Manual do Sistema - Space Invaders

Heitor Tanoue de Mello - NUSP 12547260
Prof. Dr. Robson Cordeiro

1 Resumo

O código foi dividido em pastas que compartilham algumas semelhanças, são eles:

- *ElementosDoSistema*: entidades do jogo;
- *Engine*: responsável pelo funcionamento do jogo, como movimentação e tiros;
- *InterfaceGrafica*: parte gráfica do jogo.
- *SpaceInvaders*: pacote do aplicativo principal, onde o jogo é inicializado.
- *Imagens*: pasta com todas as imagens do jogo;
- *Outros*: fontes, áudios, etc...

Nesse documento, serão explicados os pacotes ElementosDoSistema, Engine e InterfaceGrafica, já que ele contém toda a lógica do jogo.

2 Elementos do Sistema

Responsável por implementar as entidades que compõe o jogo, suas características e habilidades. A estrutura de arquivos desse pacote está representada abaixo:

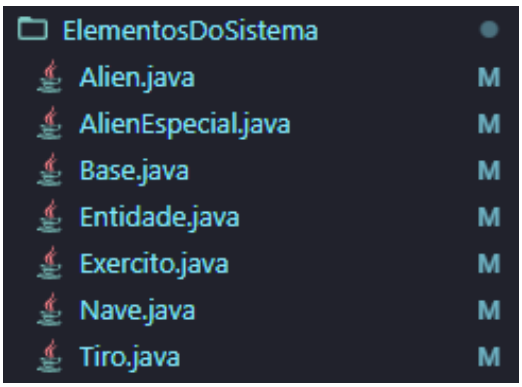


Figura 1: Estrutura de arquivos do pacote ElementosDoSistema.

2.0.1 Tuple

Classe que auxilia a representação de coordenadas e vetores de dois valores. Nela, existem valores **x** e **y** , com métodos: *setters* e *getters*, comparativos e aditivos. Todas as coordenadas de posições e velocidades do jogo são implementadas com entidades dessa classe.

2.1 Entidades

Todas classes contidas na pasta Entidades estendem a classe-mãe Entidade. Ela define uma **imagem** (do tipo Image do JavaFX) para o objeto, as coordenadas de posição na tela (**pos**), sua velocidade (**vel**) bem como sua **largura** e **altura**.

Além disso, são implementados os métodos comuns entre todos os elementos do jogo. Aqui estão destacados os mais importantes:

- *deslocar*: desloca a Entidade de acordo com sua velocidade;
- *colisaoEntidade*: detecta colisão entre duas Entidades;
- *colisaoTela*: detecta colisão com as bordas da tela;
- *colisaoTelaComVelocidade*: detecta colisão com as bordas da tela, levando a velocidade da Entidade em consideração;
- *desenhar*: desenha a Entidade no canvas da tela, sendo um método "cru"para impressão pois cada classe que herda Entidade deve implementar sua própria função *imprimir()*, que será utilizada definitivamente (Entidade implementa a interface Imprimível, que adiciona a função *public void imprimir(GraphicsContext gc)*);
- *setDimensoes*: define a **imagem** da Entidade, bem como suas dimensões.

2.1.1 Alien

Estende a classe Entidade e tem os atributos adicionais:

- **vivo**: valor booleano que indica se o Alien está vivo;
- **tiro**: guarda uma entidade da classe Tiro quando o Alien atira;
- **tipo**: valor inteiro (de 1 a 4) que corresponde ao tipo do Alien;

O **tipo** do Alien é passado em seu construtor e a quantidade de pontos fornecida ao jogador, bem como sua **imagem** dependem desse atributo. As imagens utilizadas nos Aliens foram baseadas no jogo original, mas modificadas para deixar o visual mais moderno.

(a) Alien 01 (b) Alien 02 (c) Alien 03 (d) Alien especial

Figura 2: *Sprites* desenvolvidos para o jogo

A classe AlienEspecial (*tipo* == 4) apenas estende a classe Alien e adiciona um tipo de movimentação diferente: o movimento é feito da direita da tela para a esquerda, quando ele passa da borda esquerda, teleporta para borda direita novamente. Caso ele seja destruído, o revive após 10 segundos.

