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 its a great way to challenge myself, learn Unity, and create something fun that I would want to play myself.

System Requirements

Functional Requirements:

Player can move in 2D (WASD).

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:

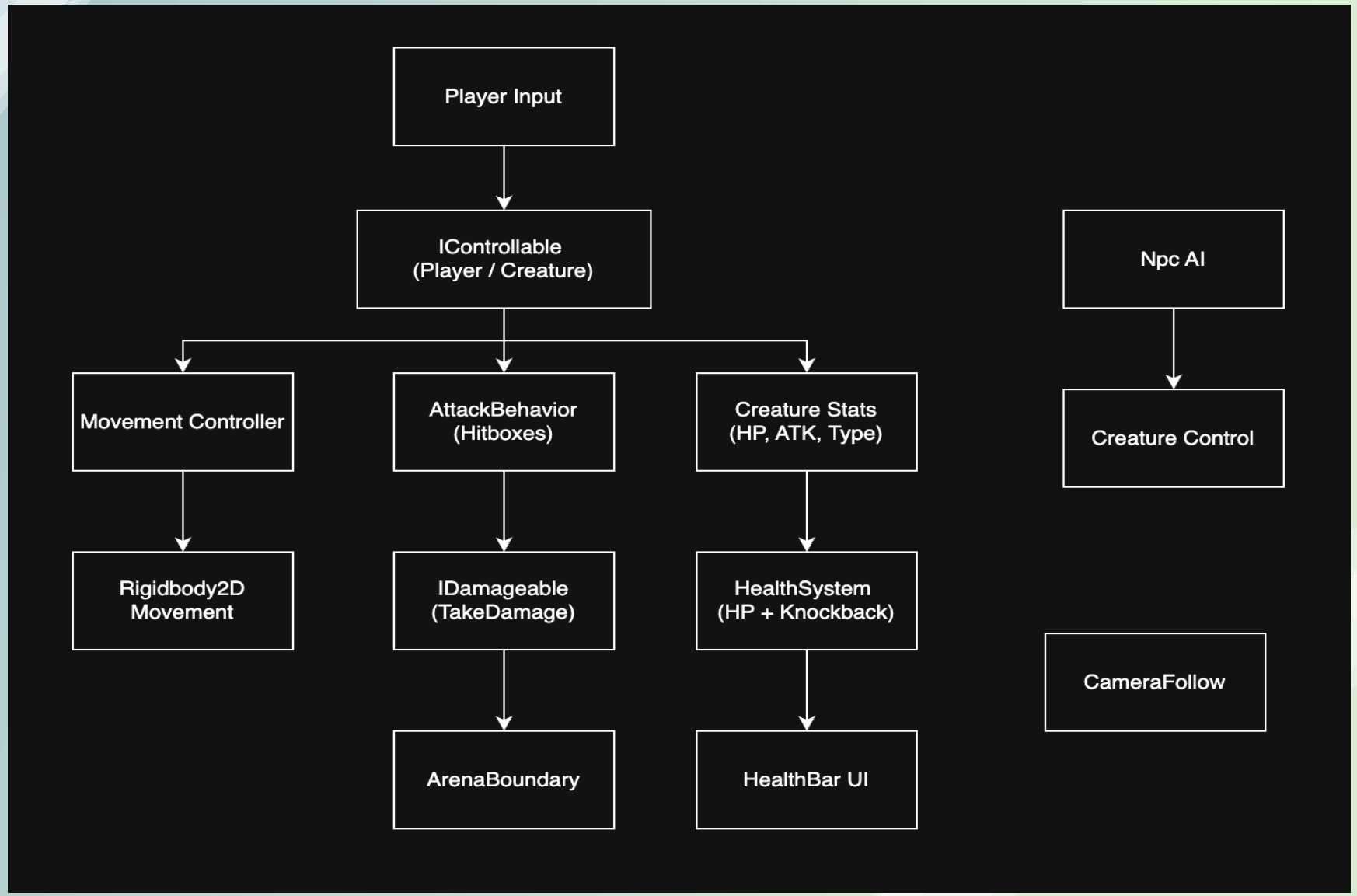
Smooth frame rate (≥60 FPS)

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.

