UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Curso de Bacharelado em Ciência da Computação

Trabalho de Conclusão de Curso

Utilizando o Algoritmo K-Nearest Neighbors (KNN) para um Sistema de Recomendação de Mapas Gerados Proceduralmente

Nidal Martins Ali

Pelotas, 2022

Nidal Martins Ali

Utilizando o Algoritmo K-Nearest Neighbors (KNN) para um Sistema de Recomendação de Mapas Gerados Proceduralmente

Trabalho de Conclusão de Curso apresentado ao Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Rafael Piccin Torchelsen
Coorientador: Prof. Dr. Marilton Sanchotene de Aguiar

Pelotas, 2022

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

A396u Ali, Nidal Martins

Utilizando o algoritmo K-Nearest Neighbors (KNN) para um sistema de recomendação de mapas gerados proceduralmente / Nidal Martins Ali ; Rafael Piccin Torchelsen, orientador ; Marilton Sanchotene de Aguiar, coorientador. — Pelotas, 2022.

52 f. : il.

Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) — Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2022.

1. PCG. 2. Mapas. 3. Jogador. 4. Recomendação. I. Torchelsen, Rafael Piccin, orient. II. Aguiar, Marilton Sanchotene de, coorient. III. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Nidal Martins Ali

Utilizando o Algoritmo K-Nearest Neighbors (KNN) para um Sistema de Recomendação de Mapas Gerados Proceduralmente

Trabalho de Conclusão de Curso aprovado, como requisito parcial, para obtenção do grau de Bacharel em Ciência da Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 18 de novembro de 2022

Banca Examinadora:

Prof. Dr. Rafael Piccin Torchelsen (orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Ulisses Brisolara Corrêa
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dra. Tatiana Aires Tavares
Doutora em Computação pela Universidade Federal do Rio Grande do Sul.

Dedico este trabalho a todas as pessoas que estiveram ao meu lado durante esta jornada.. . .

AGRADECIMENTOS

Agradeço principalmente aos meus pais e amigos, por estarem comigo tantos nos momentos mais tranquilos, quanto nos momentos mais difíceis do meu caminho, sem eles não teria conseguido...

Uma coisa é certa, quanto mais profundamente confuso você fica em sua vida, mais aberta sua mente se torna para novas ideias. — NEIL DEGRASSE TYSON

RESUMO

ALI, Nidal Martins. **Utilizando o Algoritmo K-Nearest Neighbors (KNN) para um Sistema de Recomendação de Mapas Gerados Proceduralmente**. Orientador: Rafael Piccin Torchelsen. 2022. 52 f. Trabalho de Conclusão de Curso (Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2022.

A prática de PCG(*Procedural Content Generation*) vem se tornando cada vez mais popular no cenário de desenvolvimento de jogos, principalmente por ter a capacidade de gerar uma variedade de gameplay quase que interminável através da geração de níveis inteiros, o que traz a possibilidade de criar uma experiência exclusiva para seus jogadores, aumentando assim o seu fator replay. No entanto, há jogos que usam a PCG de outras formas, seja pra a disposição de items ou inimigos num mapa, ou para a criação de textos e diálogos, etc.

Qualquer que seja o uso da PCG escolhido pelos desenvolvedores, a verdade é que cada vez mais a indústria de videogames vem sido tomada por essa prática, tendo em vista as dezenas de jogos lançados por ano usando a PCG. Tendo isso em mente, este trabalho propõe a criação de um modelo de sistema de recomendação de mapas gerados proceduralmente sobre um jogo, usando o algortimo de inteligência artificial, KNN*(K-Nearest Neighbours)*. A ideia é oferecer para o jogador uma recomendação de mapas gerados proceduralmente já existentes, baseado em jogadas anteriores deste mesmo jogador ou de outros jogadores com estilo semelhante, assim personalizando a experiência de cada jogador.

Palavras-chave: PCG. Mapas. Jogador. Recomendação.

ABSTRACT

ALI, Nidal Martins. **Using the K-Nearest Neighbors (KNN) Algorithm for a Procedurally Generated Map Recommender System.**. Advisor: Rafael Piccin Torchelsen. 2022. 52 f. Undergraduate Thesis (Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2022.

The practice of PCG(*Procedural Content Generation*) is becoming increasingly popular in the game development scenario, mainly because it has the ability to generate an almost endless variety of gameplay, through the generation of entire levels, the that brings the possibility to create an exclusive experience for its players, thus increasing its replay factor. However, there are games that use PCG in other ways, be it for the placement of items or enemies on a map, or for the creation of texts and dialogues, etc.

Whatever the use of PCG chosen by the devolopers may be, the truth is that the video game industry has been increasingly taken over by this practice, given the dozens of games released each year using PCG. With that in mind, this work proposes the creation of a model of recommendation system of procedurally generated maps about a game, using the artificial intelligence algorithm, KNN*(K-Nearest Neighbors)*. The idea is to offer the player a recommendation of existing procedurally generated maps, based on previous plays by the same player or others players with a similar style, thus customizing the experience of each player.

Keywords: PCG. Maps. Player. Recommendation.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

PCG	*Procedural Content Generation*
IA	*Inteligência Artificial*
KNN	*K-Nearest Neighbors*
SBPCG	*Search-based procedural content generation*
ML	*Machine Learning*
PCGML	*Procedural Content Generation via Machine Learning*
API	*Application Programming Interface*
CSV	*Comma-Separated Values*

SUMÁRIO

1 INTRODUÇÃO

Desde a sua criação, os videogames vem se tornado cada vez populares e influentes na indústria do entretenimento, do lançamento do Atari nos anos 80, passando pelos clássicos Super Nintendo e Mega Drive, chegando a era 3D com o Playstation 1 e GameCube.

É claro que essa indústria relativamente jovem, especialmente se comparada a outras formas de entretenimento como filmes e televisão, inclusive sendo uma indústria mais rentável que a do cinema, como por exemplo, o lançamento do jogo *Grand Theft Auto V* (GTA), uma das mais famosas franquias do mundo dos videogames, que quebrou inúmeros recordes em seu lançamento, que teve o seu custo de produção avaliado em $266 milhões de doláres[1], e após o primeiro dia de lançamento ao gerar uma receita de $800 milhões de doláres, e depois de 3 dias gerando uma receita superior a $1 bilhão de doláres.[2], um valor que nem os maiores *blockbusters* de Hollywood não conseguiam alcançar na época.

No entanto o GTA V é um ponto fora da curva, e especialmente nessa década, o grau de complexidade para o desenvolvimento de jogos vêm crescendo muito mais que a rentabilidade que os jogos tem, e com isso os desenvolvedores vêm buscando técnicas para minimizar o trabalho a ser realizado.

Uma das mais famosas é a PCG (*Procedural Content Generation*), que vêm se tornando muito popular principalmente no mercado *indie* de jogos, que é um mercado de orçamentos mais baratos, onde pequenas empresas e desenvolvedores amadores conseguem desenvolver jogos sem a alta demanda e o marketing que as grandes empresas com orçamentos astrônomicos possuem.

Como os autores (SHAKER; TOGELIUS; NELSON, 2016) explicam em seu livro, as razões para o grande aumento no uso de métodos PCG para o desenvolvimento de jogos se devem por algumas se duas principais características. A principal delas é que a PCG retira a necessidade de envolvimento humano para gerar o conteúdo, já que

[1]Noticia disponível em: https://www.gamespot.com/articles/grand-theft-auto-v-may-have-cost-266-million-to-develop-and-market-report/1100-6414188/

[2]Noticia Disponível em: https://www.theverge.com/2013/9/20/4752458/grand-theft-auto-v-earns-one-billion-in-three-days

tudo é feito algoritmicamente. Outro motivo citado é que essa prática é usada de forma mais criativa, já que uma abordagem algoritmica para a geração de conteúdo pode diferir radicalmente do tipo que um humano criaria, assim oferecendo mais variações inesperadas de conteúdo para o jogador.

Porém, apesar da enorme variedade e as possibilidades que este metódo entrega, ela não é direcionada especificamente ao jogador no sentido que ela não necessariamente é customizável para os desejos ou habilidades do jogador, sendo assim, apesar de possuir um alto valor de replay, isso não necessariamente significa que o conteúdo gerado será do agrado do usuário, ou se até mesmo é viável de ser jogado, pois como tudo é feito por algortimos, podem ocorrer alguns bugs, como o conteúdo criado não estar completo, o que impossibilita o jogador de dar continuidade em sua jogada.

Portanto, neste trabalho, a partir do uso do KNN treinado sobre um dataset com informações sobre como o usuário jogou a sua rodada, e suas opiniões. E conseguimos melhorar a experiência do jogador ao focalizar as principais vantagens do PCG de acordo com a sua customização pessoal baseado em seus padrões de comportamento durante sua jogada.

1.1 Objetivos

Este trabalho tem como objetivo de atráves de métodos de machine learning, sendo mais específico, usando o algoritmo de classificação KNN, criar um sistema de recomendação de mapas gerados proceduralmente para um jogador, e assim oferecer o melhor ou os melhores mapas baseados em seu comportamento durante sua a última rodada, sobre um conjunto de mapas gerados proceduralmente em um jogo desenvolvido no projeto do (PADILHA, 2022), no qual que este trabalho está inserido.

E no final, avaliar os mapas recomendados de acordo com os padrões do jogador.

1.2 Metodologia

Primeiramente, é importante dizer que este trabalho faz parte de uma pesquisa maior, e por consequência, faz parte de um projeto de pesquisa em andamento.

Este trabalho se propõe a realizar a integração de uma IA dentro do contexto de um jogo 2D.

Após a realização de toda uma pesquisa teórica e trabalhos relacionados sobre esse tema, o próximo passo, usando o *Scikit-Learn*, uma biblioteca de ML para a linguagem Python que suporta aprendizados de máquina supervisionados e não supervisionados, fazendo o uso dessa biblioteca, é realizada a construção de todo o conjunto de treinamento do modelo sobre um dataset.

A estratégia para a seleção das features para o treinamento do modelo foi o uso dos

heatmaps. Heatmaps são representações gráficas bimensionais de dados, onde os valores individuais desses dados que estão contidos nessa matriz são representadas em cores diferentes. A figura 21 mostra o heatmap gerado para esse trabalho.

Usando essa ferramenta foram selecionadas as features mais relavantes para o treino atráves da correlação desses dados. Após toda essa construção, foi realizada a implementação do KNN sobre esse conjunto de features.

Depois do desenvolvimento e treinamento do modelo KNN foi implementada uma API para realizar a integração de dados entre o modelo e a Unity, plataforma onde o jogo foi desenvolvido. Quando todo este processo estiver pronto, a API vai ler alguns dados selecionados do jogador, sendo estas as features do dataset, e irá mandar para o modelo KNN integrado na API, onde ela irá retornar de volta para a Unity os dados necessários para a recomendação atráves de um JSON. A figura 9 contém um diagrama em de mais baixo nível referente ao fluxo da integração da API, e a figura 13 contém um diagrama que exemplica o fluxo geral do projeto para facilitar o entendimento.

1.2.1 Contextualizando o Projeto Base

Como já citado, esse projeto tem como base o trabalho do autor (PADILHA, 2022), onde nesse trabalho é desenvolvido o jogo 2D mencionado. Esse jogo tem as características PCG, onde a cada nova rodada, o jogador terá a sua disposição um novo mapa gerado proceduralmente e único. Portanto a motivação desse trabalho é melhor o processo da PCG desse ao recomendar pro jogador mapas que melhor se assemelham ao seu estilo de jogo. Os detalhes sobre esse jogo serão melhor explicados no capítulo 3.

1.3 Organização do Trabalho

O trabalho está organizado em 7 capítulos, no capítulo 2 está toda a fundamentação teórica do trabalho, onde é discutido e explicado a origem dos principais conceitos que serão discutidos ao longo desde trabalho.

No capítulo 3 está os trabalhos relacionados, onde é oferecido um resumo sobre alguns dos principais artigos já escritos e divulgados nesta área, relacionados a este trabalho.

No capítulo 4 é a parte de desenvolvimento, onde é explicado como os o algoritmo e a API foram implementados e integrados, além de apresentar algumas limitações e problemas encontrados durante a execução do projeto.

No capítulo 5 é mostrada a metodologia de testes feita, e também a discussão e a apresentação dos resultados obtidos.

