Alt(Alternative) Pokémon

By: Sayed Kibria

Goal

The goal of this project was to create a real-time Pokémon battle game where players can directly control their Pokémon, switch between team members, and experience dynamic combat. I wanted to build a system that’s modular and flexible, so adding new Pokémon, attacks, or UI features would be straightforward, while keeping the gameplay engaging and interactive

Motivation

I’ve liked playing Pokémon games, and I wanted to try something different by making battles happen in real time instead of turn-based. A game project also gives immediate feedback: I can implement a feature and see it come to life. It felt like a great way to challenge myself, learn Unity, and create something fun that I would want to play myself.

Development timeline

Week 1	Exploring different engines and what to make
Week 2	Built first background, player movement, and start working on adding collision
Week 3	Achieved working collision, added first summon creature, and switch system
Week 4	Created design diagram and plan for future implementation
Week 5	Making a demo
Week 6	Implemented melee and ranged attack, added health bar, damage feedback, rework code for attacking, and test everything up to date
Week 7	n/a
Week 8	Draft for project

System Requirements

Functional Requirements:

Player can move in 2D (WASD or arrow keys).

Player can switch between human and Pokémon forms.

Pokémon can attack, take damage, and apply knockback.

Health bars update dynamically.

Attacks vary by type (melee/ranged) and element.

Non-Functional Requirements:

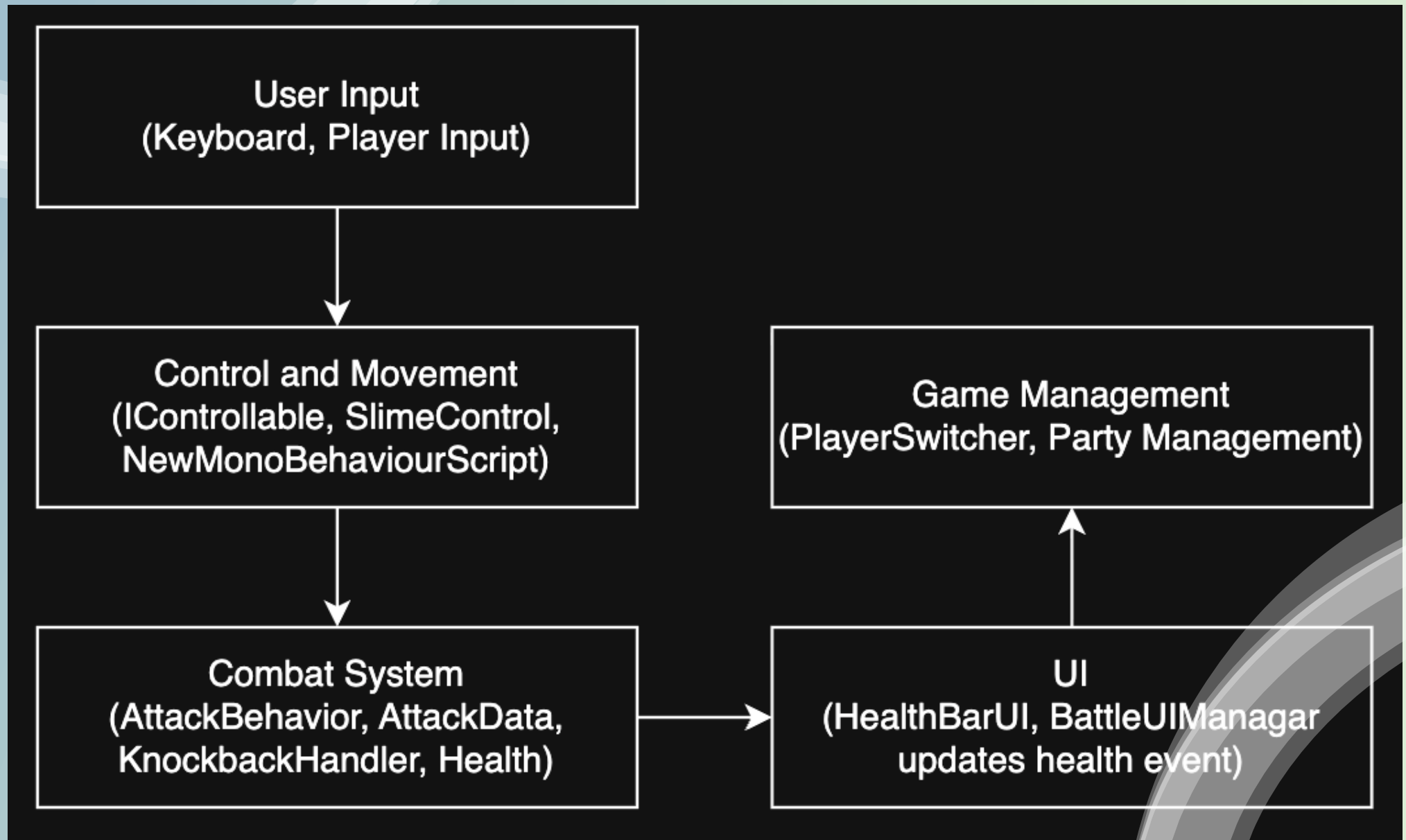
Smooth frame rate (≥60 FPS)

