Trabalho prático

Instruções

1. **O trabalho deve utilizar a linguagem Java.**
   - Deve conter o mínimo de classes do modelo especificado no Apêndice B, bem como as classes concretas e as classes de controladoras.
   - Novas classes podem ser criadas conforme a necessidade.
   - Poderá ser desenvolvido individualmente ou em duplas
2. **O trabalho de ser entregue via Moodle.**
   - Haverá um link de entrega no sistema para fazer o *upload* do arquivo.
   - O trabalho deverá ser entregue **até as 19:00 hrs do dia 18/12/2017**.
   - Coloque os nomes e R.A.s da dupla como nome do arquivo compactado. Ex: FulanoDeTal23455_SicranoDeOutro65432.rar (sem espaço);
3. **Devem ser entregues, via Moodle, os seguintes artefatos compactados em um arquivo**:
   - Código fonte do programa;
   - Documento que descreve as classes definidas no trabalho – ver Apêndice A.
4. **O trabalho deverá ser apresentado pelas duplas** a partir das 19:30 hrs do dia 18/12/2017, conforme cronograma a ser disponibilizado no Moodle.
   - Embora o trabalho possa ser feito em dupla, isso não garante mesma nota para ambos os integrantes da equipe.
   - As apresentações serão feitas na sala da Profa. e abrangerá aspectos relacionados a implementação e as decisões de projeto.

Objetivo do Trabalho

Aplicar os conceitos de orientação a objetos na programação de um sistema de software que simula uma batalha do jogo de RPG "Pokémon Red and Blue Versions", com regras simplificadas.

Descrição do Trabalho

Introdução

"*Pokémon Red Version e Pokémon Blue Version são as versões ocidentais dos primeiros jogos da série multimilionária Pokémon. Anunciados em outubro de 1995 e Lançados em 1996 no Japão como Red e Green e em 1998 nos EUA como Red e Blue, a história se passa no continente de Kanto, um lugar onde existem criaturas exóticas conhecidas como Pokémon. O protagonista é um garoto que vai iniciar sua jornada Pokémon na Cidade de Pallet e viajar por Kanto para se tornar um mestre Pokémon.*

O maior aspecto do jogo é o jogador treinar e melhorar seu Pokémon, tanto derrotando Pokémon selvagens como os de treinadores. Esse sistema de pontos de experiência, característica de todos os RPGs da série, controlam as propriedades físicas do Pokémon, incluindo os atributos de batalha e HP (Health Points). Quando o jogador encontra um treinador ou um Pokémon, começa uma batalha baseada em turnos para poder fazer com que o HP do adversário chegue a 0" (Wikipedia).

Neste trabalho deverá ser simulada uma batalha Pokémon simplificada entre dois times distintos controlados pelo computador ou por jogadores humanos. A batalha pode ser feita de modo "textual", ou seja, **apenas usando-se as saídas e entradas textuais padrão do Java (não necessitando qualquer *display* gráfico do jogo)**. O Apêndice B mostra o diagrama de classes que define as classes básicas que devem ser implementadas.

Visão geral

Cada jogador possui um time composto por até 6 pokémons. Cada pokémon tem características que variam com a sua espécie (ver o arquivo tabelaEspeciesPokemon.txt) e possui de 1 a 4 tipos de ataque (ver o arquivo tabelaAtaquesPokemon.txt). Apenas um pokémon de cada time (o primeiro do vetor de pokémons) participa da batalha em cada turno. A batalha segue o fluxograma mostrado na Figura 1.