O capítulo 6 apresentada a conclusão do trabalho, onde os principais temas do

projeto são abordados, o nível de satisfação dos resultados obtidos e as principais contribuições e também é discutido os trabalhos futuros, e de que formas ele pode ser melhorado.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo discutir os principais conceitos que são apresentados nesse trabalho, sendo eles: Jogos eletrônicos, a história da *Procedural Content Generation*(PCG), Machine Learning e as suas técnicas, o algortimo KNN, API's, a biblioteca *Scikit Learn*, e por último, a plataforma de desenvolvimento Unity.

2.1 Os Primórdios dos Jogos Eletrônicos e da PCG

Assim como o autor (LEITE, 2003) explica em seu artigo, a história dos jogos eletrônicos começa na década de 50, onde o físico nuclear William Higinbotham apresenta para o público de seu laboratório algo inusitado: uma adaptação no software de um osciloscópio que demonstraria a trajetória de uma bola em movimento e os visitantes teriam de interagir com aquele programa. Após três semanas de desenvolvimento, Higinbotham chama sua invenção de 'Tennis for Two' (Tênis para Dois).

Neste primitivo jogo eletrônico o ponto de vista é lateral e uma representação de bola salta ao longo de uma linha horizontal que simboliza a quadra, tendo uma pequena linha vertical no centro como a rede, dois aparatos cúbicos com um disco e botão cada funcionavam como controladores, sendo o disco para controlar o ângulo da trajetória da bola e o botão para rebate-la, se o jogador não rebater a bola corretamente, ela não passa da rede.

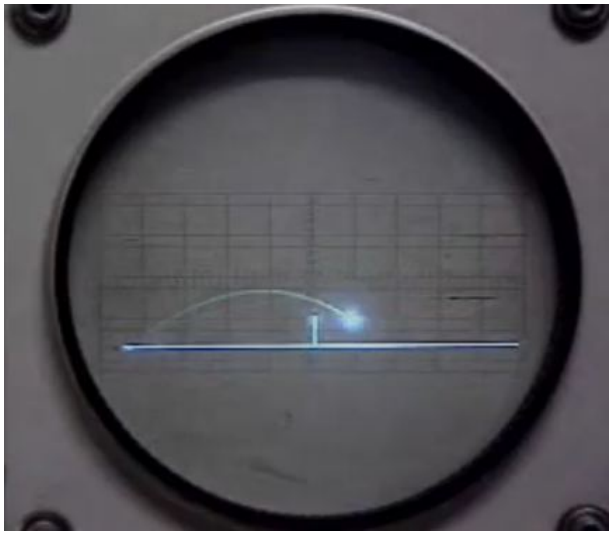

Figura 1 – Imagem do jogo Tennis for Two na tela do Osciloscópio. Fonte: (VG LEGACY, 2021)

Alguns anos mais tarde, na década de 60, estudantes do MIT liderados por Steve Russel desenvolveram a primeira versão de um jogo de duelo entre duas espaço-naves, onde os jogadores interagiam atráves de botões para controlar a velocidade direção das naves, e atiravam mísseis no adversário, esse jogo foi chamado de 'Spacewar'. Com o tempo outros estudantes incrementaram esse jogo, e contraladores específicos para o jogo também foram desenvolvidos para facilitar as interações com os jogadores. Tais controladores foram os precursores dos joysticks[3]

E foi a partir dos anos 80, com esse mercado emergente ficando maior e mais complexo, que problemas devido a limitação de memória dos dispositivos da época começaram a aparecer, e muitas vezes o conteúdo desses jogos tinha que ser gerado atráves de algoritmos, e foi aí que a prática da PCG começou a ganhar força.

2.2 Procedural Content Generation

Os autores(SHAKER; TOGELIUS; NELSON, 2016) descrevem a PCG como a criação algoritmica de conteúdo com entrada limitada ou indireta do usuário. Em outras palavras, a PCG refere-se a um software que pode criar conteúdos de jogos sozinho, ou em conjunto com um ou vários jogadores ou designers.

Na área de desenvolvimento de jogos isso quer dizer que os níveis, os mapas, quests, items, qualquer objeto que possa ser classificados como "conteúdo" dentro desse jogo é gerado dessa forma.

Sendo assim a principal vantagem desse método é a criação automática de grandes quatidades conteúdos no jogos, e dependendo da implementação, outras vantagens como menores tamanhos de arquivo, e maior aleatoriedade para uma gameplay menos previsível são incluídas.

[3]Periférico acoplado ao aparelho de jogos que permite ao jogador controlar a ação da tela.

Apesar de alguns jogos da época já apresentarem alguma forma de PCG, o primeiro exemplo realmente notável desse método em prática foi no jogo *Rogue*, lançado em 1980.

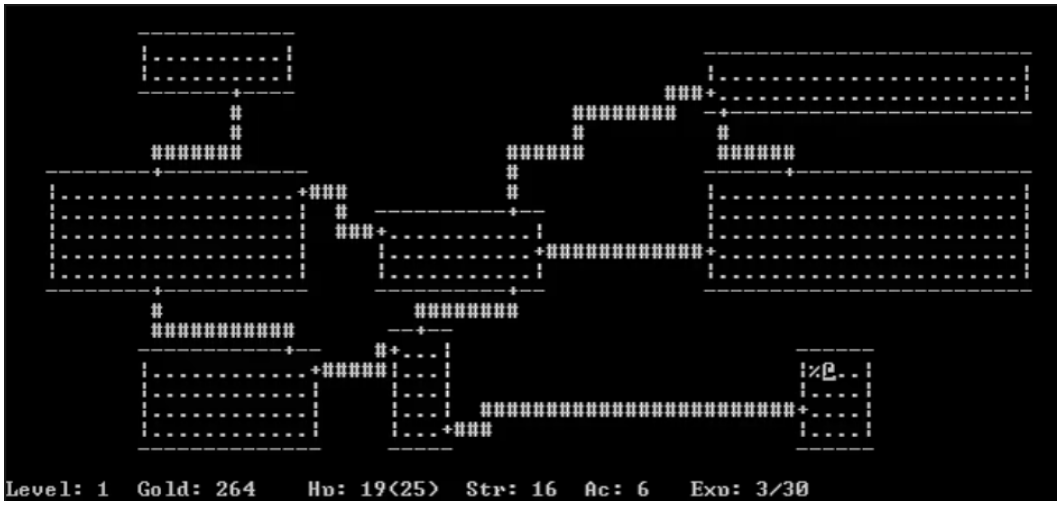

Figura 2 – Imagem do jogo Rogue. Fonte:(TIM BROOKS, 2013)

O sucesso de *Rogue* pode ser atribuído em dois fatores: O primeiro é o uso de caracteres da tabela ASCII para o desenho dos elementos visuais da tela. E segundo é a geração randômica de *layouts* de mapas organização de objetos.

Isso resultava em uma expêriencia rejogável quase infinita, onde dois jogos nunca eram iguais. Esse jogo era tão popular na época, que um subgênero foi criado a partir dele: O *Roguelike*.

Na verdade, o sucesso desse jogo foi tão grande que influenciou outros jogos fora do seu nicho, sendo um deles é o *Tetris*, um dos jogos mais famosos da história, lançado em 1984. Antes do *Tetris*, os jogos de quebra-cabeça tinham somente uma única solução, que era alcançado atráves da manipulação das peças disponíveis, e embora a jogabilidade do Tetris seja bem simples, ela ofereceu o que nenhum jogo de quebra-cabeça ofereceu na época: Um fluxo interminável de peças randômicamente selecionadas sem uma condição de vitória. E isso só é possível devido ao uso da PCG para a criação e seleção das peças.

Figura 3 – Imagem da primeira versão do jogo Tetris, de 1984. Fonte:(THE CPUSHACK MUSEUM, 2015)

Nos dias atuais, o exemplo mais famoso e bem sucedido de um jogo que usa a PCG para criação de todo o seu mundo é o Minecraft, onde o tamanho total dos mapas tem em média 64.000 quilômetros de diâmetro[3]. Para colocar em perspectiva, o planeta Terra possui 12.700 quilômetros de diâmetro. O mais incrível disso é que apesar do tamanho astrônomico que os mundos gerados proceduralmente no Minecraft podem chegar, o arquivo do jogo tem somente 1GB de tamanho.

E essa é principal vantagem da PCG, a capacidade gerar conteúdos praticamente infinitos e comprimir tudo isso em arquivos relativamente pequenos.

Figura 4 – Exemplo de um mundo criado proceduralmente no Minecraft. Fonte: Própria

[3]Notícia disponível em: https://www.gameinformer.com/b/news/archive/2012/02/09/how-big-is-a-minecraft-world.aspx

2.2.1 Roguelikes

Como já citado, devido ao grande sucesso de *Rogue* todo um subgênero foi criado. Os Roguelikes que além da usual variedade de gameplay, onde nenhum nível será igual, onde o loop do jogo para usuário algo novo a cada nova rodada. Mas a característica que realmente define o genêro roguelike, é a sua dificuldade, onde geralmente as mortes do usuário dentro de uma rodada são permanentes, assim toda a experiência, items, habilidades conquistadas são perdidas para sempre.

Então assim que o jogador iniciar sua rodada novamente, ele terá que ganhar tudo aquilo que ele perdeu de volta, porém, como nenhum nível será igual ao outro, tudo o que ele irá coletar será diferente da sua última vez.

Figura 5 – Imagem do jogo Spelunky 2, um famoso roguelike. Fonte:(JOGORAMA, 2022)

2.3 Machine Learning

Assim como (IBM, 2020) explica em seu texto, Machine Learning é um ramo da área de Inteligência Artifical (IA) e da ciência da computação que foca no uso de dados e algorítmos para imitar a forma como os seres humanos aprendem, graduamente aumentando sua acurácia.

Os classificadores de machine learning são divididos em três categorias: ML supervisionado, ML não-supervisionado e Aprendizado por reforço.

A figura 6 demonstra um diagrama que descreve a forma de como os métodos de ML são organizados

2.3.1 Machine Learning Supervisionado

O aprendizado supervisionado, também conhecido como Machine Learning supervisionado, é definido pelo uso de conjuntos de dados rotulados para treinar algoritmos para classificar dados ou prever resultados com precisão. À medida que os dados de entrada são inseridos no modelo, ele ajusta seus pesos até que o modelo seja ajustado adequadamente.

O algoritmo KNN usado neste trabalho pode ser aplicado tanto para o uso supervisionado, quanto para o não supervisionado. No caso deste trabalho foi escolhido o uso supervisionado, porém isso será melhor discutido no capítulo sobre o Desenvolvimento.

2.3.2 Machine Learning Não-Supervisionado

O aprendizado não supervisionado, usa algoritmos de ML para analisar e agrupar conjuntos de dados não rotulados. Esses algoritmos descobrem padrões ocultos ou agrupamentos de dados sem a necessidade de intervenção humana. Sua capacidade de descobrir semelhanças e diferenças nas informações o torna a solução ideal para análise exploratória de dados, estratégias de venda cruzada, segmentação de clientes, reconhecimento de imagens e padrões.

O autor (PEDRO BARROS, 2016) explica em seu texto, que essa categoria nos permite abordar problemas com pouca ou nenhuma idéia de como que os nossos resultados devem ser aparentar, e também pode ser usada para reduzir o número de dimensões em um conjunto de dados para concentrar somente nos atributos mais úteis, ou para detectar tendências.

Com aprendizagem não supervisionada não há feedback com base nos resultados da previsão.

2.3.3 Aprendizagem por Reforço

Aqui o sistema de aprendizado é chamado de agente. Esse agente irá interagir com o ambiente externo para realizar uma ação e atingir um determinado objetivo. O ambiente irá recompensar ou punir o agente de acordo com suas ações. E assim, com base no feedback que ele recebeu por ter executado tal ação, ele aprenderá por si só qual é a melhor estratégia a seguir.

Um exemplo de utilização dessa categoria são IA's usadas para os estudos de jogadores profissionais de Xadrez, onde o sistema realiza milhões de partidas contra si mesmo para descobrir as melhores estratégias e jogadas.

