FINAL PROJECT DESCRIPTION FOR PRU213 COURSE

SUMMER2025

BattlePow

Group name: 01

Members of Group:

Full Name	Student code
Nguyễn Ngọc Ánh	HE160896
Nguyễn Quang Huy	HE171567
Thái Duy Phong	HE180210
Nguyễn Xuân Hanh	HE180944
Trần Duy Long	HE186919

I. Introduction (Game story)

The year is 2047. The world collapsed after an event known as the Night Plague-a man-made parasitic bio-weapon that escaped a military research facility. Within 72 hours, over 85% of the global population was transformed into flesh-eating undead. Humanity lost its cities, its nations, its leaders — and its future.

Once-bright skylines are now silent ruins. Streets are painted with ash, blood, and echoes of screams. There are no governments. No rescue. No sanctuary. Only two choices remain:

Die… or fight to survive.

You are the last known survivor inside Quarantine Zone-17, the epicenter where the virus was born. The only remaining intel suggests there may still be a cure, but all research data and organic samples are buried deep within the zone — now completely overrun by evolving zombie mutations.

As night falls, the infected awaken with greater hunger. They grow stronger, faster, and more coordinated by the hour…

🧟 THE ENEMIES YOU WILL FACE

- **Basic Zombie** – mindless, aggressive, endless in number
- **Heal Zombie** – regenerates flesh and prolongs every encounter

- **Energy Zombie** – violently fast, infused with unstable bio-energy
- **Explosion Zombie** – detonates on death, turning mistakes into disasters
- **Mini Zombie** – small, relentless, swarming like insects
- **Boss Mutation** – the nightmare result of failed experiments, towering and brutal

They don't just chase — they hunt, coordinate, and evolve, making every encounter unpredictable.

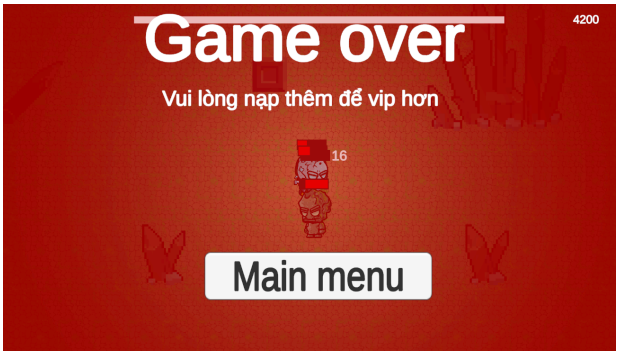
YOUR SURVIVAL MISSION

To stay alive, you must:

- Scavenge weapons, ammo, gold, and energy
- Upgrade your skills and adapt to rising difficulty
- Fight through waves of mutations
- Survive as long as possible
- And uncover the final truth behind the Night Plague — before humanity's last hope is gone

Every decision matters. Every bullet has weight. Every second alive pushes you closer to salvation… or death.

Screen Name	Screen
Main Menu	

Setting Panel	
Information Panel	
Level Select	
Game Over	

Shop	
Upgrade	

II. Plan for Project (9 weeks)

No	Task	Member	Time	Date Range
1	Create team/group	Trần Duy Long	Week 1	08/09 – 14/09/2025
2	Choose topic (final project)	Team	Week 1	08/09 – 14/09/2025
3	Choose game story	Team	Week 2	15/09 – 21/09/2025

4	Create Player	Nguyễn Xuân Hanh, Nguyễn Ngọc Ánh, Thái Duy Phong, Nguyễn Quang Huy	Week 2	15/09 – 21/09/2025
4.1	Create Enemies	Thái Duy Phong (main), Nguyễn Xuân Hanh, Nguyễn Quang Huy, Nguyễn Ngọc Ánh	Week 2	15/09 – 21/09/2025
4.2	Create Weapon system	Team	Week 2	15/09 – 21/09/2025
4.3	Create HealthBar (UI)	Nguyễn Ngọc Ánh	Week 3	22/09 – 28/09/2025
4.4	Create Projectile system	Team	Week 3	22/09 – 28/09/2025
4.5	Scene creation & management	Nguyễn Xuân Hanh, Trần Duy Long	Week 3	22/09 – 28/09/2025
4.6	Add Rigidbody & Collider to all GameObjects	Team	Week 3	22/09 – 28/09/2025
4.7	Build and run Testcases	Team	Week 3	22/09 – 28/09/2025
4.8	Collect Reference Materials	Team	Week 3	22/09 – 28/09/2025
5	Design scripts (core gameplay logic, movement)	Thái Duy Phong, Nguyễn Quang Huy, Nguyễn Ngọc Ánh	Week 4	29/09 – 05/10/2025
6	Continue scripting (Enemy AI, attacks, interactions)	Thái Duy Phong, Nguyễn Quang Huy, Nguyễn Ngọc Ánh	Week 5	06/10 – 12/10/2025
7	Design Menus (UI navigation, Start/Pause/End)	Nguyễn Ngọc Ánh	Week 5	06/10 – 12/10/2025
8	Implement Animations (Character & Enemy basic motions)	Nguyễn Nhật Minh, Nguyễn Quang Huy, Thái Duy Phong	Week 6	13/10 – 19/10/2025
9	Continue Animations (skills, transitions, combos)	Nguyễn Nhật Minh, Nguyễn Quang Huy, Thái Duy Phong	Week 7	20/10 – 26/10/2025
10	Design Maps (levels, obstacles, backgrounds)	Nguyễn Xuân Hanh, Trần Duy Long	Week 6–7	13/10 – 26/10/2025
11	Implement Scoring System (score logic & UI)	Team	Week 8	27/10 – 02/11/2025

12	Improve & balance Scoring System	Team	Week 9	03/11 – 09/11/2025
13	Write report & documentation	Team	Week 10	10/11 – 16/11/2025
14	Final review, polish & presentation	Team	Week 10	10/11 – 16/11/2025

III. Process Game Making

3.1 Understanding Problem

In designing *BattlePow*, our team identified several key challenges in creating an engaging and playable 2D action game:

1. **Gameplay Balance**
    - Designing enemies that are challenging but fair for players.
    - Balancing weapon power, ammo, and character stats.
2. **Smooth Player Control & Interaction**
    - Ensuring responsive movement, shooting, dodging, and collision detection.
    - Avoiding lag or delay between input and animation.
3. **Visual Clarity & Aesthetic Consistency**
    - Maintaining consistent art style across different maps and assets.
    - Designing clear animations and effects that help players read the action easily.
4. **Game Progression & Engagement**
    - Creating distinct levels (maps) with unique enemies, hazards, and bosses.
    - Ensuring the player feels a sense of growth and accomplishment after each map.
5. **Technical Performance**
    - Managing multiple objects, projectiles, and particle effects efficiently.
    - Keeping frame rate stable in Unity even with many enemies on screen.

3.2 Design a solution

To address these challenges, the team proposed the following design plan:

Game Architecture

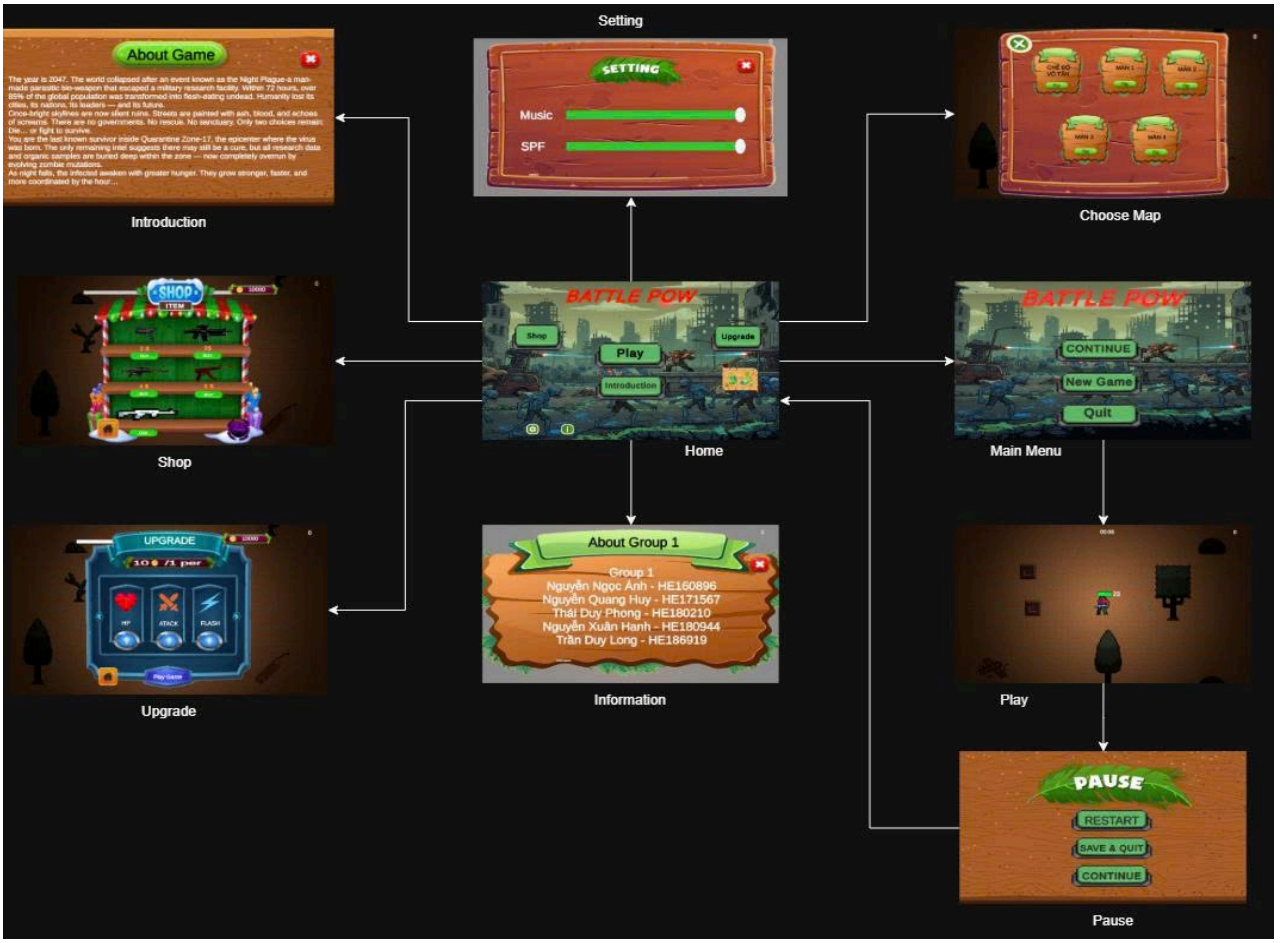
- Built using Unity 2D URP with a modular folder structure: *Scripts*, *Prefabs*, *Animations*, *UI*, and *Audio*.
- Applied component-based design, separating player movement, health, weapons, and AI logic for easy updates.
- Used Tilemap system for flexible map creation and collision control.

Gameplay Design

- Player System: Includes movement, melee attacks, health, dodge mechanics, and weapon switching.
- Enemy System: Implements AI states (Idle, Chase, Attack, Death) with unique behaviors for each monster type.
- Weapon System: Supports multiple projectile types and damage scaling.
- Scoring System: Tracks kills, combo bonuses, and survival time.
- UI System: Health bars, menu navigation, score display, and game-over panels.

Game Flow

1. Main Menu → Level Select → Game Scene → Game Over Panel



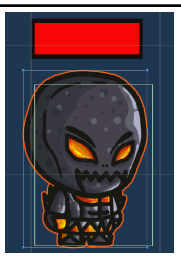
2. Players can replay completed maps to improve their score.

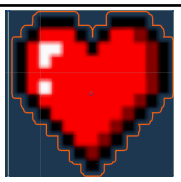
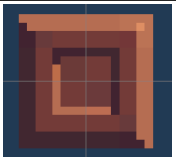
Testing & Balancing

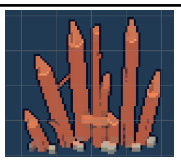
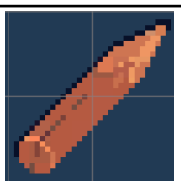
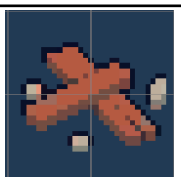
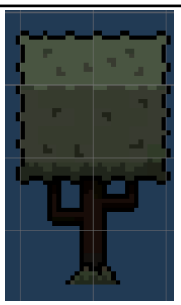
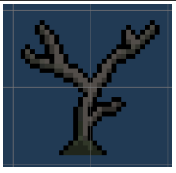
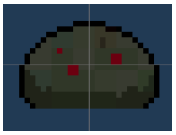
- Iterative testing each week (player control, collisions, enemy behavior).
- Gradual difficulty scaling between maps.
- Peer review of animations and scripts to maintain performance.

