**Relatório de Projeto**

**Programação**

**Avaliação Periódica 3**

**Autores:**

Karine Aparecida Braga Florêncio

Nuno

**Data:** Janeiro de 2023

Resumo

Com o objetivo de a criar uma aplicação em Java para auxiliar uma oficina de automóveis foi desenvolvido um código com dezessete classes públicas de forma a praticar todo o conhecimento aprendido em nas aulas de programação. Durante o processo de criação foram realizados testes unitários de modo a encontrar e eliminar erros de programação. São cumpridos todos os requisitos inicialmente apresentados. Apesar de bem consolidada a Linguagem Java é observada que há formas mais sucintas de desenvolver o mesmo projeto, por exemplo, usando hierarquia e polimorfismo.

**Palavras-chave**

Algoritmos, Aplicação, Classes, Jogo Pygame, Jogo 2D, Jogo de tiro, Linguagem Pyhton, Manual do Utilizador, Menu, Métodos, Multiplayer, NPC, Pygame para iniciantes, Python, Singleplayer.

**Índice**

[Resumo ii](#_Toc124085198)

[Lista de Tabelas iv](#_Toc124085199)

[Lista de Figuras iv](#_Toc124085200)

[Lista de Acrónimos v](#_Toc124085201)

[1. Introdução 1](#_Toc124085202)

[2. Objetivos e Metodologias 2](#_Toc124085203)

[2.1. Ferramentas e Tecnologias 2](#_Toc124085204)

[2.2. Planeamento 4](#_Toc124085205)

[2.3. Como Testar 5](#_Toc124085206)

[3. Trabalho Desenvolvido 6](#_Toc124085207)

[3.1. Classes 6](#_Toc124085208)

[3.2. Algoritmos 10](#_Toc124085209)

[3.3. Procedimentos de Teste 11](#_Toc124085210)

[4. Conclusões 12](#_Toc124085211)

[4.1. Forças 12](#_Toc124085212)

[4.2. Limitações 12](#_Toc124085213)

[4.3. Trabalho Futuro 12](#_Toc124085214)

[5. Referências 13](#_Toc124085215)

[6. Anexos 14](#_Toc124085216)

[6.1. Manual do Utilizador 14](#_Toc124085217)

# Lista de Tabelas

[Tabela 1 - Requisitos cumpridos. 6](#_Toc123665263)

# Lista de Figuras

[Figura 1 - Diagrama de classes 7](#_Toc123747491)

[Figura 2 - Inicialização da aplicação. 18](#_Toc123747492)

[Figura 3 - Informações ao aceder a aplicação. 18](#_Toc123747493)

[Figura 4 - Autenticação. 19](#_Toc123747494)

[Figura 5 - Adicionar Utilizador. 19](#_Toc123747495)

[Figura 6 - Listar Utilizadores. 20](#_Toc123747496)

[Figura 7 - Ordenar Utilizadores. 20](#_Toc123747497)

[Figura 8 - Pesquisar Utilizador. 21](#_Toc123747498)

[Figura 9 - Remover Utilizador. 21](#_Toc123747499)

[Figura 10 - Alterar meu nome. 22](#_Toc123747500)

[Figura 11 - Alterar minha senha. 22](#_Toc123747501)

[Figura 12 - Consultar ações do sistema. 22](#_Toc123747502)

[Figura 13 - Gerir Veículos. 23](#_Toc123747503)

[Figura 14 - Adicionar Veículo. 24](#_Toc123747504)

[Figura 15 - Pesquisar Veículo. 24](#_Toc123747505)

[Figura 16 - Listar todos os Veículos. 25](#_Toc123747506)

[Figura 17 - Ordenar Veículos. 25](#_Toc123747507)

[Figura 18 - Listar Veículos de um Cliente. 26](#_Toc123747508)

[Figura 19 - Listar Veículos com serviço em aberto. 26](#_Toc123747509)

[Figura 20 - Listar Veículos que possuem uma peça. 27](#_Toc123747510)