2.3.4 A rotulagem de dados para o aprendizado supervisionado

A rotulagem de dados faz parte do estágio de pré-processamento ao desenvolver um modelo de machine learning (ML). Requer a identificação de dados brutos (ou

seja, imagens, arquivos de texto, vídeos) e, em seguida, a adição de um ou mais rótulos aesses dados para especificar seu contexto para os modelos, permitindo que o modelo de machine learning faça previsões precisas.(IBM, 2021)

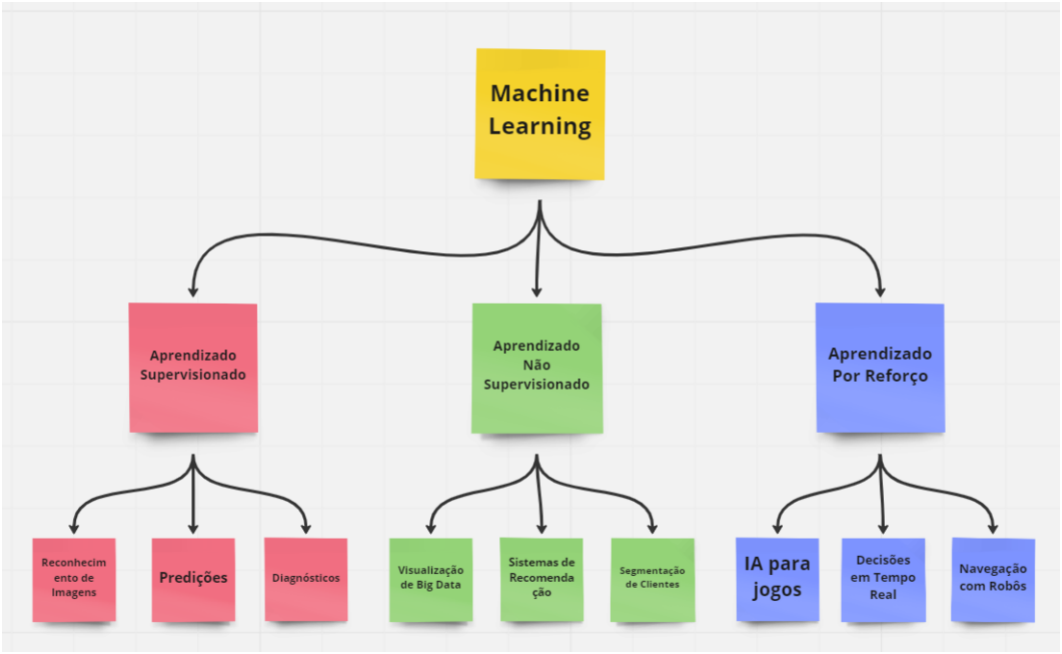

Figura 6 – Imagem do jogo Spelunky 2, um famoso roguelike

2.4 Algoritmo KNN

De acordo com a (IBM, 2018), o K-Nearest Neighbours, também conhecido como KNN, é um algortimo classificador de aprendizado supervisionado não paramétrico, que usa a proximidade para fazer classificações ou previsões sobre o agrupamento de um ponto de dados individual. É normalmente usado como um algoritmo de classificação, partindo do pressuposto de que pontos semelhantes podem ser encontrados próximos uns dos outros.

(ASHISH MEHTA, 2021) conta com mais detalhe que o KNN sendo um um algoritmo de ML supervisionado, isso significa que precisamos de um conjunto de dados de referência para determinar a categoria dos pontos de entrada que serão inseridos. Esse algoritmo classifica o conjunto de dados fornecido em diferentes grupos ou categorias, e quando um novo ponto de dados é inserido, o algoritmo nos ajuda a identificar e classificar em qual categoria esse novo ponto de dados pertence com base em várias medidas de similaridade.

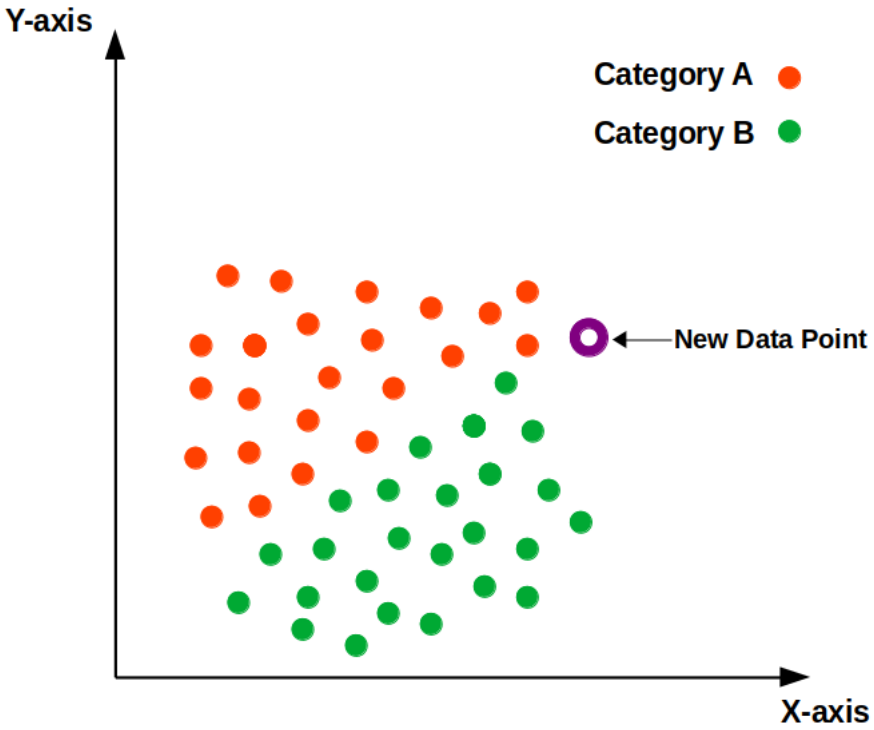

Figura 7 – Diagrama demonstando a lógica por trás do KNN. Fonte:(ASHISH MEHTA, 2021)

Na figura 7 é mostrado a lógica de como funciona o KNN. Supondo que existe um dataset bimensional, onde os dados foram classificados em duas diferentes categorias, sendo a categoria A para os pontos em vermelho, e a categoria B para os pontos em verde. Agora sempre que um novo dado entrar, o KNN tem como objetivo predizer em qual categoria ou grupo este novo dado pertence.

Para fazer isso, o KNN usa a métrica da distância euclidiana, para calcular os 'K' vizinhos mais próximos desse novo ponto de entrada, e baseados na quantidade de vizinhos pertencentes a uma categoria ou outra, a classificação é feita.

Porém para fazer o uso desse algoritmo, os dados não precisam estar necessariamente divididos em duas categorias, como é o caso deste trabalho e esse processo será explicado mais adiante no capítulo sobre o Desevolvimento.

No entanto essa divisão dos dados é a forma mais popular de uso desse algoritmo que é majoritariamente usado para problemas de classificação, que são problemas onde se busca encontrar uma categoria ou classe, dentro de possibilidades limitadas.

2.4.1 Distância Euclidiana

Como já mencionado, o algoritmo KNN usa essa métrica para fazer o cálculo entre dois pontos. Nestre trabalho, essa distância é importante para a realização dos testes, pois foi a base para a comparação dos resultados e indicar se eles foram bons ou não.

$$d(x,y) = \sqrt{\sum_{i=1}^{p}(x_{i-}y_i)^2}$$

Figura 8 – Fórmula da distância euclidiana Fonte: https://fabiobaldini.com.br/machine-learning-k-nearest-neighbors-knn-no-ms-excel-2/

A distância euclidiana é a distância entre dois vetores ou arrays, de valor real. É calculado pela raiz quadrada da soma das diferenças quadradas dos elementos nos dois vetores.

Onde Xi e Yi são os vetores de entrada, sendo o Yi o vetor correspondente a um novo conjunto de dados, e Xi o vetor com os dados destino, ou seja, os dados do qual se quer medir a distância.

2.5 Scikit Learn

Como já mencionado, para a implementação do algortimo KNN, foi utilizada o modulo Scikit-Learn. O autor (PEDREGOSA et al., 2011) explica que o Scikit-learn é um módulo Python que integra uma ampla gama de algoritmos de aprendizado de máquina de última geração para problemas supervisionados e não supervisionados de média escala. Este pacote se concentra em levar o aprendizado de máquina para não especialistas usando uma linguagem de alto nível de uso geral.

E de acordo como o autor (HACKELING, 2017) fala em seu artigo, que desde o seu lançamento em 2007, o scikit-learn se tornou uma das bibliotecas de aprendizado de máquina de código aberto mais populares para Python. O scikit-learn fornece algoritmos para tarefas de aprendizado de máquina, incluindo classificação, regressão, redução de dimensionalidade e clustering. Ele também fornece módulos para extrair recursos, processar dados e avaliar modelos.

2.6 API

Neste trabalho, uma API foi desenvolvida para fazer a integração entre a Unity e o KNN, para dar um contexto geral, APIs são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos.

API significa Application Programming Interface (Interface de Programação de Aplicação). No contexto de APIs, a palavra Aplicação refere-se a qualquer software com uma função distinta. A interface pode ser pensada como um contrato de serviço entre duas aplicações. Esse contrato define como as duas se comunicam usando solicita-

ções e respostas. A arquitetura da API geralmente é explicada em termos de cliente e servidor. A aplicação que envia a solicitação é chamada de cliente e a aplicação que envia a resposta é chamada de servidor. (AWS, 2022)

No contexto deste projeto, o jogo na Unity é o cliente, e o modelo com o algoritmo KNN é o servidor.

2.7 Flask

Para a implementação da API mencionada na seção 2.6, a framework Flask foi utilizada, dessa forma a API lê os dados do jogo em forma de string, e a Flask realiza a conversão, e envia de volta para o modelo treiando realizar a recomendação em tempo real.

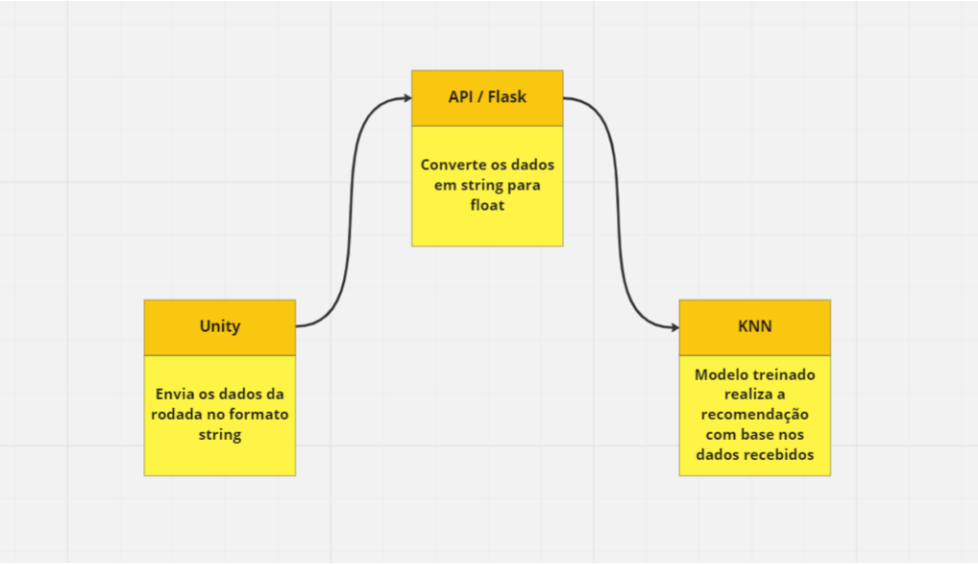

Figura 9 – Diagrama de mais baixo nível referente a figura 11, mostrando o o fluxo da API com a Unity e o modelo KNN. Fonte: Própria

De acordo com os autores (COPPERWAITE; LEIFER, 2015), Flask é uma das frameworks web Python mais usadas pelas start-ups, por ser uma ferramenta perfeita para soluções rápidas e simples para a maioria dos negócios e aplicações gerais. Em sua essência, ela fornece um conjunto de bibliotecas poderosas para lidar com as tarefas mais comuns de desenvolvimento web.

2.8 Game Engines

Antes de começar a falar sobre a Unity, primeiro é necessário falar sobre as engines de jogos. O autor (JACOBSON; LEWIS, 2002) conta em seu livro que o custo para produzir simulações mais realistas cresceu tanto que os desenvolvedores não conseguiam mais contar com a recuperação de seu dinheiro investido a partir do lançamento de único jogo.

