

ATIVIDADES PRÁTICAS SUPERVISIONADAS

DESENVOLVIMENTO DE UM JOGO COM UTILIZAÇÃO DE INTERFACE GRÁFICA

**3º Ciência da Computação (CC)**

## Elaboração:

Breno Luiz RA: T571DH2

Bruno Luiz RA: T406AE5

José Trevisan RA: T571DD0

Lucca Moreira RA: T571CF9

Sumário

1 Objetivo3

2 Introdução4

2.1 Lore4 a 5

3 Regras e funcionamento do jogo (conceitos gerais)6 a 9

**4 Plano de desenvolvimento do jogo10**

4.1 Plano10 a 12

4.2 Recursos Utilizados12 a 13

4.3 Plataforma de Desenvolvimento13 a 14

4.4 Estrutura do Mapa14

5 Estrutura do programa15

5.1 Módulos15 a 16

5.2 Fluxo de Trabalho16 a 17

5.3 Materiais17

**6 Relatório com as linhas de código18 a 27**

**8 Bibliografia28**

**Objetivo**

Este trabalho tem como propósito a criação de um jogo de espaçonave utilizando Java, com ênfase na conscientização ambiental e na limpeza do espaço. A ideia é fundir entretenimento e educação, oferecendo aos jogadores uma experiência que os faça refletir sobre a importância da preservação ambiental.

O desafio consiste em desenvolver um jogo que aborde diversos aspectos da educação ambiental, como a necessidade de reciclagem, a proteção da biodiversidade, a redução do consumo de recursos naturais e a mitigação da poluição. Através de mecânicas de jogo criativas e interativas, pretendemos envolver os jogadores em situações virtuais que simulem dilemas e desafios ambientais comuns, encorajando-os a buscar soluções sustentáveis e a desenvolver habilidades de tomada de decisão responsável.

No âmbito técnico, este projeto será desenvolvido utilizando os princípios da Programação Orientada a Objetos (POO), que permite a modelagem do jogo através de objetos com comportamentos específicos. Serão explorados temas como a encapsulação, herança e polimorfismo para criar uma estrutura de código organizada e coesa. Além disso, é feito o uso de bibliotecas gráficas como JavaX para criar uma interface visual atrativa e intuitiva. A estruturação do código será realizada de maneira a facilitar sua manutenção e escalabilidade, garantindo uma base sólida para futuras expansões e atualizações do jogo.

Por último, o objetivo não é apenas criar um jogo funcional, mas também proporcionar uma experiência educativa significativa, permitindo que os participantes aprimorem suas habilidades de programação e adquiram conhecimentos práticos sobre o desenvolvimento de software orientado a objetos com interface gráfica em Java.

**Introdução**

RecySpace é uma homenagem aos jogos de nave clássicos dos fliperamas, mantendo a essência da jogabilidade simples que marcou uma era. Neste jogo, você assume o papel de um piloto da NaveLix, encarregado de proteger a Terra no espaço sideral.

Sua missão é reciclar detritos espaciais enquanto enfrenta obstáculos em uma jornada pela preservação do meio ambiente espacial. À medida que você mergulha nesta aventura, será desafiado a demonstrar coragem e habilidade para superar os perigos do cosmos.

Mergulhe em um universo que resgata a nostalgia dos fliperamas, ao mesmo tempo em que destaca a importância da reciclagem e da proteção ambiental. É experiência que combina diversão e consciência ambiental.

Lembre-se de que a última esperança da Terra está em suas mãos. Sua determinação e coragem serão fundamentais para garantir um futuro sustentável para as gerações futuras.

**Lore**

Num futuro distante, a Terra está à beira do colapso ambiental devido à negligência e à exploração excessiva dos recursos naturais. As cidades estão envoltas em poluição e desperdício. Os líderes mundiais lançaram uma última esperança: NaveLixs.

O plano NaveLix consiste em uma frota de naves espaciais equipadas com tecnologia avançada de reciclagem. Seu objetivo é limpar o espaço ao redor da Terra, que se tornou um depósito de lixo espacial, e reciclar os detritos para transformá-los em recursos vitais para a sobrevivência da humanidade.

