



Trabalho prático

Instruções

1. O trabalho deve ser implementado utilizando a linguagem Java.
 - 1.1. Deve conter o mínimo de classes do modelo especificado na Figura 2, bem como as classes concretas e as classes de controladoras. Novas classes podem ser criadas conforme a necessidade.
 - 1.2. Poderá ser desenvolvido individualmente ou em equipe de, **no máximo**, duas pessoas.
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 23:55hs do dia **03/12/2018**. Coloque seu nome e R.A. como nome do arquivo compactado. Exemplo: Nome123456.rar (sem espaço). Se o trabalho for feito em dupla, separe os nomes e respectivos R.A.'s por um “_” (underline). Exemplo: FulanoTal23455_SicranoOutro67890.rar (sem espaço);
3. Devem ser entregues, via Moodle, os seguintes artefatos compactados em um arquivo:
 - 3.1. Código fonte do programa;
 - 3.2. Bibliotecas de terceiros (Jar) para o funcionamento correto do programa, se forem utilizadas.
 - 3.3. Documento que descreve as classes definidas no trabalho – ver Apêndice 1.
4. O trabalho deve ser apresentado a partir do dia **03/12/2018** em ordem alfabética constante na chamada.
 - 4.1. Embora o trabalho possa ser feito em dupla, isto não garante mesma nota para os integrantes da equipe.
 - 4.2. A apresentação deve abranger:
 - a) as decisões de projeto do sistema;
 - b) implementação das classes de projeto.

Objetivo do jogo

Aplicar os conceitos de orientação a objetos na programação de um sistema de software que simula um jogo que se baseia no Super Mario World.

Descrição do Trabalho

O jogo é uma simplificação do Super Mario World, no qual o personagem Mario tem a missão de resgatar princesa Peach e durante sua jornada, ele enfrenta vários inimigos.

O mundo no qual todos os personagens estão contidos é representado por uma matriz 10x10 em que cada posição visitada pelo Mário pode conter inimigos (Goomba, Piranha Plant, Boo) ou itens bônus (moedas, Fire Plant). A Figura 1 ilustra um exemplo da instância de uma partida.

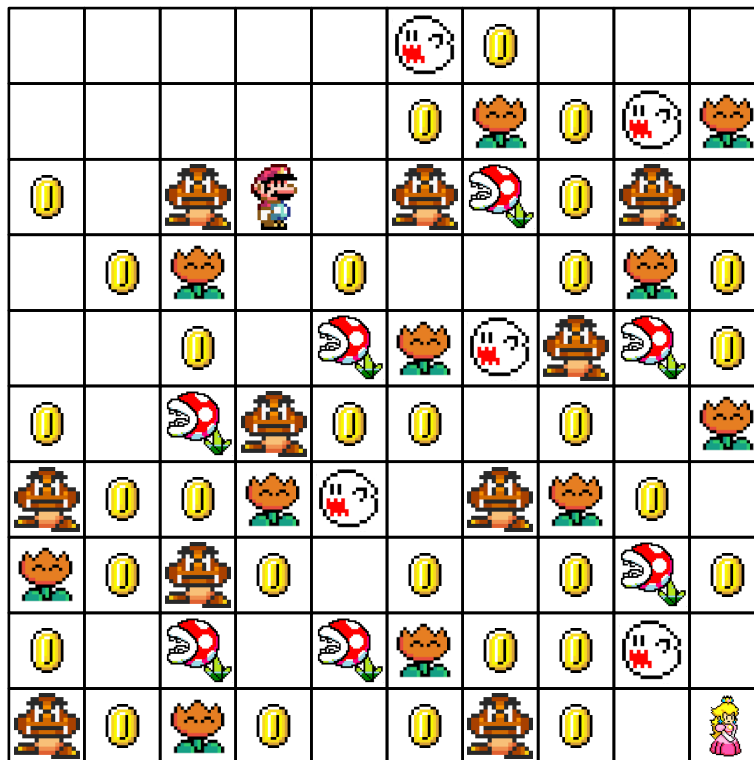


Figura 1: Exemplo de instância do jogo

O desenvolvimento do trabalho contempla duas etapas:

1. Preparação do jogo;
2. Implementação das regras do jogo.

A preparação do jogo consiste na instanciação do mundo do Mario (a matriz 10x10) e a distribuição dos inimigos e itens bônus pelo mapa. Além disso, é realizada a inicialização do estado do Mario.

