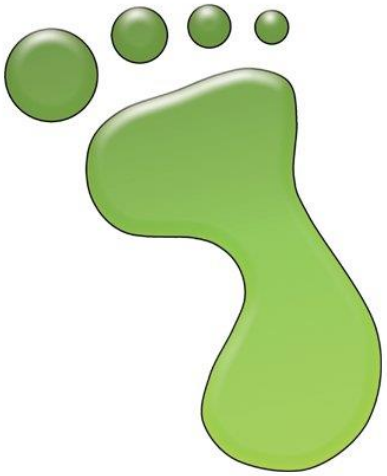
UNIVERSIDADE da MADEIRA

Faculdade de Ciências Exatas e da Engenharia

2024/2025

Programação Orientada por Objetos



THE LAST OF US

Professores:

Luis Ferreira

Yuri Almeida

Sergi Bermúdez

Trabalho elaborado por:

Nelson Macedo nº 2143723
Leandro Rodrigues nº 2104123
Manuel Gama nº 2106723

Funchal, 25 de outubro de 2024

1. Introdução:

Este relatório tem como objetivo demonstrar como foram aplicados os conhecimentos adquiridos na UC de Programação Orientada por Objetos para o desenvolvimento de um jogo cooperativo usando a plataforma *Greenfoot* sob o tema ***The Last of Us.***

Adicionalmente, explicará a ideia principal do grupo tal como a forma de implementação e os procedimentos realizados para a sua execução.

2. Ideia Principal do Jogo:

A ideia principal do grupo foi criar um jogo dividido em 3 níveis/fases que têm de ser completados sequencialmente para ganhar o jogo. O primeiro nível consiste em matar zombies e chegar ao edifício. O segundo nível consiste em matar lenhadores. Por fim, o último nível consiste em matar o "boss", personagem mais forte do jogo.

3.Procedimento e Implementação do código:

3.1. Classe World

3.1.1. SubClasse "Menu":

Este mundo permite aos jogadores iniciar o jogo, aceder às instruções do jogo ou sair do jogo.

Nesta subclasse tem-se implementado um construtor que não recebe parâmetros. Este construtor define um mundo com tamanho 1200 células por 700 células, em que cada célula mede 1 pixel por 1 píxel.

O método playFundo toca e ajusta o volume da música definida para o jogo. O colocaOpcçoes cria 3 opções no menu e posiciona-os em coordenadas específicas da tela. O método mostrarTitulo exibe o título do jogo, definindo o tamanho e cor da letra, juntamente com a sua posição na tela. O started é chamado automaticamente quando o mundo é iniciado, tocando a música em loop. O método stopped é chamado quando pomos em pausa o mundo, e com ele a música de fundo também pára.

Por fim, o método getMusica é um método estático que retorna a música de fundo atual, permitindo que outras classes e mundos acessem o som do menu.

3.1.2. Subclasse "Instruções":

Este mundo permite aos jogadores conhecer as teclas que os permite jogar.

Nesta subclasse tem-se implementado um construtor idêntico ao da subclasse Menu.

O método ColocaInstruções posiciona os textos e icónes, representados por objetos da classe Caixa, que indicam as ações para cada jogador.

3.2. Classe Actor:

3.2.1.Subclasse “Back”:

Esta subclasse tem um método rato que regista se o utilizador selecionou o objeto e criou um mundo novo da classe Menu. O método act chama o método rato.

3.2.2. Subclasse “Boss”:

O metodo prepararImagens() carrega todas as imagens do boss, entre elas ambos os ataques. O metodo animaataque1() e animaataque2(), estes 2 metodos percorrem o array para que possa animar o Boss. O metodo andar(), este é o metodo que faz com que o boss ande, com uma chance de 5%. Seguindo o metodo verificarEAtacar(), faz com o ataque um tenha uma chance de 10% de fazer o ataque 1, e 90% do ataque 2.

Já o metodo Ataque(int tipoAtaque) é o que reinicia as variaveis para 0. O método getVida(), retorna a vida que o boss contem.

O metodo tiraVida(int valor), é o metodo que tira vida ao Boss. De seguida, setBarraDeVida(VidaBoss barraDeVida), este metodo usa a subclass VidaBoss para chamar a barra de vida do mesmo. Por fim, temos o metodo morre(), que faz com que o boss desapareça e remova a barra de vida.

3.2.2.1 Subclasse “VidaBoss”:

O metodo VidaBoss(Boss boss) inicia a barra de vida e chama o metodo update, que este dá update à mesma, ou seja, faz com que o Boss perca vida.

O método act chama o update, para que sempre que levar um tiro o Boss perca vida visualmente. O Update em si dá o update visual à vida.

3.2.3. Subclasse “Cabeça”:

O método act() chama as unicas 2 funções que estão nesta subclasse, mover() e colidir(), em que esta primeira faz com que a cabeça rode e gire enquanto esta se move na direção do player. Já o colidir(), faz com que a cabeça quando encostar na borda, a cabeça desapareça.

3.2.4. Subclasse “Caixa”:

Esta subclasse contém apenas um metodo, em que este faz com que dentro do menu instruções contenha caixas à volta das palavras escritas, podendo escolher a cor.

3.2.5. Subclasse “Disparo”:

Esta subclasse contém o act(), em que o mesmo chama o método atirar() e também caso esta toque em uma das classes, Boss, Zombie, Zombie2, Lenhador, Predio, Bookshelf ou Prateleira, o objeto desta subclasse desaparece. Já o primeiro método referido faz com que o objeto se mova. Seguem-se quatro métodos semelhantes,

denominados: atingiuBoss(),atingiuzombie1(),atingiuzombie2(),atingiulenhador(), em que cada um deteta se o disparo interceptou com o cada uma das respectivas classes: Boss, Zombie, Zombie2 e Lenhador.

3.2.6. Subclasse “Fumo”

Esta subclasse contém o seu método principal, com o seu proprio nome, que chama o método imagens(), este que carrega todas as imagens do fumo. O método act() faz com que o fumo seja chamado sempre que a classe for chamada. O método fumoanima(), é o que é chamado para passar por todas as imagens do array e fazer a animação do mesmo, também é responsável por remover o mesmo, quando acabar.

3.2.7 Subclasse Missil

Esta subclasse contém o metodo missil(), este metodo inicia e carrega os sons e as imagens, do missil. O metodo act(), que faz com que verifica se já foi ativado ou não a explosão, caso não tenha sido ativado ele chama o metodo mover(). Seguidamente, tem o método animarExplosão(), que percorre o array de imagens e anima a explosão do helicoptero. O método mover() faz com que o missil mova-se até colidir com o helicoptero. Já o último método adicionarNemesis(), faz com que seja adicionado um objeto da classe Nemesis.

3.2.8. Subclasse Nemesis

Dentro desta subclasse podemos encontrar o método principal Nemesis(), que carrega todas os sons e imagens, estas ultimas através do método carregarImagens(). Posteriormente, temos o act(), que caso o Nemesis não pare ele move-se e anima, através de mover() e animar() respetivamente. Por fim, o método mostrarTelaDeFim() que chama a tela final.

3.2.9. Subclasse Plataformas

Esta subclasse contém 4 subclasses, nenhuma contém nenhum código, elas só servem para haver colisoes entre objetos

3.2.10. Subclasse Play

Esta Subclasse contém o seu método próprio, que recebe como argumento o texto a imprimir e o tamanho da fonte. Contém também um de nome atualizarImagem(Color cor), este que define e cola a imagem. Já o metodo act(), vê se o rato passou por cima de algum texto e se sim, muda-o de cor. Finalmente, o rato(), deteta qual dos textos foi selecionado e que a funcionalidade de cada um.

3.2.11. Subclasse Player

Esta é a subclasse mais complexa, pois é o que tem mais diversidade de controlos e complexidade de métodos, existindo por muitas vezes atrasos de leitura de métodos, pois tem 2 subclasses, dentro uma da outra.

Esta classe contém o método, que caso o player1 ou player2 seja tocado por um dos inimigos do jogo este irá perder vida, denominado perdeVidas (Player1 P1) e perdeVidas (Player2 P2). Contém ainda funções que colocam som, caso o player leve dano.

3.2.11.1. Subclasse Player1

Esta subclasse inicia com o método act(), que chama as funções andar() e andar2(), que faz com que o player1 ande nas plataformas e moveis do cenario.

Para a vida do player1 existem 3 funções, getNumeroVidas(), resetNumVidas(), adicionaNumeroVidas(int valor), que respetivamente, retornam o número de vidas, resetam o número de vidas para o inicial e adiciona/subtrai vidas ao player1.

No Score temos os métodos getScore(), resetScore(), adicionaScore(int valor), em que faz com que o score do player seja atualizado, retornando o mesmo,resetando e adicionando pontos.

De seguida, são carregadas as imagens através de prepararImagens().

Por outro lado, andar() e andar2(), fazem com que o player1 ande pelas plataformas configuradas, com colisões. Verifica se o mesmo esta em alguma plataforma/movel, através de verificaplat1(),verificaplat2(),verificaplat3(),verificapredio(),verificaprateleira(),verificaarmario(),verificabookshelf(), caso o mesmo esteja numa plataforma poderá saltar com a jump().

Para haver colisões entre as plataformas/moveis, são usados quedaprateleira(), quedaarmario(), quedabookshelf(), onGround5(), onGround6(), onGround7(), quedaplat1(), quedaplat2(), quedaplat3(), quedapredio(),onGround(), onGround2(), onGround3(), onGround4(), reposicionarSeNaBordaInferior(), que colocam o player1 em cima da plataforma. Em colisaoComParteVerticalprateleira(), colisaoComParteVerticalarmario(), colisaoComParteVerticalbookshelf(),colisaoComParteVertical(), colisaoComParteVerticalplat2(), colisaoComParteVerticalplat3(), colisaoComParteVerticalpredio(), são usadas as colisões para a parte vertical dos objetos, o mesmo para as mesmas funções com a palavra “esquerda” à frente.

O método atirar quando é apertado “L” faz a animação de atirar e depois o disparo com os seus sons através de atirar(). Por fim, a função reposicionarSeNaBordaInferior(), faz com que o player caso ocorra um bug e cai todas as plataformas ele sobe para o chão.

3.2.11.1.1 Subclasse Vida_Player1

Esta subclasse contém o método que atualiza a imagem das vidas em vidaPlayer1(), estas mesmas que são carregadas em Vida_Player1().

3.2.11.2. Subclasse Player2

Esta subclasse é praticamente igual a Player1, a única coisa que muda são os botões de movimento que são “W”, “A”, “D” e “E” para atirar.

3.2.11.2.1 Subclasse Vida_Player2

A Vida_Player2 contém o mesmo comportamente de Vida_Player1, só que é utilizada pelo Player2, em vez do Player1.

3.2.12. Subclasse Prateleira

Prateleira, serve exclusivamente como objeto de colisão, em que o mesmo não contém código no seu interior.

3.2.13. Subclasse Predio

Predio, assim como prateleira, serve basicamente para colisão, ou seja serve como plataforma para os players usarem.

3.2.14. Subclasse Restart

Esta classe faz com que seja reiniciado o jogo. Contém o método Restart() que carrega a imagem, depois contém o act(), em que "lê" o movimento do rato. Já o método rato(), faz com que seja colocado o nivel para voltar a ser jogado.

3.2.15. Subclasse Score

Score faz com que seja adicionado pontos a cada colisão com os inimigos. O mesmo contém Score(int inicial), em que o mesmo atualiza a pontuação no mundo. depois adicionarPontos(int valor), este método adiciona valores à pontuação. Temos ainda o método atualizarImagem() que faz com que o socre atualize. Por fim, getPontos() retorna os pontos que o player contém.

3.2.16. Subclasse Texto

Em texto, contém o método Texto(String text, int size, Color corTexto, Color corFundo), em que o mesmo carrega o tamanho e texto da imagem. Já atualizarImagem(String text, int size), atualiza a imagem.

3.2.17. Subclasse Trapdoor

Esta subclasse é a conexão entre o Nivel 1 e 2, quando os players chegam à mesma passam de nivel. A mesma apenas contém apenas duas funções, a porta(), que deteta se ambos os players estão a tocar na trapdoor e que inicia o Nivel2.

3.2.18. Subclasse Trapdooraberta

Este apenas serve como objeto animador, quando a Trapdoor for tocada pelos players, esta fica aberta.

3.2.19. Subclasse Zombie

Zombie, é um inimigo do jogo, o mesmo quando passa pelo player faz com que ele perca vida. Este contém diversos métodos, começando pelo imagens(), que carrega todas as imagens relacionadas ao inimigo.

Seguido disso contém moveZombiePlataforma1(), moveZombiePlataforma2() e moveZombiePlataforma3(), estes que fazem com que o zombie ande caso sejam verificadas as condições de verificaPlataforma1(), verificaPlataforma2() e verificaPlataforma3(), senão o mesmo fica parado, caso esteja fora de uma plataforma. Caso o zombie chegue às bordas de uma plataforma, os métodos atBordaDireita(), atBordaDireita2(), atBordaDireita3(), para a direita e, atBordaEsquerda(), atBordaEsquerda2() e atBordaEsquerda3(), para a esquerda. O zombie, contém vida, definida pelos métodos: getVida(),tiraVida(int valor), que um retorna a vida do zombie, enquanto o outro faz com q ele perca vida. No fim, mas não menos importante, quando o zombie fica sem vida o método morre() faz com que o zombie seja removido do mundo.

3.2.20. Subclasse Zombie2

A subclasse Zombie2, é exatamente igual à subclasse Zombie, a única mudança é a skin do zombie, ou seja a aparência do mesmo.

4.Conceitos de POO

4.1.Encapsulamento

O encapsulamento é um princípio da programação orientada a objetos (POO) que consiste em esconder os detalhes internos de uma classe, expondo apenas o necessário para seu uso externo. A ideia central é proteger os dados da classe e fornecer uma interface controlada para acessá-los. Abaixo um exemplo de encapsulamento, em que as variáveis são declaradas como privadas e que os métodos são todos públicos.

```
private GreenfootImage[] fumo;
private int indice;
private int counter;
public Fumo()
{
    imagens();

}
public void act()
{

    fumoanima();
}
/**
 * Função que prepara as imagens de fumo
 */
public void imagens(){
    fumo= new GreenfootImage[5];

    fumo[0]= new GreenfootImage("fumo.png");
    fumo[1]= new GreenfootImage("fumo2.png");
    fumo[2]= new GreenfootImage("fumo3.png");
    fumo[3]= new GreenfootImage("fumo4.png");
    fumo[4]= new GreenfootImage("fumo5.png");
}
/**
 * Função que percorre o array de imagens, em que anima o fundo e depois desaparece
 */
public void fumoanima(){
    indice++;
    counter++;
        if (indice >=fumo.length)
        {
            indice =0;
        }
    setImage(fumo[indice]);
```

4.2.Overloading

O overloading (ou sobrecarga de métodos) em programação orientada a objetos permite que uma classe tenha mais de um método com o mesmo nome, contanto que esses métodos tenham assinaturas diferentes (ou seja, variem nos tipos ou quantidades de parâmetros).

