CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Curso Técnico Integrado de Informática

Anna Luisa Andrade da Silva

Lucas Vinicios Santos Cruz

Victor Gabriel Ribeiro Mariano

Trabalho Prático do Segundo Bimestre: *Scrolling Game*

Contra

Contagem

2023

Anna Luisa Andrade da Silva

Lucas Vinicios Santos Cruz

Victor Gabriel Ribeiro Mariano

Trabalho Prático do Segundo Bimestre: *Scrolling Game*

Contra

Relatório técnico apresentado à disciplina de Laboratório de Linguagens e Técnicas de Programação II, ministrada pelo Prof. Alisson Rodrigo dos Santos.

Contagem

2023

Para nós, que nunca paramos até conseguir o que queremos.

AGRADECIMENTOS

Gostaríamos de expressar nossos mais sinceros agradecimentos a todos os que foram de grande ajuda para a elaboração deste trabalho. Agora, queremos dedicar nosso agradecimento aos nossos colegas de turma, que caminharam lado a lado conosco, compartilhando desafios, aprendizados ao longo do percurso.

Às nossas famílias, que estiveram ao nosso lado desde o início, oferecendo amor, suporte e encorajamento incondicionais, expressamos nossa gratidão profunda.

Aos nossos amigos, aqueles que estiveram presentes em todos os momentos, compartilhando, conselhos, apoios e experiências.

Por último, queremos estender nosso agradecimento ao nosso professor, Alisson Rodrigo dos Santos. Sua dedicação, conhecimento e paixão pela educação foram inspiradores. Você foi além das expectativas normais, nos desafiando a alcançar um nível mais elevado de excelência e nos orientando com sabedoria ao longo do processo de aprendizagem.

A todos vocês, colegas de turma, famílias, amigos e nosso professor, oferecemos nosso mais profundo agradecimento por fazerem parte deste trabalho, mas não só dele como também parte das nossas vidas. Estamos verdadeiramente gratos e honrados por compartilhar essa experiência com vocês. Que nossos caminhos continuem se cruzando e que possamos celebrar muitas conquistas futuras juntos.

A educação tem raízes amargas, mas os seus frutos são doces.

— Aristóteles

RESUMO

Este trabalho acadêmico proposto pelo professor Alisson Rodrigo dos Santos pretende recriar o jogo Contra utilizando a linguagem de programação Java e o framework de desenvolvimento de jogos libGDX. O projeto visa desenvolver habilidades de programação e a ampliação do conhecimento em Java e programação orientada a objetos (POO) por meio de técnicas de programação. O Contra é um jogo *shooter run-n-gun* lançado para o NES (Nintendinho) em 1987. O grupo visa desenvolver o jogo com base na POO utilizando classes, herança e polimorfismo. Durante o desenvolvimento, o grupo enfrentou desafios relacionados ao tempo, implementação de funcionalidades e trabalho em equipe, mas a maioria dos objetivos foram alcançados.

Palavras-chave: LibGdx, Contra, NeoContra, jogo, desenvolvimento, desenvolvedores, scrolling game.

ABSTRACT

The following academic project was proposed by Professor Alisson Rodrigo dos Santos and aims to recreate the game Contra using the Java programming language and the libGDX game development framework. The project targets to develop programming skills and to expand the knowledge about Java and object-oriented programming (OOP) by using programming techniques. Contra is a run-n-gun shooter game released for the NES (Nintendo Entertainment System) in 1987. The team aims to develop the game based on the OOP using classes, inheritance, and polymorphism. During the development, the team faced challenges related to time, implementing features and teamwork, but the majority of the objectives were achieved.

Keywords: LibGdx, Contra, NeoContra, game, development, developers, scrolling game.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

O desenvolvimento de jogos é uma das melhores formas de aprendizado quando se trata de programação. Desenvolver jogos traz desafios de programação complexos em várias áreas e demanda um conhecimento em várias áreas. Além disso, o desenvolvimento de jogos requer uma grande organização a longo prazo e dá ao programador conhecimento e experiência importantíssimos para projetos futuros e para seguir uma carreira como programador.

No trabalho proposto pelo professor Alisson Rodrigo dos Santos, "*Scrolling Game*", o objetivo é recriar um jogo em que há movi30mento de câmera, isto é, um jogo dinâmico. Para isso, o grupo deverá utilizar a linguagem de programação Java e a biblioteca libGDX, a qual é um *framework* de desenvolvimento de jogos multiplataforma escritos em Java e baseado em OpenGL.

Os objetivos gerais do trabalho são o uso da programação orientada a objetos, o uso da biblioteca libGDX, o uso de coleções de objetos *(arrays)*, ampliar o conhecimento da linguagem Java por meio de técnicas de programação e a recriação de um jogo escolhido pelo professor. Para atingir os objetivos gerais, o grupo deve dividir o programa em classes (orientação a objetos), utilizar a técnica de polimorfismo, utilizar nomes significativos para as variáveis e funções, utilizar as bibliotecas da libGDX, utilizar a estrutura da libGDX para a construção de animações, ter um conjunto de personagens que interagem com o jogador, organizar o conjunto de personagens em lista (sendo que o uso de uma estrutura de organização de coletâneas de Java é obrigatório), uma estrutura de evolução e o compartilhamento do desenvolvimento e da documentação via GitHub, adicionando o professor na equipe.

Com isso, é esperado que o grupo pesquise, aprenda e desenvolva técnicas de programação e organização em equipe, amplie significativamente o conhecimento da linguagem de programação Java e do paradigma de programação orientada a objetos (POO), além de adquirir experiência com projetos contributivos *(Git)*.

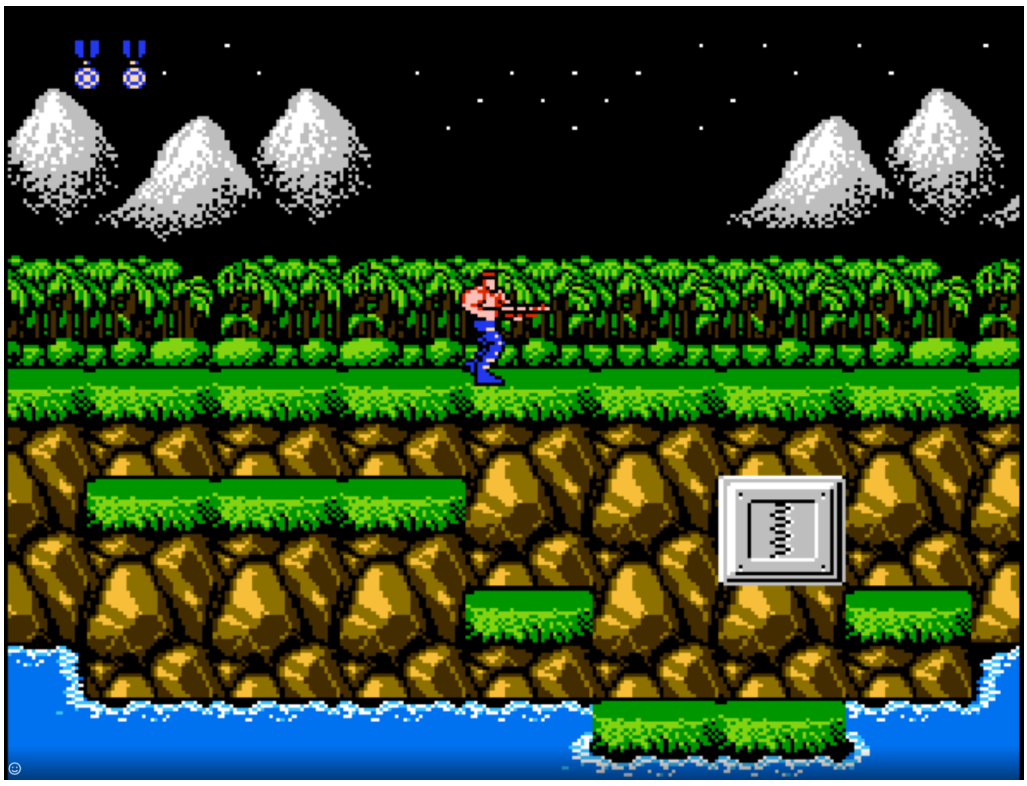
Por fim, a maioria dos objetivos foram cumpridos, porém, a equipe encontrou várias adversidades, seja em tempo, implementação de funcionalidades do jogo original a problemas de trabalho em equipe.