A seguir uma explicação sobre as duas etapas de desenvolvimento.

Etapas 1: Preparação do jogo

Criar uma matriz 10 x 10 e espalhar inimigos por coordenada aleatoriamente (utilize a função random em Java).




Os inimigos devem ser espalhados conforme determinado:

- 10 goombas
- 7 piranhas plant
- 5 boos

Isso significa que o valor de coordenada de cada inimigo deve ser definida aleatoriamente, de forma não repetida.

Cada inimigo tem uma característica, conforme descrito na tabela a seguir:





		
Goomba	Piranha Plant	Boo
Ao estar na mesma posição e o Mario não conseguir derrotá-lo, o Mario perde a vida	Lança chamas com distância de uma posição, nas quatro direções (esquerda, direita, cima e baixo). Portanto, o Mario perde a vida caso esteja em uma posição vizinha.	Ao estar na mesma posição, o Mario perde a vida. O Boo não pode ser derrotado.

Além dos inimigos, os itens bônus devem ser distribuídos na matriz, também de forma aleatória:

- 30 moedas;
- 10 fire plants.

Isso significa que o valor de coordenada de cada bônus deve ser definida aleatoriamente, de forma não repetida.

A seguir, uma descrição para cada item bônus:

	
Moeda	Fire Plant
A cada 10 moedas recolhidas, o Mario ganha 1 vida.	Dá poder ao Mario de lançar chamas à distância de duas posições. Desse modo, o inimigo deve estar na primeira ou na segunda posição para ser atingido.

O objetivo do jogo é encontrar a princesa Peach, que está na última posição da matriz. Considerando que os índices da matriz iniciam em (0,0) a princesa Peach estará na posição (9,9). O Mario começa na posição (0,0).

No início do jogo, o Mario tem os atributos:

- `quantidadeMoedas = 0;`
- `quantidadeVidas = 3;`
- `possuiPoderFogo = false.`

Etapa 2: Regras do jogo

- O Mario pode executar as ações ANDAR, PULAR, DISPARAR, para uma das direções ESQUERDA, DIREITA, CIMA e BAIXO.
- Ele pode ver o que está em cada uma das quatro direções imediatas (adjacentes).
 - Exemplo:
 - Se o Mario está na posição (3,4), ele pode enxergar o que está na posição (3,3), (2,4), (3,5), (4,4).



- Existe uma distância segura de posições que o Mario pode estar em relação a posição de seus inimigos:
 - Se for o Goomba ou o Boo, o Mario pode ficar em qualquer posição adjacente;
 - Se for a Piranha Plant, o Mario deve ficar a uma distância de uma posição longe, nas quatro direções (isto porque, originalmente, a Piranha Plant lança bolas de fogo, então se o Mario ficar em posição adjacente, ele perde a vida).
- A partir da posição corrente ele pode:
 - Se mover para cada uma das 4 posições caso esteja livre (sem inimigo) ou pode se mover para a posição em que contenha um item bônus ganhando esse item; ou
 - Pular sobre o inimigo para tentar derrotá-lo e retirá-lo da matriz; ou
 - Disparar uma bola de fogo sobre o inimigo, caso esteja com o poder da Fire Plant.
- A cada 10 moedas que ele acumula durante sua jornada, ele ganha mais uma vida. As moedas são acumuladas quando o Mario ocupa a posição onde estava a moeda, que é retirada da matriz.
- A Fire Plant não pode ser acumulada. Portanto, toda a vez que o Mario ocupa a posição de uma Fire Plant, ele ganha o poder de lançar bolas de fogo e a Fire Plant é retirada da matriz. Porém, se o Mario já estiver anteriormente com o poder da Fire Plant, o item é apenas retirado da matriz.
- O usuário pode decidir que o Mario pule para uma posição onde está um objeto do tipo Inimigo, que oferece uma resistência:
 - O Goomba têm resistência de 50%, isto é, ao pular sobre o Goomba para derrotá-lo, o Mario tem 50% de acertar, caso contrário, perde o poder da *Fire plant* e se não o tiver, perde a vida.
 - A Piranha Plant tem resistência de 75%, isto é, ao pular sobre ela para derrotá-la, o Mario tem 25% de acertar, caso contrário, perde o poder da *Fire plant* e se não o tiver, perde a vida.
 - O Boo tem resistência de 100%, isto é, ao pular sobre ele para derrotá-lo, o Mario perde o poder da Fire Plant e se não o tiver, perde a vida.
 - Se o Mario conseguir pular sobre o inimigo e conseguir retirá-lo ele ocupa a posição na qual o inimigo estava.
- Quando o Mario tem o poder da Fire Plant ele pode atirar em qualquer uma das quatro direções e a bola de fogo tem o alcance de, no máximo, duas posições. Desse modo, o inimigo deve estar na posição adjacente ao Mario num raio de até duas posições para ser atingido. Se a bola de fogo atingiu um inimigo, ela não prossegue para a posição seguinte.
- Além disso, os inimigos possuem resistência aos tiros da bola de fogo. Portanto, quando o Mario está com o poder da Fire Plant e atira na direção do inimigo, caso ele seja:
 - O Goomba, é derrotado (zero % de resistência);
 - A Piranha Plant, não é derrotado ao primeiro tiro, tendo resistência de 25%, sendo necessários mais tiros para tentar derrotá-la.
 - O Boo, não é derrotado. Ele tem resistência de 100%.
- Quando o Mario perde uma vida, o jogo pode continuar a partir da última posição em que