CreatureControl

- Handles movement, attacks, and AI/player control for all creatures
- Supports both player input and AI-driven behavior
- Integrates health, knockback, and dash mechanics
- Maintains cooldowns for multiple attacks

```
void Update() { HandleMovement(); HandleAttacks(); }
```

CreatureData

- ScriptableObject storing stats, movement, attacks, and dash settings
- Provides configurable values like health, attack, speed, dash distance, and cooldowns
- Allows easy creation of multiple creatures with different abilities

public int maxHealth = 100;

public float moveSpeed = 3f;

public AttackInput[] attacks;

Defines the creature’s core stats and attack options

npcCreatureAI

- Controls AI creatures using dynamic, semi-random movement
- Prioritizes melee or ranged attacks based on expected damage
- Predicts player attacks to decide whether to attack or evade
- Can perform random dashes and follow random movement patterns

AttackBehavior

- Handles melee and ranged attack execution
- Spawns attack prefabs in the correct direction
- Applies damage, type effectiveness, STAB, and knockback
- Melee attacks can follow the owner until executed

```
collision.GetComponent<IDamageable>()?.TakeDamage(finalDamage);
knockback.ApplyKnockback(knockbackDir);
```

Applies calculated damage and knockback on collision

ElementType & TypeChart

- Manages element types (Normal, Fire, Water, Grass)
- Calculates type effectiveness between attacks and targets
- Supports STAB and elemental strategy

```
float typeMultiplier = TypeChart.GetEffectiveness(data.elementType, targetStats.elementType);
```

Determines damage modifier based on elemental types

DashComponent

- Implements dash mechanics with distance, duration, and cooldown
- Provides temporary invincibility frames during dash
- Supports AI-triggered and player-triggered dashes
- Can visually flash during invincibility

```
float dashVelocity = data.dashDistance / data.dashDuration;
rb.linearVelocity = dashDirection * dashVelocity;
isInvincible = (elapsed >= data.dashInvincStart && elapsed <= data.dashInvincEnd);
```

Moves creature while invincible during dash frames

BattleTrigger

- Triggers battle setup when the player enters a collider
- Positions player and NPC in battle positions, disables player control
- Moves camera to focus on battle area
- Automatically triggers summons for both player and NPC Pokémon

var playerControl = player.GetComponent<IControllable>();playerControl.DisableControl();

Locks player movement during battle setup

HealthBarUI & BattleUIManager

- HealthBarUI updates fill image based on normalized health
- Registers to Health.OnHealthChanged for dynamic UI updates
- BattleUIManager singleton connects player and enemy health bars to Health instances
- Supports multiple creatures (player and NPC) simultaneously

```
targetHealth.OnHealthChanged.AddListener(UpdateHealthBar);
fillImage.fillAmount = normalizedValue;
BattleUIManager.Instance.SetPlayerTarget(health);
```

NPC Summoner

- Spawns NPC Pokémon in front of the NPC
- Handles defaultCreatureIndex, summonDelay, and automatic next-Pokémon summoning
- Marks fainted Pokémon as dead to prevent resummon
- Connects spawned Pokémon to Health and UI via BattleUIManager
- Uses coroutines for delayed summoning

```
private IEnumerator SummonAfterDelay(){
yield return new WaitForSeconds(summonDelay);
SummonCreature(defaultCreatureIndex);}
```

Spawns NPC Pokémon after a delay
Handles death and auto-summon for the next Pokémon

PlayerSwitcher

- Manages player-controlled Pokémon and the human player
- Auto-summons first Pokémon if none are out
- Manual switching via number keys (1–6) with cooldown
- Prevents switching to dead Pokémon
- Handles health initialization, UI registration, and death events

if (Input.GetKeyDown(KeyCode.Alpha1 + i) && !deadCreatures[i])

StartCoroutine(SwitchCooldownRoutine(i));

Manual Pokémon switching with cooldownPrevents switching to dead Pokémon

Interfaces & Control

- IControllable allows enabling/disabling player or Pokémon control
- Used during switching, auto-summon, and knockback
- Decouples input logic from health, summoning, and AI systemsI
- Damageable standardizes damage across ranged, melee, or environmental hazards

```
public interface IControllable { void EnableControl(); void DisableControl(); }

public interface IDamageable { void TakeDamage(int amount); }
```

System Architecture

- Player & NPC Pokémon managed through PokemonParty, PlayerSwitcher, and NPCSummoner
- Combat → Health → HealthBarUI → BattleUIManager
- Arena constraints enforced by ArenaBoundary
- Camera dynamically follows player or locks to battle using CameraFollow
- Modular, event-driven design allows adding new Pokémon, abilities, or attack types without rewriting core logic