2. IMPLEMENTAÇÃO

O jogo recriado é chamado Contra, um jogo *shooter run-n-gun*, lançado para NES (também conhecido como Nintendinho) em 1987. Jogos *shooter run-n-gun* trata-se de jogos, geralmente de plataforma, em que o jogador tem que correr e atirar em inimigos para avançar. Um exemplo recente de um jogo *shooter run-n-gun* é o famoso Cuphead.

O Contra implementa em sua *gameplay* mecânicas novas e revolucionárias para o tempo em que foi lançado, tendo dois modos de jogo, dependendo da fase em que o jogador se encontra: plataforma *side-scrolliing*, como nos primeiros jogos da série *Super Mario Bros.*, em que o personagem é visto numa perspectiva lateral e a câmera se movimenta predominantemente no eixo x (Figura 1). E o modo de câmera traseira, como nos jogos da série *Super Mario Kart*, no qual a câmera se encontra sempre nas costas do personagem (Figura 2).

Figura 1- Side-scrolling no Contra.



Fonte: autoral.

Figura 2- Câmera traseira no Contra.

Fonte: Vídeo no YouTube[1].

O personagem, quando em *side-scrolling*, poderia atirar em 8 (oito) direções diferentes, sendo que, para atirar diretamente para baixo, era necessário estar pulando enquanto atirava para baixo, pois o personagem também poderia deitar-se no chão para desviar de projéteis inimigos ou alcançar lugares difíceis.

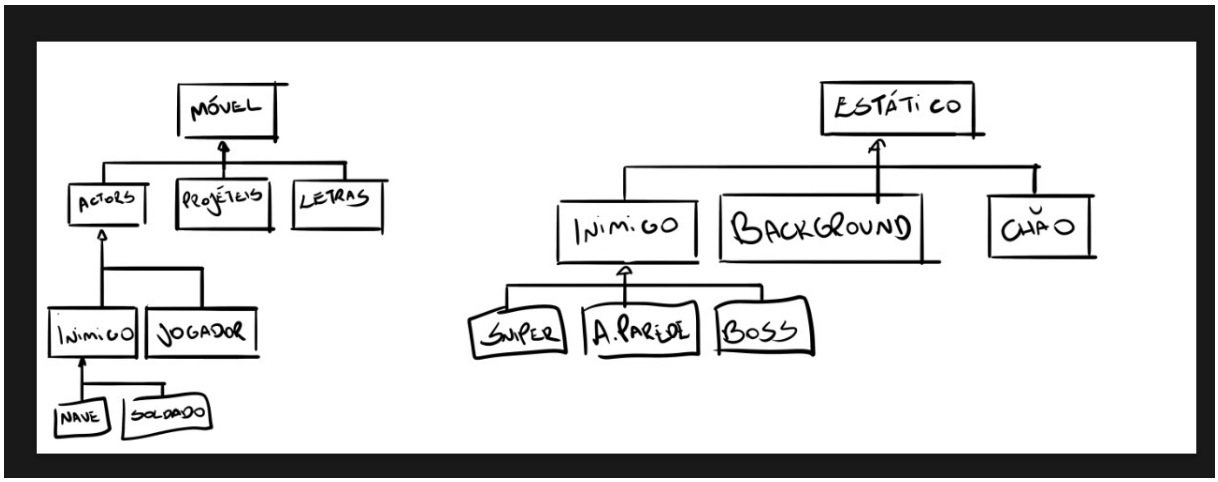
O jogo não possui sistema de pontos e as vidas do personagem são representadas por medalhas no canto superior direito da tela, tendo duas medalhas inicialmente, equivalendo a três vidas. Caso o personagem fique sem medalhas, lhe restará apenas uma última vida e caso morra, o jogo se encerrará e o jogador é mandado para a tela de *game-over*.

Tendo localizadas as mecânicas principais a serem implementadas na recriação, por não ser necessário implementar o jogo por completo, o grupo iniciou o desenvolvimento. O algoritmo é baseado no paradigma de orientação a objetos, então, antes de iniciar a

[1] PEB. Contra (NES) Full Run with No Deaths. YouTube, 26 set. 2013. Disponível em: https://www.youtube.com/watch?v=2mWZlNOzdv8. Acesso em: 1 jul. 2023.

programação, era necessário projetar o sistema. Para isso, alguns esboços de diagramas de classes e diagramas de pacote foram construídos.

Figura 3- Primeiro esboço de diagrama de classes.



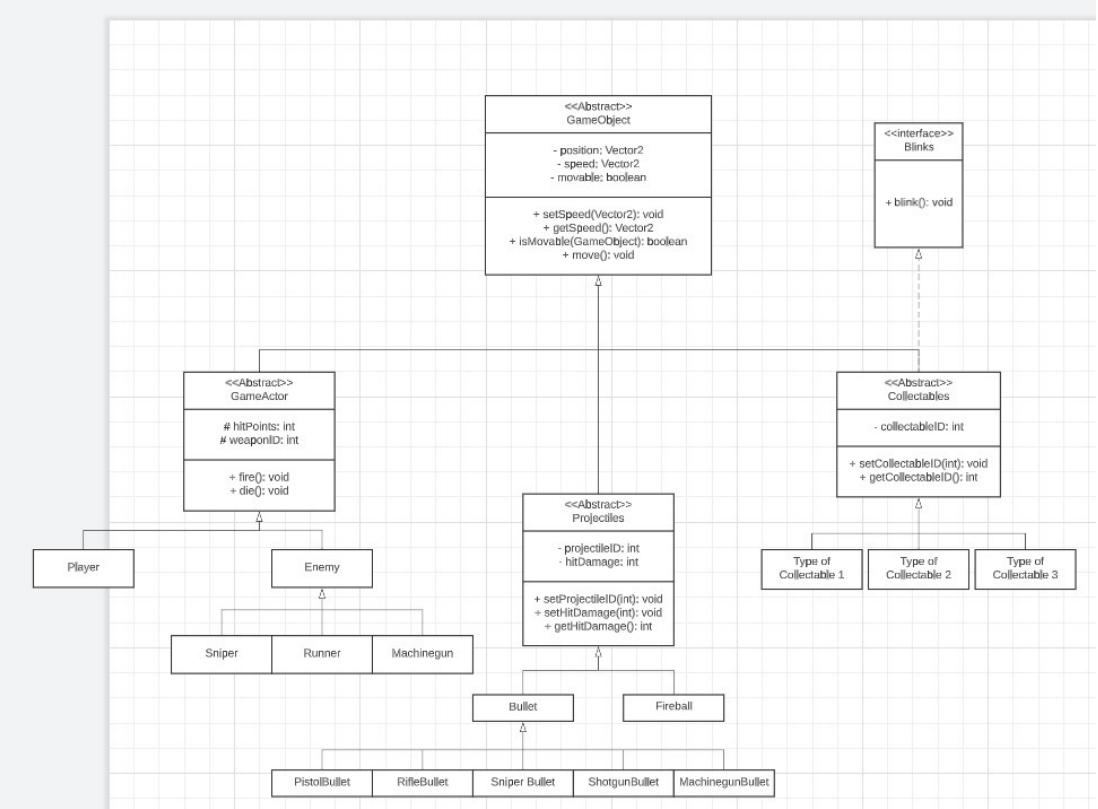
Fonte: autoral.

Nesta primeira aproximação (Figura 3) não foram consideradas as funcionalidades da libGDX. O intuito desta aproximação era somente entender a estrutura do jogo com base em uma análise rápida dos primeiros três estágios do jogo.

Ao estudar o jogo, foi cogitada a possibilidade de criar uma classe móvel que trataria das questões de movimentação, velocidade e trajetória dos objetos na tela. As classes que herdariam de móvel seriam *Actors*, que possui como filhas *Inimigo* e *Jogador*, a classe *Projéteis*, que cuidaria dos projéteis mandados pelos diversos corpos no jogo, como torretas, inimigos e o próprio jogador e a classe *Letras*, que seriam os *power-ups*.