Modular system for adding new Pokémon/attacks

Scalable UI for multiple characters

Clear player feedback (animation, knockback, hit detection)

System Architecture Diagram



Information System Alternatives

Tool	Advantage	Disadvantage
Unity 2D ✓	Lots of Learning Resource. 2D tools: physics, animator, and tilemaps. Easy to integrate UI, sound, and animations in one workflow.	Requires some optimization for 2D since Unity's engine is originally built for 3D
Unreal Engine X	Excellent graphics and physics engine. Built-in blueprint system for quick prototyping.	Heavier engine Optimized for 3D rather than 2D.
Godot X	Open-Source. Simple 2D workflow and low system requirements.	Smaller community compared to Unity. Limited C# support and fewer prebuilt assets or tutorials.

Use Case

Use Case 1:

Player Switch Player presses number key (1–6)

PlayerSwitcher destroys old Pokémon and spawns new prefab

Control shifts from Human → Pokémon

UI updates to show Pokémon’s health

Use Case 2: Real-Time Battle

Player inputs move (WASD) → handled in SlimeControl

Player presses attack key → triggers AttackBehavior prefab

Attack collides → calls IDamageable.TakeDamage()

Health updates → invokes OnHealthChanged event

HealthBarUI visually updates the fill amount

IT Risks

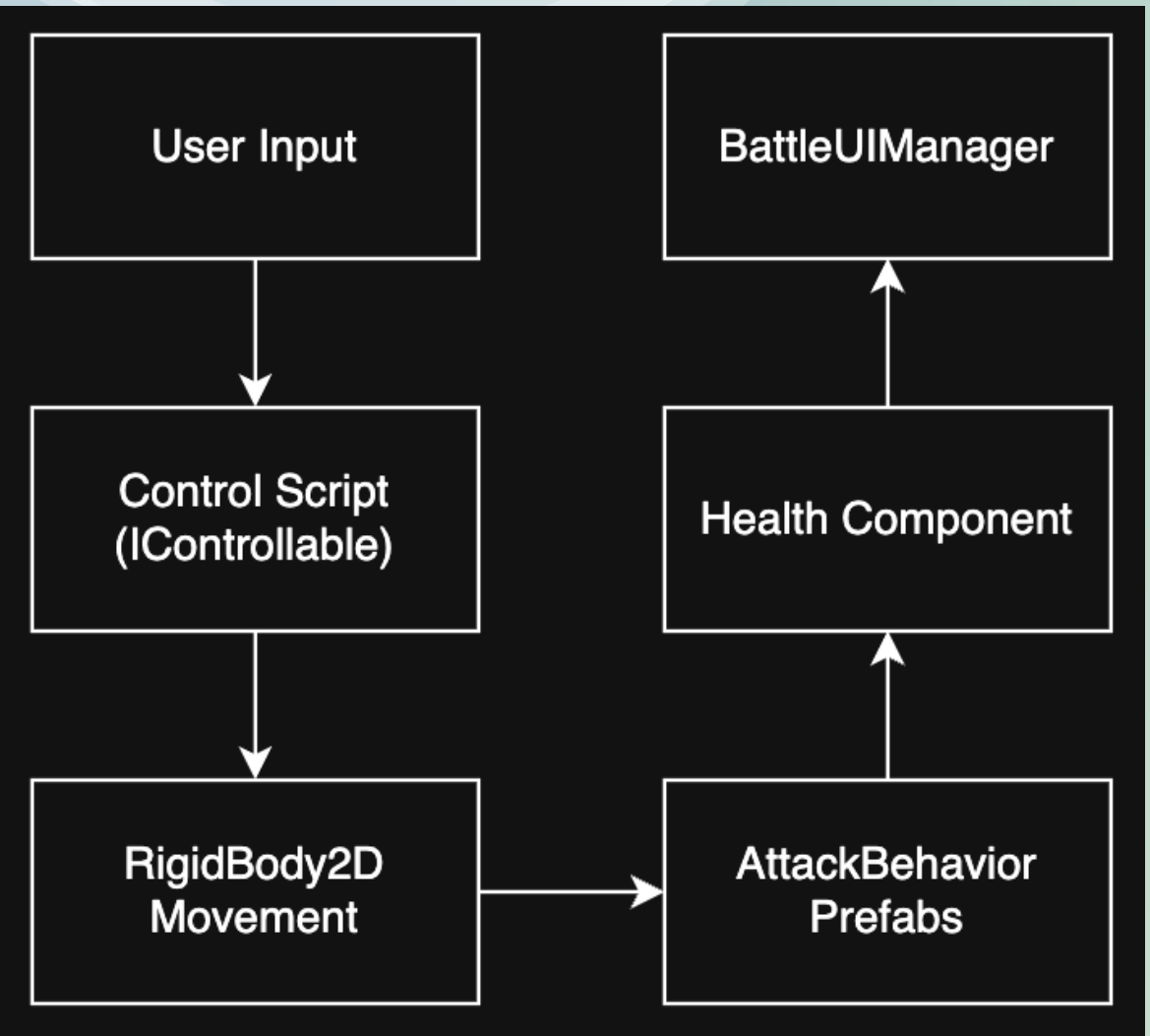
Performance bottlenecks: Multiple prefabs instantiating attacks.

Data loss: If ScriptableObjects aren’t linked properly.

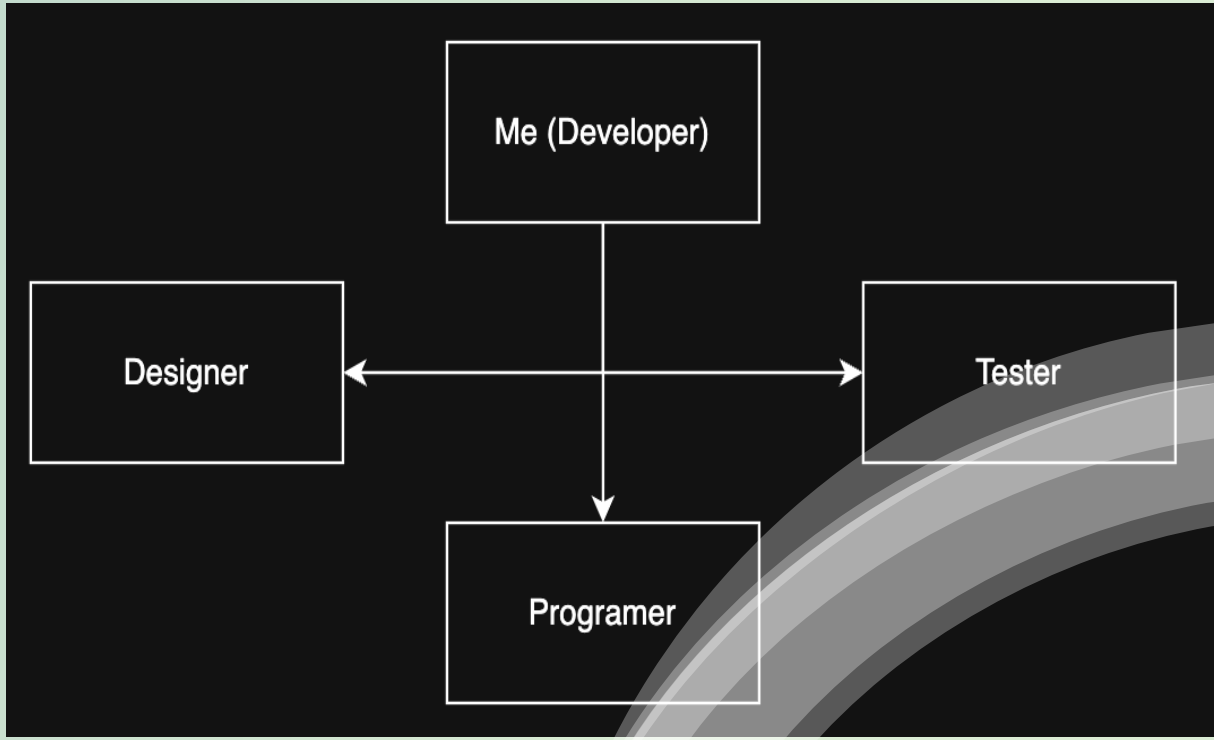
Input conflicts: Overlapping key bindings between human and Pokémon controls.