[Figura 21 - Listar Veículos servidos após ano x. 27](#_Toc123747511)

[Figura 22 - Listar Veículos que gastou mais de x horas. 28](#_Toc123747512)

[Figura 23 - Gerir Veículo. 28](#_Toc123747513)

[Figura 24 - Adicionar Serviço. 29](#_Toc123747514)

[Figura 25 - Listar Serviços. 30](#_Toc123747515)

[Figura 26 - Gerir Serviço. 30](#_Toc123747516)

[Figura 27 - Adicionar Peça ao Serviço. 31](#_Toc123747517)

[Figura 28 - Adicionar Mecânico. 31](#_Toc123747518)

[Figura 29 - Listar Peças. 32](#_Toc123747519)

[Figura 30 - Listar Mecânicos. 32](#_Toc123747520)

[Figura 31 - Alterar custo. 32](#_Toc123747521)

[Figura 32 - Alterar descrição. 33](#_Toc123747522)

[Figura 33 - Gerir Peças. 33](#_Toc123747523)

[Figura 34 - Adicionar Peça. 34](#_Toc123747524)

[Figura 35 - Pesquisar Peça. 34](#_Toc123747525)

[Figura 36 - Listar peças com a mesma designação. 35](#_Toc123747526)

[Figura 37 - Listar peças de uma marca. 35](#_Toc123747527)

[Figura 38 - Listar peças de um fornecedor. 36](#_Toc123747528)

Lista de Acrónimos

**fps** frame per second - quantidade de imagens exibidas na tela em apenas um segundo.

**NPC** Non-Player Character - é um personagem de jogo eletrônico que não pode ser controlado por um jogador.

**SDL** Simple DirectMedia Layer - é uma biblioteca de desenvolvimento de plataforma cruzada projetada para fornecer acesso de baixo nível a áudio, teclado, mouse, joystick e hardware gráfico via OpenGL e Direct3D.

# Introdução

Este trabalho pretende desenvolver uma aplicação com o objetivo de auxiliar uma oficina automóvel na gestão dos serviços realizados a veículos. Através da implementação de uma aplicação completa em linguagem **Java**, sem componente gráfica para interação com o utilizador, recorrendo ao paradigma de Programação Orientada a Objetos. Assim como o desenvolvimento e aplicação de conhecimentos e competências relacionadas com as temáticas abordadas na disciplina de programação, com especial incidência para:

* programação orientada a objetos;
* estruturas de dados;
* manipulação de ficheiros para armazenamento de dados de forma persistente;
* bem como o desenvolvimento de interfaces para interação com o utilizador (em modo texto).

# Objetivos e Metodologias

O principal objetivo deste projeto é desenvolver um jogo 2D com a biblioteca Pygame da linguagem Python. Em grupo decidimos que os objetivos que queríamos alcançar com este projeto era:

* Implementação de NPC’s .
* Animações para os personagens e elementos do jogo.
* Aumentar a dificuldade e ganho do jogador gradativamente.
* Criar um menu de comunicação amigável e intuitivo para o jogador.
* Armazenamento da maior pontuação.
* Adicionar uma funcionalidade multiplayer ao jogo, permitindo que jogadores joguem em parceria em múltiplas telas.

Esses objetivos devem ajudar a guiar o desenvolvimento do seu projeto e garantir que ele tenha uma boa quantidade de desafio, criatividade e variedade de técnicas usadas.

Para a realização deste trabalho necessitamos de 2 semanas de aulas e horário de atendimento com o professor, além de extras horas, totalizando uma média de 20 horas semanais cada desenvolvedor.

## Ferramentas e Tecnologias

### Colaboração

Para trabalhar em cooperação usamos as ferramentas Git e GitHub.

**Git** é um sistema de controle de versão de código-fonte. Ele permite que se rastreie alterações em arquivos de código e reverta para versões anteriores se necessário. Git é amplamente utilizado por desenvolvedores de software para controlar o código-fonte de projetos de software, mas também pode ser usado para rastrear alterações em qualquer tipo de arquivo.

