

Relatório: Etapa 1

André Meneghelli Vale - 4898948

andredalton@gmail.com

Taís Pinheiro - 7580421

tais.aparecida.pinheiro@usp.br

22 de Outubro

1 Introdução

O projeto da disciplina *Laboratório de Programação 2* consiste no desenvolvimento de um sistema de batalhas entre dois jogadores de pokemom baseado no jogo homônimo.

Para tanto foi escolhida a linguagem Python versão 3 e o desenvolvimento orientado a objetos.

Foram determinadas inicialmente três etapas:

- **Etapla 1:** Modelagem da batalha em modo texto;
- **Etapla 2:** Comunicação ponto a ponto via rede pra batalhas;
- **Etapla 3:** Inteligência artificial para as batalhas;

Este relatório se foca apenas na *etapa 1*.

Todas as informações do jogo foram obtidas através de pesquisas na internet e principalmente jogando. Os principais sites utilizados foram Bulbapedia e Serebii.

1.1 Desenvolvimento

O desenvolvimento desta etapa se deu pensando em um módulo de python que vai tratar de assuntos relacionados a um pokémom. Como ainda não foi necessário desenvolver um módulo de batalha adequado, foi criada a classe *Duel* que define os critérios de prioridade e executa adequadamente as ações que esta está julgando.

Futuramente espero o desenvolvimento de mais três módulos:

- **player:** Coordena quais os pokémons o jogador possui e manda as ordens adequadas a cada um deles;
- **battleserver:** Servidor de batalha, que permite a comunicação entre dois jogadores;
- **AIplayer:** Inteligência artificial para as batalhas;

O desenvolvimento se deu exclusivamente de acordo com o que estava especificado no enunciado até que foi notada a necessidade de um desempate de prioridades de ataque mais criteriosa, tal qual no jogo.

O algoritmo de desempate foi desenvolvido de forma a manter o fluxo de batalha semelhante ao jogo, por sua vez desrespeitando o enunciado. Tais problemas foram discutidos indiretamente no Paca e todo o desenvolvimento já estava concluído quando o assunto foi discutido adequadamente, portanto não foi feito.

1.2 Modo de execução

Existem quatro maneiras de se executar o programa principal:

- `./battle.py -h`: Imprime uma ajuda, `-h` pode ser substituído por `--help`;
- `./battle.py`: Modo default de execução, recebe os argumentos pela entrada padrão do sistema.
Por comodidade é aconselhável a utilização da seguinte maneira:
`cat <arquivo1> <arquivo2> | ./battle.py`
- `./battle.py -b`: Utiliza o diretório padrão de arquivos de pokémons, `./billpc/`
- `./battle.py --billpc=<diretório>`: Utiliza o diretório passado por argumento para buscar pokémons.

1.3 Teste

Também foi desenvolvido um teste, que verifica se dois pokémons originados do mesmo arquivo selecionado aleatoriamente de `./billpc/` realmente montam objetos distintos mas possuem alguns atributos instanciados no mesmo lugar.

Para executá-lo basta digitar `./teste_pokemon.py`.

2 Descrição das Classes

Nesta etapa foram desenvolvidas 23 classes. Destas existem 4 classes principais e algumas destas sofrem especialização para adequar suas funções à dinâmica do jogo.

