Beautiful Corridor: Battle System

By Brian Baron and Gabe Pereyra

The Battle System

The Battle System is the container class that will be used for the main game logic. This includes allowing for turns, as well as checking if the end condition has been met. The end condition is satisfied when either of the Pokemon reach 0 health, making the other the winner. The user has the ability to swap between the Pokemon on their team instead of using a move during their turn.

Pokemon Base Class

Within the class there are 2 Pokemon pointers for the Pokemon active in the battle. The Pokemon class is a base class that contains functions and values essential to all Pokemon to be implemented: Health, Attack, Defense, Special Attack, Special Defense, and Speed. There is a member that tracks the current health, which will trigger the end state upon reaching 0. Each Pokemon has a name, a list of 4 Moves available to them, and are assigned 2 Types.

During damage calculation, if the attacking Pokemon's Type matches the move, it gains a Same Type Attack Bonus (STAB), which multiplies the regular output by 1.5. The rest of the calculation is determined by the attacking Pokemon's offensive stats, the power of the Move, the defending Pokemon's defensive stats, and the Type weakness/resistance coefficient (which is calculated by a table containing the Type weaknesses and resistances). The output is also multiplied by a random value from .85 to 1, to allow for varying outputs within a rather controlled average.

Types

There are 16 possible Types implemented, and an individual Pokemon can have 1 or 2 types (known as MultiTyping). Each type corresponds with a table of weaknesses and resistances, stored in a 2D array. This table is used to get the coefficient for weaknesses used in the damage calculation, by checking the indexes [attacking move type][defending pokemon type].

Move Struct

Each move contains a Type, Power/strength, Accuracy, PP (Power Points/usage amount), and a name to correspond with it. Moves are separated into those dealing damage, healing moves, and stat changing moves. Each stat can be effectively changed by a coefficient, changing the output of the damage. This is to be used with moves used to boost/decrease stats of a target.

Translating to Unreal

Battle System Container Class

For this project, we had a battle system manager that handled turn order, keeping track of both the active Pokemon and the Pokemon in each player's roster, taking in user input, calling appropriate damage dealing or switching functions, and end states. This class encapsulates the logic we need for the full fledged battle system in our future port to Unreal. Ideally, we'd have a C++ class that takes keeps track of the Unreal objects and goes through the same logic found in our regular C++ solution. A potential base class for this game logic would be the Unreal GameMode class, which is meant for managing and dictating the rules of the game - perfect for the battle logic.

Pokemon Base Class

The Pokemon base class will be adjusted to work with the Unreal actor class. We will be adding functionalities such as a static mesh to contain the Pokemon model and converting any data types (such as string) to the Unreal appropriate versions. Also, as of now, our Pokemon are premade subclasses of the base Pokemon class. By the end of the project, we would like to make Pokemon creation data driven, but for the purpose of prototyping our current method was sufficient.

Move Struct

The Move struct will be merged with visual feedback, allowing for representation of attacks and damage inflicted. Other than that, the move struct simply holds valuable move data, so there shouldn't be too many changes to this struct's functionalities.

Types

Types will keep the same functionality as in a plain C++ project, because they are used solely in data, and not in the visual representation of the game.

Other Things to Mention

Our battle logic is sound as of now. If anything, the last part we'd like to add while translating the battle system to Unreal is the implementation of status effects, though for now it is not a priority. The rest of the translation is tied to everyone else's projects. VR UI, input, and art pipeline are being handled by other team members, and should be mostly ready to integrate together.