A partir dai se deu a necessidade das game engines, escritas para um jogo específico, mas generalizadas o suficiente para serem usadas em toda uma uma família de jogos semelhantes. Em outras palavras, as engines são uma coleção de códigos de simulações modulares que não especificam diretamente a lógica ou o ambiente do jogo, mas ao invés disso lidam com os módulos de input, output (renderizações em 3D ou desenhos em 2D, som, iluminação, texturas), e a física geral do mundo daquele jogo.

2.8.1 Plataforma Unity

O jogo que é utilizado como base desse projeto foi desenvolvido na engine Unity, portanto aqui descrição sobre essa ferramenta.

Unity é uma game engine de ambiente de desenvolvimento integrado (IDE) para a criação de mídias interativas, tipicamente videogames. A primeira versão da Unity foi lançada em junho de 2005, e tinha como objetivo, criar uma game engine acessível com ferramentas profissionais para desenvolvedores de jogos amadores enquanto "democratizava o desenvolvimento de jogos" da indústria.(HAAS, 2014)

A Unity muito popular no mercado atual, devido a sua simplicidade e fácil acesso e apesar de ter uma versão paga, a versão free da plataforma oferece todas as ferramentas necessárias para o desenvolvedores amadores criarem os seus projetos, e por esse motivo que essa engine é a porta de entrada de muitos aspirantes a game developers mundo afora.

A imagem abaixo mostra a tela inicial do jogo base desse projeto dentro da Unity, e a sua interface geral.

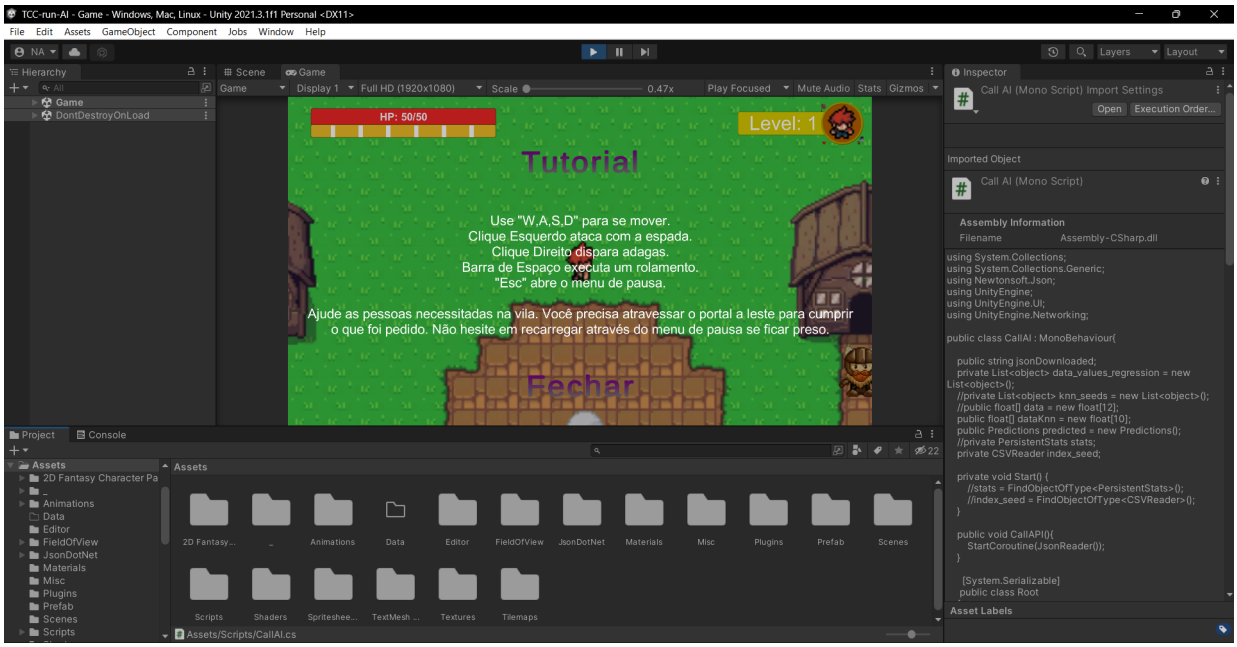

Figura 10 – Screenshot da interface principal da Unity. Fonte: Própria

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos relacionados a utilização e implementação de PCG, aplicados a emoção e comportamentos do jogador, como também trabalhos que visam investigar e classificar essas técnicas, deixando espaço para novas abordagens para trabalhos futuros.

Primeiramente, o mais importante a se citar é o (PADILHA, 2022), cujo trabalho é toda a base para o modelo KNN desenvolvido neste trabalho apresentado aqui, nele o colega se propôs a implementar um gerador procedural de mapas para um jogo 2D top-down, e uma ferramenta capaz de executar um método de avaliação da qualidade desses mapas gerados.

O jogo em si é um um RPG de ação, onde podemos andar, atacar, rolar, e atirar facas. No hub inicial (figura 11), antes de entrarmos no mapa, podemos escolher missões para realizar no mapa, ao completar essas missões, derrotar inimigos, coletar itens no mapa, subimos de expêriencia, melhorando assim os atributos do personagem.

Nas figura 12 temos um exemplo de mapa gerado para este jogo, onde todas as cavernas, disposições de items e agentes são gerados proceduralmente.

Figura 11 – Visão geral da hub inicial do jogo. Fonte: Própria

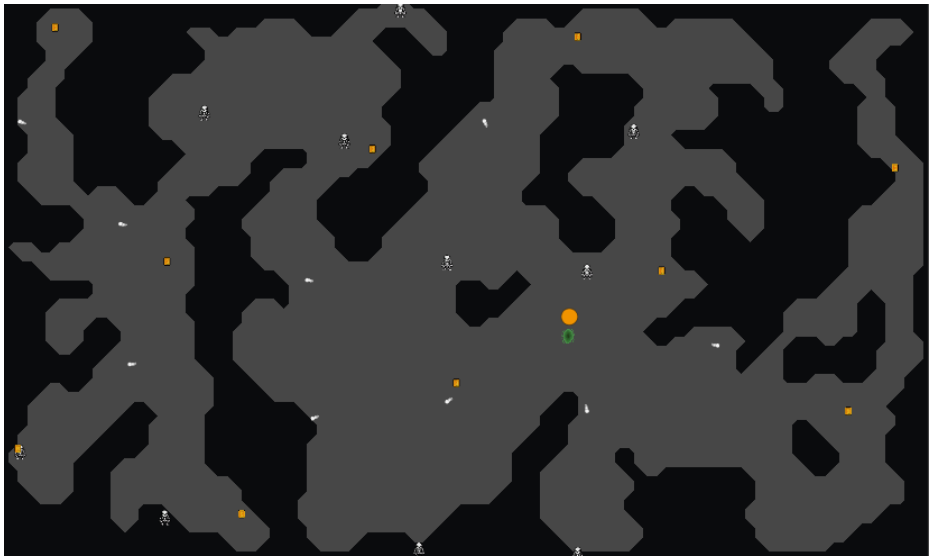

Figura 12 – Exemplo de um mapa gerado pelo gerador procedural. Fonte: Própria

Já sobre os demais trabalhos, o de (TOGELIUS et al., 2011), centra-se em que tipo de conteúdo é gerado, como esse conteúdo é representado e como a qualidade do conteúdo é avaliada, devido a um certo números de artigos que começaram a aparecer no qual os autores implementam algoritmos de busca aleatória, como computação evolutiva, para gerar conteúdo do jogo, como níveis, regras, etc..

Essse trabalho tem um teor mais teórico, onde visa principalmente em analisar como o método mais recente se comparado ao PCG padrão, *Search-based Procedural Content generation*(SBPCG) gera e representa o seu conteúdo.

Uma outra abordagem também feita pelo mesmo autor do trabalho anterior, (SHAKER; YANNAKAKIS; TOGELIUS, 2012) propõem o desenvolvimento de um algoritmo que pode observar um ser humano jogando, e corretamente julgar o que o jogador esta experienciando enquanto ele ou ela joga, já que isso permite eles a adaptar o jogo ao jogador, e também os ajuda a entender como o afeto humano é expressado no comportamento.

A abordagem realizada para alcançar esse objetivo foi primeiro construir modelos precisos da relação entre a experiência do jogador e o conteúdo do jogo. Isso requer a construção de modelos de dados orientados com base em dados coletados sobre o jogo, o comportamento do jogador e correlacionar esses dados com os anotados com as tags de experiência do jogador, esse trabalho e o que está sendo apresentado aqui neste projeto possuem propostas de certa forma, similares, onde ambos usam o comportamento e vontades do jogador como ponto principal.

Já o trabalho (DAHLSKOG; TOGELIUS, 2012) discute como a PCG e os padrões de design podem potencialmente ser combinados de várias maneiras diferentes no design de jogos, apresentando um trabalho em progresso de um gerador de level baseado em *design patterns* para o jogo *Super Mario Bros*(SMB).

E por fim, a quarto e última pesquisa explora a PCGML, definida como a geração

de conteúdo de jogos usando modelos de ML treinados em conteúdo existente. À medida que aumenta a importância do PCG para o desenvolvimento de jogos, os pesquisadores exploram novos caminhos para gerar conteúdo de alta qualidade com ou sem envolvimento humano; este artigo aborda o paradigma relativamente novo do uso de ML.

O artigo (SUMMERVILLE et al., 2018) trata da ideia da prática recente de gerar conteúdo de jogos a partir de modelos aprendidos via ML, onde eles definem como PCGML a geração de conteúdo de jogo por modelos que foram treinados em conteúdo de jogos já existentes. Esse trabalho também possui semelhanças com este apresentado, porém a diferença na PCGML, o contéudo é gerado diretamente a partir do modelo treinado, o que não acontece neste trabalho aqui, onde o modelo treinado vai somente recomendar um conteúdo já existente.

4 DESENVOLVIMENTO

Neste capítulo é apresentada todas as etapas do projeto, sendo essas, o funcionamento do jogo com a PCG citados no capítulo 3, o KNN com o modelo treinado, e por último a API que serviu para realizar a integração entre a plataforma Unity e a IA.

4.1 A PCG e o Jogo

Essa etapa pode descrita da seguinte forma: Temos o jogo gerado proceduralmente, com todo um conjunto de features necessárias para o funcionamento dele, o mapa é gerado, e após o término de uma sessão do jogador, a Unity irá mandar os dados da última rodada para uma planilha do google, onde o algortimo KNN coleta todos esses dados, seleciona as features mais relevantes para então realizar o treinamento do modelo KNN.

Na figura 13 temos o fluxo geral do projeto para exemplificar o funcionamento desse trabalho.

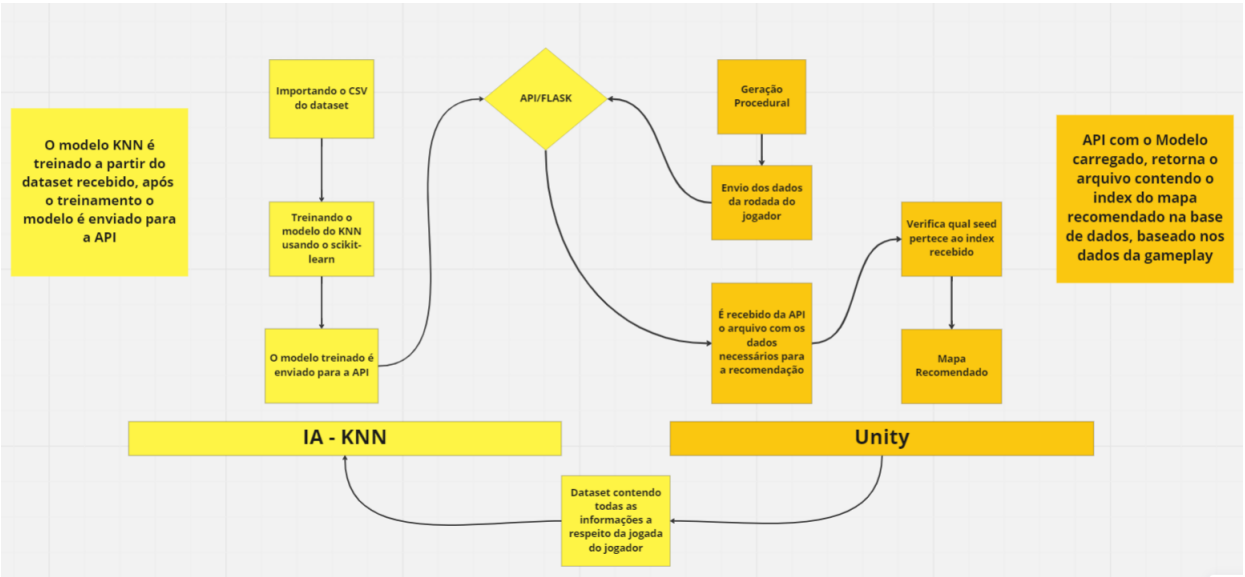

Figura 13 – Fluxo do projeto. Fonte: Própria

Quando o jogador de fato iniciar o jogo, um menu com diversos parâmetros é ofere-

cido ao jogador para ele poder customizar, esses dados são utilizados para a geração procedural dos mapas, sendo alguns exemplos a largura e altura, suavição, número de inimigos. Também existem os dados que são capturados durante a jogada do jogador, tais como: tempo percorrido, porcentagem de inimigos mortos, quantidade de vida perdida, etc. Além disso, após o término de uma rodada, os jogadores passam por um breve questionário onde eles relatam como foi a sua experiência e seu sentimentos em relação ao jogo e ao mapa.