3.2.1 Characters and Sprites

No	Name	Attributes	Sprite	Game object name	Note

1	Basic enemy	Health, Movement Speed, Attack Damage,		BasicEnemy	Auto movement
2	Boss enemy	Health, Movement Speed, Melee Attack, Bullet, Bullet speed,		BossEnemy	Auto movement
3	Energy enemy	Health, Movement Speed, Attack Damage,		EnergyEnemy	Auto movement
4	Explosion enemy	Health, Movement Speed,Attack Damage		ExplosionEnemy	Auto movement
5	Heal enemy	Health, Movement Speed,Attack Damage		HealEnemy	Auto movement
6	Mini enemy	Health, Movement Speed,Attack Damage		MiniEnemy	Auto movement

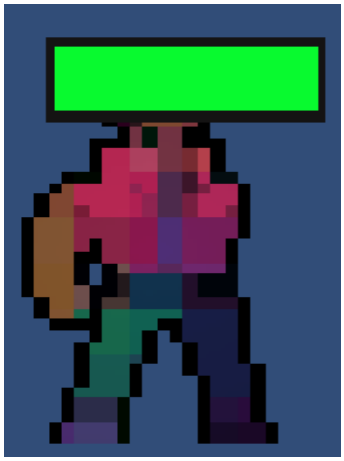
7	Player	Health, Movement Speed		Player	Main controllable character
8	Normal Cursor	Size, Format		NormalCursor	In-game Cursor
9	Reload Cursor	Size, Format		ReloadCursor	In-game Cursor
10	Shoot Cursor	Size, Format		ShootCursor	In-game Cursor
11	Box			Box	In-game item
12	Coin			Coin	In-game item
13	HealHp			HealHp	In-game item
14	Texture				In-game texture

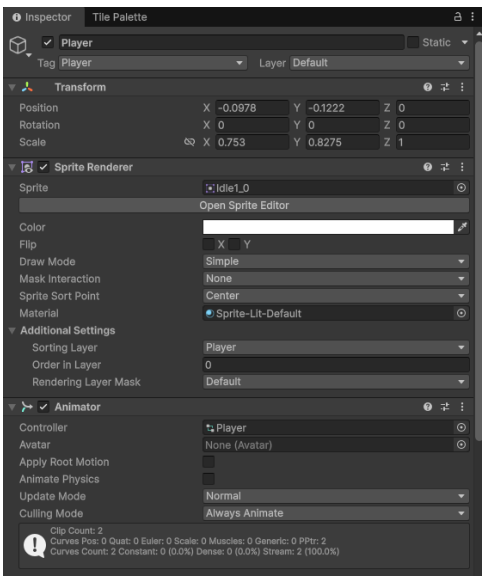
15	Tileset				In-game texture
16	Tileset				In-game texture
17	Tileset				In-game texture
18	Tree				In-game texture
19	Tree				In-game texture
20	Tree				In-game texture
21	Tree				In-game texture
22	Tree				In-game texture

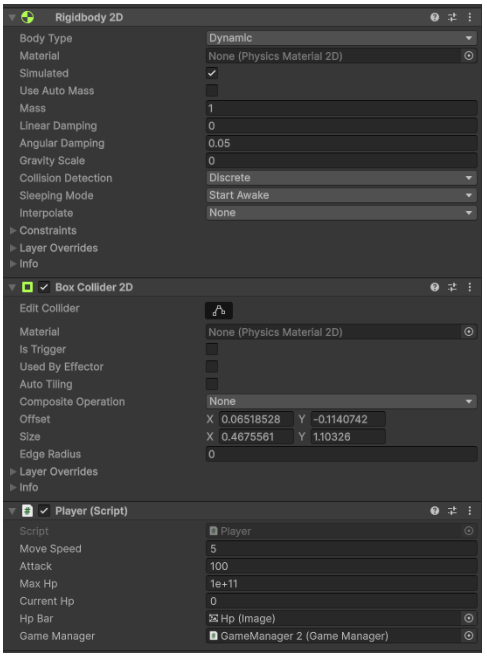
23	Gun1				In-game item
24	Gun2				In-game item
25	Gun3				In-game item
26	Gun4				In-game item
27	Gun5				In-game item
28	USB				In-game item

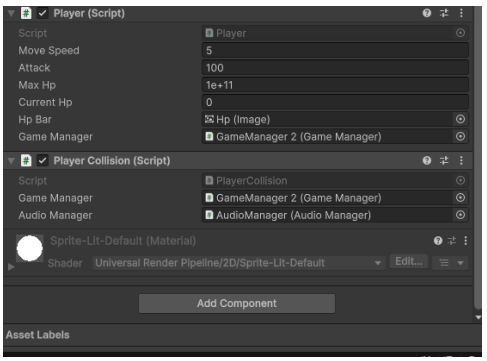
3.2.2 Design game objects

a) Player:

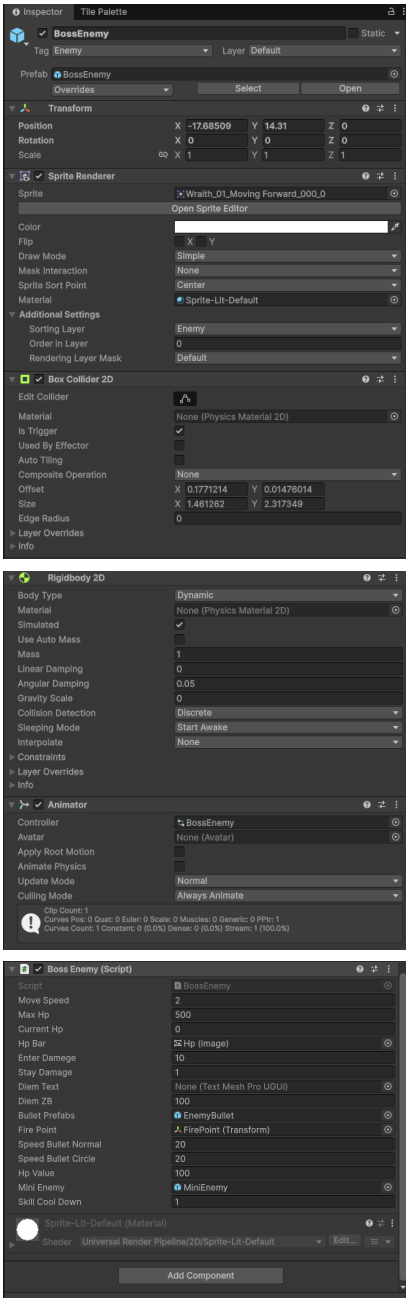


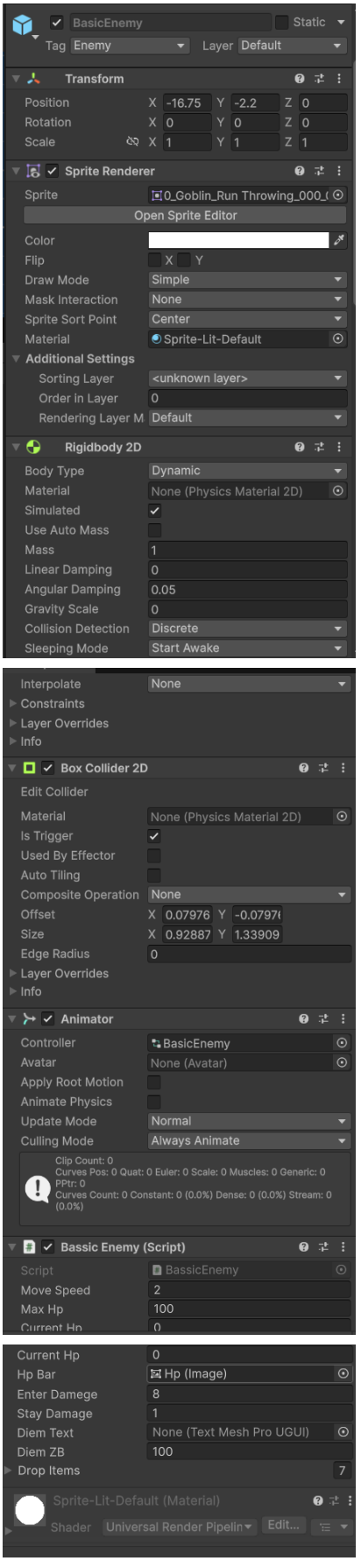


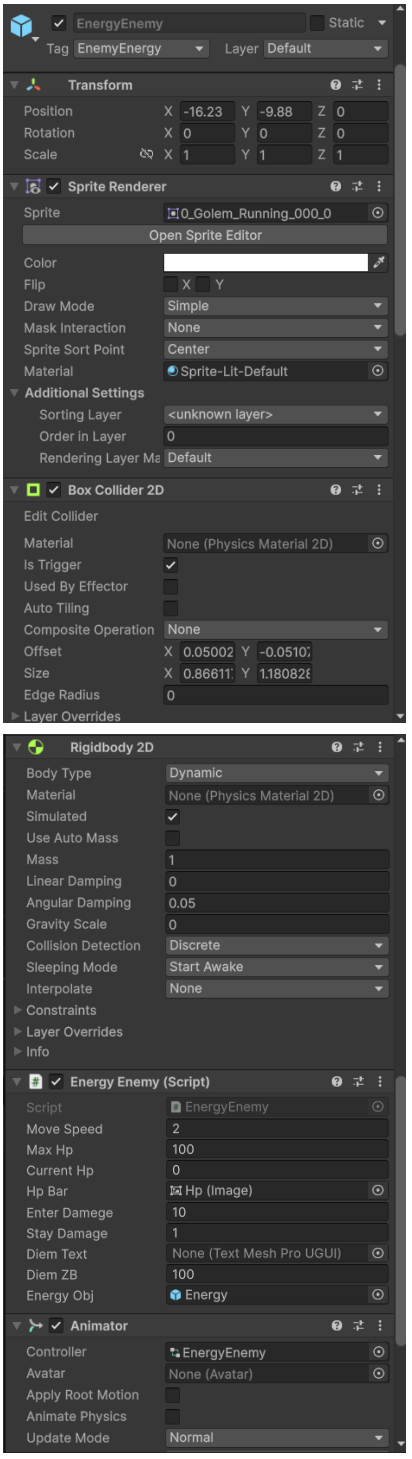

Rigidbody 2D
Body Type
Dynamic
Material
None (Physics Material 2D)
Simulated
Use Auto Mass
Mass
1
Linear Damping
0
Angular Damping
0.05
Gravity Scale
0
Collision Detection
Discrete
Sleeping Mode
Start Awake
Interpolate
None
Constraints
Layer Overrides
Info
Box Collider 2D
Edit Collider
Material
None (Physics Material 2D)
Is Trigger
Used By Effector
Auto Tiling
Composite Operation
None
Offset
X 0.06518528
Y -0.1140742
Size
X 0.4675561
Y 1.10326
Edge Radius
0
Layer Overrides
Info
Player (Script)
Script
Player
Move Speed
5
Attack
100
Max Hp
1e+11
Current Hp
0
Hp Bar
Hp (Image)
Game Manager
GameManager 2 (Game Manager)

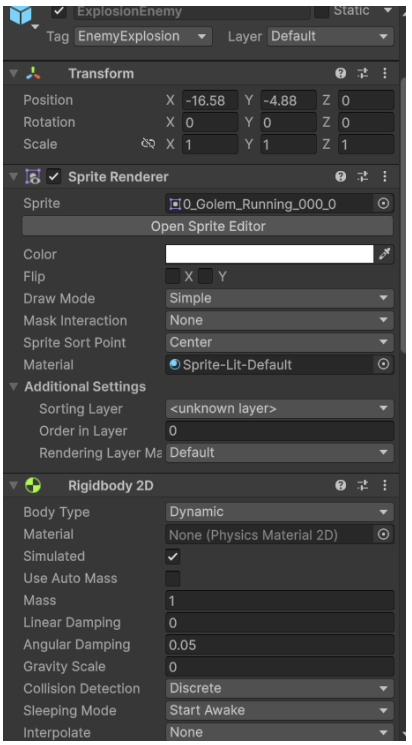

Player (Script)
Script
Player
Move Speed
5
Attack
100
Max Hp
1e+11
Current Hp
0
Hp Bar
Hp (Image)
Game Manager
GameManager 2 (Game Manager)
Player Collision (Script)
Script
PlayerCollision
Game Manager
GameManager 2 (Game Manager)
Audio Manager
AudioManager (Audio Manager)
Sprite-Lit-Default (Material)
Shader
Universal Render Pipeline/2D/Sprite-Lit-Default
Edit...
Add Component
Asset Labels

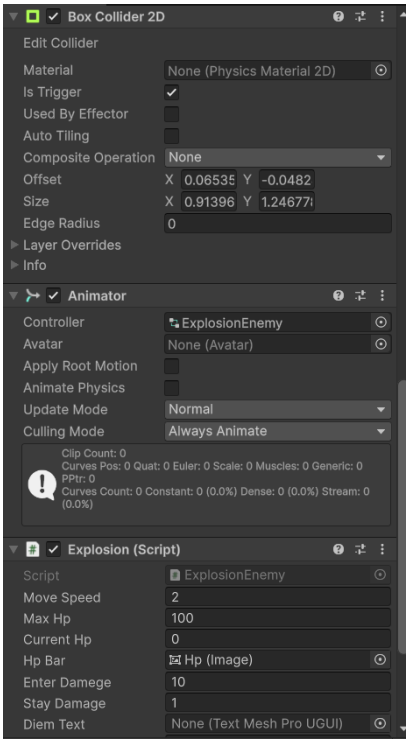
b) Enemy:


Inspector
Tile Palette
BossEnemy
Static
Tag Enemy
Layer Default
Prefab BossEnemy
Overrides
Select
Open
Transform
Position X -17.68509 Y 14.31 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sprite Renderer
Sprite Wraith_01_Moving Forward_000_0
Open Sprite Editor
Color
Flip X Y
Draw Mode Simple
Mask Interaction None
Sprite Sort Point Center
Material Sprite-Lit-Default
Additional Settings
Sorting Layer Enemy
Order in Layer 0
Rendering Layer Mask Default
Box Collider 2D
Edit Collider
Material None (Physics Material 2D)
Is Trigger
Used By Effector
Auto Tiling
Composite Operation None
Offset X 0.1771214 Y 0.01476014
Size X 1.461262 Y 2.317349
Edge Radius 0
Layer Overrides
Info
Rigidbody 2D
Body Type Dynamic
Material None (Physics Material 2D)
Simulated
Use Auto Mass
Mass 1
Linear Damping 0
Angular Damping 0.05
Gravity Scale 0
Collision Detection Discrete
Sleeping Mode Start Awake
Interpolate None
Constraints
Layer Overrides
Info
Animator
Controller BossEnemy
Avatar None (Avatar)
Apply Root Motion
Animate Physics
Update Mode Normal
Culling Mode Always Animate
Clip Count: 1
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 1
Curves Count: 1 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 1 (100.0%)
Boss Enemy (Script)
Script BossEnemy
Move Speed 2
Max Hp 500
Current Hp 0
Hp Bar Hp (Image)
Enter Damege 10
Stay Damage 1
Diem Text None (Text Mesh Pro UGUI)
Diem ZB 100
Bullet Prefabs EnemyBullet
Fire Point FirePoint (Transform)
Speed Bullet Normal 20
Speed Bullet Circle 20
Hp Value 100
Mini Enemy MiniEnemy
Skill Cool Down 1
Sprite-Lit-Default (Material)
Shader Universal Render Pipeline/2D/Sprite-Lit-Default
Edit...
Add Component


BasicEnemy
Static
Tag Enemy
Layer Default
Transform
Position X -16.75 Y -2.2 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sprite Renderer
Sprite 0_Goblin_Run Throwing_000_(
Open Sprite Editor
Color
Flip X Y
Draw Mode Simple
Mask Interaction None
Sprite Sort Point Center
Material Sprite-Lit-Default
Additional Settings
Sorting Layer <unknown layer>
Order in Layer 0
Rendering Layer M Default
Rigidbody 2D
Body Type Dynamic
Material None (Physics Material 2D)
Simulated
Use Auto Mass
Mass 1
Linear Damping 0
Angular Damping 0.05
Gravity Scale 0
Collision Detection Discrete
Sleeping Mode Start Awake
Interpolate None
Constraints
Layer Overrides
Info
Box Collider 2D
Edit Collider
Material None (Physics Material 2D)
Is Trigger
Used By Effector
Auto Tiling
Composite Operation None
Offset X 0.07976 Y -0.0797(
Size X 0.92887 Y 1.33909
Edge Radius 0
Layer Overrides
Info
Animator
Controller BasicEnemy
Avatar None (Avatar)
Apply Root Motion
Animate Physics
Update Mode Normal
Culling Mode Always Animate
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0
PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0
(0.0%)
Bassic Enemy (Script)
Script BassicEnemy
Move Speed 2
Max Hp 100
Current Hp 0
Current Hp 0
Hp Bar Hp (Image)
Enter Damege 8
Stay Damage 1
Diem Text None (Text Mesh Pro UGUI)
Diem ZB 100
Drop Items 7
Sprite-Lit-Default (Material)
Shader Universal Render Pipelin
Edit...


