**Denys Stepanenko**

**My game overall bugs pages:**

**Bug 1: Boss Damage Displays but Doesn't Apply**

**Problem Description:**

- When the boss attacks, damage is displayed (e.g., in logs or visually), but the player's health does not decrease.

**Code Before Fix:**

```csharp

// In the DemonBoss class

private void PerformAttack()

{

if (attackCooldownTimer > 0) return;

isAttacking = true;

anim.SetTrigger("Attack");

attackCooldownTimer = attackCooldown; // Setting attack cooldown timer

// Applying damage to the player

if (Vector2.Distance(transform.position, player.position) <= attackRadius)

{

player.TakeDamage(damage);

}

StartCoroutine(ResetAttackCooldown());

}

```

**Problem in the Code:**

- The `TakeDamage` method is called directly on the `player` object, which may not work if `player` does not have a `TakeDamage` method.

- It may be necessary to access the `PlayerHealth` component on the player object.

**Code After Fix:**

```csharp

// In the DemonBoss class

private void PerformAttack()

{

if (attackCooldownTimer > 0) return;

isAttacking = true;

anim.SetTrigger("Attack");

attackCooldownTimer = attackCooldown; // Setting attack cooldown timer

// Applying damage to the player

if (Vector2.Distance(transform.position, player.position) <= attackRadius)

{

PlayerHealth playerHealth = player.GetComponent<PlayerHealth>();

if (playerHealth != null)

{

playerHealth.TakeDamage(damage);

}

}

StartCoroutine(ResetAttackCooldown());

}

```

**Impact of the Fix:**

- The `TakeDamage` method is now called on the `PlayerHealth` component of the player, ensuring that damage is correctly applied to the player's health.

**Bug 2: Boss Jerks in Detection Radius**

**Problem Description:**

- When the player is within the boss's detection radius, the boss starts jerking and moving incorrectly.

- Possible cause: incorrect logic in state switching in the FSM or errors in the movement logic.

Code Before Fix:

```csharp

// In the DemonBoss class

private void Update()

{

if (isDead || player == null) return;

float distanceToPlayer = Vector2.Distance(transform.position, player.position);

// Updating state

if (distanceToPlayer > detectionRadius)

{

ChangeState(BossState.Idle);

}

else if (distanceToPlayer > attackRadius)

{

ChangeState(BossState.Walking);

MoveToPlayer();

}

else

{

ChangeState(BossState.Attacking);

if (!isAttacking) PerformAttack();

}

attackCooldownTimer -= Time.deltaTime;

FlipBoss();

}

```

**Problem in the Code:**

- Possible frequent state switching between `Idle` and `Walking`, causing jerking.

- Logic for state transitions may be incorrectly set, especially the conditions for transitioning between states.

**Code After Fix:**

```csharp

// In the DemonBoss class

private void Update()

{

if (isDead || player == null) return;

float distanceToPlayer = Vector2.Distance(transform.position, player.position);

// Updating state only if boss is not in Hurt or Dead state

if (currentState != BossState.Hurt && currentState != BossState.Dead)

{

if (distanceToPlayer > detectionRadius)

{

ChangeState(BossState.Idle);

}

else if (distanceToPlayer > attackRadius)

{

ChangeState(BossState.Walking);

}

else

{

ChangeState(BossState.Attacking);

}

}

// Performing actions based on current state

switch (currentState)

{

case BossState.Walking:

MoveToPlayer();

break;

case BossState.Attacking:

if (!isAttacking) PerformAttack();

break;

}

attackCooldownTimer -= Time.deltaTime;

FlipBoss();

}

---------------------------------------------------------------------------------------------------------------------------

**Impact of the Fix:**

- Added condition to prevent state changes when the boss is in Hurt or Dead state.

- This prevents frequent state switching and jerking.

- Action logic moved to `switch` statement, making the code more understandable and manageable.

------------------------------------------------------------------------------------------------------------

**Bug 3: Hurt Animation Doesn't Play**

**Problem Description:**

- When the boss takes damage, the hurt animation (`TakeHit`) does not play.

- Possible cause: animation trigger is not being called, or Animator settings are incorrect.

**Code Before Fix:**

```csharp

// In the BossHealth class

public void TakeDamage(int damage)

{

if (isInvincible || isDead) return;

currentHealth -= damage;

if (currentHealth <= 0)

