Pokemon Battle Simulator

Project 6 Summary

Thomas Starnes and Brian Glassman

Status Summary

The bulk of this project is implementing all of the classes for the moves. We found about 40 different types of moves that would work well as individual classes. The list of implemented classes and those that are not complete yet can be found in the sections below. We have set up a functioning SQL database with all necessary information, including the Pokemon, all of the moves, and the learnsets of every Pokemon. We have completed the basic flow of the battle: select move, determine which Pokemon attacks first (determined by the speed, move priority, and status effect), use the moves and apply the effects, and apply any end of turn damage from burn or poison. The GUI for showing the currently battling Pokemon and their health, as well as displaying and selecting a move to use has been completed.

Breakdown of work across team members

Much of the work has been simply implementing the different classes, which has been shared between the team members. Aside from that, Thomas has been the primary architect while Brian has been developing the GUI.

Changes or Issues Encountered

So far no unexpected issues have been encountered. TKinter blocks program execution while the GUI loop is running so data has to be passed in and out using publisher-subscriber or MVC rather than function returns, but this was planned for in the original design. It was expected that the variety of different moves and status effects would have to be dealt with, and we have not found a solution that doesn't have at least a bit of code smell to it.

Patterns

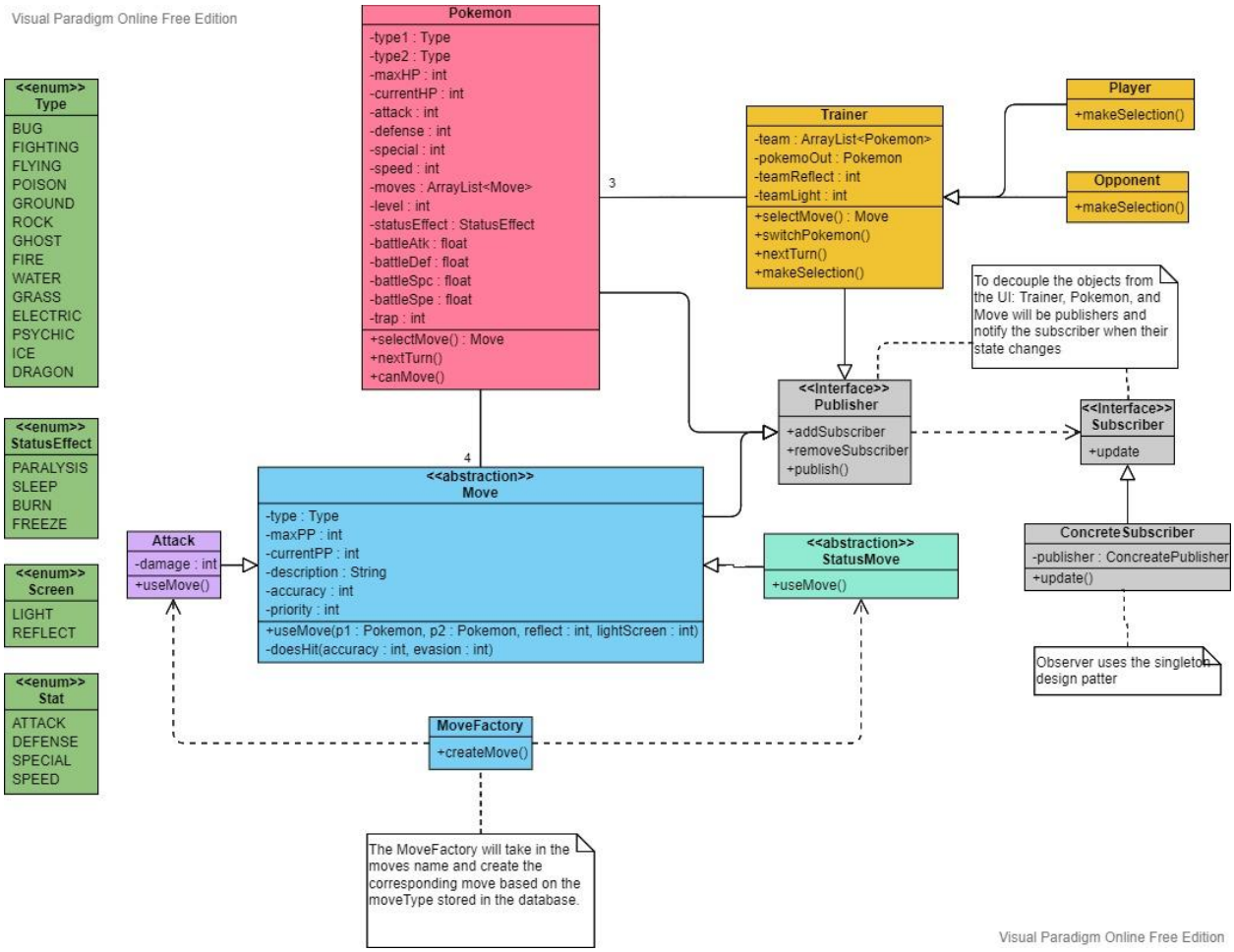
We are using the factory method pattern to create Move objects. When the user selects a move, the name of the move is passed into the factory method. The database is then queried to get the move_type (an integer value associated with the class the move should be assigned to). The factory method then creates an object based on the class and performs additional queries on relevant tables if needed.