Do outro lado, temos a classe *Estático*, que cuidaria de elementos que não se movem. Dessa classe herdariam outra classe de inimigos, a classe do fundo da tela, o *Background* e a classe do solo em que os corpos pisaram, o *Chão*. Após críticas do professor a essa estrutura, ela foi abandonada.

Figura 4- Segundo esboço de diagrama de classes.

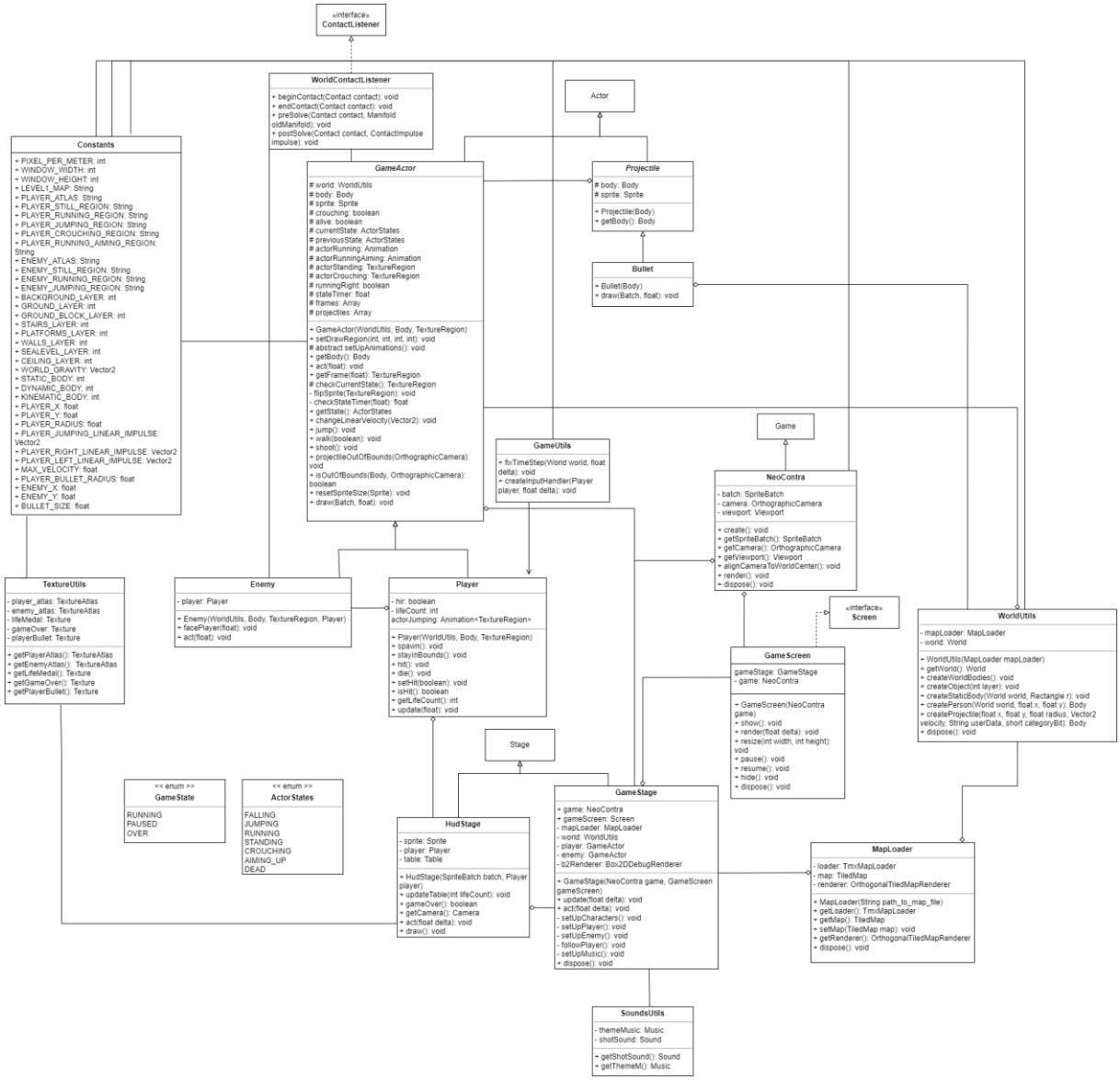


Fonte: autoral.

Na segunda aproximação (Figura 4), a estrutura foi reforçada, porém, os elementos da biblioteca ainda não eram considerados. Nesse esboço, a classe principal seria a *GameObject*, que cuidaria de elementos comuns à todas as classes, em vez de possuir uma classe para objetos móveis e imóveis.

A interface *Blinks* também foi criada, cuidando de objetos na tela que piscam, como os botões do menu ou os personagens em determinadas situações. As classes *Projéteis* e *Collectables* (anteriormente *Letras*) foram melhoradas e mais epecificadas. No entanto, esse esboço apresenta mais conteúdo do que seria possível desenvolver no tempo de 1 mês, mas foi bem aceito pelo professor. Portanto, o próximo diagrama teria de haver uma aproximação maior com o conteúdo que seria desenvolvido no jogo.

Figura 5- Diagrama de classes final.



Fonte: autoral

Devido à falta de experiência dos desenvolvedores com a projeção e análise de sistemas, o grupo decidiu que seria mais apropriado construir o diagrama de classes juntamente com o jogo, adaptando-os conforme a necessidade do momento. Em primeiro lugar, o grupo projetava o sistema que iria precisar, baseando-se nas suas fontes e, em seguida, aplicava no jogo, fazendo as alterações que fossem necessárias em ambos. Este diagrama (Figura 5) representa a totalidade do desenvolvimento do jogo.

2.1. Estrutura de Dados

As estruturas de dados utilizadas ao longo do projeto, são, na maioria, classes da própria biblioteca libGDX que adaptam as coletâneas do Java, como a classe *Array<>*, que se parece com a classe *ArrayList<>* do Java. Entretanto, essa estrutura de dados são otimizadas para evitar ao máximo a criação de lixo de memória, limpando-a com mais eficiência que as coletâneas do Java. Um exemplo, no código, que também pode ser visualizado no digrama de classes (Figura 5) é a criação de projéteis equivalentes aos tiros do jogador. Para o caso de haver vários inimigos na tela, essa coleção também seria utilizada.

Outro exemplo que também pode ser visualizado no diagrama de classes (Figura 5) é a estrutura de dados que a libGDX utiliza para construir animações, a *Animation<>*. Neste caso, foi utilizada uma animação de *TextureRegion* que são as regiões de um *sprite sheet* (folha de *sprites*) que correspondem a um frame da animação. Esta coletânea de dados costumava ser apenas uma classe comum nas versões mais antigas da biblioteca, porém, se tornou uma coleção recentemente.

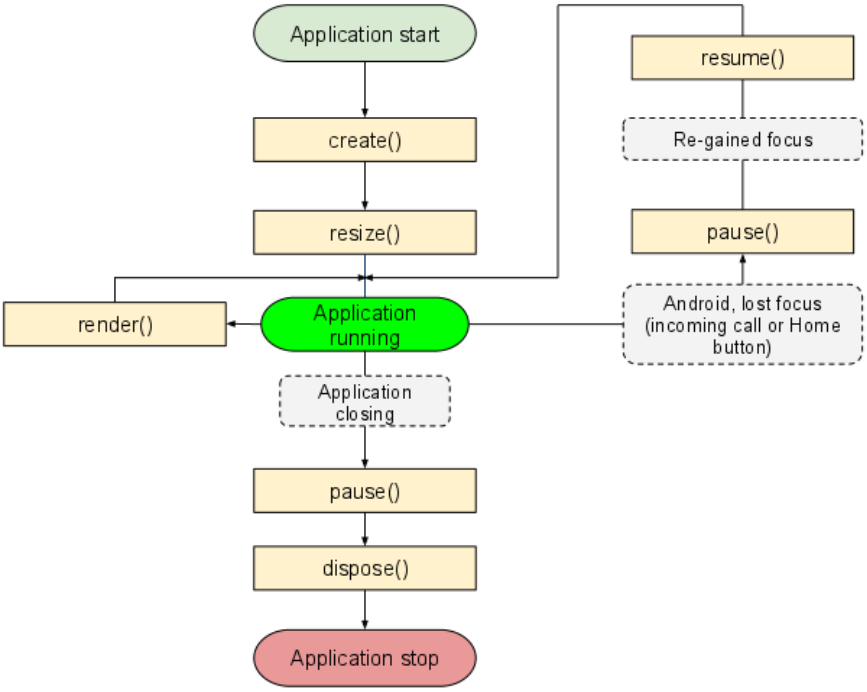
2.2. Funções e Procedimentos

As funções que compõem o programa desenvolvido são muitas, por isso, serão listadas abaixo apenas as funções mais necessárias para o bom funcionamento do jogo.