ele estava anteriormente. A matriz permanece com o mesmo estado anterior.

- O jogo termina quando:
 - O Mario consegue alcançar a posição (9,9) que é onde está a princesa Peach, vencendo o jogo;
 - O Mario perde todas as suas vidas, perdendo o jogo.

Orientação a objetos

O diagrama de classes ilustrado na Figura 2 representa uma sugestão de classes que podem ser implementados no projeto.

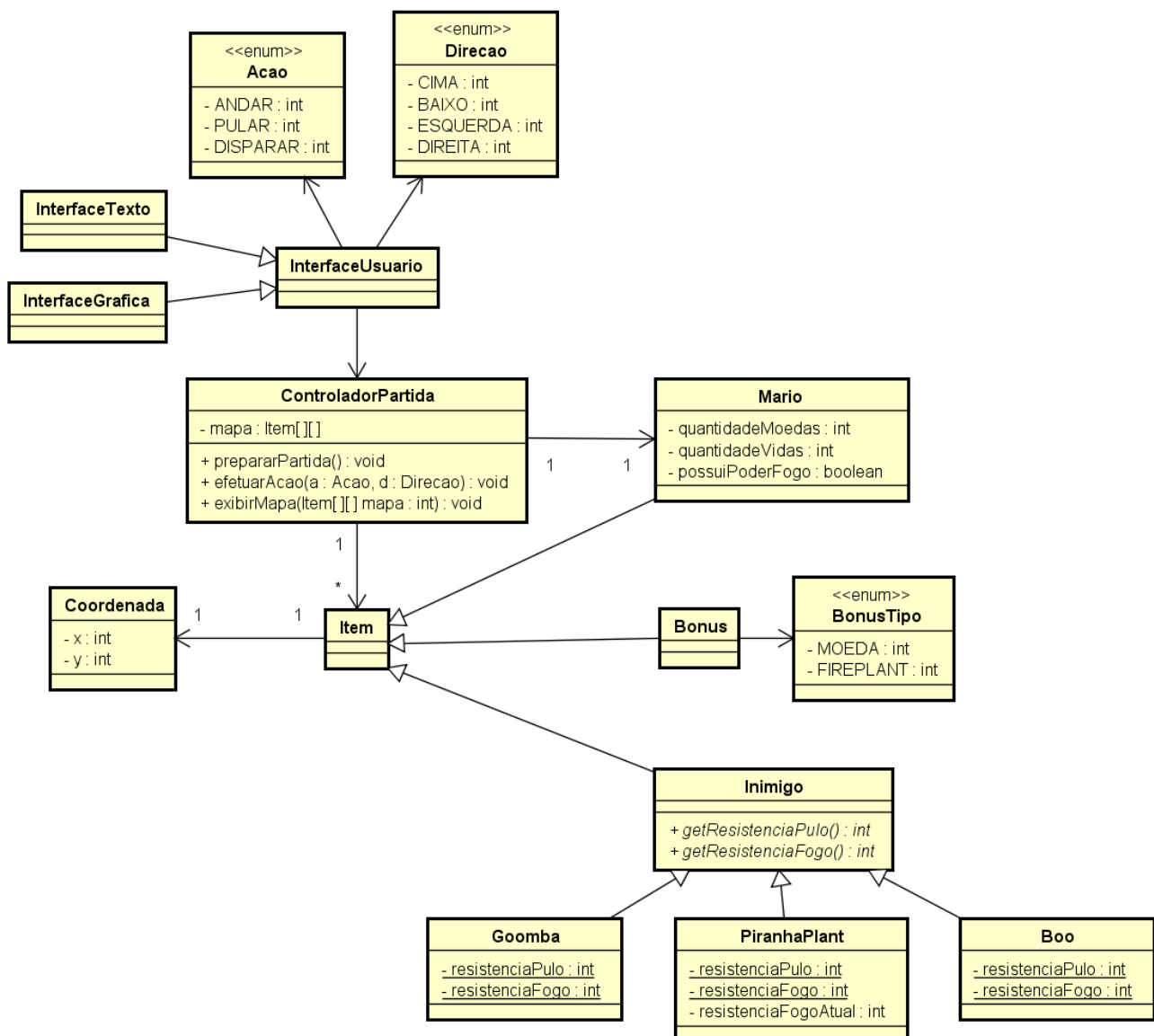


Figura 2: Diagrama de classes

O Mario, os objetos do tipo Inimigo e Bonus são especificações da classe Item, que possui um par



(x,y) de Coordenada. O atributo **mapa** da classe Controlador representa a matriz que armazena o Mario, os inimigos, os bônus e a princesa Peach.

O método **prepararPartida()** da classe Controlador é responsável por inicializar a posição de coordenada do Mario em (0,0), reservar a posição da coordenada (9,9) para a princesa Peach e distribuir os inimigos e itens bônus conforme as regras definidas anteriormente.

Uma classe que realiza a interface com o usuário (representada pela classe InterfaceUsuario) é responsável por receber um evento do usuário, composta por uma ação e uma direção no mapa, que consequentemente atualizará a coordenada e estado do Mario.

O método **exibirMapa()** da classe Controlador deve exibir o estado do mapa após a execução de uma ação.

A seguir são apresentados três exemplos de jogadas.

Exemplo 1: Andar

O usuário pode selecionar a ação ANDAR e a direção ESQUERDA na InterfaceUsuario. Essas informações serão passadas ao método **efetuarAcao (ANDAR, ESQUERDA)** da classe Controlador, que atualizará a posição do Mario de acordo com a nova direção recebida da InterfaceUsuario, e atualizará o estado do Mario de acordo com o item que for encontrado na nova posição.

