User Documentation

Pokémon Battle Simulator is a CLI tool that allows users to define their own Moves, Pokémon, and Pokémon Teams, and then have them battle against each other to determine which one is better.

The application is navigated by typing commands when prompted. Entering invalid commands will prompt again.

Main Menu

After starting the simulator, you will be presented with the main menu. Here, you can type `build` to go to the build menu, `battle` to go to the battle menu, or `exit` to exit the simulator.

Build Menu

In the build menu, you can create and manage your Pokémon, Moves, and Teams.

Valid commands in the build menu:

- `back` - Go back to the main menu.
- `listM`, `listP`, `listT` - List all created Moves, Pokémon, or Teams respectively.
- `newM`, `newP`, `newT` - Create a new Move, Pokémon, or Team respectively.
- `defaults` - Load default Moves, Pokémon, and Teams. Keep in mind that this will overwrite any existing Moves, Pokémon, or Teams you have created.
- `save` - Save all created Moves, Pokémon, and Teams to a `user.json` file.
- `load` - Load Moves, Pokémon, and Teams from a `user.json` file.

When listing Moves, Pokémon or Teams, you will be presented with a paginated list. You can navigate it using `n` (next page), `p` (previous page), or `q` (back to the build menu). You can also type `delete (number)` to delete an item from the list.

When creating a new Move, Pokémon, or Team, you will be prompted to enter the required information. For Moves, this includes name, type, category, power, accuracy, and PP. For Pokémon, this includes name, first and second type, level, stats (HP, Attack, Defense, Special Attack, Special Defense, Speed), and moves. For Teams, you will need to provide a name and list pick Pokémon. Enter the information either directly or select from a paginated list. You may at any point type `abort` to cancel the creation process.

Saving and loading will store all data in a `user.json` stored in the same directory as the simulator executable. This file will be created if it does not exist and will be overwritten if it does.

Battle Menu

In the battle menu, you can start battles between your Pokemon and Teams.

Valid commands in the battle menu:

- `back` - Go back to the main menu.
- `battle`, `battleT` - Start a battle between two Pokémon or Teams respectively.
- `battleMany`, `battleTMany` - Start multiple battles between Pokémon or Teams respectively.

Starting a battle will prompt you to select the two Pokémon or teams you want to battle. You will also be able to select the Pokémon and Team strategies. After that a whole battle will be simulated and all actions taken will be printed out. After the battle, you will be taken back to the battle menu.

When starting multiple battles, you will be prompted to select the Pokémon or Teams you want to battle. You can also select the number of battles to run. After all battles are completed, a summary of the results will be displayed. This way you can quickly determine which Pokémon or Team is better over many battles.

Developer Documentation

Application files are separated into several directories, based on their purpose. The entrypoint of the application is the Program class, which initializes the user and starts the main menu.

There is also `AssemblyInfo.cs` file, which enables tests to access internal members of the application. Tests are then located in the `PokemonBattleSimulator.Tests` directory.

User Interface

User navigation in the application is achieved using three controllers: MainController, BuildController, and BattleController. Each controller handles user input and manages the flow of the application.

Each menu (and other components that write to the console) print with a starting prefix. This is achieved using the PrefixedConsole class, which provides methods for writing to the console with a specific prefix. This allows for consistent and easy formatting across the application. (there is also IPrefixedConsole for potential mocking).

Data Management

User is able to define their own Moves, Pokémon, and Teams. All of them are stored inside Lists in the User class. User is then passed between controllers and in arguments to get and store data.

Moves, Pokémon and Teams each have their own classes: Move, Pokemon, and PokemonTeam respectively. Each of them store their specific stats and properties (e.g. Pokemon stores array of Moves).

The created Moves, Pokémon and Teams change only in the build phase. That is why each of them has a battle wrapper which tracks their state when in battle. These are BattleMove, BattlePokemon, and BattlePokemonTeam respectively. These are especially useful if we want to run multiple battles simultaneously, as they allow us to keep track of the state of each Pokémon and Team during the battle (with no conflict).

User is also able to save and load their data to/from a `user.json` file. This is achieved using the Data Persistence class, which handles serialization and deserialization of the user data. It also utilizes the File Wrapper class and IFileWrapper interface which provides an abstraction over file operations, allowing for easier testing and mocking.

Pokemon and Move Types

Pokemon types (such as Fire, Water, etc.) and Move categories (such as Physical, Special, etc.) are defined in the PokemonType and MoveCategory enums respectively.

The application also contains a TypeCalculator class, which provides methods for calculating move effectiveness in battles base on type. It utilizes a type chart that it initializes in the class constructor.

There is also an extra move defined in the FallbackMove class. This move is used when all of the Pokémon's moves are unavailable (e.g. due to PP being 0). (it is the equivalent of Struggle in the Pokémon games).

Strategies

When starting a battle, user can select strategies for Pokémon and Teams. These strategies are defined as methods in the AIStrategies and AITeamStrategies classes. These methods correspond to their respective delegates: AIStrategy and AITeamStrategy. Strategy for Pokémon picks a Move to use next, while Team strategy picks a Pokémon to use next in the battle.

When prompting user for strategies, the StrategyManager class is used. It uses reflection to find all methods that match the strategy delegates and then provides them to the user as options. This allows for easy addition of new strategies without changing the code.

Build Phase

User navigation in the build phase is handled by the BuildController class. It then utilizes the Build Manager class that provides methods for creating Moves, Pokémon, and Teams or loading default data.

Battle Phase

User navigation in the build phase is handled by the BattleController class. It then utilizes the Battle Manager to allow user to select Pokémon or Team with strategies.

After user makes their selection, the Battle class is used to run the battle. It provides methods for simulating (single or many) battles between two Pokémon or Teams. It does also utilize the Damage Calculator class which stores logic for calculating damage dealt by moves based on Pokémon stats and types.

Multiple battles are run simultaneously in parallel and then atomically increment the win count for each Pokémon or Team.

Turn and battle results are then indicated using the TurnResult and BattleResult enums.

Helper Classes

There are also some other helper classes:

- PaginatedLists provides methods for displaying paginated lists to user.
- Prompts provides methods for prompting user for input.
- Parsers provides methods returning parsers.