2.1.2 Exército

Classe responsável por ordenar o movimento e tiros dos Aliens. Os Aliens são armazenados na forma de uma matriz 5×11, por meio do atributo **exercito**. Além disso, possui uma Nave de referência, utilizada para posicionar os tiros guiados.

Os métodos mais importantes são:

- *mobilizarExercito()*: adiciona um Alien para cada posição da matriz **exercito**, levando em consideração a linha da matriz para definir seu **tipo**;
- *moverExercito()*: move o Exército, caso o Alien mais à direita/esquerda colida com a tela, muda a velocidade para baixo e anda um pouco, depois volta a andar para a outra direção. Para que isso aconteça, o atributo **_assistForDown** é utilizado, salvando a coordenada Y do Exército no inicio da descida e vendo a diferença com a posição atual. **velocidadeSalva** armazena o valor X da velocidade enquanto o Exército executa a descida, para que ela volte depois;
- *atirar()*: caso o número máximo de Tiros ainda não tenha sido atingido (**numTirosMax**), o Exército atira um tiro guiado (baseado na posição X do jogador, o Alien mais próximo atira) e outro executado por um Alien aleatório. Todos os Tiros são feitos pelos Aliens mais abaixo em suas colunas e armazenados no *ArrayList* **tiros**.

Ainda, existem outros atributos, como **direcao**, que é maior que zero quando o Exército está se movendo para a direita e menor que zero quando vai para a esquerda. A grande maioria dos métodos dessa classe é implementada através de dois *for loops*: um para a linha da matriz de Aliens e outra para a coluna. Dessa forma, é possível chamar os métodos da classe Alien para cada entidade.

2.1.3 Bases

Estende a classe Entidade e tem os atributos adicionais:

- **vivo**: valor booleano que indica se a Base está viva;
- **vidas**: quantas vidas a Base tem atualmente;
- **vidasMax**: máximo de vidas da Base (20);

Possui métodos para impressão, getters e setters para os atributos especificados, reset e decrementar a vida.

2.1.4 Nave

A classe Nave também estende a classe Entidade. Como pode atirar apenas um Tiro por vez, existe o atributo **tiro**, da classe Tiro. Ainda, os atributos **pontos** e **vidas** foram adicionados a essa classe.

Nessa classe foram implementados apenas *getters* e *setters* para **pontos**, **vidas** e **velocidadePadrao** da Nave, que é o módulo da sua velocidade no eixo x. Além disso, o método *atirar()* só permite que um Tiro exista por vez, caso não exista cria um novo Tiro que se movimenta para cima. O método *moverNave()* não deixa que a Nave atravessa as bordas laterais da tela, levando em consideração sua velocidade para calcular a colisão.

2.1.5 Tiro

Também estende a classe Entidade. Adiciona um único atributo booleano **visivel** que auxilia na sua impressão.

Os Tiros de Aliens e da Nave são diferenciados de acordo com a coordenada Y de sua velocidade: caso seja menor que zero (Tiro indo para cima), o Tiro é da Nave e colocamos sua **imagem** como tal; caso seja maior que zero (Tiro indo para baixo), o Tiro é do Exército e colocamos sua **imagem** como tal.

3 Engine

Pacote responsável pela lógica e funcionamento do jogo, bem como definições de parâmetros utilizados por outras classes, como as dimensões da tela.

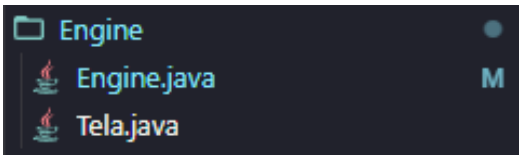


Figura 3: Estrutura de arquivos do pacote Engine.

