INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
CAMPUS DE CAJAZEIRAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Murilo de Oliveira Ferreira

Um motor de Jogos 2D em Typescript

Cajazeiras PB
2024

Murilo de Oliveira Ferreira

Um motor de Jogos 2D em Typescript

Trabalho de Conclusão de Curso apresentado junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Campus Cajazeiras, como requisito à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador
Prof. Dr. Fabio Gomes de Andrade

Cajazeiras PB
2024

IFPB / Campus Cajazeiras
Coordenação de Biblioteca
Biblioteca Prof. Ribamar da Silva
Catalogação na fonte: Cícero Luciano Félix CRB-15/750

F383m Ferreira, Murilo de Oliveira.
Um motor de Jogos 2D em *Typescript* / Murilo de Oliveira Ferreira. – 2024.

38f. : il.

Trabalho de Conclusão de Curso (Tecnólogo em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Cajazeiras, 2024.

Orientador(a): Prof. Dr. Fabio Gomes de Andrade.

1. Desenvolvimento de sistemas. 2. Programação. 3. Design de jogos. 4. Gamificação. I. Instituto Federal de Educação, Ciência e Tecnologia da Paraíba. II. Título.

IFPB/CZ CDU: 004.4(043.2)

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

MURILO DE OLIVEIRA FERREIRA

UM MOTOR DE JOGOS 2D EM TYPESCRIPT

Trabalho de Conclusão de Curso apresentado junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Campus Cajazeiras, como requisito à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador

Prof. Dr. Fabio Gomes de Andrade

Aprovada em: **16 de Outubro de 2024.**

Prof. Dr. Fabio Gomes de Andrade - Orientador

Prof. Me. Michel da Silva - Avaliador

IFPB - Campus Cajazeiras

Prof. Dr. Hudson Geovane de Medeiros

IFPB - Campus Cajazeiras

Documento assinado eletronicamente por:

- **Fabio Gomes de Andrade**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 17/10/2024 20:00:24.
- **Hudson Geovane de Medeiros**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 17/10/2024 22:37:12.
- **Michel da Silva**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 18/10/2024 18:43:33.

Este documento foi emitido pelo SUAP em 16/10/2024. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código 620964
Verificador: f17e7bdef9
Código de Autenticação:

AGRADECIMENTOS

Aos meus pais, que sempre me incentivaram e me apoiaram em todos os momentos.

Aos amigos, por me ajudarem quando precisei e por estarem sempre ao meu lado nas situações mais difíceis.

Aos professores, por ensinarem tudo que eu aprendi durante toda a duração do curso.

Ao meu orientador de trabalho de conclusão de curso, Prof. Dr. Fabio Gomes de Andrade, por fazer correções e sugestões durante o desenvolvimento deste Trabalho de Conclusão de Curso (TCC) e por sempre estar disponível quando surgiam dúvidas relacionadas a sua escrita.

RESUMO

Os jogos digitais são um tópico bastante discutido e seu impacto no mundo é fácil de ser percebido. Os jogos criaram o termo “gamificação”, que é usado para se referir ao uso de elementos do design de jogos fora do contexto de jogos. A aplicação desses elementos tem como objetivo aumentar o engajamento e motivação das pessoas em relação à alguma atividade. Entretanto, o uso da gamificação não é a única forma de aumentar o engajamento que os jogos trouxeram. Os jogos em si também são usados para o mesmo propósito, sendo usados como ferramentas para ensinar algo. Além disso, também é comum o desenvolvimento de jogos como forma de motivar estudantes a aprenderem programação, pois a criação de um jogo pode incentivar essas pessoas a procurar formas de fazerem o que o seu jogo precisa. Isso se deve ao fato de que boa parte dos programadores já tem interesse pelo seu desenvolvimento. Existem várias formas de se aprender programação desenvolvendo jogos. No entanto, algumas ferramentas têm limitações em relação ao que pode ser desenvolvido e outras são complexas, o que dificulta a implementação do que o usuário quer. Para a solução desse problema, este trabalho propõe a criação de um motor de jogos que é simples de usar, fornece muitos dos recursos necessários para o desenvolvimento de jogos e permite ao programador criar as suas próprias funcionalidades para adicioná-las a ferramenta, dessa forma, facilitando a criação de jogos e também provendo um ambiente em que os usuários possam aprender programação criando um jogo.

Palavras-chave: Design de jogos. Gamificação. Jogo. Motor de jogos.

ABSTRACT

Digital games are a widely discussed topic, and their impact on the world is easily perceived. Games have created the term "gamification," which refers to the use of game design elements outside the context of games. The application of these elements aims to increase people's engagement and motivation regarding a particular activity. However, the use of gamification is not the only way games have contributed to enhancing engagement. The games themselves are also used for the same purpose, serving as tools to teach something. Moreover, developing games is a common method to motivate students to learn programming, as creating a game can encourage them to seek ways to implement what their game requires. This is because many programmers already have an interest in game development. There are various ways to learn programming through game development. However, some tools have limitations regarding what can be developed, while others are complex, making it difficult for users to implement their ideas. To address this issue, this work proposes the creation of a game engine that is easy to use, provides many of the necessary resources for game development, and allows programmers to create their own functionalities to add to the tool. This approach facilitates game creation and provides an environment in which users can learn programming by creating a game.

Palavras-chave: Game. Game design. Game engine. Gamification.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

2D	Duas Dimensões
CSS	Cascading Style Sheets
FPS	Frames Per Second
HTML	HyperText Markup Language
IDE	Integrated Development Environment
RF	Requisito Funcional
TCC	Trabalho de Conclusão de Curso

SUMÁRIO

1. INTRODUÇÃO

A indústria de jogos vem crescendo muito nos últimos anos, e, segundo Amélio (2018), já fatura mais que as indústrias de música e cinema juntas. Ele também afirmou que, em 2017, o Brasil era o quarto maior consumidor de jogos digitais no mundo, atrás apenas da China, Estados Unidos e Japão, e que o crescimento desse mercado aconteceu muito rápido. Com jogos sendo algo tão atrativo para as pessoas, alguns dos seus aspectos estão sendo tomados como base para a criação da ”gamificação”.

De acordo com Deterding et al. (2011), o termo ”gamificação” é usado para se referir ao uso de elementos característicos do design de jogos fora do contexto de jogos. Os autores também definem alguns elementos de jogos, como níveis, *leaderboards* ou classificações, objetivos claros, variedade de estilos de jogo, desafios, curiosidade, fantasia, entre outros. Outros elementos dos jogos que podem ser usados, como apontam Silva, Tedesco e Melo (2014), são desafios e missões, *achievements* ou *badges,* que são dados ao jogador quando ele obtém uma conquista ou realiza uma tarefa, *progress bar,* que é uma barra de progresso para mostrar a evolução do jogador durante o jogo e, por fim, *rankings* ou *leaderboards,* que mostram a classificação de todos os jogadores com o objetivo de motivá-los com a competição.

Silva, Oliveira e Martins (2017) destacam que a gamificação tem o propósito de aumentar a motivação e o engajamento do usuário. Em seu trabalho, os autores a utilizaram para motivar alunos a aprenderem a programar na linguagem de programação python integrada ao jogo *Minecraft*. Nesse estudo foram usados alguns elementos dos jogos, como pontuações, missões, cooperação e competição, além de uma história que envolvia os professores e os alunos, com o objetivo de incentivar a participação. Ao final do das atividades, foi feito um questionário no qual os participantes demonstraram ter gostado da gamificação, e além disso, um dos participantes do questionário disse que um dia quer criar o próprio jogo. Como os alunos que participaram dessa pesquisa eram iniciantes no mundo da programação, pode-se pensar que ferramentas mais simplificadas ajudariam no processo de criação de um jogo. Ao concluírem o seu estudo, os autores afirmaram que a gamificação proporcionou um aumento na motivação do estudo da programação.

Os jogos têm se mostrado uma ótima maneira de manter os estudantes

motivados. SCAICO (2018) aponta vários casos em que jogos impactam positivamente na aprendizagem. Um exemplo é o trabalho de Cliburn e Miller (2008), no qual estudantes demonstraram uma preferência por executar tarefas relacionadas a jogos e que mais empenho era mostrado quando estavam codificando jogos quando comparado a tarefas tradicionais. Outro trabalho citado é o de Kurkovsky (2009), no qual o ensino orientado ao desenvolvimento de jogos mostrou ter um impacto positivo nos estudantes.