{

Die();

}

else

{

StartCoroutine(TriggerInvincibility());

StartCoroutine(TakeHitCoroutine());

}

}

```

**Problem in the Code:**

- The `TakeHit` animation trigger is not called in the `TakeDamage` method.

- The animation may not play if the trigger is not activated.

**Code After Fix:**

// In the BossHealth class

public void TakeDamage(int damage)

{

if (isInvincible || isDead) return;

currentHealth -= damage;

animator.SetTrigger("TakeHit"); // Triggering hurt animation

if (currentHealth <= 0)

{

Die();

}

else

{

StartCoroutine(TriggerInvincibility());

StartCoroutine(TakeHitCoroutine());

}

}

```

**Impact of the Fix:**

- Added `animator.SetTrigger("TakeHit");` to trigger the hurt animation.

- Now, the boss will play the hurt animation when taking damage.

**Bug 4 Spike Bug:**

using UnityEngine;

using System.Collections;

public class SpikeTrap : MonoBehaviour

{

public PlayerHealth targetPlayer; // Игрок, которого атакуют шипы (назначается вручную!)

public int damage = 25; // Урон от шипов

public float attackInterval = 1.5f; // Как часто шипы наносят урон

private bool playerInZone = false; // Флаг: находится ли игрок в зоне атаки

private Coroutine attackCoroutine; // Храним ссылку на корутину

private void OnTriggerEnter(Collider other)

{

if (targetPlayer != null && other.gameObject == targetPlayer.gameObject)

{

playerInZone = true;

attackCoroutine = StartCoroutine(AttackPlayer());

Debug.Log("Игрок вошёл в зону атаки шипов!");

}

}

private void OnTriggerExit(Collider other)

{

if (targetPlayer != null && other.gameObject == targetPlayer.gameObject)

{

playerInZone = false;

if (attackCoroutine != null)

{

StopCoroutine(attackCoroutine);

}

Debug.Log("Игрок покинул зону атаки шипов.");

}

}

private IEnumerator AttackPlayer()

{

while (playerInZone)

{

if (targetPlayer != null)

{

// Получаем компонент PlayerHealth у игрока

PlayerHealth playerHealth = targetPlayer.GetComponent<PlayerHealth>();

// Проверяем, что компонент не null

if (playerHealth != null)

{

playerHealth.TakeDamage(damage); // Наносим урон игроку

Debug.Log("Игрок получил урон от шипов!");

}

else

{

Debug.LogWarning("Компонент PlayerHealth не найден у игрока!");

yield break;

}

}

else

{

Debug.LogWarning("Игрок не назначен в SpikeTrap!");

yield break;

}

yield return new WaitForSeconds(attackInterval);

}

}

}

**Levels bugs and how I solve them:**

## **Bug 1. Enemy Count Bug**

**Problem:** On each level, the enemy count (enemyCount) was inconsistent — sometimes it was 0 at the start, sometimes negative. This caused the game to skip levels early or behave unpredictably.

**Cause:** enemyCount was not initialized automatically based on the actual enemies in the scene. It relied on manual setup or enemy scripts to call EnemySpawned().

**Fix:** In GameManager.Start(), we added automatic scanning of enemies by tag:

csharp

GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");

enemyCount = enemies.Length;

**Comment:**

csharp

// Automatically counts all enemies in the scene at the start

## **Bug 2. Player Not Found After Scene Load**

**Problem:** After loading a new level or scene, the player reference in GameManager was missing, causing NullReferenceException during save/load.

**Cause:** Player was not set again after scene reload. Unity destroys objects on scene load unless marked as DontDestroyOnLoad.

**Fix:** In GameManager, we added an OnSceneLoaded hook to search for the player automatically:

csharp

private void OnEnable()

{

SceneManager.sceneLoaded += OnSceneLoaded;

}

private void OnDisable()

{

SceneManager.sceneLoaded -= OnSceneLoaded;

}

private void OnSceneLoaded(Scene scene, LoadSceneMode mode)

{

currentScene = scene.name;

player = FindObjectOfType<PlayerHealth>();

}

**Comment:**

csharp

// Ensures player reference is updated after each scene load

## **Bug 3. Instant Level Switch Skipped Animations**

**Problem:** When the last enemy was defeated, the level switched immediately — leaving no time for death animations, sound, or player reaction.

