# Relatório do Projeto – Batalha Pokémon


| **Nomes:** | **Nusp:** |
| --- | --- |
| Eduardo Delgado Coloma Bier | 8536148 |
| Fernanda de Camargo Magano | 8536044 |
| Florence Alyssa Sakuma Shibata | 7971705 |
| Shayenne da Luz Moura | 8536235 |

----------***CLASSES, MÉTODOS E DECISÕES***----------

- Foi feita uma classe display para mostrar os pokémons e ataques;
- Foi decidido implementar também as classes: Pokemon, batalha, ataque, leitor;
- Colocou-se no _init o valor -1 para deixar mais visível caso falhasse a leitura dos arquivos;
- A classe leitor foi separada como uma classe de leitura para melhor distinção entre a parte do jogo e a de leitura dos dados;
- Decidiu-se usar “set's” e “get's” para deixar o código organizado e claro, já que em cada set e get fica evidente qual é o atributo em questão;
- Tudo o que foi usado faz parte do próprio python, por isso não foi criado o requirements.txt;

----------***CLASSES E DINÂMICA***----------

- Foram criadas as classes cliente e servidor;
- É necessário o xml para atualização dos pokemons e funciona da seguinte maneira: o cliente lê seu pokemon e envia o objeto battle_state via Post para o servidor. A resposta do servidor é um objeto battle_state com dois pokémons.
- A ideia da batalha é que com base nessa resposta do servidor, o cliente escolha o ataque que achar mais adequado. Então envia para o servidor, o qual contabiliza e contra-ataca.

----------***CLASSE LEITOR:***----------

- Uma das decisões tomadas foi montar uma lista no momento em que está sendo feita a leitura dos dados. Embora pareça que não seja de suma importância, a lista se faz importante, visto que permite um melhor controle das informações contidas na entrada, isto é, se têm linhas vazias, campos a mais ou a menos que possam tornar o pokémon ou o ataque inválidos;

----------***CLASSE BATALHA:***----------

- Responsável pela dinâmica do jogo, instanciar objetos que serão os pokémons, calcular o dano;
- Controla qual pokémon inicia, alterna os turnos, lê a tabela de tipos, calcula o critical, modifier, stab;
- Verifica se o jogo termina;

----------***CLASSE DISPLAY:***----------

- Caso o usuário não queira que apareçam os print's da batalha na tela, pode digitar a opção -s (para silenciar); a opção -v (verbose) mostra Pokémons e ataques carregados.
- Foi implemtentada a impressão da barra de hp do Pokémon.

----------**CLASSE CLIENTE**----------

- Uma das responsabilidades dessa classe é escrever o xml. Para essa finalidade foi criada uma árvore e acrescentados os nós. Cria-se o objeto battle_state. Assim, ET.Subelement foi usado para isso. O '.text' foi necessário para a edição do xml.
- Desse modo, nome, tipo, ataques, defesa, PP, entre outros são alterados durante a batalha.
- No final desse método de escrita presente na classe cliente, foi utilizado o método tostring, para a conversão da árvore numa string.
- Outro método importante foi o responsável por iniciar a batalha e fazer a comunicação ao servidor por meio dos requests.

----------**CLASSE SERVIDOR**----------

- Lê o pokemon que já está no xml (o do cliente) e, além disso, atualiza com o seu próprio pokemon;
- É no servidor que a batalha é criada.
- Utiliza o método POST, além de importar o Flask.

----------**IP E LOCAL HOST**----------

- O programa funciona usando o local host, mas é possível também colocar o ip de outras máquinas, alterando: **'def run(self, ip = '127.0.0.1', port = 5000)'** que está na classe server e o **def __init__(self, execute = False, ip = '127.0.0.1', port = 5000)** que está na classe cliente. A porta será alterada para 8080.
- Um detalhe importante é que foi notado que o uso de roteadores pode atrapalhar a

batalha entre servidor e cliente. Esse problema ocorre usando o ip externo. Contudo, se for numa rede cabeada ou usando roteador (mas com o ip interno) funciona sem problemas.