Exemplo de overloading são as duas funções perdeVidas, em que ambas têm argumentos diferentes mas têm o mesmo nome.

```java
/**
 * Função que faz com que o player2 perca vidas
 */
protected void perdeVidas(Player2 P2)
{
    if(isTouching(Zombie.class)||isTouching(Zombie2.class))
    {
        playAtingido();
        P2.adicionaNumeroVidas(-1);
        P2.adicionaScore(50);

    }
    if(isTouching(Lenhador.class)){
        playAtingido();
        P2.adicionaNumeroVidas(-2);
        P2.adicionaScore(100);
    }
    Cabeca cabeca = (Cabeca) getOneIntersectingObject(Cabeca.class);
    if (cabeca != null) {
        playAtingido();
        P2.adicionaNumeroVidas(-3);
        P2.adicionaScore(500);
        getWorld().removeObject(cabeca);
    }
    if(isTouching(Boss.class)){
        playAtingido();
        P2.adicionaNumeroVidas(-6);
        P2.adicionaScore(500);
    }
}
```

```
/**
 * Função que faz com que o player1 perca vidas
 */
protected void perdeVidas(Player1 P1)
{
        if(isTouching(Zombie.class)||isTouching(Zombie2.class))
        {
            playAtingido();
            P1.adicionaNumeroVidas(-1);
            P1.adicionaScore(50);

        }
        if(isTouching(Lenhador.class)){
            playAtingido();
            P1.adicionaNumeroVidas(-2);
            P1.adicionaScore(100);
        }
        Cabeca cabeca = (Cabeca) getOneIntersectingObject(Cabeca.class);
        if (cabeca!= null) {
            playAtingido();
            P1.adicionaNumeroVidas(-3);
            P1.adicionaScore(500);
             getWorld().removeObject(cabeca);
        }
        if(isTouching(Boss.class)){
            playAtingido();
            P1.adicionaNumeroVidas(-6);
            P1.adicionaScore(100);
        }
}
```

Overriding

Overriding (ou sobrescrita de métodos) é um conceito da programação orientada a objetos que permite a uma classe filha redefinir um método herdado de sua classe pai. O objetivo é personalizar o comportamento do método para a classe derivada, enquanto mantém o nome e a assinatura do método original.

Infelizmente no nosso projeto não conseguimos implementar

Conclusão

Este projeto nos permitiu aprofundar o conhecimento na ferramenta de software "Greenfoot" e na linguagem de programação Java, ampliando os conceitos previamente abordados nas aulas de Programação Orientada a Objetos, como encapsulamento, sobrecarga (overloading) e sobrescrita de métodos (overriding).

Além disso, ofereceu aos alunos a oportunidade de aprimorar suas práticas de trabalho em equipe, incentivando a colaboração e o suporte mútuo para resolver bugs e integrar funções e objetos criados por diferentes integrantes.

Dessa forma incutiu dentro dos alunos uma paixão para desenvolver jogos, que hajam mais projetos como este na licenciatura de Engenharia Informática.

Anexos

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

/**
 * Write a description of class MyWorld here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Menu extends World
{
    private static GreenfootSound musica;
    /**
     * Constructor for objects of class MyWorld.
     *
     */

    public Menu()
    {
        super(1200, 700, 1);
        setBackground ("background1.jpg");

        musica = new GreenfootSound("MusicaFundo.mp3");
        colocaOpcoes();
        mostrarTitulo();
        playFundo();

    }
    /**
     * Método que dá play à música de fundo
     */
    public void playFundo()
```

```
{
    musica.play();
    musica.setVolume(0);
}
/**
 * Método que coloca as opções para poder jogar
 */
private void colocaOpcoes()
{
    Play opcaoPlay = new Play("Play",70);
    Play opcaoOpcoes = new Play("Instruções",70);
    Play opcaoInstrucoes = new Play("Exit",70);

    addObject(opcaoPlay, 1000, getHeight()/2-120);
    addObject(opcaoOpcoes, 1000,  getHeight()/2-40);
    addObject(opcaoInstrucoes, 1000, getHeight()/2+40);

}
/**
 * Método que coloca o titulo do jogo
 */
private void mostrarTitulo()
{
    Color preto = Color.WHITE;

    int tamanhoFonte = 150;
    int posicaoY = 100;
    int posicaoX = 50;
    GreenfootImage titulo1 = new GreenfootImage("THE", tamanhoFonte, preto, new
Color(0, 0, 0, 0));
    getBackground().drawImage(titulo1, posicaoX, posicaoY);
```

```
        GreenfootImage titulo2 = new GreenfootImage("LAST", tamanhoFonte, preto, new
Color(0, 0, 0, 0));
        getBackground().drawImage(titulo2, posicaoX, posicaoY + titulo1.getHeight());


        GreenfootImage titulo3 = new GreenfootImage("OF US", tamanhoFonte, preto, new
Color(0, 0, 0, 0));
        getBackground().drawImage(titulo3, posicaoX, posicaoY + titulo1.getHeight() +
titulo2.getHeight());


    }
    /**
     * Método para fazer loop da musica de fundo
     */
    public void started()
    {
        musica.playLoop();
    }


    public static GreenfootSound getMusica(){
        return musica;
    }


    public void stopped()
    {
        musica.stop();
    }


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


/**
 * Write a description of class Nivel1 here.
 *
```

```
 * @author (your name)
 * @version (a version number or a date)
 */
public class Nivel1 extends World
{

    /**
     * Constructor for objects of class Nivel1.
     *
     */

    private int contadorTempo;
    private GreenfootImage mensagemNivel;
    private static final int TEMPO_VISIVEL = 200;
    private int imageCount = 0;
    private GreenfootImage background = new GreenfootImage("background1.jpg");
    private static final int SCROLL_SPEED = 1;
    private Player1 P1;
    private Player2 P2;
    private boolean control;
    private Score score;
    private Texto scoreP1, scoreP2;
    private int inicial=0;
    public Nivel1(int scoretotal) {
        super(1200, 700, 1);
        setBackground("background1.jpg");
        mensagemNivel = new GreenfootImage("Nível 1", 30, Color.WHITE, new Color(0,
0, 0, 0));
        getBackground().drawImage(mensagemNivel, getWidth() / 2 -
mensagemNivel.getWidth() / 2, 30);
        contadorTempo = 0;
        addObject(new Plataforma1(),getWidth()/2,getHeight()-30);
        Player2 player2 = new Player2();
```

```
    Player1 player1 = new Player1();
    addObject(player2, 10, getHeight() - 100);
    addObject(player1, 45, 591);

    addObject(new Plataforma3(),998, 578);
    addObject(new Predio(),getWidth(),getHeight()-260);
    addObject(new Plataforma2(),575, 509);
    addObject(new Plataforma2(),70, 445);
    addObject(new Plataforma2(),449,302);
    addObject(new Plataforma3(),836,124);



    addObject(new Zombie(), getWidth()/2+150,433);
    addObject(new Zombie(), getWidth()/2-100,227);
    addObject(new Zombie2(), 837,48);
    Trapdoor trapdoor = new Trapdoor();
    addObject(trapdoor,  1103, 238);

    addObject(new Vida_Player1(), 285/2, 125);
    addObject(new Vida_Player2(), getWidth()-285/2, 125);
    score = new Score(inicial);
    addObject(score, getWidth()/2, 10);

}

public void act() {
    contadorTempo++;
    if (contadorTempo == TEMPO_VISIVEL) {
        setBackground("background1.jpg");
    }
    gameOver(P1.getNumeroVidas(),P2.getNumeroVidas());
```

```
    }
    /**
     * Método para quando um dos players morrer
     */
    private void gameOver(int vidaJogador1, int vidaJogador2){
        if ((vidaJogador1 <=0 || vidaJogador2 <= 0) && !control ){
            addObject(new GameOver(),getWidth()/2,getHeight()/2);
            Greenfoot.playSound("GameOver.mp3");

            Restart botaoRestart = new Restart();
            addObject(botaoRestart, getWidth() / 2, getHeight() / 2 + 200);

            control =true;
            resetGame();
        }
    }
    /**
     * Método que dá reset no número de vidas
     */
    public void resetGame()
    {
        Player1.resetNumVidas();
        Player2.resetNumVidas();
    }
    public Score getScore()
    {
        return score;
    }

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

```
/**
 * Write a description of class Nivel2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Nivel2 extends World
{
    private GreenfootImage mensagemNivel;
    private int contadorTempo;
    private static final int TEMPO_VISIVEL = 200;
    private Player1 P1;
    private Player2 P2;
    private boolean control;
    private Score score;
    /**
     * Constructor for objects of class Nivel2.
     *
     */
    public Nivel2(int scoretotal)
    {
        super(1200, 700, 1);
        setBackground("Background2.png");
        mensagemNivel = new GreenfootImage("Nível 2", 30, Color.WHITE, new Color(0,
0, 0, 0));
        getBackground().drawImage(mensagemNivel, getWidth() / 2 -
mensagemNivel.getWidth() / 2, 30);
        addObject(new Armario(),390,660);
        addObject(new Ladder(),1180,130);
        addObject(new Prateleira(),850,385);
        addObject(new Prateleira(),810,462);
        addObject(new Prateleira(),763,538);
        addObject(new Bookshelf(),1020,500);
```

```
    Player2 player2 = new Player2();
    Player1 player1 = new Player1();

    addObject(player2, 92, 565);
    addObject(player1, 139, 565);
    Lenhador lenhador1= new Lenhador();
    Lenhador lenhador2 = new Lenhador();
    addObject(lenhador1,548,591);
    addObject(lenhador2,945,270);
    addObject(new Vida_Player1(), 285/2, 125);
    addObject(new Vida_Player2(), getWidth()-285/2, 125);

    score = new Score(scoretotal);
    addObject(score, getWidth()/2, 10);
}
public void act() {
    contadorTempo++;
    if (contadorTempo == TEMPO_VISIVEL) {
        setBackground("Background2.png");
    }
    gameOver(P1.getNumeroVidas(),P2.getNumeroVidas());

}
/**
 * Método que dá game over quando um dos players morre
 */
private void gameOver(int vidaJogador1, int vidaJogador2){

    if ((vidaJogador1 <=0 || vidaJogador2 <= 0) && !control ){
        addObject(new GameOver(),getWidth()/2,getHeight()/2);
        Greenfoot.playSound("GameOver.mp3");
```

```
            Restart botaoRestart = new Restart();
            addObject(botaoRestart, getWidth() / 2, getHeight() / 2 + 200);

            control =true;
            resetGame();
        }
    }
    /**
     * Método que dá reset no número de vidas do player quando volta o jogo
     */
    public void resetGame()
    {
        Player1.resetNumVidas();
        Player2.resetNumVidas();
    }

    public Score getScore()
    {
        return score;
    }

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

/**
 * Write a description of class Nivel3 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Nivel3 extends World
```

```
{
    private VidaBoss vidaBoss;
    private Boss boss;
    private int contador;
    private boolean control;
    private GreenfootImage mensagemNivel;
    private int contadorTempo;
    private static final int TEMPO_VISIVEL = 200;
    private Score score;
    private Player1 P1;
    private Player2 P2;
    /**
     * Constructor for objects of class Nivel3.
     *
     */
    public Nivel3(int scoretotal)
    {
        super(1200, 700, 1);
        setBackground("Backgroundfim.png");
        mensagemNivel = new GreenfootImage("Nível Final", 100, Color.WHITE, new
Color(0, 0, 0, 0));
        getBackground().drawImage(mensagemNivel, getWidth() / 2 -
mensagemNivel.getWidth() / 2, getHeight()/2-200);


        addObject(new Plataforma1(),getWidth()/2,getHeight()-30);
        addObject(new Plataformafim(),getWidth()/2,getHeight()-45);
        addObject(new Player2(),10,getHeight()-100);
        addObject(new Player1(),30,getHeight()-100);
        boss = new Boss();
        boss.setBarraDeVida(vidaBoss);
        vidaBoss = new VidaBoss(boss);
        addObject(boss, 1000,570);
        addObject(vidaBoss,getWidth()/2,40);
```

```
    addObject(new Vida_Player1(), 285/2, 125);
    addObject(new Vida_Player2(), getWidth()-285/2, 125);

    score = new Score(scoretotal);
    addObject(score, getWidth()/2, 10);
}
/**
 * Método que dá a tela de gameover quando um dos players morre
 */

private void gameOver(int vidaJogador1, int vidaJogador2){

    if ((vidaJogador1 <=0 || vidaJogador2 <= 0) && !control ){
        addObject(new GameOver(),getWidth()/2,getHeight()/2);
        Greenfoot.playSound("GameOver.mp3");

        Restart botaoRestart = new Restart();
        addObject(botaoRestart, getWidth() / 2, getHeight() / 2 + 200);

        control =true;
        resetGame();
    }
}
/**
 * Método que dá reset no numero de vidas de cada personagem
 */
public void resetGame()
{
    Player1.resetNumVidas();
    Player2.resetNumVidas();
}
/**
```

```
     * Método que verifica que o boss morreu e chama o helicoptero
     */
    public void bossMorreu(){
        if (getObjects(Heli.class).isEmpty()) {
            Heli helicoptero = new Heli();
            addObject(helicoptero, 1300, 100);
            helicoptero.ativarResgate();
        }
    }
    public Score getScore()
    {
        return score;
    }
    public void act() {

        gameOver(P1.getNumeroVidas(),P2.getNumeroVidas());

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

/**
 * Write a description of class Armario here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Armario extends Actor
{
    /**
     * Act - do whatever the Armario wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
```

```
     */
    public void act()
    {
        // Add your action code here.
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Back here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Back extends Actor
{


    private GreenfootImage imagem;

    public Back(){
        imagem = new GreenfootImage("back_button.png");
        setImage(imagem);
    }
    public void act()
    {
        if(Greenfoot.mouseClicked(this)){
            rato();
        }
    }
    /**
     * Método que verifica se o rato clicou numa imagem
```

```
     */
    public void rato(){
        if (getImage() == imagem) {
            Greenfoot.setWorld(new Menu());
        }
    }

}

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

/**
 * Write a description of class Bookshelf here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Bookshelf extends Actor
{
    /**
     * Act - do whatever the Bookshelf wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.util.Random;
import java.util.List;
```

```
/**
 * Write a description of class Bosss here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Boss extends Actor
{
    /**
     * Act - do whatever the Boss wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private Random randomGenerator;
    private GreenfootImage[] ataque1esquerda;
    private GreenfootImage[] ataque2esquerda;
    private GreenfootImage[] fingir;
    private GreenfootImage andar;
    private GreenfootImage preataque;
    private GreenfootImage preataqueesquerda;
    private GreenfootImage parado;
    private int indice=0;
    private int contador;
    private int contadorAnimacao;
    private boolean atacarEsquerda = false;
    private boolean emAtaque = false;
    private boolean ataque1Ativo;
    private Cabeca cabeca;
    private static int vida;
    private VidaBoss barraDeVida;
    private int ataqueAtivo;
    private int contadorTempoAtaque;
    private GreenfootSound somDano;
```

```
private GreenfootSound gritoDeMorte;
private boolean jaGritou = false;

public Boss() {
    prepararImagens();
    cabeca = new Cabeca();
    vida = 10;
    randomGenerator = new Random();
    somDano = new GreenfootSound("somDano.mp3");
    gritoDeMorte = new GreenfootSound("BossMorte.mp3");
}
public void act() {
    if (vida <= 0) {
        morre();
    } else {
        if(vida <=2 && !jaGritou){
                gritoDeMorte.play();
                jaGritou = true;
        }
        if (emAtaque) {
            if (ataqueAtivo == 1) {
                animaataque1();
            } else {
                animaataque2();
            }
        } else {
            verificarEAtacar();
        }
        if (contadorTempoAtaque < 100) {
            contadorTempoAtaque++;
        }
    }
```

```
}
/**
 * Método que carrega as imagens para os arrays das mesmas
 */
public void prepararImagens() {

    ataque1esquerda = new GreenfootImage[7];
    ataque2esquerda = new GreenfootImage[8];
    fingir = new GreenfootImage[5];
    andar = new GreenfootImage("andaresquerda.png");
    parado = new GreenfootImage("paradoesquerda.png");
    preataqueesquerda = new GreenfootImage("preataqueesquerda.png");


    ataque1esquerda[0] = preataqueesquerda;
    ataque1esquerda[1] = new GreenfootImage("ataque1.1esquerda.png");
    ataque1esquerda[2] = new GreenfootImage("ataque1.2esquerda.png");
    ataque1esquerda[3] = new GreenfootImage("ataque1.3esquerda.png");
    ataque1esquerda[4] = new GreenfootImage("ataque1.4esquerda.png");
    ataque1esquerda[5] = new GreenfootImage("ataque1.5esquerda.png");
    ataque1esquerda[6] = new GreenfootImage("ataque1.6esquerda.png");

    ataque2esquerda[0] = preataqueesquerda;
    ataque2esquerda[1] = new GreenfootImage("ataque2.1esquerda.png");
    ataque2esquerda[2] = new GreenfootImage("ataque2.2esquerda.png");
    ataque2esquerda[3] = new GreenfootImage("ataque2.3esquerda.png");
    ataque2esquerda[4] = new GreenfootImage("ataque2.4esquerda.png");
    ataque2esquerda[5] = new GreenfootImage("ataque2.5esquerda.png");
    ataque2esquerda[6] = new GreenfootImage("ataque2.6esquerda.png");
    ataque2esquerda[7] = new GreenfootImage("ataque2.7esquerda.png");

}
```

```
/**
 * Função que percorre os arrays de imagens
 */
public void animaataque1() {
    contadorAnimacao++;
    if (contadorAnimacao >= 20 && indice < ataque1esquerda.length) {
        indice++;
        setImage(ataque1esquerda[indice]);
        contadorAnimacao = 0;
    }

    if (indice == ataque1esquerda.length - 1) {
        emAtaque = false;
        setImage(parado);
    }
}
public void animaataque2()
{
    contadorAnimacao++;
    if (contadorAnimacao >= 20 && indice < ataque2esquerda.length) {
        indice++;
        setImage(ataque2esquerda[indice]);
        contadorAnimacao = 0;

    }

    if (indice == ataque2esquerda.length - 2) {
        int larguraImagem = ataque2esquerda[ataque2esquerda.length - 2].getWidth();
        int posicaoXCabeca = getX() - larguraImagem / 2;
        int posicaoYCabeca = getY();
        if (getWorld().getObjects(Cabeca.class).isEmpty()) {
            getWorld().addObject(cabeca, posicaoXCabeca, posicaoYCabeca);
```

```
        }
    }
    if (indice == ataque2esquerda.length - 1) {
        emAtaque = false;
        setImage(parado);
        indice = 0;
    }
}
/**
 * método que decide que o boss vai andar, com 1% de chance
 */
public void andar() {
    contador++;
    int numero = Greenfoot.getRandomNumber(100);
    if (numero < 5) {
        setLocation(getX() - 5, getY());
    }
    if (contador >= 10) {
        contador = 0;
    }
}
/**
 * Metodo que decide qual dos ataques o boss vai usar, um com 10%, outro com 90%
 */
 public void verificarEAtacar() {
    int numeroAleatorio = randomGenerator.nextInt(100) + 1;
        if (numeroAleatorio <= 10) {
            Ataque(1);
        } else {
            Ataque(2);
        }
```

```
    andar();
}
/**
 * Função ajudante, reinicia as variaveis
 */
public void Ataque(int tipoAtaque) {
    contador = 0;
    emAtaque = true;
    indice = 0;
    contadorAnimacao = 0;
    ataqueAtivo = tipoAtaque;
    contadorTempoAtaque = 0;
}

/**
 *
 */
public static int getVida()
{
    return vida;
}
/**
 * Método que tira vida ao boss e dá o som de dano ao mesmo
 */
public static void tiraVida(int valor)
{
    vida-=valor;
    GreenfootSound somDano = new GreenfootSound("somDano.mp3");
    somDano.play();
}

public void setBarraDeVida(VidaBoss barraDeVida) {
```

```
        this.barraDeVida = barraDeVida;
    }
    /**
     * Função que faz com que o boss morra
     */
    public void morre(){
            if (barraDeVida != null){
                getWorld().removeObject(barraDeVida);
            }
            ((Nivel3) getWorld()).bossMorreu();
            getWorld().removeObject(this);
        }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class VidaBoss here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class VidaBoss extends Boss
{
    private int Largurahealthbar = 800;
    private int Alturahealthbar = 20;
    private int vida = Boss.getVida();
    private int Vidapercentagem = (int)Largurahealthbar/vida;
    public VidaBoss(Boss boss)
    {
        boss.setBarraDeVida(this);
        update();
    }
```

```
    public void act()
    {
        update();
    }

    public void update()
    {
        setImage(new GreenfootImage(Largurahealthbar + 2, Alturahealthbar + 2));
        GreenfootImage myImage = getImage();
        vida = Boss.getVida();
        myImage.setColor(Color.WHITE);
        myImage.drawRect(0, 0, Largurahealthbar + 1, Alturahealthbar + 1);
        myImage.setColor(Color.RED);
        myImage.fillRect(1, 1, vida * Vidapercentagem, Alturahealthbar);
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Cabeca here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Cabeca extends Actor
{
    /**
     * Act - do whatever the Cabeca wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private int anguloRotacao = 0;
```

```
    public void act()
    {
        mover();
        colidir();
    }
    /**
     * Função que move a cabeça quando esta é inserida no mundo
     */
    public void mover()
    {
        setLocation(getX() - 10, getY());
        anguloRotacao += 10;
        if (anguloRotacao >= 360) {
            anguloRotacao = 0;
        }
        setRotation(anguloRotacao);
    }
    /**
     * Função que faz com que a cabeça desapareça quando colidir com algo
     */
    public void colidir()
    {
        if (isAtEdge()){
            {
                getWorld().removeObject(this);
            }
        }
    }

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