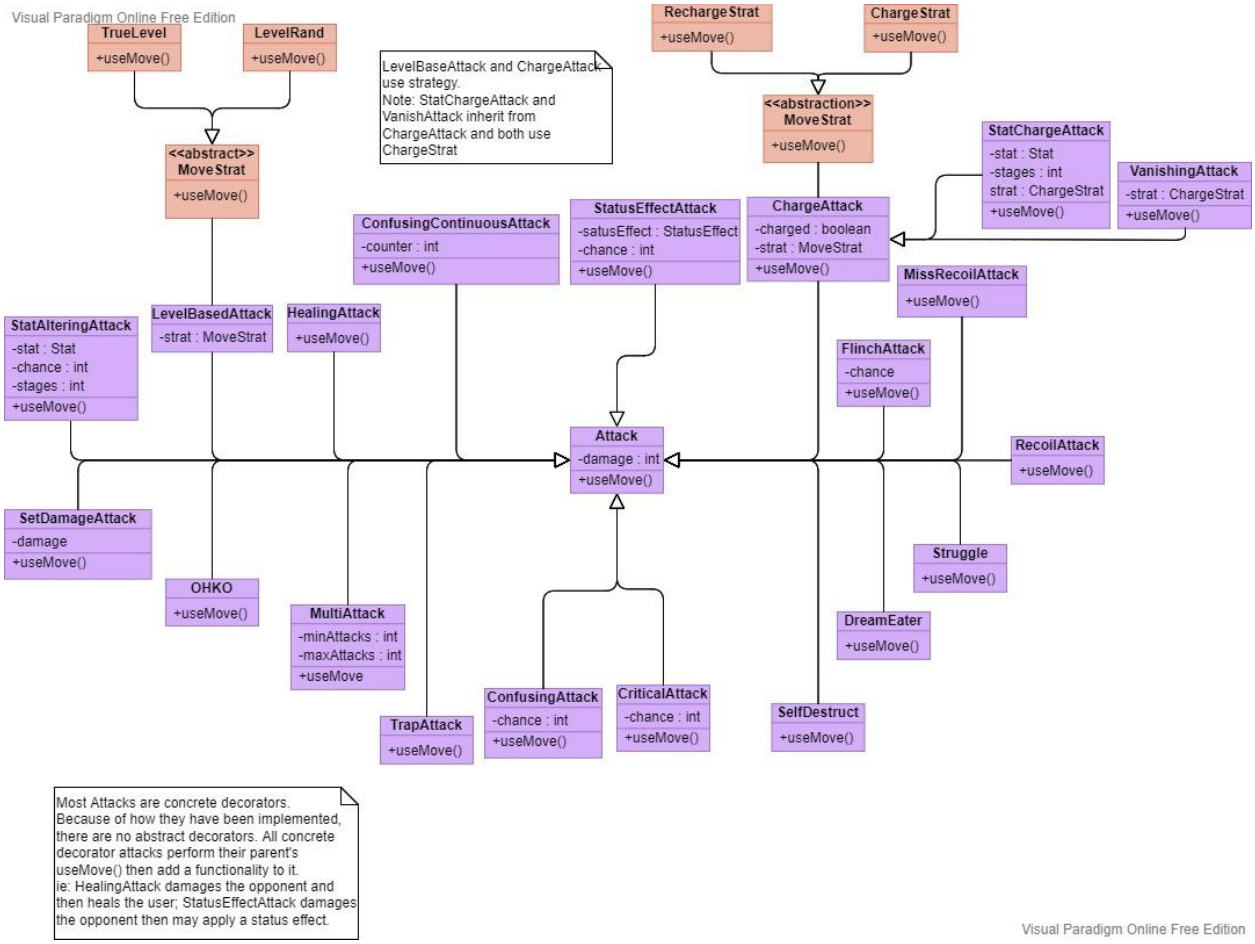
Two classes using the strategy pattern: LevelBassedAttack and ChargingAttack. The two variants of LevelBasedAttack are moves that do damage equal to the user's level and one move that does damage equal to a random value from 1 to 1.5 times the user's level, so we decided to make it a strategy. As for ChargingAttack, there are two variants: charge and recharge, which act the same but reverse.

