Programação Orientada a Objetos
Segunda Parte do Projeto Prático
Manual do Sistema

Professor Dr. Robson Leonardo Ferreira Cordeiro
Bruno Alvarenga Colturato
N° USP 11200251

Dezembro de 2020

1 Estrutura do código

O código do jogo foi divido em 4 pacotes: Elementos do Sistema, Engine, Imagens e Interface. O package ProjetoSpaceInvaders contém a *main* do programa. A seguir, explicarei o funcionamento básico de cada pacote.

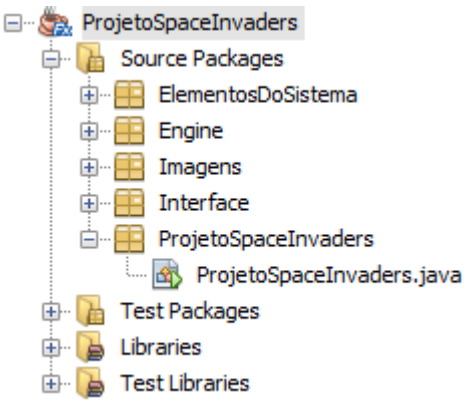

Figura 1: Divisão do trabalho em packages.

1.1 Interface Gráfica

Todo o funcionamento da interface gráfica foi baseada nos recursos da biblioteca JavaFX. Objetos do tipo Group, Scene, Canvas e GraphicsContext foram amplamente usados para controlar a janela do jogo. De forma especial, objetos do tipo GraphicsContext são os responsáveis por desenhar as imagens dos elementos do jogo na tela. Para tanto, utilizamos o GraphicsContext associado a objetos do tipo Images. O loop do jogo é gerado por um objeto do tipo AnimationTimer, o qual redesenha a tela dezenas de vezes por segundo, permitindo uma excelente fluidez para o game. Esses objetos foram declarados e definidos na classe "main"do programa, a saber *ProjetoSpaceInvaders.java*.

Na Fig. 6 podemos ver de forma mais específica as classes responsáveis pelo funcionamento geral da interface gráfica do jogo. A seguir, explicamos brevemente o funcionamento de cada uma delas.

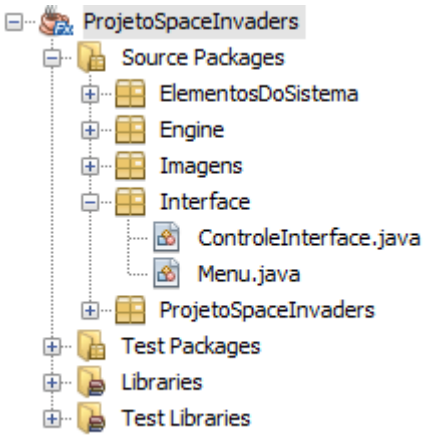

Figura 2: Conteúdo do pacote Interface.

1.1.1 Controle Interface

Esta classe é o coração do jogo, sendo a responsável desenhar na tela os elementos do game, controlar o menu, o inicio da partida, a exibição de informações na tela, entre outras funções. Entre os métodos mais importantes, temos: *definirTeclas* e *loopJogo*.

O método *definirTeclas* utiliza os recursos associados a Scene do jogo de forma a estabelecer quais métodos de quais objetos serão chamados ao ser identificado que o usuário pressionou determinada tecla do teclado.

O método *loopJogo* se utiliza de todas os atributos auxiliares criados dentro desta classe para, a partir da lógica de funcionamento do jogo, determinar se está pode ser iniciado, qual a dificuldade selecionada pelo usuário, quais objetos devemos desenhar na tela, quando o jogo deve ser finalizado etc. Vale ressaltar que é nesta funcionalidade que chamamos os métodos que verificam as colisões entre os elementos, modificam a posição de cada objeto e, de forma especial, verifica as condições que determinam se o jogo acabou ou não.

1.1.2 Menu

Esta classe visa concentrar os métodos utilizados em *ControleInterface* relacionados ao menu do jogo. Aqui encontramos os objetos que definem as imagens correspondentes às janelas do menu (por meio da classe Image) e os métodos responsáveis por desenhar esses elementos na tela.

1.2 Imagens

Este pacote tem a simples função de armazenar as imagens relacionadas a cada elemento do jogo. A criação desse pacote visa melhorar a modularização e a organização do código.