Os jogadores assumem o papel dos pilotos dessas naves de reciclagem, conhecidos como "Defensores". Cada *player* é encarregado de patrulhar uma área específica do espaço, enfrentando obstáculos perigosos, incluindo fragmentos de espaçonaves destruídas, destroços de satélites e detritos deixados para trás por anos de exploração espacial.

Porém, há mais do que apenas lixo no espaço. Os *players* devem estar preparados para defender a Terra não apenas contra os perigos do espaço, mas também contra aqueles que desejam explorar sua fragilidade.

À medida que os jogadores avançam no jogo, descobrem segredos ocultos sobre a verdadeira extensão da destruição ambiental e os esforços desesperados para restaurar o equilíbrio ecológico. Através de suas ações como Defensores, os jogadores têm a chance de fazer a diferença e salvar o planeta da iminente catástrofe.

**Regras e funcionamento do jogo (conceitos gerais)**



RecySpace, apresenta uma história centrada na sustentabilidade e na saúde pública em um ambiente espacial. O jogo é solo, onde o player navega por um mapa espacial. O objetivo é desintegrar enxames de lixos espaciais onde uma parte é reciclada para uso dos seres humanos.

**Regras e Mecânicas do Jogo:**

Os jogadores desintegram detritos espaciais jogados no espaço com o projétil futurístico que desintegra 90% do que atinge. Esses detritos representam uma ameaça ao meio ambiente espacial e à segurança das espaçonaves. Os lixos do espaço criam condições perigosas para a navegação, aumentando o risco de colisões e danos às naves.

Coleta de Detritos: Os 10% restantes do que o projétil atinge é convertido em reciclagem. Onde é mostrado o quanto foi coletado pelo Score na parte superior-esquerda do jogo:

Papel: 5 pontos.

Plástico: 10 pontos.

Vidro: 25 pontos.

Metal: 50 pontos.

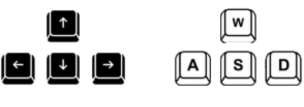
Existem 4 tipo de ‘tiros’, um projétil para cada tipo de detrito. A quantidade de tiro deles é infinito, só sendo limitado ao delay de uso.



Como existem mais detritos do que podem ser combatidos, o jogador deve usar a estratégia para conseguir a maior pontuação possível sem ser atingido.

Caso um dos detritos colidir com a NaveLix, o jogador perde uma das 3 vidas que têm, assim que perder todas as vidas, o jogador perde a partida e é encaminhado para a tela de Game Over, onde se quiser voltar a jogar, deve apertar a letra ‘R’ (Na tela de fim de jogo, o jogador é avisado disto).

A nave é controlada pelo jogador, onde pode subir (tecla W ou seta para cima), descer (tecla S ou tecla para baixo), avançar (tecla D ou seta para a direita) e até mesmo voltar (tecla A ou seta para a esquerda).



Assim que o jogador entra no jogo, é atingido, dispara projéteis, perde vida e/ou até mesmo quando ele perde por completo, um efeito sonoro diferente é ativado. Tanto para melhorar a experiência de jogo, quanto para alertar o jogador das ameaças pelo caminho.

O mapa é feito no espaço, onde tudo é escuro e há as estrelas para iluminar o caminho da NaveLix. Quanto mais o jogador avança, mais lixos são encontrados no caminho.

Sobre a perspectiva educativa, na mecânica do jogo, é coletado diferentes tipos de resíduos no próprio espaço (papel, plásticos, vidros e metais) que até então, servem tanto para ajudar no cenário futurístico das novas metrópoles, quanto para criar até bases espaciais para os NaveLixs.

O jogo aborda a importância da reciclagem e a segregação correta de lixos. Promovendo assim, a compreensão da biodiversidade e os esforços necessários para a conservação da natureza na Terra e a humanidade.

Ao unir componentes de jogo clássicos com mensagens relevantes sobre ecologia e preservação do meio ambiente, o jogo se torna um meio divertido de aprendizado e sensibilização. Ele encoraja as pessoas, fazendo com que elas tenham o hábito de reutilizar materiais e a reduzir a poluição.

Em conclusão, este jogo não apenas oferece entretenimento, mas também desempenha um papel fundamental na conscientização e na promoção de práticas ambientalmente responsáveis.

Ao educar os jogadores sobre a importância da reciclagem, da segregação adequada de resíduos e da conservação da biodiversidade, ele inspira mudanças positivas de comportamento.

