UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE SOFTWARE

ADELINE RODRIGUES CRUZ WYSE GUIMARÃES

INTELIGÊNCIA ARTIFICIAL GENERATIVA NO DESENVOLVIMENTO DE JOGOS: UM ESTUDO COMPARATIVO DAS FERRAMENTAS GEMINI E CHATGPT

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO
2024

ADELINE RODRIGUES CRUZ WYSE GUIMARÃES

INTELIGÊNCIA ARTIFICIAL GENERATIVA NO DESENVOLVIMENTO DE JOGOS: UM ESTUDO COMPARATIVO DAS FERRAMENTAS GEMINI E CHATGPT

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Software da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Dr. Paulo Augusto Nardi

CORNÉLIO PROCÓPIO
2024

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Cornélio Procópio
Nome da Diretoria
Departamento Acadêmico de Computação
Engenharia de Software

TERMO DE APROVAÇÃO

INTELIGÊNCIA ARTIFICIAL GENERATIVA NO DESENVOLVIMENTO DE JOGOS: UM ESTUDO COMPARATIVO DAS FERRAMENTAS GEMINI E CHATGPT

por

Adeline Rodriues Cruz Wyse Guimarães

Este Trabalho de Conclusão de Curso de graduação foi julgado adequado para obtenção do Título de “Bacharel em Engenharia de Software” e aprovado em sua forma final pelo Programa de Graduação em Engenharia de Software da Universidade Tecnológica Federal do Paraná.

Cornélio Procópio, 20 de junho de 2024

Dr. Paulo Augusto Nardi

Prof.Gabriel Canhadas Genvigir

M.ª'Adriane Carla Anastácio da Silva

“A Folha de Aprovação assinada encontra-se na Coordenação do Curso”

Agradeço aos meus pais, meu orientador, as equipes da COGETI-CP e Gadir-CP e aos meus amigos, em especial a Giulia e a Neidielli.

AGRADECIMENTOS

Gostaria de expressar minha sincera gratidão aos meus pais, meu orientador, as equipes da COGETI-CP e Gadir-CP e aos meus amigos, em especial a Giulia e a Neidielli. Agradeço a cada um de vocês pelo apoio, orientação e encorajamento ao longo dessa jornada. Sem vocês, eu não teria alcançado os resultados que obtive. Agradeço muito por ter pessoas tão especiais na minha vida.

RESUMO

R.C.W. Guimarães, Adeline. Inteligência Artificial Generativa no desenvolvimento de jogos: Um Estudo Comparativo das Ferramentas Gemini e ChatGPT. 2024. 33 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Software, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2024.

O desenvolvimento de jogos é um processo demorado, cheio de desafios como *burn-out* e longos períodos de desenvolvimento. Neste trabalho foi feito uma análise do uso de Inteligência Artificial Generativa (IAG) com o intuito de melhorar o desenvolvimento e qualidade dos jogos. Para exemplificar foram desenvolvidos três protótipos, um sem a utilização de ferramentas e dois utilizando ferramentas de IAG para geração de código e instruções de implementação. Foram avaliados critérios como tempo de desenvolvimento, desempenho e qualidade de código. Após uma análise individual e comparativa das métricas dos três protótipos, foi possível identificar possíveis benefícios e impactos da IAG no desenvolvimento. O objetivo deste trabalho é fornecer *insights* para a indústria de jogos em relação à adoção e implementação dessa tecnologia, visando aprimorar a eficiência no desenvolvimento de jogos.

Palavras-chave: Desenvolvimento de Jogos. Inteligência Artificial Generativa. Prototipação.

ABSTRACT

R.C.W. Guimarães, Adeline. Generative Artificial Intelligence in Game Development: A Comparative Study of the Tools ChatGPT and Gemini. 2024. 33 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Software, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2024.

Game development is a time-consuming process with challenges such as burnout and long development cycles. In this work, an analysis was conducted on the use of Generative Artificial Intelligence (GAI) with the aim of improving game development and quality. To illustrate, three prototypes were developed: one without the use of tools and two using GAI tools for code generation and implementation instructions. Criteria such as development time, performance, and code quality were evaluated. After an individual and comparative analysis of the metrics of the three prototypes, it was possible to identify potential benefits and impacts of GAI on development. The goal of this work is to provide insights to the game industry regarding the adoption and implementation of this technology, aiming to enhance efficiency in game development.

Keywords: Game Development. Generative Artificial Intelligence. Prototyping.

LISTA DE ABREVIATURAS E SIGLAS

GAN	*Generative Adversarial Networks*
IA	Inteligência Artificial
IAG	Inteligência Artificial Generativa
LLM	*Large Language Model*
PC	*Personal Computer*
VAE	*Variational Autoencoders*

LISTA DE TABELAS

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A indústria de jogos vem crescendo financeiramente nos últimos anos. Com o mercado global de jogos espera-se chegar a 211.2 bilhões de dólares até 2025 (NEWZOO, 2022). Ela está dividida em segmentos como periféricos, jogos (*mobile*, para *personal computers* - PC - e consoles). No fim o principal produto é o jogo.

O desenvolvimento de um jogo envolve várias etapas, desde o conceito até a pós-produção. O processo é estressante e demorado, o que pode levar a vários problemas. Um dos problemas mais significativos é a exaustão da equipe devido às longas horas de trabalho necessárias para concluir um projeto, que leva ao conhecido *burn-out*. Outro problema é o tempo do processo de desenvolvimento que, na indústria para jogos AAA (jogos com alto orçamento, grande porte e alta tecnologia, produzidos e distribuídos por grandes empresas (ANASTASIA, 2024)), pode chegar a mais de 5 anos. Alguns jogos, como *Final Fantasy XV, The Last Guardian* e *Prey (2011)*, podem demorar mais de 9 anos para serem desenvolvidos (HICKS, 2022).

O problema de longos tempos de desenvolvimento muitas vezes é causado por ciclos de desenvolvimento que ficam parados em atividades como prototipagem, criação de história, personagens, *assets*, código e testes.

Uma alternativa para diminuir o tempo de desenvolvimento nessas atividades é a Inteligência Artificial Generativa (IAG), uma tecnologia que usa aprendizado de máquina para gerar conteúdo de forma autônoma, um campo da Inteligência Artificial(IA)(FOSTER, 2019). Essa tecnologia tem o potencial de revolucionar o processo de desenvolvimento de jogos ao reduzir o tempo necessário para a sua finalização. Já existem exemplos dessa tecnologia sendo usada por grandes empresas como a *Ubisoft*, que usou uma IAG para criar as multidões em seu jogo *Assassin's Creed Unity*(COURNOYER, 2022), e a *CD Projekt Red* que utilizou a IA JALI para fazer a animação do *lip-sync* em 10 idiomas diferentes para o jogo *Cyberpunk 2077*(EDWARDS; LANDRETH; POPŁAWSKI, 2020).

Este trabalho explora o processo de desenvolvimento de um jogo de tiro estilo *arcade* com apoio de IAG, identificando possíveis maneiras de utilizar essa tecnologia para automatizar parte de seu desenvolvimento. Foram criados três protótipos na *engine Unity3d*: um protótipo criado sem apoio de IAG e dois criados usando cada um uma ferramenta de IAG diferente: ChatGPT e Gemini.

O objetivo principal é fornecer *insights* em relação à adoção dessa tecnologia, com o intuito de aprimorar o processo de desenvolvimento de jogos.

Os objetivos específicos são: identificar o tempo de entrega total, tempo de ajuste (tempo que o desenvolvedor demora para adequar o código gerado ao projeto), desempenho, e qualidade do código, de cada protótipo, para assim identificar os impactos da IAG no desenvolvimento.

1.1 ORGANIZAÇÃO DO TRABALHO

No Capítulo 2 deste trabalho é feita uma explicação das tecnologias utilizadas no mercado. Além disso, são comparadas as duas ferramentas de IAG escolhidas e apresentados trabalhos relacionados e usos dessa tecnologia na indústria. Quanto ao Capítulo 3, são descritos os procedimentos adotados no desenvolvimento dos protótipos, incluindo requisitos e métricas de análise. O Capitulo 4 traz detalhes sobre o processo de implementação individual de cada protótipo. No Capítulo 5 são apresentadas as métricas utilizadas para a análise dos resultados,

assim como a análise individual e comparativa dos três protótipos. Por fim no Capitulo 6, são apresentados os resultados e retomados pontos importantes para ressaltar os objetivos do trabalho.

2 ANÁLISE DE TECNOLOGIA

Atualmente existem diversas ferramentas que fazem o uso de IAG para a geração de conteúdo no mercado. Para conseguir escolher a mais indicada para o uso é preciso saber como essa tecnologia funciona e seus possíveis usos.

2.1 INTELIGÊNCIA ARTIFICIAL GENERATIVA

IAG refere-se ao uso de algoritmos de aprendizado de máquina para gerar novos dados que se assemelham a um determinado tipo de dado, como imagem, música, texto, vídeo ou código (FOSTER, 2019).

O aprendizado de máquina é um ramo da IA que se concentra no desenvolvimento de algoritmos capazes de aprender com dados e fazer previsões ou decisões. Ele permite computadores adquirir conhecimento automaticamente, identificar padrões e melhorar seu desempenho por meio da experiência. O processo de aprendizado de máquina normalmente envolve duas etapas principais: treinamento e inferência. Durante a fase de treinamento, o algoritmo é exposto a um conjunto de dados rotulado, onde a saída desejada ou variável de destino é conhecida para cada exemplo de entrada. Ao analisar os padrões e relacionamentos nos dados de treinamento, fase de inferência, o algoritmo aprende a fazer previsões ou decisões com base nos recursos de entrada.(SIMEONE, 2018)

Segundo Foster (2019) a IAG envolve treinar um modelo de aprendizado de máquina em um grande conjunto de dados e, em seguida, usar esse modelo de forma probabilística para gerar dados semelhantes em estilo ou estrutura aos originais. Existem vários tipos de modelos generativos de IA, incluindo *Autoencoders*, *Variational Autoencoders* (VAEs) e *Generative Adversarial Networks* (GANs) explicados a seguir.