2.2.1. Classe NeoContra

public void create(): Inicializa as variáveis importantes para iniciar corretamente o jogo. Se uma variável essencial não for chamada primeiramente no método *create()*, o jogo não será executado. Isso porque ela faz parte da primeira parte do ciclo de vida da libGDX (Figura 6) e é anterior à execução do jogo propriamente dita.

Figura 6- Ciclo de vida da libGDX.



Fonte: Documentação oficial da libGDX[2].

2.2.2. Classe GameScreen

public GameScreen(NeoContra): Construtor da tela do jogo. Recebe o próprio jogo como parâmetro, para poder utilizar de suas variáveis internas, como a câmera e o *viewport*. Inicializa uma instância do jogo e uma instância do *GameStage*, o palco principal do jogo.

public void render(float): Renderiza na tela o que estiver na função e recebe como parâmetro um *float* chamado *deltaTime,* o qual é o cálculo do tempo realizado pelo motor gráfico de quanto demorou para que o último *frame* fosse processado e desenhado na tela. Sem essa função, não seria possível desenhar nada na tela e o jogo seria uma tela preta caso executasse.

[2] The life cycle. libGDX. Disponível em: https://libgdx.com/wiki/app/the-life-cycle. Acesso em: 3 jul. 2023.

2.2.3. Classe GameStage

public GameStage(NeoContra, GameScreen): Construtor do palco do jogo. Recebe como parâmetros o jogo e a tela onde será renderizado. Inicializa tudo desde os personagens, o mapa, a música, a *hud* e as configurações de câmera e *viewport*.

public void update(float): Faz a atualização de objetos que precisam ser atualizados constantemente, como a câmera, teste de limite de tela para projéteis, renderização do mapa, *inputs* e *timeStep*. Recebe como parâmetro o mesmo *deltaTime* citado anteriormente.

public void act(float): Chama o método correspondente de cada ator do palco para efetuar uma atualização como a do método *update()*, porém, este cuida especificamente da atualização dos atores.

public void dispose(): Joga fora o lixo de memória gerado durante o processo de execução do jogo, como indicado na Figura 6.

2.2.4. Classe GameUtils

public static void fixTimeStep(World, float): Acerta o passo das simulações de física, fazendo com que sejam calculadas conforme o *deltaTime*. Recebe como parâmetro o mundo a ter seu *timeStep* acertado e o *deltaTime.*

public static void createInputHandler(Player, float): Cria um gerenciador de entrada e saída para o jogador. Isso faz com que os comandos do teclado sejam inicializados. Recebe como parâmetros o ator do jogador e o *deltaTime*.

2.2.5. Classe WorldUtils

public WorldUtils(MapLoader): Constrói o mundo em que o jogo se passa, em conjunto com o mapa, suas texturas e seus *Body*, estruturas da Box2D que simula corpos com propriedades físicas, como dimensões, densidade, elasticidade, etc. Recebe como parâmetro o tipo de dado responsável por carregar o mapa.

public void createStaticBody(World, Rectangle): Cria os corpos estáticos do mundo, como o chão, paredes e plataformas. Recebe como parâmetros o mundo em que estarão contidos e um retângulo que será o corpo que colide.

public Body createPerson(World, float, float): Cria os corpos dinâmicos referentes aos atores principais (jogador e inimigos). Recebe como parâmetro o mundo em que estarão contidos e dois *floats*, indicando sua posição.

public Body createProjectile(float, float, float, Vector2, String, short): Cria os projéteis atirados pelos atores priincipais. Recebe como parâmetro três *floats* referentes à posição e raio, um *Vector2* referente à velocidade, uma *String* que indica os dados daquele corpo (para colisões) e um *short* que define o estado daquele projétil.

2.2.6. Classe WorldContactListener

public void beginContact(Contact): Vem de uma interface chamada *ContactListener* e recebe o aviso de um evento de colisão como parâmetro. Ao configurar essa função, algo acontecerá para o caso de uma colisão. Neste caso, ele matará o inimigo.

2.3. Programa Principal

O programa principal localiza-se no pacote *desktop.src.com.br.grupolaz.neocontra* na classe *DesktopLauncher*. Ele, primeiramente, chama a biblioteca *LWJGL3* e configura a aplicação, criando uma variável chamada *config*. São dadas as configurações básicas daquele jogo, como tamanho da janela, título da janela e *frameRate*. Após confiigurar a tela, ele chama a classe principal do nosso jogo ao criar a tela.

A classe principal é chamada *NeoContra* e se localiza no pacote *core.src.com.br.grupolaz.neocontra*. Nela, são inicializadas as variáveis indispensáveis do nosso jogo que serão utilizadas por outros componentes: a *OrthographicCamera*, o *Viewport* e o *SpriteBatch*. A *OrthographicCamera* e o *Viewport* são responsáveis por determinar e gerenciar o uso da câmera, a área de visão do nosso jogador. O *SpriteBatch* é responsável por enviar ao *OpenGL* o que precisa ser desenhado na tela, o nosso jogo.

Após a inicialização dessas variáveis, ocorre a configuração de uma nova tela ao chamar a classe responsável pela tela do jogo, enviando uma instância do próprio objeto para essa tela. A configuração de uma nova tela é possível apenas nessa classe por ser uma instância de *Game*, uma classe que combina as funcionalidades básicas do jogo com as funções de tela.

2.4. Organização do Código, Decisões de Implementação e Detalhes Técnicos

O código segue, inicialmente, a organização padrão gerado pela libGDX quando é executado o arquivo *"gdx-setup.jar"*. O programa principal encontra-se no pacote

desktop.src.com.br.grupolaz.neocontra no arquivo *DesktopLauncher.java.* Os outros arquivos, como o código para telas, *stages*, ETC e a classe que executa o jogo na janela são encontrados no pacote *core.src.com.br.grupolaz.neocontra* e em outros pacotes dentro deste.

Visando manter uma estrutura organizada e limpa, o programa foi estruturado em diferentes pastas. Essas pastas são chamadas de *actors*, *enums*, *screens*, *stages e util.*

Na pasta util, encontram-se os utilitários, ou seja, todos os códigos que não são exatamente elementos, mas são necessários para os elementos do jogo. Na pasta *stages*, encontram-se todos os componentes relacionados ao palco do jogo. Na pasta *screens*, estão localizadas todas as interfaces do jogo. Na pasta *enums*, estão todos os elementos de enumeração do jogo. Na pasta atores, estão todos os elementos ativos do jogo, ou seja, é onde estão os códigos para o jogador e os inimigos.

