MC322 A – Programação Orientada a Objetos
Especificação do Projeto Final

Leonardo Montecchi (Professor) Thales Eduardo Nazatto
Ângelo Renato Pazin Malaguti Leonardo de Sousas Rodrigues

2s2021

1 Introdução

Pokémon[1] é o nome de uma franquia de jogos e outras mídia, criada por Satoshi Tajiri em 1995 e atualmente propriedade da *The Pokémon Company*[2]. Ela é centrada em criaturas ficcionais chamadas "Pokémon", que os seres humanos capturam e os treinam para lutarem um contra o outro como um esporte. A franquia começou com um par de jogos lançados para o Game Boy original (*Pokémon Red & Blue*)[3] e atualmente se estende em jogos, cartas colecionáveis, série de televisão, além de filmes, mangás e brinquedos.

Figura 1: Logo da franquia "Pokémon".

Existem vários jogos baseados na franquia Pokémon, cada um com as suas especificidades em termos de regras, contexto e objetivos. Tipicamente, o jogo evolui do ponto de vista de um *treinador de pokémon*, cujo objetivo é colecionar e evoluir as criaturas. No geral, podem ser identificadas três fases ou atividades que acontecem, de alguma forma, em todos os jogos da franquia.

- *Exploração*. Na fase de exploração o jogador explora o mundo do jogo, usualmente dividido em áreas com carateristicas diferentes, com o objetivo de encontrar novos Pokémons. Dependendo do jogo, na exploração podem também ser encontrados itens, lojas, inimigos, etc.

- *Captura*. Pokémon que são encontrados na exploração do mundo podem ser *capturados*, passando então a integrar a lista de Pokémons do jogador. Dependo do jogo existem várias modalidades para capturar um

[1] `https://pt.wikipedia.org/wiki/Pokémon`
[2] `https://pt.wikipedia.org/wiki/The_Pokémon_Company`
[3] `https://pt.wikipedia.org/wiki/Pokémon_Red_&_Blue`

Pokémon, por exemplo habilidades ou cartas especiais, ou rolando dados. Em alguns jogos esta fase não é esecutada, pois o jogador controla diretamente um pokémon (por exemplo, os jogos da série *Pokemon Mystery Dungeon*[4]).

- *Combate*. Pokémons podem duelar, normalmente no contexto de um duelo entre dois treinadores. Cada pokémon possui características e habilidades diferentes, que podem também ter efeitos diferentes com base no tipo de pokémon adversário. Pokémon ganham esperiência e podem *evoluir*, ficando mais poderosos.

Figura 2: Jogo de tabuleiro "Pokémon Master Trainer" da MB[5].

Detalhes sobre as mecânicas dos jogos originais, assim como sobre as habilidades dos diferentes pokémons podem ser encontrados em sites de comunidade de *fans*, como *Bulbapedia*[6] ou a página dedicada aos pokémons no *Fandom*[7].

[4] https://pokemon.fandom.com/wiki/Pokémon_Mystery_Dungeon
[6] https://bulbapedia.bulbagarden.net/
[7] https://pokemon.fandom.com

2 Especificação

O objetivo deste projeto é **projetar e desenvolver um jogo baseado na franquia Pokémon em Java, aplicando os princípios de programação orientada a objetos**. O jogo será uma *adaptação simplificada* das regras presentes nos jogos originais.

Premissas. Para facilitar a implementação, podem ser consideradas as seguintes simplificações:

- A saída do jogo será de forma textual, organizada como de preferência da equipe.
- A entrada será recebida pelo teclado.
- A atualização da tela será feita de forma cíclica. Para tal propósito, pode-se considerar como base o esqueleto de código mostrado na Figura 3.

```java
public class Runner {

    public static void main(String[] args) {

        Game g = new Game();
        g.start();
    }
}
```

```java
public class Game {

    private boolean exitSelected;

    public void start() {
        exitSelected = false;
        System.out.println("Game started!");

        while(!exitSelected) {
            drawBoard();
            readInput();
            updateBoard();
        }
        System.out.println("Game terminated. Bye!");
    }
}
```

Figura 3: Esqueleto de código para o ciclo principal do jogo.

Funcionalidades. O jogo deve implementar as seguintes funcionalidades:

1. O jogador explora um mundo constituído por *ilhas*, podendo explorar uma ilha por vez. Por simplicidade, as ilhas serão de forma retangular.

2. Ilhas do mesmo plano são conectadas por *pontes*, que permitem ao jogador de se movimentar entra elas. Pontes podem ser encontradas apenas nas bordas das ilhas.