EnergyEnemy
Static
Tag EnemyEnergy
Layer Default
Transform
Position X -16.23 Y -9.88 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sprite Renderer
Sprite 0_Golem_Running_000_0
Open Sprite Editor
Color
Flip X Y
Draw Mode Simple
Mask Interaction None
Sprite Sort Point Center
Material Sprite-Lit-Default
Additional Settings
Sorting Layer <unknown layer>
Order in Layer 0
Rendering Layer Ma Default
Box Collider 2D
Edit Collider
Material None (Physics Material 2D)
Is Trigger
Used By Effector
Auto Tiling
Composite Operation None
Offset X 0.05002 Y -0.0510
Size X 0.86611 Y 1.18082
Edge Radius 0
Layer Overrides
Rigidbody 2D
Body Type Dynamic
Material None (Physics Material 2D)
Simulated
Use Auto Mass
Mass 1
Linear Damping 0
Angular Damping 0.05
Gravity Scale 0
Collision Detection Discrete
Sleeping Mode Start Awake
Interpolate None
Constraints
Layer Overrides
Info
Energy Enemy (Script)
Script EnergyEnemy
Move Speed 2
Max Hp 100
Current Hp 0
Hp Bar Hp (Image)
Enter Damege 10
Stay Damage 1
Diem Text None (Text Mesh Pro UGUI)
Diem ZB 100
Energy Obj Energy
Animator
Controller EnergyEnemy
Avatar None (Avatar)
Apply Root Motion
Animate Physics
Update Mode Normal


ExplosionEnemy
Static
Tag EnemyExplosion
Layer Default
Transform
Position X -16.58 Y -4.88 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sprite Renderer
Sprite 0_Golem_Running_000_0
Open Sprite Editor
Color
Flip X Y
Draw Mode Simple
Mask Interaction None
Sprite Sort Point Center
Material Sprite-Lit-Default
Additional Settings
Sorting Layer <unknown layer>
Order in Layer 0
Rendering Layer Ma Default
Rigidbody 2D
Body Type Dynamic
Material None (Physics Material 2D)
Simulated
Use Auto Mass
Mass 1
Linear Damping 0
Angular Damping 0.05
Gravity Scale 0
Collision Detection Discrete
Sleeping Mode Start Awake
Interpolate None


Box Collider 2D
Edit Collider
Material None (Physics Material 2D)
Is Trigger
Used By Effector
Auto Tiling
Composite Operation None
Offset X 0.06535 Y -0.0482
Size X 0.91396 Y 1.24677
Edge Radius 0
Layer Overrides
Info
Animator
Controller ExplosionEnemy
Avatar None (Avatar)
Apply Root Motion
Animate Physics
Update Mode Normal
Culling Mode Always Animate
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0
PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0
(0.0%)
Explosion (Script)
Script ExplosionEnemy
Move Speed 2
Max Hp 100
Current Hp 0
Hp Bar Hp (Image)
Enter Damege 10
Stay Damage 1
Diem Text None (Text Mesh Pro UGUI)

HealEnemy — Static
Tag: Enemy — Layer: Default

Transform
Position: X -16.63 Y -7.43 Z 0
Rotation: X 0 Y 0 Z 0
Scale: X 1 Y 1 Z 1

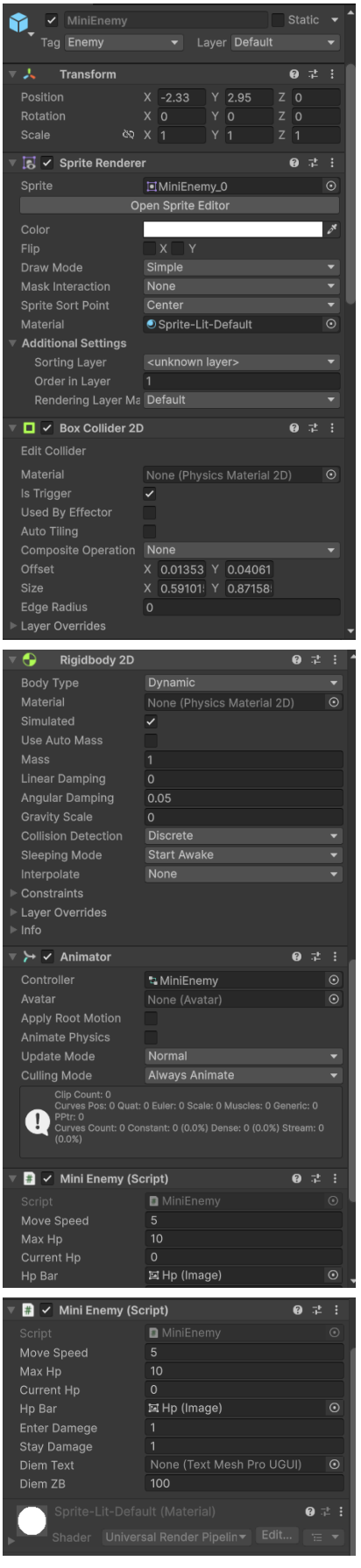
Sprite Renderer
Sprite: 0_Golem_Running_000_0
Open Sprite Editor
Color
Flip: X Y
Draw Mode: Simple
Mask Interaction: None
Sprite Sort Point: Center
Material: Sprite-Lit-Default
Additional Settings
Sorting Layer: <unknown layer>
Order in Layer: 0
Rendering Layer Ma: Default

Box Collider 2D
Edit Collider
Material: None (Physics Material 2D)
Is Trigger
Used By Effector
Auto Tiling
Composite Operation: None
Offset: X 0.04470 Y -0.10218
Size: X 0.93981 Y 1.20805
Edge Radius: 0
Layer Overrides

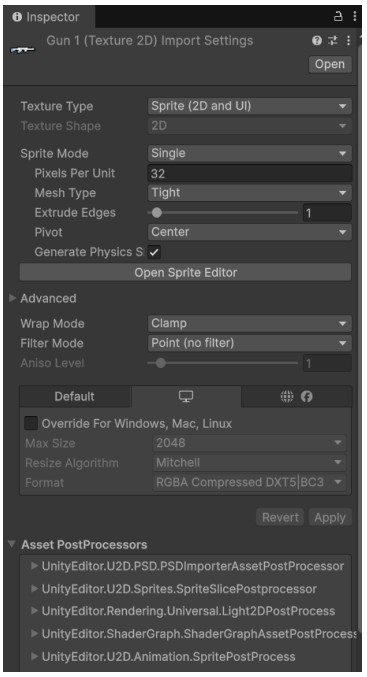
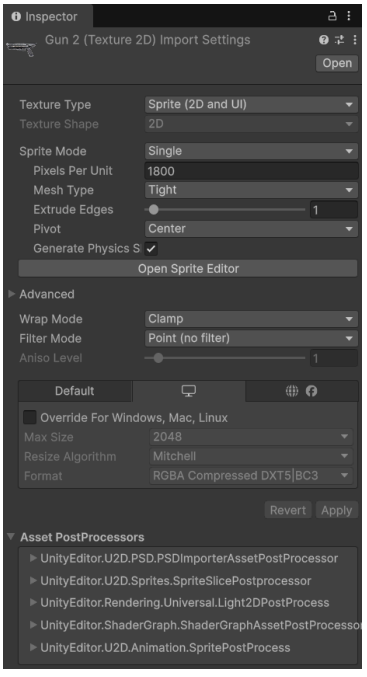
Animator
Controller: HealEnemy
Avatar: None (Avatar)
Apply Root Motion
Animate Physics
Update Mode: Normal
Culling Mode: Always Animate
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0 (0.0%)

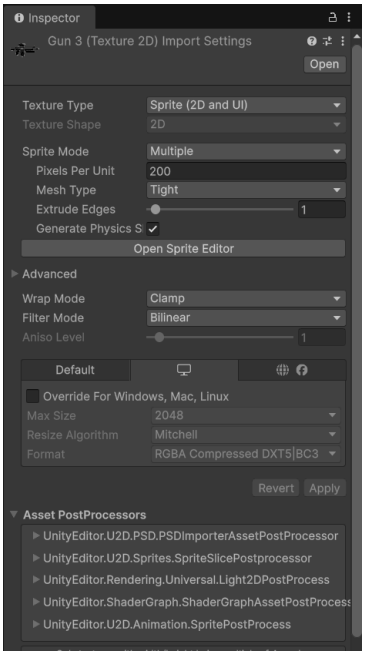
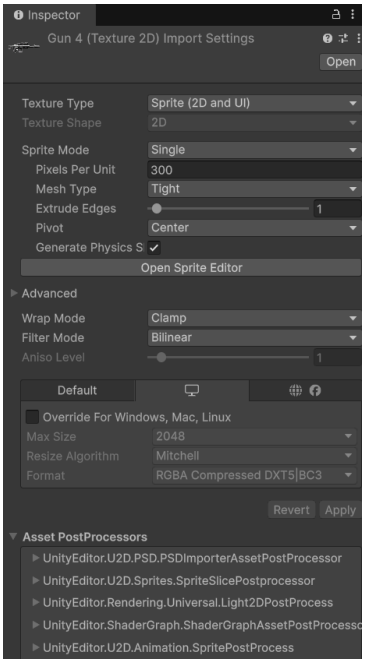
Heal Enemy (Script)
Script: healEnemy
Move Speed: 2
Max Hp: 100
Current Hp: 0
Hp Bar: Hp (Image)
Enter Damege: 10
Stay Damage: 1
Diem Text: None (Text Mesh Pro UGUI)
Diem ZB: 100
Heal: 20

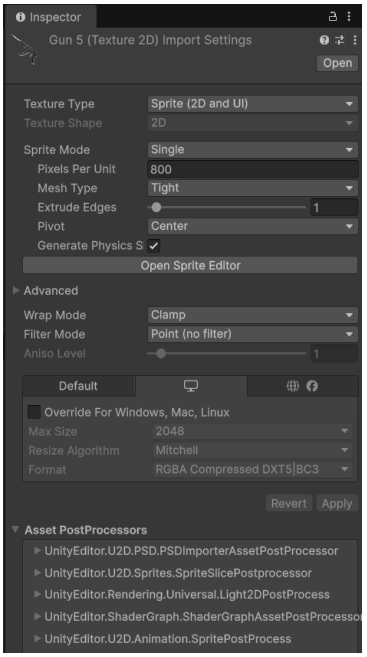
Rigidbody 2D
Body Type: Dynamic
Material: None (Physics Material 2D)
Simulated
Use Auto Mass
Mass: 1
Linear Damping: 0
Angular Damping: 0.05
Gravity Scale: 0
Collision Detection: Discrete
Sleeping Mode: Start Awake
Interpolate: None
Constraints
Layer Overrides
Info


MiniEnemy
Static
Tag Enemy
Layer Default
Transform
Position X -2.33 Y 2.95 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sprite Renderer
Sprite MiniEnemy_0
Open Sprite Editor
Color
Flip X Y
Draw Mode Simple
Mask Interaction None
Sprite Sort Point Center
Material Sprite-Lit-Default
Additional Settings
Sorting Layer <unknown layer>
Order in Layer 1
Rendering Layer Ma Default
Box Collider 2D
Edit Collider
Material None (Physics Material 2D)
Is Trigger
Used By Effector
Auto Tiling
Composite Operation None
Offset X 0.01353 Y 0.04061
Size X 0.59101 Y 0.87158
Edge Radius 0
Layer Overrides
Rigidbody 2D
Body Type Dynamic
Material None (Physics Material 2D)
Simulated
Use Auto Mass
Mass 1
Linear Damping 0
Angular Damping 0.05
Gravity Scale 0
Collision Detection Discrete
Sleeping Mode Start Awake
Interpolate None
Constraints
Layer Overrides
Info
Animator
Controller MiniEnemy
Avatar None (Avatar)
Apply Root Motion
Animate Physics
Update Mode Normal
Culling Mode Always Animate
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0 (0.0%)
Mini Enemy (Script)
Script MiniEnemy
Move Speed 5
Max Hp 10
Current Hp 0
Hp Bar Hp (Image)
Mini Enemy (Script)
Script MiniEnemy
Move Speed 5
Max Hp 10
Current Hp 0
Hp Bar Hp (Image)
Enter Damege 1
Stay Damage 1
Diem Text None (Text Mesh Pro UGUI)
Diem ZB 100
Sprite-Lit-Default (Material)
Shader Universal Render Pipelin
Edit...