Depois de todo esse processo, a Unity pega todos os dados referentes ao desempenho da rodada do jogador, juntamente com as respostas do seu questionário e os manda para uma planilha no google em tempo real, cada linha dessa planilha vai corresponder a uma jogada diferente, consequentemente, um mapa diferente, então essa planilha é salva como um arquivo CSV, esse arquivo é importado pelo KNN, que é implementado atráves da biblioteca sckit-learn, para a realização do treinamento

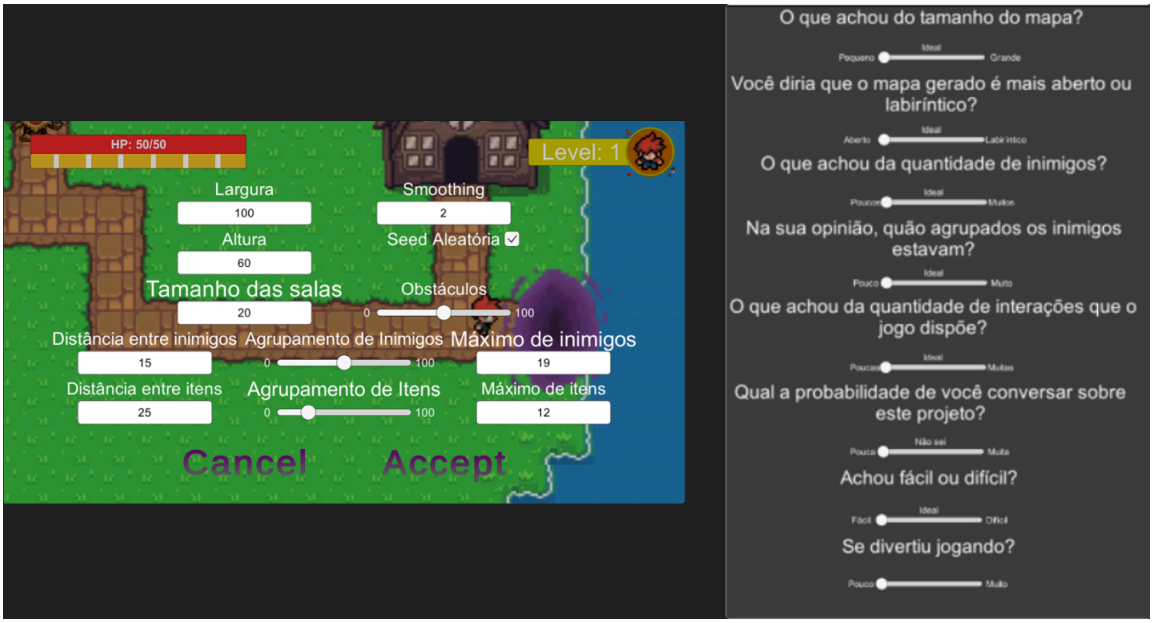

Figura 14 – A esquerda temos o menu inicial onde o jogador pode setar os parâmetros para a geraçao procedural, e a direita temos o questionário. Fonte: Própria

4.1.1 Limitações da PCG e o que o KNN incrementa.

A PCG dentro do jogo mencionado no capítulo 3, é realizada sem levar em conta nenhum parâmetro prévio em consideração, isso quer dizer que na hora que a PCG cria um mapa novo a cada nova rodada, ele não leva em consideração como o jogador se comportou na sua última jogada ou se o mesmo até mesmo gostou ou não do mapa atual, assim ignorando totalmente qualquer padrão de comportamento que esse jogador tenha vindo a mostrar anteriormente e também ignorando seu gosto pessoal e a forma em que ele prefere jogar, por exemplo, se o jogador gosta de mapas maiores com mais inimigos, e lugares mais fechados para o combate, a PCG não vai avaliar nenhum dado sobre essa jogada e simplesmente vai gerar um mapa aleatório qualquer

para sua próxima vez, baseado somente em alguns parâmetros específicos que serão explicados mais adiante, não tendo eles de qualquer relação com estilo de gameplay do jogador.

Dada a essa falta de personalização ao gerar os mapas, veio a motivação para implementação do KNN, que resumidamente, vai oferecer ao jogador uma forma única e pessoal de jogar esse jogo, baseado somente nas experiências dele, e de outros jogadores.

4.2 KNN

Sobre o KNN, ele trabalha em conjunto com a PCG nesse jogo, a ideia, nesse ambiente, é recomendar para o jogador um mapa, que de acordo com a base de dados disponível, juntamente com o treino do modelo do algoritmo KNN, se assemelhe o mais próximo possível com o estilo de gameplay daquele jogador, ou de outros jogadores com estilos semelhantes. Essa recomendação é feita atráves da *seed* do mapa.

Toda vez que um novo mapa é gerado, junto com ele, é gerado aleatóriamente também seed, que nada mais é do que um identificador único para aquele mapa, assim toda a vez que é desejado encontrar algum mapa em específico, a seed é então usada. O sistema de recomendação em si se dá pela interface do menu inicial, após uma jogada do jogador, após todo processo mostrado no fluxo da figura 13 ser completado, ao abrir o menu novamente, e desmarcando a opção de seed aleatória, uma nova seed já estará aparecendo no campo, já pronta para ser jogada, essa seed é representa o mapa recomendado pelo KNN, com seed pronta, o jogador já está apto para jogar o mapa recomendado baseado em sua última sessão de jogo, esse processo é melhor demonstrado na figura 15.

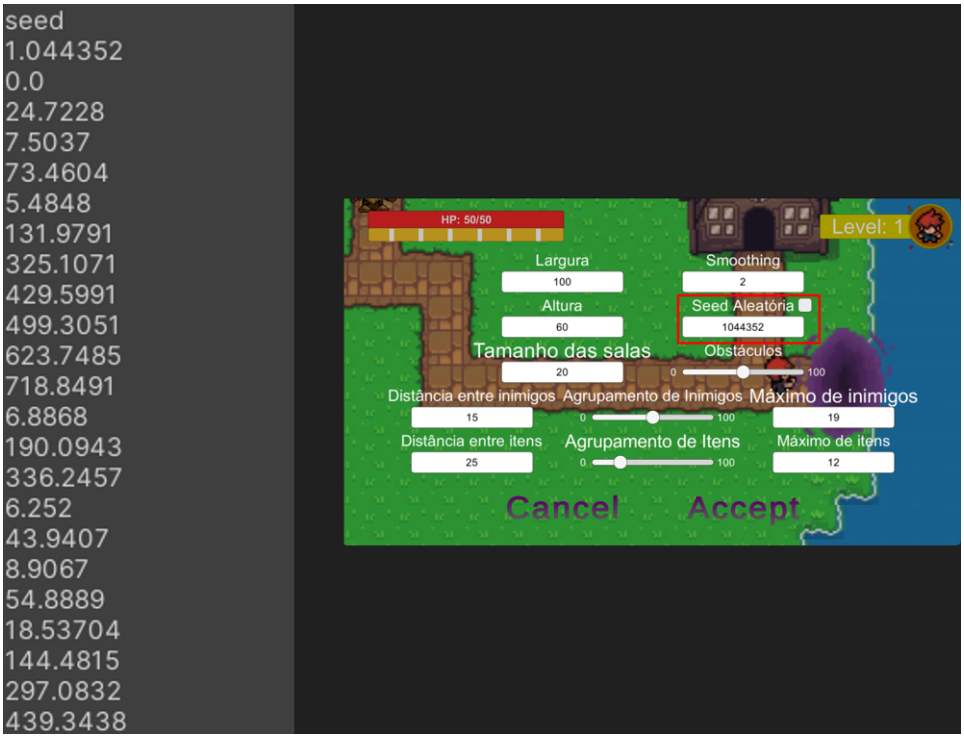

Figura 15 – A esquerda estão alguns exemplos de seeds, e a direita está o menu inicial do jogo, com a seed recomendada já estando a disposição do jogador. Fonte: Própria

Voltando ao fluxo da figura 13, a base de dados é importada para treinar o modelo, e após a seleção das *features* a serem usadas, o modelo é então treinado e exportado para a API.

O dataset ao todo conta com mais de 30 features, contendo informações desde a hora em que jogador realizou a sua jogada, coordenadas de inimigos e items, comprimento e largura dos mapas, desde ao tempo decorrido durante uma jogada, passos dados, total de vida perdida, etc.

As features selecionadas para a criação do modelo se deram após verificar a correlação entre elas, e também se elas se encaixavam no contexto deste trabalho em específico, ou seja, como o objetivo aqui é poder recomendar ao jogador um mapa que lhe proporcione uma expêriencia semelhante ao que ele já jogou e aprovou, e não necessariamente de que o mapa seja fisicamente parecido com o original, logo muitas das features selecionadas tem a ver com o dados de referentes a sua jogada, e principalmente as suas respostas nos questionários, então por esse motivo, features de coordenadas, ou de tamanhos de mapas ficaram de fora.

Ao todo 12 features foram selecionadas, mas somente 10 foram usadas no treino. As duas que não fazem partem do treino são as features *seed* e *fun*, isso porque depois do dataset ficar reduzido após a seleção, era necessário ter acesso as seeds dos mapas, já que que o KNN vai recomendar um mapa sobre esse dataset.

O mesmo vale para a *fun*, essa feature é referente a última questão do questionário, onde pergunta para o jogador se ele divertiu jogando aquele mapa ou não, além de

dar uma nota sobre o mesmo. Então é por esse motivo que existem 12 features selecionadas, mas somente 10 são usadas no treino,

Após o treinamento do modelo, ele é salvo como um arquivo *joblib*, este é o arquivo que a API irá receber e utilizar para a comunicação entre a IA e a Unity.

4.2.1 Detalhes sobre a implementação do KNN

O presente trabalho foi desenvolvido no ambiente Jupyter Notebook, uma platorma de programação em Python, que facilita o trabalho de implementação de código, e torna a visualização do próprio código e resultados muito mais clara e dinâmica.