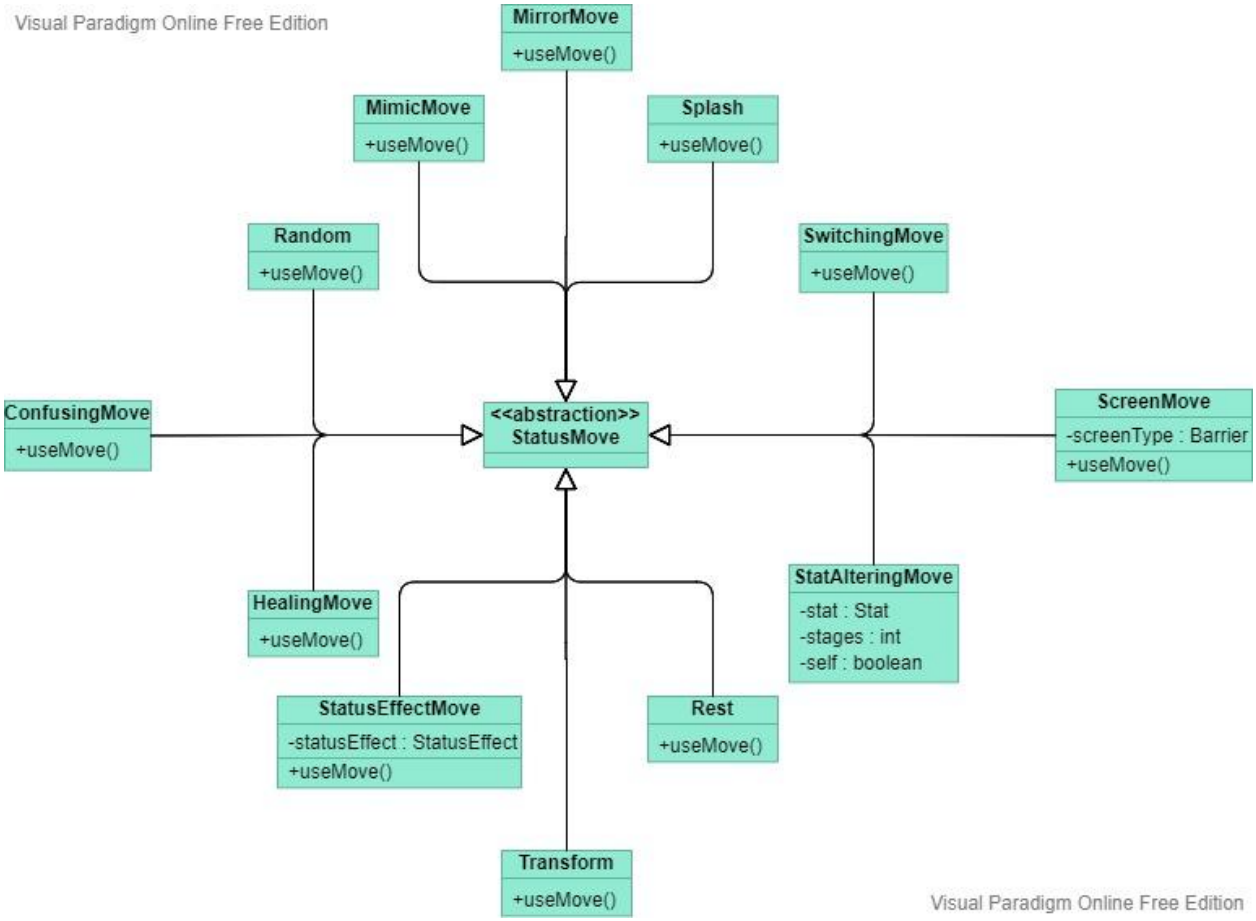
We are currently using the publisher-subscriber version of the observer pattern to send messages from the Trainer, Pokemon, and Moves to the subscriber to display. If issues arise, we will likely switch to MVC.
We are also using the decorator pattern as noted in the Attack Class Diagram.