Ao adotar abordagens criativas e envolventes, como a gamificação, podemos alcançar uma audiência mais ampla e motivá-la a agir em prol do nosso planeta.

Portanto, este jogo representa uma valiosa contribuição para a causa da sustentabilidade e da preservação ambiental, capacitando indivíduos a se tornarem agentes ativos de mudança em suas comunidades e além.

**Plano de desenvolvimento do jogo**

**Plano:**

Inicialmente, nosso grupo se reuniu e conversou muito para que diversas ideias de jogos pudessem ser desenvolvidas dentro do prazo e dos recursos disponíveis. As ideias foram avaliadas com os integrantes do grupo, viabilidade técnica e no impacto ambiental que poderiam causar, considerando que queríamos promover uma mensagem ecológica. Entre as sugestões iniciais. Acabamos que escolhemos a nave, mas teve algumas ideias além dessa.

Pac-Man: Uma das ideias propostas foi desenvolver um jogo inspirado no clássico Pac-Man, mas com um toque ecológico. Nesse conceito, ao invés de comer pac-dots (pontos pequenos) em um labirinto, o Pac-Man seria responsável por coletar resíduos que caem do céu. Onde teria sua principal função eliminar os resíduos e não deixar de se levar com os lixos ou inimigos que estaria por vir.

Limpeza do Rio: Um jogo onde o jogador navega por um rio em um barco ecológico, recolhendo resíduos flutuantes. Ou onde ficaria em um deck e teria a missão de que com uma vara de pesca arremessaria sua linha em cima do resíduos e coletando-os, lutando contra o tempo e somando pontos com cada coleta.

Herói da Floresta: Neste jogo, o jogador controla um guardião da floresta que precisa coletar lixo deixado por visitantes antes que os animais sofram as consequências. Com intuito de não deixar os animais comerem, pois se não eles acabariam morrendo e perderia ponto, mas que também a chegar perto dos resíduos e coletasse somaria pontos.

Jogo da Memória: Teria o intuito de memorizar os resíduos onde estaria em cartas e com um temporizador as cartas virariam para memorizar e logo em seguida teria que acertar sua dupla (o mesmo desenho), um jogo de 3 fases com o passar das fases o tempo de memorização diminuiria, dificultando o jogador a memorizar o local correto das cartas

Ao final das discussões, consideramos detalhadamente os pontos de interface, jogabilidade e impacto ambiental de todas as sugestões. Analisamos como cada ideia poderia ser implementada, as dificuldades técnicas envolvidas e a maneira como transmitiriam a mensagem ecológica.

Após uma avaliação cuidadosa, a proposta de desenvolver um jogo 2D com uma nave espacial eliminando resíduos no espaço foi considerada a mais adequada e atraente pelo grupo. Este conceito não só proporciona uma jogabilidade envolvente, mas também reforça de maneira eficaz a importância da reciclagem e do cuidado com o meio ambiente. Onde foi inspirada em uma ideia que com um planeta terra prejudicado de forma precária, começaram a descartar esses resíduos no espaço contaminando-o.

Onde ao final de tudo, elaboramos em uma folha de rascunho mesmo, quais resíduos usar, qual elemento a nave atiraria para eliminá-los, e como seria a interface e o conceito do jogo. Nessa etapa, identificamos os seguintes pontos:

Design do Jogo e Imagens que entraria no contexto: Procurar imagens que fazer iam parte da temática do jogo e um design que não fugiria do contexto.

Lógica e Mecânica de Jogo: Definir as regras, e o comportamento da nave com relação os resíduos.

Programação e Implementação: Definir entidades necessárias e seus comportamentos para não ocorrer erros. Em seguida, iniciar o desenvolvimento do código necessário para implementar esses comportamentos, utilizando técnicas de programação adequadas para garantir a funcionalidade correta.

Testes e Ajustes: Realizar testes contínuos para identificar e corrigir bugs gráficos e ajustar algo que possa ter saído fora de controle.

**Recursos Utilizados:**

Bibliotecas Java AWT e Swing.

Renderização Gráfica: forneceram ferramentas para renderizar gráficos e criar uma interface gráfica interativa. Utilizamos essas capacidades para os elementos visuais do jogo, como a nave espacial, os resíduos no espaço e os obstáculos.