3. Ilhas podem também ser conectadas verticalmente, por *elevadores*. Elevadores permitem ao jogador escolher de quantos andares subir (ou descer), entre os andares disponíveis. Elevadores podem ser encontrados em qualquer posição da ilha (isto é, não apenas nas bordas).

4. *Portais* permitem ao jogador se movimentar para qualquer ilha *entre as que foram já visidadas* na mesma partida. Portais podem também se encontrados em qualquer posição da ilha.

5. O jogador se movimenta rolando dois dados de 6 faces, e deve obrigatoriamente usar todos os movimentos disponíveis no turno, antes de executar qualquer outra ação. Usar uma ponte, um elevador ou um portal requere um ponto de movimento.

6. *Pokémons*. Os pokémons são criaturas com carateristicas e habilidades diferentes.

    - Todos eles possuem um nome, uma quantidade de pontos de vida (ou Hit Points, HP), uma quantidade de pontos de ataque e uma quantidade de pontos de defesa.
    - Cada pokémon possui até 2 tipos, entre *Fire*, *Water*, *Grass*, *Electric*, *Psychic*, e *Dragon*.
    - Cada pokémon pode possuir habilidades (detalhadas mais para frente)

7. Pokémons podem ser encontrados pelo jogador nas várias ilhas. Quando um jogador entra em uma ilha onde existem pokémons, cada pokémon rola três dados de *quatro* faces: o primeiro indica a distância máxima de captura d, enquanto a soma dos outros dois indica a dificuldade da captura k.

8. Ilhas também possuem um tipo entre *Fire*, *Water*, *Grass*, *Electric*, *Psychic*, e *Dragon*. Enquanto um pokémon estiver em uma ilha do mesmo tipo dele, ganha um bônus de 15% a todas as estatisticas (HP, ataque e defesa). Isso se aplica também aos pokémons capturados pelo jogador.

9. O turno do jogador é dividido em duas fases: movimento e ação. Como comentado antes, o jogaor deve efetuar todos os movimentos disponíveis antes de passar à fase de ação. Na fase de ação pode efetuar as seguintes ações.

    - *Escolher* um pokémon, entre aqueles disponíveis para ele (isto é, os que ele conseguiu capturar). O pokémon escolhido é o que fica "ativo" e que será envolvido nos combates quando necessário.
    - *Usar um item* do inventário.
    - *Atacar* um pokémon presente na ilha.
    - *Tentar a captura* de um pokémon presente na ilha.

    Não existe limite ao número de ações que podem ser feitas no mesmo turno.

10. *Captura*. Uma tentativa de captura pode ser efetuada apenas se a distância entre o jogador e o pokemon for menor ou igual da distância de captura d (veja acima). Para capturar o pokémon, o jogador rola dois dados de seis faces: se o valor for maior de $d + k$, então o pokémon é capturado pelo jogador. Uma vez capturado, o pokémon fica disponível entre as criaturas do jogador. Se o jogador falir a captura do mesmo pokémon duas vezes ou mais, o pokpemon vira hostil, e o jogo entra na modalidade de combate.

11. *Combate*. A modalidade de combate pode ser ativada: 1) pelo jogador, se ele selecionar a ação de atacar um pokémon; ou 2) pelo computador, se um pokémon virar hostil. Quem inciar o combate tem a vantegem de iniciativa, ou seja, é o primeiro a efetuar ataques.

    - A modalidade de combate evolui em turnos, alternando entre os dois *pokémons* envolvidos no combate. O combate continua até que um dos dois pokémons *desmaiar* (isto é, atingir 0 ou menos HP).
    - No próprio turno de combate, um pokémon pode escolher fazer um *ataque base* ou *usar uma habilidade*. Habilidades são detalhadas mais para frente.
    - A menos de eventuais efeitos de habilidades, o dano de um ataque base é dado pelos pontos de ataque do atacante, menos os pontos de defesa do defensor. Qualquer ataque causa no mínimo um ponto de

dano ao adversário. O valor de dano do ataque é subtraído aos HPs do defensor.

12. Se o pokémon controlado pelo computador desmaiar, este é capturado pelo jogador. Se o pokémon controlado pelo jogador desmaiar, o combate continua escolhendo um outro pokémon entre os que não estão desmaiados. Se o jogador não houver pokémons utilizáveis, o jogo termina com a derrota do jogador.

