UNITY DEVELOPER INTERNSHIP TRIAL - TASK 2 REVIEW

-

Task 2: Isometric Player Movement & Input System

Overview

This document outlines the process, decisions, and reflections on Task 2 of the Unity Developer Internship Trial at Runic Dices Entertainment. The objective of this task was to implement player movement using Unity's New Input System, ensure proper isometric movement behavior, integrate animations, and set up an appropriate camera system.

Timeline and Initial Steps

Day's Progress & Key Milestones

- Started the day by finalizing the Task 1 Dev Log and Workflow Diagrams in Lucidchart.
- Uploaded documentation to the project's Documents folder under:
  - Documents/Dev Logs/
  - Documents/Workflow Diagrams/
- Committed and pushed Task 1 under the commit: *"Task 1 Completed - Unity Project Setup & GitHub Integration."*
- Communicated with RDE staff: Sent an email to Radu C. Matusa to provide transparency about my GitHub repository and Task 1 progress.
- Attended university class (14:00 - 17:30).
- Began Task 2 Implementation.

Approach & Execution

Given the task's requirements, I focused on implementing a robust and scalable movement system while ensuring code clarity, reusability, and modularity.

1. ***Environment Setup:***

- Imported a CC4.0 Isometric Tileset**.**
- Manually sliced the tileset using Unity's Sprite Editor (auto-slicing did not work correctly).
- Created an Environment folder in Assets/ to properly structure environment assets.
- Set up an Isometric Tilemap (Z as Y) to match the isometric projection.
- Created a Tile Palette and painted the initial background with a grass layer**.**
- Added environment decorations to make the scene look better.

2. ***Player Setup & Input System:***

- **Created the Player GameObject**:
  - Parent Player GameObject (handles logic, physics, and state machine).
  - Child PlayerVisual (handles sprite rendering & animations).

- **Added essential components**:
  - Rigidbody2D (Dynamic, Gravity Scale = 0, Z Rotation Frozen, Interpolation Enabled).
  - CapsuleCollider2D (Adjusted to fit player shape).
  - EdgeCollider2D (Applied to Tilemap boundaries to prevent out-of-bounds movement).

3. ***Input System Implementation (Data-Driven Approach):***

- Implemented GameInputSO (Scriptable Object) to handle player input.
- Player can move using:
  - Keyboard (WASD / Arrow Keys).
  - Gamepad (New Input System bindings applied).
- Movement input normalizes vectors to ensure diagonal movement speed consistency.

4. *Player State Machine & Movement Logic:*

- Established a State Machine architecture:
    - State.cs & StateMachine.cs → Generic FSM framework.
    - PlayerStateMachine.cs → Core state logic for the player.
    - PlayerBaseState.cs → Base class for all player states.
    - PlayerIdleState.cs → Handles idle behavior.
    - PlayerMovementState.cs → Handles movement logic & transitions.
- Player Movement Implementation
    - Reads input via GameInputSO.
    - Converts movement input into isometric movement by rotating vectors x°.
    - Updates Rigidbody2D.linearVelocity accordingly.
    - Transitions to Idle State when no input is detected.

5. *Animation System Implementation:*

- Created PlayerAnimator.cs to handle all animation logic via code (no Unity Animator transitions).
- Created Player Animator Controller with:
    - Idle Animation.
    - 8-direction movement animations.
    - Blend Tree (2D Simple Directional) for movement.
- Handled animation updates in PlayerAnimator:
    - UpdateAnimation() updates speed, direction parameters.
    - PlayIdle() & PlayMovement() manage animation transitions.

6. *Research & Understanding Isometric Camera in 2D:*

- Question: *Does a 2D Isometric Camera exist?*
- Conclusion: Unlike in 3D, a 2D isometric setup does not require camera rotation. Instead:
    - Tilemaps and assets create the isometric look.
    - The camera remains orthographic with no rotation.
- Adjusted camera settings accordingly to fit the scene properly.

Challenges & Reflections

1. ***Understanding Isometric Camera Behavior***

- Issue: Initially tried to rotate the camera like in 3D (X = 30°, Y = 45°), but it distorted everything.
- Solution: Researched and realized that in 2D, the assets and tilemaps create the isometric effect, not the camera.
- Fix: Kept the camera orthographic (0,0,0) rotation and let the environment do the isometric projection.

2. ***Adjusting Isometric Angle for Proper Movement Alignment***

- Issue: The initial isometric movement felt off despite the code being correct.
- Solution: Discovered that the default 45° isometric angle did not align well with the environment.
- Fix: Experimented with different values and found that 20° provided the best alignment for movement consistency with the tilemap.

3. ***Animator Parameters Not Resetting on Idle***

- Issue: playerHorizontal & playerVertical were not resetting to 0 when stopping movement.
- Solution: Added a check in PlayerAnimator.UpdateAnimation()
- Ensured IdleState.Enter() explicitly resets parameters.

Time Spent & Estimations

- In my initial workflow estimation, I predicted the task would take between 3.5 and 4.5 hours.
- The actual total time spent was within this range but slightly longer due to:
    - Taking notes and conducting minor research.
    - Additional time spent on environment setup and adjustments.

- Despite the extra time spent, the task was completed within the estimated workflow timeframe.

Next Steps

- Move on to Task 3: AI State Machine for Enemy Behavior.
- Continue maintaining structured commit logs & documentation.

Final Thoughts

Task 2 provided a solid foundation for movement, input handling, and animation systems. The modular State Machine architecture, data-driven Input System, and code-controlled animations ensure the system remains scalable and maintainable.

❖ **Written by:** Juan F. Gutierrez
February 20th, 2025.