Componentes de Interface: Podemos criar componentes de interface, como botões, caixas de texto e áreas de desenho. Esses componentes foram essenciais para criar a interface do jogador e permitir interações com o jogo.

Tratamento de Eventos: Nos deu uma ampla funcionalidades para lidar com eventos de entrada do usuário, como cliques do mouse e pressionamentos de tecla. Utilizamos esses recursos para detectar e responder às ações do jogador, como movimento da nave.

Desenvolvimento de Interface Gráfica: Através das ferramentas fornecidas, projetamos e implementamos a interface gráfica do jogo, incluindo layouts de tela, posicionamento de elementos e organização visual.

Controle de Janela: Utilizamos os recursos de controle de janela para gerenciar a exibição do jogo na tela e manipular os elementos visuais dentro dele.

Integração de áudio: Utilizamos a API Java Sound. Esta API nos permitiu carregar e reproduzir arquivos de áudio em formatos como WAV e MP3, proporcionando uma experiência sonora envolvente para os jogadores.

Nos proporcionou um ambiente robusto para o desenvolvimento do jogo, permitindo a criação de uma interface gráfica interativa e a implementação das funcionalidades necessárias para uma experiência de jogo envolvente e dinâmica elaborada.

**Plataforma de Desenvolvimento:**

IntelliJ IDEA

O IntelliJ IDEA foi nossa escolha como ambiente de desenvolvimento integrado (IDE) principal. Sua versatilidade e recursos avançados para o desenvolvimento em Java foram fundamentais para o progresso do nosso projeto. Com sua interface intuitiva

O recurso de autocompletar do acelerou significativamente o desenvolvimento. Essa funcionalidade não apenas economizou tempo, mas também ajudou a evitar erros de digitação e a garantir a consistência do código.

Refatoração Simplificada: Simplificou a manutenção do código, permitindo ajustes rápidos e precisos.

Depuração Integrada Eficiente: Essencial para identificar e resolver problemas de código, durante o desenvolvimento.

Encapsulamento de Código: Facilitou o processo de encapsulamento do código-fonte do jogo. Com recursos intuitivos e assistentes de refatoração, conseguimos encapsular eficientemente.

Funcionalidades Implementadas:

Adaptabilidade de Tela: Ele é redimensionável, o que significa que ele é capaz de se adaptar dinamicamente a diferentes resoluções de tela e proporções, garantindo uma experiência de jogo consistente e otimizada em uma variedade de dispositivos.

**Estrutura do Mapa:**

Obstáculos: São os próprios resíduos, onde a maioria fica parado e um se movimenta, onde o jogador terá de ir eliminando atirando.

Score: Com a eliminação de cada resíduo o score subira de acordo com cada pontuação.

Resíduos: Papel= 5 pontos.

Plástico= 10 pontos.

Vidro= 25 pontos.

Metal= 50 pontos.

Vida: 3 corações a canto inferior esquerdo da tela de acordo com o jogador a cada trombada nos resíduos ira perder uma vida ate levar o game over.

Nave: Controlada pelo jogador a principal do jogo, onde terá de controlador usando as setas e atirando usando as teclas Z, X, C e V.

**Estrutura do Programa**

## 1. Módulo de Personagens

### Funções:

* **Criação da Nave:** A nave do jogador é criada com base em parâmetros definidos, como tamanho e velocidade. A nave é personanalizada com diferentes tipos de armamentos, que influenciam sua eficiência na coleta e reciclagem de detritos.
* **Criação de Detritos:** Detritos espaciais são gerados com características variadas, como tamanho, material e valor reciclável.

## 2. Módulo de Mundo

### Funções:

* **Criação de um Mapa Tematizado:** O mapa é gerado com estrelas e um fundo, lembrando o espaço.
* **Gerenciamento de Detritos e Obstáculos:** O módulo controla a distribuição de detritos.

## 3. Módulo de Combate

### Funções:

* **Sistema de Recolhimento de Detritos:** As naves usam de um mecanismo para que possa recolher diferentes detritos.
* **Obstáculos e Inimigos:** Desenvolvido para controlar inimigos que protegem certos detritos ou atacam a nave do jogador. Os obstáculos também são gerenciados para criar desafios dinâmicos.
* **Gestão de Reciclagem e Pontuação:** O desempenho do jogador na reciclagem de detritos é avaliado com base na eficiência e rapidez, atribuindo pontos conforme a qualidade da reciclagem.