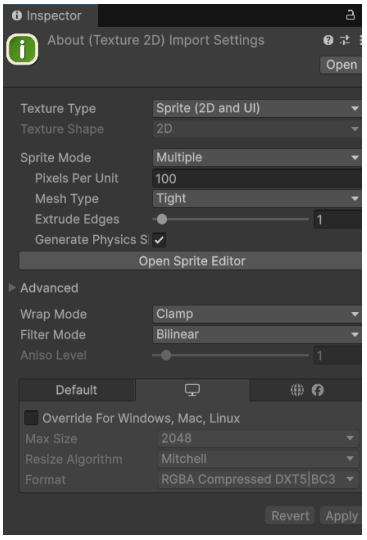
c) Weapons:

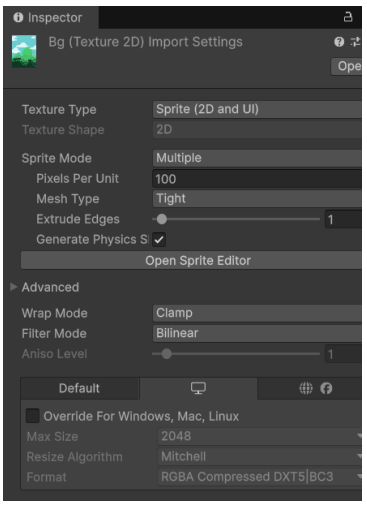
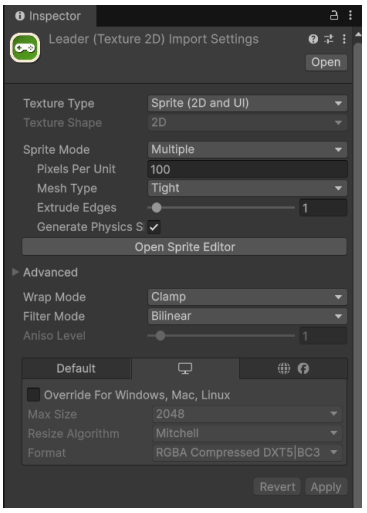
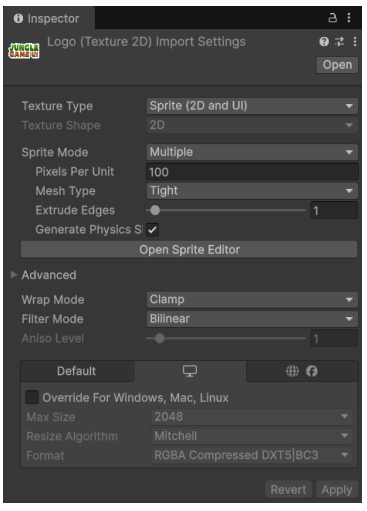
Name	Image	Inspector	Notes
Gun1			
Gun2			

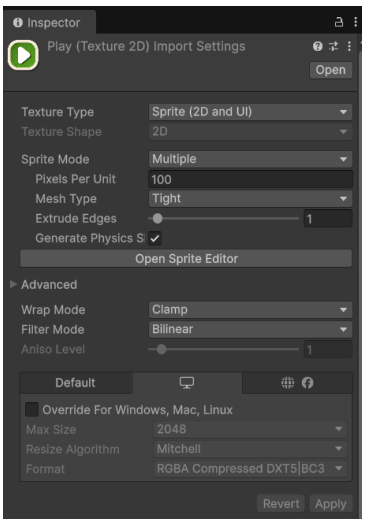
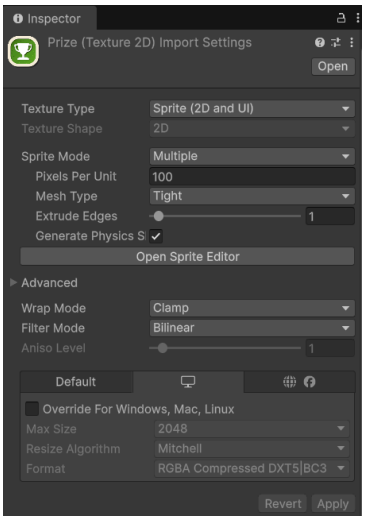
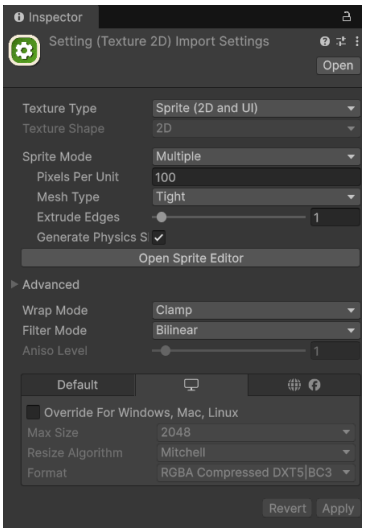
Gun3		 Inspector Gun 3 (Texture 2D) Import Settings Open Texture Type Sprite (2D and UI) Texture Shape 2D Sprite Mode Multiple Pixels Per Unit 200 Mesh Type Tight Extrude Edges 1 Generate Physics S Open Sprite Editor Advanced Wrap Mode Clamp Filter Mode Bilinear Aniso Level 1 Default Override For Windows, Mac, Linux Max Size 2048 Resize Algorithm Mitchell Format RGBA Compressed DXT5|BC3 Revert Apply Asset PostProcessors UnityEditor.U2D.PSD.PSDImporterAssetPostProcessor UnityEditor.U2D.Sprites.SpriteSlicePostprocessor UnityEditor.Rendering.Universal.Light2DPostProcess UnityEditor.ShaderGraph.ShaderGraphAssetPostProcess UnityEditor.U2D.Animation.SpritePostProcess	
Gun4		 Inspector Gun 4 (Texture 2D) Import Settings Open Texture Type Sprite (2D and UI) Texture Shape 2D Sprite Mode Single Pixels Per Unit 300 Mesh Type Tight Extrude Edges 1 Pivot Center Generate Physics S Open Sprite Editor Advanced Wrap Mode Clamp Filter Mode Bilinear Aniso Level 1 Default Override For Windows, Mac, Linux Max Size 2048 Resize Algorithm Mitchell Format RGBA Compressed DXT5|BC3 Revert Apply Asset PostProcessors UnityEditor.U2D.PSD.PSDImporterAssetPostProcessor UnityEditor.U2D.Sprites.SpriteSlicePostprocessor UnityEditor.Rendering.Universal.Light2DPostProcess UnityEditor.ShaderGraph.ShaderGraphAssetPostProcessc UnityEditor.U2D.Animation.SpritePostProcess	

Gun5			

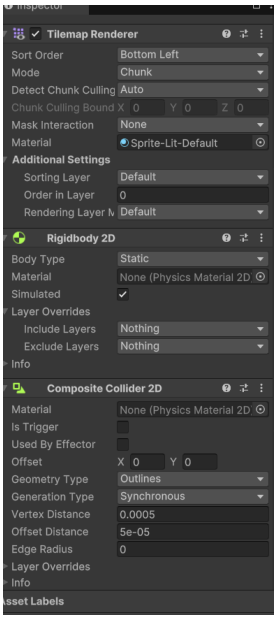
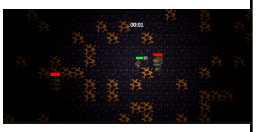
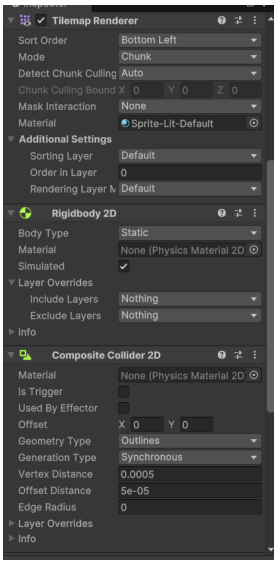
d) UI:

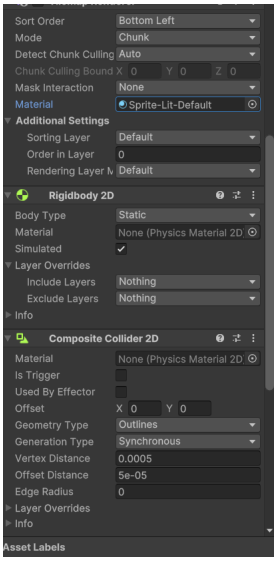
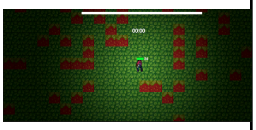
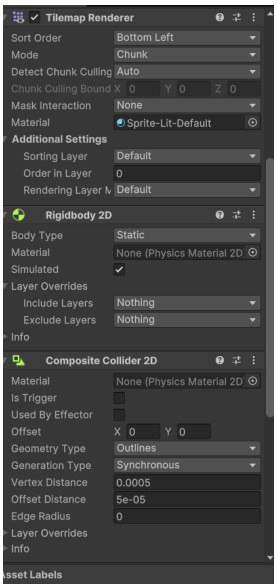
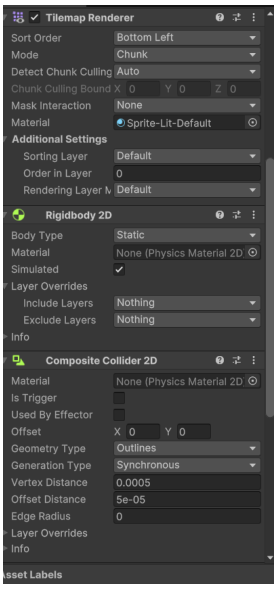
Name	Image	Inspector	Notes
About			

Bg			
Leader			
Logo			

Play			
Prize			
Setting			

e) Tilemap:

Name	Image	Inspector	Notes
Map_1			
Map_2			

Map_3			
Map_4			
Map_Endless			

3.2.3 Assets

- **Map_01 - The Dark Tide Depths:**

The journey begins in the heart of a vast and mysterious ocean. The surface ripples with deep blue waves, reflecting faint glimmers of distant light. Scattered coral-like rocks form a natural maze beneath the water's gloom, hindering the player's path forward.

At the center stands the lone survivor, weapon in hand, with a glowing health bar — a sign that the battle for survival has begun. The timer above ticks relentlessly, "00:01," echoing the tension of time running out as something unseen stirs in the depths below.

The cool blue tones and fading shadows around the edges create an eerie sense of calm before chaos. This is the first trial — a test of adaptability, where every movement matters in the deep, unforgiving sea.

- **Map_02 - The Infernal Cavern:**

After conquering the ocean's abyss, the player descends into a blazing underworld. The ground trembles beneath molten cracks, glowing with veins of flowing lava. The suffocating heat warps the air, and burnt stones shape a treacherous battlefield.

Now stronger, the hero stands ready, health bar "24" shining bright amidst the inferno. Yet the same timer — "00:01" — looms above, warning that survival here is measured by seconds.

The fiery reds and oranges contrast sharply with the ocean's cold blues, symbolizing a shift from endurance to raw aggression. This is where the true trial begins — a place where every step could mean being swallowed by fire itself.

- **Map_03 - The Abandoned Storage:**

After escaping the fiery depths, the player enters an old warehouse — dimly lit, filled with the smell of damp wood and decay. Wooden crates are scattered everywhere, serving both as cover and as obstacles that demand careful movement.

The creaking floorboards echo with every step, and the faint flicker of light through the cracks creates an uneasy atmosphere. This place feels like it holds forgotten memories — and perhaps hidden dangers. The clock still reads "00:01," signaling that another battle is about to begin.

In this stage, the challenge is no longer just the enemies, but also the **confined space** itself. Players must use the crates to dodge attacks, plan ambushes, and survive within this silent maze until the next door opens.

- **Map_04 - The Mutated Thicket:**

After escaping the deadly concrete confines of Quarantine Zone-17, the player steps into The Mutated Thicket—a segment of land utterly deformed by the *Night Plague* virus. The environment is a deep, sodden green, not of natural forest but of a dense, viscous carpet of mutated moss and bio-mass underfoot. Bloated, deep-red biological masses, resembling gigantic fleshy tumors or fungi, block all paths, forming a tight maze with only narrow corridors remaining. The air is heavy, filled only with the sound of dripping fluids and the guttural snarls of evolved Zombie strains. In this claustrophobic space, using the bio-mass blocks as cover to break line of sight and flank the enemy is the key to survival. However, the increasing appearance of faster, swarming Zombies and those that self-detonate turns this area into a death arena demanding absolute precision in movement.

- **Map_Endless - The Eternal Barrens:**

This is not a singular location, but a chilling reflection of the dead landscape surrounding Quarantine Zone-17. The arena floor is a desolate, arid brown, almost the color of clay or contaminated dust, creating a feeling of open yet merciless exposure with no sanctuary. The withered trees and stunted trunks are merely minor obstacles, offering insufficient cover and forcing the player to rely solely on their speed and reflexes.

This space is designed to push the player to their limit; every bullet and every move is critical. Zombie variants will not just arrive in waves, but in exponentially increasing frequency and density, giving the constant sensation of being relentlessly encircled from all directions. The only goal is to survive as long as possible, fighting ceaselessly in this eternal loop of devastation where hope died long ago.

3.2.4 Design Animation

No	Game Object	Animator Controller	Animation Clips
1	Player	Player_Controller	Player_Idle Player_Run Player_Die
2	Button	ButtonAnimator	ButtonShakeHover
3	Enemy	BasicEnemy BossEnemy HealthEnemy Explosion1	BasicEnemy_Run BasicEnemy_Die Boss_Run FlyEnemy_Fly HealthEnemy_Run HealthEnemy_dead MiniEnemyRun Explosion
4	Blood	Blood	Blood
5	Energy	Energy	Energy
6	Heart	Heart	Heart
7	Menu	ButtonAnimator Panel	ButtonShakeHover PanelFadeIn PanelFadeOut

3.2.5 Test case

No	Name of test case	Process (step by step)	Expected Result:
1	Basic Shooting Test	1. Start the game. 2. Ensure the player has the basic weapon equipped. 3. Aim at a stationary target (if available, otherwise just fire). 4. Press the shoot key. 5. Observe the projectile.	A projectile should be spawned from the player character in the aimed direction when the shoot key is pressed. The projectile should move forward.
2	Player Takes Damage Test	1. Start the game. 2. Allow an enemy to attack the player character. 3. Observe the player's health bar or health counter.	The player's health should decrease after being hit by an enemy. A visual or numerical representation of the health decrease should be visible in the UI.
3	Settings Button Navigation	1.Start the game. 2.Observe the main menu. 3.Click the "SETTINGS" button.	The game should transition from the main menu to the "Settings" screen, displaying the volume slider and control buttons.
4	Quit Button Functionality	1.Start the game. 2.Observe the main menu.	The game application should close completely and return

No	Name of test case	Process (step by step)	Expected Result:
		3.Click the "QUIT" button.	to the desktop or operating system.
5	Main Menu Element Visibility	1.Start the game. 2.Observe the main menu.	The "BATTLE ZONE 20" title/logo, "PLAY" button, "SETTINGS" button, and "QUIT" button should all be clearly visible and correctly positioned.