**GitHub** é um site que oferece serviços baseados em Git, incluindo armazenamento de código-fonte, rastreamento de problemas e gerenciamento de projetos. Ele é usado por milhões de desenvolvedores em todo o mundo para colaborar em projetos de software e hospedar projetos públicos e privados.

Para começar a usar Git e GitHub, é preciso instalar o Git em seu computador e criar uma conta no GitHub. Depois disso, pode criar um novo repositório (um lugar para armazenar o código) no GitHub e usar os comandos do Git para enviar (commit) alterações do código para o repositório. Também podemos usar o GitHub para trabalhar em projetos com outras pessoas, fazendo "pull requests" (solicitações de integração) para incluir alterações no código de outras pessoas em um repositório.

Aqui estão alguns dos comandos Git mais comuns para gerenciar o código:

* **git init**: Inicializa um novo repositório Git em um diretório existente.
* **git clone**: Clona um repositório Git existente em um novo diretório.
* **git add .**: Adiciona todos os arquivos ao índice (a área do Git onde as alterações são rastreadas).
* **git commit –m ‘mensagem’**: Salva as alterações no índice em um commit (uma versão do código) com o nome recebido entre aspas.
* **git push**: Envia os commits de um repositório local para um repositório remoto (como o GitHub).
* **git pull**: Puxa os commits de um repositório remoto para um repositório local.

Para trabalhar com branch (ramificações) em um projeto:

* **git branch**: Lista as branches do repositório ou cria uma nova branch.
* **git checkout**: Muda para uma branch existente ou cria uma nova branch e muda para ela.
* **git merge**: Mescla duas branches juntas.

Esses são apenas alguns exemplos dos muitos comandos que podemos usar com o Git. Existem muitos outros comandos disponíveis e cada um tem suas próprias opções e parâmetros. Para obter mais informações sobre os comandos Git, consulte a documentação do Git ou um tutorial online. **[III][IV][V]**

O projeto realizado por no desenvolvimento deste relatório pode ser encontrado ao acessar <https://github.com/MrNoino/MGC-TP3.git>

### Desenvolvimento

Neste trabalho foram usadas as seguintes bibliotecas para a linguagem de programação Python:

As bibliotecas **sys**, **random**, **time**, **datetime**, **\_thread**, **pickle** e **socket** são bibliotecas padrão do Python o que significa que elas já estão disponíveis para usar sem a necessidade de instalar nada, só precisa incluir um *import* no início do código para poder usá-las. A biblioteca **sys** fornece acesso a alguns objetos e funções relacionados ao interpretador Python. A biblioteca **random** fornece funções para gerar números aleatórios. A biblioteca **time** fornece funções para trabalhar com tempo e data. A biblioteca **datetime** fornece classes para trabalhar com data e hora, é muito poderosa e é a biblioteca recomendada para qualquer tarefa de manipulação de data e hora no Python. A biblioteca **\_thread** fornece um módulo de baixo nível para trabalhar com threads em Python, é menos poderosa e mais complexa de usar do que outras bibliotecas de threads disponíveis para o Python, como o threading. A biblioteca **pickle** fornece um módulo para serializar e desserializar objetos Python, significa que você pode usar o pickle para converter objetos Python em uma representação de sequência de bytes, que pode ser armazenada em um arquivo ou transmitida pela rede, e depois recriar os objetos a partir dessa representação. A biblioteca **socket** fornece uma interface de baixo nível para criar e usar sockets de rede, podendo usa-la para criar clientes e servidores que se comunicam através da rede usando diferentes protocolos de rede, como o TCP e o UDP.**[I]**

