Faculdade de Ciências Exatas e Engenharia

Programação Orientada por Objetos

Licenciatura em Engenharia Informática

Projeto 1

“O Naufrágio espacial”



**Trabalho Elaborado por Grupo 10:**

Pedro Diogo Gouveia Jardim Nº 2015118

Vasily Frolov Nº 2128020

João Afonso Teixera Neves Nº 2075920

Índice

[2. Índice de imagens 3](#_Toc88942755)

[3. Introdução 4](#_Toc88942756)

[4. Objetivos 4](#_Toc88942757)

[5. Requisitos do Jogo 5](#_Toc88942758)

[6. Cooperação entre dois jogadores 5](#_Toc88942759)

[7. Score com a pontuação atualizada em tempo real 8](#_Toc88942760)

[8. Indicador de vida, de balas e de tempo para cada jogador 10](#_Toc88942761)

[9. Modificação do aspeto do mundo e dos jogadores 12](#_Toc88942762)

[10. Indicação da pontuação obtida ao finalizar o jogo. 15](#_Toc88942763)

[11. Requisitos Programação orientada por Objetos 16](#_Toc88942764)

[12. Inicialização de objetos usando os construtores 16](#_Toc88942765)

[13. Herança de métodos com um mínimo de 2 níveis além de Actor 17](#_Toc88942766)

[14. Overriding de métodos 18](#_Toc88942767)

[15. Overloading de métodos 18](#_Toc88942768)

[16. Encapsulamento 19](#_Toc88942769)

[17. Conclusão 19](#_Toc88942770)

[18. Bibliografia 20](#_Toc88942771)

[19. Anexos 20](#_Toc88942772)

# Índice de imagens

[Figura 1- Construtor classe Player 6](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942729)

[Figura 2- Método moveAround() player 1 7](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942730)

[Figura 3- Método moveAround() player 1 7](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942731)

[Figura 4- Método fireBullet() 8](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942732)

[Figura 5- Figura 5 instanciar objetos player 1 e player 2 8](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942733)

[Figura 6- Classe Score 9](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942734)

[Figura 7-Método hitByBullet() da classe Alien 9](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942735)

[Figura 8- Método increaseScore() 10](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942736)

[Figura 9- Classe HealthBar 10](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942737)

[Figura 10- Classe Timer 11](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942738)

[Figura 11- Classe BulletsInfo 12](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942739)

[Figura 12- Exemplo de alteração de imagens dos jogadores 12](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942740)

[Figura 13- Atributos Globais class Player 12](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942741)

[Figura 14- Imagens dos jogadores 1 e jogadores 2 12](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942742)

[Figura 15- Método moveAround() 13](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942743)

[Figura 16- Buracos Negros 13](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942744)

[Figura 17 - Método isTouchingBlackHole 14](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942745)

[Figura 18- Métodos lauchMeteors(), lauchBulletsBoxes, lauchHealthSuply() 14](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942746)

[Figura 19- Classe FallingObjects() 15](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942747)

[Figura 20- Metodos defeat() e victory() 15](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942748)

[Figura 21- Mensagem de derrota 16](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942749)

[Figura 22- Mensagem de vitoria. 16](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942750)

[Figura 23- Contrutor da classe Alien 16](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942751)

[Figura 24- Heranças do projeto 17](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942752)

[Figura 25- Overriding método hitTheEdge() 18](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942753)

[Figura 26- Exemplo overloading 18](file:///C:\Users\pedro\Desktop\Engenharia%20Informatica\1_Ano_2021_2022\1_%20Semestre\POO\Projeto1\Relatorio%20projeto.docx#_Toc88942754)

# Introdução

Este projeto foi realizado no âmbito da cadeira “Programação Orientada por Objetos”. Este projeto teve como objetivo aplicar os conceitos de programação orientada por objetos através da plataforma “Greenfoot”.

Para a realização deste projeto foi proposto que fosse desenvolvido um jogo baseado num naufrágio, onde os náufragos têm que sobreviver num ambiente hostil até a chegada de ajuda externa.

O nosso projeto tem como base filmes de viagens espaciais, onde as personagens naufragam num planeta desconhecido, e tem que cooperar entre si até a chegada de ajuda.

O projeto foi desenvolvido para que dois jogadores possam jogar ao mesmo tempo no mesmo teclado, cooperando entre si. Os dois jogadores naufragaram num planeta desconhecido, onde terão que lutar contra seres desconhecidos (Aliens) e causas naturais (asteroides), até a chegada de ajuda externa.

Para realizar este projeto utilizamos os slides fornecidos pelos docentes nas aulas teorias e práticas, fóruns externos, vídeos no Youtube e autoinvestigação.

# Objetivos

O objetivo do nosso projeto foi conseguir realizar um jogo divertido de ser jogado e mostrar também o domínio da plataforma “Greenfoot”. Também tivemos como objetivo dominar o conceito de POO.

# Requisitos do Jogo

## Cooperação entre dois jogadores

Para ser possível utilizar dois jogadores, foi criado uma classe **Player**, que é uma subclasse da superclasse **Actor.**

Optamos por usar apenas uma classe para os dois jogadores, pois o código dos dois jogadores iam ser iguais.

Esta classe recebe três argumentos da classe **MyWorld**, estes três são:

**- playerId** é um argumento do tipo int, e serve para identificar os dois jogadores.

**- bulletsInfo** é um argumento do tipo **BulletsInfo** que é um objeto da classe **BulletsInfo**, e tem como objetivo obter a informação da quantidade de balas disponíveis, e informar quantas balas restam na arma.

**- healthBar** é um argumento do tipo **healthBar** que é um objeto da classe **HealthBar**, e tem como objetivo devolver a quantidade de vida dos jogadores.

No construtor da classe foi declarado várias variáveis para os vários tipos de imagens dos dois jogadores. Para cada jogador foi criado imagens para cada variação do jogador, quando se desloca para a direita, esquerda, cima, baixo, nas diagonais e para quando morre.

Uma imagem com texto, mesa

Descrição gerada automaticamente

Figura 1- Construtor classe Player

Esta classe contém 13 métodos no total, mas os mais importantes para o funcionamento dos dois jogadores são apenas dois:

- **moveAround**: Este método serve para movimentar os dois jogadores. Este método é composto, por um switch case que recebe o id do jogador, caso o id do jogador seja 1 o jogador 1 vai se movimentar com as teclas **W,A,S,D** e se o id do jogador for 2 o jogador 2 vai se movimentar com as teclas **UP, LEFT, DOWN, RIGHT.** Também neste método alem do jogador se movimentar conforme as teclas pressionadas, o jogador também muda a sua imagem e a sua rotação.