## 4. Módulo de Missões

### Funções:

* **Progredir a Pontuação:** As missões são estruturadas para desafiar o jogador a melhorar continuamente sua pontuação.
* **Aprender Qual Tipo de Armamento é Eficaz:** Ensina ao jogador qual tipo de armamento ou ferramenta é mais eficaz para desintegrar e recolher diferentes tipos de detritos, incentivando a experimentação e o aprimoramento de estratégias.

## 5. Módulo de Interface de Usuário (UI)

### Funções:

* **Navegação Intuitiva pelos Menus:** Desenvolve uma interface de usuário amigável, facilitando a navegação pelos diversos menus do jogo.
* **Exibição de HUD (Heads-Up Display):** O HUD fornece informações essenciais, como status da nave, quantidade de detritos coletados, pontos de reciclagem e alertas de inimigos.
* **Gerenciamento de Sons e Efeitos Visuais:** Integra efeitos sonoros e visuais que enriquecem a experiência do jogador, tornando o jogo mais imersivo e agradável.

# Fluxo de Trabalho

## 1. Exploração Espacial

* **Navegação Através do Mapa:** O jogador explora uma região do espaço, enfrentando desafios e coletando detritos para reciclagem.
* **Coleta de Detritos para Reciclagem:** A principal atividade durante a exploração é a coleta de detritos.

## 2. Missões de Reciclagem

* **Foco na Coleta de Detritos:** As missões são projetadas para maximizar a quantidade de detritos recolhidos, incentivando o jogador a desenvolver habilidades de coleta e reciclagem eficientes.

## 3. Combate e Desintegração

* **Enfrentamento de Obstáculos e Inimigos:** Durante a coleta, o jogador enfrentará obstáculos e inimigos que devem ser superados para continuar a reciclagem de detritos.

# Recursos Necessários

## Materiais:

* **Softwares de Desenvolvimento (Java, JavaX):** A escolha de Java e JavaX permite a criação de um ambiente robusto e multiplataforma para o desenvolvimento do jogo.
* **Recursos Gráficos(Swing) e Efeitos Sonoros:** Para uma experiência imersiva, são necessários gráficos de alta qualidade e efeitos sonoros realistas, que podem ser obtidos através de bibliotecas específicas ou desenvolvidos internamente.

**Relatório com as Linhas de Código**

package trashgame.Modelo; // Declaração do pacote onde a classe está contida

import java.awt.Graphics; // Importação das classes necessárias para manipulação gráfica

import java.awt.Graphics2D;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import javax.swing.\*;

import javax.sound.sampled.AudioInputStream; // Importação das classes necessárias para reprodução de áudio

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.Clip;

import javax.sound.sampled.LineUnavailableException;

import javax.sound.sampled.UnsupportedAudioFileException;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList; // Importação das classes necessárias para usar Listas

import java.util.List;

// Declaração da classe Fase, que estende JPanel e implementa ActionListener

public class Fase extends JPanel implements ActionListener {

private Image fundo; // Declaração de variável para armazenar a imagem de fundo

private Player player; // Declaração de variável para o jogador

private List<Enemies.Enemy1> enemy1; // Declaração de lista para os inimigos de tipo 1

//O mesmo para os outros tipos de detritos.

private List<Stars> stars; // Declaração de lista para as estrelas

private boolean emJogo; // Variável para controlar se o jogo está em execução

private Score score; // Declaração de variável para a pontuação do jogador

private boolean gameRestart; // Variável para controlar se o jogo deve ser reiniciado

private boolean somGameOverReproduzido = false; // Variável para controlar se o som de fim de jogo foi reproduzido

private Image coracao; // Declaração de variável para a imagem do coração

// Construtor da classe Fase

public Fase() {

try {

File soundFile = new File("res\\musicafundo.wav"); // Carrega o arquivo de áudio de fundo

AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(soundFile); // Cria um fluxo de áudio a partir do arquivo