## ----------DIFICULDADES E ESCOLHAS----------

- No começo a ideia era instanciar os pokémons fora da classe batalha, mas conforme a montagem do código foi ocorrendo, notou-se que era mais coerente fazer dentro da classe. Isso porque pensando no jogo, a batalha contém pokémons. Além disso, outro fato fundamental nessa escolha foi controlar melhor o escopo do pokémon, evitando que ele fosse global.
- Uma pequena dificuldade deparada foi para testar o damage, o qual estava dando valores muito altos. Concluiu-se que o problema seria no método, mas foi percebido em seguida que deveria ser feita uma construção bem cuidadosa na escolha de alguns atributos do Pokémon para um cálculo mais próximo da realidade do jogo;
- A questão do redirecionamento foi discutida, mas o problema foi resolvido usando arquivos que continham as informações para leitura e simulando a batalha pela entrada padrão.
- Inicialmente não estava muito claro o que deveria ser feito com o arquivo de extensão xsd. Tinha sido interpretado que esse seria o xml com o qual deveria-se manipular e extrair as informações necessárias. Depois de muitas pesquisas foi compreendido qual era o intuito de um Schema e que deveria-se, a partir deste, gerar o xml;
- Outra dificuldade foi na busca de ferramentas de manipulação do xml compatíveis com Python 3. Foi decidido que a melhor opção seria utilizar o 'xml.etree.ElementTree as ET'.

## ----------INTELIGÊNCIA ARTIFICIAL – FASE 3----------

- A ideia dessa fase 3 é que o jogo tenha IA e escolha o ataque mais apropriado de acordo com os Pokémons.
- Então, busca-se o ataque de maior dano;
- Para tal finalidade, ao olhar para a fórmula do dano, percebe-se que dois fatores responsáveis por um dano maior são: o atributo PWR do ataque e os modifiers (que dependem do ataque e do tipo de Pokémon que vai se defender).
- Contudo, outro fator importante que pode ser pensado é o fato de que deve-se ponderar e pensar qual é a melhor escolha para o dilema: grande dano e pequena acurácia ou pequeno dano com grande acurácia.

- Não adianta escolher um ataque com acurácia muito boa e dano pequeno se o hp do seu pokémon está se esgotando. Mais vale um grande estrago que, se atingir o alvo, pode proporcionar vitória, ou pelo menos uma derrota menos vergonhosa.
- O STAB também foi considerado na escolha do ataque, já que se os dois Pokémons tiverem o mesmo tipo (por exemplo, os dois forem elétricos) terá esse bônus.
- Esses fatos foram levados em consideração no método da batalha.py chamado de **EscolheAtaqueInteligente.**

----------**TESTES**----------

- Foram testados os métodos das duas importantes classes: 'server' e 'cliente', além do ataque, batalha, display e pokemon.
- Para rodar:
  **python3.4 test.py < 20Pikachu.txt**
  **python3.4 test2.py < errado.txt**

- Esta última (errado.txt) testa o caso de entradas inválidas para a classe Leitor.

----------**COMO EXECUTAR O PROGRAMA:**----------

- Para ver a batalha em ação, basta colocar no terminal: **cat nome_do_arquivo.txt - | python3 main.py**
- No lugar do nome_do_arquivo.txt, coloque um dos arquivos a seguir: **PikachuSolo.txt, StrPikachu.txt, RaichuSolo.txt ou TestePokemon.txt**
- Para começar o jogo pode-se fazer, por exemplo, ele contra ele mesmo: **cat RaichuSolo.txt - | python3.4 main.py -S** para o servidor e **cat RaichuSolo.txt - | python3.4 main.py -C** para o cliente, abrindo dois terminais;
- Os arquivos **PikachuSolo.txt e StrPikachu.txt** já têm as batalhas prontas, isto é, não deixa o usuário escolher os ataques (foi usado para testes). Caso queira escolher os ataques, pode ser usado o **TestePokemon.txt** ou **Raichu.txt.**
- Para usar a IA, seja no servidor ou no cliente, deve-se colocar a opção -i. Exemplo: **cat nome_do_arquivo.txt - | python3.4 main.py -S -i** para o servidor (ou -C para o cliente).