Um ponto importante a ser salientado, é que todos os dados antes do treino foram normalizados e transformados, usando a biblioteca *Standard Scaler*. Isso se deve ao fato de muitos valores de features que são medidos em diferentes escalas não contribuem igualmente para o modelo em treinamento, e no final, após ele ser treinado, o modelo pode acabar criando um certo bias. Então uma maneira de lidar com esse problema em potencial das features, a padronização (μ=-1, σ=1) é foi usada antes do treinamento.

$$z = \frac{x - \mu}{\sigma}$$

Figura 16 – A fórmula aplicada para a padronização dos dados. Fonte: Própria

	timeSpent	percentKills	mapSize	enemyAmount	enemyDensity	interactionAmount	conversationMaterial	difficulty	percentItems	totalLifeLost
0	8	0.076923	0	0	0	0	0	0	0.142857	0
1	14	0.000000	0	0	0	0	0	0	0.000000	10
2	76	0.384615	2	1	3	0	0	0	1.000000	30
3	45	0.000000	2	2	3	2	3	3	0.555556	35
4	52	0.105263	2	3	3	1	0	3	0.500000	75
...	...	...	...	...	...	...	...	...	...	...
165	46	0.368421	0	0	0	0	0	0	0.272727	45
166	219	0.052632	0	0	0	0	0	0	0.714286	50
167	116	0.000000	0	0	0	0	0	0	0.625000	35
168	203	0.052632	0	0	0	0	0	0	0.500000	10
169	174	0.052632	0	0	0	0	0	0	0.500000	60

Figura 17 – Dados antes da transformação. Fonte: Própria

```
array([[-0.568682  , -0.94136074, -1.2184914 , ..., -1.09658747,
        -1.45623292, -1.39403125],
       [-0.53286505, -1.21162009, -1.2184914 , ..., -1.09658747,
        -1.97880461, -1.1522095 ],
       [-0.16275647,  0.13967665,  0.24026591, ..., -1.09658747,
         1.67919828, -0.66856601],
       ...,
       [ 0.07602325, -1.21162009, -1.2184914 , ..., -1.09658747,
         0.3074472 , -0.54765513],
       [ 0.59536915, -1.0267058 , -1.2184914 , ..., -1.09658747,
        -0.14980316, -1.1522095 ],
       [ 0.42225385, -1.0267058 , -1.2184914 , ..., -1.09658747,
        -0.14980316,  0.05689923]])
```

Figura 18 – Dados após a tranformação. Fonte: Própria

Onde na figura 17 temos todos os dados originais usado para o treinos, e após a aplicação da fórmula na figura 16 em todos os valores originais, tempos os resultados na figura 18, contendo todos os dados já padronizados, prontos para a realização do treinamento.

4.3 A Integração com a API

A Unity utiliza por padrão a linguagem C# (C-sharp) para a implementação de seus scripts dentro do jogo, e atualmente o suporte nativo para a linguagem Python dentro da Unity não existe numa versão muito estável, além de algumas limitações, portanto foi decidido implementar uma API para a realizar a comunicação entre Unity e IA. Essa API foi desenvolvida usando a linguagem Python, utilizando a framework Flask. Como já mencionado anteriormente, após a API receber o modelo treinado da IA no formato joblib, quando ela receber os valores de entrada provenientes da jogada de um jogador na Unity, o modelo importado vai ler esses valores, e vai retornar para a Unity um JSON com os valores da distância euclidiana, e os indices dos mapas, de acordo com a figura 19.

Na Unity, é feita uma cópia dos dados correspondentes a atual jogada do jogador que é então enviada para a planilha em um array. Este array é convertido para o formato string e então é enviado para a API, essa conversão é necessária pois a framework Flask, usada para a implementação da API, requer entradas em formato de string. Já dentro da API essa string é convertida para valores em float, e após todas essas conversões, o modelo da IA dentro da API lê esses valores, e retorna um JSON no formato da figura 16.

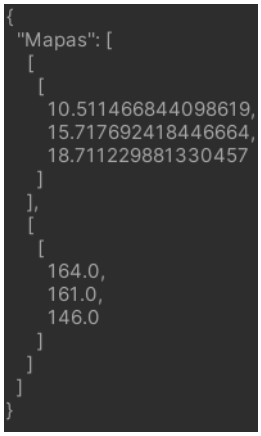

Figura 19 – Exemplo de um JSON retornado pela API após receber os dados de entrada. Fonte: Própria

Sendo o primeiro array contém a distância euclidiana de todos os mapas mais próximos em relação aos dados de entrada. O segundo array representa os indices na planilha de dados usada para o treino.

4.4 Limitações e Dificuldades

Destacando algumas limitações gerais encontradas ao longo do projeto, sendo a maior delas o próprio dataset já mencionado, onde não existe um nível de variedade de dados bom o suficiente, já que não foi possível testar o jogo com mais alunos, devido a isso o modelo treinado pode não vir a ser o mais preciso possível, o que vai afetar diretamente a parte dos resultados que virá mais a frente.

Outra limitação que se provou problemática ao longo do desenvolvimento é em relação ao formato da saida dos dados do modelo de machine learning. Usando a biblioteca scikit-learn, o formato do retorno padrão do KNN é o da imagem a seguir:

```
(array([[1.85735078, 1.8590083 , 1.93900729]]), array([[148,   6,  11]], dtype=int64))
```

Figura 20 – Exemplo de formato do retorno do algoritmo KNN implementado nesse trabalho. Fonte: Própria

Como já citado anteriormente, devido ao modelo ter um formato de saida limitado, não foi possível fazer a recomendação de forma direta, já que ele não retorna a própria seed em si, mas somente o index do mapa, ou a posição dele no dataset, a figura 20 demonstra isso, sendo o segundo array aquele que contém os indexes dos mapas recomendados.

Portanto foi necessário criar uma cópia do arquivo CSV do dataset, mas com essa cópia contendo somente a coluna com todas as seeds e seus respectivos indexes, então esse CSV foi importado dentro da plataforma Unity. Em adição a esse import, um novo script foi criado onde ele vai ler somente o primeiro valor do segundo array no

JSON retornado pela API, esse valor corresponde ao index do mapa recomendado, assim que o valor é lido, o CSV é percorrido até encontrar a seed correspondente ao indice lido, após isso essa seed é recomendada, como demonstrado na figura 15.

4.5 Decisões sobre o Projeto

Recapitulando algumas decisões do sobre o projeto: Foi utilizado o algoritmo KNN para a recomendação dos mapas gerados proceduralmente, o treino do algoritmo se deu com base no dataset gerado a partir de cada jogada do jogador. O dataset em questão contém ao todo mais de 30 features, sendo essas capturadas atráves da rodada do jogador e através também das questões do questionário, mas dessas 30 features somente 10 foram selecionadas para o treino, essa escolha se deu através de um heatmap para verificar a correlação entre essas features.

Depois da seleção das features e do treinamento do KNN ser finalizado, foi implementado uma API utilizando a framework Flask, essa API serve para realizar a comunicação entre o KNN recém treinado, e o jogo procedural na Unity. A escolha dessa framework se deu atráves da tentativa e erro, pois depois de algumas tentativas de integrar o projeto acabarem em falhas, a facilidade que o Flask proporcionou ao realizar a integração foi o ideal para esse projeto.

5 TESTES E RESULTADOS

Neste capítulo será apresentado os testes realizados e os resultados obtidos a partir da comparação dos mapas recomendados com os mapas de entrada.

5.1 Metodologia dos Testes

Para a realização dos testes, o modelo foi criado a partir da utilização de um conjunto de entrada/features selecionado com o uso da matriz de correlação, ou heatmap, a utilização deste heatmap ajudou na visualização de quais features seriam mais relevantes através da sua correlação entre si. Após a definição do conjunto, o jogo foi executado 30 vezes sobre esse grupo de features e após cada jogada, o mapa original, proveniente das jogadas, e os mapas recomendados para cada uma dessas jogadas foram postas em tabelas e a partir delas foram gerados diversos gráficos que serão apresentados logos mais a frente, para assim facilitar a visualização dos dados e a explicação sobre eles.

5.2 Conjunto Selecionado

O conjunto selecionado conta com features que representam em específico, os valores referentes ao tempo que o jogador demorou para completar o mapa, a razão entre número de inimigos derrotados no mapa, com o número total de inimgos disponíveis, os valores que correspondem as questões do questionário que perguntam para o jogador o que ele achou do tamanho do mapa, da quantidade de inimigos disponíveis, sobre o quão agrupados esses inimigos estavam, sobre o quão o grau de interação esse jogo ofereceu pra ele, sobre a disponibilidade do jogador para conversar sobre o projeto.

Também valores referentes a o que o jogador achou da dificuldade do mapa, a razão entre número de items coletados mapa no mapa, com o número total de items disponíveis, e por último a quantidade vida perdida pelo jogador ao enfrentar os inimigos. Como já citado no tópico 4.2, duas features selecionadas não fazem parte do treino do modelo, mas são anexadas de volta ao conjunto após o treinamento ser

concluído.

Abaixo, na figura 21, está o heatmap do conjunto citado.

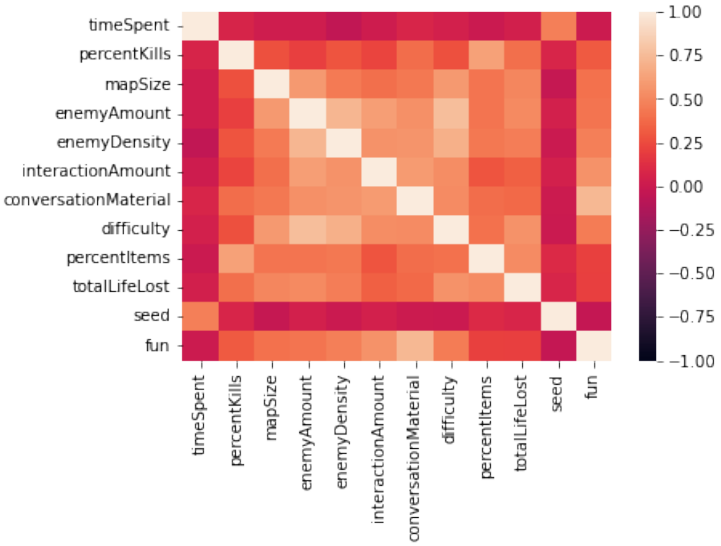

Figura 21 – Heatmap mostrando as features seleciondas e suas relações. Fonte: Própria

Ao análisar o heatmap, é possível notar que as features selecionadas estão majoritariamente correlacionadas, é possível notar isso com os quadrados de cores alaranjadas até o quadrados de tons mais claros, que representam que essas duas features estão mais correlacionadas. E em contraste com os tons mais avermelhados e escuros, representam uma correlação mais fraca entre duas features. Em especial há duas features que destoam em comparação com o resto, a primeira é a seed, que como já foi explicado, é só um valor numérico que é gerado aleatóriamente com o mapa, portanto não há nenhuma relação coerente com as outras features.

A outra é a timeSpent, que surpreendentemente não tem uma correlação com as outras features, apesar de ser uma que representa um aspecto claro de como o jogador jogou naquele mapa, mas é sempre importante frisar que especialmente nesse caso, mais ainda que o caso da seed, correlação não necessariamente implica em causalidade.

5.3 Gráficos e Discussão dos resultados

Ao longo dos testes, aconteceu um caso interessante, onde 5 mapas de entrada, ou seja, 5 dos 30 citados, recomendaram o mesmo mapa, então aqui será focado mais nesse caso, pois este é um bom exemplo para demonstrar o desempenho do algoritmo e para comparar os mapas de entrada com o recomendado.