```
/**
 * Write a description of class Caixa here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Caixa extends Actor
{
    /**
     * Act - do whatever the Caixa wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private Boolean flag;
    private Boolean prepara;
    private int player;
    private int controlo;
    private GreenfootImage image;
    /**
     * Função que coloca as imagens das teclas no World Instruções
     */
    public Caixa(int player, int controlo, String nomeImagem) {
        this.flag = true;
        this.prepara = true;
        this.player = player;
        this.controlo = controlo;

        this.image = new GreenfootImage(nomeImagem);
        setImage(image);
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
```

```
/**
 * Write a description of class municao here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Disparo extends Actor
{
    /**
     * Act - do whatever the municao wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */




    public void act() {

        atirar();

        if (isAtEdge() || atingiuBoss()||atingiuzombie1()||atingiuzombie2()||atingiulenhador()||isTouching(Predio.class)||isTouching(Bookshelf.class)||isTouching(Prateleira.class)) {
            getWorld().removeObject(this);
        }
    }
    /**
     * Função que faz com que a bala se mova
     */
    public void atirar() {
        move(15);

    }
```

```
    /**
     * Função que faz com que o disparo dê dano ao Boss
     */
    private boolean atingiuBoss()
    {
        if (isTouching(Boss.class) && getWorld().getObjects(Boss.class).get(0).getVida()
> 0)
        {
            getWorld().getObjects(Boss.class).get(0).tiraVida(1);
            Nivel3 Nivel3 = (Nivel3) getWorld();
            Nivel3.getScore().adicionarPontos(3000);
            return true;
        }
        return false;
    }
    /**
     * Função que faz com que o disparo dê dano ao Zombie1
     */
    private boolean atingiuzombie1()
    {
        if                    (isTouching(Zombie.class)                    &&
getWorld().getObjects(Zombie.class).get(0).getVida() > 0)
        {
            getWorld().getObjects(Zombie.class).get(0).tiraVida(1);
            Nivel1 Nivel1 = (Nivel1) getWorld();
            Nivel1.getScore().adicionarPontos(300);
            return true;
        }
        return false;
    }
    /**
     * Função que faz com que o disparo dê dano ao Zombie2
     */
```

```
    private boolean atingiuzombie2()
    {
        if (isTouching(Zombie2.class) && getWorld().getObjects(Zombie2.class).get(0).getVida() > 0)
        {
            getWorld().getObjects(Zombie2.class).get(0).tiraVida(1);
            Nivel1 Nivel1 = (Nivel1) getWorld();
            Nivel1.getScore().adicionarPontos(300);
            return true;
        }
        return false;
    }
    /**
     * Função que faz com que o disparo dê dano ao Lenhador
     */
    private boolean atingiulenhador()
    {
        if (isTouching(Lenhador.class) && getWorld().getObjects(Lenhador.class).get(0).getVida() > 0)
        {
            getWorld().getObjects(Lenhador.class).get(0).tiraVida(1);
            Nivel2 Nivel2 = (Nivel2) getWorld();
            Nivel2.getScore().adicionarPontos(500);
            return true;
        }
        return false;
    }

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

/**
 * Write a description of class Fumo here.
```

```
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Fumo extends Actor
{
    /**
     * Act - do whatever the Fumo wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private GreenfootImage[] fumo;
    private int indice;
    private int counter;
    public Fumo()
    {
        imagens();

    }
    public void act()
    {

        fumoanima();
    }
    /**
     * Função que prepara as imagens de fumo
     */
    public void imagens(){
        fumo= new GreenfootImage[5];

        fumo[0]= new GreenfootImage("fumo.png");
        fumo[1]= new GreenfootImage("fumo2.png");
        fumo[2]= new GreenfootImage("fumo3.png");
```

```
        fumo[3]= new GreenfootImage("fumo4.png");
        fumo[4]= new GreenfootImage("fumo5.png");
    }
    /**
     * Função que percorre o array de imagens, em que anima o fundo e depois desaparece
     */
    public void fumoanima(){
        indice++;
        counter++;
            if (indice >=fumo.length)
            {
                indice =0;
            }
        setImage(fumo[indice]);
        if(counter>10){
            getWorld().removeObject(this);
        }
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class GameOver here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class GameOver extends Actor
{
    /**
     * Act - do whatever the GameOver wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
```

```
     */
    private GreenfootImage pisca1, pisca2;
    private int contador;

    public GameOver(){
        pisca1 = new GreenfootImage("Game_Over1.png");
        pisca2 = new GreenfootImage("Game_Over2.png");
        setImage(pisca1);
        contador = 0;
    }

    public void act()
    {

    }


}
import greenfoot.GreenfootSound;
/**
 * Write a description of class Heli here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Heli extends Actor
{
    private GreenfootSound somHelicoptero;
    private boolean jaTocou = false;
    private int velocidadeX = 2;
    private int velocidadeY = 1;
    private boolean resgateAtivo = false;
```

```
private boolean parado = false;
private GreenfootImage[] imagensHeli;
private int currentImagem = 0;
private int contadorFrames = 0;

public Heli() {
    imagensHeli = new GreenfootImage[4];
    carregarImagens();
    setImage(imagensHeli[currentImagem]);
    somHelicoptero = new GreenfootSound("helicoptero.mp3");
}

/**
 * Função que carrega as imagens do helicóptero
 */
private void carregarImagens(){
    imagensHeli[0] = new GreenfootImage("helicoptero2.png");
    imagensHeli[1] = new GreenfootImage("helicoptero3.png");
    imagensHeli[2] = new GreenfootImage("helicoptero4.png");
    imagensHeli[3] = new GreenfootImage("helicoptero5.png");
}

public void act() {
    if (!jaTocou){
        somHelicoptero.playLoop();
        jaTocou = true;
    }
    if (resgateAtivo) {
        mover();
        animarDescida();
    }
}
```

```
    /**
     * Função que percorre o array de imagens e anima o helicóptero
     */
    private void animarDescida() {
        contadorFrames++;
        if (contadorFrames >= 5) {
            currentImagem = (currentImagem + 1) % imagensHeli.length;
            setImage(imagensHeli[currentImagem]);
            contadorFrames = 0;
        }
    }
    /**
     *  Função que atualiza a variável resgate
     */
    public void ativarResgate() {
        resgateAtivo = true;
    }
    /**
     * Função que move o helicóptero pelo mundo até chegar ao centro e depois chama o
missil
     */
    public void mover() {
        int centroX = getWorld().getWidth() / 2;
        int centroY = getWorld().getHeight() / 2;
        if (!parado) {
            if (getX() > centroX) {
                setLocation(getX() - velocidadeX, getY() + velocidadeY);
            } else if (getX() < centroX) {
                setLocation(getX() + velocidadeX, getY() + velocidadeY);
            }
            if (getX() <= centroX + 10 && getX() >= centroX - 10) {
                parado = true;
```

```
                setLocation(centroX, centroY);
                criarMissil();
            }
        }
    }
    /**
     * Função que remove o helicóptero do mundo e para o som do mesmo
     */
    public void explodir(){
        somHelicoptero.stop();
        getWorld().removeObject(this);
    }
    /**
     * Função que cria o missil no mundo
     */
    private void criarMissil(){
        Missil missil = new Missil();
        getWorld().addObject(missil,getWorld().getWidth() + 50, getY());
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Ladder here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Ladder extends Actor
{
    /**
     * Act - do whatever the Ladder wants to do. This method is called whenever
```

```
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        tocar();
    }
    /**
     * Função que muda de mundo quando ambos os players chegam ao fim do mesmo
     */
    public void tocar(){
        if(isTouching(Player1.class) && isTouching(Player2.class)){

            Nivel2 nivel=(Nivel2) getWorld();
            int totalScore = nivel.getScore().getPontos();
            Greenfoot.setWorld(new Nivel3(totalScore));

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

/**
 * Write a description of class Lenhador here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Lenhador extends Actor
{
    /**
     * Act - do whatever the Lenhador wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
```

```
 */
private GreenfootImage[] andarImagens;
private GreenfootImage[] melee;
private GreenfootImage[] crit;
private GreenfootImage[] morte;
private GreenfootImage[] andarImagensesquerda;
private GreenfootImage[] meleeesquerda;
private GreenfootImage[] critesquerda;
private GreenfootImage[] morteesquerda;
private int indice;
private int vida;
private boolean direita = true;
public Lenhador (){
   prepararImagens();
   vida = 10;
}
public void act()
{
   if (vida <= 0){
      morre();
      return;
   }
   if (verificaLenhador()) {
      moveLenhador();
   }
   if(verificaLenhadorb())
   {
      moveLenhadorb();
   }
      Actor player1 = getPlayerProximo(Player1.class);
      Actor player2 = getPlayerProximo(Player2.class);
```

