a new state, they will change colour based on which state they go into as this serve as an indicator to the player what state the enemy is in. Each state has their own function which is called in the Perform function. In this case, the enemy is looking for a healing zone. The speed will be given by the fuzzy logic unless the enemy have a status ailment on them like being frozen or slowed.

### StateMachine

StateMachine script are in charge of keeping track which state the enemy is in and will tell the enemy if they need to change their state based on the situation.Text

Description automatically generated

This is part of the code in StateMachine, it first exits whatever state the enemy is in to in the next state. It also gets the enemy component every time it changes state, this is so that the state will always have the latest version of the enemy.

How it looks:

Timeline

Description automatically generated

## Decision Making - Fuzzy Logic

The enemy use fuzzy logic in their decision making.

Decision for the AI is decided like the table below where low, mid and high represent the threat level.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Player Dying | Player Hurt | Player Healthy |
| Enemy Dying | Mid | High | High |
| Enemy Hurt | Low | Mid | High |
| Enemy Healthy | Low | Low | Mid |

Each threat level also calculates the speed the enemy will be moving. If enemy health is 30 which is considered low and player health is 80 which is considered high, they will move at the speed of 4. If the enemy health is at 20 and player health is at 95, the enemy will move at speed of 7 to heal up which is almost double than their previous speed.

If the enemy health is low where it should be healing, but the player health is lower, the enemy will prioritize attacking the player. The threat level basically makes the enemy either more passive or aggressive depending on the situation.

In my game, I used the Trapezoid shape to represent the values for both the player and enemy health.

Trapezoid shape is shown below. This illustration is not mine and was taken from Oisin Cawley’s slides in his Fuzzy Logic explanation.

A picture containing text, scale, device

Description automatically generated

I make a function that would take in 5 value. The first value would be the value I would want to know where it belongs in for this group. And the remaining 4 values are the x0, x1, x2 and x3.

Text

Description automatically generated

For example, if player health is 100. The values for each variable are as such:

pDying = 0, pHurt = 0, pHealthy = 1.

This is because 100 is not include in both pDying and pHurt so the function will return 0 and pHealthy will return 1 as it belongs to x3.

If player health is a value of 65. The values will change to:

pDying = 0, pHurt = 0.75, pHealthy = 0.25.

This is because 65 exist in both group for the player being hurt and healthy but is skew more towards hurt.

The enemy will then calculate the threat level using the table above.

Text

Description automatically generated

The lower the value, the faster the enemy will move to retreat, while the higher the value, the more aggressive the enemy will be in chasing and attacking the player. This speed value which is then passed to the perform function in the state machine above. As agent in Unity does not take in negative value, the value is multiplied by -1 if its less than 0.

Timeline

Description automatically generated

Combination of both the state machine and fuzzy logic allow the enemy to make more “human like” response. In the event that the enemy health is low but the player health is lower, the enemy will attack the player instead of going to heal. The minimum value for the decision is -10 and the maximum value is 10. From -0.1 to -10, it shows that the enemy should retreat will the value also served as the speed. If the decision return a value of -5 , enemy will retreat with the speed of 5 while the decision with a value of -10 means that the enemy will retreat with the speed of 10. The same applies to attacking , making the enemy chasing speed faster if the player health is lower.

## Adaptive system

As there are multiple type of gun and bullet elements, the enemy will adapt to it with their stats. The game keeps tracks on how the player kill their enemy. If player keep on using the fire bullet, enemy will take reduce over time damage and at one point will be immune to it. If player keep on using ice bullet where it froze the enemy for 2 seconds initially, the frozen duration will go down to 0 if player keep on using it, making it just like a normal bullet.

This also mean the longer the player plays the game, the harder it is to kill the enemy as they will get immune to the element’s perks.

I achieved this by using dictionary to keep track of the values for each elements.

I first create 2 dictionary. 1 is to know which element the enemy dies to. The other one is to tell the enemy what the value should be modified to.