O código do jogo possui vários valores que se mostram constantes durante toda a construção do código tendo sido implementados em mais de um arquivo, sendo necessário em todos. Estes valores foram declarados em um arquivo *Constants.java* no pacote *core.src.com.br.grupolaz.neocontra.util.*

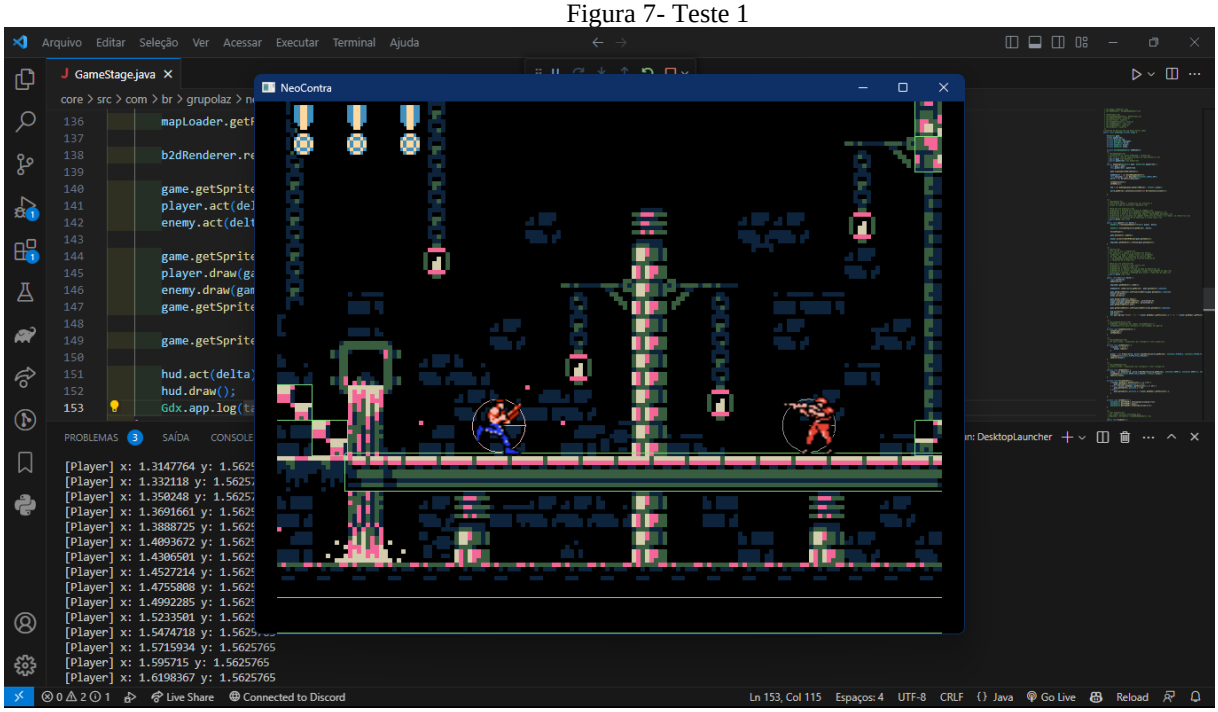
A versão do Java que foi utilizada para criar esse jogo foi a *"17.0.7" 2023-04-18 LTS.* A *IDE* utilizada foi o *Visual Studio Code*, sendo a versão dele a *"1.79".* Para executá-lo tem duas formas, pelo arquivo neocontra.jar, ou por meio da própria IDE, importando a pasta do jogo para dentro da IDE (é necessário possuir a extensão *Java Extension Pack*) e executando o jogo por meio do *Java Run*.

3. TESTES

Os testes foram conduzidos por meio da Máquina Virtual Java (JVM), a qual é um simulador de uma máquina real. Por isso, os desenvolvedores ficam independentes do sistema operacional.

O primeiro teste será o andar do personagem, se o personagem andar no console irá aparecer a posição do mesmo. Segue a baixo a figura do teste:

Figura 7- Teste 1

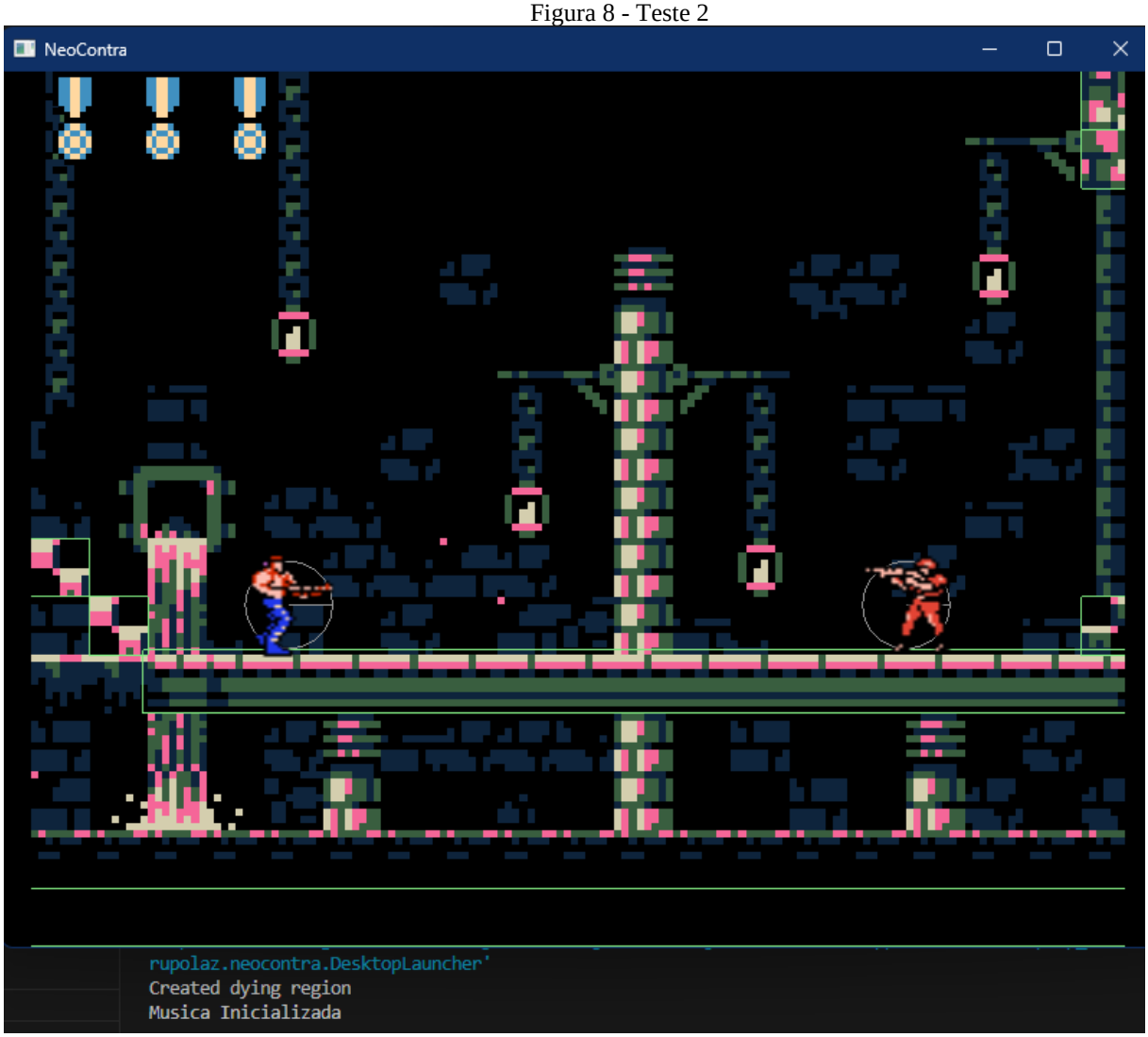


Fonte: autoral

É possível notar que o console exibe a posição do player constantemente e, ao pressionar o botão, a posição é alterada.