Clip clipMusica = AudioSystem.getClip(); // Cria um objeto Clip para reproduzir o áudio

clipMusica.open(audioInputStream); // Abre o arquivo de áudio no Clip

clipMusica.loop(Clip.LOOP\_CONTINUOUSLY); // Faz o áudio repetir continuamente

clipMusica.start(); // Inicia a reprodução do áudio

} catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) { // Tratamento de exceções

e.printStackTrace(); // Imprime informações sobre a exceção

}

setFocusable(true); // Permite que o JPanel receba foco

setDoubleBuffered(true); // Ativa o buffer duplo para desenho suave

ImageIcon referencia = new ImageIcon("res\\telapreta.png"); // Carrega a imagem de fundo

fundo = referencia.getImage(); // Obtém a imagem do ImageIcon

ImageIcon coracaoIcon = new ImageIcon("res\\coracao.png"); // Carrega a imagem do coração

coracao = coracaoIcon.getImage(); // Obtém a imagem do ImageIcon

player = new Player(); // Inicializa o jogador

player.load(); // Carrega a imagem do jogador

addKeyListener(new TecladoAdapter()); // Adiciona um KeyListener para capturar eventos do teclado

Timer timer = new Timer(5, this); // Cria um Timer que chama o método actionPerformed a cada 5 milissegundos

timer.start(); // Inicia o Timer

inicializaInimigos(); // Inicializa os inimigos

inicializaStars(); // Inicializa as estrelas

emJogo = true; // Define o jogo como em execução

gameRestart = false; // Define que o jogo não deve ser reiniciado

score = new Score(); // Inicializa o objeto de pontuação

}

// Método para inicializar os inimigos

public void inicializaInimigos() {

enemy1 = new ArrayList<>(); //Inicializa a lista de inimigos do tipo1

//Mesma coisa para enemy2, 3 e 4.

// Loop para criar e adicionar inimigos do tipo 1 em posições aleatórias

for (int i = 0; i < 50; i++) {

int x = (int) (Math.random() \* 20000 + 1920); // Gera uma posição x aleatória

int y = (int) (Math.random() \* 850 + 10); // Gera uma posição y aleatória

enemy1.add(new Enemies.Enemy1(x, y)); // Adiciona um novo inimigo do tipo 1 à lista

}

// Loop semelhante para os inimigos dos tipos 2, 3 e 4

}

// Método para atualizar os inimigos

public void updateInimigos() {

// Loop para atualizar cada inimigo do tipo 1 na lista

for (Enemies.Enemy1 inimigo : enemy1) {

inimigo.update(); // Atualiza o inimigo

}

// Loops semelhantes para os inimigos dos tipos 2, 3 e 4

// ...

}

// Método para inicializar as estrelas

public void inicializaStars() {

int[] coordenadas = new int[300]; // Array para armazenar coordenadas

stars = new ArrayList<>(); // Inicializa a lista de estrelas

// Loop para criar e adicionar estrelas em posições aleatórias

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

int x = (int) (Math.random() \* 1920 + 0); // Gera uma posição x aleatória

int y = (int) (Math.random() \* 1536 + 0); // Gera uma posição y aleatória

stars.add(new Stars(x, y)); // Adiciona uma nova estrela à lista

}

}

// Método paint responsável por desenhar os elementos na tela

public void paint(Graphics g) {

Graphics2D graficos = (Graphics2D) g; // Conversão de g para Graphics2D para melhor desempenho e funcionalidades avançadas

if (emJogo) { // Se o jogo estiver em execução

graficos.drawImage(fundo, 0, 0, null); // Desenha o fundo na tela

graficos.setColor(Color.WHITE); // Define a cor do texto como branco

graficos.setFont(new Font("Arial", Font.BOLD, 20)); // Define a fonte do texto

graficos.drawString("Score: " + score.getScore(), 10, 30); // Desenha a pontuação na tela

// Desenha os corações representando as vidas do jogador

for (int i = 0; i < player.getVida(); i++) {

int coracaoX = 10; // Posição x inicial do coração

int coracaoY = 50; // Posição y do coração

g.drawImage(coracao, coracaoX + (i \* 40), coracaoY, this); // Desenha o coração na posição correta

}

// Desenha as estrelas na tela

for (Stars q : stars) {

q.load(); // Carrega a imagem da estrela

graficos.drawImage(q.getImagem(), q.getX(), q.getY(), this); // Desenha a estrela na posição x e y