Uma imagem com texto

Descrição gerada automaticamente

Figura 2- Método moveAround() player 1

Uma imagem com texto

Descrição gerada automaticamente

Figura 3- Método moveAround() player 1

- **fireBullet:** É o método que torna possível os jogadores poderem disparar para matar os aliens. Este método tem um funcionamento similar ao método **moveAround()**, pois também usa um switch case. O switch case recebe também o id do jogador, e caso o id seja 1 o jogador 1 ira disparar com a tecla **G**  e caso o id do jogador seja 2 ira disparar com a tecla zero do numpad. Este método também é responsável por adicionar a classe **Bullet** ao mundo.

Uma imagem com texto

Descrição gerada automaticamente

Figura 4- Método fireBullet()

Para poder usar esta classe e utilizar os dois jogadores temos que instanciar dois objetos da classe Player, na classe **myWorld**. Como por exemplo na figura 5.

Uma imagem com texto

Descrição gerada automaticamente

Figura 5- Figura 5 instanciar objetos player 1 e player 2

## Score com a pontuação atualizada em tempo real

Para obter um score com a pontuação atualizada em tempo real é necessário criar uma classe **Score**. Esta classe é uma subclasse da superclasse Stats. Ela herda o método **getSize()**, e recebe dois argumentos, o argumento **size**, que serve para o tamanho do textoe o argumento **label**, que serve para mostrar o texto da estatística.

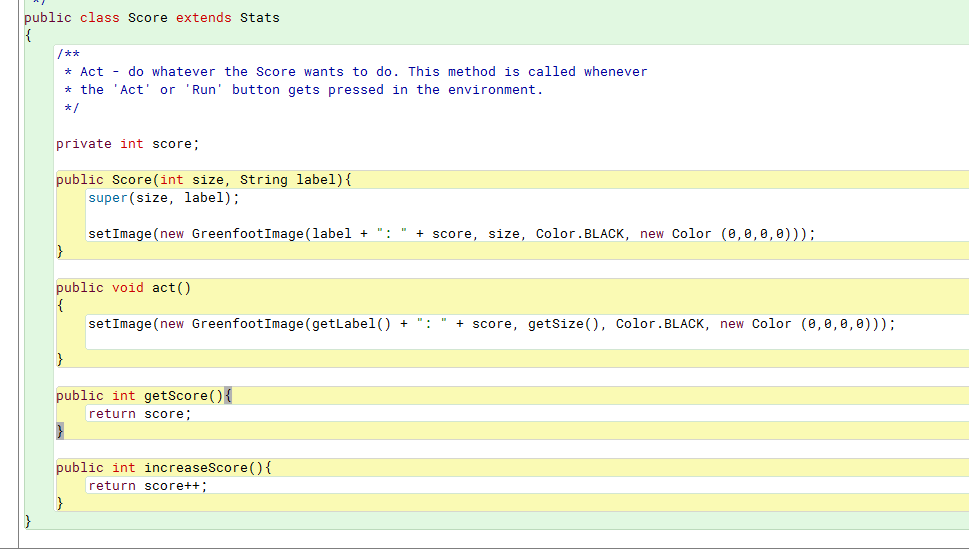


Figura 6- Classe Score

Para poder atualizar o Score em tempo real é preciso saber que ações é que irão aumentar o score. No caso deste jogo o score é aumentado quando os jogadores matam um alien. Por isso temos que ter dentro da classe alien um método que quando o alien morre, o score é incrementado.

Uma imagem com texto

Descrição gerada automaticamenteEste método serve para saber quando a bala disparada pelo player atinge o alien, e então podemos aproveitar este método para quando a vida do alien for igual a zero aumentar o score do jogo.

Figura 7-Método hitByBullet() da classe Alien

Uma imagem com texto

Descrição gerada automaticamente

Figura 8- Método increaseScore()

Para aumentar o score do jogo temos que utilizar o método **increaseScore()** para aumentar o score em tempo real.

## Indicador de vida, de balas e de tempo para cada jogador

O jogo contém dois indicadores de vida, um para cada jogador, informação sobre a quantidade de balas e um timer que vai decrementando até chegar a zero.

Estes indicadores são bastante importantes, especialmente os indicadores de vida e o timer, pois se a vida dos players acaba os jogadores perdem o jogo, e se o timer acabar e os jogadores terem sobrevivido (ou apenas um deles) ganham o jogo.

A classe **healthBar** serve para devolver aos jogadores a quantidade de vida de cada jogador, e atualizá-la em tempo real. A classe recebe como argumento os dados dos jogadores através do objeto **Player**.

Uma imagem com texto

Descrição gerada automaticamente

Figura 9- Classe HealthBar

Para criar uma barra de vida usei a manipulação de imagens do **greenfoot**, criei um retângulo vermelho, e como width passei a quantidade de vida. Depois bastou colocar o mesmo código dentro do método act, para a barra de vida poder ser atualizada quando os jogadores perdem/ganham vida.

A classe **Timer** é a classe responsável por devolver o **Timer** decrescente, informando assim os jogadores de quanto tempo falta até a chegada de ajuda externa.



Figura 10- Classe Timer

Para decrementar o timer foi simples, bastou criar um método **timer()** que decrementa tempo. E também criei um método **isTimeOver()**, que é usado para verificar através de uma condição **If** se o tempo é igual a zero. Assim podemos usar este método para verificar se a ajuda externa chegou e assim acabar o jogo.

**BulletsInfo** é a classe que retorna à quantidade de balas que se encontram disponíveis para os jogadores dispararem.

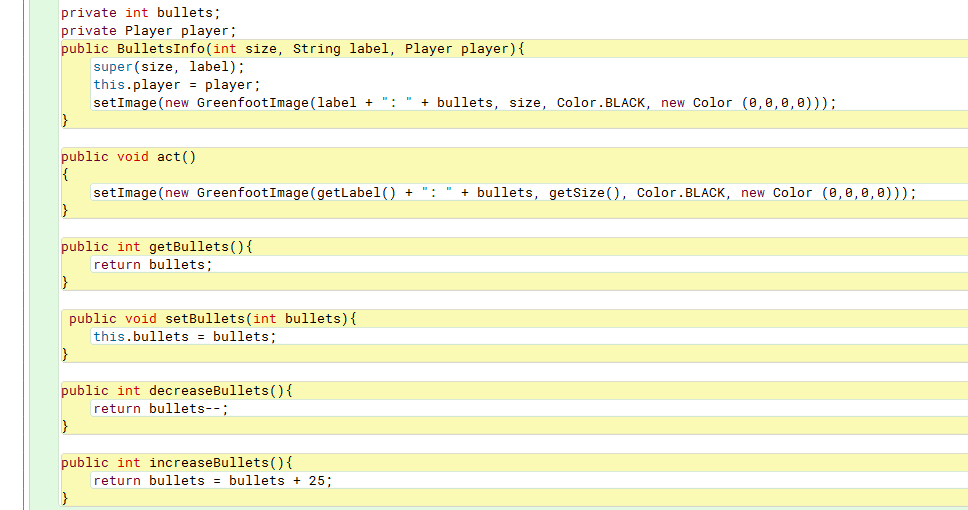


Figura 11- Classe BulletsInfo

## Modificação do aspeto do mundo e dos jogadores

Na classe player por exemplo, no método **moveAround()**, conforme o movimento do jogo a imagem do jogador é alterada, caso se mova na diagonal a imagem do jogador é alterada para dar uma melhor perspetiva de movimento aos utilizadores.