```
    if (player1 != null || player2 != null) {
        atacar();
    }
}
/**
 * Função que carrega as imagens do lenhador
 */
public void prepararImagens() {

    andarImagens = new GreenfootImage[8];
    melee = new GreenfootImage[10];
    crit = new GreenfootImage[14];
    morte = new GreenfootImage[15];

    andarImagensesquerda = new GreenfootImage[8];
    meleeesquerda = new GreenfootImage[10];
    critesquerda = new GreenfootImage[14];
    morteesquerda = new GreenfootImage[15];

    andarImagens[0] = new GreenfootImage("andar.png");
    andarImagens[1] = new GreenfootImage("andar2.png");
    andarImagens[2] = new GreenfootImage("andar3.png");
    andarImagens[3] = new GreenfootImage("andar4.png");
    andarImagens[4] = new GreenfootImage("andar5.png");
    andarImagens[5] = new GreenfootImage("andar6.png");
    andarImagens[6] = new GreenfootImage("andar7.png");
    andarImagens[7] = new GreenfootImage("andar8.png");

    melee[0] = new GreenfootImage("crit1.png");
    melee[1] = new GreenfootImage("crit2.png");
    melee[2] = new GreenfootImage("crit3.png");
    melee[3] = new GreenfootImage("crit4.png");
```

```
melee[4] = new GreenfootImage("crit5.png");
melee[5] = new GreenfootImage("crit6.png");
melee[6] = new GreenfootImage("crit7.png");
melee[7] = new GreenfootImage("crit8.png");
melee[8] = new GreenfootImage("crit9.png");
melee[9] = new GreenfootImage("crit10.png");

crit[0] = new GreenfootImage("melee.png");
crit[1] = new GreenfootImage("melee2.png");
crit[2] = new GreenfootImage("melee3.png");
crit[3] = new GreenfootImage("melee4.png");
crit[4] = new GreenfootImage("melee5.png");
crit[5] = new GreenfootImage("melee6.png");
crit[6] = new GreenfootImage("melee7.png");
crit[7] = new GreenfootImage("melee8.png");
crit[8] = new GreenfootImage("melee9.png");
crit[9] = new GreenfootImage("melee10.png");
crit[10] = new GreenfootImage("melee11.png");
crit[11] = new GreenfootImage("melee12.png");
crit[12] = new GreenfootImage("melee13.png");
crit[13] = new GreenfootImage("melee14.png");

morte[0]= new GreenfootImage("morte1.png");
morte[1]= new GreenfootImage("morte2.png");
morte[2]= new GreenfootImage("morte3.png");
morte[3]= new GreenfootImage("morte4.png");
morte[4]= new GreenfootImage("morte5.png");
morte[5]= new GreenfootImage("morte6.png");
morte[6]= new GreenfootImage("morte7.png");
morte[7]= new GreenfootImage("morte8.png");
morte[8]= new GreenfootImage("morte9.png");
morte[9]= new GreenfootImage("morte10.png");
```

```
morte[10]= new GreenfootImage("morte11.png");
morte[11]= new GreenfootImage("morte12.png");
morte[12]= new GreenfootImage("morte13.png");
morte[13]= new GreenfootImage("morte14.png");
morte[14]= new GreenfootImage("morte15.png");

andarImagensesquerda[0] = new GreenfootImage("andaresquerda1.png");
andarImagensesquerda[1] = new GreenfootImage("andaresquerda2.png");
andarImagensesquerda[2] = new GreenfootImage("andaresquerda3.png");
andarImagensesquerda[3] = new GreenfootImage("andaresquerda4.png");
andarImagensesquerda[4] = new GreenfootImage("andaresquerda5.png");
andarImagensesquerda[5] = new GreenfootImage("andaresquerda6.png");
andarImagensesquerda[6] = new GreenfootImage("andaresquerda7.png");
andarImagensesquerda[7] = new GreenfootImage("andaresquerda8.png");

meleeesquerda[0] = new GreenfootImage("meleeesquerda1.png");
meleeesquerda[1] = new GreenfootImage("meleeesquerda2.png");
meleeesquerda[2] = new GreenfootImage("meleeesquerda3.png");
meleeesquerda[3] = new GreenfootImage("meleeesquerda4.png");
meleeesquerda[4] = new GreenfootImage("meleeesquerda5.png");
meleeesquerda[5] = new GreenfootImage("meleeesquerda6.png");
meleeesquerda[6] = new GreenfootImage("meleeesquerda7.png");
meleeesquerda[7] = new GreenfootImage("meleeesquerda8.png");
meleeesquerda[8] = new GreenfootImage("meleeesquerda9.png");
meleeesquerda[9] = new GreenfootImage("meleeesquerda10.png");

critesquerda[0] = new GreenfootImage("critesquerda1.png");
critesquerda[1] = new GreenfootImage("critesquerda2.png");
critesquerda[2] = new GreenfootImage("critesquerda3.png");
critesquerda[3] = new GreenfootImage("critesquerda4.png");
critesquerda[4] = new GreenfootImage("critesquerda5.png");
critesquerda[5] = new GreenfootImage("critesquerda6.png");
```

```
    critesquerda[6] = new GreenfootImage("critesquerda7.png");
    critesquerda[7] = new GreenfootImage("critesquerda8.png");
    critesquerda[8] = new GreenfootImage("critesquerda9.png");
    critesquerda[9] = new GreenfootImage("critesquerda10.png");
    critesquerda[10] = new GreenfootImage("critesquerda11.png");
    critesquerda[11] = new GreenfootImage("critesquerda12.png");
    critesquerda[12] = new GreenfootImage("critesquerda13.png");
    critesquerda[13] = new GreenfootImage("critesquerda14.png");

    morteesquerda[0]= new GreenfootImage("morteesquerda1.png");
    morteesquerda[1]= new GreenfootImage("morteesquerda2.png");
    morteesquerda[2]= new GreenfootImage("morteesquerda3.png");
    morteesquerda[3]= new GreenfootImage("morteesquerda4.png");
    morteesquerda[4]= new GreenfootImage("morteesquerda5.png");
    morteesquerda[5]= new GreenfootImage("morteesquerda6.png");
    morteesquerda[6]= new GreenfootImage("morteesquerda7.png");
    morteesquerda[7]= new GreenfootImage("morteesquerda8.png");
    morteesquerda[8]= new GreenfootImage("morteesquerda9.png");
    morteesquerda[9]= new GreenfootImage("morteesquerda10.png");
    morteesquerda[10]= new GreenfootImage("morteesquerda11.png");
    morteesquerda[11]= new GreenfootImage("morteesquerda12.png");
    morteesquerda[12]= new GreenfootImage("morteesquerda13.png");
    morteesquerda[13]= new GreenfootImage("morteesquerda14.png");
    morteesquerda[14]= new GreenfootImage("morteesquerda15.png");
}
/**
 * Função que percorre o array de imagens e anima o lenhador
 */
public void animaandar()
{
    indice++;
        if (indice >=andarImagens.length)
```

```
        {
            indice =0;
        }
    setImage(andarImagens[indice]);

}
/**
 * Função que percorre o array de imagens e anima o ataque critico
 */
public void animacrit()
{
    indice++;
        if (indice >=crit.length)
        {
            indice =0;
        }
    setImage(crit[indice]);

}
/**
 * Função que percorre o array de imagens e anima o ataque melee
 */
public void animamelee()
{
    indice++;
        if (indice >=melee.length)
        {
            indice =0;
        }
    setImage(melee[indice]);

}
```

```
/**
 * Função que percorre o array de imagens e anima a morte
 */
public void animamorte()
{
   indice++;
      if (indice >=morte.length)
      {
         indice =0;
      }
   setImage(morte[indice]);

}
/**
 * Função que percorre o array de imagens e anima o andar para o lado esquerdo
 */
public void animaandaresquerda()
{
   indice++;
      if (indice >=andarImagensesquerda.length)
      {
         indice =0;
      }
   setImage(andarImagensesquerda[indice]);

}
/**
 * Função que percorre o array de imagens e anima a morte para o lado esquerdo
 */
public void animamorteesquerda()
{
   indice++;
```

```
        if (indice >=morteesquerda.length)
        {
            indice =0;
        }
    setImage(morteesquerda[indice]);

}
/**
 * Função que percorre o array de imagens e anima o ataque critico para a esquerda
 */
public void animacritesquerda()
{
    indice++;
        if (indice >=critesquerda.length)
        {
            indice =0;
        }
    setImage(critesquerda[indice]);

}
/**
 * Função que percorre o array de imagens e anima o ataque melee para a esquerda
 */
public void animameleeesquerda()
{
    indice++;
        if (indice >=meleeesquerda.length)
        {
            indice =0;
        }
    setImage(meleeesquerda[indice]);
```

```
    }
    /**
     * Função que verifica se o lenhador esta em cima do armario
     */
    public boolean verificaLenhador() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Armario.class);
        return emCima != null;
    }
    /**
     * Função que verifica se o lenhador chegou à borda direita do armario
     */
    public boolean atBordaDireita() {
        Actor bordaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, getImage().getHeight() / 2, Armario.class);
        return bordaDireita == null;
    }
    /**
     * Função que verifica se o lenhador chegou à borda esquerda do armario
     */
    public boolean atBordaEsquerda() {
        Actor bordaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, getImage().getHeight() / 2, Armario.class);
        return bordaEsquerda == null;
    }
    public boolean verificaLenhadorb() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Bookshelf.class);
        return emCima != null;
    }
    public boolean atBordaDireitab() {
        Actor bordaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, getImage().getHeight() / 2, Bookshelf.class);
        return bordaDireita == null;
```

```
    }

    public boolean atBordaEsquerdab() {
        Actor bordaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, getImage().getHeight() / 2, Bookshelf.class);
        return bordaEsquerda == null;
    }
    /**
     * Função que move o lenhador em cima do armario
     */
    public void moveLenhador() {
        if (atBordaDireita() && direita) {
            direita = false;
        } else if (atBordaEsquerda() && !direita) {
            direita = true;
        }

        if (direita) {
            setLocation(getX() + 10, getY());
            animaandar();
        } else {
            setLocation(getX() - 10, getY());
            animaandaresquerda();
        }
    }
    /**
     * Função que move o lenhador em cima da bookshelf
     */
    public void moveLenhadorb() {
        if (atBordaDireitab() && direita) {
            direita = false;
        } else if (atBordaEsquerdab() && !direita) {
```

```
        direita = true;
    }

    if (direita) {
        setLocation(getX() + 10, getY());
        animaandar();
    } else {
        setLocation(getX() - 10, getY());
        animaandaresquerda();
    }
}
public int getVida()
{
    return vida;
}

public void tiraVida(int valor)
{
    vida-=valor;
}
/**
 * Função que retira o lenhador do mundo
 */
public void morre(){
    if (vida <= 0){
        if (direita) {

            getWorld().removeObject(this);
        }else{

            getWorld().removeObject(this);
        }
```

```
        }
    }
    /**
     * Método que vefica se tem um jogador próximo
     */
    public Actor getPlayerProximo(Class jogadorClass) {
        return getOneObjectAtOffset(100, 0, jogadorClass);
    }
    /**
     * Método que faz com que o lenhador tenha a aniamção de ataque
     */
    public void atacar() {
        int escolhaAtaque = Greenfoot.getRandomNumber(2);
        if (direita) {
            if (escolhaAtaque == 0) {
                animamelee();
            } else {
                animacrit();
            }
        } else {
            if (escolhaAtaque == 0) {
                animameleeesquerda();
            } else {
                animacritesquerda();
            }
        }
    }

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

```
/**
 * Write a description of class Missil here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Missil extends Actor
{
    /**
     * Act - do whatever the missil wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private GreenfootImage[] imagensExplosao;
    private int contadorExplosao = 0;
    private int currentExplosao = 0;
    private boolean emExplosao = false;
    private int posicaoXExplosao;
    private int posicaoYExplosao;
    private boolean explosaoIniciada = false;
    private GreenfootSound somExplosao;

    public Missil(){
        setImage("missil.png");
        imagensExplosao = new GreenfootImage[48];
        for (int i = 0; i < 48; i++) {
            imagensExplosao[i] = new GreenfootImage("explosao" + i + ".png");
            imagensExplosao[i].scale(imagensExplosao[i].getWidth() * 5,
imagensExplosao[i].getHeight() * 5);
        }
        somExplosao = new GreenfootSound("explosao.mp3");
    }
```

```
    public void act()
    {
        if(emExplosao) {
            animarExplosao();
        }else{
            mover();
        }
    }
    /**
     * Função que anima a explosão do missil
     */
    private void animarExplosao() {
        contadorExplosao++;
        if (contadorExplosao >= 5) {
            contadorExplosao = 0;
            currentExplosao++;
            if (currentExplosao < imagensExplosao.length) {
                setImage(imagensExplosao[currentExplosao]);
                setLocation(posicaoXExplosao, posicaoYExplosao);
            } else {
                adicionarNemesis();
                getWorld().removeObject(this);
            }
        }
    }
    /**
     * Função que faz o missil mover-se e quando tocar no helicóptero fazer a animação e
som de explosão
     */
    private void mover() {
        setLocation(getX() - 10, getY());
```

```
        if (isTouching(Heli.class)) {
            Heli helicoptero = (Heli) getOneIntersectingObject(Heli.class);
            if (helicoptero != null) {
                posicaoXExplosao = helicoptero.getX();
                posicaoYExplosao = helicoptero.getY();
                helicoptero.explodir();
                emExplosao = true;
                somExplosao.play();
            }
            emExplosao = true;
        }
    }
    /**
     * Função que spawna o Nemesis
     */
    private void adicionarNemesis() {
        if (getWorld() != null) {
            Nemesis nemesis = new Nemesis();
            getWorld().addObject(nemesis, getWorld().getWidth(), getWorld().getHeight() -
150);
        }
    }



}

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

/**
 * Write a description of class Nemesis here.
 *
 * @author (your name)
```

```
 * @version (a version number or a date)
 */
public class Nemesis extends Actor {
    private GreenfootImage[] imagensNemesis;
    private int currentImagem = 0;
    private int contadorFrames = 0;
    private int ciclosCompletos = 0;
    private final int CICLOS_PARA_PARAR = 1;
    private int velocidadeX = 1;
    private boolean parou = false;
    private int contadorAtraso = 0;
    private final int ATRASO_FIM = 150;
    private GreenfootSound somNemesis;
    public Nemesis() {
        imagensNemesis = new GreenfootImage[4];
        carregarImagens();
        setImage(imagensNemesis[currentImagem]);
        somNemesis = new GreenfootSound("somNemesis.mp3");
        somNemesis.play();
    }
    /**
     * Função que carrega as Imagens do Nemesis
     */
    private void carregarImagens() {
        for (int i = 0; i < 4; i++) {
            imagensNemesis[i] = new GreenfootImage("nemesis" + i + ".png");
            imagensNemesis[i].scale(150, 200);
        }
    }

    public void act() {
        if (!parou) {
```

```
            mover();
            animar();
        } else {
            contadorAtraso++;
            if(contadorAtraso >= ATRASO_FIM){
                mostrarTelaDeFim();
            }
        }
    }
    /**
     * Função que move o Nemesis
     */
    private void mover() {

        if (getX() > getWorld().getWidth()/2) {
            setLocation(getX() - velocidadeX, getY());
        }
    }
    /**
     * Função que percorre o array e anima o Nemesis
     */
    private void animar() {
        contadorFrames++;
        if (contadorFrames >= 40) {
            currentImagem++;
            if (currentImagem >= imagensNemesis.length) {
                currentImagem = 0;
                ciclosCompletos++;
            }
            setImage(imagensNemesis[currentImagem]);
            contadorFrames = 0;
```

```
        if (ciclosCompletos >= CICLOS_PARA_PARAR && currentImagem == imagensNemesis.length -1) {
            parou = true;
        }
      }
   }
   /**
    * Função que mostra o ecrã final
    */
   private void mostrarTelaDeFim(){
      Greenfoot.setWorld(new Fim());
   }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Plataformas here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Plataformas extends Actor
{
   /**
    * Act - do whatever the Plataformas wants to do. This method is called whenever
    * the 'Act' or 'Run' button gets pressed in the environment.
    */
   public void act()
   {
      // Add your action code here.
   }
}
```