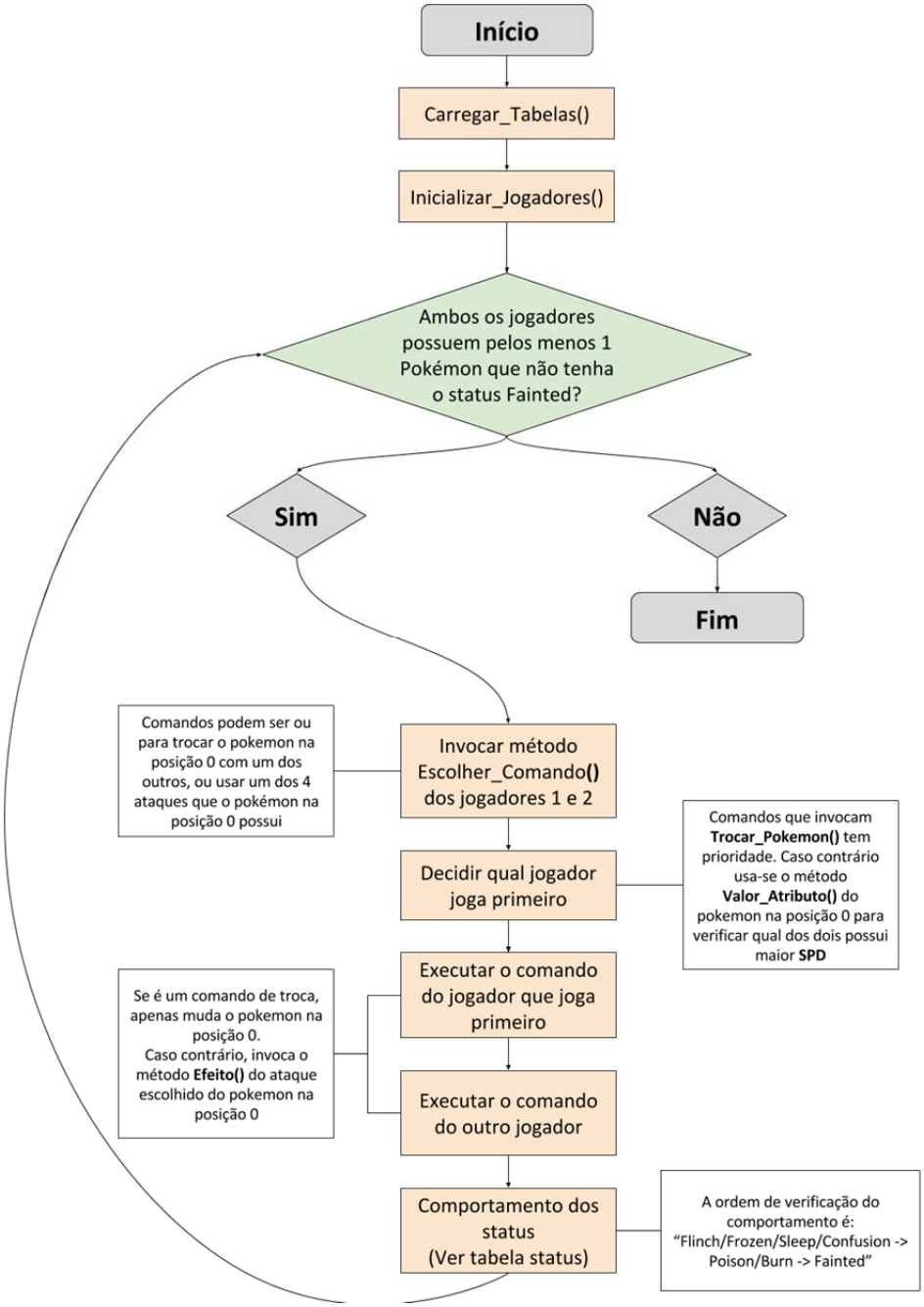


Figura 1: Fluxograma principal da batalha

No início da execução do programa algumas tabelas contendo as informações do jogo são carregadas (todas as tabelas necessárias encontram-se anexadas a este documento). Em seguida, leem-se os parâmetros passados ao início da execução do programa para que se inicializem os times que participarão da batalha.

Em seguida, entra-se no *loop* principal da batalha. Enquanto ambos os jogadores tiverem pokémons em seus times que possam lutar (isto é, que não tenham o status *Fainted*), um novo turno é executado. Caso contrário, a batalha termina.

A Figura 2 apresenta o Fluxograma para a execução de um turno da batalha:

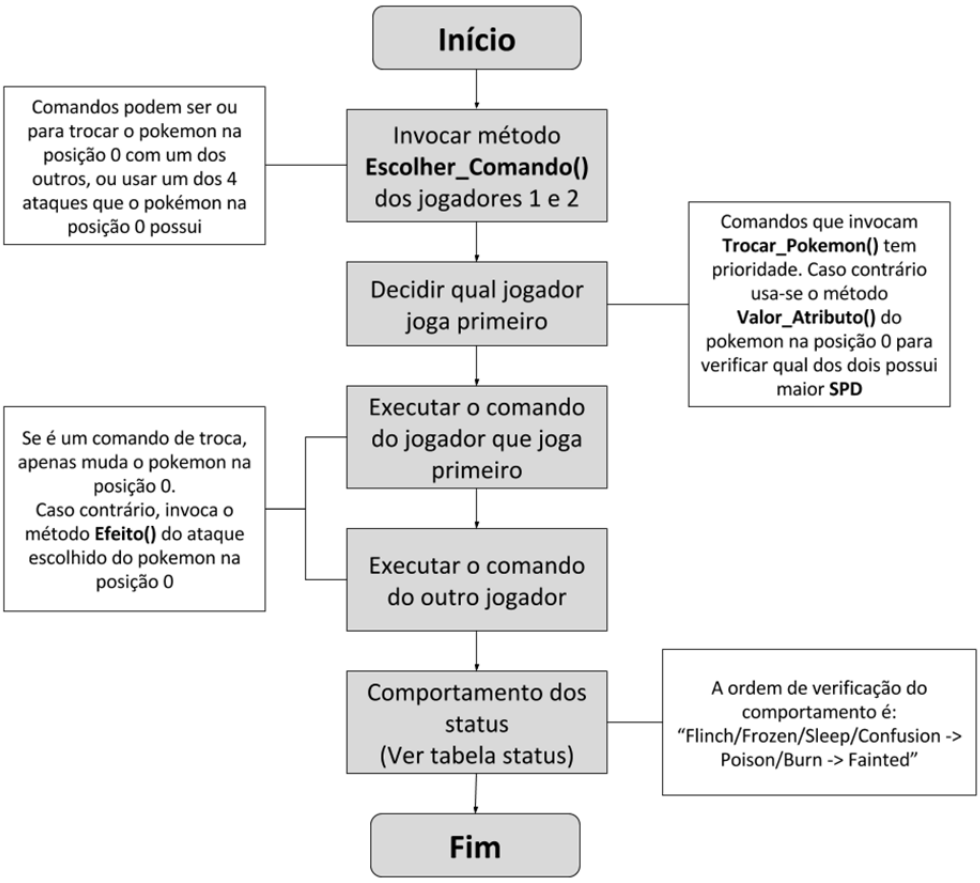


Figura 2: Fluxograma da função Executa_Turno()