- *Autoencoder*: um modelo de rede neural composto por duas partes principais: o codificador (encoder) e o decodificador (decoder). O codificador transforma os dados de entrada em uma representação de menor dimensão. O decodificador então tenta reconstruir os dados originais a partir dessa codificação. A principal finalidade dos *autoencoders* é aprender uma representação compacta dos dados, preservando ao máximo a informação essencial. São utilizados em tarefas como compressão de dados e extração de características e geração de dados.
- VAE (Variational Autoencoder): é um tipo específico de *autoencoder* que, além de reconstruir dados, aprende uma distribuição de probabilidade sobre os dados de entrada. Isso permite a geração de novos dados que se assemelham aos dados originais. Assim como os *autoencoders* tradicionais, os VAEs são aplicáveis na síntese de dados, assim como para a geração de texto, código, imagem e música.
- GAN: um tipo de algoritmo que envolve duas redes neurais, uma rede geradora que cria novos dados e uma rede discriminadora que tenta distinguir entre os dados gerados e os dados reais. A rede geradora é treinada para criar dados o mais semelhantes possível aos dados reais, enquanto a rede discriminadora é treinada para identificar se um determinado dado é real ou falso. As GANs podem ser usadas para geração de imagem, vídeo e música, bem como para síntese e aumento de dados.

A principal diferença entre os *autoencoders* e os VAEs está na abordagem da codificação. Enquanto os *autoencoders* tradicionais produzem uma codificação fixa, os VAEs aprendem uma distribuição de probabilidade sobre as codificações, o que permite a geração de novos dados a partir de amostras dessa distribuição. Já os GANs diferem ao utilizar duas redes que

competem entre si e resultam em uma maior capacidade de gerar dados de melhor qualidade. Os GANs não apenas recriam dados, mas criam dados novos e convincentes, muitas vezes superando VAEs em qualidade visual.

Autoencoders são mais indicados para compressão de dados, redução de dimensionalidade, extração de características e redução de ruído. VAEs para geração de novos dados que se assemelham aos dados de entrada. E GANs para tarefas que exigem a criação de dados realistas e detalhados, como na geração de imagens, vídeos e música.

Essa tecnologia pode ser usada para uma ampla gama de aplicações, incluindo arte, música e literatura e código.

2.1.1 *Large Language Models*

Large Language Model (LLM) é um modelo de IAG projetado para interpretar e gerar conteúdo em formato de linguagem humana. Grande parte dos LLMs modernos utilizam a arquitetura Transformadora, que é uma arquitetura de aprendizado profundo que utiliza uma combinação de autoencoders, VAEs e GANs para processar entrada e saída. (OZDEMIR, 2023)

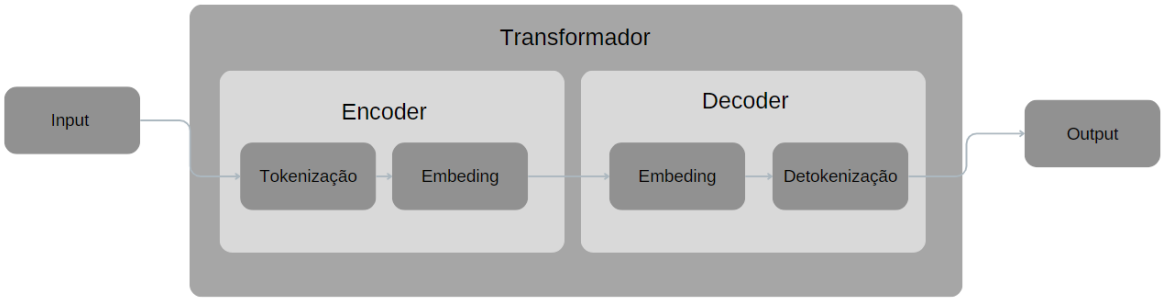

Figura 1 – Diagrama de funcionamento de um LLM - Fonte: Elaboração Própria.

A Figura 1 exemplifica o funcionamento de um LLM na arquitetura Transformador. Ao passar um *input* de texto dentro do *Encoder* é feito um processo de tokenização, onde o input é dividido em pequenas partes para poder ser processado. Os dados tokenizados passam por um processo de *embeding* onde são processados e armazenados em vetores de informação. Esses vetores são passados para o *Decoder* responsável por tratar essa informação em um novo processo de *embeding* onde gera a informação de resposta e passar por uma detokenização que traz dados em formato de linguagem humana. Quais tecnologias e arquiteturas são usadas dentro de *Encoders* e *Decoders* são especificas de cada modelo e muitas vezes mantidas em sigilo por seus desenvolvedores. (OZDEMIR, 2023).

2.2 FERRAMENTAS ESCOLHIDAS

Para esse trabalho foram escolhidas duas ferramentas de IAG, conhecidas como *chatbots* que utilizam LLM para a geração de respostas a perguntas do usuário.

2.2.1 Gemini

O Gemini é um LLM sendo desenvolvido pelo *Google AI*, treinado em um grande conjunto de dados de texto e código. Ele pode gerar texto, traduzir idiomas, escrever diferentes tipos de conteúdo e responder a perguntas de forma informativa.

O Gemini analisa o input do usuário e compara a tarefa pedida a sua base de dados e gera a resposta no formato de explicações e exemplo de código.(GOOGLE, 2023)

2.2.1.1 Prós

- Especialização em programação: O Gemini foi desenvolvido com foco em programação. Assim o modelo tem um grande conhecimento em diversas linguagens de programação. Incluindo C, C++ e Python usadas no desenvolvimento de jogos.(CIRIACO, 2023)
- Aumento de produtividade: Pode gerar grandes quantidades de código rapidamente, o que possibilita mais tempo aos desenvolvedores para focar em tarefas de qualidade e planejamento.(CIRIACO, 2023)
- Checagem de código: Pode checar e sugerir mudanças em códigos já existentes.(GOOGLE, 2023)

2.2.1.2 Contras

- Qualidade Variável: O Gemini pode gerar código de baixa qualidade por ser uma ferramenta ainda em desenvolvimento.(GOOGLE, 2023)
- Respostas complexas: O código pode ser de difícil entendimento. Especialmente para leigos.(GOOGLE, 2023)
- Código depreciado: Por usar informações antigas pode levar a geração de códigos que utilizam métodos já em desuso.(GOOGLE, 2023)

2.2.2 ChatGPT

O ChatGPT é um modelo de linguagem desenvolvido pela OpenAI que utiliza a tecnologia GPT-3.5, sendo um tipo de LLM para compreender e gerar texto em linguagem natural. É projetado para fornecer respostas contextualmente relevantes e pode ser aplicado em uma variedade de tarefas, incluindo a geração de código.(OPENAI, 2023)

Opera conforme o contexto fornecido e as instruções que recebe. Ou seja, pode ser alimentado com descrições de tarefas, pedidos de código ou qualquer outra informação relevante para a geração de código e correção de bugs.

Especificamente para a geração de código, o ChatGPT usa a compreensão contextual da linguagem de programação e os padrões observados no treinamento do modelo para criar *scripts*, funções ou instruções que correspondam à descrição da tarefa fornecida.(OPENAI, 2023)

2.2.2.1 Prós

- Aumento de produtividade: Pode acelerar o desenvolvimento, gerando código rapidamente com base em descrições de tarefas. O que economiza tempo e recursos para os desenvolvedores. (PENG, 2023)
- Acessibilidade: Ele torna a programação mais acessível para pessoas com níveis variados de habilidade técnica.(GARCIA, 2023)
- Criatividade: Pode sugerir diferentes soluções para o mesmo problema. Já que nunca é gerado a mesma resposta para o mesmo prompt.

2.2.2.2 Contras

- Qualidade variável: A qualidade do código gerado pelo ChatGPT pode variar.(OPENAI, 2023)
- Falta de conhecimento: O ChatGPT pode não entender a tarefa ou não ter detalhes importantes do projeto, levando a geração de código defeituoso.(OPENAI, 2023)
- Vulnerabilidades de Segurança: A utilização de código gerado automaticamente pode introduzir riscos de segurança se não for avaliado corretamente.(GARCIA, 2023)

Tabela 1 – Comparação Entre as Ferramentas Escolhidas - Fonte: Elaboração Própria.

Feature	**Gemini**	**ChatGPT**
Desenvolvedor	Google AI	OpenAI
Aplicação principal	Geração de texto em linguagem natural e código	Geração de texto em linguagem natural e código
Especialização	Programação	Nenhuma
Acesso	Livre	Livre

A Tabela 1 apresenta uma comparação das direfenças entre as duas ferramentas apresentadas.

Essas ferramentas foram desenvolvidas com o intuito de serem acessíveis e de fácil compreensão. Para atingir um nível de geração de conteúdo em geral levas-se em torno de 2 a 3 horas. Para tarefas mais complexas como geração e análise de código o tempo de aprendizagem pode aumentar para 1 a 2 dias dependendo do conhecimento prévio de programação.

2.3 TRABALHOS RELACIONADOS

Estudos têm explorado o potencial da IAG e suas aplicações e impactos no desenvolvimento de *software* e de jogos. Nesta seção, discutiremos alguns estudos, ressaltando sua importância para o tema em questão.

O artigo de Schatsky (2020) destaca que a IA desempenha um papel fundamental no aprimoramento de *software*. Profissionais estão utilizando a IA para auxiliar no design, desenvolvimento e implantação de *software*. A capacidade dessa tecnologia de analisar dados, identificar padrões e gerar soluções inovadoras tem sido uma vantagem significativa para os desenvolvedores. Essa abordagem permite a criação de softwares mais sofisticados.

No seu artigo Peng (2023) investiga o impacto da IAG na produtividade dos desenvolvedores por meio do *GitHub Copilot*. Esse estudo evidência como essa tecnologia pode auxiliar os desenvolvedores durante o processo de codificação, fornecendo sugestões de código e acelerando o desenvolvimento em 55.8%. Com a ajuda da IAG, os desenvolvedores podem economizar tempo e recursos, focando em tarefas mais criativas e complexas relacionadas ao design.

Em seu artigo Houde et al. (2022) exploram as oportunidades da IAG na modernização da experiência do usuário (UX). Ao utilizar técnicas de IAG, os desenvolvedores podem criar interfaces mais intuitivas. A capacidade da IAG em compreender os padrões e de automatizar o trabalho libera os desenvolvedores para focar no lado criativo do processo.