A screenshot of a computer

Description automatically generated with medium confidence

Every time an enemy dies, a function updates the other enemy’s resistance towards that element to be increased while the resistance towards the other elements to be decrease which reward player for changing their bullet element types regularly.

## Vision Cone

The enemy have a vision cone which act as the enemy eyes. The vision will be displayed on screen so player can see to avoid them. The vision cone cannot go through walls and obstacle so player can choose to take cover so the enemy will stop chasing them.

For the vision cone, I simply used the ray cast in unity to check if its colliding with the walls to stop it from going through. I then just set the layering to every walls and obstacle, so it obstructs the vision cone for the enemy.

In Unity, you can set the range on how far the ray cast will be. I used this to check if the player is in range and is colliding with the vision cone, if they are, simply set the bool for player found to true and change the behaviour to chase player.

## Pathfinding

As the game is generated at run time, I use NavMesh which is a component from Unity which allow the enemy to automatically walk on the terrain based on the settings that I set. Unity NavMesh allow AI to transverse the world area automatically based on the settings that you set. Default version, however, only lets you set it up before run time which causes issue for me as I have a procedural level generation.

To overcome this problem, I download an extra component called NavMesh component and build the component after the layout and obstacles has been spawned. The code for this is shown in the level generation section.

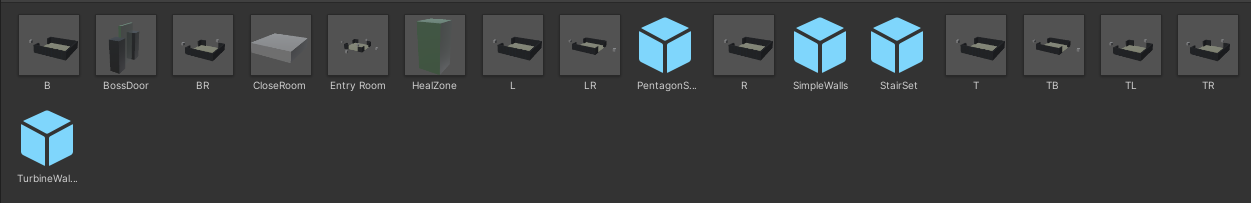
# Procedural level generation

A picture containing diagram

Description automatically generated

The game level generation is by using templates of room that were created before run time. The room, however, is flat surface with different entry and exit points. The room spawn by checking if there’s a room at the exit point of the current room. If there’s no room, it will spawn a room corresponding to that room.

For example, if a room with an opening at the bottom of it, a room with an opening at the top will be spawned in. If 2 rooms clashed while it was spawning, it will spawn a closed room which acts as a wall, this is to prevent the room from spawning on top of each other as room that handle the spawning can spawn different type of rooms.



A screenshot of a computer

Description automatically generated with medium confidence

The naming presents the opening the room have. BR means there are 2 opening to the room which is bottom and right side of the room. B would mean bottom.

Once the layout of the level has been spawn, the game then randomly picks obstacles to spawn in the room so even if the room is of the same layout, the obstacles will make it different.

Top-down view of some of the obstacles.

Chart

Description automatically generated A picture containing graphical user interface

Description automatically generated

I was planning to set random rotation to the obstacles to add a little bit more randomness but the current rotation works better than I thought so I left it as it is.

The level also spawn in the gun modification items and bullet elements item in each room at their own corner.

After all the static items like the layout , obstacles , walls , gun modification and bullet elements items has been spawned , the level then spawn enemy in every room before spawning the player in the starting room.

For refrences , it looks like the code below.

Text

Description automatically generated

Level generation progress

A picture containing text, clipart

Description automatically generated

The game spawn out the layout of the level as shown above without anything on it. These are all empties room.

A picture containing diagram

Description automatically generated

It then spawn in the obstacles form the obstacles prefabs.After it has build the layout and placed the obstacles , it will then build the NavMesh for the enemy to use to move around.