Assim, percebe-se que conceitos de jogos podem ser utilizados para motivar e interessar as pessoas. Entretanto, também existem jogos que são desenvolvidos com o objetivo de ensinar, que são chamados de ”jogos sérios”. Rocha, Bittencourt e Isotani (2015) descrevem esse tipo de jogo como um jogo que tem o objetivo de ensinar ou treinar, além do entretenimento. Um exemplo desse tipo de jogo é o *Rocksmith* (2014), que Oprach (2021) aborda em seu trabalho. Para jogar, o jogador conecta a sua guitarra e pode tocar músicas para aprender o instrumento com a mínima ajuda de um professor. O jogo aplica as práticas de gamificação citadas anteriormente, como níveis e fases. O jogo ainda tem diferentes níveis de dificuldade para que o jogador gradativamente melhore sua habilidade tocando arranjos mais difíceis até tocar o mesmo da música original.

1.1 Motivação

Aprender programação pode ser difícil para iniciantes. Segundo Júnior, Fechine e Costa (2009), isso ocorre porque eles devem desenvolver a habilidade de solucionar problemas algoritmicamente, e que a ferramenta e metodologia correta pode facilitar o aprendizado. Já Bezerra et al. (2022) evidenciam que os cursos da área de Tecnologia da Informação apresentam altos índices de reprovação e evasão, principalmente nas disciplinas de Introdução a Programação e Algoritmos, e isso acontece porque o ensino básico não aborda as habilidades de lógica e interpretação. Além disso, os autores dizem que, com toda essa dificuldade, iniciantes perdem a motivação para estudar. Uma forma encontrada para motivar os alunos, segundo os autores, seria o desenvolvimento de jogos digitais, uma vez que, ao aplicarem conceitos de gamificação, os alunos muitas vezes demonstram mais interesse, o que pode elevar o índice de aprovação.

Hoje, existem muitas ferramentas para a criação de jogos, mas, em muitos

casos, o usuário tem que configurar muitas partes da ferramenta para que ela funcione da forma que ele deseja. Em outros casos, ele deve criar funções para usar na ferramenta, funções essas que funcionam da mesma forma para todos os jogos e poderiam já ser parte da ferramenta base. Isso deixa a ferramenta mais complexa, tornando-a mais difícil de usar para iniciantes que querem fazer um jogo como forma de se manterem motivados a aprender programação.

Tendo em vista que ferramentas podem ajudar com a aprendizagem e que tarefas ligadas a jogos são algo que mantém as pessoas interessadas e motivadas mais do que a resolução de questões normais, e também que ferramentas para a criação de jogos podem ser complexas para iniciantes, este Trabalho de Conclusão de Curso (TCC) propõe o desenvolvimento de uma ferramenta para a criação de jogos que possa ser usada por iniciantes na programação para aplicarem o que aprendem em sala de aula e assim praticarem mais com algo que os interessa.

1.2 Objetivos

Esta seção descreve os objetivos alcançados com o desenvolvimento deste TCC.

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é o desenvolvimento de um motor de jogos 2D que providencie uma boa quantidade dos recursos necessários para facilitar o desenvolvimento de jogos.

1.2.2 Objetivos Específicos

O trabalho tem ainda os seguintes objetivos específicos:

- tornar a aprendizagem de programação mais simples;
- apresentar uma solução que reduza a complexidade inicial para o desenvolvimento de jogos;
- simplificar o desenvolvimento de novas funcionalidades inerentes ao desenvolvimento de jogos.

1.3 Trabalhos relacionados

Existem várias formas de se aprender programação por meio de jogos, e alguns trabalhos já foram desenvolvidos com este objetivo. Um desses trabalhos é a plataforma *CodinGame*[1], desenvolvida pela *CoderPad*. Ince (2021) descreve essa plataforma como um ambiente web educacional que dá suporte à programação em várias linguagens e foca na gamificação do desenvolvimento de código. No site existem missões e conquistas que o usuário pode completar para evoluir de nível além de competições com outros usuários. Entretanto, como a plataforma apenas usa gamificação para incentivar a aprendizagem de programação, só é possível estudar programação na plataforma, não sendo possível o usuário criar usá-la para criar o seu próprio jogo, algo que é do interesse de muitos programadores.

Outra solução para ensinar programação com gamificação é a linguagem *Scratch*[2], criada pelo *Media Lab* em 2007. Neto (2013) a descreve como uma linguagem gráfica desenvolvida para introduzir a programação, conceitos matemáticos, pensamento criativo, raciocínio sistemático e trabalho colaborativo. Ela permite o uso de mídias como imagens, gráficos, sons e músicas, possibilitando a criação de histórias, animações e jogos. Neto ainda diz que *Scratch* é uma excelente forma de ensinar lógica de programação, acostumando o usuário com o desenvolvimento de um algoritmo sem usar diretamente uma linguagem de programação. Porém, como programar em *Scratch* envolve apenas o uso de elementos gráficos, o ambiente de programação se diferencia bastante das outras linguagens, onde o código é feito com texto. Além disso, ela não oferece recursos avançados de programação para que programadores aprimorem ainda mais as suas habilidades e possam aplicar esses recursos em seus jogos.

Outro trabalho relacionado é o *Pygame*[3], criado no ano 2000, que é um conjunto de módulos na linguagem de programação *python* feitos para a criação de jogos. No *Pygame,* o programador tem acesso aos dispositivos de entrada do computador, como o mouse e o teclado, e pode reproduzir mídias como áudio ou imagens na tela do computador. Além disso, o *Pygame* tem suporte para muitas plataformas, como a maior parte das distribuições dos sistemas operacionais *Linux*, *Windows* e *Mac*. Também é possível usar o *Pygame* integrado com outras

[1] https://www.codingame.com/
[2] https://scratch.mit.edu/
[3] https://www.pygame.org/wiki/about

bibliotecas, caso o usuário queira usar apenas a função de áudio, por exemplo. Entretanto, no *Pygame*, se o programador quiser usar animações ou trocar o estado atual do jogo como a fase ou tela exibida, ele deve implementar essas funcionalidades do zero, o que pode ser difícil para um iniciante.

1.4 Metodologia

Para garantir o bom desenvolvimento da ferramenta, foi fundamental definir os processos necessários para atingir o resultado desejado. Foram então definidas as seguintes atividades para assegurar que o desenvolvimento deste trabalho acontecesse da melhor forma:

- **Análise do estado da arte (A_1):** ao longo desta atividade foram analisados trabalhos com temas similares e ferramentas com funcionalidades semelhantes ao que está sendo proposto neste trabalho. Os trabalhos analisados falam sobre os benefícios que os jogos trazem para a aprendizagem e ferramentas que são usadas para a sua criação. Essa análise foi de grande importância para manter o objetivo do trabalho bem definido;
- **Elaboração do documento (A_2)**: esta atividade se refere à escrita deste documento de trabalho de conclusão de curso;
- **Levantamento de requisitos (A_3):** nesta atividade foram levantados os requisitos que a ferramenta desenvolvida deveria cumprir para que ela pudesse atingir os objetivos gerais e específicos definidos;
- **Definir a arquitetura do projeto (A_4):** nesta atividade foi definida a arquitetura do projeto, para identificar quais módulos deveriam ser desenvolvidos e como eles deveriam interagir entre si;
- **Implementar o módulo para programação do jogo (A_5):** nesta atividade foi desenvolvido o módulo onde o usuário cria todos os arquivos que usará em seu jogo. Esse módulo também é responsável por prover ao usuário funções e recursos prontos para serem usados em seu jogo para que o mesmo só precise criar a lógica do próprio jogo;
- **Implementar o módulo para execução do jogo (A_6):** nesta atividade foi desenvolvido o módulo no qual o jogo pode ser executado. Esse

módulo é responsável por mostrar a parte visual do jogo com as imagens que o usuário colocou. Ele também é responsável por receber os dados de entrada do teclado e do mouse durante a execução do jogo. Além disso, este módulo também reproduz os sons adicionados pelo usuário.