Em março de 2023 Wyatt Cheng, Diretor de Jogos na *Blizzard Entertainment* e Designer de Jogos no *Youtube*, lançou um vídeo explicando um experimento seu onde utilizou o ChaGPT para desenvolver um clone do jogo *Flappy Bird*. Em seu vídeo, Cheng comenta que a geração do código pela ferramenta deixa o desenvolvedor livre para focar em outros aspectos

como design de níveis e especificação de requisitos. A conclusão do experimento foi de que atualmente não é possível que uma pessoa sem conhecimento de programação crie um jogo apenas com ferramentas de IAG, porém para pessoas com conhecimento prévio o processo é simplificado e mais criativo. (CHENG, 2023)

Esses artigos apresentam diferentes perspectivas sobre o uso da IAG no desenvolvimento de software e jogos, destacando seus benefícios e desafios. Através da aplicação dessa tecnologia, os profissionais da área podem aprimorar a qualidade dos produtos, proporcionando experiências mais imersivas e personalizadas com tempo de desenvolvimento reduzido.

2.4 USO NA INDÚSTRIA

Como a IAG é considerada uma nova tecnologia ainda em estudo, poucas empresas, seja na indústria de jogos ou de desenvolvimento de software, implementam seu uso em seus processos e produtos. Porém, o seu uso vem aumentando como pode ser visto abaixo:

A *Ubisoft* é uma renomada empresa de jogos que empregou a IAG para criar multidões de personagens em seu jogo *"Assassin's Creed Unity"* em 2014. O uso da IAG nesse contexto permitiu que a empresa gerasse automaticamente uma grande quantidade de personagens para preencher as cidades do jogo, tornando o ambiente mais rico e dinâmico. (COURNOYER, 2022) Em 2023 lançou a ferramenta *Ubisoft Ghostwriter*, uma ferramenta de IAG que gera diálogos e sons para NPCs com o intuito diminuir a repetição de tarefas relacionas a criação de NPCs. (ENTERTAINMENT, 2023)

A *CD Projekt Red*, responsável pelo desenvolvimento de jogos como *"Cyberpunk 2077"*, utilizou a IA JALI para a animação de *lip-sync* em 10 idiomas diferentes para o referido jogo. O *lip-sync* envolve a sincronização dos movimentos da boca dos personagens com a fala correspondente. A IA JALI foi empregada para automatizar esse processo, economizando tempo e recursos. (EDWARDS; LANDRETH; POPŁAWSKI, 2020)

Na *COMPUTEX 2023* a empresa Nvidia anunciou o desenvolvimento do *NVIDIA Avatar Cloud Engine (ACE)*. Uma tecnologia que utiliza IAG para aprimorar NPCs em jogos. Esse serviço oferece modelos de IA personalizados, como o *NVIDIA NeMo* para linguagem, o *NVIDIA Riva* para reconhecimento de fala e geração de texto, e o*NVIDIA Omniverse Audio2Face* para animações faciais, permitindo que desenvolvedores melhorem as interações dos jogadores com NPCs em seus jogos. Desenvolvedores e startups como *GSC Game World* e *Fallen Leaf* já estão usando essas tecnologias para melhorar animações de personagens em seus jogos. (BURNES, 2023)

3 METODOLOGIA

Nesta seção, abordamos o processo de desenvolvimento de um protótipo de jogo de tiro 2D estilo *arcade* com o suporte de IAG. O objetivo central é explorar maneiras de incorporar essa tecnologia para automatizar partes do desenvolvimento de jogos. Para atingir esse objetivo, foram criados três protótipos na *engine Unity3D*.

Cada um desses protótipos foi desenvolvido de forma diferente, com os mesmos requisitos. Dois deles fizeram o uso de duas ferramentas de IAG distintas: Gemini e ChatGPT, que foram responsáveis pela geração do código e instruções gerais de como implementar as funcionalidades na *engine*. O terceiro protótipo foi criado sem qualquer apoio de IAG.

Os objetivos específicos deste estudo são:

1. Identificar o tempo de entrega necessário para cada protótipo.
2. Avaliar o tempo de ajuste, ou seja, o tempo que os desenvolvedores levam para adaptar o código gerado pela IAG ao projeto.
3. Analisar o desempenho dos protótipos.
4. Avaliar a qualidade do código em cada protótipo.

3.1 Os Protótipos

"Ships Invader" é um jogo de ação *arcade* onde o jogador controla uma nave espacial. O objetivo é destruir inimigos, evitar ser atingido por projéteis e acumular pontos. O jogo apresenta um fundo infinito em movimento e inimigos que se movem horizontalmente na tela gerados aleatoriamente. Abaixo exemplos de como foi prototipado as telas.

A Figura 2 mostra protótipo da tela de Inicio do jogo. Com o titulo no centro e logo abaixo botões para começar e sair do jogo. A Figura 3 representa a tela de *Game Over* seguindo o *design* que a tela de Inicio que aparece quando o jogador é atingindo repetidamente por projéteis ou naves inimigas.

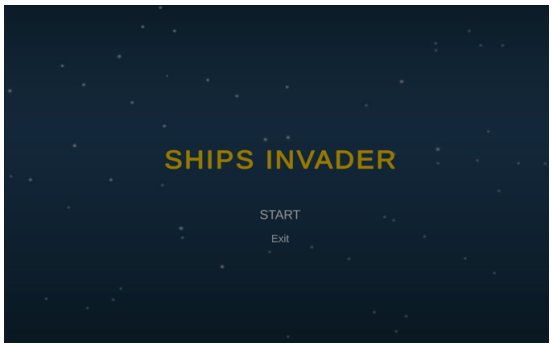

Figura 2 – Tela de Inicio - Fonte: Elaboração Própria.

Figura 3 – Tela de *Game Over* - Fonte: Elaboração Própria.

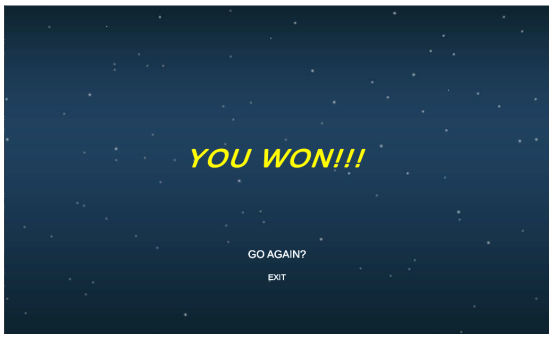

Figura 4 – Tela de *Win* - Fonte: Elaboração Própria.

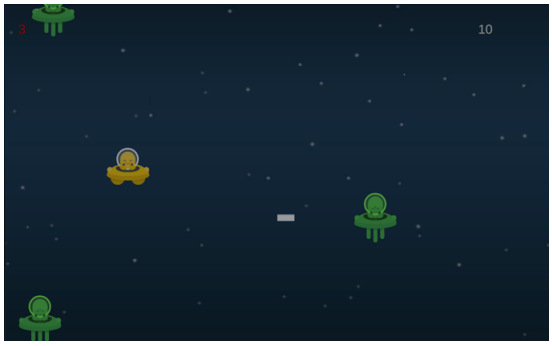

Figura 5 – Tela do jogo - Fonte: Elaboração Própria.

A Figura 4 representa de *Win* com o mesmo *design* das outras duas telas e aparece quando o jogador acumula 100 pontos. A Figura 5 mostra a tela do jogo em si, onde o jogador está representado pela nave amarela enquanto os inimigos pela nave verde. Inimigos se movem da direita para a esquerda, assim como o plano de fundo.

3.1.1 Definição dos Requisitos

O jogo contem os seguintes requisitos:

- R01 - O jogo deve conter um fundo infinito que se move.
- R02 - O jogador deve ser capaz de controlar a direção de uma nave espacial para a esquerda e direita, cima e baixo, utilizando as teclas direcionais "AWSD".
- R03 - O movimento do jogador deve ficar contido em um retângulo que representa metade da tela na esquerda.
- R04 - O jogador deve ser capaz de atirar projéteis que se movem para direita em direção aos inimigos utilizando o mouse ou a tecla "Espaço".
- R05 - O jogo deve apresentar inimigos que se movem para a esquerda na horizontal. Eles devem ser destruídos ao sair da tela.
- R06 -O jogo deve gerar inimigos fora da área visível a intervalos aleatórios e em posições aleatórias na vertical.
- R07 - O inimigo deve atirar projéteis a intervalos "aleatórios" na direção em que o jogador está se movendo.
- R08 - O jogador tem 3 chances de ser acertado por um projétil inimigo ou colidir com um inimigo antes receber uma tela de "*game over*", o número de chances deve ser mostrado na tela.
- R09: O jogador deve receber 5 pontos por inimigo atingido por projetil disparado pelo player, ou colisão do inimigo com player. Na colisão de projéteis os dois deve ser destruídos sem adicionar pontos. O número de pontos deve ser mostrado na tela.e atualizá-la à medida que o jogador acerta inimigos.
- R10 - O jogo deve conter uma tela inicial.
- R11 - O jogo deve conter uma tela de "*Game Over*".
- R12 - O jogo deve conter uma tela de "*Win*" que deve ser mostrada quando o jogador acumula 100 pontos.
- R13 - O jogo deve conter efeitos sonoros para tiros e impactos.
- R14 - O jogo deve conter música de fundo.

Todos os *assets* utilizados no jogo foram obtidos com a licença *creative commons*.

3.1.2 Utilização das ferramentas de IAG

Para implementar cada requisito nas ferramentas foi usada a seguinte metodologia:

- **Prompt Inicial:** Promtpt inicial explicando o que vai ser feito, como e a versão do software utilizado.
  Ex:
  Let's make a a 2d arcade game in Unity3d 2020.2.5. I will provide the requisites and follow the generated instructions on the engine. Can you provide instructions to implement them? .

- **Geração de código:** Descrição do requisito.
  Ex:
  `R02` - O jogador deve ser capaz de controlar a direção da nave espacial para a esquerda e direita, cima e baixo, utilizando as teclas direcionais "AWSD".