O funcionamento básico de um turno é:

- Ambos os jogadores escolhem que comando de batalha querem executar 'às cegas' (isto é, ambos escolhem os comandos e somente depois os comandos serão executados);
- Os comandos possíveis são:
    - **Trocar pokémon principal** - Na batalhas apenas o pokémon à frente de cada time pode participar ativamente, mas os jogadores podem ter até 6 pokémons em seus times. Esse comando permite que o jogador troque o pokémon que está participando da batalha (Posição 0 do vetor) com qualquer um dos outros participantes (Posições 1 a 5 do vetor), desde que eles possam lutar;
    - **Usar um ataque do pokémon principal** - Todo pokémon possui de 1 a 4 ataques que pode utilizar na batalha. Uma explicação mais detalhada sobre os tipos de ataques será dada em seguida;

- Após a escolha dos comandos de ambos os jogadores, checa-se qual deles é executado primeiro;
    - Comandos de troca acontecem primeiro que comandos de ataque;
    - Caso ambos os comandos sejam do mesmo tipo (ambos são comandos *Troca* ou de *Ataque*), o empate é resolvido comparando-se a **Velocidade atual (SPD)** dos pokémon à frente dos times;
- Decidindo-se qual time jogo primeiro, executa-se o comando escolhido por cada jogador;
- Ao final do turno, são feitas as verificações de status necessárias. O efeito de cada status será discutido em seguida;
- Note que quando os **Pontos de Vida (HP)** de um pokémon chegam a 0, ele recebe o status ***Fainted*** e é **automaticamente trocado pelo próximo pokémon do time que não tem o status *Fainted***. Se essa troca não é possível, o time não possui mais pokémons que podem participar da batalha, perdendo então a partida.

Parâmetros de Inicialização

No início do programa são passados parâmetros para que ambos os times sejam inicializados corretamente. Essa inicialização pode ser feita por linha comando, conforme explicado a seguir.

Cada time é definido por uma sequência ordenada de números, como segue:

- O primeiro parâmetro somente pode ser **0 ou 1** e especifica se o time em questão será controlado pelo **computador** ou por um **jogador humano**;
- O segundo parâmetro é um número de **1 a 6** e representa **quantos pokémons compõem o time especificado**. Em seguida, para cada pokémon no time:
    - O primeiro parâmetro é um número de **1 a 151** e representa a **espécie do pokémon** descrito;
    - O segundo parâmetro é um número de **1 a 100** e representa o **Lvl (Nível)** do pokémon descrito;
    - Os próximos quatro parâmetros são números de **0 a 165** e representam os quatro **ataques do pokémon** descrito;
        - Note que, mesmo que um pokémon possa ter menos de 4 ataques, sempre são passados 4 ids por parâmetro;
        - Sendo assim, quando o pokémon não possuir todos os 4 ataques, os espaços vazios são representados pelo número **0**;

- Repete-se, então, a mesma verificação acima para a definição do segundo time que participará da batalha;

Exemplos de inicialização de batalhas válidas:

- 1 1 6 40 23 25 0 0 0 2 7 20 157 0 0 0 2 30 33 0 0 0
    - Time 1 - Controlado por um jogador (1) - 1 pokémon:
        - Charizard (#6) - Lvl 40
            - Ataques: Flamethrower (#23) e Fly (#25)
    - Time 2 - Controlado pelo computador (0) - 2 pokémons:
        - Squirtle (#7) - Lvl 20
            - Ataques: Bubble (#157)
        - Ivysaur (#2) - Lvl 30
            - Ataques: Absorb (#33)
- 0 1 11 8 80 0 0 0 0 1 11 8 80 0 0 0
    - Time 1 - Controlado pelo computador (0) - 1 pokémon:
        - Metapod (#11) - Lvl 8
            - Ataques: Harden (#80)
    - Time 2 - Controlado pelo computador (0) - 1 pokémon:
        - Metapod (#11) - Lvl 8
            - Ataques: Harden (#80)

Classe Jogador

Representa os jogadores que participam da batalha, com os seguintes métodos.

- Escolher_Comando() é um método abstrato, que é devidamente implementado nas classes Jogador_IA e Jogador_Humano:
    - Em Jogador_IA, o método sempre escolher aleatoriamente um dos ataques do pokémon da posição 0 do time, nunca executando comandos de troca;
    - Em Jogador_Humano, o método pergunta ao usuário que ação ele quer tomar entre troca e ataque, e em seguida pergunta-se qual pokémon será trocado com o atual, ou qual ataque do pokémon atual será usado. Verificações devem ser feitas para que apenas posições/ataques válidos sejam escolhidos;
    - Em ambos os casos, a classe deve retornar o comando escolhido para a classe Batalha;

- Trocar_Pokemon() é invocado na classe Batalha pela classe Jogador, e apenas troca de lugar o pokémon na posição 0 da equipe com o pokémon desejado;
- Usar_Ataque() é invocado pela classe Batalha, e invoca o método Efeito() do ataque escolhido (entre os 4 possíveis).