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

/**
 * Write a description of class Plataforma here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Plataforma1 extends Plataformas
{
    /**
     * Act - do whatever the Plataforma wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Plataforma2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Plataforma2 extends Plataformas
{
    /**
     * Act - do whatever the Plataforma2 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
```

```
/**
 * Write a description of class Plataforma3 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Plataforma3 extends Plataformas
{
    /**
     * Act - do whatever the Plataforma3 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Plataformafim here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Plataformafim extends Plataformas
{
    /**
     * Act - do whatever the Plataformafim wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

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

/**
 * Write a description of class Opcoes here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Play extends Actor
{
    /**
     * Act - do whatever the Opcoes wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private String texto;
    private Color corPadrao = Color.WHITE;
    private Color corSecundaria = Color.YELLOW;
    private Color Cordefault = new Color(0, 0, 0, 0);
    private int tamanhoFonte;
    private int totalscore=0;
    public Play(String texto, int tamanhoFonte) {
        this.texto = texto;
        this.tamanhoFonte = tamanhoFonte;
        atualizarImagem(corPadrao);
    }
    /**
     * Função que muda a cor da palavra quando passar o rato por cima
     */
    private void atualizarImagem(Color cor) {
        GreenfootImage imagem = new GreenfootImage(texto, tamanhoFonte, cor, Cordefault);
        setImage(imagem);
```

```
    }

    public void act() {
        if (Greenfoot.mouseMoved(this)) {
            atualizarImagem(corSecundaria);
        } else if (Greenfoot.mouseMoved(null)) {
            atualizarImagem(corPadrao);
        }
        if (Greenfoot.mouseClicked(this)) {
            rato();
        }
    }

    /**
     * Função que verifica em qual dos textos o rato clicou
     */
    public void rato() {
        if (texto.equals("Play")) {
            Greenfoot.setWorld(new Nivel1(totalscore));
        } else if (texto.equals("Instruções")) {
            Greenfoot.setWorld(new Instrucoes());
        } else if (texto.equals("Exit")) {
            Greenfoot.stop();
        }
    }

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

/**
 * Write a description of class boneco here.
 *
```

```
 * @author (your name)
 * @version (a version number or a date)
 */
public class Player extends Actor
{
    /**
     * Act - do whatever the boneco wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private int tempoJump=15;
    private int tempoQueda=15;
    private boolean podeSaltar=false;
    private boolean saltou=false;
    private boolean direita =true;
    private static int vidaPlayer1 = 6;
    private static int vidaPlayer2 = 6;
    private GreenfootSound atingido;

    public Player()
    {
        atingido = new GreenfootSound("PlayerAtingido.mp3");
    }
    /**
     * Função que faz com que o player1 perca vidas
     */
    protected void perdeVidas(Player1 P1)
    {
            if(isTouching(Zombie.class)||isTouching(Zombie2.class))
            {
                playAtingido();
                P1.adicionaNumeroVidas(-1);
```

```
        }
        if(isTouching(Lenhador.class)){
            playAtingido();
            P1.adicionaNumeroVidas(-2);

        }
        Cabeca cabeca = (Cabeca) getOneIntersectingObject(Cabeca.class);
        if (cabeca!= null) {
            playAtingido();
            P1.adicionaNumeroVidas(-3);

             getWorld().removeObject(cabeca);
        }
        if(isTouching(Boss.class)){
            playAtingido();
            P1.adicionaNumeroVidas(-6);

        }
}
/**
 * Função que faz com que o player2 perca vidas
 */
protected void perdeVidas(Player2 P2)
{
    if(isTouching(Zombie.class)||isTouching(Zombie2.class))
    {
        playAtingido();
        P2.adicionaNumeroVidas(-1);


    }
```

```
    if(isTouching(Lenhador.class)){
        playAtingido();
        P2.adicionaNumeroVidas(-2);
    }
    Cabeca cabeca = (Cabeca) getOneIntersectingObject(Cabeca.class);
    if (cabeca != null) {
        playAtingido();
        P2.adicionaNumeroVidas(-3);
        getWorld().removeObject(cabeca);
    }
    if(isTouching(Boss.class)){
        playAtingido();
        P2.adicionaNumeroVidas(-6);
    }
}
public void act()
{

}
/**
 * Função que dá play ao som do player ao ser atingido
 */
public void playAtingido()
{
    atingido.play();
    atingido.setVolume(100);
}
public static void setVidaPlayer1(int vidas) {
    vidaPlayer1 = vidas;
}

public static void setVidaPlayer2(int vidas) {
```

```
        vidaPlayer2 = vidas;
    }

}

    import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.util.List;

/**
 * Write a description of class Player1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Player1 extends Player
{
    /**
     * Act - do whatever the Player1 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private GreenfootImage[] andarImagens;
    private GreenfootImage[] andarImagensesquerda;
    private GreenfootImage[] atirar;
    private GreenfootImage[] atiraresquerda;
    private GreenfootImage image3;
    private GreenfootImage image3esquerda;
    private int velocidadeVertical = 0;
    private boolean noAr = true;
    private int currentImage = 0;
    private boolean direita =true;
    private int indice;
    private int indiceAndar = 0;
```

```
private int indiceAtirar = 0;
private int delay =0;
private String tecla="up";
private boolean podeSaltar=false;
private boolean saltou;
private int tempoQueda;
private int tempoJump=15;
private List<Actor> listaDeAtores;
private int velocidade;
private int aceleracao;
private static int numeroVidas=6;
private static int score=0;
private Player1 P1;
private int delay2;
public void act()
{
    prepararImagens();
    if (getWorld() instanceof Nivel1) {
        andar();
    }
    else if(getWorld() instanceof Nivel2){
        andar2();
    }
    else if(getWorld() instanceof Nivel3){
        andar();
    }
    atirar();
    delay2++;
    if(delay2>15){
        perdeVidas(P1);
        delay2=0;
    }
```

```
}
public static int getNumeroVidas()
{
   return numeroVidas;
}

public static void resetNumVidas()
{
   numeroVidas=6;
}
/**
 * Função que adiciona vidas ao player
 */
public static void adicionaNumeroVidas(int valor)
{
   if (valor<0 || (valor > 0 && numeroVidas<5))
   {
      numeroVidas += valor;
   }
   else
   {
      if(numeroVidas<6)
      {
         numeroVidas++;
      }
   }
}
public static int getScore()
{
   return score;
}
```

```
public static void resetScore()
{
   score=0;
}

public static void adicionaScore(int valor)
{
   score+=valor;
   if(score<0)
   {
      score-=valor;
   }
}
/**
 * Função que carrega as imagens do player
 */
public void prepararImagens() {
   andarImagens = new GreenfootImage[3];
   image3 = new GreenfootImage("Joel.png");
   image3esquerda = new GreenfootImage("Joel.png");

   andarImagens[0] = new GreenfootImage("Joelandar.png");
   andarImagens[1] = new GreenfootImage("Joelandar2.png");
   andarImagens[2] = new GreenfootImage("Joelandar3.png");


   andarImagensesquerda = new GreenfootImage[3];

   andarImagensesquerda[0] = new GreenfootImage("Joelandaresquerda.png");
   andarImagensesquerda[1] = new GreenfootImage("Joelandar2esquerda.png");
   andarImagensesquerda[2] = new GreenfootImage("Joelandar3esquerda.png");
```

```
    atirar = new GreenfootImage[6];

    atirar[0]= new GreenfootImage("Joelatirar.png");
    atirar[1] = new GreenfootImage("Joelatirar1.png");
    atirar[2] = new GreenfootImage("Joelatirar2.png");
    atirar[3] = new GreenfootImage("Joelatirar2.png");
    atirar[4] = new GreenfootImage("Joelatirar2.png");
    atirar[5] = new GreenfootImage("Joelatirar2.png");

    atiraresquerda = new GreenfootImage[6];

    atiraresquerda[0]= new GreenfootImage("Joelatiraresquerda.png");
    atiraresquerda[1] = new GreenfootImage("Joelatirar1esquerda.png");
    atiraresquerda[2] = new GreenfootImage("Joelatirar2esquerda.png");
    atiraresquerda[3] = new GreenfootImage("Joelatirar2esquerda.png");
    atiraresquerda[4] = new GreenfootImage("Joelatirar2esquerda.png");
    atiraresquerda[5] = new GreenfootImage("Joelatirar2esquerda.png");
}
/**
 * Função que faz com que o player ande, salte... no nivel 1 e 3
 */
public void andar()
{
    if(verificaplat1()||verificaplat2()||verificaplat3()||verificapredio()){
        podeSaltar=true;
        saltou = false;
    }
    if (Greenfoot.isKeyDown("up") && podeSaltar){
        saltou=true;
        podeSaltar=false;
    }
    if(saltou){
```

```
            jump();
            verificarColisaoCima();
            verificarColisaoCimaplat2();
            verificarColisaoCimaplat3();
            verificarColisaoCimapredio();

        }

if(Greenfoot.isKeyDown("left")&&(colisaoComParteVerticalesquerda()||colisaoComPa
rteVerticalplat2esquerda()||colisaoComParteVerticalplat3esquerda()||colisaoComParteV
erticalpredioesquerda())){
            andaranimaesquerda();
            setLocation(getX()+5, getY());
            direita=false;
        }

if(Greenfoot.isKeyDown("right")&&(colisaoComParteVertical()||colisaoComParteVerti
calplat2()||colisaoComParteVerticalplat3()||colisaoComParteVerticalpredio())){
            andaranima();
            setLocation(getX()-5, getY());
            direita=true;
        }
        if (Greenfoot.isKeyDown("left")){
            setLocation(getX()-5, getY());
            andaranimaesquerda();
            direita=false;
        }
        else if (Greenfoot.isKeyDown("right") ){
            setLocation(getX()+5, getY());
            andaranima();
            direita=true;
        }else{
            setImage("Joel.png");
```

```
    }
    quedaplat1();
    quedaplat2();
    quedaplat3();
    quedapredio();
    onGround();
    onGround2();
    onGround3();
    onGround4();
    reposicionarSeNaBordaInferior();
}
/**
 * Função que faz com que o player ande, salte... no nivel 2
 */
public void andar2()
{
    if(verificaprateleira()||verificaarmario()||verificabookshelf()){
        podeSaltar=true;
        saltou = false;
    }
    if (Greenfoot.isKeyDown("up") && podeSaltar){
        saltou=true;
        podeSaltar=false;
    }
    if(saltou){
        jump();

        verificarColisaoCimaprateleira();
        verificarColisaoCimabookshelf();
        verificarColisaoCimaarmario();
    }
```

```
if(Greenfoot.isKeyDown("left")&&(colisaoComParteVerticalprateleiraesquerda()||colis
aoComParteVerticalarmarioesquerda()||colisaoComParteVerticalbookshelfesquerda())){
            andaranimaesquerda();
            setLocation(getX()+5, getY());
            direita=false;
        }

if(Greenfoot.isKeyDown("right")&&(colisaoComParteVerticalprateleira()||colisaoCom
ParteVerticalarmario()||colisaoComParteVerticalbookshelf())){
            andaranima();
            setLocation(getX()-5, getY());
            direita=true;
        }
        if (Greenfoot.isKeyDown("left")){
            setLocation(getX()-5, getY());
            andaranimaesquerda();
            direita=false;
        }
        else if (Greenfoot.isKeyDown("right") ){
            setLocation(getX()+5, getY());
            andaranima();
            direita=true;
        }else{
            setImage("Joel.png");
        }
        quedaprateleira();
        quedaarmario();
        quedabookshelf();
        onGround5();
        onGround6();
        onGround7();
    }
```

```
/**
 * Função que percorre o array de imagens que anima o player
 */
public void animaatirar()
{
   indice++;
      if (indice >=atirar.length)
      {
         indice =0;
      }
   setImage(atirar[indice]);

}
public void animaatiraresquerda()
{
   indice++;
      if (indice >=atiraresquerda.length)
      {
         indice =0;
      }
   setImage(atiraresquerda[indice]);

}
public void andaranima(){
   indice++;
      if (indice >=andarImagens.length)
      {
         indice =0;
      }
   setImage(andarImagens[indice]);
}
public void andaranimaesquerda(){
```

```
        indice++;
            if (indice >=andarImagensesquerda.length)
           {
              indice =0;
           }
        setImage(andarImagensesquerda[indice]);
    }
    /**
     * Função que faz com que o player dispare, desde a animação, ao disparo e ao efeito de fumo
     */
    public void atirar() {
        delay++;
        if (Greenfoot.isKeyDown("L")) {
            int posicaoX = getX();
            int posicaoY;
            if (direita == true && delay>25) {
                animaatirar();
                Greenfoot.playSound("bala.mp3");
                posicaoY = getY() - 10;
                Disparo municao = new Disparo();
                Fumo fumo = new Fumo();
                getWorld().addObject(municao, posicaoX + 30, posicaoY);
                getWorld().addObject(fumo, posicaoX + 30, posicaoY);
                delay=0;
            } else if(direita ==false && delay>25){
                animaatiraresquerda();
                Greenfoot.playSound("bala.mp3");
                posicaoY = getY() - 10;
                Disparo municao = new Disparo();
                Fumo fumo = new Fumo();
                getWorld().addObject(municao, posicaoX - 30, posicaoY);
```

```
            getWorld().addObject(fumo, posicaoX - 30, posicaoY);
            municao.setRotation(180);
            delay=0;
        }
    }

}
/**
 * Função que verifica se o player esta na platadorma
 */
public boolean verificaplat1() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma1.class);
    return emCima != null;
}
public boolean verificaplat2() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma2.class);
    return emCima != null;
}
public boolean verificaplat3() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma3.class);
    return emCima != null;
}
public boolean verificapredio() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Predio.class);
    return emCima != null;
}
public boolean verificabookshelf() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Bookshelf.class);
    return emCima != null;
```

```
    }
    public boolean verificaarmario() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Armario.class);
        return emCima != null;
    }
    public boolean verificaprateleira() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Prateleira.class);
        return emCima != null;
    }
    /**
     * Função que verifica a queda do player e que o puxa para cima quando este cai dentro da imagem
     */
    protected void quedaplat1(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2, Plataforma1.class);
        if (plat==null && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }else if (plat != null){
            setLocation(getX(),plat.getY()- plat.getImage().getHeight()/2- getImage().getHeight()/2);
            tempoQueda=0;
            podeSaltar=true;