  Prompt: *I want to create the movement of a 2D spaceship sprite using the AWSD or the keyboard arrows in Unity3D. How do I do this?*

  Erros no console: Descrição do erro. Se necessário, pedaços do código já gerado devem ser adicionados ao prompt. Pergunta de como resolver.
  Ex:
  `Error:` *MissingComponentException: There is no 'Rigidbody2D' attached to the "Player"game object, but a script is trying to access it. You probably need to add a Rigidbody2D to the game object "Player". Or your script needs to check if the component is attached before using it.UnityEngine.Rigidbody2D.get_velocity()(at <54833848250c4d208c4b96b077464260>:0) Player.Update ()*

  Prompt: *I got MissingComponentException: There is no 'Rigidbody2D' attached to the "Player"game object, but a script is trying to access it. You probably need to add a Rigidbody2D to the game object "Player". Or your script needs to check if the component is attached before using it.UnityEngine.Rigidbody2D.get_velocity()(at <54833848250c4d208c4b96b077464260>:0) Player.Update (). How do I fix it?*

- **Comportamentos inesperados:** Descrição do comportamento esperado, descrição do comportamento inesperado ocorrendo. Se possível, exemplo do código de onde vem o comportamento. Pergunta de como corrigir o comportamento.
  Ex:
  Comportamento Esperado: Objeto *player* se move para a direita quando "D"/"Seta direita"é pressionada e para a esquerda quando "A"/"Seta esquerda".

  Comportamento Inesperado Ocorrendo: Objeto *player* se move para a esquerda quando "D"/"Seta direita"é pressionada e para a direita quando "A"/"Seta esquerda".

  Prompt: *I want the player object to go right when I press "D"or "right arrow"and left when I press "A"or "left arrow", but the player object is going left when I press "D"or "right arrow"and right when I press "A"or "left arrow."The player code is: [Colar código da classe PlayerMovement]. How do I fix this?*

3.1.3 Utilização das respostas geradas

A utilização das respostas geradas pelas ferramentas serão implementadas da seguinte maneira:

- **Geração inicial de código:** As respostas geradas foram usadas para a implementação dos requisitos. O código gerado foi adaptado ao projeto e a códigos já gerados pela ferramenta. Ao ocorrer erros ou comportamentos inesperados, novos *prompts* foram utilizados.
- **Erros no console:** As respostas geradas para resolver erros ou problemas no código foram aplicadas para solucionar problemas de programação relacionados a erros no console.
- **Comportamentos inesperados:** As respostas geradas para corrigir comportamentos inesperados foram implementadas para garantir que o jogo atenda aos requisitos estabelecidos no caso do código não apontar erros, mas não funcione da forma esperada.

3.2 Análise dos resultados

Para avaliar o uso das ferramentas IAG cada protótipo foi avaliado pelas seguintes métricas: tempo de entrega, tempo de ajuste, desempenho e qualidade do código. Comparar essas análises permitiu determinar o impacto da IAG no desenvolvimento na automação de tarefas de programação.

3.2.1 Tempo de Entrega

O tempo de entrega avalia o tempo necessário para concluir cada protótipo com e sem o auxílio das ferramentas de IAG. Isso permite a comparação dessas ferramentas na aceleração do desenvolvimento do jogo.

3.2.2 Tempo de Ajuste

O tempo de ajuste representa o tempo que um desenvolvedor leva para implementar as sugestões geradas ou criar o código a mão e consultar documentação quando necessário para cada requisito. Quando se utiliza a IAG é importante medir esse aspecto, uma vez que a automação da criação de código pode acelerar significativamente o processo de desenvolvimento, mas também requer modificações e ajustes para se alinhar perfeitamente aos requisitos específicos do jogo e da *engine* sendo usada. Esse tempo avalia se o uso da IAG resulta em economia de tempo e recursos no processo de desenvolvimento. O que ajuda a determinar se a automação da geração de código por IAG é vantajosa, mesmo se considerar o tempo necessário para personalizar o código gerado. Essa métrica ajuda a entender a eficiência da IAG em produzir código próximo ao desejado e se supera a escrita manual de código em termos de economia de tempo e esforço.

3.2.2.1 Comparação Entre Protótipos

Como foram feitos dois protótipos com IAG e um sem seu uso, é importante especificar como isso foi medido em cada situação.

- **Protótipos com IAG:** Nos protótipos que fazem uso da ferramenta, o tempo e de ajuste é o tempo que demora para gerar, modificar, personalizar o código gerado e corrigir quaisquer inconsistências para se alinhar ao projeto. O tempo de juste mede a dificuldade na criação de código inicial e o esforço necessário para adaptá-lo.

- **Protótipo sem IAG:** No protótipo desenvolvido sem auxílio de IAG, o código e implementação são desenvolvidos manualmente para atender aos requisitos utilizando conhecimentos prévios, documentação e fóruns. Nesse caso, a dificuldade de ajuste representa o tempo necessário para encontrar as informações necessárias para corrigir erros e comportamentos inesperados. Pode ser mais curto em comparação com os protótipos com IA, uma vez que o código é desenvolvido manualmente conforme as necessidades do projeto.

3.2.3 Desempenho

As métricas taxa de quadros por segundo (FPS - *Frames Per Second*), o uso da memória RAM e o uso da CPU foram escolhidas para medir o desempenho dos protótipos.

- **Taxa de Quadros por Segundo:** A FPS mede a quantidade de quadros de animação que o jogo exibe por segundo. Para uma experiência de jogo suave. Uma taxa recomendável de FPS é entre 30 e 60 FPS.
- **Uso de Memória RAM:** A quantidade de memória RAM utilizada pelo jogo é relevante para garantir que o jogo seja eficiente em termos de recursos. Excesso de uso de RAM pode levar a problemas de desempenho e baixo FPS. Portanto, monitorar o uso de memória é crítico para identificar vazamentos de memória ou alocações excessivas.
- **Uso da CPU:** O uso da CPU está diretamente relacionado ao desempenho do jogo. É importante garantir que o jogo não sobrecarregue a CPU, pois isso pode resultar em travamentos e impactar negativamente a jogabilidade. A análise do uso da CPU ajuda a identificar gargalos de desempenho no código do jogo.

3.2.4 Qualidade do Código

Para avaliar a qualidade do código dos protótipos, foi optado utilizar critérios baseados no livro *"Clean Code"*(MARTIN, 2009). Os critérios selecionados são:

- **Identificadores Descritivos**: uso de identificadores para variáveis, funções e classes que sejam descritivos e revelem sua intenção.
- **Função única**: funções pequenas e focadas em uma única tarefa.
- **Código sem repetição (DRY - Don't Repeat Yourself)**: evitar a duplicação de código.
- **Alta Coesão**: agrupamento de funcionalidades relacionadas em classes coesas, porém com uma única funcionalidade.
- **Redundância**: existência de lógica redundante ou desnecessária.

A razão para a escolha desses critérios está relacionada ao escopo dos protótipos. Devido à baixa complexidade em comparação com sistemas mais complexos, foi decidido focar em critérios que são especialmente relevantes para manter o código claro, simples e de fácil manutenção. Esses critérios ajudarão a garantir que o código seja legível, modular e coeso, o que é fundamental para o desenvolvimento de jogos.

Para contabilizar a qualidade do código, cada protótipo recebeu uma pontuação de 0 a 10 para cada métrica relativa a sua adesão a cada um dos critérios. Após foi reunido todos os pontos e feita uma média geral de cada protótipo.

3.3 Cronograma

O desenvolvimento dos protótipos, análise, escrita e revisão do TCC 2 foram previstos para durar 15 semanas, começando em 26/03/2024 e terminando em 03/06/2024, seguindo o

cronograma da Tabela 2:

Tabela 2 – Cronograma - Fonte: Elaboração Própria.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Atividade 1															
Atividade 2															
Atividade 3															
Atividade 4															
Atividade 5															
Atividade 6															
Atividade 7															
Atividade 8															
Atividade 9															
Atividade 10															
Atividade 11															
Atividade 12															
Atividade 13															
Atividade 14															
Atividade 15															
Atividade 16															

- **Atividade 1:** Obtenção de *Assets*
- **Atividade 2:** Protótipo 1 - Protótipo sem IAG
- **Atividade 3:** Protótipo 2 - Gemini
- **Atividade 4:** Protótipo 3 - ChatGPT
- **Atividade 5:** Atualizar Introdução
- **Atividade 6:** Atualizar Analise Tecnologia
- **Atividade 7:** Atualizar Metodologia - Requisitos
- **Atividade 8:** Reescrever Metodologia - Ferramentas
- **Atividade 9:** Reescrever Metodologia - Analise de Resultados e Métricas
- **Atividade 10:** Analise Protótipo 1
- **Atividade 11:** Analise Protótipo 2
- **Atividade 12:** Analise Protótipo 3
- **Atividade 13:** Analise comparativa
- **Atividade 14:** Escrever Análise Resultados
- **Atividade 15:** Escrever Conclusão
- **Atividade 16:** Revisão

4 IMPLEMENTAÇÃO

A implementação dos protótipos foi feita requisito a requisito conforme apresentado no Capítulo 3. Apresentamos o desenvolvimento de cada um dos protótipos. Para os três protótipos foi inicialmente criado um projeto 2d na Unity3d e importados os *assets* necessários para o jogo (*sprites* e arquivos de áudio).

4.1 Sem IAG

Durante o desenvolvimento do protótipo sem o uso de ferramentas de IAG foi utilizada a documentação oficial da *Unity*(UNITY, 2020), pesquisa de dúvidas na internet, incluindo fóruns de desenvolvimento e vídeos tutoriais.

Na organização do projeto as classes foram separadas por funcionalidade geral, seguindo as boas práticas de programação mencionadas no capítulo 3. Essa abordagem modular facilitou a leitura, a manutenção e o *debugging* do código, além de permitir um melhor controle sobre as diferentes partes do jogo.

4.1.1 Prefabs

A implementação foi voltada a utilização dos prefabs da Unity. Prefabs são objetos compostos por componentes, valores, propriedade e scripts salvos em um modelo no projeto. Prefabs podem ser usados para facilitar o desenvolvimento e para a rápida criação de objetos durante a execução do jogo.(UNITY, 2020). Naves inimigas, projéteis, telas de interação (*Game Over*, Inicio, *Win*) foram criados com prefabs tanto para agilizar o tempo de desenvolvimento quanto para simplificar a lógica do código. Um exemplo desse uso foram os projéteis onde o código para o movimento do projetil inimigo e do projetil do jogador é o mesmo. A mudança de direção acontece apenas pelo mudança de sinal da variável responsável.