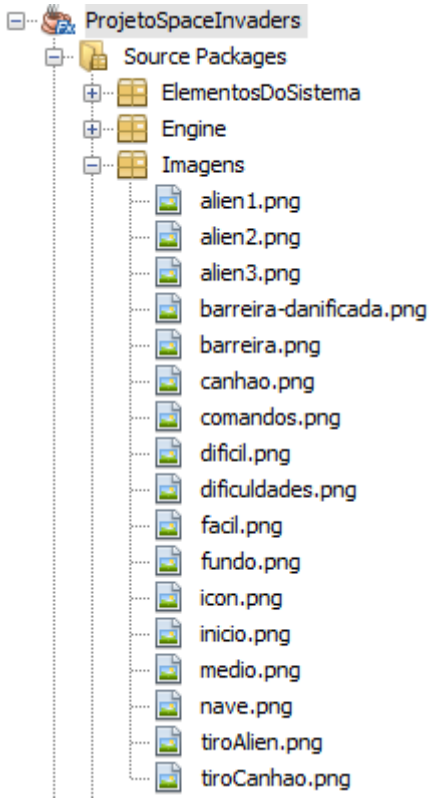

Figura 3: Conteúdo do pacote Imagens.

1.3 Elementos do Sistema

Na Fig. 4 podemos ver as classes que definem os elementos do jogo. Essas classes apenas implementam as **características** dos elementos, portanto não são responsáveis pelas ações dos objetos (como se movimentar, atirar etc). A seguir, explicamos brevemente o funcionamento de cada uma delas.

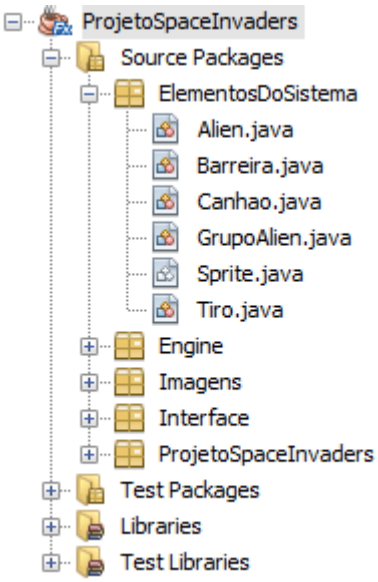

Figura 4: Conteúdo do pacote Elementos do Sistema.

1.3.1 Sprite

Esta classe é a mãe de praticamento todos as classes que vêm em seguida. A ideia por trás dela é reunir os atributos e métodos mais comuns aos elementos do jogo, a saber: um objeto do tipo Image (do JavaFX) para armazenar a imagem associada ao elemento em questão, variáveis para guardar as posições x e y do objeto, variáveis para guardar as velocidades em x e y, além dos tradicionais métodos *get* e *set*.

Implementamos nesta classe o método que verifica **colisões** com outros objetos do tipo Sprite e os métodos responsáveis por perimitir a **definição das imagens do objeto** e o **desenho** destes na tela do jogo (com o auxílio, obviamente, de um objeto do tipo GraphicsContext). Note que a classe também guarda as dimensões da tela do jogo, pois essa informação é muito utilizada na movimentação dos objetos na tela.

A inspiração para a criação desta classe veio do site `https://gamedevelopment.tutsplus.com`, no qual é possível encontrar um **tutorial** de como usar JavaFX para o desenvolvimento de jogos.

1.3.2 Alien

A classe alien é herdeira da classe Sprite, possuindo, portanto, todos os atributos e métodos da classe mãe. Como cada alien do jogo pode disparar um tiro contra o canhão, achei mais coerente que cada objeto do tipo alien possuísse como atributo um objeto do tipo Tiro, sendo a imagem associada a um tiro alienígena definida dentro da classe alien (o tiro alien é o mesmo para qualquer tipo alienígena).

Além dessas características considerei que um invasor possui apenas uma vida e, nesse sentido, adicionei à classe os atributos e métodos necessários para saber se um alien se encontra vivo ou não em determinado momento do jogo. Vale destacar que a imagem associada ao alien não é definida nesta classe, pois dependerá da posição que o invasor ocupa no grupo de invasores.

1.3.3 Barreira

A classe barreira é herdeira da classe Sprite, possuindo, portanto, todos os atributos e métodos da classe mãe. A imagem associada a uma barreira é padrão, logo já é atribuída dentro desta classe. Um aspecto relevante da barreira é que ela se degrada aos poucos, logo precisa possuir uma certa quantidade de vida que diminui aos poucos. Por padrão, defini a quantidade inicial de vida da barreira como 100. Essa vida diminui ao haver colisões com o tiro de um alien, sendo que cada tiro diminui a vida da barreira em 10 unidades, logo uma barreira consegue suportar 10 tiros aliens por partida.