**Cause:** LevelManager.LoadNextLevel() was called instantly on enemyCount <= 0.

**Fix:** We added a coroutine delay:

csharp

private IEnumerator WaitBeforeNextLevel()

{

yield return new WaitForSeconds(1.5f);

LevelManager.Instance.LoadNextLevel();

}

Called inside:

csharp

if (enemyCount <= 0)

{

StartCoroutine(WaitBeforeNextLevel());

}

**Comment:**

csharp

// Adds delay before loading the next level to allow for animation/sound

## 

## **Bug 4. No Pause Menu / ESC Button**

**Problem:** There was no way to pause the game or exit to the main menu properly. Pressing ESC did nothing.

**Fix:** Created PauseMenu.cs that listens to ESC and activates pause UI:

csharp

if (Input.GetKeyDown(KeyCode.Escape))

{

TogglePause();

}

With support for:

* ResumeGame() — resumes time
* ExitToMenu() — saves game & returns to main menu

And Time.timeScale = 0 to pause game logic.

## **Bug 5. Missing Autosave Before Exiting**

**Problem:** Player progress was not saved when exiting to the main menu via ESC.

**Fix:** Added a call to GameManager.Instance.SaveGame() inside ExitToMenu() of the pause script:

csharp

public void ExitToMenu()

{

GameManager.Instance.SaveGame();

LevelManager.Instance.LoadMainMenu();

}

## **Bug 6: Enemies Reappear After Reloading the Level**

### **Problem Summary:**

After reloading a saved game, enemies that had already been defeated by the player would reappear in the scene. This behavior broke gameplay continuity and made the save system unreliable, especially on levels where defeating all enemies was required to progress (such as boss levels or full-clear zones).

### **Cause of the Problem:**

The core issue was that enemies did not have a persistent, unique identifier. The save system was only storing general level state, not specific instances of enemies. As a result, it had no way of determining which enemies had been defeated prior to saving.

### **Why the Fix Was Necessary:**

In any game where enemy defeat affects progression or player state, it is critical that enemy persistence is tracked across play sessions. Without that, the player’s effort is invalidated, and the game becomes inconsistent and potentially frustrating. This issue also made testing and level design difficult, as enemy states could not be reliably reproduced.

### **Solution Implemented:**

To address the problem, a persistent enemy tracking system was implemented based on unique IDs:

1. **EnemyID Component:** A new component called EnemyID was added to all enemy prefabs. This component generates and stores a unique identifier (GUID or incremental integer) at runtime or via the editor.
2. **GameManager Integration:** A list of defeated enemy IDs (defeatedEnemies) was added to the GameManager (or central save manager). When an enemy dies, its ID is added to this list.
3. **Save and Load Logic:**
   * On save: The list of defeated enemy IDs is serialized and written to the save file.
   * On load: Each enemy checks its ID against the saved list. If it is found, the enemy is either destroyed or deactivated during initialization, preventing it from reappearing.

### **Code Example (Simplified):**

// EnemyHealth.cs

void Die()

{

GameManager.Instance.defeatedEnemies.Add(GetComponent<EnemyID>().id);

Destroy(gameObject);

}

// On level load (GameManager or enemy initializer)

if (GameManager.Instance.defeatedEnemies.Contains(enemy.GetComponent<EnemyID>().id))

{

Destroy(enemy);

}

### **Result of the Fix:**

* Defeated enemies no longer reappear after reloading the game.
* Player progress is now preserved consistently.
* Future features such as checkpoints, dynamic enemy types, or partial saves are easier to implement.

## **Bug 7: Level Exit Door Opens Before All Enemies Are Defeated**

### **Problem Summary:**

In certain scenarios, the exit door to the next level would become active before all enemies had been eliminated. This allowed players to skip parts of the level or bypass intended challenge sequences.

### **Cause of the Problem:**

Originally, the condition to open the exit door relied on a specific event trigger, such as defeating a boss or activating an object. It did not account for the presence of regular enemies remaining in the scene. This resulted in premature access to the door.

### **Why the Fix Was Necessary:**

This issue directly impacted game design integrity. In a game structured around combat and exploration, requiring players to clear all enemies is a fundamental pacing and progression mechanic. Allowing players to leave without clearing a level removes challenge, undermines game balance, and contradicts player expectations.