Figura 12- Exemplo de alteração de imagens dos jogadores

Figura 13- Atributos Globais class Player

Figura 14- Imagens dos jogadores 1 e jogadores 2

Uma imagem com texto

Descrição gerada automaticamente

Figura 15- Método moveAround()

Também foi adicionado dois buracos negros ao mundo, quando um jogador entra no buraco negro é redirecionado para o outro buraco negro.

Uma imagem com texto, exterior, transporte, luz

Descrição gerada automaticamente

Figura 16- Buracos Negros

Uma imagem com texto

Descrição gerada automaticamente Caso um jogador ou um alien toquem no buraco negro, serão movidos para outro canto do mundo onde se encontra o outro buraco negro.

Figura 17 - Método isTouchingBlackHole

Durante o jogo há vários objetos a cair do céu, caiem meteoritos que ao tocar nos jogadores tiram 25 pontos de vida, e também tem a queda de recargas de balas e de vidas.

Uma imagem com texto

Descrição gerada automaticamente

Figura 18- Métodos lauchMeteors(), lauchBulletsBoxes, lauchHealthSuply()

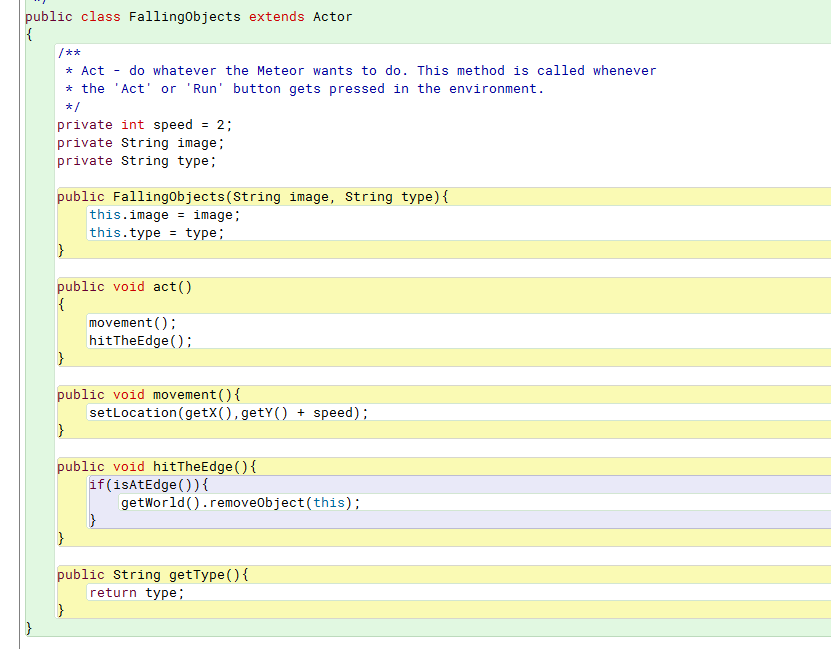
 Estas 3 classes são subclasses da superclasse **FallingObjects**, esta classe é responsável pela queda dos 3 objetos.

Figura 19- Classe FallingObjects()

A classe recebe duas Strings como argumentos, o argumento image recebe uma string com o nome da imagem, e a string type recebe uma string com o tipo de objeto.

## Indicação da pontuação obtida ao finalizar o jogo.

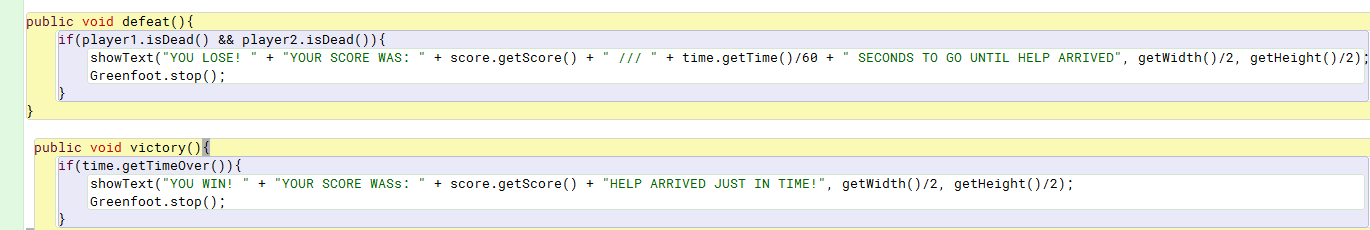
 Quando o jogador perde, ou ganha é mostrado uma mensagem com a pontuação final.

Figura 20- Metodos defeat() e victory()

Estes métodos são criados e usados dentro da classe **myWorld**. Tem que ser chamados dentro do método act.

Uma imagem com texto

Descrição gerada automaticamente

Figura 21- Mensagem de derrota

Uma imagem com texto

Descrição gerada automaticamente

Figura 22- Mensagem de vitoria.

# Requisitos Programação orientada por Objetos

## Inicialização de objetos usando os construtores

Um construtor é responsável por criar o objeto em memória, ou sejam instância a classe que foi definida. Os construtores são bastante importantes, pois facilitam a atribuição de uma imagem aos objetos, como a passagem de parâmetros entre as classes.

Uma imagem com texto

Descrição gerada automaticamente

Figura 23- Contrutor da classe Alien

## Herança de métodos com um mínimo de 2 níveis além de Actor

Em java, podemos criar classes que herdem métodos de outras classes, assim evitando repetição de código.

Para podermos usar este tipo de relacionamento, devemos usar a palavra reservada **extends**, de forma a identificar a classe que a nova classe ira herdar os métodos.

Por exemplo:

public class **Marca** **extends** **Carro.**

Assim **Marca** será uma **subclasse** da **superclasse** Carro.

No nosso projeto temos algumas classes com herança.

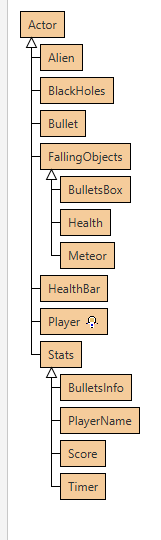


Figura 24- Heranças do projeto

## Overriding de métodos

“Overriding” é um recurso que permite a uma subclasse fornecer uma implementação específica de um método que já é fornecido por uma das suas superclasses.