3.2.6 Test case Result

No	Name of test case	Process (step by step	Expected Result:	Results	Note (Passed/Not passed)
2	Basic Shooting Test	1. Start the game. 2. Ensure basic weapon is equipped. 3. Aim. 4. Press shoot key. 5. Observe projectiles.	A projectile is spawned and moves in the aimed direction.	Passed	
4	Player Takes Damage Test	1. Start the game. 2. Let an enemy attack the player. 3. Observe	Player health decreases visibly in UI when hit.	Passed	

No	Name of test case	Process (step by step	Expected Result:	Results	Note (Passed/Not passed)
		health.			
7	Settings Button Navigation	1. Start a game. 2. Observe the main menu. 3. Click "SETTINGS".	Navigates to settings screen showing volume slider and controls.	Passed	
8	Quit Button Functionality	1. Start a game. 2. Observe the main menu. 3. Click "QUIT".	The game exists on desktop/OS.	Passed	
10	Main Menu Button Responsiveness	1. Start a game. 2. Hover and click each main menu button.	Each button gives visual feedback on hover/click (highlight, animation).	Passed	

3.3 Code

1. Save Data

1. btnManager

```
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class btnManager : MonoBehaviour
{
    public string buttonTag = "btnShop";  // Tag để tìm các Button
    private GameObject lastUsedButton;
```

```
    private void Start()
    {
        AssignButtonEvents();
    }
    private void Update()
    {


    }

    #region Shop
    private void AssignButtonEvents()
    {

        GameObject[] buttonObjects = GameObject.FindGameObjectsWithTag(buttonTag);

        foreach (GameObject obj in buttonObjects)
        {
            Button btn = obj.GetComponent<Button>();

            if (btn != null)
            {
                TextMeshProUGUI buttonText = btn.GetComponentInChildren<TextMeshProUGUI>();

              btn.onClick.AddListener(() => goldManager.Instance.canSpend = (buttonText.text ==
"Buy"));


                btn.onClick.AddListener(() => OnShopButtonClick(obj));
            }
            else
            {
                Debug.Log("kh co btn nao duoc tim thay");
            }
        }
    }

    void OnShopButtonClick(GameObject buttonObj)
    {
        TextMeshProUGUI buttonText = buttonObj.GetComponentInChildren<TextMeshProUGUI>();
        Button btn = buttonObj.GetComponent<Button>();
        Image colorBtn = buttonObj.GetComponent<Image>();
        if (buttonText == null || btn == null) return;

        if (buttonText.text == "Buy")
        {
            if (colorBtn != null) colorBtn.color = Color.blue;
```

```
            buttonText.text = "Use";
        }


        if (buttonText.text == "Use")
        {




            // Bật lại nút trước đó (nếu có)
            if (lastUsedButton != null && lastUsedButton != buttonObj)
            {
                lastUsedButton.GetComponent<Button>().interactable = true;
                lastUsedButton.GetComponentInChildren<TextMeshProUGUI>().text = "Use";
            }

            buttonText.text = "Used";
            btn.interactable = false;
            lastUsedButton = buttonObj; // Lưu lại nút đã sử dụng
        }

    }
    #endregion

    #region  Upgrade



    #endregion


}
```

1. dataManager

```
using System.IO;
using UnityEngine;

public class dataManager : MonoBehaviour
{

    public static dataManager Instance { get; private set; }

    private string dataFilePath;
    public GameData gameData = new GameData();

    private void Awake()
    {
```

```
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
            return;
        }

        dataFilePath = Path.Combine(Application.dataPath, "gameData.json");
        LoadGameData();
    }

    public void SaveGameData()
    {
        string json = JsonUtility.ToJson(gameData, true);
        File.WriteAllText(dataFilePath, json);
        Debug.Log("Game data saved!");
    }

    public void LoadGameData()
    {
        if (File.Exists(dataFilePath))
        {
            string json = File.ReadAllText(dataFilePath);
            gameData = JsonUtility.FromJson<GameData>(json);
            Debug.Log("Game data loaded!");
        }
        else
        {
            Debug.Log(" Chưa có file gameData.json, tạo mới!");
            SaveGameData();
        }
    }


}

[System.Serializable]
public class GameData
{
    public int gold = 10000;
    public float attack = 1;
    public float moveSpeed = 5f;
    public float maxHp = 100f;
    //public float currentHp = 100f;
```

```
}
```

2. enemyManager

```
using System.Collections.Generic;
using System.IO;
using UnityEngine;


[System.Serializable]
public class EnemyData
{
   public string enemyType;
   public float x, y, z , currentHpEnemy;

}

[System.Serializable]
public class EnemyDataList // Lưu danh sách quái
{
   public List<EnemyData> enemies = new List<EnemyData>();
}

public class enemyManager : MonoBehaviour
{
   public List<GameObject> enemyPrefabs; // Danh sách Prefabs của quái
   private string path;

   public static enemyManager Instance { get; private set; }


   void Awake()
   {
      if (Instance == null)
         Instance = this;
      else
         Destroy(gameObject);
   }

   void Start()
   {
      path = Path.Combine(Application.dataPath, "enemyData.json");

   }

   void Update()
   {
      //if (Input.GetKeyDown(KeyCode.I))
      //{
```

```
    //    SaveEnemies();
    //}
    //if (Input.GetKeyDown(KeyCode.O))
    //{
    //    LoadEnemies();
    //}
}

public void SaveEnemies()
{
    EnemyDataList saveData = new EnemyDataList();
    GameObject[] allEnemies = GameObject.FindGameObjectsWithTag("Enemy");

    foreach (GameObject enemy in allEnemies)
    {
        Enemy enemyScript = enemy.GetComponent<Enemy>();
        if (enemyScript != null)
        {
            EnemyData data = new EnemyData
            {
                enemyType = enemy.name.Replace("(Clone)", "").Trim(), // Loại quái
                x = enemy.transform.position.x,
                y = enemy.transform.position.y,
                z = enemy.transform.position.z,
                currentHpEnemy = enemyScript.currentHp,
            };
            saveData.enemies.Add(data);
        }
    }

    string json = JsonUtility.ToJson(saveData, true);
    File.WriteAllText(path, json);
    Debug.Log("Đã lưu vị trí của tất cả quái vào enemyData.json");
}

public void LoadEnemies()
{
    if (!File.Exists(path))
    {
        Debug.LogWarning("Không tìm thấy file enemyData.json để load quái!");
        return;
    }

    string json = File.ReadAllText(path);
    EnemyDataList saveData = JsonUtility.FromJson<EnemyDataList>(json);

    // Xóa tất cả quái cũ trước khi load lại
    GameObject[] allEnemies = GameObject.FindGameObjectsWithTag("Enemy");
```

```
        foreach (GameObject enemy in allEnemies)
        {
            Destroy(enemy);
        }

        foreach (EnemyData data in saveData.enemies)
        {
            GameObject prefab = enemyPrefabs.Find(p => p.name == data.enemyType);
            if (prefab != null)
            {
                GameObject enemyInstance = Instantiate(prefab, new Vector3(data.x, data.y, data.z),
Quaternion.identity);
                Enemy enemyScript = enemyInstance.GetComponent<Enemy>();
                if (enemyScript != null)
                {
                    enemyScript.SetCurrentHp(data.currentHpEnemy); // Gọi phương thức SetCurrentHp để
khôi phục HP

                }

            }
            else
            {
                Debug.LogWarning("Không tìm thấy Prefab cho loại quái: " + data.enemyType);
            }
        }

        Debug.Log("Đã load tất cả quái từ enemyData.json.");
    }
    public void ResetEnemies()
    {
        if (File.Exists(path))
        {
            File.Delete(path);
        }


        GameObject[] allEnemies = GameObject.FindGameObjectsWithTag("Enemy");
        foreach (GameObject enemy in allEnemies)
        {
            Destroy(enemy);
        }


    }
```

```
}
```

3. goldManager

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;



public class goldManager : MonoBehaviour
{
   public static goldManager Instance { get; private set; }

   public TextMeshProUGUI goldText;
   public bool canSpend = true;


   private void Awake()
   {
      if (Instance == null)
      {
         Instance = this;
         DontDestroyOnLoad(gameObject);
      }
      else
      {
         Destroy(gameObject);
      }

   }
   private void Start()
   {

      if (dataManager.Instance == null)
      {
         Debug.LogError("dataManager.Instance bị NULL");
         return;
      }

      // Kiểm tra nếu goldText chưa được gán
      if (goldText == null)
      {
         Debug.LogError(" goldText chưa được gán");
         return;
      }

      UpdateGoldUI();
```

```
    }

    public int GetGold()
    {
        return dataManager.Instance.gameData.gold;
    }

    public void UpdateGoldUI()
    {
        if (goldText != null)
        {
            goldText.text = dataManager.Instance.gameData.gold.ToString();

        }
        else
        {
            Debug.LogError(" goldText bị NULL! Hãy gán TextMeshProUGUI trong Inspector (Data).");
        }
    }

    public void AddGold(int amount)
    {
        dataManager.Instance.gameData.gold += amount;
        dataManager.Instance.SaveGameData();
        UpdateGoldUI();
    }


    public void SpendGold(int amount)
    {
        if (!canSpend) { return; }
        if (dataManager.Instance.gameData.gold >= amount)
        {
            dataManager.Instance.gameData.gold -= amount;
            dataManager.Instance.SaveGameData();
            UpdateGoldUI();
        }
        else
        {
            Debug.Log("Not enough gold!");
        }
    }
}
```

4. GunSwitcher

```
using UnityEngine;
```

```
public class GunSwitcher : MonoBehaviour
{
    public GameObject[] guns; // Danh sách các súng
    private int currentGunIndex = -1; // Không có súng ban đầu
    public Transform gunHolder; // Vị trí gắn súng trên Player

    void Start()
    {
        if (guns.Length > 0)
        {
            EquipGun(1); // Trang bị súng đầu tiên nếu có
        }
        else
        {
            Debug.LogWarning("Không có súng nào trong danh sách guns!");
        }
    }

    void Update()
    {
        //// Chọn súng bằng phím số (1 -> n)
        //for (int i = 0; i < guns.Length; i++)
        //{
        //    if (Input.GetKeyDown(KeyCode.Alpha1 + i))
        //    {
        //        EquipGun(i + 1);
        //    }
        //}
    }

    public void EquipGun(int index)
    {
        if (index <= 0 || index > guns.Length) return;

        // Vô hiệu hóa tất cả súng
        for (int i = 0; i < guns.Length; i++)
        {
            bool isSelected = (i == index - 1);
            guns[i].SetActive(isSelected);

            if (isSelected)
            {
                guns[i].transform.SetParent(gunHolder);
                guns[i].transform.localPosition = Vector3.zero; // Gắn vào Player
                guns[i].transform.localRotation = Quaternion.identity;
            }
        }
```

```
        currentGunIndex = index;
        Debug.Log("Đã trang bị súng: " + index);
    }
}
```

5. playerData

```
using UnityEngine;
using System.IO;


[System.Serializable]
public class PlayerData
{
    public float x, y, z;
    public float currentHp =100f;
}
public class playerData : MonoBehaviour
{
    public Transform player; // Kéo Player vào đây trong Unity Inspector
    private string path;


    public static playerData Instance { get; private set; }

    private void Awake()
    {
        if (Instance == null) Instance = this;
        else Destroy(gameObject);
    }
    void Start()
    {
        path = Application.dataPath + "/playerData.json";
    }
    private void Update()
    {
        //if (Input.GetKeyDown(KeyCode.I))
        //{
        //    SavePlayer();
        //}
        //if (Input.GetKeyDown(KeyCode.O))
        //{
        //    LoadPlayer();
        //}
    }
    public void SavePlayer()
    {
        if (player == null)
        {
```

```
        Debug.LogWarning("Player chưa được gán vào Data (script Player Data)!");
        return;
    }
    Player playerScript = player.GetComponent<Player>();
    PlayerData data = new PlayerData
    {
        x = player.position.x,
        y = player.position.y,
        z = player.position.z,
        currentHp = playerScript.currentHp
    };

    string json = JsonUtility.ToJson(data, true);
    File.WriteAllText(path, json);
    Debug.Log(" Đã lưu tọa độ Player: " + json);
}

public void LoadPlayer()
{
    if (player == null)
    {
        Debug.LogWarning("Player chưa được gán vào PlayerSave!");
        return;
    }

    if (File.Exists(path))
    {
        string json = File.ReadAllText(path);
        PlayerData data = JsonUtility.FromJson<PlayerData>(json);

        // Debug log để kiểm tra dữ liệu đọc được từ file
        Debug.Log("Dữ liệu đọc được từ file JSON: " + json);

        player.position = new Vector3(data.x, data.y, data.z);
        Player playerScript = player.GetComponent<Player>();

        // Debug log để kiểm tra giá trị currentHp
        Debug.Log("currentHp trước khi gán: " + playerScript.currentHp);

        playerScript.currentHp = data.currentHp;

        // Debug log để kiểm tra giá trị currentHp sau khi gán
        Debug.Log("currentHp sau khi gán: " + playerScript.currentHp);

        Debug.Log("Đã load vị trí Player và currentHp từ JSON");
    }
    else
    {
```

```
            Debug.LogWarning("Không tìm thấy file JSON để load!");
        }
    }

    public void ResetPlayer()
    {

        if (File.Exists(path))
        {
            File.Delete(path);
        }

        // 2. Đặt lại vị trí Player về mặc định
        if (player != null)
        {
            player.position = new Vector3(0, 0, 0);
            Player playerScript = player.GetComponent<Player>();