O segundo teste é para ver se está tocando música durante o jogo, se estiver, haverá a música de abertura, por estar em um looping, então assim que começar, tocará até o fim do jogo. Segue abaixo a figura do teste:

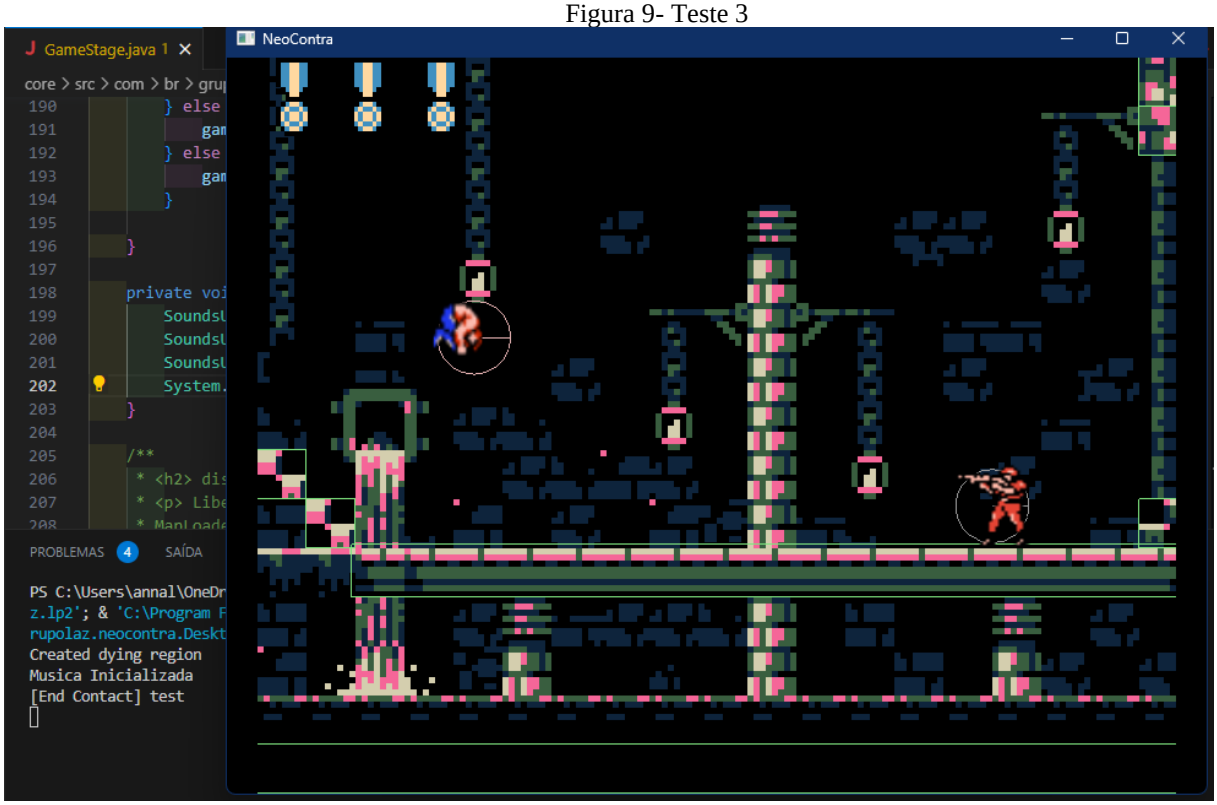
Figura 8 - Teste 2



Fonte: autoral

O terceiro teste é determinar se o jogador continua pulando, caso ele não esteja mais colidindo com o chão, ou seja, se ele terminou a colisão. Segue abaixo a figura do teste:

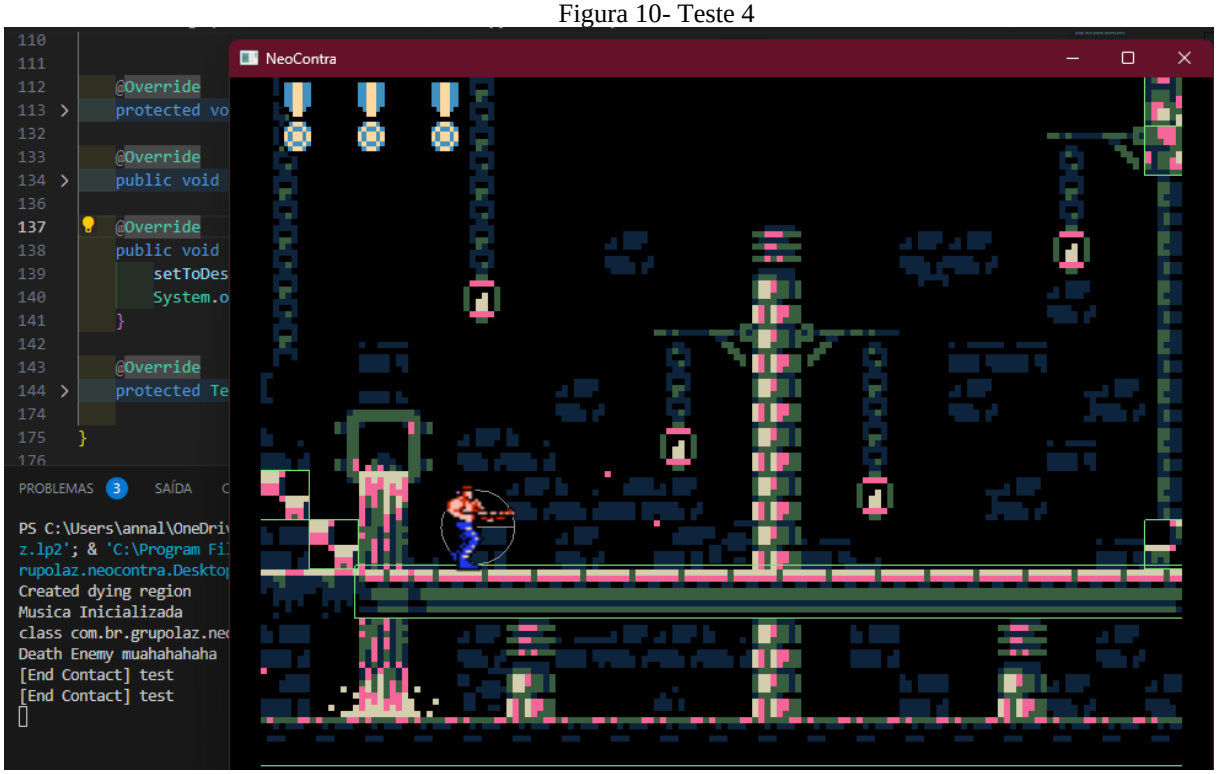
Figura 9- Teste 3



Fonte: autoral

O quarto teste, é ver se o inimigo está sendo morto, quando ele for morto, ocorrerá uma animação, em que o inimigo explode e some. Segue abaixo a figura deste teste.