}

graficos.drawImage(player.getImagem(), player.getX(), player.getY(), this); // Desenha o jogador na tela

// Desenha os tiros do jogador na tela

List<Tiro.Tiro1> tiros1 = player.getTiros1();

for (Tiro.Tiro1 m : tiros1) {

m.load(); // Carrega a imagem do tiro

graficos.drawImage(m.getImagem(), m.getX(), m.getY(), this); // Desenha o tiro na posição x e y

}

// Desenha os inimigos do tipo 1 na tela

for (Enemies.Enemy1 in : enemy1) {

in.load(); // Carrega a imagem do inimigo

graficos.drawImage(in.getImagem(), in.getX(), in.getY(), this); // Desenha o inimigo na posição x e y

}

// Outros loops semelhantes para desenhar os tiros, inimigos e inimigos do tipo 2, 3 e 4 na tela

// ...

} else { // Se o jogo não estiver em execução

ImageIcon fimJogo = new ImageIcon("res\\fimdejogo.png"); // Carrega a imagem de fim de jogo

graficos.drawImage(fimJogo.getImage(), 0, 0, null); // Desenha a imagem de fim de jogo na tela

graficos.setColor(Color.WHITE); // Define a cor do texto como branco

graficos.setFont(new Font("Arial", Font.BOLD, 20)); // Define a fonte do texto

graficos.drawString("Aperte R para Reiniciar o jogo", 800, 800); // Exibe a mensagem para reiniciar o jogo

// Reproduz o som de fim de jogo se ainda não tiver sido reproduzido

if (!somGameOverReproduzido) {

try {

File soundFile = new File("res/gameover.wav"); // Carrega o arquivo de áudio de fim de jogo

AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(soundFile); // Cria um fluxo de áudio a partir do arquivo

Clip clip = AudioSystem.getClip(); // Cria um objeto Clip para reproduzir o áudio

clip.open(audioInputStream); // Abre o arquivo de áudio no Clip

clip.start(); // Inicia a reprodução do áudio

somGameOverReproduzido = true; // Define que o som de fim de jogo foi reproduzido

} catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) { // Tratamento de exceções

e.printStackTrace(); // Imprime informações sobre a exceção

}

}

}

g.dispose(); // Libera os recursos gráficos

}

// Método actionPerformed, chamado pelo Timer a cada intervalo especificado

@Override

public void actionPerformed(ActionEvent e) {

// Se o jogo deve ser reiniciado

if (gameRestart) {

if (somGameOverReproduzido) {

somGameOverReproduzido = false; // Define que o som de fim de jogo não foi reproduzido

}

// Reinicializa os inimigos, estrelas, jogador e pontuação

inicializaInimigos();

inicializaStars();

player = new Player();

player.load();

score = new Score();

emJogo = true; // Define que o jogo está em execução

gameRestart = false; // Define que o jogo não deve ser reiniciado

}

player.update(); // Atualiza o jogador

// Loop para atualizar as estrelas

for (Stars on : stars) {

if (on.isVisivel()) {

on.update(); // Atualiza a estrela

}

}

// Outros loops semelhantes para atualizar os tiros e os inimigos do jogo

//

checarColisoes(); // Verifica colisões entre os elementos do jogo

updateInimigos(); // Atualiza os inimigos

repaint(); // Redesenha a tela

}

// Método para verificar colisões entre os elementos do jogo

public void checarColisoes() {

Rectangle formaNave = player.getBounds(); // Obtém a área de colisão do jogador

Rectangle formaEnemy1; // Declaração de variável para a área de colisão do inimigo do tipo 1

//

for (Enemies.Enemy1 tempEnemy1 : enemy1) { // Loop para cada inimigo do tipo 1

formaEnemy1 = tempEnemy1.getBounds(); // Obtém a área de colisão do inimigo do tipo 1

if (formaNave.intersects(formaEnemy1)) { // Verifica se houve colisão entre o jogador e o inimigo do tipo 1

player.perderVida(); // O jogador perde uma vida

tempEnemy1.setVisivel(false); // Define que o inimigo do tipo 1 não está mais visível

if (player.isVisivel()) { // Se o jogador ainda estiver visível após perder a vida

emJogo = false; // O jogo é encerrado

}

}

}

// Outros loops semelhantes para verificar colisões entre o jogador e os inimigos do tipo 2, 3 e 4

// ...