1.5 Organização do documento

O restante deste documento está organizado da seguinte forma: o capítulo 2 desenvolve a fundamentação teórica, onde são descritos os conceitos e tecnologias importantes para o desenvolvimento deste trabalho. No capítulo 3 é descrita a solução proposta, apresentando a ferramenta que foi desenvolvida. Por fim, o capítulo 4 tem a conclusão deste trabalho e aponta possíveis trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos e tecnologias utilizados para a construção da ferramenta proposta por este trabalho. Nele são abordados os seguintes temas: motor de jogos, ciclo de jogos, motor de física e objetos de jogos.

2.1 Motor de jogo

Um Motor de jogo (do inglês *Game engine),* segundo Andrade (2015), é um software que permite a criação de jogos. O autor também define vários recursos que esse tipo de software pode oferecer e alguns deles são:

- **uma cena gráfica:** responsável por gerenciar todos os elementos gráficos e seus relacionamentos;
- **manipulação de entrada:** responsável por gerenciar os botões pressionados pelo jogador, podendo ser usados o teclado, o mouse, dispositivos de toque, entre outros;
- **som:** o motor deve poder reproduzir sons;
- **animação:** o motor deve poder fazer animações.

Ainda existem mais recursos como ciclo de jogo (do inglês, *Game loop)*, e motor de física, que serão discutidos nas próximas seções. Alguns motores também podem prover recursos como inteligência artificial, *networking* para conexões online, suporte multi-plataforma e outros. É possível que um motor ofereça todos esses recursos de forma nativa, mas eles também podem usar bibliotecas de terceiros para isso. De acordo com Andrade, trazer todas essas características em um ambiente de desenvolvimento como uma IDE (*Integrated Development Environment)* é o objetivo de um desenvolvedor de um motor de jogo.

2.2 Ciclo de jogo

Segundo Zamith, Valente e Clua (2016), os jogos criam ambientes virtuais que fazem eventos dentro do jogo acontecerem de acordo com a nossa percepção de tempo. Os autores chamam esse ambiente de “ilusão de jogo” e ela é criada pela apresentação de mídias como imagens, animações e áudio em ciclos contínuos com alta frequência. Este processo executa várias outras tarefas para poder computar o

estado atual do jogo. O nome dado pelos autores ao sistema que computa o estado atual do jogo é ciclo de jogo, e eles explicam diferentes conceitos importantes para o seu entendimento. Esses conceitos são a simulação discreta, tarefas de jogo, estrutura de fluxo de tempo e interatividade.

2.2.1 Simulação discreta

O jogo cria um ambiente virtual a cada vez em que o estado do jogo é computado. Essa tarefa é feita em sequência ao longo do tempo. Existem três conceitos importantes sobre tempo para o entendimento do ciclo de jogo: tempo de relógio, tempo de simulação e tempo real. Tempo de relógio é a passagem regular de tempo. Já o tempo de simulação é o tempo que cada simulação representa na linha do tempo. A linha do tempo pode estar dividida em partes menores chamadas de “passo de tempo”. Quando a simulação avança na linha do tempo na mesma velocidade que o tempo de relógio, dizemos que ela está em tempo real. Para jogos que usam passos de tempo, a simulação é computada em cada passo. A quantidade de passos que existe em um segundo de simulação é chamado de taxa de amostragem de simulação. A quantidade de estados que o jogo exibe na tela por segundo é chamado de taxa de amostragem de apresentação. Jogos podem ter valores iguais ou diferentes para os dois. O termo *Frames Per Second (FPS)* é usado para representar estas unidades.

2.2.2 Tarefas de jogo

A cada ciclo de jogo, são executadas várias tarefas para calcular o próximo estado do jogo. Essas tarefas são divididas em três categorias executadas em sequência, são elas: aquisição de dados de entrada, computação do estado do jogo (simulação) e apresentação. A aquisição de dados de entrada envolve o fornecimento de dados ao motor de jogo para que a simulação seja computada. Essas informações podem ser muitas coisas, como teclas de um teclado, mouse ou controles, comandos de voz, sensores de movimento, toques na tela e outros. Na etapa de computação do estado do jogo, os dados de entrada, juntamente com o estado atual do jogo, são computados conforme a lógica do jogo para que o próximo estado seja simulado. Ainda nessa etapa são feitas simulações da física do jogo e comportamentos de inteligência artificial, caso o motor de jogo ofereça este recurso.

Por fim, na etapa de apresentação, o estado do jogo é exibido na tela por meio de imagens, vídeos, animações, sons e outros. Todas essas tarefas devem ser executadas na duração de um passo de tempo para que o jogo funcione em tempo real.

2.2.3 Estrutura de fluxo de tempo

Anteriormente, foi definido que uma simulação funciona em tempo real quando ela avança na linha do tempo na mesma velocidade que o tempo de relógio. É importante que um jogo funcione em tempo real para que a experiência do jogador não seja comprometida. Caso isso não ocorra, a “ilusão de jogo” será quebrada. Isso pode ocorrer quando as tarefas do jogo definidas anteriormente não conseguem ser completadas dentro de um passo de tempo, ou quando elas são finalizadas muito rápido e o motor de jogo não leva em conta este tempo de sobra, fazendo com que a etapa de apresentação seja muito rápida. Na literatura de desenvolvimento de jogos, é comumente dito que a performance do jogo deve estar entre no mínimo entre 30 e 60 FPS, ou seja, cada segundo de simulação é dividido entre 30 e 60 passos de tempo. Isso quer dizer que as tarefas de jogo devem ser executadas de 30 a 60 vezes por segundo para que a “ilusão de jogo” seja mantida.

2.2.4 Interatividade

A interatividade, segundo Steuer et al. (1995), refere-se a quando os usuários podem influenciar um ambiente em tempo real, alterando a sua forma e conteúdo. Como já foi dito antes, a aquisição de dados de entrada é uma parte importante de um ciclo de jogo. É assim que um jogador consegue interagir com o mundo virtual e influenciar no estado do jogo. Essa interatividade pode acontecer de várias formas, como o uso de um teclado, mouse, tela de toque e outros.

2.3 Motor de física

Conceitos de física já são usados em jogos digitais há mais de 30 anos, de acordo com Millington (2007). Cada comportamento era programado para casos específicos. Por exemplo, se um jogo precisava calcular a trajetória que determinadas flechas deveriam seguir, este comportamento era programado diretamente e funcionava apenas para essas flechas. Para jogos simples isso não

era um problema, mas, com o aumento na complexidade dos jogos e a quantidade de jogos que tem procedimentos similares, surgiu a necessidade de criação de tecnologias que pudessem ser reutilizadas. Millington afirma que para a reutilização de tecnologias desse tipo é preciso que ela seja bem geral. Por exemplo, um jogo poderia lidar com flechas de uma forma, mas se fosse lidar com balas, o funcionamento seria diferente. Para isso, foram criados “motores físicos” (ou *physical engines*), que sabem como lidar com o comportamento físico de objetos em geral, mas o comportamento específico é programado para cada jogo.

O autor aponta que, se o motor de física apenas souber calcular projéteis no geral, e precisarmos calcular a trajetória de uma flecha, teremos que informar todos os atributos dessa flecha para que a simulação ocorra de forma correta. Esse mesmo comportamento é necessário para a simulação de qualquer objeto de jogo, como caixas, paredes, balas, entre outros.

2.4 Objeto de jogo

Objetos de jogo são definidos por Porter (2012) como todos os atores no mundo do jogo. Esses objetos podem ser estáticos ou dinâmicos. Objetos estáticos não reagem quando recebem estímulos, ou seja, não interagem com o mundo do jogo. Eles são usados como objetos de cenário, como árvores, pedras e outros. Objetos dinâmicos, por outro lado, podem interagir com outros objetos do jogo, tanto estáticos como dinâmicos. Outros autores também definem objetos de jogo. Para Chady (2009), objetos de jogo são qualquer coisa que tenha uma representação no jogo, como personagens, veículos, mísseis, câmeras, luzes e mais. Já Bilas (2002) diz que objetos de jogo são pedaços de conteúdo interativo lógico, e executam tarefas como renderização, fala, encontrar caminhos e animações. Esses objetos podem ser árvores, arbustos, monstros, alavancas, heróis e itens no inventário do jogador. Ainda em seu trabalho, são definidos objetos que são “pura lógica”, como gatilhos, sequências de câmera e outros.

