Hackmon Deliverable #1

Alexandra Girard, Ben Langlois, Ariel Lam

Design Discussion

The first major section of our design is the game logic. The game starts with team selection. Each player will be able to enter their name and assemble a team of 6 Hackmon, either from the list of pre-existing species or by creating a species of their own, with arbitrary characteristics. Each species has four stats which determine its effectiveness in battle: HP, Attack, Defense, and Speed. The players will then agree on whether they would like to play a 1v1 or 2v2 battle. We will support variable battle size and party size to facilitate potential future changes to our design.

After the selection process, the battle will begin. Players will take turns attacking each other's Hackmon and using items to strengthen their team. We plan to implement three types of moves which affect a different property of the Hackmon. The first type is stat-changing moves, which raises the stats of one of the Hackmon on your team. The second type is debuff-inflicting moves, which cause an enemy Hackmon to suffer a debuff. A debuff lowers the stats of a Hackmon, but it can be cured by using an item. The final type is mobility-changing moves, which can prevent an enemy Hackmon from using a move on future turns. There will also be three types of items: potions, debuff-healing, and mobility-healing. Potions raise the stat of a Hackmon (similar to a stat-changing move). Debuff- and mobility-healing items remove the debuff and mobility effects from a Hackmon. If a Hackmon's HP drops to 0, then it faints and is out of the battle. You win when all of your opponent's Hackmon faint. After the battle ends, players will be able to play again by using the same team or by re-selecting their Hackmon.

The second part of our design is the object-oriented structure. This is outlined in the UML diagram. Each section is outlined in more detail below.

Players

The gameplay is between two different players, each of which are class instances, allowing for the potential for more players competing at a time. The `Player` class keeps track of small details like name and win total, but also contain a vector of the player's Hackmon.

Hackmon & Species

Each Hackmon belongs to a specific species, but there can be multiple Hackmon of the same species. To create this relationship we have different `Species` objects for each species. Each species object has different stats and a unique move which can only be used by members of that species. It also has a `createHackmon` method for constructing a `Hackmon` of that species which inherit their stats and moves. This allows for multiple Hackmon of the same species, allows us to easily create new species, and also allows the user to create new species during program execution.

Since players will have a vector of `Hackmon`, in the `Hackmon` class we keep track of if the Hackmon is currently in battle, and we also keep track of the Hackmon's mobility and any debuffs against it.

<u>Actions</u>

There are two types of actions: moves and items. Moves and items override the `doAction` method in their derived classes. Since moves and items inherit from the action class, it is possible to store the player's action in a vector, regardless of what they decided to do on their turn. If we decided to allow the user to do something other than make a move or use an item on their turn, it would inherit from the `Action` class.

<u>Moves</u>

All moves in the game have a family, which determines how effective it will be against its target. For example, water moves are strong against fire Hackmon but weak against grass Hackmon. This is determined by the `effectiveness` method in the `Family` class. All moves also have an accuracy, which determines how likely the move is to hit the opponent. The effectiveness of a move is affected by the Attack stat of the attacking Hackmon and the Defense stat of the target Hackmon. The speed of all of the Hackmon determines the order of attacks for the round.

We also plan to implement three types of moves, as described in the game logic discussion above. Each of the three subclasses of `Move` override the `doAction` method. The first type of move is stat-changing moves. The `doAction` method lowers the specified stat of the target Hackmon by a predetermined value. Debuff-inflicting moves and mobility-changing moves both set the Hackmon's debuff and mobility fields respectively in their `doAction` method. Debuffs and mobility effects are explained in more detail later in this document.

We designed moves using inheritance to easily implement more types of moves. If we wanted to do this, we would simply have to create a new class which inherits from the abstract `Move` class and overrides the `doAction` method.

In battle, if a player chooses to use a move, there are a few different things that might occur. If a move has only one target but there are multiple possible targets, then the player will have to select which enemy they wish to attack. Other moves will target all enemy Hackmon currently in battle. The number of targets for a move is determined by its scope, which is stored as an enum in the move object. A problem arises if the opposing player wishes to swap out one or more of their Hackmon which is the target of a move. If this happens, the move should target the Hackmon which was swapped in for the previous target. Because of this, we decided to store the target(s) of a move as a vector of indices. That way, a move can target an arbitrary number of enemy Hackmon and could easily support a larger battle size (instead of only 1v1 or 2v2).

<u>Items</u>

The design of the `Item` class is very similar to the design of the `Move` class. There are three types of items: Potions, which raises the stats of one or more friendly Hackmon; Mobility-healing items, which makes the target(s) fully mobile, and; Debuff-healing items, which cures the target(s) of a debuff. Each subclass overrides the `doAction` method from the `Action` base class. Again, this makes it easier to add new types of items in the future.