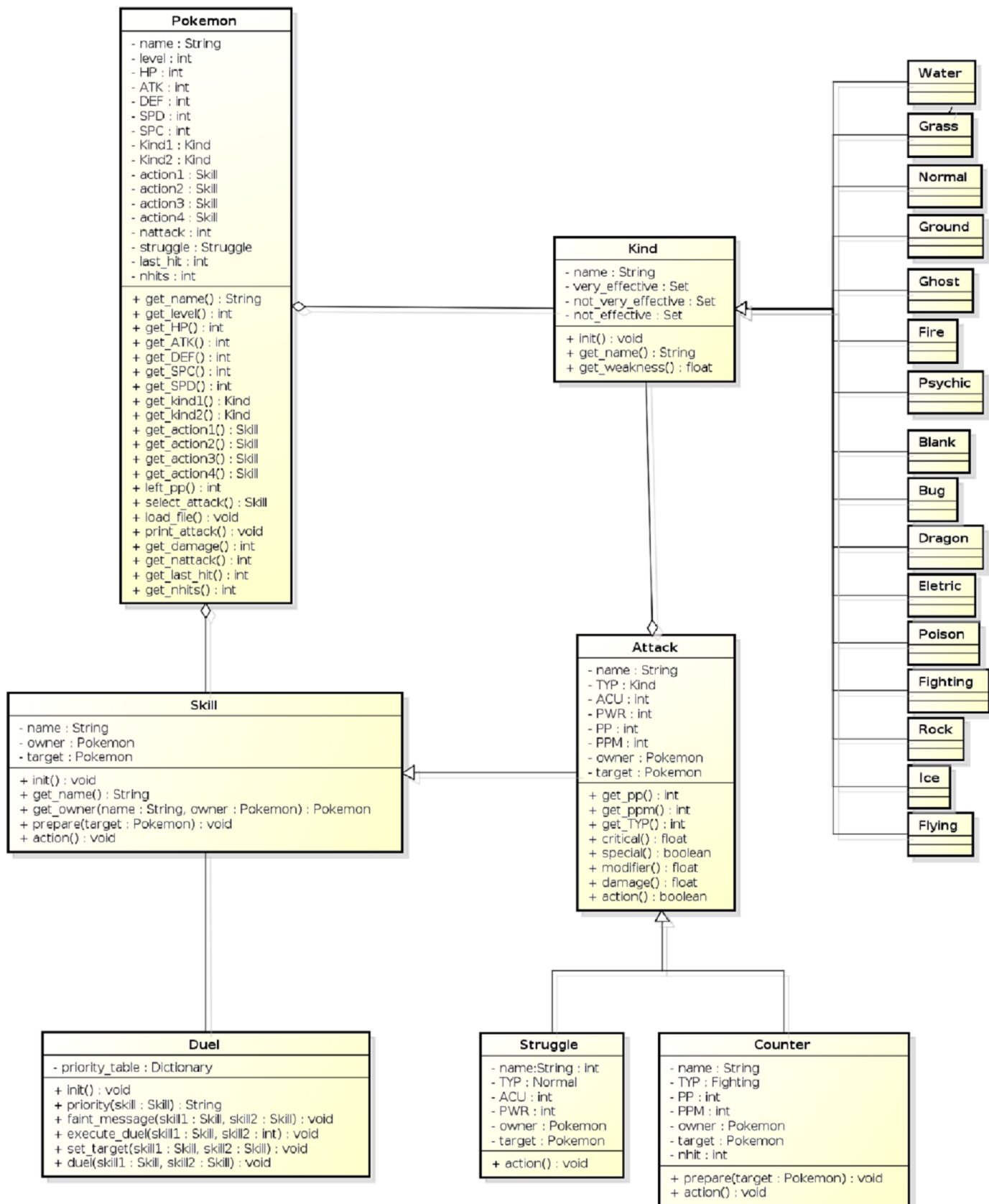


Diagrama de classes

2.1 Pokémon

Esta é a classe de maior importância. Agrega as demais classes de maneira a gerenciar e servir de interface principal de acesso às outras classes do módulo.

Possui os seguintes atributos:

- **name**: Nome deste pokémon;
- **level**: Nível deste pokémon;
- **HP**: HP deste pokémon;
- **ATK**: Ataque base deste pokémon;
- **DEF**: Defesa deste pokémon;
- **SDP**: Velocidade deste pokémon;
- **SPC**: Atributo para ataques especiais;
- **kind1**: Primeiro tipo deste pokémon;
- **kind2**: Segundo tipo deste pokémon;
- **action1**: Primeira ação deste pokémon;
- **action2**: Segunda ação deste pokémon;
- **action3**: Terceira ação deste pokémon;
- **action4**: Quarta ação deste pokémon;
- **natack**: Número de ataques disponíveis para este pokémon;
- **struggle**: A ação *Struggle* está disponível para todos os pokémons e portanto é uma ação extra;
- **last_hit**: Último dano que este pokémon, utilizado na implementação de *Counter*;
- **nhits**: Total de danos que este pokémon já sofreu, utilizado na implementação de *Counter*;

Possui os seguintes métodos:

- `__init__()`: Inicializador de um *Pokemon* vazio;
- `get_name()`: Retorna o nome;
- `get_level()`: Retorna o nível;
- `get_HP()`: Retorna o HP;
- `get_ATK()`: Retorna o ataque;
- `get_DEF()`: Retorna a defesa;
- `get_SPC()`: Retorna o atributo especial;
- `get_SPD()`: Retorna a velocidade;
- `get_kind1()`: Retorna o primeiro *Kind*;
- `get_kind2()`: Retorna o segundo *Kind*;
- `get_action1()`: Retorna a primeira *Skill*;
- `get_action2()`: Retorna a segunda *Skill*;
- `get_action3()`: Retorna a terceira *Skill*;
- `get_action4()`: Retorna a quarta *Skill*;
- `left_pp()`: Retorna a quantidade de PP restantes em todas as *Skills* disponíveis;
- `select_attack(n)`: Retorna o *n*-ésimo ataque;
- `load_file(name)`: Carrega um *Pokemon* de um arquivo cuja caminho é passado pelo parâmetro `name`;
- `load()`: Carrega um *Pokemon* através da entrada padrão;
- `print_attack()`: Imprime os ataques disponíveis;
- `get_recoil(damage)`: Recebe o dano e decrementa o HP;
- `get_damage(damage, kind)`: Recebe a tupla dano e tipo de ataque e decrementa o HP;
- `get_nattack()`: Retorna a quantidade de ataques disponíveis;
- `get_last_hit()`: Retorna o dano recebido no último ataque, utilizado na implementação de *Counter*;
- `get_last_kind_hit()`: Retorna tipo do ultimo ataque sofrido;
- `get_nhits()`: Retorna a quantidade de hits que este *Pokemon* já sofreu, utilizado na implementação de *Counter*.

2.2 Skill

A classe *Skill* é o topo de um relacionamento que segue o padrão *Abstract Factory*, sendo especializada para cada uma das possíveis ações que um pokémon pode realizar durante o jogo.

Possui os seguintes atributos:

- **name**: Nome desta habilidade;
- **owner**: Pokémon dono desta habilidade;
- **target**: Pokémon alvo desta habilidade.

Possui os seguintes métodos:

- **__init__(name, owner)**: Inicializa uma habilidade;
- **get_name()**: Retorna o nome da habilidade;
- **get_owner()**: Retorna o dono desta habilidade;
- **get_target()**: Retorna o alvo desta habilidade;
- **prepare(target)**: Seleciona o alvo *target*;
- **action()**: Realiza a ação desta habilidade.

Esta classe sofre uma especialização para *Attack* que, por sua vez é especializada em *Struggle* e em *Counter*. Servindo assim de interface para qualquer nova habilidade que o pokémon possa vir a desenvolver, por exemplo habilidades de cura do próprio pokémon, que não necessita de *target* ou a ordem dada pelo treinador para que o pokémon utilize um item ou seja substituído em batalha.

Assim como ocorre na classe *Kind*, é possível evitar desperdício de memória utilizando o padrão de projeto *Singleton*. Para tanto seria necessário desacoplar a informação sobre o *PP*, já que é a única parte variável entre os pokémons trazendo muito mais dificuldades do que facilidades.

2.2.1 Attack

Esta classe, herdeira de *Skill*, define um ataque genérico de um turno direto ao alvo. Possui os seguintes atributos:

- **name**: Herdado de *Skill*;
- **TYP**: Composição para a classe *Kind*;
- **ACU**: Acuidade deste ataque;
- **PWR**: Poder base deste ataque;
- **PP**: Power Point atual deste ataque;
- **PPM**: Power Point máximo deste ataque - pode ser utilizado caso implementemos o uso de itens;
- **owner**: Herdado de *Skill*;
- **target**: Herdado de *Skill*.

E possui os seguintes métodos:

- **__init__(name, typ, acu, power, pp, owner)**: Inicializa um ataque;
- **get_pp()**: Retorna os Power Points restantes deste ataque;
- **get_ppm()**: Retorna os Power Points máximos deste ataque;
- **get_TYP()**: Retorna o *Kind* deste ataque;
- **critical()**: Retorna o valor do incremento de um ataque crítico ou 1;
- **special()**: Retorna se é um attack especial.;
- **modifier()**: Retorna o modificador do cálculo do dano entre dois pokémons.;
- **damage()**: Retorna o dano deste ataque entre dois pokémons.;
- **action()**: Realiza um ataque caso ainda possua PP.

2.2.2 Struggle

Diferente de um ataque qualquer, todo pokémon pode realizar um struggle, para tanto basta que não possua mais nenhum *PP* restante. Sobreescreve apenas o construtor e a action:

- **__init__(owner)**: Inicializa uma struggle, como todos os outros atributos além do dono já estão definidos, não é necessário os passar ao construtor;
- **action()**: Nesta ação é necessário que o utilizador receba metade do dano infligido ao alvo, portanto foi sobreescrita.

2.2.3 Counter

O ataque *Counter*, caso o usuário receba algum dano de um ataque *Normal* ou *Fighting*, causa o dobro deste dano ao oponente. Por depender do ataque do oponente é o ataque com menor prioridade, possibilitando assim a utilização deste ataque por pokémons muito velozes.