3. A SOLUÇÃO PROPOSTA

Este capítulo descreve o desenvolvimento do trabalho proposto por este TCC. Primeiramente, são descritos os possíveis usuários dessa ferramenta. Após isso, são descritos os requisitos funcionais da ferramenta. Em seguida, é apresentada a sua arquitetura, juntamente com a descrição dos seus módulos e das interações entre os mesmos. E, por fim, é apresentada a implementação dos módulos desenvolvidos.

3.1 Possíveis usuários

Os possíveis usuários desse sistema são pessoas que estão aprendendo programação, que tem interesse em jogos, e querem aprender criando um jogo para se manterem motivados, pessoas que fazem jogos como forma de se divertirem e também professores que querem usar jogos como uma forma de incentivar a aprendizagem de programação.

3.2 Requisitos funcionais

Os requisitos funcionais para o desenvolvimento deste trabalho foram levantados levando-se em consideração quais recursos os programadores precisam para criar jogos e também com base em outras ferramentas usadas para o desenvolvimento de jogos. Esses requisitos eram:

- **Criar objetos (RF1):** o motor deveria permitir ao usuário criar objetos para serem usados no jogo. Esses objetos poderiam ser personagens controlados pelo jogador, aliados do jogador ou inimigos presentes na fase, itens que seriam usados por outros objetos como uma arma para um personagem ou um veículo, e ou ainda objetos para a construção do cenário, como árvores, portas, construções etc.;
- **Programar o comportamento de objetos (RF2):** o motor deveria permitir ao usuário programar o comportamento de objetos, como fazer um personagem controlado pelo jogador, aliados que o ajudam, inimigos que o atacam, itens como armas ou acessórios para personagens, objetos da fase, como paredes ou portas que impedem o jogador de prosseguir, e objetos que interagem com outros objetos;
- **Criar fases (RF3):** o motor deveria permitir ao usuário criar fases para o jogador jogar. As fases seriam compostas por personagens, itens e

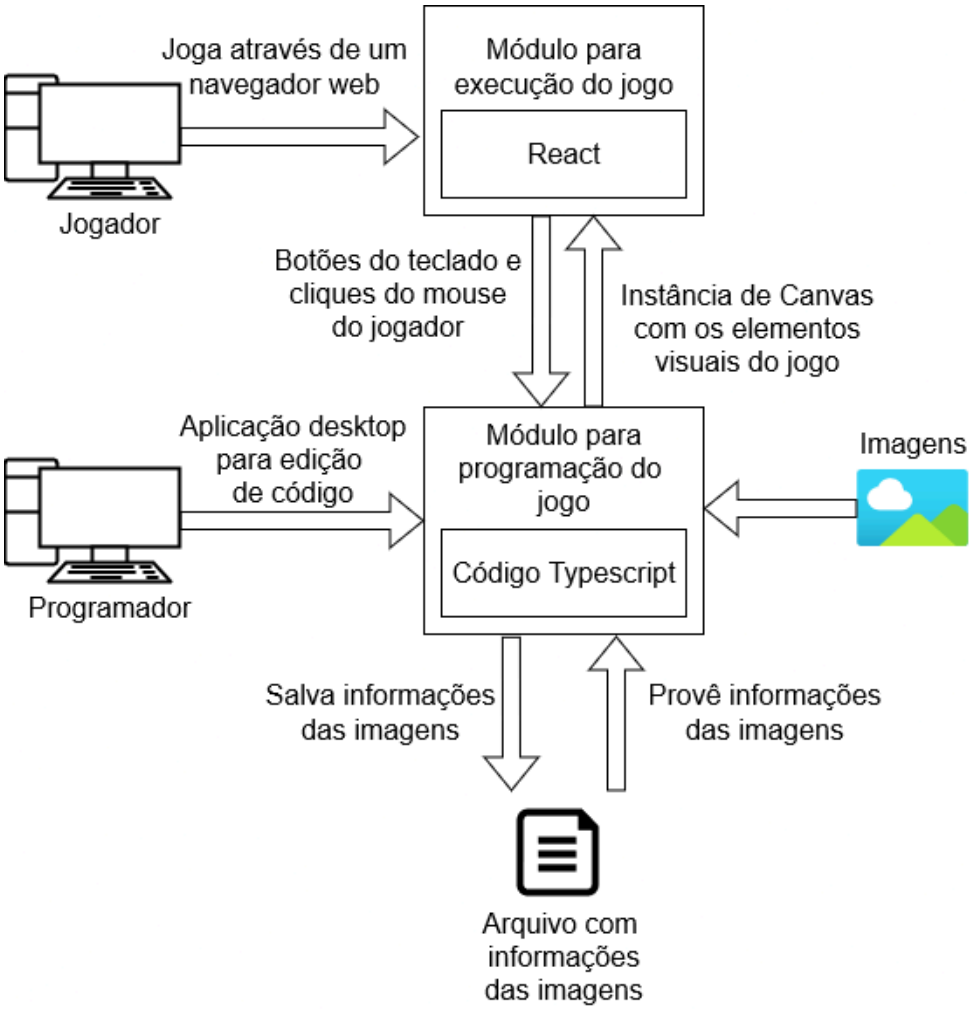
objetos;

- **Colocar imagens (RF4):** o motor deveria permitir ao usuário colocar as próprias imagens para serem usadas na ferramenta durante a criação do seu jogo. As imagens poderiam ser usadas para representar os personagens, itens e objetos;
- **Colocar sons (RF5):** o motor deveria permitir ao usuário colocar os próprios efeitos sonoros e músicas para serem usados durante o desenvolvimento do seu jogo;
- **Executar o jogo (RF6):** o motor deveria permitir ao usuário executar o jogo e jogá-lo na ferramenta.

3.3 Arquitetura da Ferramenta

A ferramenta a ser desenvolvida é baseada na arquitetura apresentada na Figura 1. Como pode ser observado, essa arquitetura da ferramenta é dividida em duas partes: o módulo para execução do jogo, no qual o programador pode testar e jogar o jogo desenvolvido, e o módulo para programação do jogo, no qual todos os arquivos do código do jogo são guardados junto com todas as imagens que o usuário forneceu.

Figura 1 - Arquitetura da ferramenta



Fonte: Autor

O módulo para execução do jogo tem a responsabilidade de receber o código feito pelo usuário e executar o jogo para que o mesmo possa ser jogado. A exibição do jogo é exibida no navegador do computador em uma página *web* feita com a biblioteca *React*, que Fedosejev (2015) descreve como uma biblioteca para a construção de interfaces de usuário.

A biblioteca *React* constrói páginas nas linguagens HTML (usada para definir a estrutura da página) e CSS (usada para estilização das páginas). É possível utilizar a biblioteca com *JavaScript,* que Flanagan (2012) descreve como uma linguagem de programação web usada pela ampla maioria dos sites, ou *TypeScript,* que segundo Bierman, Abadi e Torgersen (2014) é uma extensão de *JavaScript* com a intenção de facilitar o desenvolvimento de aplicações em *JavaScript* de larga escala. Neste trabalho optou-se por utilizar a tecnologia *TypeScript*. Alguns motivos

influenciaram esta escolha. Um deles é que *TypeScript* é uma linguagem fortemente tipada, o que ajuda a detectar erros durante o desenvolvimento. Além disso, ela proporciona mais recursos de programação orientada a objetos do que a linguagem *JavaScript*, como interfaces e sobrecarga de métodos, o que ajudou na criação de objetos de jogo. Outro motivo é que ela é a quarta linguagem mais utilizada no GitHub, segundo um artigo do Octoverse (2022). Isso quer dizer que muitos programadores poderão utilizar a ferramenta.