Classe Especie

Contém informações básicas sobre as espécies de pokémon possíveis na batalha. No início do programa lê-se a tabela de espécies disponibilizada juntamente com esta especificação para carregar as informações das 151 espécies de pokémons. Cada espécie possui os seguintes atributos e métodos:

- id é o número do pokémon na tabela de espécies;
- Nome é o nome da espécie;
- Tipo1 e Tipo2 são os tipos da espécie, usados para calcular alguns tipos de dano dos ataques;
- Base_HP, Base_ATK, Base_DEF, Base_SPE e Base_SPD são os atributos-base para cada espécie, representando Vida, Ataque, Defesa, Especial e Velocidade, respectivamente;
- Calculo_Atributo() é chamado pela classe Pokemon quando ela é criada e retorna os atributos de um pokémon pertencente a espécie descrita, quando está no nível atual:
    - Para calcular o valor de HP_Atual/HP_Max:
        - HP = 2 * B * L / 100 + L + 10
        - Onde L é o Lvl do pokémon e B é Base_HP da espécie
    - Para calcular o valor inicial dos outros atributos (ATK/DEF/SPE/SPD):
        - atributo = 2 * B * L / 100 + 5
        - Onde L é o Lvl do pokémon e B é Base_atributo da espécie.

Classe Pokemon

Representa cada pokémon que está sendo usado na batalha, com os seguintes atributos e métodos:

- Lvl é o Level (Ou Nível) atual, representando basicamente o quão forte o pokémon é;
- HP_Atual é a quantidade de pontos de vida restantes ao pokémon, nunca podendo ser maior que seu HP_Max;

- HP_MAX/ATK/DEF/SPE/SPD são os atributos do pokémon:
    - HP_MAX - Quantidade máxima de vida;
    - ATK - Valor normal de Ataque;
    - DEF - Valor normal de Defesa;
    - SPE - Valor normal de Especial (Usado para cálculo de dano em certos ataques);
    - SPD - Valor normal de Velocidade;
    - Esses atributos são calculados pela classe Espécie de cada pokémon, e não são alterados durante a batalha;
- Modifier_Accuracy/Evasion/ATK/DEF/SPE/SPD são modificadores dos atributos do pokémon, causados por certos ataques, que fazem com que o valor do atributo usado na batalha sejam melhores ou piores;
    - Modificadores podem ser apenas números entre -6 e 6;
    - No início da batalha, todos os modificadores começam em 0;
    - Mais sobre modificadores no método Valor_Atributo() desta classe;
- Status_Primario é o Status Primário que o pokémon tem;
    - Pode ser Ok, Fainted, Burn, Frozen, Paralysis, Poison ou Sleep. Mais sobre status em seguida;
- Status_Confusion e Status_Flinch representam os possíveis Status Secundários do pokémon:
    - Ambas apresentam apenas valores booleanos para guardar se o pokémon possui ou não os status;
    - Diferentemente dos status primários, um pokémon pode ter ambos os status secundários ativos ao mesmo tempo. Mais sobre status em seguida;
- Valor_Atributo() é o método usado por qualquer classe que queira acessar os valores de atributos (ATK/DEF/SPE/SPD) do pokémon;
    - Como existem ataques que aplicam modificadores aos atributos, este método implementa as fórmulas que retornam os valores de atributos modificados;
    - retorno = Atributo * (max(2, 2 + modifier) / max(2, 2 - modifier))
    - Note que existem 2 modificadores não utilizados nesta fórmula, o Modifier_Accuracy e Modifier_Evasion, que são usados pelas classes de ataque.

Classe Ataque

Representa os ataques que pokémons podem ter, possuindo 6 variações (subclasses) possíveis. No início do programa lê-se a tabela de ataques disponibilizada juntamente com esta especificação para carregar as informações dos ataques. A classe Ataque possui os seguintes atributos e métodos:

- id é o número do ataque na tabela de ataques;
- Nome é o nome do ataque;
- Tipo é o tipo do ataque, usado para cálculo de dano;
- Max_PP (Power Points) representa o número máximo de vezes que o ataque pode ser usado;
- PP_Atual representa o número de vezes que o ataque ainda pode ser usado e é decrementado toda vez que o ataque é executado;
- Power e Accuracy são atributos representando quão forte e preciso o ataque é (sendo que Accuracy representa uma porcentagem);
- Cálculo_Critico() é chamado para verificar se o ataque usado será crítico ou não, ou seja, se o ataque dará mais dano que o normal ao oponente;
    - A chance do método retornar Verdadeiro é:
    - SPD do atacante / 512
    - Ou seja, quanto maior a Velocidade do pokémon atacante, maior é a probabilidade do método retornar Verdadeiro;
- Calculo_Dano() é chamado por certas variações do ataque e calcula quanto de HP o pokémon atacado irá perder se o ataque acertar;
    - Os cálculos do dano acontecem nos seguintes passos:
        - L = Lvl do pokémon que está usando o ataque
        - P = Power do ataque
        - A = “Valor de ataque” do pokémon usuário
        - D = “Valor de defesa” do pokémon oponente
            - Se o tipo do ataque for: Normal, Fighting, Flying, Poison, Ground, Rock, Bug ou Ghost:
                - A = ATK do usuário
                - D = DEF do oponente
            - Se o tipo do ataque for: Fire, Water, Electric, Grass, Ice, Psychic ou Dragon:
                - A = SPE do usuário
                - D = SPE do oponente