**Pygame** é uma biblioteca de jogos para Python. Ela fornece uma ampla variedade de recursos para criar jogos 2D, incluindo gráficos, sons, entrada do usuário e sistemas de colisão. A Pygame é uma das bibliotecas mais populares para criar jogos em Python devido à sua simplicidade e à grande quantidade de recursos disponíveis. A Pygame adiciona funcionalidade à excelente biblioteca SQL, permitindo que você crie jogos completos e programas multimídia na linguagem python. A Pygame é altamente portátil e roda em quase todas as plataformas e sistemas operacionais. A instalação pode ser feita usando o gerenciador de pacotes Python pip. O Comando ***pip*** é automaticamente instalado após a instalação do Python e o comando instala tudo no ambiente Python. Abra o terminal e digite o seguinte comando: **[II]**

**pip install pygame**

## Planeamento

O projeto foi planejado por etapas de desenvolvimento, sendo elas e suas cargas horárias:

1. Adaptação Git/GitHub – 6 horas.
2. Criação das classes – 3 horas.
3. Gerir NPC’s, Threads – 13 horas.
4. Correção de desafios encontrados. – 8 horas.
5. Animações. – 4 horas.
6. Menu / Teclas de atalhos. – 3 horas.
7. Implementar servidor. – 15 horas.
8. Introduzir sons. – 2 horas.
9. Dados permanentes. – 4 horas.
10. Elaborar relatório – 10 horas.
11. Elaborar manual do utilizador – 4 horas.

## Como Testar

Os ficheiros necessários para que o jogo funcione são :

# Trabalho Desenvolvido

A aplicação é capaz de gerir Utilizadores, Veículos, Serviços e Estoque de peças de uma oficina. Isto é, adicionar, pesquisar, remover, listar são algumas das principais funcionalidades.

## Classes

### Entity

Representa um Sprite da aplicação.

##### Atributos:

**\_image** – Imagem do sprite.

**rect** – Objeto que armazena as coordenadas retangulares do sprite.

##### Métodos:

**setImage**(image) – Atualiza a imagem do sprite.

**getPostion**() – Fornece o ponto central do sprite.

**setPosition**(position, update) – Atualiza a posição do sprite de acordo com a position recebida alterada pelo update.

**move**(position) – Movimenta o sprite de acordo com o valor recebido.

**draw**(surface) – Imprime a imagem do sprite na posição que este se encontra.

### Enemy

Representa um NPC do jogo.

##### Atributos:

**\_speed\_animation** – velocidade em que a animação ocorre, quanto menor mais rápido.

**\_value** – valor usado para definir o index no array de imagens para realizar a animação de movimento do NPC.

**\_value\_dead** – valor usado para definir o index no array de imagens para realizar a animação de termino de um NPC.

**\_images** – array de imagens do NPC.

##### Métodos:

**setPosition**(position) – Atualiza a posição do NPC para 50 pixels a frente.

**getFinal**(): boolean – Verifica se um NPC chegou ao fim da tela, ganhando assim o jogo.

**move**(velocity) – Movimenta o NPC de acordo com o valor recebido e atualiza a animação.

**animatedead**(): boolean – Realiza a animação para finalizar o NPC.

### Player

Representa um jogador do jogo.

##### Atributos:

**\_username** – nome do jogador.

**\_score** – pontuação do jogador.

**\_moving** – define se é permitido a movimentação do jogador.

**\_speed\_animation** – velocidade em que a animação ocorre, quanto menor mais rápido.

**\_value** – valor usado para definir o index no array de imagens para realizar a animação de movimento do jogador.

**\_value\_shoot** – valor usado para definir o index no array de imagens para realizar a animação de movimento de tiro de um jogador.

**\_value\_dead** – valor usado para definir o index no array de imagens para realizar a animação de termino de um jogador.

**\_images** – array de imagens do jogador.

##### Métodos:

**shoot**(all\_shots) – Realiza um disparo caso o jogador pressionar o ‘SPACE’ e tiver tiros disponíveis, e verifica se um dos tiros passou do fim da tela e recoloca como disponível para o jogador.