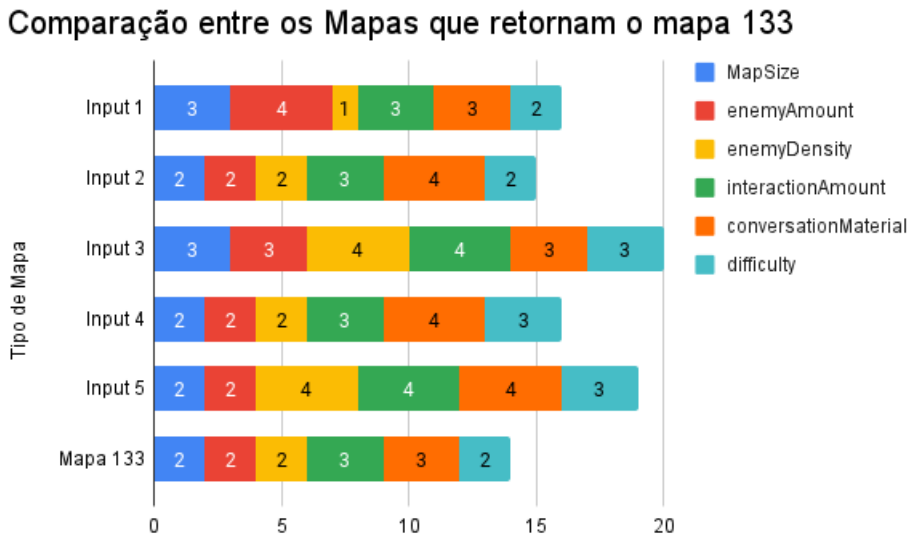

Figura 22 – Comparação entre os Mapas de entrada onde o KNN recomenda o mapa 133. Fonte: Própria

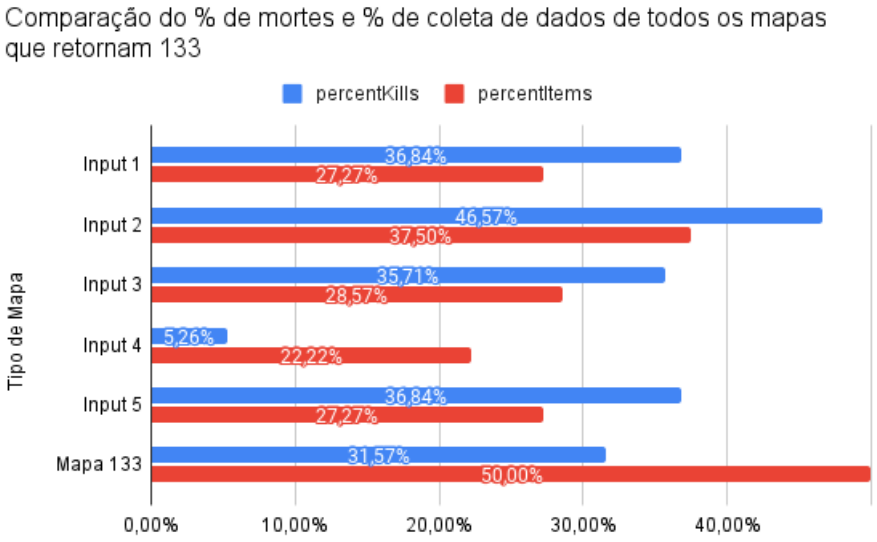

Figura 23 – Comparação entre o percentual de mortes e percentual de items coletados dos mapas onde o KNN recomenda o mapa 133. Fonte: Própria

Aqui temos a primeira comparação, aonde todos os mapas inputs recomendam o mapa 133, lembrando que 133 é o indice desse mapa no dataset, e para fins de identificação aqui, esse será o seu nome. Como já foi explicado, a métrica usada pelo KNN é a distância euclidiana entre dois pontos, que aplicado neste contexto, os dois pontos seriam dois mapas.

Na figura 22, podemos notar que o Input 2 é o mais semelhante ao 133, e o Input 3 é o mais distante entre os que recomendam, isso se deve a distância entre esses mapas como a figura 24 mostra com mais detalhes

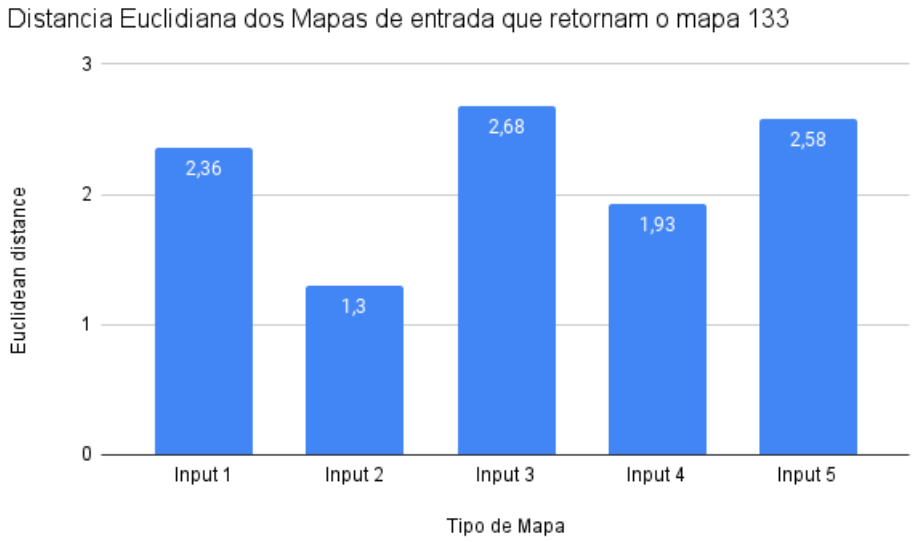

Figura 24 – Distância Euclidiana de todos os mapas de entrada que o KNN recomenda o 133. Fonte: Própria

Para colocar em perspectiva as distâncias dos 5 Inputs deste caso, sobre os 30 testes realizados ao todo, foram selecionados a menor e a maior distância encontradas no geral.

Dentre os mapas de entrada, a menor distância dentre esses 5, sofreu um acréscimo de 27.45% em relação a menor encontrada no geral, e a maior dentre os 5, sofreu um decréscimo de 11.19% em relação a maior encontrada no geral. Esses resultados podem ser considerados medianos, pois demonstram que houve um acréscimo relativamente significativo em relação ao mapa mais semelhante encontrado dentre os 30 testes gerais, e o mesmo vale para para a maior distância, onde houve um decréscimo pequeno se comparado com o pior resultado encontrado ao todo. Mas um dos motivos para isso, se devem ao fato do dataset curto e pouca variabilidade de dados, porém estes problemas serão discutidos mais a frente.

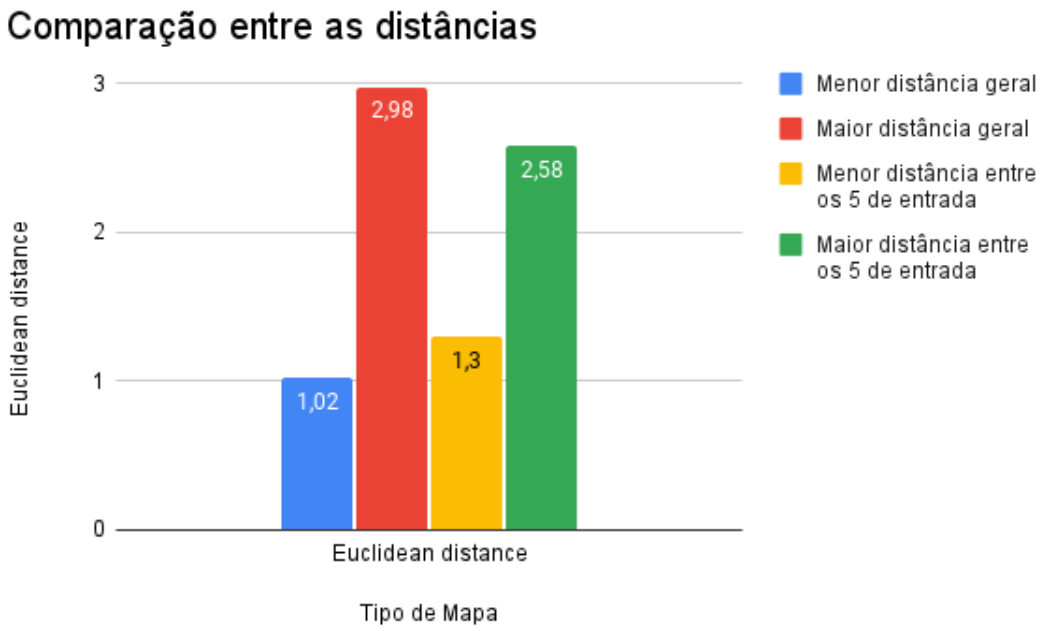

Figura 25 – Menor e Maior distâncias. Fonte: Própria

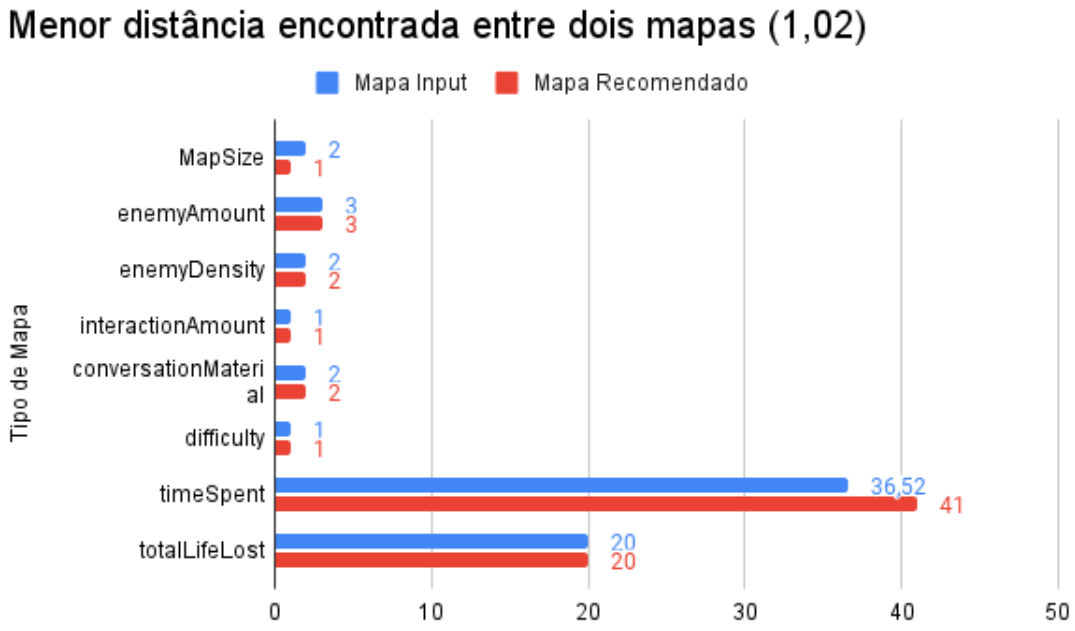

Figura 26 – Mapas com a menor distância encontrada. Fonte: Própria

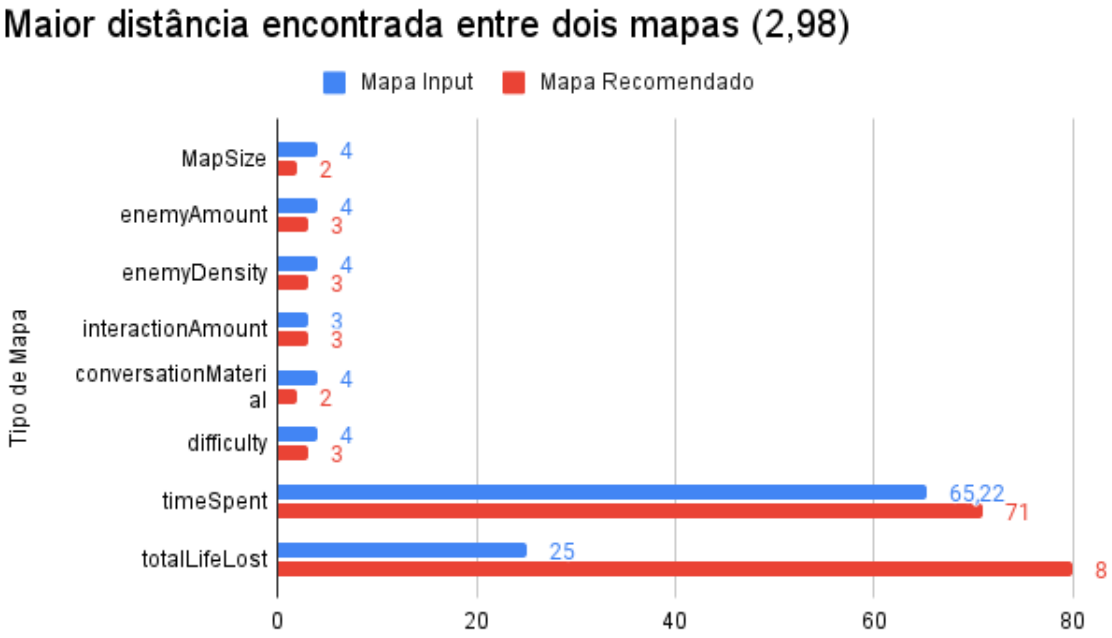

Figura 27 – Mapas com a menor distância encontrada. Fonte: Própria

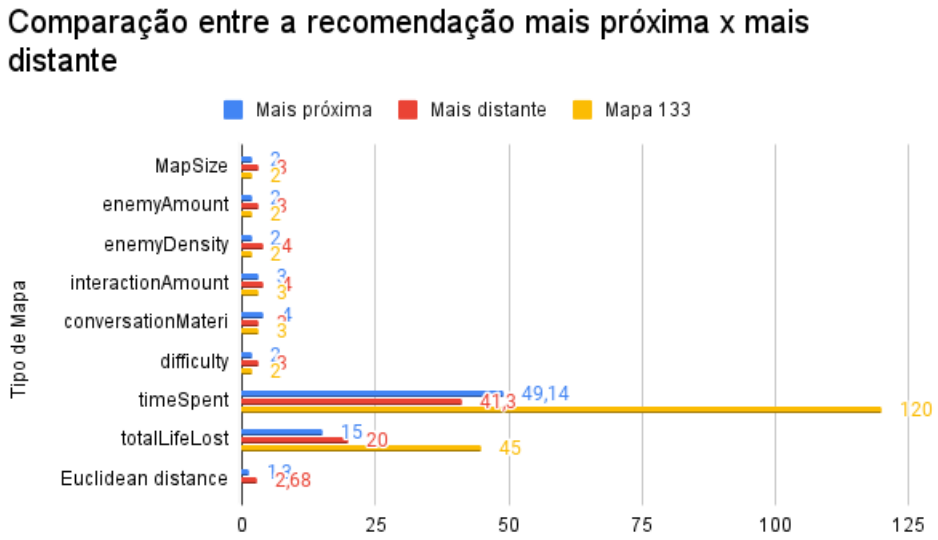

Figura 28 – Comparação do recomendado mais próximo com o mais distante. Fonte: Própria