Mobility

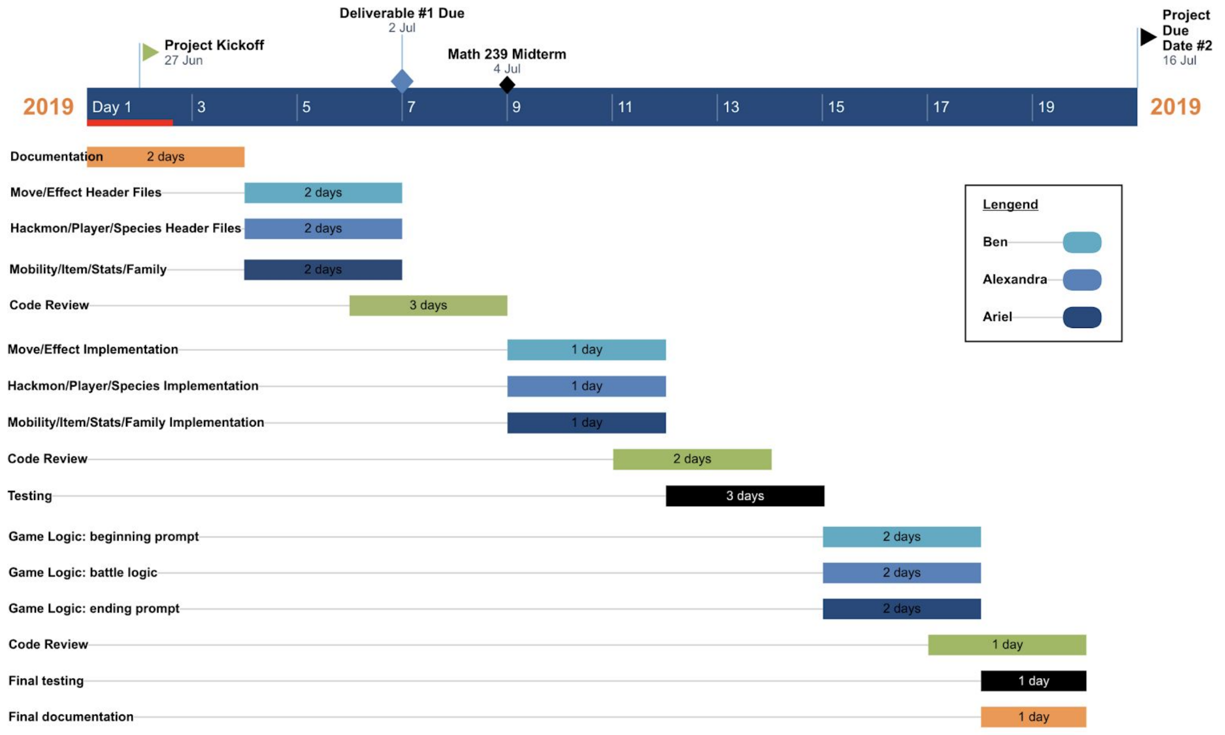
When a Hackmon is affected by a mobility-changing move, we store the effects of this move in the Hackmon in an object called `Mobility`. By default, the Hackmon's Mobility is Mobile, which means the Hackmon can always move. Mobile's `canMove` function will always return true, while the others will programmatically determine if the Hackmon can move or not.

Debuff

Debuff-inflicting moves are handled by storing a debuff tuple, which has an enum to the Debuff, the stat it affects, and how potent the Debuff is. Rather than creating a class (like `Mobility`), we chose to use tuples. While the Debuffs can vary in strength and Stat type, the function to apply the changes is the same, unlike the abstract `canMove` method of Mobility.

Project Timeline/Plan of Attack

Shown below is a timeline view of our plan of implementation for the project. The documentation (shown in orange), code review (green), and the testing (black) will be completed by all of the group members, while the shades of blue represent the classes and areas each member will be responsible for implementing.


Deliverable #1 Due
2 Jul
Project Kickoff
27 Jun
Math 239 Midterm
4 Jul
Project Due Date #2
16 Jul
Day 1
Documentation
2 days
Move/Effect Header Files
2 days
Hackmon/Player/Species Header Files
2 days
Mobility/Item/Stats/Family
2 days
Code Review
3 days
Move/Effect Implementation
1 day
Hackmon/Player/Species Implementation
1 day
Mobility/Item/Stats/Family Implementation
1 day
Code Review
2 days
Testing
3 days
Game Logic: beginning prompt
2 days
Game Logic: battle logic
2 days
Game Logic: ending prompt
2 days
Code Review
1 day
Final testing
1 day
Final documentation
1 day
Lengend
Ben
Alexandra
Ariel