O módulo para programação do jogo é responsável por disponibilizar o ambiente no qual o usuário da ferramenta poderá programar o seu jogo. Esse ambiente fornece ao usuário várias funções para que o mesmo não precise criar todas as funções necessárias para o jogo. Nesse módulo, o usuário cria os arquivos de todos os objetos que serão usados no jogo e configura a forma como eles deverão se comportar. É neste ambiente que o usuário também pode criar funcionalidades específicas para o seu jogo caso a ferramenta não tenha o que o mesmo precise. Os arquivos são escritos em *TypeScript* e são usados pelo módulo para execução do jogo. O módulo de programação do jogo também armazena as imagens que são utilizadas, assim como informações sobre essas imagens que serão úteis durante a execução do jogo.

3.4 Implementação

Nesse tópico serão apresentados os módulos presentes na imagem da arquitetura da ferramenta, descrevendo as tecnologias utilizadas e como é o funcionamento de cada um deles.

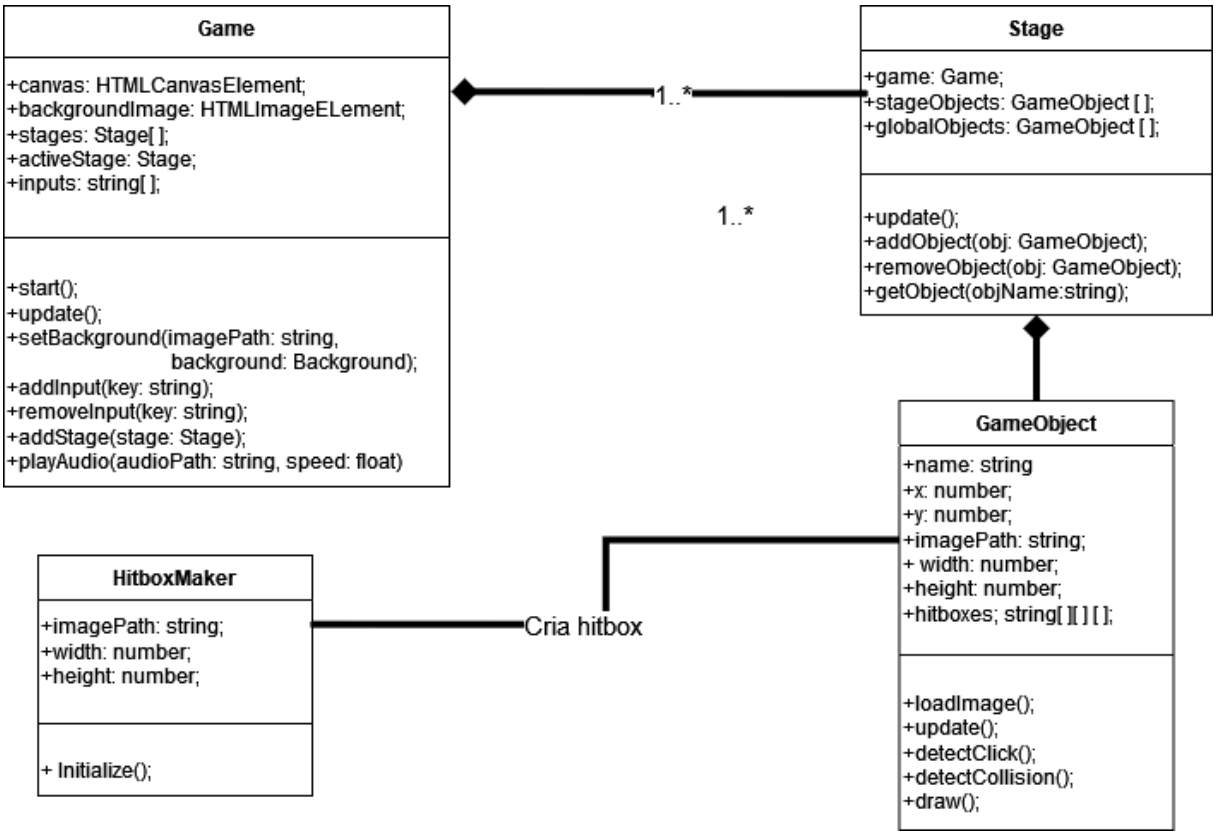
3.3.1 Módulo para programação do jogo

O módulo de programação do jogo oferece um ambiente onde o usuário pode desenvolver seu jogo utilizando várias funções prontas, evitando a necessidade de criar tudo do zero. Nele, o usuário define objetos, configura seus comportamentos e pode adicionar funcionalidades específicas. Os arquivos são escritos em TypeScript e o módulo também armazena imagens e informações úteis para a execução do jogo.

Nesse módulo é onde todos os arquivos do jogo estão localizados. O módulo é dividido em duas pastas: a *GameEngine* e a *MainGame*. A pasta *GameEngine*

contém as classes principais do jogo. São elas: *Game, Stage, GameObject* e *HitboxMaker*. A Figura 2 mostra como elas se relacionam. A pasta *MainGame* é onde o programador coloca seus próprios arquivos como objetos, na pasta *GameClasses*, fases, na pasta *Stages*, imagens, na pasta *GameImages* e sons, na pasta *GameSounds*.

Figura 2 - Diagrama de classes



A classe *Game* é responsável por gerenciar todo o funcionamento do jogo. Ela contém todas as fases, que são instâncias da classe *Stage*, que foram criadas pelo programador. Ela executa a função de atualizar, chamada *update*, da fase que está ativa no momento. Esta classe também contém uma instância da classe *Canvas*, uma classe do próprio *Javascript* ou *Typescript*, que é usada para exibir todos os elementos visuais do jogo, como a imagem de fundo, as imagens utilizadas em objetos e elementos textuais. Ela também guarda a imagem de fundo ativa no momento (*backgroundImage)*, a lista de fases e a fase ativa atualmente (*stages* e *activeStage*) além de também ter a lista de teclas que estão sendo pressionadas no teclado (*inputs*). Também existe o método *start*, que chama a função *update* de forma contínua para atualizar o estado da fase ativa. O método *setBackground* serve

para trocar a imagem de fundo, os métodos *addInput* e *removeInput* adicionam e removem as teclas pressionadas no teclado respectivamente, a função *addStage* adiciona fases ao jogo e a função *playAudio* reproduz um arquivo de áudio. Outra função da classe *Game* é reproduzir arquivos de áudio, ofertando também funções para pausar e parar a reprodução do mesmo. Além disso, ela armazena informações sobre quais botões do teclado o jogador está pressionando e qual a posição do cursor do mouse quando está em cima do *Canvas* exibido no módulo de exibição.

A classe *Stage* é usada para criar fases do jogo. Ela contém os objetos de jogo em uma lista chamada *stageObjects* para objetos da fase em si e *globalObjects* para objetos globais, da classe *GameObject*. Ela também tem os métodos *addObject* e *removeObjects* para adicionar e remover objetos respectivamente além de um método que recebe um objeto que tenha o nome passado para ele, o método *getObject*. Por exemplo, se existir um objeto na lista de objetos da fase com o atributo *name* igual a “jogador” e o método for usado com esse parâmetro, o objeto com aquele nome será retornado. Para fazer sua própria fase, o programador deve implementar uma nova classe que estenda a classe *Stage,* para que o método *update* esteja disponível. Esse método percorre todos os objetos da fase e chama a função *update* em cada um deles, para que executem o que foi programado quando foram criados.