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 CreatureControl

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

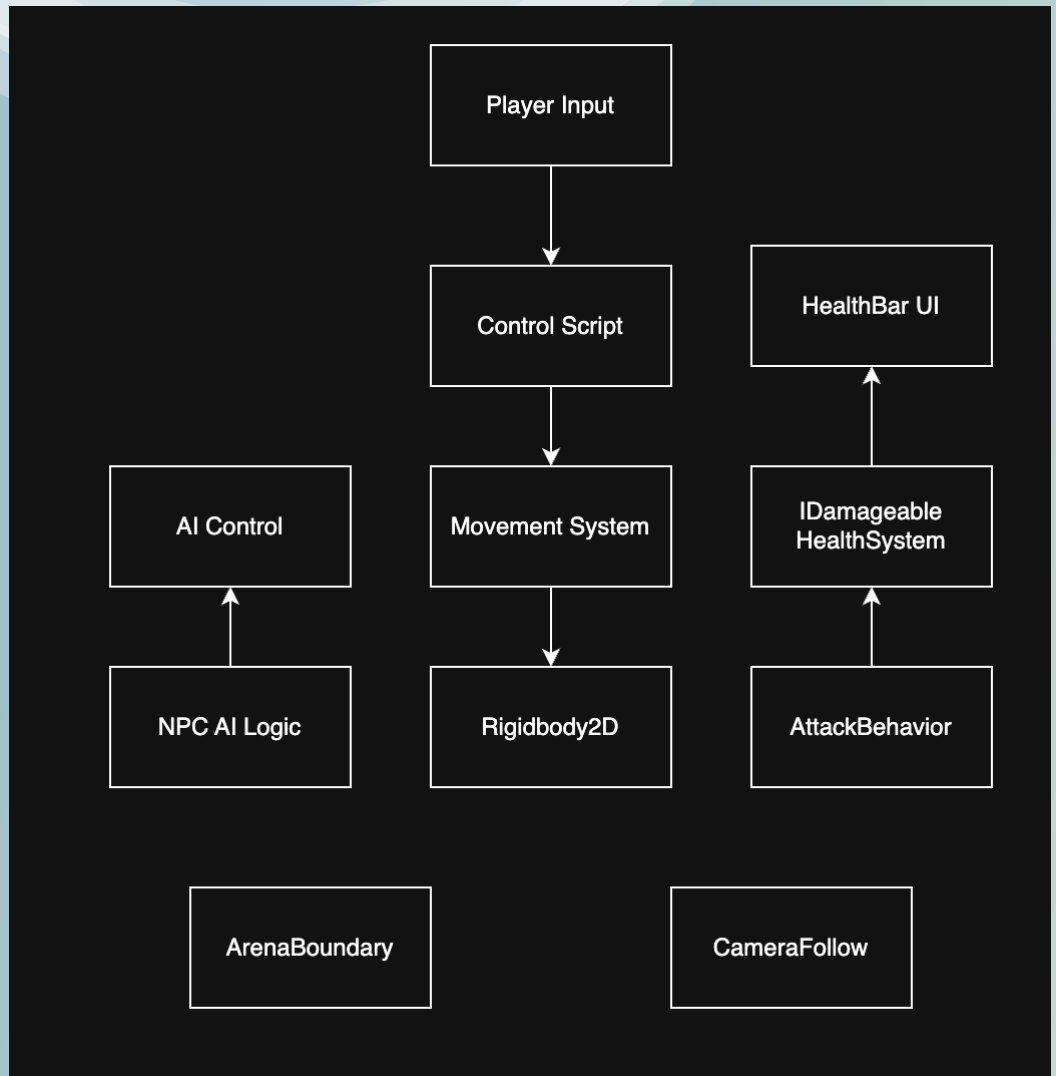
Performance bottlenecks: Multiple prefabs instantiating attacks.

Data loss: If ScriptableObjects aren’t linked properly.