1.3.4 Canhão

A classe canhão é herdeira da classe Sprite, possuindo, portanto, todos os atributos e métodos da classe mãe. Assim como no caso dos aliens, tournou-se natural considerar que um objeto do tipo canhão possuísse um objeto do tipo Tiro como um de seus atributos. Além disso, foram definidos os atributos de vida e de pontos com seus respectivos métodos de *get* e *set*. Por padrão, um canhão possui 3 vidas e sua pontuação inicial é nula.

Outros atributos necessários para controlar a movimentação do canhão foram criados (por exemplo, criei uma variável booleana que indica se o canhão está se movimentando para a direita ou não). A imagem do canhão é a mesma do jogo original, portanto é fixa e foi definida dentro desta classe.

1.3.5 Grupo Alien

Esta é uma das principais classes do game, não sendo herdeira da classe Sprite por implementarem conceitos muito divergentes. A ideia por trás desta classe é o de abrigar todos os 55 aliens do grupo de invasores. Para tanto, definimos uma matriz 5x11 cujos elementos são objetos da classe alien.

Ao inicializarmos a matriz de aliens, definimos a imagem correspondente a cada um dos invasores, sendo que esta é determinada pela posição destes na matriz (vide Fig. 5). A ideia de usar uma matriz veio da necessidade de controlar os aliens como um grupo, o que é bem facilitado ao utilizarmos a estratégia empregada. Para além das imagens, a única coisa que diferencia os tipo de aliens é a pontuação associadas a cada um deles (que é melhor explicada no Manual do Usuário).

Figura 5: Distribuição dos aliens na matriz e suas respectivas imagens.

Para o controle dos tiros do grupo, foram estabelecidos atributos responsáveis por controlar o número máximos de tiros simultâneos que o grupo pode disponibilizar na tela com seus respectivos métodos de *get* e *set*. Semelhantemente a movimentação do canhão, o controle da movimentação do grupo é auxiliada pela existência de 3 atributos que indicam se o grupo está se movendo para a direita, para a esquerda ou para baixo.

1.3.6 Tiro

A classe tiro é herdeira da classe Sprite, possuindo, portanto, todos os atributos e métodos da classe mãe. A ideia desta classe foi a de implementar todos os dois tipos de tiros possíveis no jogo: tiro de canhão e tiro de alien. Para tanto foram definidos atributos que controlam se o tiro está subindo ou descendo, além de um indicador (chamado, convencionalmente, de vida) para indicar se o tiro está ativo ou não. Como a classe implementa dois tipos possíveis de tiro, sua imagem somente é definida na classe que possui um objeto do tipo tiro como atributo, isto é, as classes canhão e alien.

1.4 Engine

Na Fig. 6 podemos ver as classes responsáveis pelo funcionamento da engine do jogo. Seguindo a modularização pedida pela especificação do trabalho, estas classes possuem a única função de modificar as características dos objetos definidos em 1.3. A seguir, explicamos brevemente o funcionamento de cada uma delas.

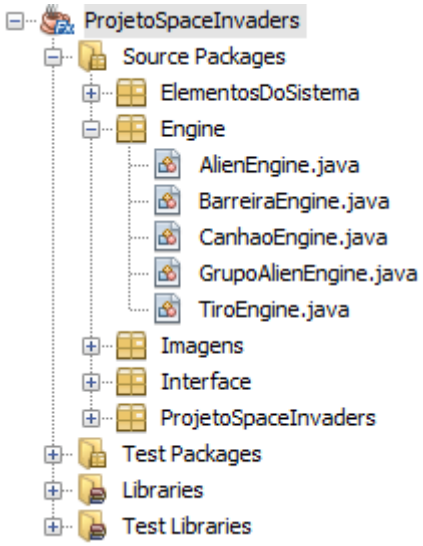

Figura 6: Conteúdo do pacote Engine.

1.4.1 Alien Engine

As ações relacionadas a um alien qualquer (lembrando que NÃO há distinção entre os aliens em termos de mecânica dentro do jogo) são basicamente duas: se movimentar e atirar. A movimentação dos aliens ocorre em grupo, logo é implementada pela classe GrupoAlienEngine. A fim de modularizar o código, criei métodos que controlam o tiro do alien, porém esses métodos se utilizam de um objeto do tipo TiroEngine para controlar o comportamento do tiro.

1.4.2 Barreira Engine

A engine de uma barreira consiste basicamente em identificar se houve colisão com o grupo de aliens (ou um de seus tiros) e, dessa maneira, diminuir a vida da barreira. Caso a vida da barreira seja menor que a metade de sua vida inicial de 100, sua imagem é modificada por meio do método *setImagem* definido na classe Sprite.