3.0.1 Tela

Classe abstrata com as medidas de **alturaTela** e **larguraTela**. A classe Entidade estende essa clase, assim, todas as entidades do jogo conseguem ter acesso aos parâmetros da tela.

3.0.2 Engine

Guarda todos os objetos do jogo (**nave**, **exercito**, **base[]**, **alienEspecial**), além do nível de **dificuldade** atual. Essa classe é a responsável pela lógica principal do jogo, juntando em seus métodos as funções de todas as Entidades.

- *rodarJogo()*: loop principal da classe, implementa as lógicas de derrota e vitória, tal como os movimentos das entidades, colisões e tiros;
- *resetarJogo()*: reseta todas as entidades do jogo;
- *aumentarDificuldade()*: aumenta a dificuldade do jogo de acordo com a porcentagem de Aliens vivos (a cada 33% de Aliens destruídos aumenta a dificuldade);
- *moveTiros()*: move todos os Tiros instanciados no momento, usando em consideração sua velocidade;
- *trataColisoes()*: define quais Entidades devem colidir com o que. Sendo que todos os Tiros devem colidir entre si e com as Bases. Já a Nave deve colidir apenas com os Tiros dos Aliens e vice-versa. Colisões são tratadas de modo individual pelas funções *colisaoComBase()*, *colisaoComTiros()*, *colisaoComAliens()* e *colisaoComNave()*.

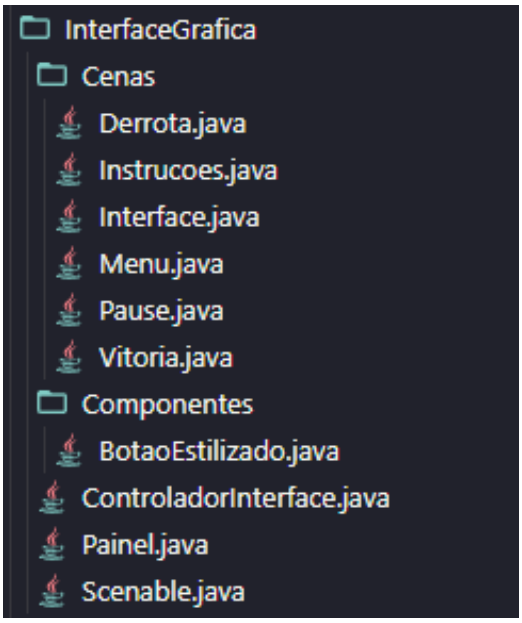


Figura 4: Estrutura de arquivos do pacote InterfaceGrafica.

4 Interface gráfica

Pacote responsável pelas cenas e pelo controle de toda a interface gráfica do jogo.

4.0.1 Interface

Implementa a interface Imprimivel, apresentando em si o método *imprimir()*. Imprime, durante o jogo, a pontuação e as vidas atuais da Nave. As vidas são impressas com imagens de corações.

4.0.2 Controlador Interface

Classe responsável por controlar a parte gráfica do jogo, bem como os comandos do teclado. Guarda em seus atributos todos as entidades do jogo, além das entidades da parte gráfica.

Os principais métodos implementados são:

- *rodarLoop()*: loop principal do jogo, roda dentro de si o método *rodarJogo()* da Engine, além de imprimir as entidades e dar início ao jogo trocando a cena de Menu para a tela do jogo. Os atributos **iniciado**, **pausado** (guarda o menu de Pause) e **telaAtual** (guarda a instancia de uma classe que estende Tela) auxiliam na implementação dessa função;
- *resetarJogo()*: reseta todas as entidades do jogo;
- *aumentarDificuldade()*: aumenta a dificuldade do jogo de acordo com a porcentagem de Aliens vivos (a cada 33% de Aliens destruídos aumenta a dificuldade);
- *moveTiros()*: move todos os Tiros instanciados no momento, usando em consideração sua velocidade;
- *trataColisoes()*: define quais Entidades devem colidir com o que. Sendo que todos os Tiros devem colidir entre si e com as Bases. Já a Nave deve colidir apenas com os

Tiros dos Aliens e vice-versa. Colisões são tratadas de modo individual pelas funções *colisaoComBase()*, *colisaoComTiros()*, *colisaoComAliens()* e *colisaoComNave()*.