Se na posição à esquerda houver uma Moeda, o atributo **quantidadeMoedas** do Mario será incrementada, se estiver vazia, nada acontece.

Se na posição à esquerda houver uma Fire Plant, o atributo booleano **possuiPoderFogo** do Mario é alterado para *true* e a Ação de DISPARAR é habilitada. A posição ocupada pela Fire Plant é atualizada para vazio, ou a Fire Plant é removida da posição em que estava.

Se na posição à esquerda houver um inimigo, o Mario perde uma vida, isto é, o atributo **quantidadeVidas** do Mario é decrementada.

Se na posição à esquerda estiver vazio, porém na posição adjacente estiver o inimigo Piranha Plant, o Mario perde uma vida, isto é, o atributo **quantidadeVidas** do Mario é decrementada. Isto porque a Piranha Plant lança fogo com alcance de uma posição de distância nas quatro direções. Entretanto, se o Mario estiver com o poder da Fire Plant, o atributo **possuiPoderFogo** é atualizado para *false* e ele não perde a vida.

Exemplo 2: Pular

O usuário pode selecionar a ação PULAR e a direção DIREITA na InterfaceUsuario. Essas informações serão passadas ao método **efetuarAcao (PULAR, DIREITA)** da classe Controlador, que atualizará a posição do Mario de acordo com a nova direção recebida da InterfaceUsuario, e atualizará o estado do Mario de acordo com o item que for encontrado na nova posição.

Se na posição à direita houver uma Moeda, o atributo **quantidadeMoedas** do Mario será incrementada, se estiver vazia, nada acontece.