- Se Calculo_Critico() retornar Verdadeiro
  - L *= 2
- Se o usuário estiver sendo afetado pelo status Burned
  - A /=2
- Valores negativos de A e D vão para 0
- Valores maiores que 255 para A e D vão para 255
- Dano = (L * A * P / D / 50) + 2
- Se o Tipo do ataque for igual ao Tipo1 ou Tipo2 do pokémon usuário:
  - Dano *= 1.5
- Multiplica-se o Dano de acordo com os valores da Tabela 1. Considerando-se o Tipo do ataque e os tipos (Tipo1 e Tipo2) do oponente, se obtém dois multiplicadores de dano: um para o Tipo1 e outro para o Tipo2.

Multiplicador de dano		Tipo do oponente														
		Normal	**Fighting**	**Flying**	**Poison**	**Ground**	**Rock**	**Bug**	**Ghost**	**Fire**	**Water**	**Grass**	**Electric**	**Psychic**	**Ice**	**Dragon**
Tipo do ataque	**Normal**	1	1	1	1	1	0.5	1	0	1	1	1	1	1	1	1
	Fighting	2	1	0.5	0.5	1	2	0.5	0	1	1	1	1	0.5	2	1
	Flying	1	2	1	1	1	0.5	2	1	1	1	2	0.5	1	1	1
	Poison	1	1	1	0.5	0.5	0.5	2	0.5	1	1	2	1	1	1	1
	Ground	1	1	0	2	1	2	0.5	1	2	1	0.5	2	1	1	1
	Rock	1	0.5	2	1	0.5	1	2	1	2	1	1	1	1	2	1
	Bug	1	0.5	0.5	2	1	1	1	0.5	0.5	1	2	1	2	1	1
	Ghost	0	1	1	1	1	1	1	2	1	1	1	1	0	1	1
	Fire	1	1	1	1	1	0.5	2	1	0.5	0.5	2	1	1	2	0.5
	Water	1	1	1	1	2	2	1	1	2	0.5	0.5	1	1	1	0.5
	Grass	1	1	0.5	0.5	2	2	0.5	1	0.5	2	0.5	1	1	1	0.5
	Electric	1	1	2	1	0	1	1	1	1	2	0.5	0.5	1	1	0.5
	Psychic	1	2	1	2	1	1	1	1	1	1	1	1	0.5	1	1
	Ice	1	1	2	1	2	1	1	1	1	0.5	2	1	1	0.5	2
	Dragon	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2

Tabela 1: Multiplicadores para todas as combinações de tipos possíveis

        - Em seguida, um inteiro R é escolhido aleatoriamente entre 217 e 255 e aplica-se a fórmula:
        - Dano = (Dano * R) / 255
        - Retorne Dano
- Calculo_Acerto() é chamado para calcular se o ataque acerta ou não o oponente:
    - A probabilidade de o ataque acertar é calculada pela fórmula:
    - Prob = AccuracyAtaque * (A/E)
    - sendo que A e E correspondem aos valores Modifier_Accuracy do atacante e Modifier_Evasion do oponente, respectivamente, seguindo a Tabela 2:

Modifier	Valor de Accuracy ou Evasion
-6	33%
-5	37%
-4	43%
-3	50%
-2	60%
-1	75%
0	100%
1	133%
2	166%
3	200%
4	233%
5	266%
6	300%

Tabela 2: Modificadores de Accuracy e Evasion e seus valores

    - Calcula-se então se o ataque acertou ou não, usando-se sua Prob de acerto;
- O método Efeito() possui comportamento diferente dependendo de que tipo de ataque é utilizado. Seu comportamento comum pode ser observado no fluxograma da Figura 3:

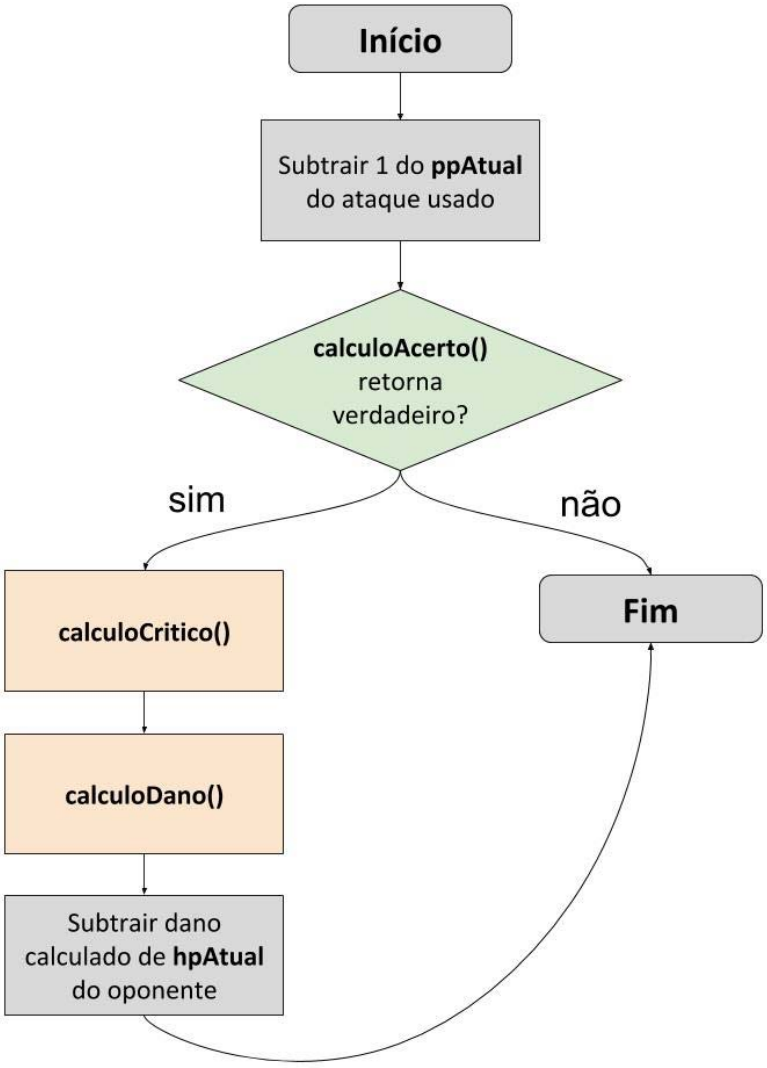


Figura 3: Fluxograma do método Efeito() de um ataque comum

Os demais tipos de ataques implementam o método Efeito() com variações deste fluxograma ou reescrevendo-o completamente.

A Tabela 3 descreve o comportamento do método Efeito() para as diferentes classes de ataque. Além do ataque comum (implementado pela classe Ataque), há 6 tipos de ataques especializados (implementados pelas subclasses de ataque). Nessa tabela, a coluna **Classe** identifica a classe do ataque; a coluna **Modificadores** identifica os atributos do ataque e a coluna **Efeito** descreve qual deve ser o efeito do ataque em questão. Note que os valores das colunas **Classe** e **Parâmetros** da tabela de ataques correspondem aos valores da coluna **Classe** e **Modificadores** da Tabela 3.

Classe	Modificadores	Efeito
comum		Nenhum efeito especial além do cálculo comum de dano. Classe padrão.
modifier	mod, n, chance	Além do cálculo comum de dano, o ataque possui chance de alterar modifiers do oponente ou do atacante, sendo que o máximo que um modifier pode chegar é -6 até 6. O primeiro valor [mod] apresenta qual modifier será alterado (podendo ser Accuracy, Evasion, ATK, DEF, SPE ou SPD). O segundo valor [n] representa o quanto o modifier será alterado (valores positivos alteram o modifier do atacante e valores negativos alteram o modifier do oponente). O terceiro valor [chance] representa a probabilidade em % da mudança do status ocorrer.
status	status, chance	Além do cálculo comum de dano, o ataque possui chance de dar ao oponente um status primário (Burn, Frozen, Paralysis, Poison, Sleep, Fainted) ou secundário (Confusion, Flinch). Um oponente pode ter apenas um status primário, mas pode ter vários status secundários de uma única vez. O primeiro valor [status] representa qual dos status será dado ao oponente e o segundo valor [chance] apresenta a probabilidade em % de que esse status ocorra (caso o ataque acerte o oponente).
multihit	min, max	O cálculo do dano do ataque é realizado diversas vezes aleatoriamente (a checagem de acerto e crítico é feita apenas uma vez no começo). O primeiro valor [min] define o número mínimo de vezes que o dano será dado ao oponente, enquanto o segundo valor [max] define o número máximo de vezes que o dano será aplicado.
hp	valor, porcentagem	Além do cálculo comum de dano, neste ataque o HP do usuário também é alterado. O valor de HP alterado pode ser calculado tanto a partir de seu HP máximo ou do dano que foi dado ao oponente. O primeiro valor [valor] pode ser "max_hp" ou "dano" e representa o valor base, enquanto que o segundo valor [porcentagem] representa o valor que será multiplicado ao primeiro. O resultado final é acrescentado ao HP atual do usuário.
fixo	val	Ignora todos os cálculos de dano comuns, dando sempre uma quantidade fixa de dano ao oponente. O valor [val] pode ser "lvl" ou um número. Se for igual a "lvl" o valor de dano é igual ao LVL do usuário, caso contrário o dano é o valor passado.
charge		O ataque "espera" até o próximo turno para fazer o cálculo de dano. Na prática seu turno é pulado e o oponente pode agir até duas vezes seguidas.