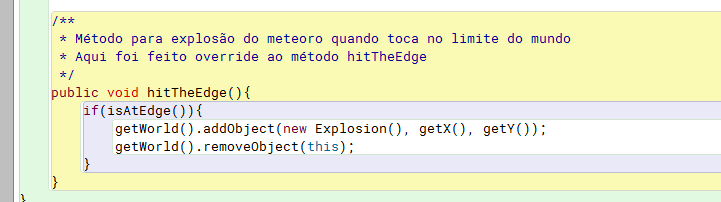
****Quando um método em uma subclasse tem o mesmo nome, os mesmos parâmetros e o mesmo tipo de retorno de um método em sua superclasse, diz-se que o método na subclasse sobrescreve o método no superclasse.

Figura 25- Overriding método hitTheEdge()

## Overloading de métodos

**Uma imagem com texto

Descrição gerada automaticamente** “Overloading” permite que métodos diferentes tenham o mesmo nome, mas assinaturas diferentes. “Overloading” pode servir para quando temos duas ou mais funções com comportamentos parecidos, mas que recebem diferentes tipos de argumentos.

Figura 26- Exemplo overloading

## Encapsulamento

O encapsulamento é um dos pilares da orientação a objetos. Através do encapsulamento é possível simplificar bastante a programação, bem como proteger informações sigilosas ou sensíveis.

O encapsulamento pode ser assegurado de três formas usando as palavras-chave:

**Private:** Membros declarados como **private**, só podem ser acessados dentro da classe em que foram declarados.

**Public:** Nível de acesso livre. Indica que o método ou atributo da classe é público, ou seja, pode-se acessar este atributo, ou executar esse método, por qualquer código de programa.

**Protected:** Nível de acesso intermediário. Serve para que, o método ou o atributo seja, público dentro do código da própria classe e de qualquer classe que herde daquela onde está o método ou propriedade protected. É privado e não acessível de qualquer outra parte.

# Conclusão

Com a realização deste projeto tivemos a oportunidade de aprofundar o conhecimento em JAVA e nos paradigma de programação POO. Também com este projeto foi possível ganhar uma noção de como criar um projeto deste tipo, e podemos tirar boas ideias e experiência para futuros projetos.

Concluído conseguimos criar um jogo divertido e competitivo através dos requisitos estabelecidos no enunciado do projeto.

# Bibliografia

1. <https://opengameart.org/> (Open Game Art é um repositório de mídia destinado ao uso com projetos de jogos de software livre e de código aberto, oferecendo ativos de conteúdo aberto.)
2. <https://www.greenfoot.org/home>
3. <https://www.youtube.com/watch?v=t8I3kfM0a1g&ab_channel=TannerCrow> (Java Game Programming Tutorial)
4. <https://www.pixilart.com/draw> (Website de desenho em pixels)
5. <https://www.w3schools.com/java/> (JAVA tutorial)

# Anexos

# Código do Projeto

# Classe **MyWorld**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class MyWorld extends World

{

/\*\*

\* Constructor for objects of class MyWorld.

\*

\*/

/\*\*

\* Contador para usar nas subclasses da superclasse FallingObjects

\*/

private int count = 0;

/\*\*

\* Velocidade do aparecimento de aliens

\*/

private int spawnSpeed = 50, randomSpawn = Greenfoot.getRandomNumber(8), life = 50, horizontalMove = 1, player1Id = 1, player2Id = 2, bulletsQuantity = 50, health = 50;

private Player player1, player2;

private Score score;

private HealthBar healthBarPlayer, healthBarPlayer2;

private PlayerName player1Name, player2Name;

private Timer time;

private BulletsInfo player1BulletsInfo, player2BulletsInfo;

private String meteorImage = "Meteor0.png", meteorType = "Meteor", bulletsBoxImage = "BulletsBox.png", bulletsBoxType = "BulletsBox", healthSuplyImage = "HealthSuply.png",

healthSuplyType = "HealthSuply", bulletsLabel = "Bullets", scoreLabel = "Score", playerLabel = "Player", timeLabel = "Time";

private BlackHoles blackHole1, blackHole2;

/\*\*

\* Construtor

\*/

public MyWorld()

{

super(800, 700, 1);

/\*\*

\* Adicionar cor ao background

\*/

GreenfootImage background = new GreenfootImage("planetBackground.jpg");

background.scale(getWidth(), getHeight());

setBackground(background);

/\*\*

\* Inicialização dos metodos de invocação

\*/

inicializeBulletsInfo();

populateWorld();

inicializeHealthBar();

inicializeScore();

inicializeStats();

addBlackHoles();

}

/\*\*

\* Método para inicializar o score no mundo

\*/

private void inicializeScore(){

score = new Score(40, scoreLabel);

addObject(score, getWidth()/2, getHeight()-20);

}

/\*\*

\* Método para inicializar a informação das balas no mundo

\*/

private void inicializeBulletsInfo(){

player1BulletsInfo = new BulletsInfo(20, bulletsLabel, player1);

player2BulletsInfo = new BulletsInfo(20, bulletsLabel, player2);

}

/\*\*

\* Método para inicializar a informação das barras de vida

\*/

private void inicializeHealthBar(){

healthBarPlayer = new HealthBar(player1);

healthBarPlayer2 = new HealthBar(player2);

healthBarPlayer.setHealth(health);

healthBarPlayer2.setHealth(health);

}

/\*\*

\* Método para inicializar as estatisticas do jogo

\*/

private void inicializeStats(){

player1Name = new PlayerName(25, playerLabel, player1Id);

player2Name = new PlayerName(25, playerLabel, player2Id);

time = new Timer(40, timeLabel);

addObject(time, getWidth()/2,20);

addObject(player1Name, getWidth()/10,getHeight()/20);

addObject(player1BulletsInfo, player1Name.getX(),player1Name.getY()+20);

addObject(healthBarPlayer, player1Name.getX() + 70,player1Name.getY());

addObject(player2Name, getWidth()/10,getHeight()/8);

addObject(player2BulletsInfo, player2Name.getX(),player2Name.getY()+20);

addObject(healthBarPlayer2, player2Name.getX() + 70,player2Name.getY());

}

/\*\*

\* Método para inicializar os jogadores e adiciona-los no mundo

\*/

private void populateWorld()

{

player1BulletsInfo.setBullets(bulletsQuantity);

player2BulletsInfo.setBullets(bulletsQuantity);

player1 = new Player(player1Id, player1BulletsInfo, healthBarPlayer);

player2 = new Player(player2Id, player2BulletsInfo, healthBarPlayer2);

addObject(player1,getWidth()/10,getHeight()/10);

addObject(player2,getWidth()/10 + 50,getHeight()/10);

}

/\*\*

\* Método para inicializar os buracos negros no mundo

\*/

private void addBlackHoles(){

blackHole1 = new BlackHoles();

blackHole2 = new BlackHoles();

addObject(blackHole1, 100, 600);

addObject(blackHole2, 700,100);

}

public void act(){

count++;

spawAliens();

lauchMeteors();

lauchBulletsBoxes();

lauchHealthSuply();

defeat();

victory();