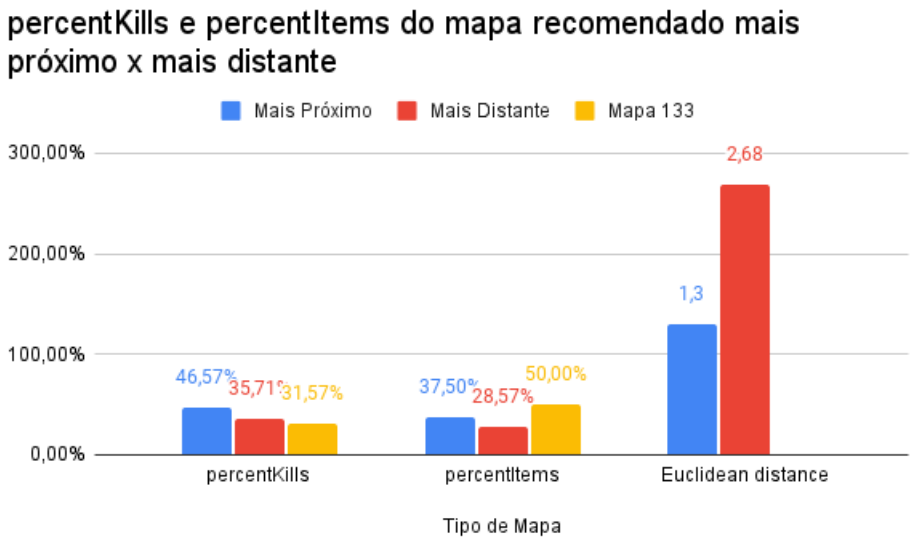

Figura 29 – percentKills e percentItems do mais recomendado mais próximo e do mais distante.
Fonte: Própria

Sobre os resultados dos mapas de entrada em si, fica claro perceber o desempenho do Input 2 em relação ao mapa recomendado no que se refere as features da figura 22. No entanto, na figura 23, onde é comparado as features percentKill e percentItems, é mais díficil afirmar qual seria o mapa mais próximo, devido a maior variedade de resultados, mas é interessante notar que o Input 4, onde ele na primeira comparação é o segundo mapa mais semelhante ao recomendado, é o que possui os dados mais discrepantes aqui, o que possa significar que apesar de ele ter a segunda menor distância, não necessariamente significa que ele é a recomendação mais precisa, e sim a recomendação mais plausível dado ao contexto atual, com os dados utilizados aqui, e isso não vale somente para o caso do Input 4, mas para todos os testados.

Para explicar essa divergência de resultados, seja nas distâncias, quanto nas features em si, o que resultam numa não tão alta precisão na hora de recomendar um mapa, se devem ao fato de que, primeiro, o dataset utilizado não possui um número de amostrar grande o suficiente de amostrar para a realização do treino do algoritmo. O dataset conta com apenas 212 amostras, um número relativamente baixo para se obter resultados verdadeiramente bons e precisos.

E segundo, é que além do dataset em questão ser pequeno, as amostras em si possuem pouca variabilidade de dados, ou seja, muito dos dados de entrada eram iguais e também na pouca váriação de respostas disponíveis nos questionários, isso levou ao algoritmo a aprender alguns padrões errados, que simplesmente eram repetições, e para esse tipo de modelo de ML, quanto mais diversificado e maior, melhor é o desempenho do mesmo.

Mas mesmo com estas limitações, sobre o conjunto de testes apresentados, o

trabalho apresentou bons resultados, onde mesmo o mapa mais distante, ainda pode oferecer para o jogador uma experiência semelhante com a de outros jogadores, ou até a dele mesmo em jogadas anteriores, assim customizando o estilo de gameplay da pessoa, para algo mais exclusivo.

6 CONCLUSÃO

A PCG é um método que vem crescendo cada vez mais no mercado de jogos modernos por vários fatores foram discutidos aqui, sendo os principais a ótimização de tempo, corte de gastos, variadade de conteúdo disponível, processo de implementação mais simples. A proposta desse trabalho foi a utilização desse metodo, integrado ao um modelo de ML supervisionado, sendo este o KNN.

E apesar dos testes demonstrados terem sido prejudicados pelas limitações já mencionadas, acredito que este trabalho obteve resultados satisfatórios com os dados disponíveis. Porém, também penso que uma evolução natural deste projeto seja a capacidade de filtrar aquilo que o jogador não gosta sobre as suas jogadas, o que não foi viável realizar neste projeto devido a pouca variabilidade dados, assim podendo até mesmo criar perfis para estes jogadores, sendo eles perfis de jogadores que preferem explorar, descobrir segredos, ou perfis de jogadores que preferem o combate, e que gostam de acabar com todos os inimigos do jogo. Dada a realização de tais perfis, e focalizando o uso do modelo KNN, independentes um do outro, serão obtidos dados ainda mais únicos e exclusivos, gerando assim resultados bem melhores.

Considero como a principal contribuição deste trabalho a perspectiva de utilizar o melhor que a PCG têm a oferecer, com a utilização do KNN para poder oferecer para o usuário uma experiência customizável, baseada somente em como esse jogador prefere jogar o seu jogo.

6.1 Trabalhos Futuros

Este projeto conta com uma boa perspectiva para trabalhos futuros, como por exemplo a implementação de um sistema que automatize os testes do jogo, que já está sob desenvolvimento de um colega no projeto, este trabalho poderá resolver algumas limitações encontradas aqui, como a geração de um dataset maior, e com maior variabilidade de dados, para o caso de outros algoritmos de ML venham a ser implementados sobre este jogo no futuro.

Outro trabalho já em desenvolvimento é o detecção de padrões de gameplay, onde

este poderá fazer o uso do questionário ao final de uma rodada obseleto, o uso do questionário foi um processo bem debatido durante a realização do projeto, pois ele de certa forma, quebrava a imersão do jogador ao fazer perguntas sobre o jogo para ele. Também é possível criar um trabalho onde sistemas de acessibilidade poder ser criados no jogo, assim abrangendo o leque de pessoas que poderão jogar este jogo, e quanto mais pessoas jogando, mais feedback é gerado, e dados com mais qualidade e quantidade serão eventualmente coletados.

Também é possível realizar trabalhos em conjunto com outras áreas, como a de Design, por exemplo, onde toda a interface, e identidade do jogo podem ser elevadas a um novo patamar, ou um trabalho em conjunto com o pessoal dos cursos de música, para a criação de trilhas sonoras exclusivas para o jogo, assim melhorando a experiência geral e imersão do mesmo, com isso, tornando o jogo mais atrativo para o público em geral.

REFERÊNCIAS

Ashish Mehta. **K-Nearest Neighbors (KNN) Algorithm For Machine Learning**. Disponível em: https://ai.plainenglish.io/k-nearest-neighbors-knn-algorithm-for-machine-learning-ab6f17df4a7f. Acesso em: 4 de junho de 2022.

AWS. **O que é uma API?** Disponível em: https://aws.amazon.com/pt/what-is/api/. Acesso em: 31 de Outubro de 2022.

COPPERWAITE, M.; LEIFER, C. **Learning flask framework**. [S.l.]: Packt Publishing Ltd, 2015.

DAHLSKOG, S.; TOGELIUS, J. Patterns and procedural content generation: Revisiting mario in world 1 level 1. In: FIRST WORKSHOP ON DESIGN PATTERNS IN GAMES, 2012. **Proceedings...** [S.l.: s.n.], 2012. p.1–8.

HAAS, J. A history of the unity game engine. **Diss. WORCESTER POLYTECHNIC INSTITUTE**, [S.l.], 2014.

HACKELING, G. **Mastering Machine Learning with scikit-learn**. [S.l.]: Packt Publishing Ltd, 2017.

IBM. **What is the k-nearest neighbors algorithm?** Disponível em: https://www.ibm.com/topics/knn?mhsrc=ibmsearch_a&mhq=KNN. Acesso em: 3 de Junho de 2022.

IBM. **What is machine learning?** Disponível em: https://www.ibm.com/cloud/learn/machine-learning#toc-machine-le-K7VszOk6. Acesso em: 3 de Junho de 2022.

IBM. Disponível em: https://www.ibm.com/cloud/learn/data-labeling. Acesso em: 3 de Junho de 2022.

JACOBSON, J.; LEWIS, M. GAME ENGINES IN SCIENTIFIC RESEARCH. **Communications of The ACM**, [S.l.], v.45, p.27–31, 2002.

Jogorama. **Spelunky 2**. Disponível em: https://jogorama.com.br/jogos/xbox-one/9705/spelunky-2/. Acesso em: 6 de Novembro de 2022.

LEITE, L. **Introdução à História dos Jogos Eletrônicos**. 2003. Tese (Doutorado em Ciência da Computação) — Dissertação PUC-RIO.

PADILHA, R. F. **ANALISANDO O ENGAJAMENTO DE JOGADORES ATRAVÉS DE MAPAS PROCEDURAIS**. [S.l.]: Universidade Federal de Pelotas, 2022.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **the Journal of machine Learning research**, [S.l.], v.12, p.2825–2830, 2011.

Pedro Barros. **Aprendizagem de Maquina**: Supervisionada ou Não Supervisionada? Disponível em: https://medium.com/opensanca/aprendizagem-de-maquina-supervisionada-ou-n˜ao-supervisionada-7d01f78cd80a. Acesso em: 4 de Novembro de 2022.

SHAKER, N.; TOGELIUS, J.; NELSON, M. J. **Procedural content generation in games**. [S.l.]: Springer, 2016.

SHAKER, N.; YANNAKAKIS, G. N.; TOGELIUS, J. Towards player-driven procedural content generation. In: COMPUTING FRONTIERS, 9., 2012. **Proceedings...** [S.l.: s.n.], 2012. p.237–240.

SUMMERVILLE, A. et al. Procedural content generation via machine learning (PCGML). **IEEE Transactions on Games**, [S.l.], v.10, n.3, p.257–270, 2018.

The CPUshack Museum. **The Electronika MK1 red3 PDP-11 Chipset and Tetris**. Disponível em: https://www.cpushack.com/2015/09/06/the-electronika-mk1-red3-pdp-11-chipset-and-tetris/. Acesso em: 6 de Novembro de 2022.

Tim Brooks. **Roguelikes**: A Unique Challenging Spin On The RPG Genre. Disponível em: https://www.makeuseof.com/tag/roguelikes-a-unique-challenging-spin-on-the-rpg-genre/. Acesso em: 6 de Novembro de 2022.

TOGELIUS, J.; YANNAKAKIS, G. N.; STANLEY, K. O.; BROWNE, C. Search-based procedural content generation: A taxonomy and survey. **IEEE Transactions on Computational Intelligence and AI in Games**, [S.l.], v.3, n.3, p.172–186, 2011.

VG Legacy. **Tennis for Two**. Disponível em: https://vglegacy.com/hardware/tennis-for-two/. Acesso em: 4 de Novembro de 2022.