            // Reset currentHp to maxHp
            playerScript.currentHp = playerScript.maxHp;
            playerScript.updateHpBar();
            Debug.Log("Đã đặt lại vị trí Player về mặc định!");
        }
    }
}
```

2. Other

1. AudioManager

```
using UnityEngine;

public class AudioManager : MonoBehaviour
{
        [SerializeField] AudioSource effectAudioSource;
        [SerializeField] AudioSource defaultAudioSource;
        [SerializeField] AudioSource bossAudioSource;
        [SerializeField] AudioClip Shotclip;
        [SerializeField] AudioClip ReloadClip;
        [SerializeField] AudioClip EnergyClip;

        public void PlayShot()
        {
                effectAudioSource.PlayOneShot(Shotclip);
        }

        public void PlayReload()
        {
```

```
            effectAudioSource.PlayOneShot(ReloadClip);
        }

        public void PlayEnerty()
        {
            effectAudioSource.PlayOneShot(EnergyClip);
        }

        public void PlayDefaultAudio()
        {
            bossAudioSource.Stop();
            defaultAudioSource.Play();
        }
        public void PlayBossAudio()
        {
            defaultAudioSource.Stop();
            bossAudioSource.Play();
        }

        public void StopAudioGame()
        {
            effectAudioSource.Stop();
            defaultAudioSource.Stop();
            bossAudioSource.Stop();
        }

}
```

2. Auto Generator Map

```
using UnityEngine;
using UnityEngine.Tilemaps;

public class autoGeneratorMap : MonoBehaviour
{
        public int width = 10;
        public int height = 10;
        public Tilemap collisionTilemap; // Chứa cả tường và chướng ngại vật
        public Tilemap groundTilemap;    // Chứa mặt đất
        public TileBase wallTile;
        public TileBase groundTile;
        public TileBase obstacleTile;

        void Start()
        {
            GenerateMap();
        }

        void GenerateMap()
```

```
        {
            for (int x = 0; x < width; x++)
            {
                for (int y = 0; y < height; y++)
                {
                    Vector3Int tilePosition = new Vector3Int(x, y, 0);

                    if (x == 0 || x == width - 1 || y == 0 || y == height - 1)
                    {
                        // Tạo tường (Wall) trong Tilemap_Collision
                        collisionTilemap.SetTile(tilePosition, wallTile);
                    }
                    else
                    {
                        // 20% tạo chướng ngại vật, 80% tạo mặt đất
                        if (Random.value > 0.8f)
                        {
                            collisionTilemap.SetTile(tilePosition, obstacleTile);

                            TilemapRenderer tilemapRenderer =
collisionTilemap.GetComponent<TilemapRenderer>();
                            tilemapRenderer.sortingLayerName = "Grass";

                        }
                        else
                        {
                            groundTilemap.SetTile(tilePosition, groundTile);

                            TilemapRenderer tilemapRenderer2 =
groundTilemap.GetComponent<TilemapRenderer>();
                            tilemapRenderer2.sortingLayerName = "Group";
                        }
                    }
                }
            }
        }
}
```

3. Basic Enemy

```
using System.Collections.Generic;
using UnityEngine;

public class BassicEnemy : Enemy
{
   [SerializeField] private List<DropItem> dropItems;
   private void OnTriggerEnter2D(Collider2D collision)
        {
            if (collision.CompareTag("Player"))
```

```
                {
                        if (player != null)
                        {
                                player.takeDame(enterDamege);
                        }
                }
        }

        private void OnTriggerStay2D(Collider2D collision)
        {
                if (collision.CompareTag("Player"))
                {
                        if (player != null)
                        {
                                player.takeDame(stayDamage);
                        }
                }
        }

    protected override void die()
    {
        GameObject dropItem = GetRandomDropItem();
        if (dropItem != null)
        {
            Instantiate(dropItem, transform.position, Quaternion.identity);
        }
        Destroy(gameObject);
    }



    private GameObject GetRandomDropItem()
    {
        float randomValue = Random.value;
        float cumulativeRate = 0f;

        foreach (var item in dropItems)
        {
            cumulativeRate += item.dropRate;
            if (randomValue <= cumulativeRate)
            {
                return item.itemPrefab;
            }
        }
        return null;
    }

}
```

```
[System.Serializable]
public struct DropItem
{
   public GameObject itemPrefab; // Prefab của vật phẩm
   [Range(0, 1)] public float dropRate; // Tỷ lệ rơi (0.0 - 1.0)
}
```

4. BossEnemy

```
using UnityEngine;

public class BossEnemy : Enemy
{
   [SerializeField] private GameObject bulletPrefabs;
   [SerializeField] private Transform firePoint;
   [SerializeField] private float speedBulletNormal = 20f;
   [SerializeField] private float speedBulletCircle = 20f;
   [SerializeField] private float hpValue = 10f;
   [SerializeField] private GameObject miniEnemy;
   [SerializeField] private float skillCoolDown = 5f;
   private float nextSkillTime = 5f;

   protected override void Update()
   {
      base.Update();
      if (Time.time >= nextSkillTime)
      {
         UseSkill();
      }
   }

   private void OnTriggerEnter2D(Collider2D collision)
   {
      if (collision.CompareTag("Player"))
      {
         player.takeDame(enterDamege);
      }
   }

   private void OnTriggerStay2D(Collider2D collision)
   {
      if (collision.CompareTag("Player"))
      {
         player.takeDame(stayDamage);
      }
   }

   private void BanDanThuong()
   {
```

```
        if(player != null)
        {
            Vector3 directionToPlayer = player.transform.position - firePoint.position;
            directionToPlayer.Normalize();
            GameObject bullet = Instantiate(bulletPrefabs, firePoint.position, Quaternion.identity);
            EnemyBullet enemyBullet = bullet.AddComponent<EnemyBullet>();
            enemyBullet.SetMoveDirection(directionToPlayer * speedBulletNormal);
        }
    }

    private void BanDanTron()
    {
        const int bulletCount = 12;
        float angleStep = 360f / bulletCount;
        for(int i = 0; i < 12; i++)
        {
            float angle = i * angleStep;
            Vector3 bulletDirection = new Vector3(Mathf.Cos(Mathf.Deg2Rad * angle),
Mathf.Sin(Mathf.Deg2Rad * angle));
            GameObject bullet = Instantiate(bulletPrefabs, transform.position, Quaternion.identity);
            EnemyBullet enemyBullet = bullet.AddComponent<EnemyBullet>();
            enemyBullet.SetMoveDirection(bulletDirection * speedBulletCircle);
        }
    }

    private void HoiMau(float hpAmount)
    {
        currentHp = Mathf.Min(currentHp + hpAmount, maxHp);
        updateHpBar();
    }

    private void CreateMiniEnemy()
    {
        Instantiate(miniEnemy, transform.position, Quaternion.identity);
    }

    private void DichChuyen()
    {
        if (player != null)
        {
            transform.position = player.transform.position;
        }
    }

    private void RandomSkill()
    {
        int ranDom = Random.Range(0, 4);
        switch (ranDom)
```

```
        {
            case 0:
                BanDanThuong();
                break;
            case 1:
                BanDanTron();
                break;
            case 2:
                HoiMau(hpValue);
                break;
            case 3:
                CreateMiniEnemy();
                break;
        }
    }

    private void UseSkill()
    {
        nextSkillTime = Time.time + skillCoolDown;
        RandomSkill();
    }
}
```

5. BoxRandom

```
using UnityEngine;

public class BoxRandom : MonoBehaviour
{
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

6. ChunkTrigger

```
using UnityEngine;

public class ChunkTrigger : MonoBehaviour
{
    MapController mc;
    public GameObject targetMap;
```

```
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {
        mc = FindObjectOfType<MapController>();
    }

    // Update is called once per frame
    void Update()
    {

    }

    private void OnTriggerStay2D(Collider2D col)
    {
        if (col.CompareTag("Player"))
        {
            mc.currentChunk = targetMap;
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            if(mc.currentChunk = targetMap)
            {
                mc.currentChunk = null;
            }
        }
    }
}
```

7. CursorManager

```
using UnityEngine;

public class CursorManage : MonoBehaviour
{
    [SerializeField] private Texture2D cursorNomal;
    [SerializeField] private Texture2D cursorShoot;
    [SerializeField] private Texture2D cursorReload;
    private Vector2 hotspot = new Vector2(32, 32);
    void Start()
    {
        Cursor.SetCursor(cursorNomal, hotspot, CursorMode.Auto);
    }

    void Update()
```

```
    {
        if (Input.GetMouseButtonDown(0))
        {
            Cursor.SetCursor(cursorShoot, hotspot, CursorMode.Auto);
        }
        else if (Input.GetMouseButtonUp(0))
        {
            Cursor.SetCursor(cursorNomal, hotspot, CursorMode.Auto);
        }

        if (Input.GetKeyDown(KeyCode.R))
        {
            Cursor.SetCursor(cursorReload, hotspot, CursorMode.Auto);
        }
        else if (Input.GetKeyUp(KeyCode.R))
        {
            Cursor.SetCursor(cursorNomal, hotspot, CursorMode.Auto);
        }
    }

}
```

8. demTime

```
using UnityEngine;

public class CursorManage : MonoBehaviour
{
    [SerializeField] private Texture2D cursorNomal;
    [SerializeField] private Texture2D cursorShoot;
    [SerializeField] private Texture2D cursorReload;
    private Vector2 hotspot = new Vector2(32, 32);
    void Start()
    {
        Cursor.SetCursor(cursorNomal, hotspot, CursorMode.Auto);
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Cursor.SetCursor(cursorShoot, hotspot, CursorMode.Auto);
        }
        else if (Input.GetMouseButtonUp(0))
        {
            Cursor.SetCursor(cursorNomal, hotspot, CursorMode.Auto);
        }

        if (Input.GetKeyDown(KeyCode.R))
        {
```

```
            Cursor.SetCursor(cursorReload, hotspot, CursorMode.Auto);
        }
        else if (Input.GetKeyUp(KeyCode.R))
        {
            Cursor.SetCursor(cursorNomal, hotspot, CursorMode.Auto);
        }
    }

}
```

9. Enemy

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public abstract class Enemy : MonoBehaviour
{
        [SerializeField] public float moveSpeed = 1.0f;

        protected Player player;

        [SerializeField] public float maxHp = 50f;
        public float currentHp;
        [SerializeField] private Image hpBar;

        [SerializeField] public float enterDamege = 10f;
        [SerializeField] public float stayDamage = 1f;

        [SerializeField] public TextMeshProUGUI diemText;
        private static float diem = 0f;
        [SerializeField] public float diemZB = 100f;

        protected virtual void Start()
        {
                player = FindAnyObjectByType<Player>();
                if (currentHp <= 0)
                {
                        currentHp = maxHp;
                }

                updateHpBar();
                //enemyManager.Instance.LoadEnemies();

                if (diemText == null)
                {
                        diemText =
GameObject.Find("Diem").GetComponent<TextMeshProUGUI>();
                }
```

```
    }
    protected virtual void Update()
    {
        moveToPlayer();
    }

    protected void moveToPlayer()
    {
        if (player != null)
        {
            transform.position = Vector2.MoveTowards(transform.position,
player.transform.position, moveSpeed * Time.deltaTime);
            flipEnemy();
        }
    }

    protected void flipEnemy()
    {
        if (player != null)
        {
            transform.localScale = new Vector3(player.transform.position.x <
transform.position.x ? -1 : 1, 1, 1);
        }
    }
    public virtual void takeDame(float damege)
    {
        currentHp -= damege;
        currentHp = Mathf.Max(currentHp, 0);
        updateHpBar();
        if (currentHp <= 0)
        {
            diem += diemZB;
            if (Time.timeScale == 0)
            {
                diemText.text = "";
            }
            else
            {
                diemText.text = diem.ToString();
            }
            die();
        }
    }

    protected virtual void die()
    {
        Destroy(gameObject);
    }
```

```
        public void updateHpBar()
        {
                if (hpBar != null)
                {
                        hpBar.fillAmount = currentHp / maxHp;
                }
        }

        public void SetCurrentHp(float hp)
        {
                Debug.Log("Before update: currentHp = " + currentHp);
                currentHp = hp;
                updateHpBar();
                Debug.Log("After update: currentHp = " + currentHp);
        }



}
```

10. EnemyBullet

```
using UnityEngine;

public class EnemyBullet : MonoBehaviour
{
   private Vector3 movementDirection;


   void Start()
   {
      Destroy(gameObject, 5f);
   }

   // Update is called once per frame
   void Update()
   {
      if(movementDirection == Vector3.zero) return;
      transform.position += movementDirection * Time.deltaTime;
   }

   public void SetMoveDirection(Vector3 direction)
   {
      movementDirection = direction;
   }
}
```

11. EnemySpam

```
using System.Collections;
using UnityEngine;
```

```
public class EnemySpam : MonoBehaviour
{
   [SerializeField] private GameObject[] enemies;
   [SerializeField] private Transform[] spamPoints;
   [SerializeField] private float timeDelay = 1f;
   void Start()
   {
      StartCoroutine(spamEnemy());
   }
   void Update()
   {

   }

   private IEnumerator spamEnemy()
   {
      while (true) {
         yield return new WaitForSeconds(timeDelay);
         GameObject enemy = enemies[Random.Range(0,enemies.Length)];
         Transform spamPoint = spamPoints[Random.Range(0, spamPoints.Length)];
         Instantiate(enemy, spamPoint.position,Quaternion.identity);
      }
   }
}
```

12. Energy Enemy

```
using UnityEngine;

public class EnergyEnemy : Enemy
{
   [SerializeField] private GameObject energyObj;
   private void OnTriggerEnter2D(Collider2D collision)
   {
      if (collision.CompareTag("Player"))
      {
         if (player != null)
         {
            player.takeDame(enterDamege);
         }
      }
   }
   private void OnTriggerStay2D(Collider2D collision)
   {
      if (collision.CompareTag("Player"))
      {
         if(player != null)
         {
            player.takeDame(stayDamage);
         }
```

```
        }
    }
    protected override void die()
    {
        if (gameObject != null) {
            GameObject energy = Instantiate(energyObj, transform.position, Quaternion.identity);
            Destroy(energy,5f);
        }
        base.die();
    }
}
```

13. explosion

