

Introduction

My project was heavily inspired by other popular survival games such as Project Zomboid, 60 Seconds!, and Valheim. I wanted to create a game that not only would I enjoy working on but also one that I would enjoy playing. This thought process led me to creating a survival game due to it being one of my favorite video game genres.

Perimeter is meant to create a challenging but engaging experience where players must manage their limited resources while dealing with the dangers of being in a zombie infested city. The player has to consider their actions and priorities, weighing the risk and reward of any given situation or decision in order to survive as long as possible.

Methodology

1. What to Develop

When first deciding what game I should make I thought about what game I would like to play as well as what mechanics I thought I could code. I took inspiration from games I have played in the past as well as popular games that I have seen.

2. How to Develop

Once I decided on the type of game I wanted to create I had to decide how to develop my game. To make my game I decided to use the Roblox Engine. Although I had a small amount of experience using the Roblox Engine my biggest obstacle was learning and trying to piece together how to correctly create and input the mechanics into the game as many were more complex then I had originally thought.

3. Development

When I first began development of my game, I quickly realized how many basic mechanics could be extremely difficult. In order to break the development process into more manageable pieces I organized the components into 3 categories which I worked on dynamically to avoid getting stuck or burnt out.

I. User Interface and Map Design

The User Interface(UI) consists of showing the players Health, Hunger, and Thirst meters. The UI is there to help the player visualize how healthy, hungry or thirsty they are. The map design consists of the ground and mountains present in the playable area

II. Modelling/Animation

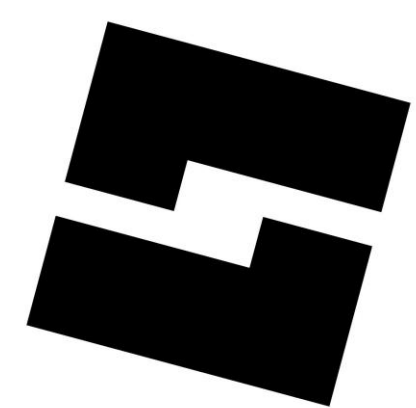
Modelling refers to all the items and building that are visible in the game. All the buildings and weapons were custom made by myself along with the animations for the combat system.

III. Coding

The code allows the player to interact with items, consume items, dictates the amount of damage the player receives and deals, and how much and how often hunger and thirst decreases. The code also dictates how much healing, hunger, or thirst is regained upon consuming items such as bandages, food, and water.

Software

To create my game I used Roblox Studio which is a free game engine and development environment used to create “experiences” for the Roblox platform. Roblox Studio runs the Roblox Engine which uses C++ for features such as physics simulations, rendering, networking, memory management, audio systems, and core engine loops. Roblox Engine also uses Lua for game scripting which is Roblox’s optimized version of Lua, this handles game logic, UI updates, player interactions, and events(Touched, Clicked, RemoteEvents).



Lua



Code

Hunger/Thirst Script

Lines 1-5: Control how frequently and how much players hunger and thirst decrease over time

Lines 7-21: Establishes how much Hunger and Thirst the player has

Lines 26-28: Kills player if either value hits 0

Lines 30-35: Decreases the player Hunger and Thirst if the values are larger than 0

Lines 40-45: Restores Hunger and Thirst if the player eats or drinks

```
1 local HUNGER_DECREASE_FREQUENCY = 1
2 local HUNGER_DECREASE_AMOUNT = 1
3 local THIRST_DECREASE_FREQUENCY = 1
4 local THIRST_DECREASE_AMOUNT = 1
5 local INCREASE_AMOUNT = 15
6
7 game.Players.PlayerAdded:Connect(function(Player)
8     local Hunger = Instance.new("NumberValue", Player)
9     Hunger.Name = "Hunger"
10    Hunger.Value = 100
11
12    -- Thirst
13    local Thirst = Instance.new("NumberValue", Player)
14    Thirst.Name = "Thirst"
15    Thirst.Value = 100
16
17    Player.CharacterAdded:Connect(function()
18        Hunger.Value = 100
19        Thirst.Value = 100
20    end)
21
22    while true do
23        wait(HUNGER_DECREASE_FREQUENCY)
24        if Hunger.Value <= 0 or Thirst.Value <= 0 then
25            Player.Character:BreakJoints()
26            break
27        else
28            if Hunger.Value > 0 then
29                Hunger.Value -= HUNGER_DECREASE_AMOUNT
30            end
31            if Thirst.Value > 0 then
32                Thirst.Value -= THIRST_DECREASE_AMOUNT
33            end
34        end
35    end
36
37 end)
38
39 game.ReplicatedStorage.Got_OnServerEvent:Connect(function(Player)
40     Player.Hunger.Value += INCREASE_AMOUNT
41 end)
42
43 game.ReplicatedStorage.Drink_OnServerEvent:Connect(function(Player)
44     Player.Thirst.Value += INCREASE_AMOUNT
45 end)
```

```
1 local NPC = script.Parent
2 local HumanoidRootPart = NPC.HumanoidRootPart
3 local MaxDistance = math.round(25)
4 local Debounce = false
5
6 NPC.Humanoid.WalkSpeed = 8
7 NPC.Humanoid:SetStateEnabled(Enum.HumanoidStateType.Climbing, false)
8
9
10 NPC.Humanoid.Touched:Connect(function(hit)
11     local character = hit.Parent
12     local humanoid = character:FindFirstChild("Humanoid")
13
14     if humanoid and not Debounce then
15         local player = game.Players:GetPlayerFromCharacter(character)
16
17         if player then
18             Debounce = true
19             humanoid.Health -= 10
20             wait(5)
21             Debounce = false
22         end
23     end
24 end)
25
26 while wait() do
27     local Players = game.Players:GetPlayers()
28     local Closest
29
30     for i,plr in pairs(Players) do
31         if plr.Character and plr.Character:FindFirstChild("Humanoid") and plr.Character.Humanoid.Health > 0 then
32             local playerHumanoidRootPart = plr.Character.HumanoidRootPart
33             local Distance = (humanoidRootPart.Position-playerHumanoidRootPart.Position).Magnitude
34
35             if not Closest then
36                 Closest = playerHumanoidRootPart
37             end
38
39             if (humanoidRootPart.Position - Closest.Position).Magnitude > Distance then
40                 Closest = playerHumanoidRootPart
41             end
42         end
43     end
44
45     if Closest and (humanoidRootPart.Position - Closest.Position).Magnitude <= MaxDistance then
46         NPC.Humanoid:MoveTo(Closest.Position)
47     end
48 end
```