**incrementScore**(quantity) – atualiza a pontuação do jogador com o valor recebido.

**notMove**() – tira a permissão do jogador se mover.

**move**() – Movimenta o jogador de acordo com a tecla precionada e atualiza a animação.

**animatedead**(): boolean – Realiza a animação para finalizar o jogador e devolve True para o jogo terminar.

### Shot

Representa um tiro disparado pelo jogador.

##### Atributos:

**\_speed\_animation** – velocidade em que a animação ocorre, quanto menor mais rápido.

**\_value** – valor usado para definir o index no array de imagens para realizar a animação de movimento do jogador.

**\_value\_dead** – valor usado para definir o index no array de imagens para realizar a animação de termino de um NPC.

**\_img** – array de imagens para animação do tiro.

**\_img\_dead** – array de imagens para realizar a animação de termino de um NPC.

##### Métodos:

**setPosition**(position) – Atualiza a posição do tiro para 30 pixels atras e 10 pixels acima.

**move**(velocity) – Movimenta o tito de acordo com o valor recebido e atualiza a animação.

**animatedead**(): boolean – Realiza a animação para finalizar o tiro.

### Network

Representa uma conexão network realizada pelo parte cliente do cliente-servidor.

##### Atributos:

**\_client** – cria um novo socket objeto.

**\_host** – host (endereço do servidor) que o cliente ira se conectar.

**\_port** – porta onde o servidor esta a escuta do cliente.

**\_addr** – junção de host e port.

##### Métodos:

**connect**(name,skin) – tenta realizar a conexão do cliente através do host e port ao servidor e recebe feedback e este for aceito é enviado nome e skin escolhida pelo utilizador retornando a resposta recebida.

**send**(data): boolean – tenta converter a data em um objeto pickle e envia o tamanho do objeto pickle seguido do objeto pickle, devolve *True* ao sucesso da operação.

**recv**(): pickle – tenta receber o tamanho dos dados e depois recebe os dados e retornando os em seguida.

**disconnect**() – termina a conexão entre cliente-servidor.

### Server

Representa uma conexão network realizada pelo parte cliente do cliente-servidor.

##### Atributos:

**\_client** – cria um novo socket objeto.

**\_host** – host (endereço do servidor) que o cliente ira se conectar.

**\_port** – porta onde o servidor esta a escuta do cliente.

**\_addr** – junção de host e port.

##### Métodos:

**connect**(name,skin) – tenta realizar a conexão do cliente através do host e port ao servidor e recebe feedback e este for aceito é enviado nome e skin escolhida pelo utilizador retornando a resposta recebida.

**send**(data): boolean – tenta converter a data em um objeto pickle e envia o tamanho do objeto pickle seguido do objeto pickle, delvove verdadeiro ao sucesso da operação.

**recv**(): pickle – tenta receber o tamanho dos dados e depois recebe os dados e retornando os em seguida.

**disconnect**() – termina a conexão entre cliente-servidor.

## Algoritmos

Os principais e mais comuns algoritmos da aplicação foram desenvolvidos de acordo com os apresentados durante as aulas de Multimédia e Computação Gráfica.

### game

Ficheiro de execução do jogo para multiplayer. Recolhe do utilizador o nome e skin do personagem, o host e port em que o cliente quer se conectar, cria a conexão através da classe **Network** e tenta se conectar. E fica a espera da resposta com seu número de ID.

### app

Ficheiro para execução do servidor. Inicializa e gere o servidor.

### utils

Armazena funções globais uteis para a aplicação.

##### Função:

**\_saveLog**(text) – tenta abrir para fazer apêndice, escrever o text recebido e fechar um ficheiro com nome “exceptions.log” onde está armazenado as exceções do jogo. Retorna *True* ao sucesso da ação.

### globals

Armazena variáveis globais da aplicação.

### game\_single

Ficheiro de execução do jogo para singleplayer.