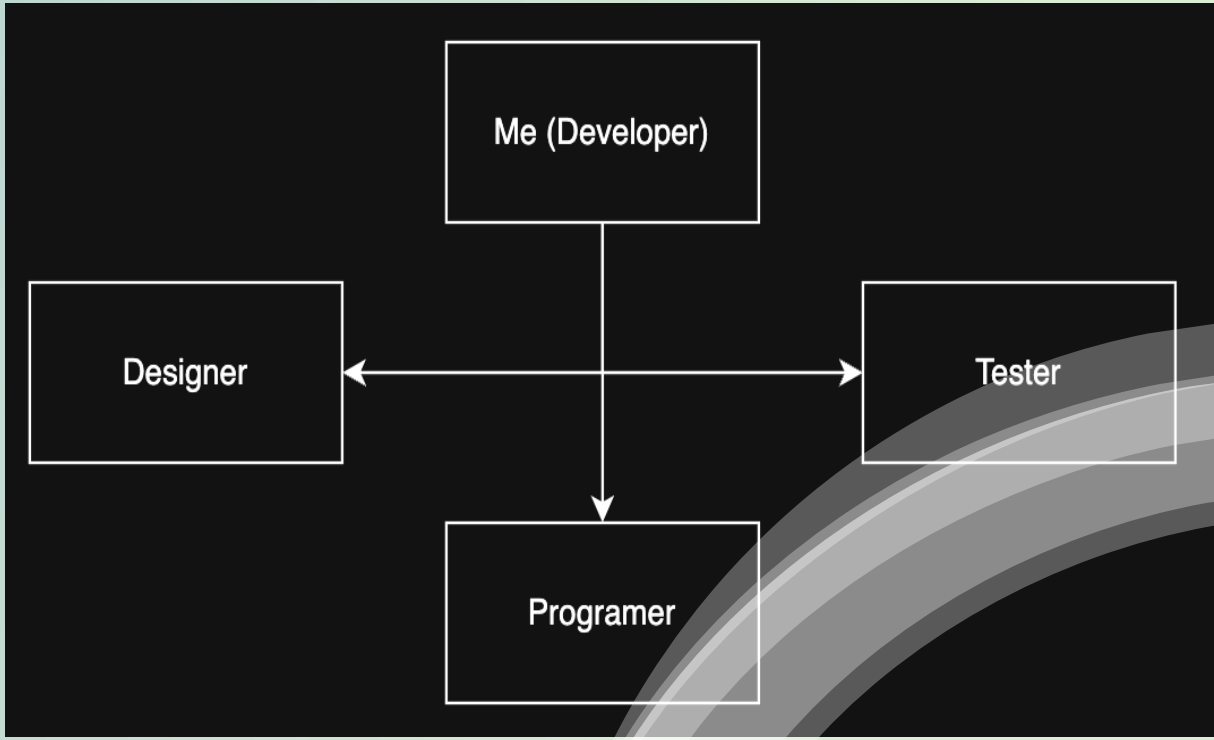
Input conflicts: Overlapping key bindings between human and Pokémon controls.

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

Abandoned Features

- Start Menu
- Party Management UI
- Game Over State
- Polished OverWorld
- More Creatures

The system crashed, implementing feature that wasn’t planed in the beginning, some features that were planed took longer to implement than expected, and not taking into account the time required to do other things ex. presentations

Learning and Technical Skill Development

Learned new technologies:

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

Self-learning and troubleshooting using documentation and experimentation.

Unity does not auto save, backup are essential.

Timelog Summary

Week 1:

- Researching different engines and what type of game to make

Week 2:

- Build background, player movement, and working on adding collision

Week 3:

- Achieved working collision
- Added one creature, ability to summon the creature, switching control from player to creature, camera following player, a party system for future full implementation.

Week 4:

- Creating design diagram, updating GitHub, planning for future implementation

Week 5:

- Making a video demo

Timelog Summary

Week 6:

- Implement melee and ranged attack, improve code for creature attack, health bar, damage feedback (knock back), test using dummy

Week 7:

- Nothing was done

Week 8:

- Project draft

Week 9:

- Nothing was done

Week 10:

- Making an arena for battle, transition to arena when encountering npc, rework code to make player only be able to summon creature after battle triggers, working on making camera change view

Timelog Summary

Week 11:

- Working on making the camera change from following the player to be centered when battle triggers
- Restructured some codes
- Added a new creature, party system for npc, ai for npc creature

Week 12:

- Trying to change players position after battle ends
- Implemented creature death, storing the current health of creature for when switching between creatures
- Added different stats(attack, defense, special attack, special defense), different types, damage formula
- Implemented dash with cooldown, i-frame(invincibility)
- Cooldown between creature swap

Timelog Summary

Week 13:

- Boundary for the arena so attacks and creature wont just leave, making the melee attack follow the player in front
- Trying to make the player change position from arena after a battle ends for game over (unsuccessful)
- Auto summon players first creature, npc swap creature, after current creature die auto summon next creature
- Start improving npc creature ai

Week 14:

- Improved npc creature ai
- Improving appearance of the overworld
- Improving the resume for an unexpected job opportunity
- Working on presentation.

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