Hunger UI Script

Lines 11-12: Retrieves the current value stored in Hunger.Value from the Hunger/Thirst Script then converts it into a Percentage

Line 14: Tells the animation of the UI how to behave: the animation starts quickly and slows down near the end, the animation move at a constant speed, animation duration 0.15 seconds

```
1 local Player = game.Players:LocalPlayer
2 local Hunger = Player:WaitForChild("Hunger")
3
4
5 local GUI = script.Parent
6 local BarInterior = GUI.BarInterior
7 local BarInterior = BarInterior.Bar
8
9 local function UpdateHunger()
10    local CurrentValue = Hunger.Value
11    local Formula = math.Clamp(CurrentValue/100, 0, 1)
12    BarInterior.TweenSize(UDim2.new(Formula, 0, 1, 0), Enum.EasingDirection.Out, Enum.EasingStyle.Linear, 0.15, true)
13 end
14
15 UpdateHunger()
16 Hunger.Changed:Connect(function()
17     UpdateHunger()
18 end)
```

Health UI Script

Line 9: Changes the text in the UI to the players current health to the nearest whole number

Lines 14-15: Watches the Health property of the player and creates an event that activates when the health of the player changes. It then tells the server to run the functions which updates the players UI to visually express the players current health

```
1 local Player = game:GetService("Players"):LocalPlayer
2 local Character = Player.Character or Player.CharacterAdded:Wait()
3 local Humanoid = Character:WaitForChild("Humanoid")
4
5 localTextLabel = script.Parent
6
7 local function UpdateHealthBar()
8    TextLabel.Text = "Health: " .. math.round(Humanoid.Health)
9 end
10
11 UpdateHealthBar()
12
13 Humanoid:GetPropertyChangedSignal("Health"):Connect(UpdateHealthBar)
14 Humanoid:GetPropertyChangedSignal("MaxHealth"):Connect(UpdateHealthBar)
```

Swing Animation Script

Lines 11-12: Checks if cooldown is active, if not proceeds while setting cooldown to true to prevent continuous activations

Lines 13-14: Loads animations into the player then plays the swing animation

Line 15: Fires the RemoteEvent named “SwingEvent” which triggers damage/hit detection

```
1 local player = game.Players:LocalPlayer
2 repeat wait() until player.Character
3 local Character = player.Character
4 local Humanoid = Character.Humanoid
5
6 local Cooldown = false
7
8 local Animation = script.Parent.Swing
9
10 script.Parent.Activated:Connect(function()
11     if Cooldown == false then
12         Cooldown = true
13         local LoadedAnimation = Humanoid:LoadAnimation(Animation)
14         LoadedAnimation:Play()
15         script.Parent.SwingEvent:FireServer()
16         wait(1)
17         Cooldown = false
18     end
19 end)
```

Gameplay

Perimeter is a zombie survival game where the main objective is to simply survive as long as you can. The core mechanics consist of managing your resources to ensure you do not die of hunger or thirst. The limited resources you can collect at the starting area force the player to explore and navigate their way through the zombie infested city. This adds to the challenge of not only surviving by ensuring your character has enough food and water but also surviving dangerous encounters with the undead.



Future Improvements

Although I got all the core mechanics that I wanted for my game there are still many aspects and improvements that I would like to add.

➤ Larger Variety of Enemies/NPCs

I would like to add more enemy types with different classes or even factions. For the Zombies I would add a tank class as well as a ranged class to add more variety to the base enemies you encounter. I would also like to add more factions that the player could interact with such as Raiders who are aggressive towards the player and Survivors who the player can help and/or add to their team.

➤ More Weapons/Types

I would like to add more weapons as well as weapon types. More weapons would include knives, swords, axes, etc. Weapon types would include ranged weapons such as guns, bows, or thrown weapons such as spears, or even traps that the player could use against large groups of enemies.

➤ Procedural World Generation and Random Loot Spawns

Procedural generation for the world and loot spawn would greatly increase the replayability of the game as it would allow for a completely unique experience every single time the game was played.

Lessons Learned

I learned a lot while working on this project. Some of my biggest takeaways were the technical and mechanical skills I learned as well as the expectations versus reality of game development. Through the many hours I spent working on this project I became more skilled in my technical ability to code and more adept in my mechanical skills for using hotkeys and keyboard shortcuts. I quickly learned that a lot of the ideas I had for my game were way more complex than I originally thought and I had underestimated how much time it would take me to learn how to code the features I wanted and successfully implement them into my game.