Graphical user interface

Description automatically generated with medium confidence

The last step is adding in the gun mods, bullet elements and the player. The player always spawns in the starting room. The starting room is placed before the gun runs , it then build the level around it.

# Accessibility settings

## Magnifier

Chart

Description automatically generated with medium confidence

Magnifier can be accessed by pressing P on the keyboard. It will enlarge item or objects that are further away from player, so it is easier to be seen. This is done by having another camera in the scene which will handle the zooming in. Put the camera view onto a texture and put it on a game object to create zoom in like effect. Some maths calculation was required to get the effect I need.

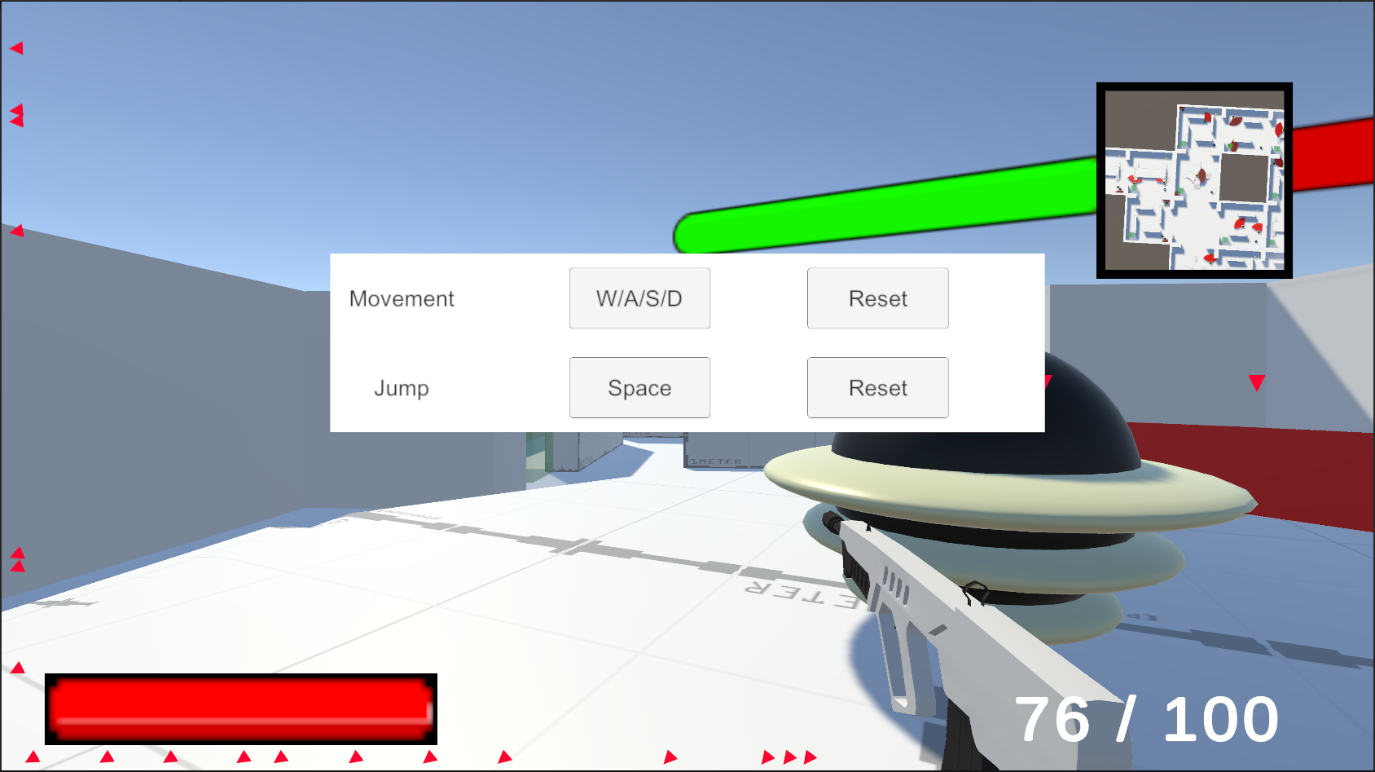
If an eye tracker is attached, player move the magnifier around the screen. The position is default to the middle of the screen. Player can also zoom in and zoom out if they close one of their eyes to better fit their needs.