}

/\*\*

\* Método para adicionar aliens pelas bordas do mapa

\*/

private void spawAliens(){

if(count % spawnSpeed == 0){

randomSpawn = Greenfoot.getRandomNumber(8);

switch(randomSpawn){

case 0 : addObject(new Alien(player1, player2, score), 0,0);

break;

case 1 : addObject(new Alien(player1, player2, score), getWidth()/2,0);

break;

case 2 : addObject(new Alien(player1, player2, score), getWidth(),0);

break;

case 3 : addObject(new Alien(player1, player2, score), 0, getHeight()/2);

break;

case 4 : addObject(new Alien(player1, player2, score), getWidth(), getHeight()/2);

break;

case 5 : addObject(new Alien(player1, player2, score), 0, getHeight());

break;

case 6 : addObject(new Alien(player1, player2, score), getWidth()/2,getHeight());

break;

case 7 : addObject(new Alien(player1, player2, score), getWidth(), getHeight());

break;

}

}

}

/\*\*

\* Método para lançar meteoritos

\*/

public void lauchMeteors(){

if(count % (spawnSpeed\*2) == 0){

int randomWidth = Greenfoot.getRandomNumber(getWidth());

addObject(new Meteor(meteorImage, meteorType), randomWidth,0);

}

}

/\*\*

\* Método para lançar recargas de balas

\*/

public void lauchBulletsBoxes(){

if(count % (spawnSpeed\*4) == 0){

int randomWidth = Greenfoot.getRandomNumber(getWidth());

addObject(new BulletsBox(bulletsBoxImage, bulletsBoxType), randomWidth,0);

}

}

/\*\*

\* Método para lançar recargas de vida

\*/

public void lauchHealthSuply(){

if(count % (spawnSpeed\*6) == 0){

int randomWidth = Greenfoot.getRandomNumber(getWidth());

addObject(new Health(healthSuplyImage, healthSuplyType, horizontalMove), randomWidth,0);

}

}

/\*\*

\* Método para devolver uma mensagem quando os jogadores perderem

\*/

public void defeat(){

if(player1.isDead() && player2.isDead()){

showText("YOU LOSE! " + "YOUR SCORE WAS: " + score.getScore() + " /// " + time.getTime()/60 + " SECONDS TO GO UNTIL HELP ARRIVED", getWidth()/2, getHeight()/2);

Greenfoot.playSound("gameOver.wav");

Greenfoot.stop();

}

}

/\*\*

\* Método para devolver uma mensagem quando os jogadores ganharem

\*/

public void victory(){

if(time.getTimeOver()){

showText("YOU WIN! " + "YOUR SCORE WASs: " + score.getScore() + "HELP ARRIVED JUST IN TIME!", getWidth()/2, getHeight()/2);

Greenfoot.playSound("victory.wav");

Greenfoot.stop();

}

}

}

# Classe **Alien**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Alien extends Actor

{

/\*\*

\* Act - do whatever the Alien wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

private int count, health = 1;

private Player player1, player2;

private Score score;

/\*\*

\* Construtor

\*/

public Alien(Player player1, Player player2, Score score){

this.player1 = player1;

this.player2 = player2;

this.score = score;

/\*\*

\* Aqui definimos a imagem do alien e o seu tamanho e para onde tem que estar virado

\*/

setImage("alien1.png");

getImage().rotate(180);

getImage().scale(80,80);

}

public void act()

{

count++;

moveAround();

hitByBullet();

if(health > 0){

isTouchingBlackHole();

}

}

/\*\*

\* Método para o movimento dos aliens

\*/

public void moveAround(){

move(1);

/\*\*

\* Aqui foi criado estas duas variaveis double

\* para saber a distancia que os aliens estao de cada jogador

\*/

double player1Distance = Math.sqrt(Math.pow(getX()-player1.getX(),2)+Math.pow(getY()-player1.getY(),2));

double player2Distance = Math.sqrt(Math.pow(getX()-player2.getX(),2)+Math.pow(getY()-player2.getY(),2));

/\*\*

\* Os aliens irao se mover em diração ao jogador mais proximo

\*/

if(player1Distance > player2Distance){

turnTowards(player2.getX(), player2.getY());

}else{

turnTowards(player1.getX(), player1.getY());

}

/\*\*

\* E aqui verificamos se tem algum jogador morto,

\* se tiver os aliens irao se mover em direção ao jogador vivo

\*/

if(player1.isDead()){

turnTowards(player2.getX(), player2.getY());

}else if(player2.isDead()){

turnTowards(player1.getX(), player1.getY());

}

}

/\*\*

\* Método para verificar se o alien é atingido por uma bala

\*/

public void hitByBullet(){

Actor bullet = getOneIntersectingObject(Bullet.class);

if(bullet != null){

/\*\*

\* Aqui caso o alien seja atingido,

\* a sua vida sera decrementada

\*/

health--;

getWorld().removeObject(bullet);

}

/\*\*

\* Caso o Alien morra,

\* a pontuação sera aumentada

\* e o objeto sera removido do mundo

\*/

if(health == 0){

score.increaseScore();

getWorld().removeObject(this);

}

}

/\*\*

\* Este metodo verifica se o alien esta em contacto com o buraco negro

\*/

public void isTouchingBlackHole() {

Actor blackHole = getOneObjectAtOffset(0, 0, BlackHoles.class);

if (blackHole != null) {

if(getX() > 100 && getX() < 120 && getY() > 600 && getY() < 620){

setLocation(601, 101);

}else if(getX() > 700 && getX() < 720 && getY() > 100 && getY() < 120){

setLocation(101, 701);

}

}

}

}

# Class **BlackHoles**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class BlackHoles extends Actor

{

/\*\*

\* Act - do whatever the BlackHoles wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

/\*\*

\* Construtor

\*/

public BlackHoles(){

/\*\*

\* Adicionamos a imagem ao objeto

\*/

setImage("blackhole.png");

getImage().scale(80,80);

}

public void act()

{

/\*\*

\* Para tornar o buraco negro mais realista adicionamos uma rotação ao objeto

\*/

getImage().rotate(5);

}

}

# Classe **Bullet**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Bullet extends Actor

{

private int speed;

/\*\*

\* Act - do whatever the bullet wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public Bullet(int rotation){

setRotation(rotation);

speed = 15;

setImage(new GreenfootImage(10,2));

getImage().setColor(Color.BLACK);

getImage().fillRect(0,0,10,2);

}

public void act()

{

move(speed);

hitTheEdge();

}

public void hitTheEdge(){

if(isAtEdge()){

getWorld().removeObject(this);

}

}

}

# Classe **Explosion**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Explosion extends Actor

{

/\*\*

\* Declaração de um array

\*/

private GreenfootImage[] images;

/\*\*

\* Index para percorrer o array

\*/

private int index;

/\*\*

\* Construtor

\*/

public Explosion()

{

images = new GreenfootImage [20];