Recolhe do utilizador a skin do personagem. Inicializa o **Pygame** e inicializa as fontes de texto e o configura o tamanho do display e a superfície do jogo. Invoca a função **initGame** que tem o objetivo de apresentar os gráficos de abertura do jogo e invoca a função **gameMenu** que tem o objetivo de apresentar um menu ao jogador onde este pode iniciar o jogo ou recomeçar o jogo invocando assim a função **game**, continuar o jogo da onde foi parado ou sair da aplicação. Esta função está a correr com o jogo e é invocada ao clicar “ESC” ou ao inicializar/finalizar o jogo.

A função **game** inicializa suas variáveis de imagem e som, inicializa o jogador, cria grupos de sprites necessários no jogo e os atualiza, inicializa variáveis de jogo, invoca a função **generateNPCS** com o objetivo atualizar o array com os NPC’s criados. Após é iniciado o ***game loop*** com 30 fps.

O ***game loop*** atualiza a barra de informações do jogador com a função **displayBackground**, movimenta e desenha todos os sprites do jogo, invoca o método de criação de tiro do jogador, verifica os NPC’s atingidos por tiros com a função **groupcollide** do módulo sprite e “termina” as “atividades” do NPC, assim como aumenta a pontuação do jogador, verifica se um NPC atingiu o fim da tela invocando o método **getFinal** da classe Enemy e verifica se um jogador atingiu um NPC com a função **groupcollide** do módulo sprite . Se essas duas verificações forem confirmadas, a função **gameOver** é invocada, exibindo a animação de fim de jogo e informações sobre o desempenho do jogador, como pontuação e hordas concluídas. O método **kill** é então invocado para todos os sprites do jogo e o jogo é retornado ao menu.

O ***game loop*** também verifica se todos os NPC foram finalizados, incrementando variáveis de jogo e invocando a função **start\_new\_thread** da biblioteca \_thread para que está rode a função **generateNPCS** para gerar uma nova onda de NPC's enquanto o jogo está em andamento. Esses NPC's ficam aguardando a conclusão da onda anterior antes de serem ativados no jogo e a geração de NPC's é novamente invocada.

Por ultimo o ***game loop*** verifica se o utilizador pressionou ‘ESC’ pausando o jogo pela função **gameMenu**.

## Procedimentos de Teste

Foram adotados diversos testes unitários não automatizados a vários componentes isolados da aplicação para a identificação de erros. Houveram testes unitários com intuito de testar cada funcionalidade da aplicação, comportamento dos métodos e classes da aplicação. E com diferentes valores de leitura, sendo algum destes não esperado inicialmente pelo programador e corrigidos ao longo da fase de testes.

# Conclusões

Após a concretização deste trabalho posso concluir que o conhecimento teórico e prático em relação a linguagem Java foram bem consolidados, contudo é necessária prática de outros componentes que não foram desenvolvidos neste projeto.

Existindo outros métodos para a realização deste trabalho, eu escolhi aplicar aquilo que foi desenvolvido em sala de aula e em consulta com o professor.

Por fim posso dizer que consegui cumprir com todos os requisitos e funcionalidades do trabalho.

Este projeto desenvolveu uma aplicação para auxiliar uma oficina de automóveis na gestão dos serviços realizados a veículos, assim como registos de peças, mecânico, clientes e veículos na sua “base de dados”. A aplicação não está pronta para acesso de vários threads, ou seja não é sincronizada para múltiplos acessos.

## Forças

O projeto foi desenvolvido a facilitar a interação com o utilizador, disponibiliza uma interface em modo texto onde o utilizador interage e controla a aplicação. As entradas de dados por parte do utilizador são testadas e protegidas contra erros ou falhas que possam surgir. Há a apresentação de mensagens informativas por toda aplicação, sabendo quais ações são realizadas com sucesso ou insucesso. Há ainda o desenvolvimento do menu e separação das possíveis ações por categorias e níveis, com intuito de facilitar a localização do utilizador na aplicação.