```
using UnityEngine;

public class explosion : MonoBehaviour
{
    [SerializeField] private float damage =10f;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        Player player = collision.GetComponent<Player>();
        Enemy enemy = collision.GetComponent<Enemy>();
        if (collision.CompareTag("Player"))
        {
            player.takeDame(damage);
        }
        if (collision.CompareTag("Enemy"))
        {
            enemy.takeDame(damage);
        }
    }
    public void DestroyExplosion()
    {
        Destroy(gameObject);
    }
}
```

14. ExplosionEnemy

```
using UnityEngine;

public class Explosion : Enemy
{
    [SerializeField] private GameObject explorePrefabs;

    private void CreateExplosion()
    {
        if (explorePrefabs != null) {

        Instantiate(explorePrefabs, transform.position, Quaternion.identity);
```

```
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            CreateExplosion();
        }
    }
    protected override void die()
    {
        CreateExplosion() ;
        base.die();
    }

    //private void OnTriggerStay2D(Collider2D collision)
    //{
    //    if (collision.CompareTag("Player"))
    //    {
    //        if (player != null)
    //        {
    //            player.takeDame(stayDamage);
    //        }
    //    }
    //}
}
```

15. GameManager

```
using System.IO;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;


public class GameManager : MonoBehaviour
{
    private int currentEnergy;
    [SerializeField] private int energyHold = 3;
    [SerializeField] private GameObject boss;
    private bool callBoss = false;
    [SerializeField] private GameObject enemySpam;
    [SerializeField] private Image energyBar;
    [SerializeField] private GameObject[] panels;


    [SerializeField] GameObject mana;
    [SerializeField] private GameObject gold;

    [SerializeField] private AudioManager audioManager;
```

```
[SerializeField] private float powerUpInterval = 30f;
private float nextPowerUpTime = 0f;

void Start()
{
   currentEnergy = 0;
   UpdateEnergyBar();
   boss.SetActive(false);
   MainMenu();
   Home();
                audioManager.StopAudioGame();
      }

void Update()
{
   if (Time.time >= nextPowerUpTime)
   {
      nextPowerUpTime = Time.time + powerUpInterval;
      PowerUpPlayer();
      PowerUpEnemies();
   }
}

private void PowerUpPlayer()
{
   Player player = FindObjectOfType<Player>();
   if (player != null)
   {
      player.maxHp += 10;
      player.attack += 2;
      player.moveSpeed *= 1.05f;
   }
}

public void PowerUpEnemies()
{
   Enemy[] enemies = FindObjectsOfType<Enemy>();
   foreach (Enemy enemy in enemies)
   {
      enemy.maxHp *= 1.2f;
      enemy.enterDamege *= 1.15f;
      enemy.stayDamage *= 1.15f;
   }
}



public void addE()
```

```
{
    if (callBoss)
    {
        return;
    };
    currentEnergy += 1;
    UpdateEnergyBar();
    if (currentEnergy == energyHold)
    {
        CallBoss();
    }
}
public void CallBoss()
{
    callBoss = true;
    boss.SetActive(true);
    //enemySpam.SetActive(false);

    //hidden energyBar
    mana.SetActive(false);
                audioManager.PlayBossAudio();

     }
private void UpdateEnergyBar()
{
    if (energyBar != null)
    {
        float fill = Mathf.Clamp01((float) currentEnergy / (float)energyHold);
        energyBar.fillAmount = fill;
    }

}


public void MainMenu()
{
    //mainMenu.SetActive(true);
    //pauseMenu.SetActive(false);
    //overMenu.SetActive(false);

    ShowPanelByName("MainMenu");
    Time.timeScale = 0f;
}
public void OverMenu()
{
    ShowPanelByName("GameOver");
    Time.timeScale = 0f;
```

```
}
public void PauseMenu()
{
   ShowPanelByName("GamePause");
   Time.timeScale = 0f;
                audioManager.StopAudioGame();
       }
public void NewGame()
{
   playerData.Instance.ResetPlayer();
   enemyManager.Instance.ResetEnemies();
   ShowPanelByName();
   mana.SetActive(true);
   Time.timeScale = 1f;
                audioManager.PlayDefaultAudio();
       }
public void ResumeGame()
{
   ShowPanelByName();
   Time.timeScale = 1f;
}

public void Shop()
{

   ShowPanelByName("Shop");
   gold.SetActive(true);
   Time.timeScale = 0f;
}
public void Home()
{
   ShowPanelByName("Home");
   Time.timeScale = 0f;
       }
public void UpGrade()
{

   ShowPanelByName("UpGrade");
   gold.SetActive(true);
   Time.timeScale = 0f;
}
public void Save_Quit()
{
   enemyManager.Instance.SaveEnemies();
   playerData.Instance.SavePlayer();
   ShowPanelByName("Home");
   Time.timeScale = 0f;
}
```

```
public void Continue()
{
   enemyManager.Instance.LoadEnemies();
   playerData.Instance.LoadPlayer();
   mana.SetActive(true);
   ShowPanelByName();
   Time.timeScale = 1f;
                audioManager.PlayDefaultAudio();
       }
       public void ContinuePauseMenu()
{

   mana.SetActive(true);
   ShowPanelByName();
   Time.timeScale = 1f;
                audioManager.PlayDefaultAudio();
       }
public void Quit()
{

   ShowPanelByName("Home");
   Time.timeScale = 0f;
}

public void ChooseScene()
{
   ShowPanelByName("ChooseLevel");
   Time.timeScale = 0f;
}
//public void RestartGame()
//{

//    playerData.Instance.ResetPlayer();
//    enemyManager.Instance.ResetEnemies();



//    mana.SetActive(true);
//    //mana.SetActive(false);
//    //ShowPanelByName();
//    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
//    Time.timeScale = 1f;

//}


public void ShowPanelByName(string panelName)
{
```

```
        // Ẩn tất cả panel
        foreach (GameObject panel in panels)
        {
            //panel.SetActive(false);
            panel.SetActive(panel.name == panelName);
        }

        mana.SetActive(true);

    }
    public void ShowPanelByName()
    {
        // Ẩn tất cả panel
        foreach (GameObject panel in panels)
        {
            panel.SetActive(false);

        }



    }

}
```

16. GameUI

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class GameUI : MonoBehaviour
{
    [SerializeField] private GameManager gameManager;
    public void StartGame()
    {
        gameManager.NewGame();
    }
    public void QuitGame()
    {
        Application.Quit();
    }
    public void ContiueGame()
    {
        gameManager.ResumeGame();
    }
    public void MainMenu()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

17. Gun

```
using System.Collections;
using TMPro;
using Unity.Jobs;
using UnityEngine;

public class Gun : MonoBehaviour
{
        private float rotateOffset = 180f;
        [SerializeField] private Transform firePos;
        [SerializeField] private GameObject bulletPrefabs;
        [SerializeField] private float shotDelay = 0.15f;
        private float nextShot;
        [SerializeField] private int maxAmmo = 24;
        [SerializeField] private int currentAmmo;
   private int defaultMaxAmmo;
   [SerializeField] private TextMeshProUGUI amoText;
        [SerializeField] private TextMeshProUGUI timeBuff;

        private bool isBuff = false;
        private float buffEndTime = 0f;
   [SerializeField] private int levelBuff = 1;

        [SerializeField] private AudioManager audioManager;
        // Start is called once before the first execution of Update after the MonoBehaviour is created
        void Start()
        {
      defaultMaxAmmo = maxAmmo;
      currentAmmo = maxAmmo;
                updateAmoText();
        }

        // Update is called once per frame
        void Update()
        {
                RotateGun();
                ReLoad();
      ShotByNumberBullet(levelBuff);
                updateTimeBuff();
        }

        void RotateGun()
        {
                if (Input.mousePosition.x < 0 || Input.mousePosition.x > Screen.width ||
Input.mousePosition.y < 0 || Input.mousePosition.y > Screen.width)
                {
                        return;
```

```
                }
                Vector3 displacement = transform.position -
Camera.main.ScreenToWorldPoint(Input.mousePosition);
                float angle = Mathf.Atan2(displacement.y, displacement.x) * Mathf.Rad2Deg;
                transform.rotation = Quaternion.Euler(0, 0, angle + rotateOffset);
                if (angle < -90 || angle > 90)
                {
                        transform.localScale = new Vector3(1, 1, 1);
                }
                else
                {
                        transform.localScale = new Vector3(1, -1, 1);
                }
        }
        void Shot()
        {
    if (Time.timeScale == 0f) return;
    if (Input.GetMouseButtonDown(0) && currentAmmo > 0 && Time.time > 0)
                {
                        nextShot = Time.time + shotDelay;
                        Instantiate(bulletPrefabs, firePos.position, firePos.rotation);
                        currentAmmo--;
                        updateAmoText();
                        audioManager.PlayShot();
                }
        }

        void ShotByNumberBullet(int bullet)
        {
    if(levelBuff % 5 == 0)
                {
                        bullet = 5;
    }else
                {
                        bullet = levelBuff % 5;
    }
    if (Input.GetMouseButtonDown(0) && currentAmmo > 0 && Time.time > nextShot)
                {
                        if(currentAmmo < bullet)
                        {
                                bullet = currentAmmo;
      }
                        nextShot = Time.time + shotDelay;
       Quaternion leftRotation = firePos.rotation * Quaternion.Euler(0, 0, -15);
       Quaternion leftRotation2 = firePos.rotation * Quaternion.Euler(0, 0, -30);
       Quaternion rightRotation = firePos.rotation * Quaternion.Euler(0, 0, 15);
       Quaternion rightRotation2 = firePos.rotation * Quaternion.Euler(0, 0, 30);
       if (bullet == 1)
```

```
                {
    Instantiate(bulletPrefabs, firePos.position, firePos.rotation);
} else if (bullet == 2)
                {
    Instantiate(bulletPrefabs, firePos.position, leftRotation);
    Instantiate(bulletPrefabs, firePos.position, rightRotation);
}
else if (bullet == 3)
{
    Instantiate(bulletPrefabs, firePos.position, firePos.rotation);
    Instantiate(bulletPrefabs, firePos.position, leftRotation);
    Instantiate(bulletPrefabs, firePos.position, rightRotation);
}
else if (bullet == 4)
{
    Instantiate(bulletPrefabs, firePos.position, leftRotation);
    Instantiate(bulletPrefabs, firePos.position, rightRotation);
                        Instantiate(bulletPrefabs, firePos.position, leftRotation2);
    Instantiate(bulletPrefabs, firePos.position, rightRotation2);
}
else if (bullet == 5)
{
    Instantiate(bulletPrefabs, firePos.position, firePos.rotation);
    Instantiate(bulletPrefabs, firePos.position, leftRotation);
    Instantiate(bulletPrefabs, firePos.position, rightRotation);
    Instantiate(bulletPrefabs, firePos.position, leftRotation2);
    Instantiate(bulletPrefabs, firePos.position, rightRotation2);
}
currentAmmo -= bullet;
updateAmoText();
audioManager.PlayShot();
            }
 }
 private void updateTimeBuff()
 {
            if (timeBuff != null)
            {
                        if (isBuff)
                        {
                                    float timeLeft = buffEndTime - Time.time;
                                    timeBuff.text = Mathf.CeilToInt(timeLeft).ToString();
                        }
                        else
                        {
                                    timeBuff.text = "";
                        }
            }
 }
```

```
    public void ActivateBuff(int plusLevelBuff)
    {
            isBuff = true;
            if(levelBuff + plusLevelBuff >= 1)
            {
   levelBuff = levelBuff + plusLevelBuff;
} else
            {
                    levelBuff = 1;
}
buffEndTime = Time.time + 15f;
            StopAllCoroutines();
            StartCoroutine(DisableBuffAfterTime(15f));
    }

    private IEnumerator DisableBuffAfterTime(float duration)
    {
            yield return new WaitForSeconds(duration);
            isBuff = false;
    }

    void ReLoad()
    {
            if (Input.GetKeyDown(KeyCode.R) && currentAmmo < maxAmmo)
            {
                    StartCoroutine(DelayReLoad(1f));
                    audioManager.PlayReload();
            }
    }

    private IEnumerator DelayReLoad(float duration)
    {
            yield return new WaitForSeconds(duration);
            currentAmmo = maxAmmo;
            updateAmoText();
    }


    private void updateAmoText()
    {
            if (amoText != null)
            {
                    if (currentAmmo > 0)
                    {
                            amoText.text = currentAmmo.ToString();
                    }
                    else
```

```
                        {
                            amoText.text = "0";
                        }
                    }
                }
    public void ActivateAmmoBuff()
    {
        StopCoroutine(ResetAmmoBuff());

        maxAmmo = 50;
        currentAmmo = maxAmmo;
        updateAmoText();

        StartCoroutine(ResetAmmoBuff());
    }
    private IEnumerator ResetAmmoBuff()
    {
        yield return new WaitForSeconds(15f);
        maxAmmo = defaultMaxAmmo;
        if (currentAmmo > maxAmmo)
        {
            currentAmmo = maxAmmo;
        }
        updateAmoText();
    }

}
```

18. HealEnemy

```
using UnityEngine;

public class healEnemy : Enemy
{
    [SerializeField] private float heal = 20f;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            if (player != null)
            {
                player.takeDame(enterDamege);
            }
        }
    }
    private void OnTriggerStay2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
```

```
            if (player != null)
            {
                player.takeDame(stayDamage);
            }
        }
    }
    protected override void die()
    {
        HealHp();
        base.die();
    }
    private void HealHp()
    {
        if (player != null)
        {
            player.Heal(heal);
        }
    }
}
```

19. MapController

```
using System.Collections.Generic;
using UnityEngine;

public class MapController : MonoBehaviour
{
    public List<GameObject> terrainChunks;
    public GameObject player;
    public float checkerRadius;
    public Vector3 noTerrainPosition;
    public LayerMask terrainMask;
    Player pm;
    public GameObject currentChunk;