/\*\*

\* Preencher o array com imagens de tamanho crescente

\*/

for (int i =0; i < images.length; i++ )

{

images[i] = new GreenfootImage("explosion.png");

images[i].scale(5\*i + 5, 5\*i + 5);

}

setImage(images[0]);

}

public void act()

{

animateExplosion();

}

/\*\*

\* Método para fazer a animação da explosão

\*/

public void animateExplosion()

{

index++;

if (index < images.length)

setImage(images[index]);

else

getImage().clear();

}

}

# Classe **FallingObjects**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Superclasse

\*/

public class FallingObjects extends Actor

{

/\*\*

\* Act - do whatever the Meteor wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

private int speed = 2;

private String image;

private String type;

public FallingObjects(String image, String type){

this.image = image;

this.type = type;

}

public void act()

{

movement();

hitTheEdge();

}

/\*\*

\* Este metodo serve para mover os

\* objetos na vertical

\*/

public void movement(){

setLocation(getX() ,getY() + speed);

}

/\*\*

\* Este metodo serve para remover o objeto quando o mesmo tocar nos limites

\* do mundo

\*/

public void hitTheEdge(){

if(isAtEdge()){

getWorld().removeObject(this);

}

}

/\*\*

\* Getter type

\*/

public String getType(){

return type;

}

/\*\*

\* Getter speed

\*/

public int getSpeed(){

return speed;

}

}

# Classe **BulletsBox**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Subclass da superclasse FallingObjects

\*/

public class BulletsBox extends FallingObjects

{

/\*\*

\* Act - do whatever the BulletsBox wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public BulletsBox(String image, String type){

super(image, type);

setImage(image);

getImage().scale(25,25);

}

public void act()

{

/\*\*

\* Aqui apenas usamos os metodos da superclasse FallingObjects

\*/

movement();

hitTheEdge();

}

}

# Classe **Health**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Subclasse da superclasse FallingObjects

\*/

public class Health extends FallingObjects

{

/\*\*

\* Act - do whatever the BulletsBox wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

private int horizontalMove;

public Health(String image, String type, int horizontalMove ){

super(image, type);

/\*\*

\* Para se tornar mais dificil de apanhar uma recarga de vida

\* colocamos esta variavel

\* para o objeto poder se mover na horizontal

\* assim ele desce na diagonal

\*/

this.horizontalMove = horizontalMove;

setImage(image);

getImage().scale(25,25);

}

public void act()

{

movement(horizontalMove);

hitTheEdge();

}

/\*\*

\* Método para o movimento da recarga da vida

\* Aqui foi feito overload ao método movement, para o objeto poder se descolar na diagonal

\*/

public void movement(int horizontalMove){

setLocation(getX() + horizontalMove ,getY() + getSpeed());

}

}

# Classe **Meteor**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Subclass da superclasse FallingObjects

\*/

public class Meteor extends FallingObjects

{

/\*\*

\* Act - do whatever the Meteor wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

private int speed = 2, animateMeteor = 0, animateSpeed = 5, count;

public Meteor(String image, String type){

super(image, type);

setImage(image);

getImage().scale(100,100);

}

public void act()

{

count++;

animateMeteor();

movement();

hitTheEdge();

}

/\*\*

\* Método para mover a imagem do meteorito

\* Deixando ela assim mais real

\*/

public void animateMeteor(){

if(count % animateSpeed == 0){

if(animateMeteor > 2){

animateMeteor = 0;

}

setImage(getType() + animateMeteor + ".png");

getImage().scale(100,100);

animateMeteor++;

}

}

/\*\*

\* Método para explosão do meteoro quando toca no limite do mundo

\* Aqui foi feito override ao método hitTheEdge

\*/

public void hitTheEdge(){

if(isAtEdge()){

getWorld().addObject(new Explosion(), getX(), getY());

getWorld().removeObject(this);

}

}

}

# Classe **HealthBar**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class HealthBar extends Actor

{

/\*\*

\* Act - do whatever the HealthBar wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

private int health;

private Player player;

public HealthBar(Player player){

this.player = player;

/\*\*

\* Criamos uma imagem ataves do getImage()

\* para fazer uma barra de vida

\*/

setImage(new GreenfootImage(52, 12));

getImage().drawRect(0,0,51,11);

getImage().setColor(Color.RED);

getImage().fillRect(1,1,getHealth(),10);

}

public void act()

{

/\*\*

\* Conforme o desenvolver do jogo o jogadores irao perder, ou ganhar vida

\*/

setImage(new GreenfootImage(52, 12));

getImage().drawRect(0,0,51,11);

getImage().setColor(Color.RED);

getImage().fillRect(1,1,getHealth(),10);

loseHealth();

gainHealth();

isHealthZero();

}

/\*\*

\* Método que reduz a vida dos jogadores

\*/

public void loseHealth(){

if(health > 0){

/\*\*

\* Atraves dos metodos getIntersectingAlien e

\* getIntersectingMeteor da classe player

\* podemos verificar se os players

\* foram atingidos por meteoritos, ou atacados

\* por aliens

\*/

if(player.getIntersectingAlien()){

health--;

}

if(player.getIntersectingMeteor()){

health = health - 25;

}

};

}

/\*\*

\* Este metodo verifica se a vida do jogador esta a zero,

\* e caso esteja é passo o argumento a true para a classe player atraves do metodo

\* isPlayerDead

\*/

public void isHealthZero(){

if(health == 0){

player.isPlayerDead(true);

}else{

player.isPlayerDead(false);

}

}

/\*\*

\* Getter health

\*/

public int getHealth(){

return health;

}

/\*\*

\* Setter Health

\*/

public void setHealth(int health){

this.health = health;

}

/\*\*

\* Este metodo serve para saber quando os players

\* apanharam a recarga de vida para poder

\* ser incrementada a vida

\*/

public void gainHealth(){

if(player.getIntersectingHealthSuply()){

health = health + 25;

};

}

}

# Classe **Player**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Player extends Actor {

private GreenfootImage playerBack, playerFront, playerDownLeft, playerDownRight, playerLeft, playerRight, playerUpperLeft, playerUpperRight, playerDead;

private int position, rotation, playerId;

private Boolean intersectAlien = false, intersectMeteor = false, intersectHealthSuply = false,

isDead = false;

private final int player1 = 1, player2 = 2;

private HealthBar healthBar;

private BulletsInfo bulletsInfo;

/\*\*

\* Act - do whatever the Player wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

/\*\*

\* Construtor

\*/

public Player(int playerId, BulletsInfo bulletsInfo, HealthBar healthBar) {

this.playerId = playerId;

this.bulletsInfo = bulletsInfo;

this.healthBar = healthBar;