13. A cada turno do jogador *fora do combate*, todos os pokémons do jogador recuperam 1 HP.

14. Nas ilhas o jogador pode encontrar também *itens* usáveis, que podem ser coletados para uso posterior. É suficiente o jogador passar por cima do item para coletá-lo. Existem tipos diferentes de itens:

    - *Fruto*: Podem ser usados para restaurar os HPs de um pokémon.
    - *Technical Record (TR)*: Pode ser usado para ensinar uma nova habilidade a um dos pokémons capturados. Cada TR é específico pra uma determinada habilidade. Isto é, o jogador *não* escolhe a habilidade.

    Deve ser possível estender o código para contemplar mais itens no futuro.

15. Cada *habilidade* possui um nome e uma lista de tipos de pokémon que podem aprender (e então usar) a habilidade. Os efeitos de uma habiliadede podem ser de dois tipos gerais: *passivos*, que se aplicam apenas por possuir a habilidade e modificam as estatisticas do pokémon ou o dano recebido; e *ativos*, que se aplicam na hora de usar de fato a habilidade.

16. Alguns exemplos de habilidade são detalhados a seguir. O jogo deve ser extensível com novas habilidades.

    - *Flame Body* (tipos: *Fire* or *Dragon*). Quando o pokémon receber dano tem uma chance do 30% de causar 5 pontos de dano ao atacante. Não funciona contra pokémons do tipo *Water* ou *Fire*.
    - *Berserk* (tipos: *Electric*, *Dragon*). Enquanto o pokemon tiver menos da metade dos HPs, os seus pontos de ataco são dobrados.
    - *Huge Power* (todos os tipos). Pode ser ativada para dobrar ataco e defesa por um turno de ataco. Porém, fazendo isso, o pokémon perde 5 HP.

- *Healer* (tipos: *Water*, *Grass*, *Psychic*). Pode ser ativada para curar o pokémon. Tem uma chance de 70% de curar 10% dos HPs do pokémon, 20% de curar 20% dos HPs do pokémon, e 10% de não funcionar (curar 0 HPs)
- *Intimidate* (tipos: *Fire*, *Dragon*, *Psychic*). Reduz todos os danos recebidos por outros pokémons deo 50%, exceto danos recebidos de outros pokémon *Psychic*.

17. Os pokémons que não são controlados pelo jogador devem ser controlados automaticamente pelo computador. Como simplificação, considere que os pokémons controlados pelo computador atuam de forma aleatória, escolhendo aleatoriamente entre as ações disponíveis.

18. A cada turno, o sistema deve mostrar a situação atual do jogo e listar as ações possíveis para o jogador. Caso o jogador queira, o sistema deve mostrar o detalhamento dos pokémons capturados com suas características e estatisticas atuais.

19. O jogo termina quando o jogador tiver coletado todos os pokémons presentes nas ilhas (vitória), ou quando durante um combate todos os pokémons tiverem desmaiado (derrota).

20. No começo de cada partida o jogador escolhe se 1) jogar um dos níveis (mundos) pré-definidos, ou 2) jogar um nível (mundo) gerado aleatóriamente. A definição de um mundo inclui quais ilhas existem e como estão conectadas, quais pokémons são encontrados em cada ilha (ou a probabilidade de encontrar pokémons específicos), etc.

21. O código deve ser extensível para permitir a introdução de novos mundos e/ou novas formas de criação de mundos. Por exemplo, deve ser possível estender o jogo para ler mundos a partir de arquivos de texto. *Sugestão*: Uma forma de fazer isso é usar os padrões de projeto *Factory* ou *Factory Method*[8].

Comportamentos que não foram especificados nesta lista podem ser decididos pelas equipes ou colocados como parâmetros de configuração do jogo. Esco-

[8]Na hora da disponibilização deste texto não teremos visto padrões de projeto ainda. Padrões de projetos são formas estabelecidas (padrões) de organizar as classes ao fim de resolver um problema comunemente encontrado em sistemas diferentes.

lhas deste tipo devem ser documentadas no relatório do projeto e/ou com comentários no código.

Funcionalidades Adicionais. É permitido a inclusão de funcionalidades adicionais (ex: interface gráfica, limite de tempo para as jogadas, organização cliente/servidor para dois jogadores), desde que as funcionalidades básicas listadas acima sejam realizadas corretamente.

Relatório. Junto com o código fonte deve ser produzido um breve relatório que explique as funcionalidades implementadas e as escolhas tomadas durante o desenvolvimento. O propósito deste relatório é explicar porque foi decidido organizar o código de uma certa forma, quais problemas foram encontrados, como eles foram resolvidos, e eventualmente detalhar funcionalidades adicionais que foram incluídas. O relatório deve conter um diagrama UML das *principais* classes implementadas e as relações entre elas. É possível dividir o diagrama em várias partes para simplificar a explicação.