Listing 4.1 – Exemplo código classe Projectile

```
public class Projectile : MonoBehaviour
{
    public float movingSpeed = 5;
    public float movingDirection = 1;

    // Update is called once per frame
    void Update()
    {
        MoveProjectile();
    }

    private void MoveProjectile()
    {
        transform.Translate(Vector3.left * Time.deltaTime *
           movingSpeed * movingDirection);
    }
}
```

4.1.2 Resultados

O processo contando o tempo de desenvolvimento de cada requisito, configurações de *build* e possíveis correções foi de 02h:55m:44s. Conforme a Tabela 3 os requisitos que mais levaram tempo para serem implementados foram os que envolveram a movimentação de objetos, elementos de largura e altura da tela e colisões de múltiplos objetos.

Requisito	**Tempo**
R01	00:31:05
R02	00:09:49
R03	00:21:52
R04	00:15:07
R05	00:05:57
R06	00:07:58
R07	00:10:17
R08	00:22:34
R09	00:08:54
R10	00:08:14
R11	00:04:56
R12	00:08:36
R13	00:08:49
R14	00:01:43
Build	00:09:53
Total	02:55:44

Tabela 3 – Tempo de desenvolvimento por requisito sem IAG - Fonte: Elaboração Própria.

Abaixo imagens do protótipo finalizado Sem IAG. A Figura 6 mostra a tela de Início do jogo. Com o titulo no centro e logo abaixo botões para começar e sair do jogo. A Figura 7 representa a tela de *Game Over* que aparece quando o jogador é atingindo repetidamente por projéteis ou naves inimigas, segue o mesmo desing que a tela de Inicio.

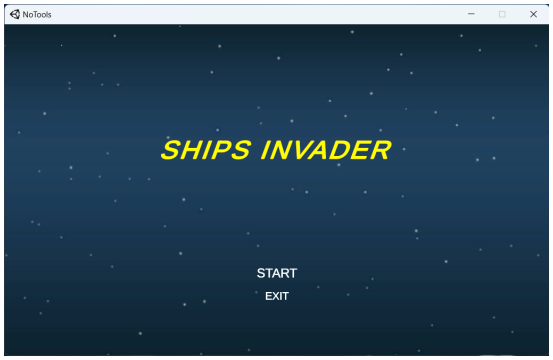

Figura 6 – Tela de Inicio - Sem IAG - Fonte: Elaboração Própria.

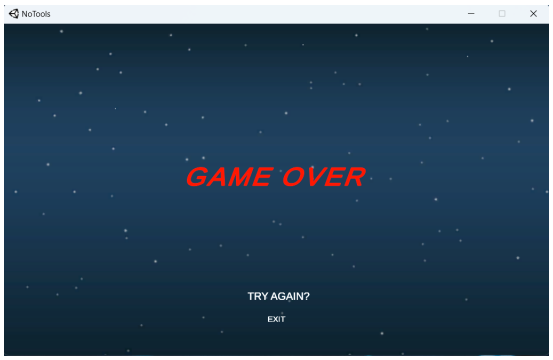

Figura 7 – Tela de *Game Over* - Sem IAG - Fonte: Elaboração Própria.

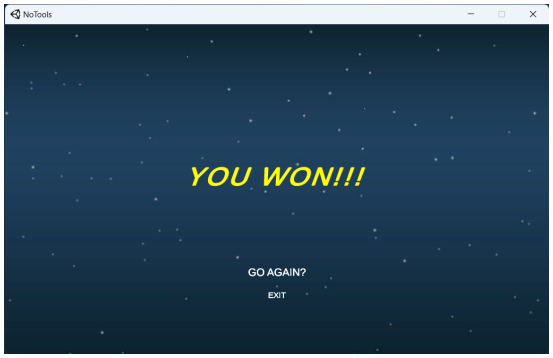

Figura 8 – Tela de *Win* - Sem IAG - Fonte: Elaboração Própria.

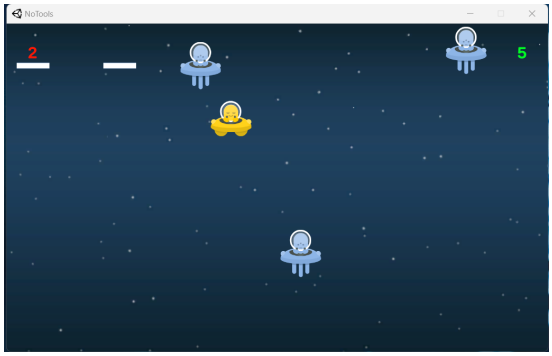

Figura 9 – Tela do jogo - Sem IAG - Fonte: Elaboração Própria.

A Figura 8 representa de *Win* com o mesmo *design* das outras duas telas e aparece quando o jogador acumula 100 pontos. A Figura 9 mostra a tela do jogo em si, onde o jogador está representado pela nave amarela enquanto os inimigos pela nave verde. Inimigos se movem da direita para a esquerda, assim como o plano de fundo.

4.2 Utilizando Ferramentas

Durante o desenvolvimento dos protótipos onde foram utilizadas ferramentas de IAG primeiro foi dado um *prompt* informando a ferramenta o que ia ser feito:

Let's make a 2d arcade game in unity3d 2020.2.5. I will provide the requisites and follow the generated instructions on the engine. Can you provide instructions to implement them? (Don't start now, just confirm if you can or can't)

Vamos fazer um jogo 2d arcade na unity3d 2020.2.5. Eu vou dar os requisitos e seguir as instruções geradas dentro da engine. Consegue gerar as instruções para implementá-los?(Não comece agora, apenas confirme se consegue ou não)

Após a confirmação foi enviado requisito por requisito e seguidas as instruções geradas pela ferramenta. No caso das instruções serem insuficientes ou causarem erros, ou comportamentos estranhos, foi discutido via *prompts* com a ferramenta como criar o resultado desejado.

Para marcar o fim de um requisito e seguir para o próximo, foi adicionada uma confirmação de que o requisito anterior tinha sido completo e a descrição do próximo a ser implementado.

4.2.1 *Prompts* iniciais para cada requisito

Abaixo estão os requisitos utilizados para iniciar o projeto e também avançar para o próximo requisito.

- *Let's make a 2d arcade game in unity3d 2020.2.5. I will provide the requisites and follow the generated instructions on the engine. Can you provide instructions to implement them? (Don't start now, just confirm if you can or can't)*
- *Ok, here is a requeriment R01 - The game must feature an infinite scrolling background.*
- *Ok this works. Now make R02 - The player must be able to control the direction of a spaceship left and right, up and down, using the directional keys "AWSD".*
- *Ok, that is working. Next R03 - The player's movement must be confined to a rectangle representing half of the screen in the left side*

- *That works. Ok, now R04 - The player must be able to shoot projectiles that move right towards enemies using the mouse or the "Space"key.*
- *OK, this is working. Next R05 - The game must feature enemies moving towards the left. They should be destroyed on leaving the camera view (only on the left side)*
- *Ok, this is working. Now R06 The game must generate enemies outside the visible area at random intervals and positions vertically.*
- *Ok, this works. Now R07 - Enemies must shoot projectiles at random intervals in the direction the player is moving.*
- *Ok now R08 - The player has 3 chances to be hit by an enemy projectile or collide with an enemy before receiving a "game over"screen, and the number of chances must be displayed on the screen.*
- *Ok, this works. Now R09 - The player must receive 5 points for each enemy hit by a projectile fired by the player or collision with the player. The number of points must be displayed on the screen and updated as the player hits enemies. Destroy enemies on any type of collision*
- *Ok, this works. Now R10 - The game must contain an initial screen.*
- *Ok now R11 - The game must contain a "Game Over"Scene*
- *Ok, now R12 - The game must contain a "win"screen that should be shown when the player accumulates 100 points.*
- *Now R13 - The game must feature sound effects for shots and impacts. Everytime a projectile is initiated there should be a sound. Every impact should be a sound before the object is destroyed*
- *Ok now R14 - The game must include background music.*

4.2.2 Gemini

No desenvolvimento com a ferramenta Gemini houve problemas com a implementação do requisito R01 onde mesmo depois de várias explicações a ferramenta continuou gerando a movimentação do fundo com comportamentos inesperados, utilizando métodos que não foram possíveis de implementar devido a instruções para utilizar funcionalidades não existentes dentro da *engine*.

Pensando também em próximos requisitos que a ferramenta ainda não tinha conhecimento, foi pedido para alterar alguns pontos do desenvolvimento. Como mudar o método de movimentação de projéteis para se adequar a ter projéteis que se movem em direções diferentes dependendo do tipo de prefab.

4.2.2.1 Requisito R01

Foram necessárias várias descrições da mesma funcionalidade para a ferramenta gerar instruções e código correto, levando um tempo maior do que todos os outros requisitos em todos os protótipos para implementar.

A primeira parte da instrução pedia para fazer a movimentação do fundo baseada na movimentação da textura do material aplicado a um Quad, um objeto 3d que consiste de quatro vértices de uma unidade de tamanho (UNITY, 2020). Contudo, nas opções dentro da *engine* não foi possível encontrar a propriedade que a Gemini estava indicando.

Pedindo uma nova abordagem, a ferramenta trouxe instruções para criar o movimento usando o movimento de textura utilizando *sprites*. Mas trouxe incompatibilidades de objetos e problemas no código que criaram um código que não reproduzia o comportamento desejado.

Após isso foi explicado novamente o requisito:

Lets start this again. What I need is an infinite 2d scrolling background that moves from right to left. No manipulating materials or using quads.

Vamos começar novamente. Eu preciso de um fundo infinito em 2d que se move de direita para a esquerda.

Após esse *prompt*, a ferramenta trouxe instruções e código mais próximo do desejado. Dividiu o movimento do fundo em dois objetos que andam juntos em fila, quando um não é mais visível na tela ele é transportado para a parte final da fila, criando a impressão de movimento infinito. Mas apenas um dos objetos estava se movimentando como desejado.

Foi preciso mais dois prompts de descrição e alguns prompts de erro para a ferramenta gerar o comportamento desejado:

Look, this should work like this: Both backgrounds should move to the left until the one on the front leaves the screen then this background should go back to the second background initial position with now makes a loop.

Olha, deve funcionar assim: Os dois fundos devem se mover para a esquerda até que o da frente saia da tela, então esse fundo deve voltar para a posição inicial do segundo fundo, o que agora cria um loop.