- **__init__(pp, owner)**: Inicializa um counter, como todos os outros atributos além do dono e do pp já estão definidos, não é necessário os passar ao construtor;
- **action()**: Nesta ação é necessário que o utilizador devolva o dobro do dano infligido pelo alvo anteriormente, portanto foi sobreescrita.

2.3 Kind

Assim como *Skill* esta classe é o topo de um relacionamento que segue o padrão *Abstract Factory*, possui um escopo bastante simplificado e é especializada em todos os tipos possíveis. Sendo utilizada nos dois tipos dos pokémons e no tipo do ataque.

Inicialmente esta classe se chamaria *Type*, mas para evitar problemas com a função nativa *type* resolvemos modificar o nome para *Kind*.

Os atributos foram pensados de forma a tornar todo o escopo de efetividade visível dentro do próprio tipo. Possui os seguintes atributos:

- **name**: Nome da *Kind*;
- **very_effective**: Set de *Kinds* que ataques deste tipo causam duas vezes mais dano;
- **not_very_effective**: Set de *Kinds* que ataques deste tipo causam metade do dano;
- **not_effective**: Set de *Kinds* que ataques deste tipo não causam nenhum dano.

Possui apenas dois métodos além do construtor:

- **__init__(pp, owner)**: Inicializa um *Kind* vazio;
- **get_name()**: Retorna o nome da *Kind*;
- **get_weakness(kind)**: Retorna o multiplicador de fraqueza específico desta *Kind* em relação a outra - por enquanto a comparação é dada pelo nome.

Sendo especializada para cada um dos tipos existentes em *Pokémon Geração 1* e em cada uma destas especializações o método construtor é sobreescrito por outro que inicializa os sets de fraqueza adequadamente. Facilitando muito a visualização das fraquezas que esta *Kind* enfrenta.

Decidimos utilizar o padrão de projetos *Singleton* para as classes herdadas. Evitando assim um desperdício elevado de memória, já que estes objetos são agregados em todos os pokémons e ataques.

2.4 Duel

Esta classe é responsável por definir a prioridade entre dois ataques e os executar adequadamente. Ela não faz muito sentido no escopo do módulo *pokemon*. De tal forma que provavelmente deva ser realocada ou reimplementada em um módulo que trate especificamente da batalha.

Possui os seguintes métodos:

- **`__init__()`**: Inicializa um dicionário com os ataques de maior prioridade, que é o único atributo da classe;
- **`priority(skill)`**: Retorna a prioridade do ataque passado por argumento;
- **`faint_message(skill1, skill2)`**: Recebe como argumento dois objetos da classe Skill e imprime uma mensagem de morte caso um dos donos da skill tenha desmaiado devido a falta de HP. Resolve o problema do duplo nocaute;
- **`execute_duel(skill1, skill2)`**: Executa o duelo já pré ordenado. Caso o segundo pokémon tenha desmaiado não prossegue com o ataque e chama *faint_message()*, caso contrário executa a ação do segundo pokémon e chama *faint_message()* para imprimir possíveis mensagens de derrota;
- **`set_target(skill1, skill2)`**: Inicializa as ações de ambos os pokémons passando o dono da outra ação como parâmetro;
- **`duel(skill1, skill2)`**: Recebe as duas skills a serem executadas, as inicializa com *set_target()* e define a ordem de execução com *priority()* e comparando a velocidade dos pokémons quando necessário. Então as executa usando *execute_duel()*. É o único método que não é privado.

3 Conclusão

O objetivo principal desta etapa é a realização de um duelo entre dois pokémons em modo texto. Recebendo estes pokémons de arquivos ou da entrada padrão. Um dos requisitos desta etapa foi a utilização de Programação Orientada a Objetos (POO).

Foi possível verificar que o python tem boas formas de tratar a POO facilitando muito a manutenção de código. Para isso as definições das responsabilidades de cada uma das classes teve de ser muito bem pensada.

Embora a quantidade de arquivos que foram desenvolvidos tenha sido um pouco elevada, eles apenas implementam o que é necessário de acordo com as responsabilidades da classe. Horas sobreescrevendo atributos, como o caso das classes herdadas de *Kind* ou então sobreescrevendo métodos como nas especializações que a classe *Attack* sofre.

A escolha da classe *Skill* foi feita pensando na necessidade de implementação futura de ataques com modificação de atributos que não sejam o HP, que é de responsabilidade da classe *Attack* - habilidades de cura ou alteração de algum status do pokémon, por exemplo. Sendo que na etapa atual não se faz necessária e pode ser removida facilmente caso preciso.

Referências

- [1] Enunciado
- [2] Bulbapedia
- [3] Serebii
- [4] Pokémon Red e Blue - Wikipedia
- [5] Modules - Python