// Loop para verificar colisões entre os tiros do jogador e os inimigos do tipo 1

List<Tiro.Tiro1> tiros1 = player.getTiros1();

for (Tiro.Tiro1 tempTiro1 : tiros1) {

formaTiro1 = tempTiro1.getBounds(); // Obtém a área de colisão do tiro do jogador

for (Enemies.Enemy1 tempEnemy1 : enemy1) { // Loop para cada inimigo do tipo 1

formaEnemy1 = tempEnemy1.getBounds(); // Obtém a área de colisão do inimigo do tipo 1

if (formaTiro1.intersects(formaEnemy1)) { // Verifica se houve colisão entre o tiro e o inimigo do tipo 1

tempEnemy1.setVisivel(false); // Define que o inimigo do tipo 1 não está mais visível

tempTiro1.setVisivel(false); // Define que o tiro não está mais visível

score.incrementScore(5); // Incrementa a pontuação do jogador

}

}

}

// Outros loops semelhantes para verificar colisões entre os tiros do jogador e os inimigos do tipo 2, 3 e 4

// ...

// Loop para verificar colisões entre os tiros do jogador e os inimigos do tipo 4

List<Tiro.Tiro4> tiros4 = player.getTiros4();

for (Tiro.Tiro4 tempTiro4 : tiros4) {

formaTiro4 = tempTiro4.getBounds(); // Obtém a área de colisão do tiro do jogador

for (Enemies.Enemy4 tempEnemy4 : enemy4) { // Loop para cada inimigo do tipo 4

formaEnemy4 = tempEnemy4.getBounds(); // Obtém a área de colisão do inimigo do tipo 4

if (formaTiro4.intersects(formaEnemy4)) { // Verifica se houve colisão entre o tiro e o inimigo do tipo 4

tempEnemy4.setVisivel(false); // Define que o inimigo do tipo 4 não está mais visível

tempTiro4.setVisivel(false); // Define que o tiro não está mais visível

score.incrementScore(50); // Incrementa a pontuação do jogador

}

}

}

}

// Classe interna TecladoAdapter para tratar eventos do teclado

private class TecladoAdapter extends KeyAdapter {

@Override

public void keyPressed(KeyEvent e) { // Método chamado quando uma tecla é pressionada

player.keyPressed(e); // Chama o método de movimento do jogador com a tecla pressionada

if (e.getKeyCode() == KeyEvent.VK\_R) { // Se a tecla pressionada for R

gameRestart = true; // Define que o jogo deve ser reiniciado

}

}

@Override

public void keyReleased(KeyEvent e) { // Método chamado quando uma tecla é solta

player.keyRelease(e); // Chama o método de parar o movimento do jogador com a tecla solta

}

}

}

**Bibliografia**

RECICLAGEM. Disponível em

<https://www.todamateria.com.br/reciclagem/>. Acesso: 01 de abril de 2024.

A onda de calor e o planeta lixo. Disponível em

</https://revistaviverbrasil.com.br/a-onda-de-calor-e-o-planeta-lixo/>. Acesso: 02 de abril de 2024.

FreeGames2D Itch.io. Disponível em

<https://itch.io/games/free/tag-2d/>. Acesso: 09 de abril de 2024.

Terminal Root. Aprenda Java Criando um Jogo 2D com LibGDX. YouTube, 09 de abril de 2024. Disponível em: <https://www.youtube.com/watch?v=2bmvlwvnirk&list=PLUJBQEDDLNclxZvKTT2Icq9aVBPqLRMCA&index=2>

Tácio [Code]. Como Fazer um JOGO 2D em JAVA. YouTube, 10 de abril de 2024. Disponível em: <https://www.youtube.com/watch?v=5aE55KZ93xw&list=PLlW3qrNjsvBwUmUk9kio7bNT3GR554FH5&index=1>

SpaceX: o futuro da comercialização das viagens espaciais. Disponível em:</https://www.indioeduca.org/spacex-o-futuro-da-comercializacao-das-viagens-espaciais/>. Acesso: 12 de abril de 2024.

Lixo Espacial. Disponível em

</https://brasilescola.uol.com.br/geografia/lixo-espacial.htm/>. Acesso: 12 de abril de 2024.