        }
        plat=null;
    }
    protected void quedaplat2(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2, Plataforma2.class);
        if (plat==null && !saltou){
```

```
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }else if (plat != null){
            setLocation(getX(),plat.getY()-plat.getImage().getHeight()/2-
getImage().getHeight()/2);
            tempoQueda=0;
            podeSaltar=true;

        }
        plat=null;
    }
    protected void quedaprateleira(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Prateleira.class);
        if (plat==null && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }else if (plat != null){
            setLocation(getX(),plat.getY()-plat.getImage().getHeight()/2-
getImage().getHeight()/2);
            tempoQueda=0;
            podeSaltar=true;

        }
        plat=null;
    }
    protected void quedabookshelf(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Bookshelf.class);
        if (plat==null && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }else if (plat != null){
```

```
        setLocation(getX(),plat.getY()-                plat.getImage().getHeight()/2-
getImage().getHeight()/2);
        tempoQueda=0;
        podeSaltar=true;


    }
    plat=null;
  }
  protected void quedaarmario(){
    Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Armario.class);
    if (plat==null && !saltou){
        setLocation(getX(),getY()+tempoQueda);
        tempoQueda++;
    }else if (plat != null){
        setLocation(getX(),plat.getY()-                plat.getImage().getHeight()/2-
getImage().getHeight()/2);
        tempoQueda=0;
        podeSaltar=true;


    }
    plat=null;
  }
  protected void quedaplat3(){
    Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Plataforma3.class);
    if (plat==null && !saltou){
        setLocation(getX(),getY()+tempoQueda);
        tempoQueda++;
    }else if (plat != null){
        setLocation(getX(),plat.getY()-                plat.getImage().getHeight()/2-
getImage().getHeight()/2);
        tempoQueda=0;
        podeSaltar=true;
```

```
        }
        plat=null;
    }
    protected void quedapredio(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Predio.class);
        if (plat==null && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }else if (plat != null){
            setLocation(getX(),plat.getY()- plat.getImage().getHeight()/2-
getImage().getHeight()/2);
            tempoQueda=0;
            podeSaltar=true;


        }
        plat=null;
    }
    /**
     * Função que verifica se o player colide com a parede vertical direita
     */
    public boolean colisaoComParteVertical() {
        Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0,
Plataforma1.class);
       return actorNaDireita != null;
    }
    public boolean colisaoComParteVerticalplat2() {
        Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0,
Plataforma2.class);
        return  actorNaDireita != null;
    }
    public boolean colisaoComParteVerticalplat3() {
```

```
    Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Plataforma3.class);
    return  actorNaDireita != null;
  }
  public boolean colisaoComParteVerticalpredio() {
    Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Predio.class);
    return  actorNaDireita != null;
  }
  public boolean colisaoComParteVerticalprateleira() {
    Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Prateleira.class);
    return  actorNaDireita != null;
  }
  public boolean colisaoComParteVerticalbookshelf() {
    Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Bookshelf.class);
    return  actorNaDireita != null;
  }
  public boolean colisaoComParteVerticalarmario() {
    Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Armario.class);
    return  actorNaDireita != null;
  }

  /**
   * Função que verifica se o player colide com a parede vertical esquerda
   */
  public boolean colisaoComParteVerticalesquerda() {
    Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Plataforma1.class);
    return actorNaEsquerda != null ;
  }
  public boolean colisaoComParteVerticalplat2esquerda() {
```

```
    Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Plataforma2.class);

    return actorNaEsquerda != null ;

  }

  public boolean colisaoComParteVerticalplat3esquerda() {

    Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Plataforma3.class);

    return actorNaEsquerda != null ;

  }

  public boolean colisaoComParteVerticalprateleiraesquerda() {

    Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Prateleira.class);

    return actorNaEsquerda != null ;

  }

  public boolean colisaoComParteVerticalpredioesquerda() {

    Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Predio.class);

    return actorNaEsquerda != null ;

  }

  public boolean colisaoComParteVerticalbookshelfesquerda() {

    Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Bookshelf.class);

    return actorNaEsquerda != null ;

  }

  public boolean colisaoComParteVerticalarmarioesquerda() {

    Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Armario.class);

    return actorNaEsquerda != null ;

  }



  /**

   * Função que verifica se o personagem bate com a cabeça no teto, ou seja, em baixo de alguma plataforma

   */
```

```
    public boolean verificarColisaoCima() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2, Plataforma1.class);
        if (colisaoCima != null) {
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() / 2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
    }
    public boolean verificarColisaoCimaplat2() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2, Plataforma2.class);
        if (colisaoCima != null) {
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() / 2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
    }
    public boolean verificarColisaoCimaplat3() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2, Plataforma3.class);
        if (colisaoCima != null) {
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() / 2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
    }
    public boolean verificarColisaoCimapredio() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2, Predio.class);
        if (colisaoCima != null) {
```

```
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() /
2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
    }
    public boolean verificarColisaoCimaarmario() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2,
Armario.class);
        if (colisaoCima != null) {
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() /
2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
    }
    public boolean verificarColisaoCimaprateleira() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2,
Prateleira.class);
        if (colisaoCima != null) {
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() /
2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
    }
    public boolean verificarColisaoCimabookshelf() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2,
Bookshelf.class);
        if (colisaoCima != null) {
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() /
2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
```

```
    }
    /**
     * Função que verifica se o personagem está dentro do chão
     */
    public boolean onGround()
    {
        Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma1.class);
        if (platformUnder == null ) {
            return false;

        } else {
            if (platformUnder != null) {
                moveToGround(platformUnder);
                podeSaltar = true;
            }
            return true;
        }

    }
    public boolean onGround2()
    {
        Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma2.class);
        if (platformUnder == null ) {
            return false;
        } else {
            if (platformUnder != null) {
                moveToGround(platformUnder);
                podeSaltar = true;
            }
            return true;
        }
```

```
    }
    public boolean onGround3()
    {
        Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Plataforma3.class);
        if (platformUnder == null ) {
            return false;
        } else {
            if (platformUnder != null) {
                moveToGround(platformUnder);
                podeSaltar = true;
            }
            return true;
        }
    }
    public boolean onGround4()
    {
        Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Predio.class);
        if (platformUnder == null ) {
            return false;
        } else {
            if (platformUnder != null) {
                moveToGround(platformUnder);
                podeSaltar = true;
            }
            return true;
        }
    }
    public boolean onGround5()
    {
        Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Prateleira.class);
```

```
        if (platformUnder == null ) {
            return false;
        } else {
            if (platformUnder != null) {
                moveToGround(platformUnder);
                podeSaltar = true;
            }
            return true;
        }
    }
    public boolean onGround6()
    {
        Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Bookshelf.class);
        if (platformUnder == null ) {
            return false;
        } else {
            if (platformUnder != null) {
                moveToGround(platformUnder);
                podeSaltar = true;
            }
            return true;
        }
    }
    public boolean onGround7()
    {
        Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Armario.class);
        if (platformUnder == null ) {
            return false;
        } else {
            if (platformUnder != null) {
                moveToGround(platformUnder);
```

```
            podeSaltar = true;
        }
        return true;
    }
}
/**
 * Função que puxa o personagem para cima, se o mesmo tiver dentro do chão
 */
public void moveToGround(Actor under)
{
    int underHeight = under.getImage().getHeight();
    int newY = under.getY() - (underHeight + getImage().getHeight()) / 2+1;
    setLocation(getX(), newY);
    podeSaltar = true;
}
/**
 * Função que faz com que o personagem salte
 */
public void jump(){
    if (tempoJump>0){
        setLocation(getX(),getY()-tempoJump);
        tempoJump--;
    }else{
        tempoJump=20;
        saltou=false;
    }
}
/**
 * Método que tira o player de dentro da parte debaixo do mundo
 */
public void reposicionarSeNaBordaInferior() {
    int limiteInferior = getWorld().getHeight();
```

```
        if (getY() >= limiteInferior - 1) {
            int alturaPlayer = getImage().getHeight();
            int novaPosicaoY = getY() - 63 - (alturaPlayer / 2);
            setLocation(getX(), novaPosicaoY);
        }
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Vida_player1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Vida_Player1 extends Player1
{
    /**
     * Act - do whatever the Vida_player1 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private GreenfootImage[] vidas;

    public Vida_Player1()
    {
        vidas = new GreenfootImage[6];
        for (int i=0; i < vidas.length; i++)
        {
            vidas[i] = new GreenfootImage("Vida/"+(i+1)+".png");
        }
    }
```

```
    public void act()
    {
        vidaPlayer1();
    }
    /**
     * Método que atualiza a imagem das vidas
     */
    public void vidaPlayer1()
    {
        if (Player1.getNumeroVidas() >0)
        {
            setImage(vidas[Player1.getNumeroVidas()-1]);
        }
        else {
            getWorld().removeObject(this);
        }
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Player2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Player2 extends Player
{
    /**
     * Act - do whatever the Player2 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
```

```
private GreenfootImage[] andarImagens;
private GreenfootImage[] andarImagensesquerda;
private GreenfootImage[] atirar;
private GreenfootImage[] atiraresquerda;
private GreenfootImage image3;
private GreenfootImage image3esquerda;
private int velocidadeVertical = 0;
private boolean noAr = true;
private int currentImage = 0;
private boolean direita =true;
private int indice;
private int indiceAndar = 0;
private int indiceAtirar = 0;
private int delay=0;
private int delayimg=0;
private int delayatirar=0;
private String tecla="W";
private int tempoJump=20;
private int tempoQueda;
private boolean saltou=false;
private boolean podeSaltar=false;
private static int numeroVidas=6;
private Player2 P2;
private static int score;
private int delay2;
public void act()
{
    prepararImagens();
    if (getWorld() instanceof Nivel1) {
        andar();
    }
    else if(getWorld() instanceof Nivel2){
```

```
            andar2();
        }
        else if(getWorld() instanceof Nivel3){
            andar();
        }
        atirar();
        delay2++;
        if(delay2>15){
            perdeVidas(P2);
            delay2=0;
        }

    }
    public static int getNumeroVidas()
    {
        return numeroVidas;
    }

    public static void resetNumVidas()
    {
        numeroVidas=6;
    }
    /**
     * Função que adiciona vidas ao player
     */
    public static void adicionaNumeroVidas(int valor)
    {
        if (valor<0 || (valor > 0 && numeroVidas<5))
        {
            numeroVidas += valor;
        }
        else
```

```
    {
        if(numeroVidas<6)
        {
            numeroVidas++;
        }
    }
}
public static int getScore()
{
    return score;
}

public static void resetScore()
{
    score=0;
}

public static void adicionaScore(int valor)
{
    score+=valor;
    if(score<0)
    {
        score-=valor;
    }
}
/**
 * Função que carrega as imagens do player
 */
public void prepararImagens() {
    andarImagens = new GreenfootImage[5];
    image3 = new GreenfootImage("Ellie3.png");
    image3esquerda = new GreenfootImage("Ellie3esquerda.png");
```

```
andarImagens[0] = new GreenfootImage("Elliecorrer.png");
andarImagens[1] = new GreenfootImage("Elliecorrer2.png");
andarImagens[2] = new GreenfootImage("Elliecorrer3.png");
andarImagens[3] = new GreenfootImage("Elliecorrer4.png");
andarImagens[4] = new GreenfootImage("Elliecorrer5.png");

andarImagensesquerda = new GreenfootImage[5];

andarImagensesquerda[0] = new GreenfootImage("Elliecorreresquerda.png");
andarImagensesquerda[1] = new GreenfootImage("Elliecorrer2esquerda.png");
andarImagensesquerda[2] = new GreenfootImage("Elliecorrer3esquerda.png");
andarImagensesquerda[3] = new GreenfootImage("Elliecorrer4esquerda.png");
andarImagensesquerda[4] = new GreenfootImage("Elliecorrer5esquerda.png");

atirar = new GreenfootImage[7];

atirar[0]= new GreenfootImage("Ellie1.png");
atirar[1] = new GreenfootImage("Ellie2.png");
atirar[2] = new GreenfootImage("Ellie3.png");
atirar[3] = new GreenfootImage("Ellie3.png");
atirar[4] = new GreenfootImage("Ellie3.png");
atirar[5] = new GreenfootImage("Ellie3.png");
atirar[6] = new GreenfootImage("Ellie3.png");

atiraresquerda = new GreenfootImage[8];

atiraresquerda[0]= new GreenfootImage("Ellie1.png");
atiraresquerda[1] = new GreenfootImage("Ellie2esquerda.png");
atiraresquerda[2] = new GreenfootImage("Ellie3esquerda.png");
atiraresquerda[3] = new GreenfootImage("Ellie3esquerda.png");
atiraresquerda[4] = new GreenfootImage("Ellie3esquerda.png");
atiraresquerda[5] = new GreenfootImage("Ellie3esquerda.png");
```

```
        atiraresquerda[6] = new GreenfootImage("Ellie3esquerda.png");
        atiraresquerda[7] = new GreenfootImage("Ellie3esquerda.png");

    }
    /**
     * Função que faz com que o player ande e salte no Nivel 1 e 3
     */
    public void andar()
    {
        if(verificaplat1()||verificaplat2()||verificaplat3()||verificapredio()){
            podeSaltar=true;
            saltou = false;
        }
        if (Greenfoot.isKeyDown("W") && podeSaltar){
            saltou=true;
            podeSaltar=false;
        }
        if(saltou){
            jump();
            verificarColisaoCima();
            verificarColisaoCimaplat2();
            verificarColisaoCimaplat3();
            verificarColisaoCimapredio();

        }


if(Greenfoot.isKeyDown("A")&&(colisaoComParteVerticalesquerda()||colisaoComPart
eVerticalplat2esquerda()||colisaoComParteVerticalplat3esquerda()||colisaoComParteVer
ticalpredioesquerda())){
            andaranimaesquerda();
            setLocation(getX()+5, getY());
            direita=false;
```

```
        }

if(Greenfoot.isKeyDown("D")&&(colisaoComParteVertical()||colisaoComParteVertical
plat2()||colisaoComParteVerticalplat3()||colisaoComParteVerticalpredio())){
            andaranima();
            setLocation(getX()-5, getY());
            direita=true;
        }
        if (Greenfoot.isKeyDown("A")){
            setLocation(getX()-5, getY());
            andaranimaesquerda();
            direita=false;
        }
        else if (Greenfoot.isKeyDown("D") ){
            setLocation(getX()+5, getY());
            andaranima();
            direita=true;
        }else{
            setImage("Ellie1.png");
        }
        quedaplat1();
        quedaplat2();
        quedaplat3();
        quedapredio();
        onGround();
        onGround2();
        onGround3();
        onGround4();
        reposicionarSeNaBordaInferior();
    }
    /**
     * Função que faz com que o player ande e salte no Nivel 2
     */
```