O nível de detalhamento do relatório é deixado ao bom senso da equipe. Observe que relatório serve para reduzir o esforço necessário para entender o que foi feito. Um texto de 50 páginas ou um diagrama com todos os detalhes de todas as classes (mesma informação que já está no código) não ajuda muito neste sentido. Não é recomendável usar ferramentas de extração automática de diagaramas a partir do código, pois tipicamente trazem muitos detalhes irrelevantes e ignoram outros necessários (ex: diferença entre associação e composição).

3 Equipe

O projeto deve ser desenvolvido **em equipe de 2-3 alunos**. Os integrantes de cada equipe devem ser sinalizados na plataforma Google Classroom, por meio da atividade "Definição das Equipes do Projeto". Eventuais alterações das equipes devem ser discutidas com o professor. Fiquem à vontade para usar os canais da disciplina (Classroom, Discord) para achar integrantes caso seja necessário.

É recomendado o uso da ferramenta de controle de versão **git**[9] (ou equivalente) para facilitar a integração do código desenvolvido por cada integrante. Existem plataformas que permitem criar e manter repositórios git online, como

o GitHub ou o GitLab. A Unicamp também mantém uma instalação institucional do GitLab[10]. Caso escolham usar umas destas plataformas, **é recomendado deixar o repositório como privado**, pelo menos até a data de entrega do projeto.

4 Submissão

O projeto deve ser submetido no link de entrega no Google Classroom da disciplina, junto com um breve relatório como descrito acima. O relatório deve descrever as escolhas de implementação e qualquer informação necessária para executar a aplicação. **A submissão deve ser feita em formato de arquivo compactado (zip/tar.gz/rar) e o nome do arquivo deve estar no formato {NomeGrupo}_Projeto**, onde {NomeGrupo} deve ser substituído pelo nome da própria equipe.

- O arquivo deve conter:
  1. O código fonte do projeto.
  2. O relatório em formato pdf.
- Quando: Até dia **5 de dezembro de 2021**.

5 Avaliação

A nota base do projeto será entre 0 e 10. Faz parte da avaliação decidir quais classes devem ser criadas e com quais responsabilidades. Em particular, o código será avaliado de acordo com os seguintes aspectos:

1. Definição de classes (30%) — Ex: classes, métodos, responsabilidades.
2. Aplicação de princípios de POO (30%) — Ex: *encapsulamento*, aplicação de *herança*, uso de *polimorfismo*.
3. Funcionalidades implementadas (30%) — Completude e corretude das funcionalidades previstas[11].
4. Relatório (10%)

Apenas no caso em que todas as funcionalidades requeridas forem implementadas corretamente será considerado o acréscimo de até um (1) ponto para funcionalidades adicionais implementadas. Isto é, a nota final do projeto pode

[9] `https://guides.github.com/introduction/git-handbook/`

[10] `https://gitlab.unicamp.br`

[11] Ignorar completamente algumas das funcionalidades previstas pode também afetar as outras dimenções de avaliação, pois não permite avaliar quais escolhas de projeto seriam tomadas para a sua inclusão.

atingir até 11 pontos. Funcionalidade "implementada corretamente" significa 1) funcionando corretamente e 2) implementada de acordo com os princípios de programação orientada a objetos.

6 Dicas

O objetivo do projeto é avaliar a capacidade da equipe de projetar e implementar um sistema complexo de acordo com os princípios da POO. Lembrem-se que 60% da nota do projeto será baseada na correta aplicação dos conceitos de POO, 10% no relatório que descreve a solução projetada, e *apenas 30%* na corretude das funcionalidades implementadas, portanto:

- Começem a pensar nas classes e seus métodos desde já, refinando a solução até a data de entrega final.
- Abordem o projeto como um todo desde o começo, e *não* focando apenas em alguma funcionalidade. Definam a maioria de classes e métodos que puderem *antes* de começar codar as implementações.
- Para simplificar, podem criar inicialmente métodos com implementações simplificadas (ex., retornando sempre *true* ou com efeitos parciais), que depois serão substituidos com imlementações "reais" mais detalhadas.
- Tentem sempre antecipar possíveis mudanças nas especificações. Isso vai também servir caso, depois, percebam que algo deve ser modificado (ex: por que esquceram considerar algo).

7 Fraudes

Qualquer tipo de fraude acarretará em nota final zero para todos os envolvidos. Exemplos de fraudes são:

- compartilhar código não trivial entre equipes diferentes;
- copiar código não trivial da Internet;
- comprar implementações prontas.