Pokemon Battle - EP1

Bruno Sesso	8536002
César Yapunari Nontol Rodríguez	9137902
Gustavo Estrela de Matos	8536051
Rafael Mendonça Miller	7581818

25 de outubro de 2014

INTRODUÇÃO

Em nossa primeira reunião, nos focamos em dividir como o trabalho seria feito e que métodos utilizaríamos. Dessa forma, construímos o primeiro diagrama de classes, tendo em mente de que deveríamos modularizar o programa de forma que seja expansível e que mantenha os conceitos de programação orientada a objetos. Dividimos as tarefas que cada um deveria fazer e cada um fez sua parte individualmente.

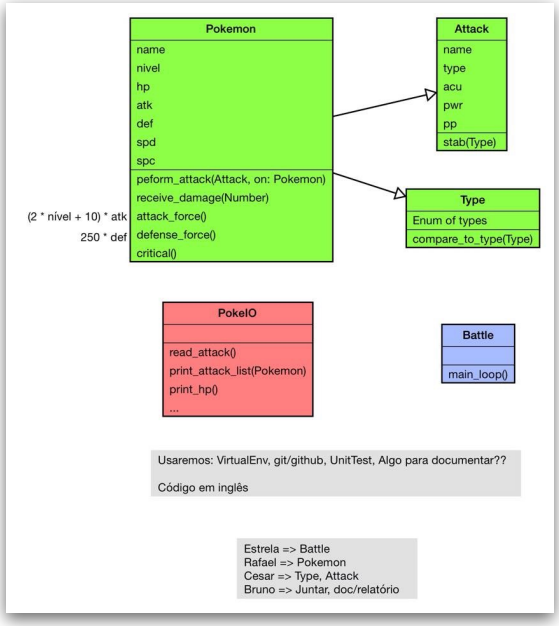
Estrutura

A princípio dividimos o programa em 5 classes principais com métodos e atributos não muito bem definidos:

- Pokemon
- Move
- Type
- Battle
- PokeIO

Sendo que a divisão entre cada pessoa era fazer:

1. Pokemon
2. Battle e PokeIO
3. Move e Type
4. Estruturar o programa como um todo e juntar o código



Ferramentas

Evidentemente, o código é feito em python3.4. Além disso optamos por utilizar o *git* em conjunto com o *github* para versionar e compartilhar o código durante o desenvolvimento. Para gerar a documentação utilizamos o programa *Sphinx*. Os testes seriam feitos utilizando o módulo *unittest*.

DESENVOLVIMENTO

Problemas

A princípio nos complicamos bastante devido ao encapsulamento. Como a *Damage Formula* envolve atributos de dois *pokemons* diferentes, não havia opção em que não teríamos que acessar os atributos de um objeto. Isso se aliviou quando entendemos que podemos acessar os atributos desde que isso seja feito através de *getters* e *setters*.

Achamos também que os *inits* das classes iriam ficar muito grandes. Portanto, criamos outra classe *Stats* que gerencia os *stats* de um *pokemon*. Além disso tentamos utilizar alguns valores padrões para os argumentos dos *inits*, para que dessa forma se tornassem opcionais.

A princípio não pensamos em como lidaríamos com erros e o polimorfismo dos métodos. Isso se mostrou mais próximo a entrega do projeto e não implementamos nenhuma solução especial para esse problema. Ao longo do código usamos muitos *try* que são parecidos entre si.

O maior problema dessa fase foi o primeiro diagrama de classes, pois não foi pensado em detalhes. Logo ao decorrer da produção do código, passamos a precisar de novos métodos ou atributos de outras classes. Isso acabou motivando pessoas não responsáveis por uma classe a mexerem nela. Isso foi ruim, pois quebra a modularidade, e a ideia de usar um código sem saber sua implementação. Se o projeto envolvesse mais classes e fosse maior, talvez tivesse ficado mais bagunçado, no entanto, como eram poucas classes, é fácil para uma pessoa olhar todas.

Soluções

O uso de *getters* e *setters* torna-se muito cansativo, o que nos fez procurar por alternativas. Achamos o uso de *@property* adequado, ainda que pareça estranho acessar um atributo de uma forma que parece que quebra o encapsulamento. Nos esforçamos para fazer o uso de *@property*, o menos possível. Ainda, usamos essa ferramenta para acessar *properties* de atributos da classe: *self._attr.prop*, o que sinceramente, não sabemos se foi abuso ou não.