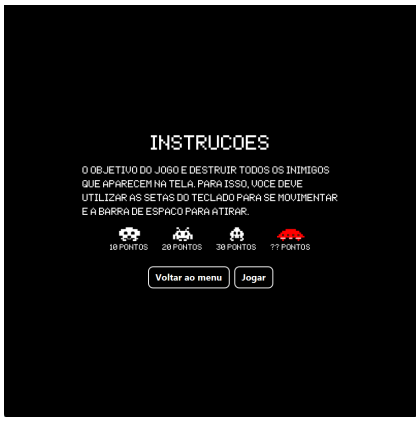
4.0.3 Scenable

Interface responsável por implementar apenas um método: *createPane()*. Esse método retorna um *Pane* do JavaFX para que possa ser utilizado em uma cena.

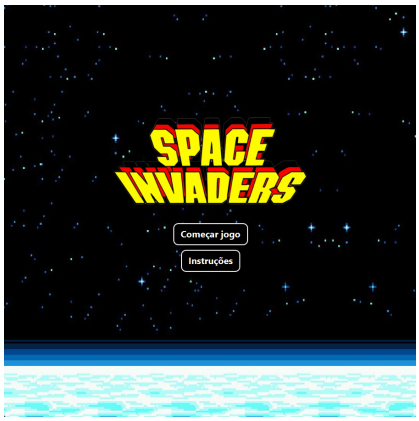
4.0.4 Painel

Essa classe estende a classe Tela e implementa a interface Scenable. Dessa forma, tem acesso a todos os parâmetros da tela. As únicas funções implementadas são os *setters* e *getters* do atributo **pane**.

(a) Tela de derrota



(b) Tela de instruções



(c) Menu principal

(d) Tela de pause

(e) Tela de vitória

Figura 5: Cenas do jogo

4.1 Cenas

Todas as classes desse subpacote estendem a classe abstrata Painel e tem como única função *createPane()*, o método implementado pela interface Scenable.

4.1.1 Derrota

Tela que aparece quando o jogador é derrotado pelos Aliens (perde as três vidas ou os Aliens se aproximam demais da parte inferior da tela). Na tela temos um texto de "GAME OVER"com botões de "Voltar ao menu"e "Sair".

4.1.2 Instruções

Tela que ensina o jogador novato as regras do jogo e seus controles, bem como a pontuação de cada Alien. É possível acessá-la através do botão "Instruções"do Menu. Nessa cena temos duas ações possíveis: iniciar o jogo ou voltar para o Menu.

4.1.3 Menu

Tela principal do jogo que aparece quando o jogador abre o programa. Na tela temos o logo do jogo, e os botões de começar um novo jogo e de ir para a tela de Instruções.

4.1.4 Pause

A tela de Pause se sobrepõe à tela do jogo quando acionada, dando opções para o jogador continuar o jogo ou voltar para o Menu.

4.1.5 Vitória

Tela que aparece para o jogador quando todos os Aliens são destruídos. O jogador consegue ver sua quantidade de pontos e tem a opção única de voltar para a tela inicial.

4.2 Componentes

Classes que estendem algum componente JavaFX para dar um estilo personalizado a ele.

4.2.1 Botão estilizado

Estende a classe Button do JavaFx, colocando cor de fundo, mudança do texto interno, *padding* e efeito de *hover* (passar o mouse por cima).

5 Space Invaders

Pacote principal que estende a classe do JavaFx Application onde fica o método *main()*. Para inicialização padrão do JavaFX foi implementado o método *start()*, que cria todas as entidades do jogo, as cenas e o ControladorInterface.

O jogo em si roda em um objeto da classe Canvas do JavaFX e o método *rodarLoop()* é iniciado dentro da função *handle()* de um AnimationTimer. Esse foi a estratégia usada para que as animações do jogo funcionassem.