Figura 10- Teste 4

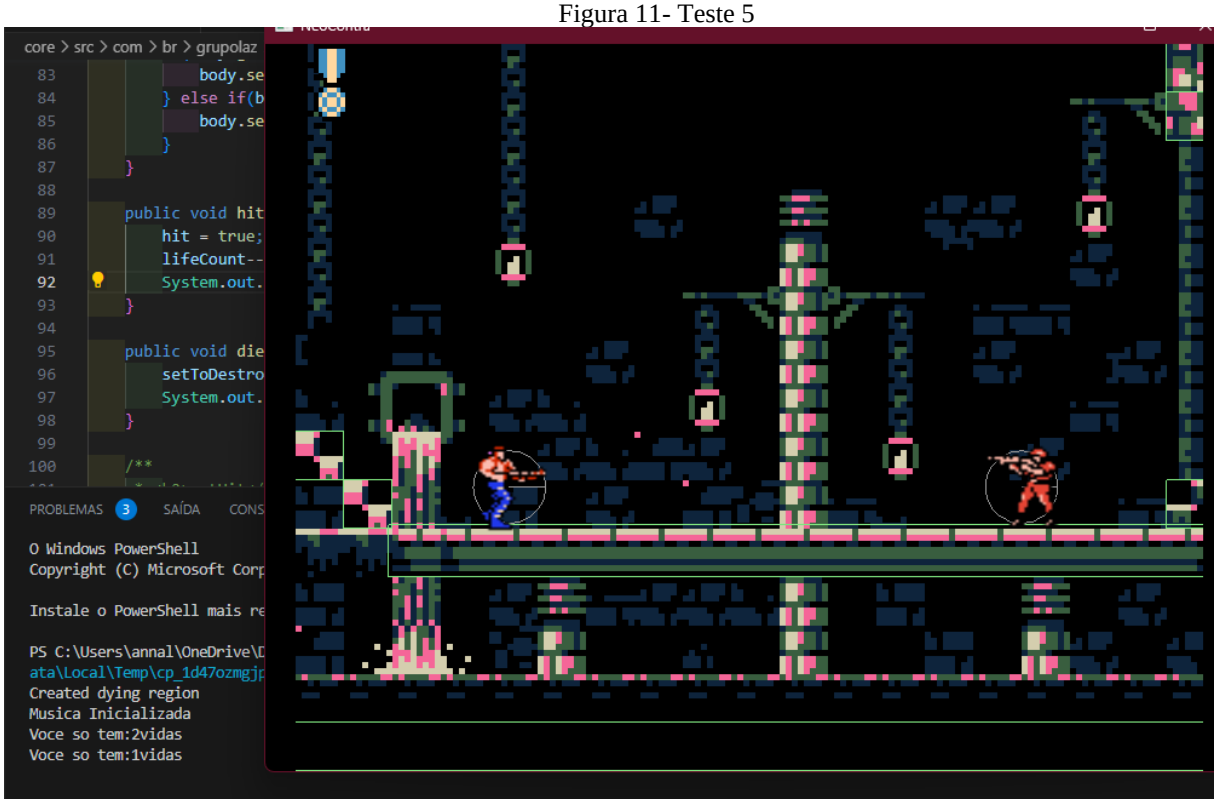


Fonte: autoral

Observa-se que quando o inimigo morre, aparece uma mensagem no console avisando que o inimigo morreu.

No quinto teste, é realizada a contagem das vidas, exibindo no console a quantidade restante de vidas cada vez que o jogador perde uma. Segue abaixo a figura do teste:

Figura 11- Teste 5

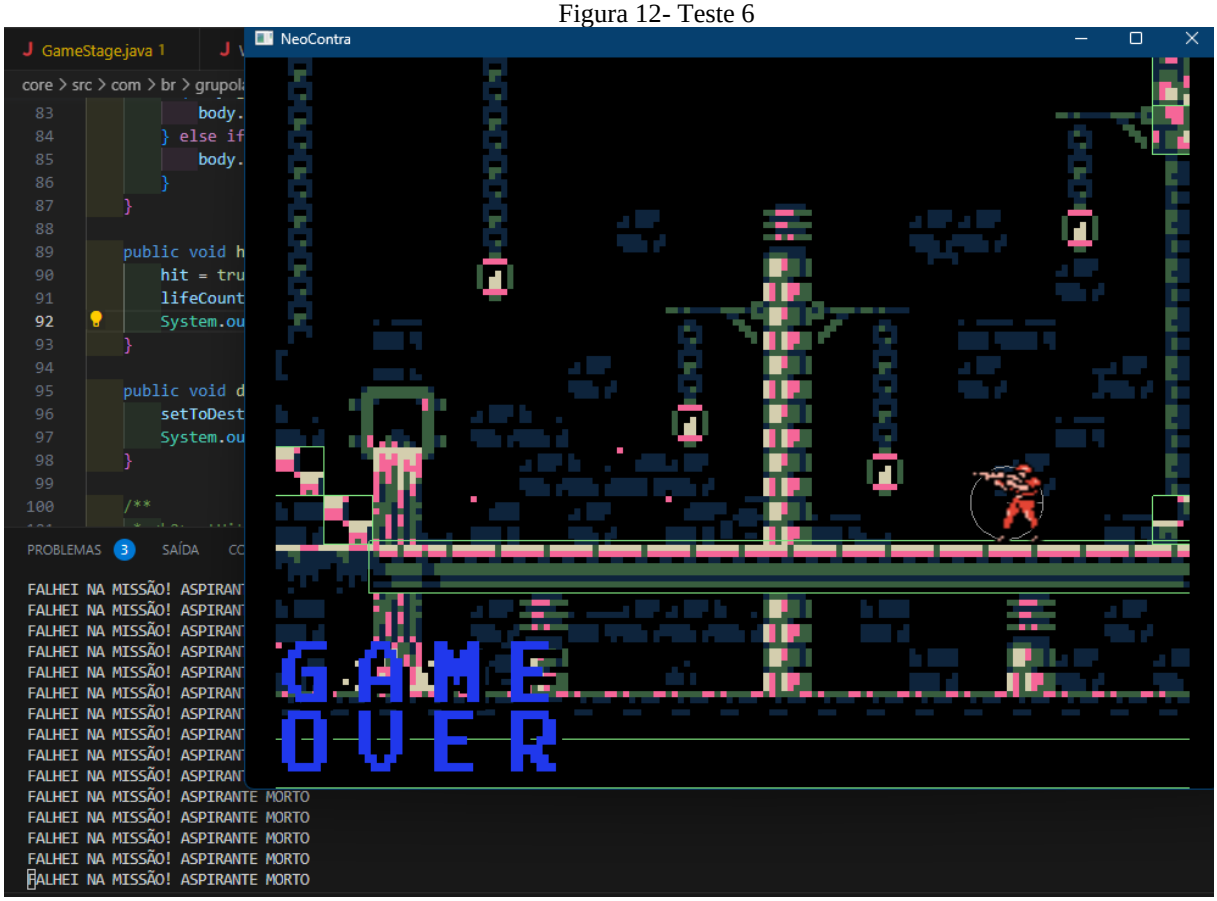


Fonte: autoral

Nota-se que a cada vida perdida, mostra quantas restam no console e no jogo, com as medalhas as representando.

O sexto teste, é o do *game-over*, quando o player perder todas as vidas, aparece a frase *game-over*, o player morre e não se consegue mexer mais na tela do jogo. Segue abaixo a figura que representa este teste.