```
public void andar2()
{
    if(verificaprateleira()||verificaarmario()||verificabookshelf()){
        podeSaltar=true;
        saltou = false;
    }
    if (Greenfoot.isKeyDown("W") && podeSaltar){
        saltou=true;
        podeSaltar=false;
    }
    if(saltou){
        jump();
        verificarColisaoCimaprateleira();
        verificarColisaoCimabookshelf();
        verificarColisaoCimaarmario();
    }


if(Greenfoot.isKeyDown("A")&&(colisaoComParteVerticalplat2esquerda()||colisaoCo
mParteVerticalplat3esquerda()||colisaoComParteVerticalpredioesquerda())){
        andaranimaesquerda();
        setLocation(getX()+5, getY());
        direita=false;
    }

if(Greenfoot.isKeyDown("D")&&(colisaoComParteVerticalprateleira()||colisaoComPart
eVerticalarmario()||colisaoComParteVerticalbookshelf())){
        andaranima();
        setLocation(getX()-5, getY());
        direita=true;
    }
    if (Greenfoot.isKeyDown("A")){
        setLocation(getX()-5, getY());
```

```
        andaranimaesquerda();
        direita=false;
    }
    else if (Greenfoot.isKeyDown("D") ){
        setLocation(getX()+5, getY());
        andaranima();
        direita=true;
    }else{
        setImage("Ellie1.png");
    }
    quedaprateleira();
    quedaarmario();
    quedabookshelf();
    onGround5();
    onGround6();
    onGround7();
}
/**
 * Função que percorre array de imagens e anima o personagem
 */
public void animaatirar()
{
    indice++;
        if (indice >=atirar.length)
        {
            indice =0;
        }
    setImage(atirar[indice]);

}
public void animaatiraresquerda()
{
```

```
    indice++;
        if (indice >=atiraresquerda.length)
        {
            indice =0;
        }
    setImage(atiraresquerda[indice]);

}
public void andaranima(){
    indice++;
        if (indice >=andarImagens.length)
        {
            indice =0;
        }
    setImage(andarImagens[indice]);
}
public void andaranimaesquerda(){
    indice++;
         if (indice >=andarImagensesquerda.length)
        {
            indice =0;
        }
    setImage(andarImagensesquerda[indice]);
}
/**
 * Método que faz com que o player atire para ambos os lados, incluindo o disparo
 */
public void atirar() {
    delay++;
    if (Greenfoot.isKeyDown("E")) {
        int posicaoX = getX();
        int posicaoY;
```

```
        if (direita == true && delay>25) {
            animaatirar();
            Greenfoot.playSound("bala.mp3");
            posicaoY = getY() - 10;
            Disparo municao = new Disparo();
            Fumo fumo = new Fumo();
            getWorld().addObject(municao, posicaoX + 30, posicaoY);
            getWorld().addObject(fumo, posicaoX + 30, posicaoY);
            delay=0;
        } else if(direita ==false && delay>25){
            animaatiraresquerda();
            Greenfoot.playSound("bala.mp3");
            posicaoY = getY() - 10;
            Disparo municao = new Disparo();
            Fumo fumo = new Fumo();
            getWorld().addObject(municao, posicaoX - 30, posicaoY);
            getWorld().addObject(fumo, posicaoX - 30, posicaoY);
            municao.setRotation(180);
            delay=0;
        }
    }

}
/**
 * Método que verifica se o player esta em cima de uma plataforma
 */
public boolean verificaplat1() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma1.class);
    return emCima != null;
}
public boolean verificaplat2() {
```

```
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Plataforma2.class);
    return emCima != null;
  }
  public boolean verificaplat3() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Plataforma3.class);
    return emCima != null;
  }
  public boolean verificapredio() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Predio.class);
    return emCima != null;
  }
  public boolean verificabookshelf() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Bookshelf.class);
    return emCima != null;
  }
  public boolean verificaarmario() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Armario.class);
    return emCima != null;
  }
  public boolean verificaprateleira() {
    Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Prateleira.class);
    return emCima != null;
  }
  /**
   * Método que verifica a queda do player dentro de uma plataforma
   */
  protected void quedaplat1(){
    Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Plataforma1.class);
```

```
        if (plat==null && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }else if (plat != null){
            setLocation(getX(),plat.getY()-                    plat.getImage().getHeight()/2-
getImage().getHeight()/2);
            tempoQueda=0;
            podeSaltar=true;


        }
        plat=null;
    }
    protected void quedaplat2(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Plataforma2.class);
        if (plat==null && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }else if (plat != null){
            setLocation(getX(),plat.getY()-                    plat.getImage().getHeight()/2-
getImage().getHeight()/2);
            tempoQueda=0;
            podeSaltar=true;


        }
        plat=null;
    }
    protected void quedaplat3(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Plataforma3.class);
        if (plat==null && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
```

```
    }else if (plat != null){
        setLocation(getX(),plat.getY()-                plat.getImage().getHeight()/2-
getImage().getHeight()/2);
        tempoQueda=0;
        podeSaltar=true;


    }
    plat=null;
  }
  protected void quedapredio(){
    Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Predio.class);
    if (plat==null && !saltou){
        setLocation(getX(),getY()+tempoQueda);
        tempoQueda++;
    }else if (plat != null){
        setLocation(getX(),plat.getY()-                plat.getImage().getHeight()/2-
getImage().getHeight()/2);
        tempoQueda=0;
        podeSaltar=true;


    }
    plat=null;
  }
  protected void quedaprateleira(){
    Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Prateleira.class);
    if (plat==null && !saltou){
        setLocation(getX(),getY()+tempoQueda);
        tempoQueda++;
    }else if (plat != null){
        setLocation(getX(),plat.getY()-                plat.getImage().getHeight()/2-
getImage().getHeight()/2);
        tempoQueda=0;
```

```
            podeSaltar=true;


        }
        plat=null;
    }
    protected void quedabookshelf(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Bookshelf.class);
        if (plat==null && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }else if (plat != null){
            setLocation(getX(),plat.getY()-                plat.getImage().getHeight()/2-
getImage().getHeight()/2);
            tempoQueda=0;
            podeSaltar=true;


        }
        plat=null;
    }
    protected void quedaarmario(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
Armario.class);
        if (plat==null && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }else if (plat != null){
            setLocation(getX(),plat.getY()-                plat.getImage().getHeight()/2-
getImage().getHeight()/2);
            tempoQueda=0;
            podeSaltar=true;


        }
```

```
        plat=null;
    }
    /**
     * Método que verifica a colisão do player com a parede direita do objeto
     */
    public boolean colisaoComParteVertical() {
        Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Plataforma1.class);
       return actorNaDireita != null;
    }
    public boolean colisaoComParteVerticalplat2() {
        Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Plataforma2.class);
        return  actorNaDireita != null;
    }
    public boolean colisaoComParteVerticalplat3() {
        Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Plataforma3.class);
        return  actorNaDireita != null;
    }
    public boolean colisaoComParteVerticalpredio() {
        Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Predio.class);
        return  actorNaDireita != null;
    }
    public boolean colisaoComParteVerticalprateleira() {
        Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Prateleira.class);
        return  actorNaDireita != null;
    }
    public boolean colisaoComParteVerticalbookshelf() {
        Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Bookshelf.class);
        return  actorNaDireita != null;
```

```
    }
    public boolean colisaoComParteVerticalarmario() {
        Actor actorNaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, 0, Armario.class);
        return  actorNaDireita != null;
    }

    /**
     * Método que verifica a colisão com a parede vertical esquerda
     */
    public boolean colisaoComParteVerticalesquerda() {
        Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Plataforma1.class);
        return actorNaEsquerda != null ;
    }
    public boolean colisaoComParteVerticalplat2esquerda() {
        Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Plataforma2.class);
        return actorNaEsquerda != null ;
    }
    public boolean colisaoComParteVerticalplat3esquerda() {
        Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Plataforma3.class);
        return actorNaEsquerda != null ;
    }
    public boolean colisaoComParteVerticalpredioesquerda() {
        Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Predio.class);
        return actorNaEsquerda != null ;
    }
    public boolean colisaoComParteVerticalprateleiraesquerda() {
        Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Prateleira.class);
        return actorNaEsquerda != null ;
```

```
    }
    public boolean colisaoComParteVerticalbookshelfesquerda() {
        Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Bookshelf.class);
        return actorNaEsquerda != null ;
    }
    public boolean colisaoComParteVerticalarmarioesquerda() {
        Actor actorNaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, 0, Armario.class);
        return actorNaEsquerda != null ;
    }

    /**
     * Método que verifica quando o player bate com a cabeça numa plataforma
     */
    public boolean verificarColisaoCima() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2, Plataforma1.class);
        if (colisaoCima != null) {
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() / 2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
    }
    public boolean verificarColisaoCimaplat2() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2, Plataforma2.class);
        if (colisaoCima != null) {
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() / 2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
```

```
    }

    public boolean verificarColisaoCimaplat3() {

        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2,
Plataforma3.class);

        if (colisaoCima != null) {

            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() /
2 + getImage().getHeight() / 2);

            return true;

        }

        return false;

    }

    public boolean verificarColisaoCimapredio() {

        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2,
Predio.class);

        if (colisaoCima != null) {

            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() /
2 + getImage().getHeight() / 2);

            return true;

        }

        return false;

    }

    public boolean verificarColisaoCimaarmario() {

        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2,
Armario.class);

        if (colisaoCima != null) {

            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() /
2 + getImage().getHeight() / 2);

            return true;

        }

        return false;

    }

    public boolean verificarColisaoCimaprateleira() {

        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2,
Prateleira.class);

        if (colisaoCima != null) {
```

```
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() /
2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
    }
    public boolean verificarColisaoCimabookshelf() {
        Actor colisaoCima = getOneObjectAtOffset(0, -getImage().getHeight() / 2,
Bookshelf.class);
        if (colisaoCima != null) {
            setLocation(getX(), colisaoCima.getY() + colisaoCima.getImage().getHeight() /
2 + getImage().getHeight() / 2);
            return true;
        }
        return false;
    }

    /**
     * método que verifica se um player entrou dentro de uma plataforma
     */
    public boolean onGround()
    {
        Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Plataforma1.class);
        if (platformUnder == null ) {
            return false;

        } else {
            if (platformUnder != null) {
                moveToGround(platformUnder);
                podeSaltar = true;
            }
            return true;
```

```
    }


}
public boolean onGround2()
{
    Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Plataforma2.class);
    if (platformUnder == null ) {
        return false;
    } else {
        if (platformUnder != null) {
            moveToGround(platformUnder);
            podeSaltar = true;
        }
        return true;
    }
}
public boolean onGround3()
{
    Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Plataforma3.class);
    if (platformUnder == null ) {
        return false;
    } else {
        if (platformUnder != null) {
            moveToGround(platformUnder);
            podeSaltar = true;
        }
        return true;
    }
}
public boolean onGround4()
{
```

```
    Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2, Predio.class);
    if (platformUnder == null ) {
        return false;
    } else {
        if (platformUnder != null) {
            moveToGround(platformUnder);
            podeSaltar = true;
        }
        return true;
    }
}
public boolean onGround5()
{
    Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2, Prateleira.class);
    if (platformUnder == null ) {
        return false;
    } else {
        if (platformUnder != null) {
            moveToGround(platformUnder);
            podeSaltar = true;
        }
        return true;
    }
}
public boolean onGround6()
{
    Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2, Bookshelf.class);
    if (platformUnder == null ) {
        return false;
    } else {
```

```
            if (platformUnder != null) {
                moveToGround(platformUnder);
                podeSaltar = true;
            }
            return true;
        }
    }
    public boolean onGround7()
    {
        Actor platformUnder = getOneObjectAtOffset(0, getImage().getHeight() / 2, Armario.class);
        if (platformUnder == null ) {
            return false;
        } else {
            if (platformUnder != null) {
                moveToGround(platformUnder);
                podeSaltar = true;
            }
            return true;
        }
    }
    /**
     * Método que retira o player de dentro de uma plataforma, quando o mesmo entra dentro
     */
    public void moveToGround(Actor under)
    {
        int underHeight = under.getImage().getHeight();
        int newY = under.getY() - (underHeight + getImage().getHeight()) / 2+1;
        setLocation(getX(), newY);
        podeSaltar = true;
    }
    /**
```

```
     * Método que faz com que o player salte
     */
    public void jump(){
        if (tempoJump>0){
            setLocation(getX(),getY()-tempoJump);
            tempoJump--;
        }else{
            tempoJump=20;
            saltou=false;
        }
    }
    /**
     * Método que tira o player de dentro da parte debaixo do mundo
     */
    public void reposicionarSeNaBordaInferior() {
        int limiteInferior = getWorld().getHeight();
        if (getY() >= limiteInferior - 1) {
            int alturaPlayer = getImage().getHeight();
            int novaPosicaoY = getY() - 63 - (alturaPlayer / 2);
            setLocation(getX(), novaPosicaoY);
        }
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Vida_player2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Vida_Player2 extends Player2
```

```
{
    /**
     * Act - do whatever the Vida_player2 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private GreenfootImage[] vidas;

    public Vida_Player2()
    {
        vidas = new GreenfootImage[6];
        for (int i=0; i < vidas.length; i++)
        {
            vidas[i] = new GreenfootImage("Vida/"+(i+1)+".png");
        }
    }