I will explain AGAIN: There are two backgrounds B1 and B2. B1 and B2 Should move at the same speed to the left. When a background leaves the screen (meaning outside the camera view to the left) this background should go back to B2 initial position.

Vou explicar de NOVO: Existem dois fundos, B1 e B2. B1 e B2 devem se mover à mesma velocidade para a esquerda. Quando um fundo sair da tela (ou seja, fora da vista da câmera para a esquerda), esse fundo deve voltar para a posição inicial de B2.

4.2.2.2 Movimentação de Projéteis

Como mencionado no desenvolvimento sem ferramentas, foi utilizado bastante prefabs. Durante do desenvolvimento, a desenvolvedora já sabia que existiriam dois tipos de prefabs que se movem em direções opostas. Então após dar o *prompt* inicial do R04 e o código para a movimentação do projetil do jogador ser gerado. Foi pedido a ferramenta que o alterasse para a movimentação depender de um sinal, assim como foi feita na versão sem ferramentas.

Listing 4.2 – Código classe Projectile Inicial

```
public class Projectile : MonoBehaviour
{
    public float projectileSpeed = 10f; // Adjust speed for
       projectile movement

    private Rigidbody2D projectileRigidbody; // Reference to the
       projectile's Rigidbody2D component

    void Start()
    {
        // Get the projectile's Rigidbody2D component
        projectileRigidbody = GetComponent<Rigidbody2D>();

        // Apply initial force to the projectile to move it right
        projectileRigidbody.AddForce(Vector2.right * projectileSpeed,
           ForceMode2D.Impulse);
```

```
    }
}
```

Listing 4.3 – Pedaço código Projectile Final

```
public class Projectile : MonoBehaviour
{
    public float projectileSpeed = 10f; // Adjust speed for
       projectile movement

    private Rigidbody2D projectileRigidbody; // Reference to the
       projectile's Rigidbody2D component

    void Start()
    {
        // Get the projectile's Rigidbody2D component
        projectileRigidbody = GetComponent<Rigidbody2D>();

        // Set initial velocity based on projectileSpeed sign
        projectileRigidbody.velocity = Vector2.right *
           projectileSpeed;
    }
}
```

4.2.2.3 Resultados

O processo contando o tempo de desenvolvimento de cada requisito, configurações de *build* e possíveis correções foi de 02h:32m:47s. Conforme a Tabela 4 os requisitos que mais levaram tempo para serem implementados foram os de movimentação do fundo, que levou mais da metade do tempo total, e a movimentação dos projéteis.

Requisito	Tempo
R01	01:09:52
R02	00:03:38
R03	00:04:18
R04	00:15:49
R05	00:04:18
R06	00:02:25
R07	00:04:15
R08	00:08:38
R09	00:08:00
R10	00:02:36
R11	00:06:46
R12	00:04:30
R13	00:10:55
R14	00:00:58
Build	00:02:08
Correção	00:03:38
Total	02:32:47

Tabela 4 – Tempo de desenvolvimento por requisito utilizando Gemini - Fonte: Elaboração Própria.

Abaixo imagens do protótipo Gemini finalizado. A Figura 10 mostra a tela de Início do jogo. Com o titulo no centro e logo abaixo botões para começar e sair do jogo. A Figura 11 representa a tela de *Game Over* que aparece quando o jogador é atingindo repetidamente por projéteis ou naves inimigas, segue o mesmo desing que a tela de Início.

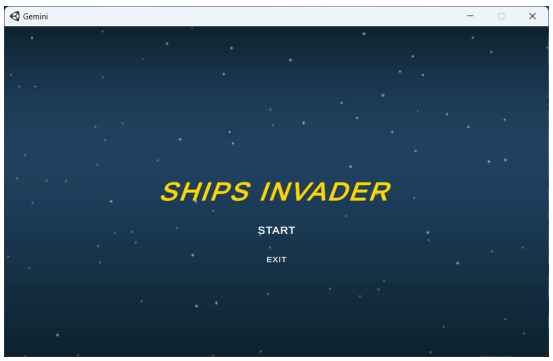

Figura 10 – Tela de Inicio - Gemini - Fonte: Elaboração Própria.

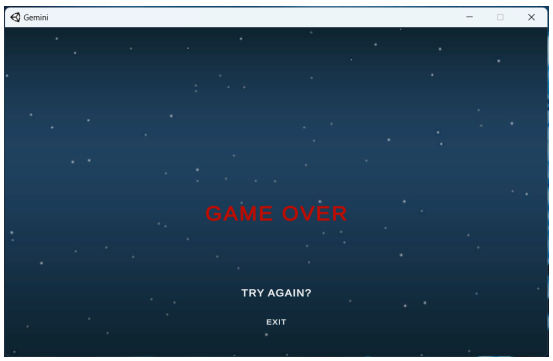

Figura 11 – Tela de *Game Over* - Gemini - Fonte: Elaboração Própria.

A Figura 13 representa de *Win* com o mesmo *design* das outras duas telas e aparece quando o jogador acumula 100 pontos. A Figura 14 mostra a tela do jogo em si, onde o jogador está representado pela nave amarela enquanto os inimigos pela nave verde. Inimigos se movem da direita para a esquerda, assim como o plano de fundo.

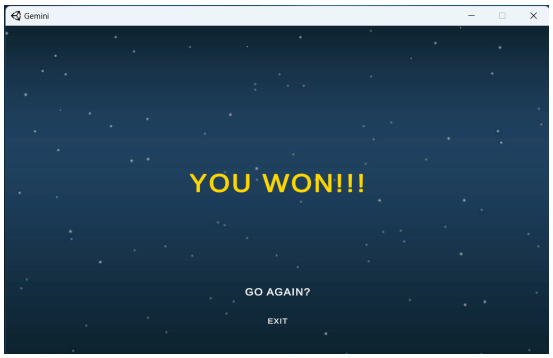

Figura 12 – Tela de *Win* - Gemini - Fonte: Elaboração Própria.

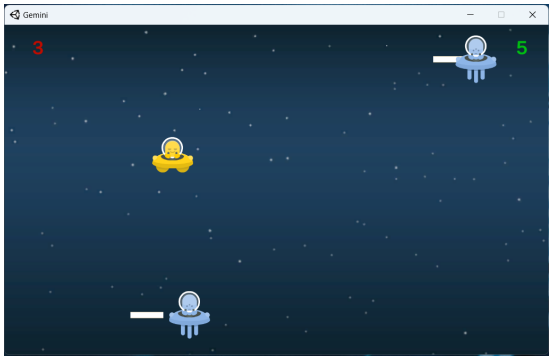

Figura 13 – Tela do jogo - Gemini - Fonte: Elaboração Própria.

4.2.3 ChatGPT

No desenvolvimento utilizando ChatGPT em comparação com o desenvolvimento com a ferramente Gemini foi preciso de menos instruções para chegar ao resultado. Em sua maioria uma ou duas, tirando instruções de modificações decididas na hora para melhorar jogabilidade. Enquanto a Gemini levou entre duas e três explicações do requisito antes de apresentar instruções e código com a funcionalidade correta.

Ocorreu de um requisito(R07) sobrescrever outro que já estava funcional(R05).

4.2.3.1 Sobreposição de requisitos

Quando foi dado o *prompt* para o requisito R07 a ChatGPT utilizou o mesmo script gerado para o requisito R05 e apagou o código responsável pelo movimento do inimigo.

Abaixo estão os códigos gerados. O primeiro código (4.4) controla o movimento básico do inimigo, movendo para a esquerda com uma velocidade aleatória e destruindo quando sai da tela. O segundo código (4.5) cria a funcionalidade de disparo, fazendo com que o inimigo atire projéteis em direção ao jogador em intervalos aleatórios. Porém, exclui a funcionalidade de 4.4. O código (4.6) combina ambas as funcionalidades.

Listing 4.4 – Classe EnemyMovement para requisito R05

```
using UnityEngine;

public class EnemyMovement : MonoBehaviour
{
    private float speed; // Enemy speed
    private bool isMovingLeft = true; // Direction of movement

    void Start()
    {
        // Generate a random speed between 3 and 5
        speed = Random.Range(3f, 5f);
    }

    void Update()
    {
```

```
        // Move the enemy
        Move();

        // Check if the enemy has left the camera view on the left
           side
        if (IsOutsideCameraView() && isMovingLeft)
        {
            Destroy(gameObject);
        }
    }

    void Move()
    {
        // Move the enemy towards the left
        transform.Translate(Vector3.left * speed * Time.deltaTime);
    }

    bool IsOutsideCameraView()
    {
        Vector3 screenPos =
           Camera.main.WorldToScreenPoint(transform.position);
        return screenPos.x < 0;
    }
}
```

Listing 4.5 – Classe EnemyMovement para requisito R07

```
using System.Collections;
using UnityEngine;

public class EnemyMovement : MonoBehaviour
{
    public GameObject enemyProjectilePrefab; // Reference to the
       enemy projectile prefab
    public float minShootInterval = 2f; // Minimum interval between
       shots
    public float maxShootInterval = 5f; // Maximum interval between
       shots
    public float projectileSpeed = 8f; // Speed of the enemy
       projectiles

    private GameObject player; // Reference to the player GameObject
    private Vector3 playerLastPosition; // Last known position of the
       player
    private bool isShooting = false; // Flag to control shooting
       coroutine

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
```

```
        playerLastPosition = player.transform.position;

        // Start shooting coroutine
        StartCoroutine(ShootProjectiles());
    }

    IEnumerator ShootProjectiles()
    {
        while (true)
        {
            // Calculate interval between shots
            float shootInterval = Random.Range(minShootInterval,
               maxShootInterval);
            yield return new WaitForSeconds(shootInterval);

            // Determine direction to shoot
            Vector3 direction = (player.transform.position -
               transform.position).normalized;

            // Instantiate enemy projectile
            GameObject projectile =
               Instantiate(enemyProjectilePrefab, transform.position,
               Quaternion.identity);
            Rigidbody2D rb = projectile.GetComponent<Rigidbody2D>();
            rb.velocity = direction * projectileSpeed;
        }
    }

    void Update()
    {
        // Update the last known position of the player
        playerLastPosition = player.transform.position;
    }
}
```