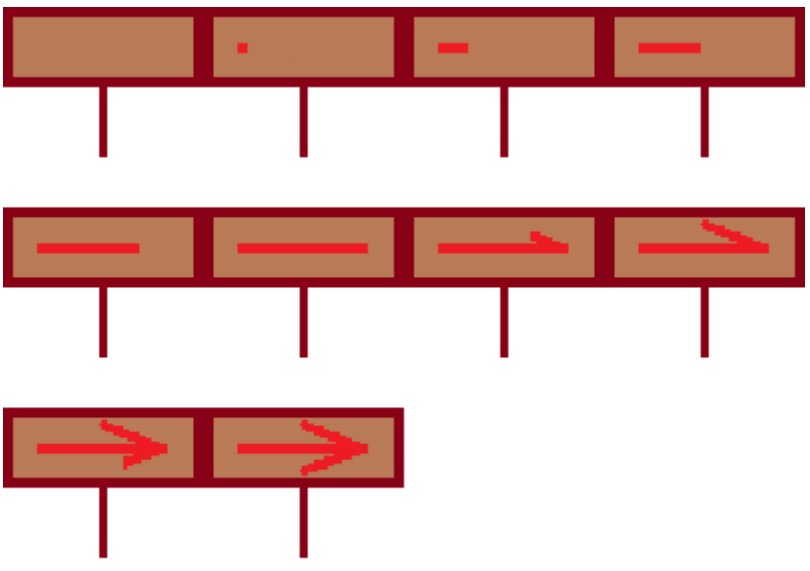
As fases também contém instâncias da classe *Background*, que são usadas para trocar a imagem de fundo do jogo. Elas contém o nome da imagem e alguns atributos para definir se a imagem deve repetir horizontalmente ou verticalmente. Existe também uma fase global. O propósito dessa fase é passar seus objetos para a fase atual do jogo, fazendo com que qualquer fase tenha aquele objeto. Se o programador colocou um valor para o nome do objeto, o programador também pode acessar aquele objeto.

A classe *GameObject* é como o programador cria objetos para serem usados no seu jogo. Ela tem atributos para armazenar dados como nome (*name*), tamanho (*width* para largura e *height* para altura), imagem com o *imagePath* que armazena o caminho até a imagem no computador, posição durante a execução do jogo, atributo “x” indica a posição no eixo x e “y” indica a posição no eixo y, se o objeto é clicável, se ele tem que ser desenhado espelhado horizontalmente, informações sobre

animações, entre outros. O nome serve para que o objeto possa ser selecionado a partir da lista de objetos de uma fase. É importante que os nomes dos objetos não se repitam para que o objeto correto seja retornado. A função *update* é chamada a partir da fase que o contém. Para criar um objeto, o programador deve implementar uma nova classe que estenda a classe *GameObject* e altere a função de *update* para fazer o comportamento desejado para o objeto, como mover para a direita caso o jogador pressione alguma tecla específica.

É importante que o programador utilize a versão da classe que dá suporte ao uso de animações (a classe *GameObjectDynamic*), caso queira que o objeto seja animado. Além disso, objetos com animações precisam receber uma imagem feita da forma correta, isto é, ela deve ser um *Spritesheet*. Por exemplo, se o programador quiser criar uma placa que tenha uma seta vermelha que deva aparecer aos poucos, e ele queira que essa seta tenha o tamanho de 100 pixels de largura por 100 pixels de altura, e que o objeto tenha 10 representações, a imagem usada pelo jogo deve ser composta de imagens menores, (neste caso são 10 imagens, cada uma com tamanho de 100 pixels de largura por 100 pixels de altura para formar uma imagem maior), e cada uma das imagem menores representa o *frame*, ou quadro de animação do objeto naquele momento. A Figura 2 exemplifica como isso deve ser feito.

Figura 3 - Exemplo de imagem de objeto



Fonte: Autor

Além disso, existe o atributo *hitboxes*, que é uma lista de todas *hitboxes* do objeto. Tendo a Figura 2 como exemplo, o atributo seria uma lista com 10 elementos, cada elemento sendo uma lista de todas as partes da imagem, que no caso da Figura 2, como cada uma das pequenas imagens é apenas uma parte, cada elemento dessa lista seria composto por apenas um elemento.

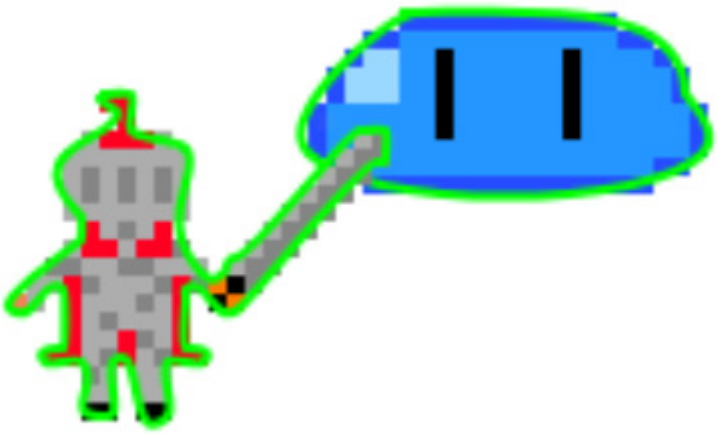
A classe *HitboxMaker* é responsável por ler as imagens e dividi-las em partes com base no tamanho definido pelo programador no objeto que as utiliza. Usando a imagem anterior como exemplo, ela seria dividida em 10 partes de 100 de tamanho e altura. Após dividir a imagem, o próximo passo é fazer o contorno dessa imagem, também chamado de *hitbox* em muitos jogos, para que ela seja utilizada quando for necessário saber se um objeto está colidindo com outro. Isso pode acontecer, por exemplo, se o jogador acertar um inimigo ou vice-versa. Ao terminar de ler a imagem, algumas informações são armazenadas em um arquivo no computador, como quantas partes essa imagem tem, a posição de cada uma das divisões, e as informações sobre o contorno da imagem. Essas informações são obtidas pelo programa detectando quais áreas da imagem são transparentes e quais não são. A Figura 4 mostra como esse arquivo é definido, *hitboxCount* é o número de divisões da imagem, *hitboxes* contém a lista de contornos da imagem, e *animationImagePosition* tem a posição de cada divisão da imagem. A Figura 5 mostra com uma linha verde a posição do contorno, ou *hitbox*, da imagem. Para detectar uma colisão, é verificado se a linha verde usada para mostrar o contorno toca em algum ponto da linha verde do outro objeto.

Figura 4 - Arquivo gerado para salvar informações de imagem


```
"inimigo.png": {
    "hitboxCount": 2,
    "hitboxes": [
        [
            [
                "m72,60c23.50052,0 63,-4.55183 63,18c0,0.3566 8.5896,8.4104 4,13c-12.73029,12.73029 -74.26971,12.73029 -87,
                0c-7.9307,-7.9307 17.47273,-31 24,-31"
            ]
        ],
        [
            [
                "m72,56c24.47352,0 63,-4.80183 63,19c0,0.46558 8.07465,11.92535 4,16c-12.73029,12.73029 -74.26971,12.73029 -87,
                0c-9.85404,-9.85404 17.09257,-35 24,-35"
            ]
        ]
    ],
    "animationImagePosition": [{"x": 0,"y": 0},{"x": 200,"y": 0}]
```


Fonte: Autor

Figura 5 - Exemplo de contorno em objetos



Fonte: Autor

Com essas informações, tudo que o programador tem que fazer para trocar a imagem que está sendo exibida é alterar o valor do atributo *activeFrame* de uma instância da *GameObject* para um número que representa a imagem que deve ser usada no momento. Por exemplo, considerando-se a imagem da Figura 3, os números possíveis seriam de 0 até 9. O programador também pode definir nomes para cada estado de animação, quantidade de ciclos de jogo entre cada troca de imagem, chamados de duração na ferramenta, e definir uma lista de quais imagens serão utilizadas naquela animação para que as animações sejam feitas mais facilmente. Por exemplo, pode-se ter um estado chamado início com uma lista com os números 0 a 4, e outro chamado final com os números de 5 a 9, ambos com 6 ciclos de duração. Dessa forma, é possível trocar a animação que é exibida apenas trocando em qual estado o objeto está. Essas informações são armazenadas antes do jogo começar para que a execução do mesmo seja mais rápida, pois as informações já estarão disponíveis imediatamente e não precisarão ser calculadas na hora em que forem utilizadas.