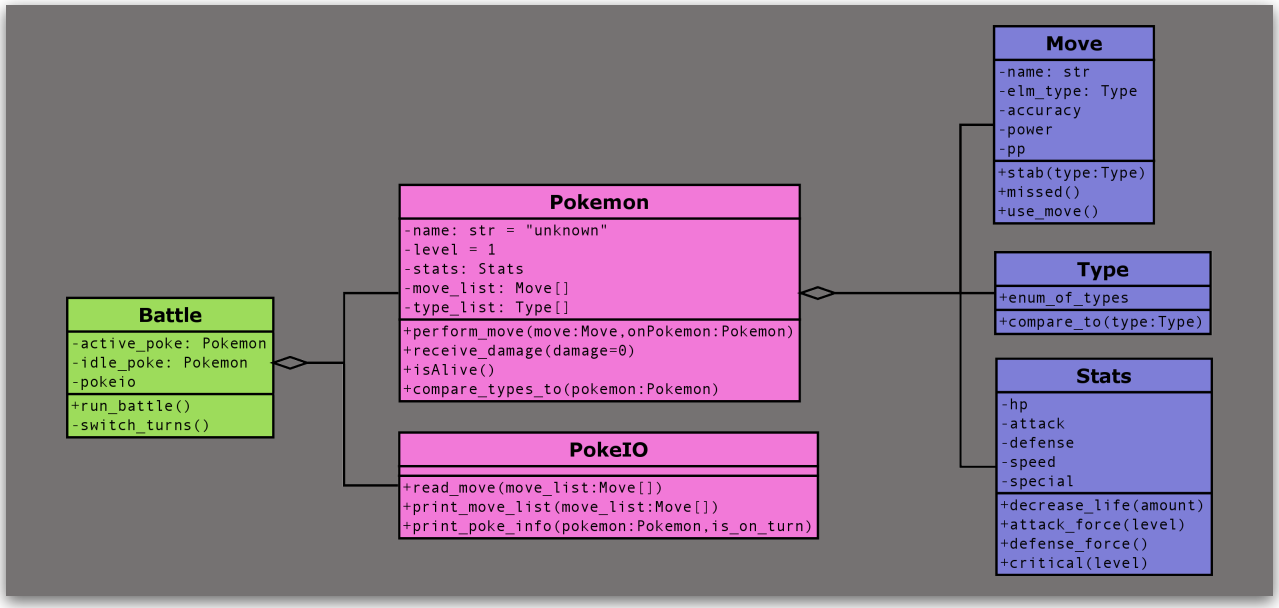
A ideia que usamos ao fazer os *inits* de cada classe é de que ele deve retornar uma instancia usável. Por exemplo: na classe Pokemon *name* e *level* são atributos que não são totalmente influenciáveis na funcionalidade da classe e que podem ter um valor padrão razoável. No entanto, não conseguimos inferir um *type_list, stats e move_list* que sejam padrões para qualquer *pokemon*. Com essa mentalidade desenvolvemos as outras classes.

Com os inúmeros *try* semelhantes, pensamos em fazer uma classe ou um conjunto de funções que realizassem esses testes de forma menos cansativa no entanto, não foi implementado nenhuma solução.

Para solucionar nosso maior problemas, concluímos que a preparação antes deveria ter sido maior. Ao fazer a primeira reunião e construir o primeiro diagrama de classes deveríamos ter pensado mais a fundo no que cada classe e método precisava. Também achamos interessante desenvolver os testes antes, pois de certa forma, eles representam como nós esperamos que as classes se comportam. Ainda com esses problemas, consideramos que aplicamos bem as ideias que aprendemos durante as aulas até agora. Solucionar problemas que apareceram durante a produção do código foi sempre simples e rápido. E mesmo mudar o código foi simples, por exemplo: durante a produção desse relatório, notamos que deveríamos mudar o atributo *move_list* no *init* da classe Pokemon (a principal classe). Ainda assim, após mudar o construtor dessa classe, notamos que só usamos esse construtor em uma parte do programa. Isso foi graças ao trabalho de pensar em modulariza-lo e pensar previamente em expandi-lo.

A última coisa que nos resta solucionar nas próximas fases é fazer testes durante a produção do código, evitando solucionar muitos erros de uma vez ao final do projeto.

DIAGRAMA DE CLASSES FINAL



DOCUMENTAÇÃO

A documentação do código foi feita usando o programa *sphinx* e pode ser encontrada no endereço:

http://www.ime.usp.br/~bsesso/PokemonBattle