    public void act()
    {
        vidaPlayer2();
    }
    /**
     *  Método que atualiza a imagem das vidas
     */
    public void vidaPlayer2()
    {
        if (Player2.getNumeroVidas() >0)
        {
            setImage(vidas[Player2.getNumeroVidas()-1]);
        }
        else {
            getWorld().removeObject(this);
        }
```

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

/**
 * Write a description of class Prateleira here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Prateleira extends Actor
{
    /**
     * Act - do whatever the Prateleira wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Predio here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Predio extends Actor
{
    /**
```

```
     * Act - do whatever the Predio wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void Predio(){

        GreenfootImage image = new GreenfootImage("Predio.png");
        setImage(image);

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

/**
 * Write a description of class Restart here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Restart extends Actor
{
    private int inicial=0;
    private GreenfootImage imagem;

    public Restart(){
        imagem = new GreenfootImage("Restart.png");
        setImage(imagem);
    }
    public void act(){
        if(Greenfoot.mouseClicked(this)){
            rato();
        }
    }
```

```
    /**
     * Método que deteta se o rato clicou no restart
     */
    public void rato(){
        if (getImage() == imagem) {
            Greenfoot.setWorld(new Nivel1(inicial));
        }
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Score here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Score extends Actor
{
    /**
     * 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 pontos;
    private GreenfootImage imagem;
    private int score;
    public void act() {

    }
    public Score(int inicial) {
        this.score =inicial;
        atualizarImagem();
```

```
    }
    /**
     * Método que adiciona pontos ao score
     */
    public void adicionarPontos(int valor) {
        pontos += valor;
        atualizarImagem();
    }
    /**
     * Método que atualiza imagens
     */
    private void atualizarImagem() {
        imagem = new GreenfootImage("Score: " + pontos, 24, Color.WHITE, new Color(0,
0, 0, 0));
        setImage(imagem);
    }

    public int getPontos() {
        return pontos;
    }

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

/**
 * Write a description of class Texto here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Texto extends Actor
{
```

```
    /**
     * Act - do whatever the Texto wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */

    private Color corTexto;
    private Color corFundo;

    public Texto(String text, int size, Color corTexto, Color corFundo) {
        this.corTexto = corTexto;
        this.corFundo = corFundo;
        atualizarImagem(text, size);
    }
    /**
     * O método serve para atualizar a imagem exibida pelo objeto
     */
    private void atualizarImagem(String text, int size) {
        GreenfootImage image = new GreenfootImage(text.toUpperCase(), size, corTexto,
corFundo);
        setImage(image);
    }
}

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

/**
 * Write a description of class Titulo here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Titulo extends Actor
```

```
{
    /**
     * Act - do whatever the Titulo wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Trapdoor here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Trapdoor extends Actor
{
    /**
     * Act - do whatever the Trapdoor wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public boolean passardenivel=false;
    public void act()
    {
        porta();
    }
    /**
     * Função que verifica se ambos os players estão em cima da trapdoor e passa do Nivel 1 para o 2
```

```
     */
    public void porta(){

        if(isTouching(Player1.class) && isTouching(Player2.class)){
            passardenivel=true;
            Trapdooraberta trapdooraberta = new Trapdooraberta();
            getWorld().addObject(trapdooraberta, 1103, 238);

            Nivel1 nivel=(Nivel1) getWorld();
            int totalScore = nivel.getScore().getPontos();
            Greenfoot.setWorld(new Nivel2(totalScore));
        }

    }

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

/**
 * Write a description of class Trapdooraberta here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Trapdooraberta extends Actor
{
    /**
     * Act - do whatever the Trapdooraberta wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
```

```
        // Add your action code here.
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class zombie here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Zombie extends Actor
{
    /**
     * Act - do whatever the zombie wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private int indice;
    private GreenfootImage[] zombie;
    private GreenfootImage[] zombieesquerda;
    private int counter;
    private boolean direita = false;
    private int vida;
    public Zombie(){
        imagens();
        vida = 2;
    }
    public void act()
    {   if (vida <= 0){
            morre();
            return;
        }
```

```
    if (verificaPlataforma1()) {
        moveZombiePlataforma1();
    } else if (verificaPlataforma2()) {
        moveZombiePlataforma2();
    }else if(verificaPlataforma3()){
        moveZombiePlataforma3();
    }
}
/**
 * Função que carrega as imagens do zombie
 */
public void imagens(){
    zombie= new GreenfootImage[8];

    zombie[0]= new GreenfootImage("zombie1.png");
    zombie[1]= new GreenfootImage("zombie2.png");
    zombie[2]= new GreenfootImage("zombie3.png");
    zombie[3]= new GreenfootImage("zombie4.png");
    zombie[4]= new GreenfootImage("zombie5.png");
    zombie[5]= new GreenfootImage("zombie6.png");
    zombie[6]= new GreenfootImage("zombie7.png");
    zombie[7]= new GreenfootImage("zombie8.png");

    zombieesquerda= new GreenfootImage[8];

    zombieesquerda[0]= new GreenfootImage("zombie1esquerda.png");
    zombieesquerda[1]= new GreenfootImage("zombie2esquerda.png");
    zombieesquerda[2]= new GreenfootImage("zombie3esquerda.png");
    zombieesquerda[3]= new GreenfootImage("zombie4esquerda.png");
    zombieesquerda[4]= new GreenfootImage("zombie5esquerda.png");
```

```
    zombieesquerda[5]= new GreenfootImage("zombie6esquerda.png");
    zombieesquerda[6]= new GreenfootImage("zombie7esquerda.png");
    zombieesquerda[7]= new GreenfootImage("zombie8esquerda.png");
}
/**
 * Função que verifica se o zombie está em contacto com a Plataforma1
 */
public void moveZombiePlataforma1() {
    if (atBordaDireita() && direita) {
        direita = false;
    } else if (atBordaEsquerda() && !direita) {
        direita = true;
    }

    if (direita) {
        setLocation(getX() + 5, getY());
        zombieanima();
    } else {
        setLocation(getX() - 5, getY());
        zombieanimaesquerda();
    }
}
/**
 * Função que verifica se o zombie está em contacto com a Plataforma2
 */
public void moveZombiePlataforma2() {
    if (atBordaDireita2() && direita) {
        direita = false;
    } else if (atBordaEsquerda2() && !direita) {
        direita = true;
    }
```

```
    if (direita) {
        setLocation(getX() + 5, getY());
        zombieanima();
    } else {
        setLocation(getX() - 5, getY());
        zombieanimaesquerda();
    }
}
/**
 * Função que verifica se o zombie está em contacto com a Plataforma3
 */
public void moveZombiePlataforma3() {
    if (atBordaDireita3() && direita) {
        direita = false;
    } else if (atBordaEsquerda3() && !direita) {
        direita = true;
    }

    if (direita) {
        setLocation(getX() + 5, getY());
        zombieanima();
    } else {
        setLocation(getX() - 5, getY());
        zombieanimaesquerda();
    }
}
/**
 * Função que percorre o array de imagens e anima o zombie
 */
public void zombieanima() {
    counter++;
    if (counter > 10) {
```

```
            indice++;
            if (indice >= zombie.length) {
                indice = 0;
            }
            counter = 0;
        }
        setImage(zombie[indice]);
    }
    /**
     * Função que percorre o array de imagens e anima o zombie mas do lado esquerdo
     */
    public void zombieanimaesquerda() {
        counter++;
        if (counter > 10) {
            indice++;
            if (indice >= zombieesquerda.length) {
                indice = 0;
            }
            counter = 0;
        }
        setImage(zombieesquerda[indice]);
    }
    /**
     * Função que verifica se o zombie esta em cima da plataforma
     */
    public boolean verificaPlataforma1() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Plataforma1.class);
        return emCima != null;
    }
    public boolean verificaPlataforma2() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2,
Plataforma2.class);
```

```
        return emCima != null;
    }
    public boolean verificaPlataforma3() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma3.class);
        return emCima != null;
    }

    /**
     * Função que verifica se o zombie chegou à borda esquerd da plataforma
     */
    public boolean atBordaDireita() {
        Actor bordaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma1.class);
        return bordaDireita == null;
    }
    public boolean atBordaDireita2() {
        Actor bordaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma2.class);
        return bordaDireita == null;
    }
    public boolean atBordaDireita3() {
        Actor bordaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma3.class);
        return bordaDireita == null;
    }

    /**
     * Função que verifica se o zombie chegou à borda esquerda da plataforma
     */
    public boolean atBordaEsquerda() {
        Actor bordaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma1.class);
        return bordaEsquerda == null;
```

```
    }
    public boolean atBordaEsquerda2() {
        Actor bordaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma2.class);
        return bordaEsquerda == null;
    }
    public boolean atBordaEsquerda3() {
        Actor bordaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma3.class);
        return bordaEsquerda == null;
    }

    public int getVida()
    {
        return vida;
    }

    public void tiraVida(int valor)
    {
        vida-=valor;
    }
    /**
     * Função que retira o zombie do mundo quando ele morre
     */
    public void morre(){
        if (vida <= 0){
            getWorld().removeObject(this);
        }
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
```

```
 * Write a description of class zombie2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Zombie2 extends Actor
{
    /**
     * Act - do whatever the zombie2 wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private int indice;
    private GreenfootImage[] zombie;
    private GreenfootImage[] zombieesquerda;
    private int counter;
    private boolean direita = false;
    private int vida;
    public Zombie2(){
        imagens();
        vida = 2;
    }
    public void act()
    {
        if (vida <= 0){
            morre();
            return;
        }
        if (verificaPlataforma1()) {
            moveZombiePlataforma1();
        } else if (verificaPlataforma2()) {
            moveZombiePlataforma2();
        }else if(verificaPlataforma3()){
```

```
        moveZombiePlataforma3();
    }

}
/**
 * Função que carrega as imagens do zombie
 */
public void imagens(){
    zombie= new GreenfootImage[8];

    zombie[0]= new GreenfootImage("zombie21.png");
    zombie[1]= new GreenfootImage("zombie22.png");
    zombie[2]= new GreenfootImage("zombie23.png");
    zombie[3]= new GreenfootImage("zombie24.png");
    zombie[4]= new GreenfootImage("zombie25.png");
    zombie[5]= new GreenfootImage("zombie26.png");
    zombie[6]= new GreenfootImage("zombie27.png");
    zombie[7]= new GreenfootImage("zombie28.png");

    zombieesquerda= new GreenfootImage[8];

    zombieesquerda[0]= new GreenfootImage("zombie21esquerda.png");
    zombieesquerda[1]= new GreenfootImage("zombie22esquerda.png");
    zombieesquerda[2]= new GreenfootImage("zombie23esquerda.png");
    zombieesquerda[3]= new GreenfootImage("zombie24esquerda.png");
    zombieesquerda[4]= new GreenfootImage("zombie25esquerda.png");
    zombieesquerda[5]= new GreenfootImage("zombie26esquerda.png");
    zombieesquerda[6]= new GreenfootImage("zombie27esquerda.png");
    zombieesquerda[7]= new GreenfootImage("zombie28esquerda.png");
}
/**
 * Funnção que move o zombie em cima da plataforma
```

```
 */
public void moveZombiePlataforma1() {
    if (atBordaDireita() && direita) {
        direita = false;
    } else if (atBordaEsquerda() && !direita) {
        direita = true;
    }

    if (direita) {
        setLocation(getX() + 5, getY());
        zombieanima();
    } else {
        setLocation(getX() - 5, getY());
        zombieanimaesquerda();
    }
}
public void moveZombiePlataforma2() {
    if (atBordaDireita2() && direita) {
        direita = false;
    } else if (atBordaEsquerda2() && !direita) {
        direita = true;
    }

    if (direita) {
        setLocation(getX() + 5, getY());
        zombieanima();
    } else {
        setLocation(getX() - 5, getY());
        zombieanimaesquerda();
    }
}
public void moveZombiePlataforma3() {
```

```
        if (atBordaDireita3() && direita) {
            direita = false;
        } else if (atBordaEsquerda3() && !direita) {
            direita = true;
        }

        if (direita) {
            setLocation(getX() + 5, getY());
            zombieanima();
        } else {
            setLocation(getX() - 5, getY());
            zombieanimaesquerda();
        }
    }
    /**
     * Função que percorre o array de imagens do zombie e anima-o
     */
    public void zombieanima() {
        counter++;
        if (counter > 10) {
            indice++;
            if (indice >= zombie.length) {
                indice = 0;
            }
            counter = 0;
        }
        setImage(zombie[indice]);
    }
    /**
     * Função que percorre o array de imagens do zombie e anima-o mas para o lado esquerdo
     */
```

```
    public void zombieanimaesquerda() {
        counter++;
        if (counter > 10) {
            indice++;
            if (indice >= zombieesquerda.length) {
                indice = 0;
            }
            counter = 0;
        }
        setImage(zombieesquerda[indice]);
    }
    /**
     * Função que verifica se o zombie esta em cima da plataforma
     */
    public boolean verificaPlataforma1() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma1.class);
        return emCima != null;
    }
    public boolean verificaPlataforma2() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma2.class);
        return emCima != null;
    }
    public boolean verificaPlataforma3() {
        Actor emCima = getOneObjectAtOffset(0, getImage().getHeight() / 2, Plataforma3.class);
        return emCima != null;
    }
    /**
     * Função que verifica se o zombie está na borda direita da plataforma
     */
    public boolean atBordaDireita() {
```

```
        Actor bordaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma1.class);
        return bordaDireita == null;
    }
    public boolean atBordaDireita2() {
        Actor bordaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma2.class);
        return bordaDireita == null;
    }
    public boolean atBordaDireita3() {
        Actor bordaDireita = getOneObjectAtOffset(getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma3.class);
        return bordaDireita == null;
    }

    /**
     * Função que verifica se o zombie está na borda esquerda da plataforma
     */
    public boolean atBordaEsquerda() {
        Actor bordaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma1.class);
        return bordaEsquerda == null;
    }
    public boolean atBordaEsquerda2() {
        Actor bordaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma2.class);
        return bordaEsquerda == null;
    }
    public boolean atBordaEsquerda3() {
        Actor bordaEsquerda = getOneObjectAtOffset(-getImage().getWidth() / 2, getImage().getHeight() / 2, Plataforma3.class);
        return bordaEsquerda == null;
    }
```

```
    public int getVida()
    {
        return vida;
    }

    public void tiraVida(int valor)
    {
        vida-=valor;
    }
    /**
     * Função que remove o zombie do mundo quando morre
     */
    public void morre(){
        if (vida <= 0){
            getWorld().removeObject(this);
        }
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Fim here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Fim extends World
{

    /**
     * Constructor for objects of class Fim.
     *
```

```
     */
    public Fim()
    {
        super(1200, 700, 1);
        Back botaoBack = new Back();
        addObject(botaoBack,getWidth()/2, 500);
    }
}
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Instruções here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Instrucoes extends World
{

    /**
     * Constructor for objects of class Instruções.
     *
     */
    private GreenfootImage mensagemNivel;
    private GreenfootImage background = new GreenfootImage("background1.jpg");
    private final int TamanhoTexto=50;
    private int contadorTempo;
    private static final int TEMPO_VISIVEL = 200;
    public Instrucoes()
    {
        super(1200, 700, 1);
        setBackground("Instrucoes1.png");
```

```
        ColocaInstruções();
    }
    /**
     * Função que coloca as instruções, teclas e indicações
     */
    private void ColocaInstruções()
    {
        Color cinzento = new Color(128, 128, 128);


        addObject(new Texto("Player 1", TamanhoTexto, Color.WHITE, cinzento), getWidth() / 3+50, getHeight() / 6);
        addObject(new Texto("Player 2", TamanhoTexto, Color.WHITE, cinzento), (2 * getWidth()) / 3, getHeight() / 6);
        addObject(new Texto("Saltar", TamanhoTexto, Color.WHITE, cinzento), getWidth() / 8, (2 * getHeight()) / 6);
        addObject(new Texto("Esquerda", TamanhoTexto, Color.WHITE, cinzento), getWidth() / 8, (3 * getHeight()) / 6);
        addObject(new Texto("Direita", TamanhoTexto, Color.WHITE, cinzento), getWidth() / 8, (4 * getHeight()) / 6);
        addObject(new Texto("Atirar", TamanhoTexto, Color.WHITE, cinzento), getWidth() / 8, (5 * getHeight()) / 6);


        addObject(new Caixa(1,0,"Salta1.png"), getWidth()/3+50, (2*getHeight())/6);
        addObject(new Caixa(1,1,"Esquerda1.png"), getWidth()/3+50, (3*getHeight())/6);
        addObject(new Caixa(1,2,"Direita1.png"), getWidth()/3+50, (4*getHeight())/6);
        addObject(new Caixa(1,3,"Atirar1.png"), getWidth()/3+50, (5*getHeight())/6);
        addObject(new Caixa(2,0,"Salta2.png"), (2*getWidth())/3, (2*getHeight())/6);
        addObject(new Caixa(2,1,"Esquerda2.png"), (2*getWidth())/3, (3*getHeight())/6);
        addObject(new Caixa(2,2,"Direita2.png"), (2*getWidth())/3, (4*getHeight())/6);
        addObject(new Caixa(2,3,"Atirar2.png"), (2*getWidth())/3, (5*getHeight())/6);


        Back botaoBack = new Back();
        addObject(botaoBack,55,50);
    }
```

```
    public void act(){
        contadorTempo++;
        if (contadorTempo == TEMPO_VISIVEL) {
            setBackground("Instrucoes2.png");
        }
    }
}
```