3.3.2 Módulo para exibição do jogo

Esse módulo é responsável por exibir os elementos visuais do jogo, como as imagens fornecidas pelo usuário da ferramenta. Isso é feito a partir de uma instância da classe *Canvas* que vem do módulo de programação do jogo. Além disso, esse módulo também tem a função de passar os botões do teclado que o jogador

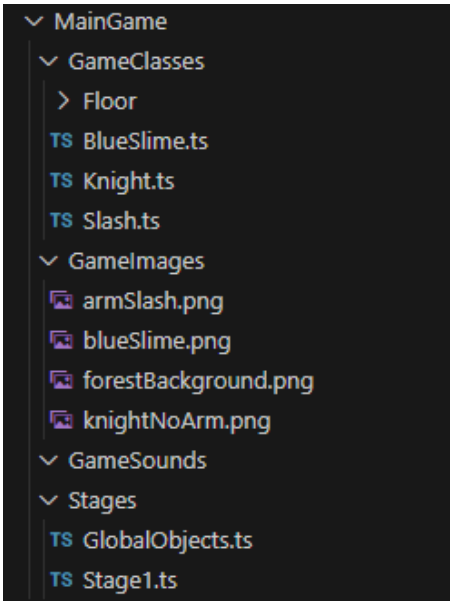
pressiona, a posição do cursor do mouse e se o mesmo está sendo clicado para o módulo de programação do jogo. Esse módulo é executado em um navegador de internet e foi feito com a biblioteca *React* com a linguagem *Typescript*.

Outra função deste módulo consiste em inicializar os objetos das fases criadas para carregar suas imagens com o *HitboxMaker,* caso as informações da imagem ainda não estejam armazenadas. Nesse caso, a função de salvá-las é chamada. É importante que o programador crie uma fase com todos os objetos que vai usar com o propósito de guardar todas as imagens utilizadas com o *HitboxMaker* para que esses objetos possam ser usados no jogo.

3.4 Estudo de caso

Para validar o funcionamento da ferramenta criada neste trabalho, foi desenvolvido um jogo simples. De início, o código do repositório no *GitHub* do link https://github.com/mmurilo03/tcc deve ser clonado para uma ferramenta de edição de código. Depois disso, acessa-se a pasta *MainGame*, onde são criados os arquivos do jogo. Primeiramente coloca-se todas as imagens necessárias para o jogo na pasta *GameImages*. Depois, cria-se os arquivos dos objetos de jogo na pasta *GameClasses*. E, por fim, cria-se um arquivo para a fase do jogo. A Figura 6 exemplifica como esses arquivos ficam distribuídos ao final da criação de todos eles e a Figura 7 mostra um exemplo de implementação de um objeto de jogo.

Figura 6 - Exemplo de estrutura dos arquivos



Fonte: Autor

Figura 7 - Exemplo de implementação de um objeto de jogo com animação

```
import { GameObjectDynamic } from "../../gameEngine/GameObject/GameObjectDynamic";

export class BlueSlime extends GameObjectDynamic {
    constructor(game: Game, x: number, y: number) {
        super(
            {
                clickable: false,
                game: game,
                x: x,
                y: y,
                height: 200,
                width: 200,
                imagePath: "blueSlime.png",
            },
            {
                animationFrames: {
                    animacao: { duration: 15 , frames: [0, 1, 2, 3] },
                },
                state: "animacao",
            }
        );
    }
}
```

Fonte: Autor

Na Figura 7 é ilustrado como é criado um objeto de jogo com animação. Ao criar a classe que estende a classe *GameObjectDynamic*, usa-se a palavra *super* para criar a instância da classe. Dentro do *super* são passados os atributos *clickable* para dizer se o objeto é clicável, uma instância de *Game* para saber informações sobre o jogo, as posições dos eixos “x” e “y” do objeto, a altura e largura e por fim o nome da imagem usada. Após isso, são passadas as informações sobre as animações, como o nome do estado daquela animação, chamada de “animacao” que, nesse caso, corresponde à duração de cada imagem, sendo 15 ciclos de jogo, e quais partes da imagem serão usadas, que são as 4 primeiras imagens, começando de 0 e indo até 3.

Os inimigos do jogo aparecem na tela e o jogador ganha pontos ao atacar um inimigo e perde pontos se um inimigo tocar o jogador. Para movimentar o personagem, o jogador pressiona as teclas “W”, “A”, “S” e “D” para se mover para cima, esquerda, baixo e direita respectivamente, e aperta a tecla Espaço para atacar os inimigos. A Figura 8 mostra um exemplo de como detectar se o jogador está

pressionando a tecla “D” no teclado. Primeiro coloca-se o método *update* na classe e usa-se o método *super.update* para que as animações continuem funcionando, já que o método foi sobrescrito. Após isso, verifica-se se a letra “D” está na lista de teclas pressionadas, e se estiver, move-se o personagem para a direita.

Figura 8 - Exemplo de como detectar tecla pressionada

```
update() {
    super.update();

    if (this.game.inputs.includes("d")) {
        this.x += 5;
    }
}
```

Fonte: Autor

Depois disso, os objetos podem ser colocados na fase. Cria-se, então, um atributo para o personagem do jogo, *knight*, para o inimigo, *enemy*, e pontuação. É passado o nome da fase para a chamada de *super*, sendo “Stage 1” e são criadas instâncias do personagem *knight* e inimigo *blueSlime* e são adicionadas na fase com o método *addObject*. Coloca-se então a imagem de fundo com *setBackground* e adicionamos a pontuação com o método *addText* para colocar texto na tela, para esse método é passado o tamanho do texto, um identificador, o texto exibido, e a posição no eixo “x” e “y” respectivamente. Por fim, o método *update* é sobrescrito para verificar se o inimigo está tocando no jogador com o método *checkCollision*, se estiver, o jogador perde um ponto, e se o jogador estiver atacando o inimigo, usamos o método *attack* para verificar adicionamos um ponto caso o inimigo seja atacado. Para atualizar a pontuação, o texto é removido da tela com *clearText* e criado novamente da mesma forma. A Figura 9 mostra como a implementação da fase deve ficar para o funcionamento correto.

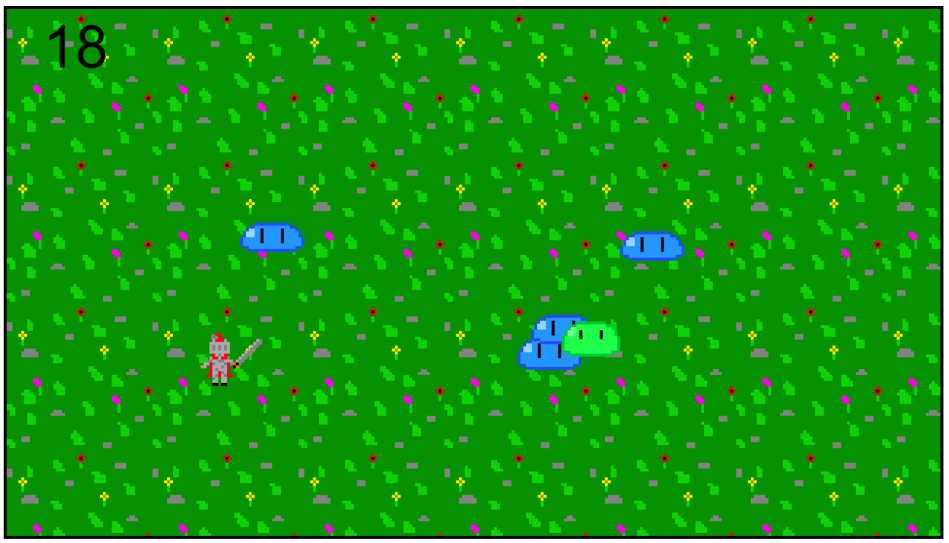
Figura 9 - Exemplo de implementação de fase

```
import { Game } from "../../gameEngine/Game";
import { Background } from "../../gameEngine/GameStages/Background";
import { Stage } from "../../gameEngine/GameStages/Stage";
import { BlueSlime } from "../GameClasses/BlueSlime";
import { Knight } from "../GameClasses/Knight";

export class Stage1 extends Stage {
    knight: Knight;
    enemy: BlueSlime;
    score: number = 0;
    constructor(game: Game) {
        super(game, "Stage 1");

        this.knight = new Knight({ game, x: 200, y: 5 });
        this.addObject(this.knight);
        this.enemy = new BlueSlime({ game, x: 200, y: 5 });
        this.addObject(this.enemy);

        const bg1 = new Background("forestBackground.png");
        this.addBackground(bg1);
        this.setBackground("forestBackground.png");

        this.game.addText({
            fontSize: "80px",
            id: "score",
            text: `${this.score}`,
            textPositionX: 50,
            textPositionY: 80,
        });
    }

    update(): void {
        super.update();
        this.game.clearText();
        if (this.enemy.checkCollision(this.knight)) {
            this.score -= 1;
        }
        if (this.knight.attack(this.enemy)) {
            this.score += 1;
        }
        this.game.addText({
            fontSize: "80px",
            id: "score",
            text: `${this.score}`,
            textPositionX: 50,
            textPositionY: 80,
        });
    }
}
```

Fonte: autor

Os inimigos do jogo se movem na direção do jogador e sempre que existem menos de 5 inimigos na tela, um novo inimigo aparece. A Figura 10 mostra como é o visual da tela do jogo no final.