Null reference errors: Missing components (e.g., Rigidbody2D or Animator).

Data Flow Diagram



Organization Chart



Testing Methodology and System Evaluation

Unit Testing:

Health.TakeDamage() reduces HP correctly

KnockbackHandler.ApplyKnockback() applies correct force

Integration Testing:

Ensure AttackBehavior interacts properly with Health and KnockbackHandler.

Confirm PlayerSwitcher properly disables and enables controls.

Testing Methodology and System Evaluation

Playtesting scenario:

Performance Test:

Stress test by spawning multiple attacks simultaneously.

Test Case	Expected Result
Press 1 to summon Pokemon	Pokemon spawns, UI updates
Attack enemy	Enemy health decreases
Take damage	Knock back and health bar update
Press H	Control returns to human

Development Challenges

Lost progress: Unity crash → no auto-save → rebuilt from scratch

Lesson: Keep a backup copy of the progress made

Collision headaches: Character stuck or not colliding

Solution: Rewrote code, simplified assets, fixed Rigidbody settings

Breakthrough Moment

Added first creature and switching control between human and Pokémon

Coded attacks using modular AttackData

Implemented HealthBarUI and KnockbackHandler

Technical Challenges

Problem: In SlimeControl.cs Movement input and knockback forces overlapped resulted in no visual feedback.

Solution: added IsBeingKnockedBack to temporarily disable movement input.

```
// Prevent movement during knockback
var knockback = GetComponent<KnockbackHandler>();
if (knockback != null && knockback.IsBeingKnockedBack)
    return;
```

Modular Codes

Attack Creation

```
atkObj.GetComponent<AttackBehavior>().Initialize(attackData, direction, gameObject);
```

SlimeControl.cs

AttackData = ScriptableObject (data separation)

attacks are data-driven.

Health Update via Event

```
public UnityEvent<float> OnHealthChanged; // Sends normalized value (0–1)
OnHealthChanged?.Invoke((float)currentHealth / maxHealth);
```

Health.cs

UnityEvent for for observer-pattern UI updates.

UI Reaction

```
targetHealth.OnHealthChanged.AddListener(UpdateHealthBar);
```

HealthBarUI.cs

loose coupling between UI and backend logic.

Learning and Technical Skill Development

Learned new technologies:

- Unity 2D framework, Animator, Prefabs
- ScriptableObjects and event-driven UI updates

Research phase (first week) compared multiple engines — evaluated trade-offs

Self-learning and troubleshooting using documentation and experimentation.

Links

GitHub:

https://github.com/SayedKibria/CISC.-4900-Project

Project Board:

https://github.com/users/SayedKibria/projects/3

Time Log:

https://docs.google.com/spreadsheets/d/1soJuc1z6rqxbsAKXREDqb9Z19dRSlbDLC821jA02Mts/edit?gid=0#gid=0