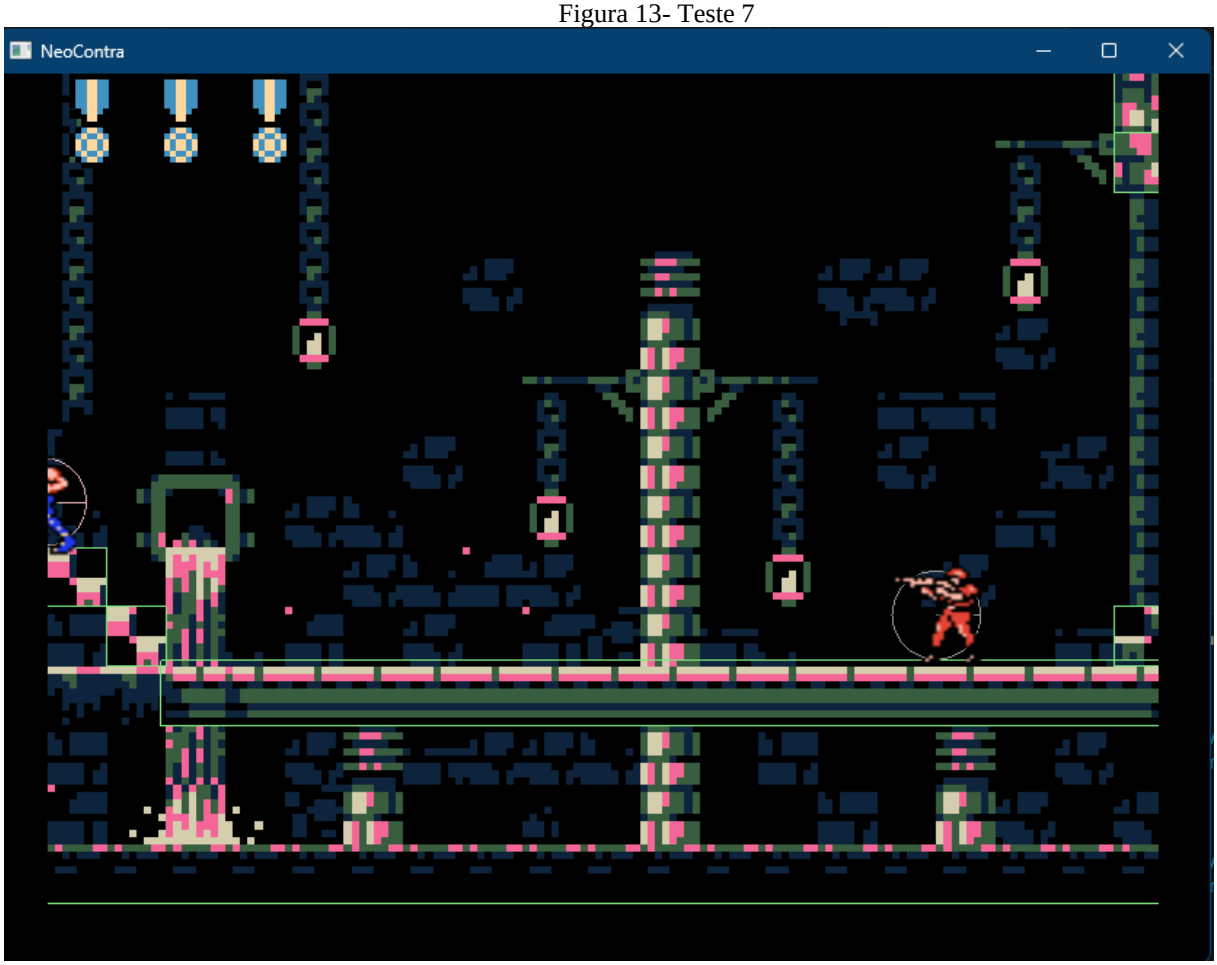
Figura 12- Teste 6



Fonte: autoral

Por fim, o último teste, é se o player está passando da tela do jogo. Segue abaixo a figura do teste.

Figura 13- Teste 7



Fonte: autoral

Observa-se que o player está andando, mas não ultrapassa a tela.

4. CONSIDERAÇÕES FINAIS

A implementação do código, em comparação ao último trabalho prático do grupo, foi um sucesso. Apesar disso, alguns objetivos não foram atendidos como o esperado. A recriação do jogo, por mais que tenha sido bem sucedida, em sua maior parte, não cumpriu com todos os requisitos iniciais do jogo.

A maioria dos objetivos especificados, no entanto, foram atendidos visto que o grupo dividiu o programa em classes, utilizou a técnica de polimorfismo, não utilizou de variáveis globais, utilizou de nomes significativos para variáveis e funções, utilizou das bibliotecas da libGDX de multimídia e para construção de animações, o jogo possui um conjunto de objetos que interage com o personagem, mesmo que em nível inferior ao do jogo original e a equipe se organizou compartilhando o desenvolvimento e a documentação via GitHub, incluindo o professor na equipe. O jogo não possui, todavia, um sistema de evolução e sequer possui muitas funções para o inimigo.

Muitas dificuldades foram encontradas pelos desenvolvedores durante o processo, porém, em comparação com o trabalho prático anterior, muitas dificuldades também foram superadas. Em primeiro lugar, a organização da equipe foi maior que a última vez e, por isso, foi possível alcançar a maioria dos objetivos. Contudo, ainda há muito a melhorar em relação à organização. Em segundo lugar, a implementação de certas funcionalidades do jogo original que, apesar de ser um jogo de mais de 30 anos atrás, ainda possui diversos desafios de programação que novos programadores podem considerar barreiras.

O tempo também foi uma dificuldade. Devido à carga horária excessiva da escola, o grupo encontrou dificuldades em administrar o tempo para o desenvolvimento do trabalho. Isso fez com que nem todas as mecânicas do jogo original fossem implementadas até a data de entrega.

No fim, conseguimos cumprir também com os objetivos gerais, isto é, o uso da programação orientada a objetos, o uso da biblioteca libGDX, o uso de coleções de objetos e ampliamos bastante o nosso conhecimento da linguagem Java por meio de técnicas de programação aprendidas durante o processo.

REFERÊNCIAS

@GAMEDEVWITHMARK. **How to make the enemy face & follow the player in Unity**. Disponível em: <https://www.youtube.com/shorts/eZBm-VNAYDA>. Acesso em: 3 jul. 2023.

ANDERSON, C. **LibGDX - Gdx-AI - Intro to Steering Behaviors: Arrival (Pt. 1)**. Disponível em: <https://www.youtube.com/watch?v=pnKcuJQT31A>. Acesso em: 3 jul. 2023.

AURELI, B. **LibGDX - Super Mario Bros - YouTube**. Disponível em: <https://www.youtube.com/playlist?list=PLZm85UZQLd2SXQzsF-a0-pPF6IWDDdrXt>. Acesso em: 3 jul. 2023.

Diagrama de Classes. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/diagramas/classes/classes2.htm>. Acesso em: 3 jul. 2023.

FIEDLER, G. **Fix Your Timestep!** Disponível em: <https://gafferongames.com/post/fix_your_timestep/>. Acesso em: 3 jul. 2023.

FREE Zombie Shooter (Soundtracks/FX pack) by Tiziano Bellu. Disponível em: <https://tizianobellu.itch.io/zombie?download>. Acesso em: 3 jul. 2023.

FREECODECAMP.ORG. **UML Diagrams Full Course (Unified Modeling Language)**. **YouTube**, 21 abr. 2021. Disponível em: <https://www.youtube.com/watch?v=WnMQ8HlmeXc>

GAMEFROMSCRATCH. **Introduction to Scene2D in LibGDX**. Disponível em: <https://www.youtube.com/watch?v=DPIeERAm2ao>. Acesso em: 3 jul. 2023.

HOLLOWBIT. **LibGDX 2D Tutorials - YouTube**. Disponível em: <https://www.youtube.com/playlist?list=PLrnO5Pu2zAHKAIjRtTLAXtZKMSA6JWnmf>. Acesso em: 3 jul. 2023.

JANO, T. **Create a 2D Platformer with libGDX - Part 1**. Disponível em: <https://obviam.net/posts/2012/02.libgdx-create-prototype-tutorial-part1/>. Acesso em: 3 jul. 2023.

LIBGDX. **libgdx-demo-cuboc**. Disponível em: <https://github.com/libgdx/libgdx-demo-cuboc>. Acesso em: 3 jul. 2023.

LIBGDX. **Spritebatch, Textureregions, and Sprites**. Disponível em: <https://libgdx.com/wiki/graphics/2d/spritebatch-textureregions-and-sprites>. Acesso em: 3 jul. 2023.

LIBGDX. **Graphics**. Disponível em: <https://libgdx.com/wiki/graphics/graphics>. Acesso em: 3 jul. 2023.

LIBGDX. **Box2d**. Disponível em: <https://libgdx.com/wiki/extensions/physics/box2d>. Acesso em: 3 jul. 2023.

MACAROTTI, J. C. **UML - Diagrama de classes**. Disponível em: <https://www.macoratti.net/net_uml1.htm>. Acesso em: 3 jul. 2023.

MILINDA. **O Tutorial de Diagramas de Classe Ultimate para Ajudar a Modelar Facilmente os Seus Sistemas**. Disponível em: <https://creately.com/blog/pt/diagrama/tutorial-diagrama-de-classes/>.

MORA, W. **Martian Run!** Disponível em: <https://github.com/wmora/martianrun>. Acesso em: 3 jul. 2023.

Not Jam Music Pack by Not Jam. Disponível em: <https://not-jam.itch.io/not-jam-music-pack?download>. Acesso em: 3 jul. 2023.

SMALL PIXEL GAMES. **2D Platformer - YouTube**. Disponível em: <https://www.youtube.com/playlist?list=PLVNiGun9focYT2OVFUzL30wUtOToo6frD>. Acesso em: 3 jul. 2023.
What is a Delta time in LIBGDX. Disponível em: <https://stackoverflow.com/questions/34479099/what-is-a-delta-time-in-libgdx/>. Acesso em: 3 jul. 2023.
What is Package Diagram? Disponível em: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/>. Acesso em: 3 jul. 2023.

ANEXOS

- NeoContra.java
- WorldUtils.java
- WorldContactListener.java
- TextureUtils.java
- SoundsUtils.java
- MapLoader.java
- GameUtils.java
- Constants.java
- GameStage.java
- HudStage.java
- GameScreen.java
- ActorStates.java
- GameState.java
- Projectile.java
- Player.java
- GameActor.java
- Enemy.java
- Bullet.java