/\*\*

\* Variações de imagens conforme o movimento do jogador

\*/

GreenfootImage playerImage = new GreenfootImage("player" + playerId + ".png");

playerImage.scale(100, 100);

setImage(playerImage);

playerBack = new GreenfootImage("player" + playerId + "-back.png");

playerBack.scale(100, 100);

playerFront = new GreenfootImage("player" + playerId + "-front.png");

playerFront.scale(100, 100);

playerDownLeft = new GreenfootImage("player" + playerId + "-down-left.png");

playerDownLeft.scale(100, 100);

playerDownRight = new GreenfootImage("player" + playerId + "-down-right.png");

playerDownRight.scale(100, 100);

playerLeft = new GreenfootImage("player" + playerId + "-left.png");

playerLeft.scale(100, 100);

playerRight = new GreenfootImage("player" + playerId + "-right.png");

playerRight.scale(100, 100);

playerUpperLeft = new GreenfootImage("player" + playerId + "-upper-left.png");

playerUpperLeft.scale(100, 100);

playerUpperRight = new GreenfootImage("player" + playerId + "-upper-right.png");

playerUpperRight.scale(100, 100);

playerDead = new GreenfootImage("player" + playerId + "dead.png");

playerDead.scale(100, 100);

}

public void act() {

/\*\*

\* Os jogadores so se podem mover e ser atingidos quando o jogador estiver vivo

\*/

if(!isDead){

moveAround();

fireBullet();

isHitByAlien();

isHitByMeteor();

isTouchingBlackHole();

pickHealthSuply();

pickAmmoBox();

}

dead();

}

/\*\*

\* Método para os jogadores se poderem mover

\*/

public void moveAround() {

/\*\*

\* Foi utilizado aqui um switch case para ser mais facil

\* de ler o codigo

\*/

switch (playerId) {

case player1:

/\*\*

\* Caso o id do jogador seja igual ao id do player 1 o jogador ira se movimentar

\* atraves das teclas WASD

\*/

if (Greenfoot.isKeyDown("w")) {

setLocation(getX(), getY() - 2);

/\*\*

\* Conforme as teclas pressionadas a imagem do jogador

\* é alterada, assim os jogadores terao uma maior noção

\* das posições de cada player

\*/

setImage(playerBack);

rotation = 270;

}

if (Greenfoot.isKeyDown("s")) {

setLocation(getX(), getY() + 2);

setImage(playerFront);

rotation = 90;

}

if (Greenfoot.isKeyDown("a")) {

setLocation(getX() - 2, getY());

setImage(playerLeft);

rotation = 180;

}

if (Greenfoot.isKeyDown("d")) {

setLocation(getX() + 2, getY());

setImage(playerRight);

rotation = 0;

}

if (Greenfoot.isKeyDown("a") && Greenfoot.isKeyDown("w")) {

setLocation(getX() - 1, getY() - 1);

setImage(playerUpperLeft);

rotation = 225;

}

if (Greenfoot.isKeyDown("d") && Greenfoot.isKeyDown("w")) {

setLocation(getX() + 1, getY() - 1);

setImage(playerUpperRight);

rotation = 315;

}

if (Greenfoot.isKeyDown("a") && Greenfoot.isKeyDown("s")) {

setLocation(getX() - 1, getY() + 1);

setImage(playerDownLeft);

rotation = 135;

}

if (Greenfoot.isKeyDown("d") && Greenfoot.isKeyDown("s")) {

setLocation(getX() + 1, getY() + 1);

setImage(playerDownRight);

rotation = 45;

}

break;

/\*\*

\* Caso o id do jogador seja igual ao id do player 2 o jogador ira se movimentar

\* atraves das teclas UP DOWN LEFT RIGHT

\*/

case player2:

if (Greenfoot.isKeyDown("up")) {

setLocation(getX(), getY() - 2);

setImage(playerBack);

rotation = 270;

}

if (Greenfoot.isKeyDown("down")) {

setLocation(getX(), getY() + 2);

setImage(playerFront);

rotation = 90;

}

if (Greenfoot.isKeyDown("left")) {

setLocation(getX() - 2, getY());

setImage(playerLeft);

rotation = 180;

}

if (Greenfoot.isKeyDown("right")) {

setLocation(getX() + 2, getY());

setImage(playerRight);

rotation = 0;

}

if (Greenfoot.isKeyDown("left") && Greenfoot.isKeyDown("up")) {

setLocation(getX() - 2, getY() - 2);

setImage(playerUpperLeft);

rotation = 225;

}

if (Greenfoot.isKeyDown("right") && Greenfoot.isKeyDown("up")) {

setLocation(getX() + 2, getY() - 2);

setImage(playerUpperRight);

rotation = 315;

}

if (Greenfoot.isKeyDown("left") && Greenfoot.isKeyDown("down")) {

setLocation(getX() - 2, getY() + 2);

setImage(playerDownLeft);

rotation = 135;

}

if (Greenfoot.isKeyDown("right") && Greenfoot.isKeyDown("down")) {

setLocation(getX() + 2, getY() + 2);

setImage(playerDownRight);

rotation = 45;

}

break;

}

}

/\*\*

\* Método para verificar se o jogador esta a tocar no buraco negro

\*/

public void isTouchingBlackHole() {

Actor blackHole = getOneObjectAtOffset(0, 0, BlackHoles.class);

if (blackHole != null) {

/\*\*

\* Caso o jogador esteja a tocar no buraco negro

\*/

if(getX() > 100 && getX() < 120 && getY() > 600 && getY() < 620){

/\*\*

\* E caso esteja dentro destas coordenadas,

\* o player é redirecionado para o outro buraco negro

\*/

Greenfoot.playSound("teleport.wav");

setLocation(601, 101);

}else if(getX() > 700 && getX() < 720 && getY() > 100 && getY() < 120){

Greenfoot.playSound("teleport.wav");

setLocation(101, 701);

}

}

}

/\*\*

\* Método para o jogador poder disparar

\*/

public void fireBullet() {

switch (playerId) {

case player1:

/\*\*

\* Caso o id do jogador seja igual ao id do player 1 o jogador ira

\* disparar na tecla G

\*/

if (Greenfoot.isKeyDown("g")) {

if (bulletsInfo.getBullets() > 0) {

/\*\*

\* Ao disparar as balas e se o jogador ainda ter balas na arma

\* elas irao sendo decrementadas

\*/

bulletsInfo.decreaseBullets();

/\*\*

\* Conforme a rotação do jogador a bala ira nessa direção

\*/

getWorld().addObject(new Bullet(rotation), getX(), getY());

}

}

break;

case player2:

/\*\*

\* Caso o id do jogador seja igual ao id do player 2 o jogador ira

\* disparar na tecla 0 do numpad

\*/

if (Greenfoot.isKeyDown("0")) {

if (bulletsInfo.getBullets() > 0) {

bulletsInfo.decreaseBullets();

getWorld().addObject(new Bullet(rotation), getX(), getY());