### **Solution Implemented:**

A more reliable and dynamic enemy check was introduced using runtime scene scanning:

1. **Scene-wide Enemy Check:** A check using FindObjectsOfType<EnemyHealth>() was added to determine how many active enemies remain.
2. **Dynamic Door Activation:** The door remains inactive until the enemy count reaches zero. This check occurs either continuously in Update(), or is triggered after each enemy death.
3. **Optional Optimization:** In larger scenes, caching enemy references or using an event system would improve performance, but for early prototypes, a direct FindObjectsOfType call is acceptable.

### **Code Example (Simplified):**

void Update()

{

if (!doorOpened && FindObjectsOfType<EnemyHealth>().Length == 0)

{

door.SetActive(true);

doorOpened = true;

}

}

### **Result of the Fix:**

* The level exit now becomes available only after all enemies are defeated.
* Game pacing and challenge design are preserved.
* Level logic is simplified and centralized, making it easier to maintain in future iterations.

## **Bug 8: Boss doesn't die at 0 HP**

### **Symptom:**

The boss continued attacking and behaving normally **even after reaching 0 HP**. The health bar disappeared, but no death animation or state change occurred — effectively making the boss immortal.

### **Root Cause:**

In the BossHealth script, the check for death was missing a condition that ensures death logic runs **only once**. As a result:

* Die() was not called when currentHealth dropped to 0.
* The isDead flag was either not set or ignored.
* The boss could **still act, move, and attack** even with 0 HP.

### **Fix:**

The bug was resolved by adding a **guard clause** inside the damage handler:

if (!isDead && currentHealth <= 0)

{

Die();

isDead = true;

}

This ensures:

* Die() is called **only once**, right when HP drops to 0.
* All death-related behavior is triggered (animation, disabling AI, etc.).
* isDead prevents further damage processing or AI actions.

### **Outcome:**

* The boss now **dies properly** when HP reaches 0.
* All related systems (UI, AI, animations) react correctly.
* This fix also **prevents other potential edge cases**, like negative HP or redundant death calls.

### **Tested scenarios:**

* Boss takes damage normally → dies at 0 HP → ✅ works.
* Boss with multiple attacks or damage over time → dies correctly on time → ✅ works.
* Replaying level → boss dies again correctly → ✅ persistent fix.

**9. Bug: Platforms didn’t scale to level width** **Symptom:** Platforms were visually cut off or didn’t span the expected level width, especially on wide levels.  
 **Cause:** Platforms were manually scaled or made of a single prefab tile without dynamic adjustment. This broke visual consistency and caused gameplay issues.  
 **Fix:**

* Created a script that instantiates middle tiles dynamically based on the desired width.
* End tiles were placed separately or as part of the prefab.
* Logic used a loop to place tile prefabs along the X-axis relative to the parent object.  
   **Result:** Platforms now correctly span any required width, maintain modular design, and fit visually across the whole level.

**10. Bug: Skeletons glitched on slopes or terrain edges** **Symptom:** Skeleton enemies jittered, froze, or flipped erratically when walking over slopes or touching level edges.  
 **Cause:** Movement used Translate() or had imprecise raycasts, leading to detection failures on uneven ground. Flip logic was also too sensitive, causing rapid direction switching.  
 **Fix:**

* Switched movement to Rigidbody2D.velocity for physics consistency.
* Improved raycast checks using groundCheck and wallCheck with adjustable length and layers.
* Added a cooldown timer between flips to avoid flickering.  
   **Result:** Skeletons now move smoothly over slopes and corners, no longer get stuck or behave unpredictably.

**11.Bug: Enemies respawn after level reload** **Symptom:** Killed enemies reappeared after reloading the level, even though they were defeated in the previous session.  
 **Cause:** The game did not store the state of defeated enemies. Upon loading the scene, all enemies were instantiated again without checking whether they had been killed before.  
 **Fix:**

* Created a component EnemyID that assigns a unique ID to each enemy prefab.
* Stored all defeated enemy IDs in a list within the GameManager.
* On scene load, the system checks all enemies in the scene. If an enemy’s ID is in the defeated list, that enemy is destroyed before gameplay begins.  
   **Result:** Enemies no longer respawn after being killed in a previous session. The game world now reflects the correct enemy states, improving the consistency and reliability of saved progress.