Class Diagram

A few changes have occurred to the diagrams: Trainer and Pokemon have new methods; SetDamageAttack is now a stand alone class without use of strategy; OHKO is a new class created from the separation of SetDamageAttack from the last class diagram as well as LevelBasedAttack which uses strategy; ChargeAttack uses strategy, and any class that inherits directly from it uses ChargeStrat and not RechargeStrat.
The classes that have been implemented are: Trainer, Pokemon, Publisher, Subscriber, Move, Attack, StatAlteringAttack, StatusEffectAttack, SetDamageAttack, OHKO, FlinchAttack, RecoilAttack, MissRecoilAttack,HealingAttack, ChargingAttack, MultiAttack,ConfusingAttack,CritAttack, DreamEater, SelfDestruct, ScreenMove, StatAlteringMove, StatusEffectMove, HealingMove, ConfusingMove, Mimic, RandomMove, Rest, Splash.




Visual Paradigm Online Free Edition
TrueLevel
+useMove()
LevelRand
+useMove()
LevelBaseAttack and ChargeAttack use strategy.
Note: StatChargeAttack and VanishAttack inherit from ChargeAttack and both use ChargeStrat
RechargeStrat
+useMove()
ChargeStrat
+useMove()
<<abstraction>>
MoveStrat
+useMove()
<<abstract>>
MoveStrat
+useMove()
StatChargeAttack
-stat : Stat
-stages : int
strat : ChargeStrat
+useMove()
VanishingAttack
-strat : ChargeStrat
+useMove()
ConfusingContinuousAttack
-counter : int
+useMove()
StatusEffectAttack
-satusEffect : StatusEffect
-chance : int
+useMove()
ChargeAttack
-charged : boolean
-strat : MoveStrat
+useMove()
MissRecoilAttack
+useMove()
StatAlteringAttack
-stat : Stat
-chance : int
-stages : int
+useMove()
LevelBasedAttack
-strat : MoveStrat
HealingAttack
+useMove()
FlinchAttack
-chance
+useMove()
Attack
-damage : int
+useMove()
RecoilAttack
+useMove()
SetDamageAttack
-damage
+useMove()
OHKO
+useMove()
MultiAttack
-minAttacks : int
-maxAttacks : int
+useMove
Struggle
+useMove()
DreamEater
+useMove()
ConfusingAttack
-chance : int
+useMove()
CriticalAttack
-chance : int
+useMove()
TrapAttack
+useMove()
SelfDestruct
+useMove()
Most Attacks are concrete decorators. Because of how they have been implemented, there are no abstract decorators. All concrete decorator attacks perform their parent's useMove() then add a functionality to it.
ie: HealingAttack damages the opponent and then heals the user; StatusEffectAttack damages the opponent then may apply a status effect.
Visual Paradigm Online Free Edition



Planning Ahead

Our next steps are to complete the GUI and start work on the functionality to switch out pokemon in battle. We also need to finish up the remaining moves. The majority are implemented, but we still need to complete Transform, ScreenMove, LevelBasedAttack, ChargingAttack, ConfusingContinuousAttack, and Struggle. We also need to implement a better “AI” for the player to battle against. This will allow the “AI” to select moves based on how much damage it will do, if the attack affects the opposing Pokemon, the status effect applied, etc.