Tabela 3: Classes e comportamentos de ataques

Status Primários e Secundários

Durante a batalha, os pokémons podem fazer com que seus oponentes ganhem status que os afetam negativamente. Por ex., um pokémon pode ser envenenado e perder um pouco de vida a cada turno, ou congelado e não poder se mover, etc. A Tabela 4 apresenta os status que podem afetar os pokémon, bem como os efeitos de cada um:

Status Primários	
Somente um desses status pode estar ativo ao mesmo tempo	
Status	**Efeito**
Ok	Status padrão, nenhum efeito especial.
Fainted	Derrotado em batalha. Recebido quando HP vai para 0 ou por certos ataques. Pokémon não pode ser escolhido para ser trocado para a primeira posição. Ao final do turno em que entra nesse status, deve ser colocado na última posição do time, trazendo o da posição 1 para a posição 0 e assim por diante. Se todos os pokémons de um jogador possuem este status, a partida termina.
Burn	No final de cada turno, o pokémon afetado perde 6.25% de seu HP_MAX. Há também uma checagem no método Calculo_Dano() de seus ataques, que considera seu ATK como sendo 50% do valor normal enquanto ele possui este status.
Frozen	Enquanto este status está ativo, uma verificação adicional acontece no método Calculo_Acerto(), adicionando 100% de chance do ataque escolhido pelo usuário não ser executado. Este status tem 10% de chance de ser curado no final do turno.
Paralysis	Enquanto este status está ativo, uma verificação adicional acontece no método Calculo_Acerto(), adicionando 25% de chance do ataque escolhido pelo usuário não ser executado.
Poison	No final de cada turno, o pokemon afetado perde 6.25% de seu HP_MAX.
Sleep	Enquanto este status está ativo, uma verificação adicional acontece no método Calculo_Acerto(), adicionando 100% de chance do ataque escolhido pelo usuário não ser executado. Este status tem 20% de chance de ser curado no final do turno.
Status Secundários	
Todos podem estar ativos ao mesmo tempo	
Status	**Efeito**
Confusion	Enquanto este status está ativo, uma verificação adicional acontece no método Efeito() de todas as classes de ataques que usam o método Calculo_Dano(), adicionando 50% de chance do dano calculado ser dado ao usuário em vez do oponente. Este status tem 20% de chance de ser curado no final do turno.
Flinch	Enquanto este status está ativo, uma verificação adicional acontece no método Calculo_Acerto(), adicionando 100% de chance do ataque escolhido pelo usuário não ser executado. Este status sempre é curado no final do turno.

Tabela 4: Status e seus efeitos

Note que há status que assumem verificações e cálculos especiais nos métodos da classe Ataque e ao final de Executar_Tuno(). Note também que, enquanto um pokémon pode ter todos os status secundários ativos ao mesmo tempo, somente um status primário pode estar ativo em cada momento, ou seja, um novo status primário sempre remove o status que já estava ativo anteriormente.

Pontos importantes a serem considerados na realização do trabalho

1) Originalidade da implementação das classes concretas e classes de projeto.
2) Implementação das regras do jogo de forma correta
3) Uso dos conceitos de Programação Orientada a Objetos
    a) Encapsulamento
    b) Herança
    c) Polimorfismo

    Implemente o trabalho o mais orientado a objetos possível!
4) Organização das classes em pacotes (seguindo a convenção de Java)
5) Tratamento de exceções
6) Implementação ou uso de classes de interfaces (opcional)
7) Implementação ou uso de classes genéricas (opcional)
8) Documentação do sistema
9) Usabilidade do sistema:
    a) Clareza na execução das jogadas: qual jogador (usuário ou sistema) está na vez de jogar; qual o estado dos personagens em cada equipe; etc.
    b) Organização dos menus (caso a interação do usuário seja em modo texto)
    c) Organização da interface gráfica (caso a interação do usuário seja por interface gráfica – opcional)

Apêndice A

Além do código da implementação e apresentação do mesmo, deverá ser entregue um documento especificando:

1) **Passo-a-passo** para a compilação e execução do programa
2) **Conceitos de Orientação a Objetos**