Limitation: only works on PC as you would need to press the P button on the keyboard.

The steps I took to create the magnifier:

* Create a new camera attached to the player
* Position it slightly in front of the player and lower the field of view to half of the player main camera.
* I create a texture and assign the camera to the albedo of the texture.
* Assign the texture to a material
* Assign the new material to a game object. The material will update every time the scene the camera is looking at is change, giving a magnifier that updates regularly.

## Rebind Controls



Player can rebind controls to suit their needs. This is done in script where it checks how many buttons is responsible for that particular action such as movement and jumping. If it is a composite action like the movement, it will ask for 4 inputs. The script then updates the bindings and store it in a Json file, so player does not need to rebind it every time they play the game.

Player can also click on the reset button to reset to the default settings. These settings are not limited to keyboard so player can connect their controller to rebind the actions.

Limitations: interact button and the sensitivity of the camera movement cannot be rebind / change to fit the player preference.

How it works:

When player click on one of the button to rebind, it will call the do rebind function which will takes in the input name, for example above, it can be movement or jump. Based on these name the function will check if the action is a composite action or not. Movement would be considered as composite action as it has multiple button attached to it. In the script, it checks if player is pressing on a valid button when they are rebinding. Only invalid button in my case would be the mouse click buttons. If the action is composite, the function will keep asking for new button and once every action button has been assigned, it will store the new bindings, removing the old one, not the default one from the Json file.

When player load the game the next time, it will then read the Json file to check for the bindings. If player click reset, it simply set the bindings into the default values.

## Visual cues as alternate to Audio cue



The red arrows are visual indicators that help players with impaired hearing locate enemies. The arrows can go through walls and show where enemies are if they are in front of the player, and they adjust based on where the player is looking. The arrows also change opacity to show how far away enemies are, with faded arrows indicating greater distance.

There are 3 scripts to achieved this, the first script is on the canvas to calculate the position and where the arrow should be pointing. The second script is attached to the enemy as it needs to be updated all the based on the enemy position. The last script is attached to the player, as the calculation is also based on the player position and the target position.

I implemented the visual indicator based on a tutorial from YouTube called [Unity tutorial – Off screen Target Indicators](https://www.youtube.com/watch?v=Ffi8kz6AheA). You can click the link to find out more.

Summary of how it works.

Step 1: WorldToScreenPoint for In-Front Enemies

* Use the Unity Camera.WorldToScreenPoint function to get the screen position of the enemy, which takes in the enemy's position as a Vector3.
* Set the position of the enemy indicator to the screen position of the enemy.
* Add a y-offset to the indicator's position to make it appear on top of the enemy.

Step 2: Intersection Calculation for Out-of-View Enemies

* If the enemy is outside of the camera view, calculate the intersection of the enemy's position with the screen borders (x and y) of the camera.
* If the enemy's x position intersects with the camera's x border, set the x position of the indicator to the x border value of the camera.
* Get the signed angle between the enemy's position and the camera's position to determine whether the indicator should move up or down.
* Use the tangent function to convert the angle from degrees to radians (Radians = degree \* (3.14 / 180)).
* Multiply the indicator's position with the radian value to determine the final position of the indicator.

This Illustration does not belong to me, and the credit goes to the owner which is youtuber who upload the tutorial. I have linked the video up above.

Step 1 and 2 visualization

Diagram

Description automatically generated

Step 3 and 4 visualization

Diagram

Description automatically generated with low confidence

The opacity of the indicator is calculated by using Vector3.Distance and putting in the enemy position and the camera position (which is attached to the player) and multiply the value by 0.2 until it reaches the opacity of 1 or 0 which is the limit.