Figura 10 - Imagem do jogo desenvolvido



Fonte: Autor

4. CONCLUSÃO

Os jogos digitais são um interesse muito comum entre as pessoas, e o grande crescimento dessa indústria é uma prova disso. A gamificação é um exemplo de algo que os jogos trouxeram e que tem um grande impacto na sociedade. Além disso, os jogos são um interesse comum para programadores, e muitos deles têm interesse no seu desenvolvimento.

Com o propósito de diminuir a complexidade inicial na criação de jogos, este trabalho propõe o desenvolvimento de um motor de jogos, uma ferramenta que oferece muitos recursos necessários para o desenvolvimento de jogos. A ferramenta desenvolvida tem como objetivo facilitar a programação de jogos oferecendo recursos já prontos para o usuário usar, além de dar ao usuário a opção de criar suas próprias funcionalidades e usá-las no ambiente de desenvolvimento do seu jogo.

A ferramenta foi desenvolvida seguindo todos os requisitos funcionais estabelecidos, fazendo ser possível criar um jogo com a linguagem de programação *Typescript* que é executado em um navegador de internet e que pode ser usada

para ajudar no aprendizado de programação, oferecendo uma aproximação gamificada que pode ser mais atrativa do que os métodos convencionais.

Existem trabalhos futuros que podem ser feitos para estender a ferramenta proposta neste trabalho. Alguns deles são a criação de um módulo para a ferramenta onde possam ser criadas imagens para o jogo, o suporte para o uso de controles de videogame para jogar os jogos, a otimização do desempenho da ferramenta quando o desenvolvedor precise de centenas de objetos sendo processados ao mesmo tempo e um sistema de depuração de código para ajudar programadores a detectar erros em seus códigos.

REFERÊNCIAS

AMÉLIO, C. de O. A indústria e o mercado de jogos digitais no brasil. **XVII SBGames, Foz do Iguaçu, Paraná, Brasil**, p. 1497–1506, 2018.

ANDRADE, A. Game engines: a survey. **EAI Endorsed Transactions on Serious Games**, v. 2, n. 6, 2015.

BEZERRA, A. P. B.; BATISTA, E. D. de A.; SOUSA, J. F. de; OLEGÁRIO, T. de A.; EVANGELISTA, E. B.; LIRA, R. V. Bora jogar: desenvolvimento de jogos para auxiliar na aprendizagem de programação. **Research, Society and Development**, v. 11, n. 5, p. e15511527668–e15511527668, 2022.

BIERMAN, G. M.; ABADI, M.; TORGERSEN, M. Understanding typescript. In: JONES, R. E. (Ed.). ECOOP. Springer, 2014. (Lecture Notes in Computer Science, v. 8586), p. 257–281. ISBN 978-3-662-44201-2. Disponível em: ⟨http: //dblp.uni-trier.de/db/conf/ecoop/ecoop2014.html#BiermanAT14⟩.

BILAS, S. A data-driven game object system. In: **Game Developers Conference Proceedings**. [S.l.: s.n.], 2002. v. 2

CHADY, M. Theory and practice of game object component architecture. In: **Game Developers Conference**. [S.l.: s.n.], 2009.

CLIBURN, D. C.; MILLER, S. Games, stories, or something more traditional: the types of assignments college students prefer. In: **Proceedings of the 39th SIGCSE technical symposium on Computer science education**. [S.l.: s.n.], 2008. p. 138–142.

DETERDING, S.; DIXON, D.; KHALED, R.; NACKE, L. From game design elements to gamefulness: defining”gamification”. In: **Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments**. [S.l.: s.n.], 2011. p. 9–15.

FEDOSEJEV, A. React. js essentials. [S.l.]: Packt Publishing Ltd, 2015.

FLANAGAN, D. JavaScript: o guia definitivo. [S.l.]: Bookman Editora, 2012.

GITHUB. Octoverse 2022: top programming languages. 2022. Disponível em: https://octoverse.github.com/2022/top-programming-languages. Acesso em: 09 out. 2024.

INCE, E. Y. Students' perceptions on learning programming with codingame. **International Journal of Technology in Teaching and Learning**, ERIC, v. 17, n. 1, p. 38–46, 2021.

JÚNIOR, G. P. dos S.; FECHINE, J. M.; COSTA, E. de B. Analogus: Um ambiente para auxílio ao ensino de programação orientado pelo raciocínio por analogia. In: **Anais do Workshop sobre Educação em Computação (WEI)**. [S.l.: s.n.], 2009. p. 499-508.

KURKOVSKY, S. Can mobile game development foster student interest in computer science? In: IEEE. **2009 International IEEE Consumer Electronics Society's Games Innovations Conference**. [S.l.], 2009. p. 92–100.

MILLINGTON, I. **Game physics engine development**. [S.l.]: CRC Press, 2007.

NETO, V. d. S. M. A utilização da ferramenta scratch como auxílio na aprendizagem de lógica de programação. In: **Anais dos Workshops do Congresso Brasileiro de Informática na Educação**. [S.l.: s.n.], 2013. v. 2, n. 1.

OPRACH, J. M. Aprendizagem musical do jogo rocksmith: possibilidades no campo da educação musical. 2021.

PORTER, N. Component-based game object system. **Carleton University**, 2012

PYGAME. About. Disponível em: https://www.pygame.org/wiki/about. Acesso em: 18

nov. 2023

ROCHA, R. V. da; BITTENCOURT, I. I.; ISOTANI, S. Análise, projeto, desenvolvimento e avaliação de jogos sérios e afins: uma revisão de desafios e oportunidades. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [S.l.: s.n.], 2015. v. 26, n. 1, p. 692.

SCAICO, P. D. Um estudo sobre o desenvolvimento de interesse pela aprendizagem de programação. Universidade Federal de Pernambuco, 2018.

SILVA, J. A. L. da; OLIVEIRA, F. C. S.; MARTINS, D. J. S. Storytelling e gamificação como estratégia de motivação no ensino de programação com python e minecraft. **SBC–Proceedings of SBGames**, 2017.

SILVA, T. S. C. da; TEDESCO, P. C. de A.; MELO, J. C. de. A importância da motivação dos estudantes e o uso de técnicas de engajamento para apoiar a escolha de jogos no ensino de programação. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [S.l.: s.n.], 2014. v. 25, n. 1, p. 11.

STEUER, J.; BIOCCA, F.; LEVY, M. R. et al. Defining virtual reality: Dimensions determining telepresence. **Communication in the age of virtual reality**, v. 33, p. 37–39, 1995

ZAMITH, M.; VALENTE, L.; CLUA, E. Game loop model properties and characteristics on multi-core cpu and gpu games. **SBGames 2016**, 2016.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Cajazeiras - Código INEP: 25008978
Rua José Antônio da Silva, 300, Jardim Oásis, CEP 58.900-000, Cajazeiras (PB)
CNPJ: 10.783.898/0005-07 - Telefone: (83) 3532-4100

Documento Digitalizado Ostensivo (Público)

Trabalho de Conclusão de Curso

Assunto:	Trabalho de Conclusão de Curso
Assinado por:	Murilo Ferreira
Tipo do Documento:	Anexo
Situação:	Finalizado
Nível de Acesso:	Ostensivo (Público)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Murilo de Oliveira Ferreira**, **DISCENTE (202122010022) DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - CAJAZEIRAS**, em 22/10/2024 18:08:39.

Este documento foi armazenado no SUAP em 22/10/2024. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1287899
Código de Autenticação: f733785532