Listing 4.6 – Classe EnemyMovement combinado

```
using System.Collections;
using UnityEngine;

public class EnemyMovement : MonoBehaviour
{
    public GameObject enemyProjectilePrefab; // Reference to the
       enemy projectile prefab
    public float minShootInterval = 2f; // Minimum interval between
       shots
    public float maxShootInterval = 5f; // Maximum interval between
       shots
    public float minEnemySpeed = 3f; // Minimum enemy speed
    public float maxEnemySpeed = 5f; // Maximum enemy speed
```

```
public float minProjectileSpeedOffset = 2f; // Minimum offset for
   projectile speed relative to enemy speed
public float maxProjectileSpeedOffset = 4f; // Maximum offset for
   projectile speed relative to enemy speed

private GameObject player; // Reference to the player GameObject
private float enemySpeed; // Speed of the enemy

void Start()
{
    player = GameObject.FindGameObjectWithTag("Player");

    // Set random enemy speed between minEnemySpeed and
       maxEnemySpeed
    enemySpeed = Random.Range(minEnemySpeed, maxEnemySpeed);

    // Start shooting coroutine
    StartCoroutine(ShootProjectiles());
}

IEnumerator ShootProjectiles()
{
    while (true)
    {
        // Calculate interval between shots
        float shootInterval = Random.Range(minShootInterval,
           maxShootInterval);
        yield return new WaitForSeconds(shootInterval);

        // Determine direction to shoot (always to the left)
        Vector3 direction = Vector3.left;

        // Calculate projectile speed offset relative to enemy
           speed
        float projectileSpeedOffset =
           Random.Range(minProjectileSpeedOffset,
           maxProjectileSpeedOffset);

        // Set projectile speed to be higher than
        enemy speed
        float projectileSpeed = enemySpeed +
           projectileSpeedOffset;

        // Instantiate enemy projectile
        GameObject projectile =
           Instantiate(enemyProjectilePrefab, transform.position,
           Quaternion.identity);
        Rigidbody2D rb = projectile.GetComponent<Rigidbody2D>();
        rb.velocity = direction * projectileSpeed;
```

```
        }
    }

    void Update()
    {
        // Move the enemy towards the left
        transform.Translate(Vector3.left * enemySpeed *
           Time.deltaTime);

        // Check if the enemy has left the camera
        view on the left side
        if (IsOutsideCameraView())
        {
            Destroy(gameObject);
        }
    }

    bool IsOutsideCameraView()
    {
        Vector3 screenPos =
           Camera.main.WorldToScreenPoint(transform.position);
        return screenPos.x < 0;
    }
}
```

4.2.3.2 Resultados

O processo contando o tempo de desenvolvimento de cada requisito, configurações de *build* e possíveis correções foi de 01h:40m:45s. Conforme a Tabela 5 os requisitos que mais levaram tempo para serem implementados foram o de movimentação do fundo, colisões e efeitos sonoros.

Requisito	**Tempo**
R01	00:10:25
R02	00:03:53
R03	00:04:22
R04	00:08:54
R05	00:03:50
R06	00:05:36
R07	00:06:59
R08	00:10:40
R09	00:08:25
R10	00:07:16
R11	00:05:19
R12	00:03:15
R13	00:16:02
R14	00:02:54
Build	00:01:57
Correção	00:00:58
Total	01:40:45

Tabela 5 – Tempo de desenvolvimento por requisito ChatGPT - Fonte: Elaboração Própria.

Abaixo imagens do protótipo ChatGPT finalizado. A Figura 14 mostra a tela de Início do jogo. Com o titulo no centro e logo abaixo botões para começar e sair do jogo. A Figura 16 representa a tela de *Game Over* que aparece quando o jogador é atingindo repetidamente por projéteis ou naves inimigas, segue o mesmo desing que a tela de Início.

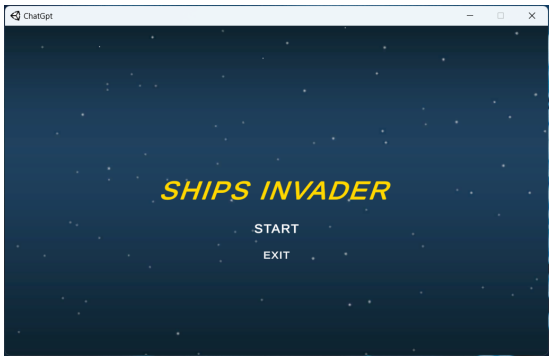

Figura 14 – Tela de Inicio - ChatGPT - Fonte: Elaboração Própria.

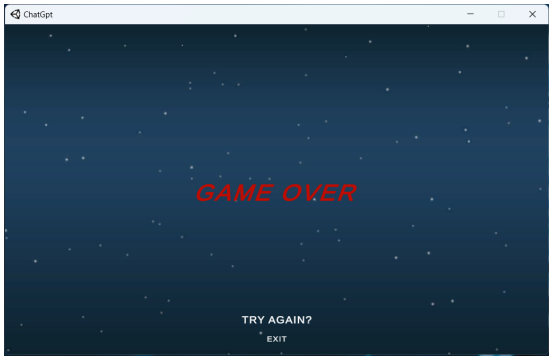

Figura 15 – Tela de *Game Over* - ChatGPT - Fonte: Elaboração Própria.

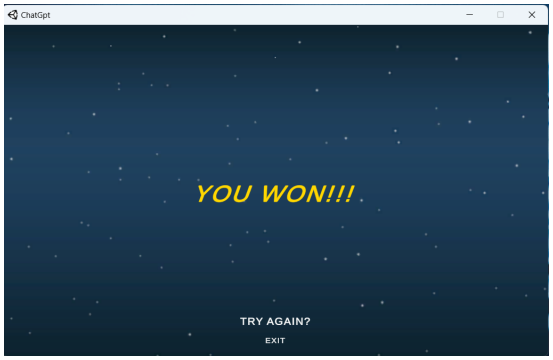

Figura 16 – Tela de *Win* - ChatGPT - Fonte: Elaboração Própria.

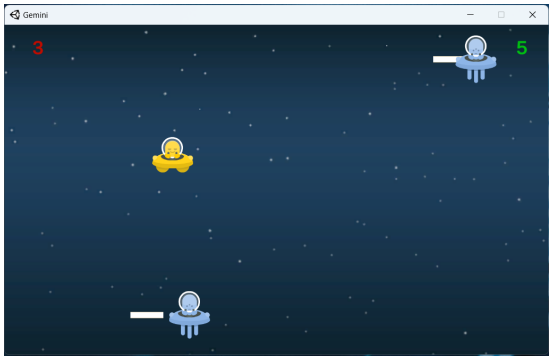

Figura 17 – Tela do jogo - ChatGPT - Fonte: Elaboração Própria.

A Figura 16 representa de *Win* com o mesmo *design* das outras duas telas e aparece quando o jogador acumula 100 pontos. A Figura 17 mostra a tela do jogo em si onde o jogador está representado pela nave amarela enquanto os inimigos pela nave verde. Inimigos se movem da direita para a esquerda, assim como o plano de fundo.

5 ANÁLISE DOS RESULTADOS

Abaixo está apresentada a análise dos resultados considerando os aspectos descritos no Capítulo 3

5.0.1 Tempo de Entrega e Tempo de Ajuste

O tempo total de entrega dos protótipos foi calculado somando o tempo necessário para concluir cada um dos requisitos (R01 a R14) de cada protótipo, mais o tempo de configuração de *build* e possíveis correções posteriores ao desenvolvimento. Como pode ser observado na Tabela 6, o protótipo utilizando ChatGPT teve o menor tempo total de entrega, com um total de 1 hora, 40 minutos e 45 segundos. Em seguida, o protótipo utilizando Gemini que teve um tempo total de entrega de 2 horas, 32 minutos e 47 segundos, enquanto o protótipo Sem IAG levou um total de 2 horas, 55 minutos e 44 segundos para ser entregue.

Tabela 6 – Tempo de Entrega Total dos Protótipos - Fonte: Elaboração Própria.

	Sem IAG	Gemini	ChatGPT
Tempo Total	02h 55min 44s	02h 32min 47s	01h 40min 45s

Sobre o tempo de ajuste foi calculado requisito por requisito, exemplificado na Tabela 7. Para o protótipo Sem IAG o tempo de ajuste foi inclui o tempo de codificação, tempo de pesquisa de implementação e correções de erros. Para os protótipos com o uso de ferramentas de IAG o tempo de ajuste inclui o tempo de codificação e manipulação da ferramenta para implementar e corrigir erros no código.

Em sua maioria, o tempo de cada requisito foi menor nos protótipos utilizando ferramentas de IAG. Contudo, é importante ressaltar que em relação aos protótipos que utilizaram de ferramenta, o protótipo criado usando ChatGPT levou menos tempo na maioria dos requisitos. Também um ponto a ser mencionado é o tempo despendido no requisito R01 no protótipo feito com Gemini levou um tempo consideravelmente maior se comparado aos outros dois protótipos.

Tabela 7 – Comparação do Tempo de desenvolvimento por requisito dos protótipos - Fonte: Elaboração Própria.

Requisito	**Sem IAG**	**Gemini**	**ChatGPT**
R01	00:31:05	01:09:52	00:10:25
R02	00:09:49	00:03:38	00:03:53
R03	00:21:52	00:04:18	00:04:22
R04	00:15:07	00:15:49	00:08:54
R05	00:05:57	00:04:18	00:03:50
R06	00:07:58	00:02:25	00:05:36
R07	00:10:17	00:04:15	00:06:59
R08	00:22:34	00:08:38	00:10:40
R09	00:08:54	00:08:00	00:08:25
R10	00:08:14	00:02:36	00:07:16
R11	00:04:56	00:06:46	00:05:19
R12	00:08:36	00:04:30	00:03:15
R13	00:08:49	00:10:55	00:16:02
R14	00:01:43	00:00:58	00:02:54
Build	00:09:53	00:02:08	00:01:57
Correção	-	00:03:38	00:00:58
Total	02:55:44	02:32:47	01:40:45

Em geral, os protótipos utilizando ferramentas de IAG levaram menos tempo para serem implementados. Dois exemplos são o R02 e R08 onde o tempo sem ferramentas foi de pelo menos dez minutos a mais no protótipo Sem IAG do que nos protótipos Gemini e ChatGPT. Os dois requisitos envolveram questões de movimentação e física, o que pode apontar para que a utilização de ferramentas IAG traga benefícios de tempo para esses casos de uso.