Se na posição à direita houver uma *Fire plant*, o atributo booleano **possuiPoderFogo** do Mario é alterado para *true* e a ação de DISPARAR é habilitada.

Se na posição à direita houver um inimigo, as restrições de resistência ao pulo dele devem ser



verificadas de acordo com as regras para atualizar o estado do Mario.

Se na posição à direita estiver a princesa Peach, o usuário vence o jogo.

Observação: O Mario pode pular uma posição por vez. Então ele só pode pular em uma Piranha Plant caso tenha “sobrevivido” na jogada anterior, isto é, o Mario está em uma posição adjacente a uma Piranha Plant e perdeu seu poder de disparar bolas de fogo. Logo, o Mario pode tentar pular sobre a Piranha Plant para tentar removê-la do mapa.

Exemplo 3: Disparar

O usuário pode selecionar a ação DISPARAR somente se o atributo **possuiPoderFogo** do Mario for igual a *true*. Caso verdadeiro, o usuário deve selecionar a direção. Supondo que seja possível efetuar o disparo, e a direção BAIXO foi selecionada, temos que a InterfaceUsuario deverá invocar o método **efetuarAcao (DISPARAR, BAIXO)** que atualizará o mapa dependendo do Item localizado que é atingido pela bola de fogo.

Se em algumas das duas próximas posições houver um inimigo, as restrições de resistência ao fogo dele devem ser verificadas para definir se o Mario conseguiu derrotar esse inimigo, removendo-o do mapa.

Se na posição adjacente ao Mario já houver um inimigo, a posição seguinte não é verificada.

Por exemplo, suponha que o Mario esteja na posição (3,4), conforme a Figura 1:

- Suponha que o usuário selecione (DISPARAR, BAIXO). Como nas posições (4,4) e (5,4) não há inimigo, nada acontece.
- Suponha que o usuário selecione (DISPARAR, DIREITA). Como na (3,5) não há inimigo, é verificada a posição (3,6). Como há um inimigo na posição (3,6) é aplicado as regras de resistência ao fogo do inimigo. Caso o inimigo seja derrotado, o mapa deve ser atualizado, tornando a posição vazia – sem nenhum item.

Pontos importantes a serem considerados durante a realização do trabalho:

- Originalidade da implementação das classes concretas e classes de projeto.
- Implementação das regras de forma correta
- Implementação dos conceitos de Programação Orientada a Objetos
 - Encapsulamento
 - Herança
 - Polimorfismo
 - Implemente o trabalho o mais orientado a objetos possível.
- Organização das classes em pacotes (seguindo a convenção de Java)
- Tratamento de exceções
- Implementação ou uso de classes de interfaces (opcional)
- Implementação ou uso de classes genéricas (opcional)
- Documentação do sistema
- Usabilidade do sistema:
 - Clareza na execução das jogadas: qual o estado do jogo antes e após cada jogada, o estados dos objetos, opções de jogadas intuitivas;
- Organização dos menus (caso a interação do usuário seja em modo texto)



- Organização da interface gráfica (caso a interação do usuário seja por interface gráfica – opcional)

Bônus

- Implementação de um padrão de projeto catalogado (Padrões GoF, padrões GRASP entre outros)
- Além da implementação, o padrão deve constar na documentação (diagrama de classes de projeto).

Apêndice 1

Documentação do sistema

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

1. **Passo-a-passo** para a compilação e execução do programa.
2. **Funcionamento geral** do programa
 - a) Como as entradas devem ser fornecidas
 - b) Qual o processamento realizado a partir das entradas
 - c) Qual(is) a(s) saída(s) esperada(s).
3. **Conceitos de Orientação a Objetos**

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

 - Encapsulamento
 - Herança
 - Polimorfismo
4. **Decisões de projeto**
 - a) Definição de novas classes

Explique o funcionamento geral do sistema (jogo) quanto:

 - Classes controladoras implementadas
 - Classes de interação com o usuário (menus de opções)
 - Tratamento de exceções (se houver)
 - Utilização de *Generics* (se houver)
 - Organização das classes em pacotes (coesão e acoplamento)
 - Implementação de padrões de projeto (se houver)
 - b) Diagrama de classes de projeto (opcional)

Apresente o diagrama de classes de projeto do sistema.