## Limitações

A maior dificuldade encontrada durante a solução do código foi o estranhamento a uma nova Linguagem, não ter facilidades com componentes que Java disponibiliza como objeto Date, biblioteca Time, Interface Map, entre outros.

## Trabalho Futuro

Para o futuro é necessário arriscar mais em componentes novos, desenvolver o código aplicando hierarquia e polimorfismo essencial na Linguagem Java. Estudar a manipulação de dados persistente e como otimizar o código.

# Referências

*Página da Internet:*

1. The Python Standard Library — Python 3.10.9 documentation [Internet]. [citado 15 de dezembro de 2022]. Disponível em: <https://docs.python.org/3.10/library/index.html#library-index>
2. Pygame Front Page — pygame v2.1.4 documentation [Internet]. [citado 15 de dezembro de 2022]. Disponível em: <https://www.pygame.org/docs/>
3. Git - Documentation [Internet]. [citado 15 de dezembro de 2022]. Disponível em: https://git-scm.com/doc
4. Documentação de ajuda do GitHub.com [Internet]. GitHub Docs. [citado 15 de dezembro de 2022]. Disponível em: <https://ghdocs-prod.azurewebsites.net/pt>
5. Guia de início rápido do GitHub Codespaces - Documentação do GitHub [Internet]. GitHub Docs. [citado 15 de dezembro de 2022]. Disponível em: <https://ghdocs-prod.azurewebsites.net/_next/data/ACitfX18__IMXCzy9DWtx/pt/free-pro-team@latest/codespaces/getting-started/quickstart.json?versionId=free-pro-team%40latest&productId=codespaces&restPage=getting-started&restPage=quickstart>

# Anexos

## Manual do Utilizador

### Instalação e Rodar a Aplicação

Para a instalação é necessário exportar todos os ficheiros para uma diretória própria(Figura 1). E abrir a diretória game a partir desta deverá ser aberto o terminal(Figura 2) e digitar:

**py game.py**

ou para singleplayer:

**py game\_single.py**

Então você pode começar por digitar seu nome e escolher sua skin digitando ‘R’ ou ‘r’ para usar o robô ou ‘C’ ou ‘c’ para utilizar a cobói (Figura 3).

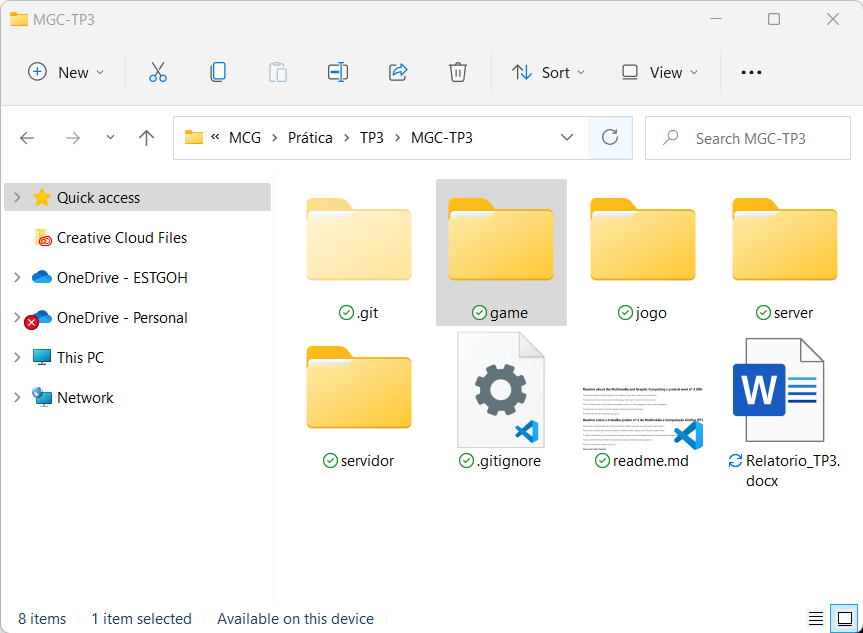


Figura 1 - Diretória própria.