Para identificarmos uma colisão entre uma barreira e um tiro alien, chamamos o método *colisaoTiro*, que recebe como parâmetro um objeto do tipo GrupoAlien. A ideia é percorrer toda a matriz de aliens a fim de verificar a posição atual do tiro de cada um deles e comparar com a posição atual da barreira. Essa comparação é feita por meio do método *intersects* definido na classe Sprite. Se houver colisão, diminuimos a vida da barreira em 10 unidades.

Outra forma de colisão com o grupo de invasores se dá quando o grupo desce tanto na tela do jogo que chega a sobrepor a posição da barreira. Para identificar essa ocorrência, chamamos o método *colisaoAlien*, que calcula a distância entre a posição y da barreira e a posição y do alien vivo mais abaixo do grupo alien. Se essa distância for menor que um valor de controle, consideramos que houve colisão com o grupo e, portanto, a barreira perde instantaneamente toda sua vida.

1.4.3 Canhão Engine

As ações relacionadas a um canhão são basicamente duas: movimentar e atirar. Analogamente ao que foi feito com a AlienEngine, o tiro do canhão é controlado por um objeto

do tipo TiroEngine, cujo funcionamento é explicado abaixo. O controle de possíveis colisões entre o canhão e um tiro alien é feita percorrendo toda a matriz de aliens de um objeto GrupoAlien e verificando, por meio do método *intersects* da classe Sprite, se houve colisão entre o canhão e o tiro de algum dos aliens. Se houver colisão, diminuímos a vida do canhão em uma unidade e o redesenhamos na parte inferiro esquerda da tela.

A movimentação do canhão, por sua vez, segue o seguinte princípio: ao verificar-se que o usuário pressionou, por exemplo, a seta para a direita, o atributo que controla se o canhão está se movimentando para a direita é posto como "verdadeiro"e, nesse instante, a posição x do canhão é acrescida em certa quantidade padrão. O mesmo acontece quando o usuário pressiona a seta para a esquerda, sendo que nesse caso a posição x do canhão é decrescida em certa quantidade padrão. Além disso, com base nas dimensões da tela do jogo, é possível calcular até que valores mínimos e máximos de x a posição do canhão pode admitir, impedindo, assim, que o canhão desapareça da tela.

1.4.4 Grupo Alien Engine

As ações relacionadas ao grupo de invasores são: movimentar e atirar. A movimentação do grupo se dá verificando as variáveis que indicam se o grupo está se movimentando para a direita, para a esquerda ou para baixo. Vamos supor que os invasores estão se movimentando para a direita. Nesse caso, o procedimento padrão é percorrer a matriz de aliens para calcularmos a distância entre a parede direita e o alien vivo mais a direita. Calculada essa distância, verificamos se podemos movimentar cada um dos aliens certa quantidade para a direita ou não. Caso possamos, percorremos toda a matriz de aliens e somamos certa quantidade padrão a posição x do alien. Caso não seja possível, modificamos as variáveis que controlam o movimento do grupo de forma a indicar que o grupo deve se movimentar para baixo no próximo loop do jogo. O raciocínio é análogo para a movimentação para a esquerda e para baixo.

O funcionamento do tiro segue uma lógica parecida: percorremos toda a matriz de aliens de forma a ativar, desativar ou movimentar os tiros ativos de cada um dos invasores. Para verificarmos colisões entre um tiro de canhão e um dos aliens do grupo, percorremos toda a matriz de aliens e utilizamos o método *intersects* da classe Sprite (passando, obviamente, o tiro do canhão como parâmetro da função) para fazermos essa verificação. Caso haja colisão, setamos o alien como morto e, dessa forma, ele não mais aparecerá na tela do jogo. Além disso, a partir de determinado quantidade de aliens mortos, passamos a gradativamente aumetar a velocidade de moviementação dos alienígenas. Maiores detalhes sobre a implementação são encontrados no JavaDoc do programa.

1.4.5 Tiro Engine

O funcionamento do tiro consiste em ir para cima (se for um tiro controlado por um canhão) ou ir para baixo (se for um tiro controlado por um alien). Essa distinção (se o tiro se movimenta para cima ou para baixo) é estabelecida dentro de cada classe que possui um objeto do tipo tiro.

A movimentação do tiro é muito simples. Caso o tiro esteja se movimentando para cima, simplesmente subtraímos certa quantidade padrão da posição y do tiro, dando, assim, a sensação de movimentação para cima. A movimentação para baixo segue o mesmo raciocínio, porém adicionando certa quantidade padrão à posição y do tiro. Com base nas dimensões da tela do jogo é possível verificar se este passou pela borda superior

ou inferior da tela. Se isso ocorrer, devemos desativar o tiro e indicar que ele pode ser utilizado novamente.