}

}

}

}

/\*\*

\* Método para saber quando o jogador esta em contacto com o alien

\*/

public void isHitByAlien() {

Actor alien = getOneObjectAtOffset(0, 0, Alien.class);

if (alien != null) {

/\*\*

\* Sempre que o jogador estiver

\* em contacto com o alien

\* ira perder vida

\*/

intersectAlien = true;

} else {

intersectAlien = false;

}

}

/\*\*

\* Método para saber quando o jogador apanha

\*/

public void pickHealthSuply() {

Actor healthSuply = getOneObjectAtOffset(0, 0, Health.class);

if (healthSuply != null) {

intersectHealthSuply = true;

/\*\*

\* Quando o jogador apanhar uma recarga de vida

\* a sua vida sera aumentada em 25 pontos

\*/

getWorld().removeObject(healthSuply);

} else {

intersectHealthSuply = false;

}

}

/\*\*

\* Método para saber quando o jogador é atingido por um meteorito

\*/

public void isHitByMeteor() {

Actor meteor = getOneObjectAtOffset(0, 0, Meteor.class);

if (meteor != null) {

intersectMeteor = true;

/\*\*

\* Quando o jogador for atingido pelo meteorito

\* ira perder vida e sera provocada uma explosao

\*/

getWorld().addObject(new Explosion(), getX(), getY());

getWorld().removeObject(meteor);

} else {

intersectMeteor = false;

}

}

/\*\*

\* Método para saber quando o jogador apanha recargas de balas

\*/

public void pickAmmoBox() {

Actor bulletsBox = getOneObjectAtOffset(0, 0, BulletsBox.class);

if (bulletsBox != null) {

/\*\*

\* Quando o jogador apanhar uma recarga

\* sera adicionado 25 balas a arma

\*/

bulletsInfo.increaseBullets();

getWorld().removeObject(bulletsBox);

}

}

/\*\*

\* Método para alterar a imagem do jogador quando morrer

\*/

public void dead(){

if(isDead){

setImage(playerDead);

}

}

/\*\*

\* Este método é usado na classe healthbar

\* e quando a vida do jogador for zero

\* é retornado o argumento a true

\*/

public void isPlayerDead(Boolean isDead){

this.isDead = isDead;

}

/\*\*

\* Método para retornar se o jogador esta vivo ou morto

\*/

public boolean isDead(){

return isDead;

}

/\*\*

\* Método para retornar se o jogador foi atacado por um alien

\*/

public boolean getIntersectingAlien() {

return intersectAlien;

}

/\*\*

\* Método para retornar se o jogador foi atingido pelo meteoro

\*/

public boolean getIntersectingMeteor() {

return intersectMeteor;

}

/\*\*

\* Método para retornar se o jogador apanhou a recarga da vida

\*/

public boolean getIntersectingHealthSuply() {

return intersectHealthSuply;

}

}

# Classe **Stats**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Superclasse

\*/

public class Stats extends Actor

{

/\*\*

\* Act - do whatever the Stats wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

private int size;

private String label;

/\*\*

\* Construtor

\*/

public Stats(int size, String label){

this.size = size;

this.label = label;

}

/\*\*

\* Getter size

\*/

public int getSize(){

return size;

}

/\*\*

\* setter size

\*/

public void setSize(int statSize){

this.size = statSize;

}

/\*\*

\* getter label

\*/

public String getLabel(){

return label;

}

/\*\*

\* setter label

\*/

public void setLabel(String statLabel){

this.label = statLabel;

}

public void act()

{

}

}

# Classe **BulletsInfo**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* A classe score é uma subclasse da superclasse Stats

\*/

public class BulletsInfo extends Stats

{

/\*\*

\* Act - do whatever the Score wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

private int bullets;

private Player player;

public BulletsInfo(int size, String label, Player player){

super(size, label);

this.player = player;

setImage(new GreenfootImage(label + ": " + bullets, size, Color.WHITE, new Color (0,0,0,0)));

}

public void act()

{

setImage(new GreenfootImage(getLabel() + ": " + bullets, getSize(), Color.WHITE, new Color (0,0,0,0)));

}

/\*\*

\* getter bullets

\*/

public int getBullets(){

return bullets;

}

/\*\*

\* setter bullets

\*/

public void setBullets(int bullets){

this.bullets = bullets;

}

/\*\*

\* decrementa as balas

\*/

public int decreaseBullets(){

return bullets--;

}

/\*\*

\* incrementa as balas em 25 unidades

\*/

public int increaseBullets(){

return bullets = bullets + 25;

}

}

# Classe **PlayerName**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Subclasse da superclasse Stats

\*/

public class PlayerName extends Stats

{

/\*\*

\* Act - do whatever the Score wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

private int playerId;

public PlayerName(int size, String label, int playerId){

super(size, label);

this.playerId = playerId;

/\*\*

\* Imagem para devolver o nome dos Jogadores

\*/

setImage(new GreenfootImage(label + playerId + ": ", size, Color.WHITE, new Color (0,0,0,0)));

}

public void act()

{

}

}

# Class **Score**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* A classe score é uma subclasse da superclasse Stats

\*/

public class Score extends Stats

{

/\*\*

\* Act - do whatever the Score wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

private int score;

public Score(int size, String label){

/\*\*

\* Super é usado para aceder ao construtor da superclasse

\*/

super(size, label);

/\*\*

\* Criação de uma image Com texto para retornar o score

\*/

setImage(new GreenfootImage(getLabel() + ": " + score, getSize(), Color.WHITE, new Color (0,0,0,0)));

}

public void act()

{

/\*\*

\* Colocamos a imagem aqui para ser atualizada conforme o score incrementa

\*/

setImage(new GreenfootImage(getLabel() + ": " + score, getSize(), Color.WHITE, new Color (0,0,0,0)));

}

/\*\*

\* Retorna o score

\*/

public int getScore(){

return score;

}

/\*\*

\* incrementa o score

\*/

public int increaseScore(){

return score++;

}

}

# Classe **Timer**

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* A classe score é uma subclasse da superclasse Stats

\*/

public class Timer extends Stats

{

private int time = 7200;

private boolean timeOver = false;

public Timer(int size, String label){

super(size, label);

setImage(new GreenfootImage(label + ": " + (time)/60, size, Color.WHITE, new Color (0,0,0,0)));

}

public void act()

{

timer();

setImage(new GreenfootImage(getLabel() + ": " + (time)/60, getSize(), Color.WHITE, new Color (0,0,0,0)));

isTimeOver();

}

/\*\*

\* Getter time

\*/

public int getTime(){

return time;

}

/\*\*

\* Método para saber quando o timer acabou

\*/

public void isTimeOver(){

if(time == 0){

timeOver = true;

}else{

timeOver = false;

}

}

/\*\*

\* O metodo devolve se o timer ja acabou ou nao

\*/

public boolean getTimeOver(){

return timeOver;

}

/\*\*

\* Metodo serve para decrementar o tempo

\*/

public void timer(){

if(time > 0){

time--;

}

}

}