Contudo, houve casos onde o protótipo Sem IAG levou menos tempo, como em R13 e R11. Dois requisitos que envolvem questões de desing de tela e efeitos sonoros. O que pode indicar que em tarefas mais criativas e mecânicas da *engine* a IAG não tem tanta influência.

5.0.2 Desempenho

Não houve diferença significativa do desempenho entre os três protótipos. Isso se deve ao fato de ser um jogo com poucas funcionalidades que não exigem muito de processamento e memória. Os dados da Tabela 8 foram adquiridos com a ferramenta *Xbox Game Bar* e com a ferramenta *"Stats"* dentro na *engine*. Foram registrados os dados para cada uma das telas apresentadas anteriormente e feito uma média simples de cada aspecto para cada protótipo.

Tabela 8 – Desempenho dos Protótipos - Fonte: Elaboração Própria.

	Sem IAG	Gemini	ChatGPT
Taxa de FPS	347.6	369.3	386.0
Uso de Memória RAM (%)	59	59	59
CPU	2.9ms	2.7ms	2.6ms

5.0.3 Qualidade do Código

A análise da qualidade do código feita seguindo os conceitos do livro *Clean Code*(MARTIN, 2009) mostra que o protótipo Sem IAG obteve a pontuação total mais alta, seguido pelo protótipo desenvolvido com a ferramenta ChatGPT e, por último, pelo protótipo desenvolvido com a ferramenta Gemini.

Tabela 9 – Qualidade do Código dos Protótipos - Fonte: Elaboração Própria.

	Sem IAG	Gemini	ChatGPT
Identificadores Descritivos	9.49	9.61	9.37
Função Única	9.68	8.57	8.33
DRY	9.00	10.00	10.00
Coesão	10.00	9.0	9.00
Redundância	9.00	8.00	10.00
Pontuação Total	9.434	9.036	9.340

Para a análise dos pontos Identificadores Descritivos e Função Única foi feito uma contagem da quantidade de variáveis ou funções que atendiam os critérios, divida pelo total de cada aspecto e multiplicada por 10. Por exemplo, para um total de 59 variáveis onde 53 podem ser consideradas "boas variáveis"o valor final é 9,49.

Para DRY foi analisada a totalidade do código e contada a quantidade de código igual entre as classes. No geral, em todos os protótipos não houve ocorrência de repetição de código.

A análise da coesão consistiu em olhar as classes e analisar se estavam responsáveis por funcionalidades fora de seu escopo, ou funcionalidades que deveriam estar em outras classes. Importante ressaltar que nos protótipos feitos com IAG as ferramentas têm a tendência de juntar as funcionalidades todas em uma mesma classe quando a ferramenta não percebe que foi trocado o requisito. Sendo nesse caso necessário adicionar prompts para separar as classes.

Para redundância foi verificada se variáveis, funções ou classes eram inutilizadas, ou complicadas logicamente quando existem outras formas mais simples de atingir o requisito desejado.

5.0.4 Resumo

O protótipo ChatGPT teve o menor tempo de entrega e segunda maior qualidade de código, enquanto o protótipo Gemini teve um tempo de entrega e qualidade de código entre os dois outros protótipos. No entanto, o protótipo Sem IAG teve a melhor pontuação na qualidade do código, o que sugere que, apesar de levar mais tempo, o código gerado a mão pode ser mais limpo e de fácil manutenção a longo prazo.

6 CONSIDERAÇÕES FINAIS

Neste trabalho, foi explorado o uso da IAG no processo de desenvolvimento de jogos. Como apresentado, o desenvolvimento de um jogo é um processo constituído por várias etapas que constituem uma indústria de bilhões de dólares anuais. No entanto, esse processo frequentemente enfrenta desafios, como a exaustão da equipe devido a longas horas de trabalho e atrasos nas entregas.

Diante disso, grandes empresas de jogos estão sempre em busca de métodos e ferramentas que otimizem o seu processo de desenvolvimento. A IAG tem sido uma opção para empresas como a *Ubisoft* e a *CD Projekt Red*, que já a utilizam em seus projetos.

Com o intuito de avaliar o uso da IAG no desenvolvimento de jogos, neste trabalho foram desenvolvidos três protótipos de um jogo de tiro 2D estilo *arcade* na *engine Unity3D*. Dos três protótipos, um foi desenvolvido sem o auxílio de IAG e dois utilizaram as ferramentas: ChatGPT e Gemini, utilizando os mesmos requisitos e estrutura de *prompts* Esses protótipos foram desenvolvidos com o intuito de realizar uma análise comparativa do processo de desenvolvimento entre os três, bem como possíveis diferenças de desempenho.

Durante o desenvolvimento foi constatado que os protótipos feitos utilizando as ferramentas de IAG tiveram um tempo menor para seu desenvolvimento. Com um desempenho similar devido à simplicidade do protótipo. Um ponto a destacar é que o protótipo desenvolvido Sem IAG teve uma qualidade do código superior.

Sendo assim, a IAG tem o potencial de melhorar o desenvolvimento de jogos e pode reduzor o tempo necessário para a conclusão de projetos. No entanto, a qualidade do código gerado é um fator a ser considerado. Uma hipótese é que com o avanço das ferramentas a qualidade do código gerado aumente. O uso da IAG pode ser benéfico para tarefas de rotina e permite que os desenvolvedores se concentrem nos aspectos mais criativos.

No futuro, é importante explorar ainda mais o uso da IAG na indústria de jogos e desenvolver práticas para uma integração mais eficaz da IAG no processo de desenvolvimento. Além disso, é fundamental continuar avaliando o impacto dessa tecnologia em diferentes tipos de jogos e projetos.

Referências

ANASTASIA. **WHAT IS AAA GAME DEVELOPMENT**. 2024. Disponível em: <https://ejaw.net/what-is-aaa-game-development/>. Citado na página 1.

BURNES, A. **https://www.nvidia.com/en-us/geforce/news/nvidia-ace-for-games-generative-ai-npcs/**. 2023. Disponível em: <https://www.nvidia.com/en-us/geforce/news/nvidia-ace-for-games-generative-ai-npcs/>. Citado na página 7.

CHENG, W. **Can AI code Flappy Bird? Watch ChatGPT tryg**. 2023. Youtube. Disponível em: <https://www.youtube.com/watch?v=8y7GRYaYYQg&t=3s&ab_channel=candlesan>. Citado na página 7.

CIRIACO, D. **Google lança IA Gemini capaz de ensinar matemática e programar**. 2023. Disponível em: <https://canaltech.com.br/inteligencia-artificial/google-lanca-gemini-para-bard-e-android-e-acirra-disputa-com-a-openai-272253/>. Citado na página 5.

COURNOYER, F. **Massive Crowd on Assassin's Creed Unity: AI Recycling**. 2022. GDC Presentation. Disponível em: <https://www.youtube.com/watch?v=Rz2cNWVLncI&ab_channel=GDC>. Citado 2 vezes nas páginas 1 e 7.

EDWARDS, P.; LANDRETH, C.; POPŁAWSKI, M. **JALI Driven Expressive Facial Animation Multilingual Speech in CYBERPUNK 2077**. 2020. Siggraph 2020 presentation. Disponível em: <https://www.youtube.com/watch?v=uFIxiz0jwRE&ab_channel=JaliResearchInc>. Citado 2 vezes nas páginas 1 e 7.

ENTERTAINMENT, U. **Ubisoft is Developing an AI Ghostwriter to Save Scriptwriters Time [NA]g**. 2023. Disponível em: <https://www.youtube.com/watch?v=8OeSOXHBvCI&t=1s&ab_channel=Ubisoft>. Citado na página 7.

FOSTER, D. **Generative Deep Learning Teaching Machines to Paint, Write, Compose, and Play**. Second. [S.l.]: O'Reilly Media, Inc., 2019. ISBN 978-1-492-04194-8. Citado 2 vezes nas páginas 1 e 3.

GARCIA, G. **ChatGPT: O que é, Funcionamento, Vantagens e Desvantagens [Guia]**. 2023. Disponível em: <https://mercadoonlinedigital.com/blog/chatgpt/>. Citado 2 vezes nas páginas 5 e 6.

GOOGLE. **Perguntas frequentes sobre o Gemini**. 2023. 23/09/2023. Disponível em: <https://gemini.google.com/faq>. Citado na página 5.

HICKS, L. **The 10 Longest Development Times in Video Game History**. 2022. Disponível em: <https://screenrant.com/longest-development-times-video-games/>. Citado na página 1.

HOUDE, S. et al. Opportunities for generative ai in ux modernization. **Workshops at the International Conference on Intelligent User Interfaces (IUI)**, CEUR-WS.ORG, v. 3124, 2022. Citado na página 6.

MARTIN, R. C. **Clean Code A Handbook of Agile Software Craftsmanship**. First. [S.l.]: Pearson Education, Inc, 2009. ISBN 9780132350884. Citado 2 vezes nas páginas 12 e 30.

NEWZOO. **Newzoo Global Games Market Report (free version)**. 2022. Disponível em: <https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2022-free-version>. Acesso em: 28 de abril de 2023. Citado na página 1.

OPENAI. **What is ChatGPT? - Commonly asked questions about ChatGPT**. 2023. 23/09/2023. Disponível em: <https://help.openai.com/en/articles/6783457-what-is-chatgpt>. Citado 2 vezes nas páginas 5 e 6.

OZDEMIR, S. **Quick Start Guide to Large Language Models - Strategies and Best Practices for using ChatGPT and Other LLMs**. First. [S.l.]: Addison-Wesley Professional, 2023. ISBN 9780138199425. Citado na página 4.

PENG, S. The impact of ai on developer productivity: Evidence from github copilot. eprint arXiv:2302.06590, 2023. Citado 2 vezes nas páginas 5 e 6.

SCHATSKY, D. Ai is helping to make better software - professionals are using artificial intelligence for help in design, development, and deployment. **Deloitte Insights**, 2020. Citado na página 6.

SIMEONE, O. A brief introduction to machine learning for engineers. 2018. Citado na página 3.

UNITY. **Unity User Manual 2020.2**. 2020. Disponível em: <https://docs.unity3d.com/2020.2/Documentation/Manual/index.html>. Citado 2 vezes nas páginas 14 e 17.