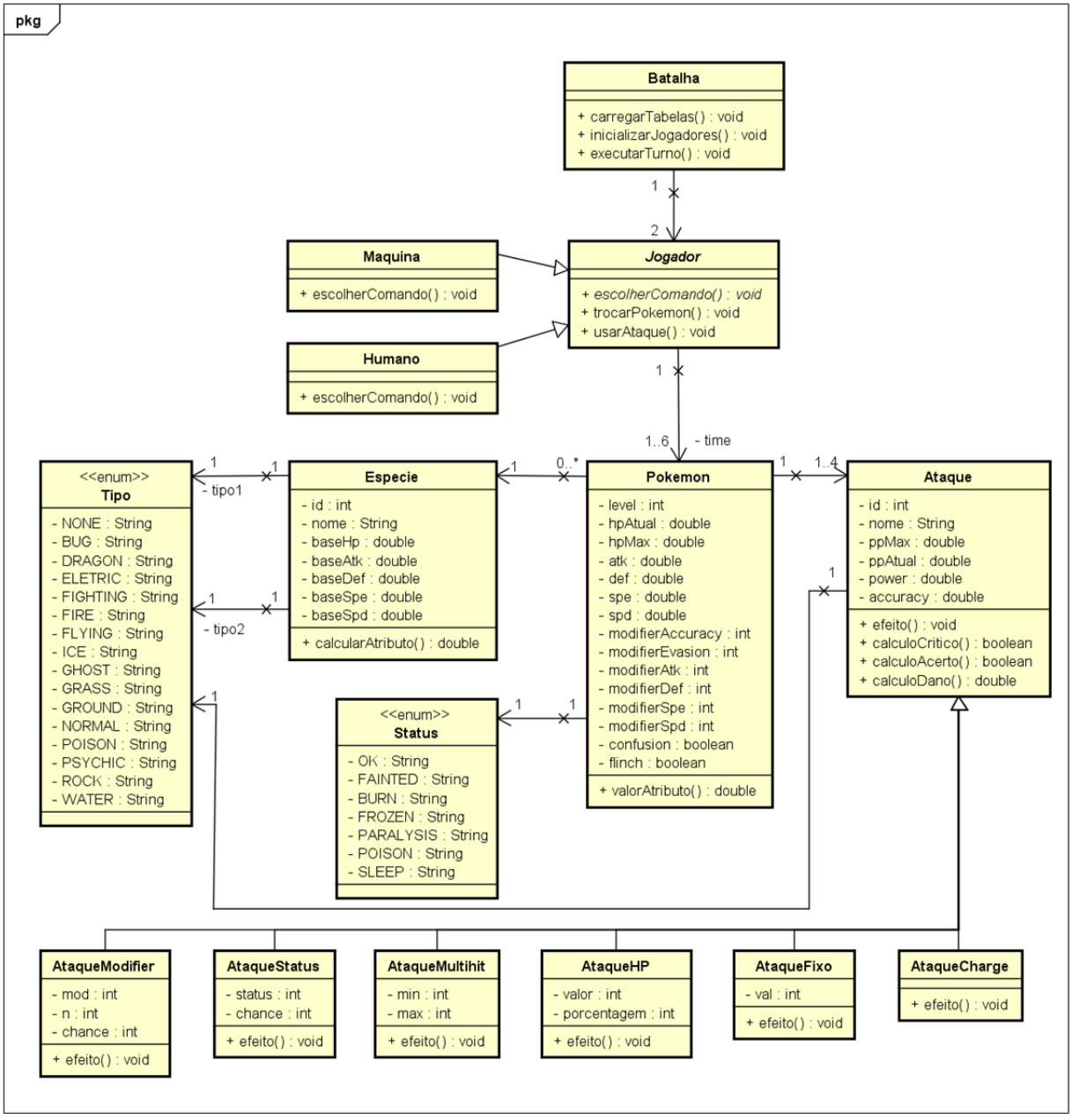
   Escolha classes utilizadas no projeto para exemplificar como foram implementados os seguintes conceitos de orientação a objetos:

   a) Encapsulamento
   b) Herança
   c) Polimorfismo
3) **Decisões de projeto**

   Explique o funcionamento geral do sistema (jogo) quanto:

   a) Classes controladoras implementadas
   b) Classes de interação com o usuário (menus de opções)
   c) Classes concretas
   d) Tratamento de exceções (se houver)
   e) Utilização de *Generics* (se houver)
   f) Organização das classes em pacotes

Apêndice B


pkg
Batalha
+ carregarTabelas() : void
+ inicializarJogadores() : void
+ executarTurno() : void
1
2
Maquina
+ escolherComando() : void
Jogador
+ escolherComando() : void
+ trocarPokemon() : void
+ usarAtaque() : void
Humano
+ escolherComando() : void
1
1..6
- time
<<enum>>
Tipo
- NONE : String
- BUG : String
- DRAGON : String
- ELETRIC : String
- FIGHTING : String
- FIRE : String
- FLYING : String
- ICE : String
- GHOST : String
- GRASS : String
- GROUND : String
- NORMAL : String
- POISON : String
- PSYCHIC : String
- ROCK : String
- WATER : String
1
1
- tipo1
1
1
- tipo2
Especie
- id : int
- nome : String
- baseHp : double
- baseAtk : double
- baseDef : double
- baseSpe : double
- baseSpd : double
+ calcularAtributo() : double
1
0..*
Pokemon
- level : int
- hpAtual : double
- hpMax : double
- atk : double
- def : double
- spe : double
- spd : double
- modifierAccuracy : int
- modifierEvasion : int
- modifierAtk : int
- modifierDef : int
- modifierSpe : int
- modifierSpd : int
- confusion : boolean
- flinch : boolean
+ valorAtributo() : double
1
1..4
Ataque
- id : int
- nome : String
- ppMax : double
- ppAtual : double
- power : double
- accuracy : double
+ efeito() : void
+ calculoCritico() : boolean
+ calculoAcerto() : boolean
+ calculoDano() : double
1
1
<<enum>>
Status
- OK : String
- FAINTED : String
- BURN : String
- FROZEN : String
- PARALYSIS : String
- POISON : String
- SLEEP : String
1
AtaqueModifier
- mod : int
- n : int
- chance : int
+ efeito() : void
AtaqueStatus
- status : int
- chance : int
+ efeito() : void
AtaqueMultihit
- min : int
- max : int
+ efeito() : void
AtaqueHP
- valor : int
- porcentagem : int
+ efeito() : void
AtaqueFixo
- val : int
+ efeito() : void
AtaqueCharge
+ efeito() : void
powered by Astah