    [Header("Optimazation")]
    public List<GameObject> spawnedChunks;
    GameObject lastesChunk;
    public float maxOpDist;
    float opDist;

    float optimazerCoolDown;
    public float optimazerCoolDur;


    void Start()
    {
        pm = FindObjectOfType<Player>();
    }
```

```
    void Update()
    {
        ChunkChecker();
        ChunkOptimizer();
    }

    void ChunkChecker()
    {
        if (!currentChunk)
        {
            return;
        }



        Vector2 moveDir = new Vector2(Input.GetAxisRaw("Horizontal"),
Input.GetAxisRaw("Vertical"));

        if (moveDir.x > 0 && moveDir.y == 0) //right
        {
            if (!Physics2D.OverlapCircle(currentChunk.transform.Find("Right").position, checkerRadius,
terrainMask))
            {
                noTerrainPosition = currentChunk.transform.Find("Right").position;
                SpawnChunk();
            }
        }
        else if (moveDir.x < 0 && moveDir.y == 0) //left
        {
            if (!Physics2D.OverlapCircle(currentChunk.transform.Find("Left").position, checkerRadius,
terrainMask))
            {
                noTerrainPosition = currentChunk.transform.Find("Left").position;
                SpawnChunk();
            }
        }
        else if (moveDir.x == 0 && moveDir.y > 0)
        {
            if (!Physics2D.OverlapCircle(currentChunk.transform.Find("Up").position, checkerRadius,
terrainMask))
            {
                noTerrainPosition = currentChunk.transform.Find("Up").position;
                SpawnChunk();
            }
        }
        else if (moveDir.x == 0 && moveDir.y < 0)
        {
```

```
            if (!Physics2D.OverlapCircle(currentChunk.transform.Find("Down").position, checkerRadius,
terrainMask))
            {
                noTerrainPosition = currentChunk.transform.Find("Down").position;
                SpawnChunk();
            }
        }
        else if (moveDir.x > 0 && moveDir.y > 0) //right up
        {
            if (!Physics2D.OverlapCircle(currentChunk.transform.Find("Right Up").position,
checkerRadius, terrainMask))
            {
                noTerrainPosition = currentChunk.transform.Find("Right Up").position;
                SpawnChunk();
            }
        }
        else if (moveDir.x > 0 && moveDir.y < 0)
        {
            if (!Physics2D.OverlapCircle(currentChunk.transform.Find("Right Down").position,
checkerRadius, terrainMask))
            {
                noTerrainPosition = currentChunk.transform.Find("Right Down").position;
                SpawnChunk();
            }
        }
        else if (moveDir.x < 0 && moveDir.y > 0)
        {
            if (!Physics2D.OverlapCircle(currentChunk.transform.Find("Left Up").position,
checkerRadius, terrainMask))
            {
                noTerrainPosition = currentChunk.transform.Find("Left Up").position;
                SpawnChunk();
            }
        }
        else if (moveDir.x < 0 && moveDir.y < 0)
        {
            if (!Physics2D.OverlapCircle(currentChunk.transform.Find("Left Down").position,
checkerRadius, terrainMask))
            {
                noTerrainPosition = currentChunk.transform.Find("Left Down").position;
                SpawnChunk();
            }
        }
    }

    void SpawnChunk()
    {
        int rand = Random.Range(0, terrainChunks.Count);
```

```
        lastesChunk = Instantiate(terrainChunks[rand], noTerrainPosition, Quaternion.identity);
        spawnedChunks.Add(lastesChunk);
    }
    void ChunkOptimizer()
    {
        optimazerCoolDown -= Time.deltaTime;
        if (optimazerCoolDown <= 0f)
        {
            optimazerCoolDown = optimazerCoolDur;
        }
        else
        {
            return;
        }

        foreach (GameObject chunk in spawnedChunks)
        {
            opDist = Vector3.Distance(player.transform.position, chunk.transform.position);
            if (opDist > maxOpDist)
            {
                chunk.SetActive(false);
            }
            else
            {
                chunk.SetActive(true);
            }
        }
    }

}
```

20. MiniEnemy

```
using UnityEngine;

public class MiniEnemy : Enemy
{

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            player.takeDame(enterDamege);
        }
    }

    private void OnTriggerStay2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
```

```
        {
            player.takeDame(stayDamage);
        }
    }
}
```

21. nextLevel

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class nextLevel : MonoBehaviour
{

    public List<string> sceneNames = new List<string>();

    void Update()
    {
        // Kiểm tra các phím từ 1 đến 5 và tải scene tương ứng
        for (int i = 0; i < 5; i++)
        {
            if (Input.GetKeyDown(KeyCode.Alpha0 + i) && i < sceneNames.Count)
            {
                LoadScene(i);
                break; // Ngừng vòng lặp sau khi tải scene
            }
        }
    }


    public void LoadScene(int sceneIndex)
    {
        LoadNextScene(sceneIndex);
    }
    public void LoadNextScene(int sceneIndex)
    {
        if (sceneNames.Count == 0 || sceneIndex >= sceneNames.Count)
        {
            Debug.LogWarning("No scenes to load or invalid index.");
            return;
        }

        SceneManager.LoadScene(sceneNames[sceneIndex]);
    }
}
```

22. Player

```
using UnityEngine;
using UnityEngine.UI;
```

```
public class Player : MonoBehaviour
{
        public float moveSpeed = 5f;
   public float attack = 1;
   public float maxHp = 100f;
   public float currentHp;
   private Rigidbody2D rb;
        private SpriteRenderer spriteRenderer;
        private Animator animator;



        [SerializeField] private Image hpBar;

        [SerializeField] private GameManager gameManager;
        private void Awake()
        {
                rb = GetComponent<Rigidbody2D>();
                spriteRenderer = GetComponent<SpriteRenderer>();
                animator = GetComponent<Animator>();
        }
        void Start()
   {
      LoadDataFromManager();
      playerData.Instance.LoadPlayer();


      // Ensure currentHp is properly initialized
      if (currentHp <= 0)
      {
         currentHp = maxHp;
      }

      // Update the health bar
      updateHpBar();
   }

        void Update()
        {
                MovePlayer();
                if (Input.GetKeyDown(KeyCode.Escape))
                {
                        gameManager.PauseMenu();
                }
        }

        void MovePlayer()
        {
```

```
                Vector2 playerInput = new Vector2(Input.GetAxisRaw("Horizontal"),
Input.GetAxisRaw("Vertical"));
                rb.linearVelocity = playerInput * moveSpeed;
                if (playerInput.x < 0)
                {
                        spriteRenderer.flipX = true;
                }
                else if (playerInput.x > 0)
                {
                        spriteRenderer.flipX = false;
                }
                if (playerInput != Vector2.zero)
                {
                        animator.SetBool("isRun", true);
                }
                else
                {
                        animator.SetBool("isRun", false);
                }
        }

        public void takeDame(float damege)
        {
                currentHp -= damege;
                currentHp = Mathf.Max(currentHp, 0);
                updateHpBar();
      playerData.Instance.SavePlayer();
      if (currentHp <= 0)
                {
                        die();
                }
        }

        private void die()
        {
                //Destroy(gameObject);
                gameManager.OverMenu();
        }

        public void updateHpBar()
        {
                if (hpBar != null)
                {
                        hpBar.fillAmount = currentHp / maxHp;
                }
        }
        public void Heal(float hp)
        {
```

```
                if(currentHp < maxHp)
                {
                        currentHp += hp;
                        currentHp = Mathf.Min(currentHp, maxHp);
                        updateHpBar();
                }
        playerData.Instance.SavePlayer();
    }

    #region updatefunction
    public void UpgradeAttack(float amount)
    {
                goldManager.Instance.canSpend = true;
                goldManager.Instance.SpendGold(10);
        attack += amount;
        dataManager.Instance.gameData.attack = attack;
        dataManager.Instance.SaveGameData();
    }
    public void UpgradeMoveSpeed(float amount)
    {
        goldManager.Instance.canSpend = true;
        goldManager.Instance.SpendGold(10);
        moveSpeed += amount/10;
        dataManager.Instance.gameData.moveSpeed = moveSpeed;
        dataManager.Instance.SaveGameData();
    }
    public void UpgradeMaxHp(float amount)
    {
        goldManager.Instance.canSpend = true;
        goldManager.Instance.SpendGold(10);
        maxHp += amount;
        dataManager.Instance.gameData.maxHp = maxHp;
        dataManager.Instance.SaveGameData();
    }
    #endregion
    private void LoadDataFromManager()
    {
        GameData data = dataManager.Instance.gameData;
        attack = data.attack;
        moveSpeed = data.moveSpeed;
        maxHp = data.maxHp;
        //currentHp = data.maxHp;
    }

}
```

23. PlayerBullet

```
using UnityEngine;
```

```
public class PlayerBullet : MonoBehaviour
{
        [SerializeField] private float moveSpeed = 25f;
        [SerializeField] private float timeDestroy = 0.5f;
        [SerializeField] private float damege = 10f;
        [SerializeField] GameObject bloodPrefabs;
    protected Player player;


    private bool isBulletLevel2;

    void Start()
    {
        player = FindAnyObjectByType<Player>();
        isBulletLevel2 = CompareTag("BulletLevel2"); // Kiểm tra tag của đạn
        Destroy(gameObject, timeDestroy);
    }
    // Update is called once per frame
    void Update()
        {
            MoveBullet();
        }
        void MoveBullet()
        {
            transform.Translate(Vector2.right * moveSpeed * Time.deltaTime);
        }

    private void OnTriggerEnter2D(Collider2D collider)
    {
        if (collider.CompareTag("Enemy"))
        {
            DealDamage(collider, player.attack + damege);
        }
        else if (collider.CompareTag("EnemyExplosion"))
        {
            float finalDamage = isBulletLevel2 ? (player.attack / 2 + damege / 2) : (damege +
player.attack);
            DealDamage(collider, finalDamage);
        }
        else if (collider.CompareTag("EnemyEnergy"))
        {
            float finalDamage = isBulletLevel2 ? ((damege + player.attack) * 1.5f) : (damege +
player.attack);
            DealDamage(collider, finalDamage);
        }
    }
```

```
    private void DealDamage(Collider2D collider, float damageAmount)
    {
        Enemy enemy = collider.GetComponent<Enemy>();
        if (enemy != null)
        {
            enemy.takeDame(damageAmount);
            GameObject blood = Instantiate(bloodPrefabs, transform.position, Quaternion.identity);
            Destroy(blood, 1f);
        }
        Destroy(gameObject);
    }
}
```

24.PlayerCollision

```
using UnityEngine;

public class PlayerCollision : MonoBehaviour
{
        [SerializeField] private GameManager gameManager;

        [SerializeField] private AudioManager audioManager;

        private void OnTriggerEnter2D(Collider2D collision)
        {
                if (collision.CompareTag("EnemyBullet"))
                {
                        Player player = GetComponent<Player>();
                        player.takeDame(20f);
                }
                else if (collision.CompareTag("Energy"))
                {
                        //gameManager.addE();
                        GameObject[] allGuns = GameObject.FindGameObjectsWithTag("Gun");
                        foreach (var e in allGuns)
                        {
                                Gun gun = e.GetComponent<Gun>();
                                if (gun != null)
                                {
                                        gun.ActivateBuff(1);
                                }
                        }
                        Destroy(collision.gameObject);
                        audioManager.PlayEnerty();
                }else if (collision.CompareTag("HealHp"))
                {
        Player player = GetComponent<Player>();
        player.Heal(player.maxHp);
        Destroy(collision.gameObject);
```

```
}
else if (collision.CompareTag("Box"))
{
    Player player = GetComponent<Player>();

    int randomEffect = Random.Range(0, 3);

    if (randomEffect == 0)
    {
        player.Heal(player.maxHp);
    }
    else if(randomEffect == 1)
    {
        GameObject[] allGuns = GameObject.FindGameObjectsWithTag("Gun");
        foreach (var e in allGuns)
        {
            Gun gun = e.GetComponent<Gun>();
            if (gun != null)
            {
                gun.ActivateAmmoBuff();
            }
        }
    }
    else if (randomEffect == 2)
    {
        player.Heal(player.maxHp);
        GameObject[] allGuns = GameObject.FindGameObjectsWithTag("Gun");
        foreach (var e in allGuns)
        {
            Gun gun = e.GetComponent<Gun>();
            if (gun != null)
            {
                gun.ActivateAmmoBuff();
            }
        }
    }
    Destroy(collision.gameObject);
}

else if (collision.CompareTag("ItemBuff"))
{
    Player player = GetComponent<Player>();
    player.Heal(player.maxHp);
    GameObject[] allGuns = GameObject.FindGameObjectsWithTag("Gun");
    foreach (var e in allGuns)
    {
        Gun gun = e.GetComponent<Gun>();
        if (gun != null)
```

```
            {
                gun.ActivateAmmoBuff();
            }
        }
        Destroy(collision.gameObject);
    }else if (collision.CompareTag("coin"))
    {
        goldManager.Instance.AddGold(1);
        Destroy(collision.gameObject);
    }
  }
}
```

25. PropsRandom

```
using System.Collections.Generic;
using UnityEngine;

public class PropsRandom : MonoBehaviour
{
  public List<GameObject> propSpawnPoints;
  public List<GameObject> propPrefabs;


  // Start is called once before the first execution of Update after the MonoBehaviour is created
  void Start()
  {
    SpawnProps();
  }

  // Update is called once per frame
  void Update()
  {

  }

  void SpawnProps()
  {
    foreach (GameObject sp in propSpawnPoints)
    {
      int rand =  Random.Range(0, propPrefabs.Count);
      GameObject prop = Instantiate(propPrefabs[rand], sp.transform.position, Quaternion.identity);
      prop.transform.parent = sp.transform;
    }
  }
}
```

VI. Role of members

Roll Number	Full name	Roles
HE160896	Nguyễn Ngọc Ánh	Menu +UI
HE171567	Nguyễn Quang Huy	Animations + Skills (Character)
HE180210	Thái Duy Phong	Animations + Skills (Enemy)
HE180944	Nguyễn Xuân Hanh	Maps
HE186919	Trần Duy Long	Maps

VII. REFERENCE MATERIALS

[1]: https://www.youtube.com/watch?v=h9AAc-9LYq4&t=4222s

[2]: https://www.youtube.com/watch?v=8K_QWs1Hj5A&t=4377s

[3]: Source 3

[4]: Linh Canva: Group1-Battle Pow