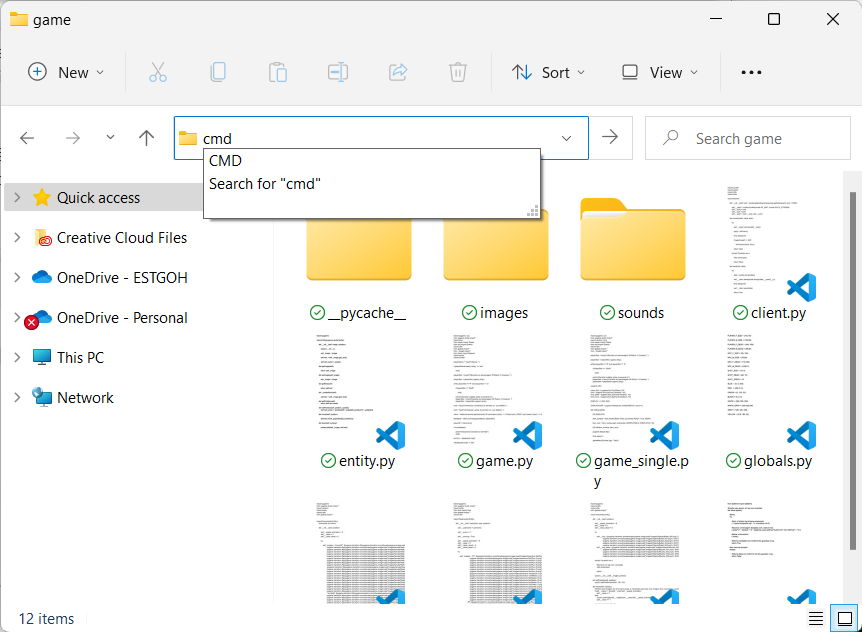


Figura 2 - Abrindo terminal.

Uma imagem com texto

Descrição gerada automaticamente

Figura 3 - Terminal.

### Menu

No inicio o menu apresenta a opção de ‘Iniciar jogo’ ou ‘Sair’. Ao pausar o jogo é apresentado o menu com as opções ‘Continuar jogo’ ou ‘Sair’. A opção ‘Continuar jogo’ volta o jogo de onde este foi pausado. Ao finalizar o jogo é apresentado o menu com as opções ‘Recomeçar jogo’ ou ‘Sair’. A opção ‘Recomeçar jogo’ inicializa um novo jogo.

Para utilizar o menu deverá usar as teclas ‘UP’ e ‘DOWN’ para selecionar uma das opções e a tecla ‘ENTER’ para confirmar a escolha.

Uma imagem com texto

Descrição gerada automaticamente

Figura 4 - Menu.

### Jogo

O objetivo do jogo é atingir zumbis com tiros, os zumbis aparecem no extremo direito da tela. A cada horda de zumbis a pontuação por atingir um zumbi se multiplica, quanto mais hordas finalizares mais pontos ganha.

O jogo termina quando um zumbi alcança o extremo esquerdo da tela ou quando um jogador chega perto demais de um zumbi.

Teclas de ação:

* Para mover o jogador para a esquerda, pressione a seta para a esquerda ou ‘A’.
* Para mover o jogador para a direita, pressione a seta para a direita ou ‘D’.
* Para mover o jogador para cima, pressione a seta para cima ou ‘W’.
* Para mover o jogador para baixo, pressione a seta para baixo ou ‘S’.
* Para atirar com o jogador, pressione a tecla ‘SPACE’. Você tem munição limitada, então é importante gerenciar suas disparos com sabedoria. Os disparos disponíveis estão na barra superior da tela.

O jogador só poderá se mover enquanto está dentro da tela.

Outras informações na barra superior da tela é sua pontuação atual, pontuação a ganhar por matar um zumbi e o nível de horda que você se encontra.



Figura 5 - Simulação do jogo.