



**Escola Superior
de Tecnologia
e Gestão**

Politécnico de Coimbra

Relatório de Projeto

Programação

Avaliação Periódica 3

Autores:

Karine Aparecida Braga Florêncio – a2021106016

Nuno Miguel Santos Lopes – a2020126392

Data: Janeiro de 2023

Resumo

O relatório documenta de forma foi desenvolvido o jogo, ao explicar a sua arquitetura e as tecnologias usadas. Também poderá ver que refere as tomadas de decisão dos alunos e quais as suas opiniões relativas a este projeto.

Palavras-chave

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

Índice

Resumo.....	ii
Lista de Figuras	iv
Lista de Acrónimos.....	v
1. Introdução.....	1
2. Objetivos e Metodologias.....	2
2.1. Ferramentas e Tecnologias	2
2.2. Planeamento	4
3. Trabalho Desenvolvido	5
3.1. Classes	5
3.2. Algoritmos	9
3.3. Procedimentos de Teste.....	10
4. Discussão	11
5. Conclusões	12
5.1. Forças	12
5.2. Limitações	12
5.3. Trabalho Futuro.....	12
6. Referências.....	13
7. Anexos	14
7.1. Manual do Utilizador.....	14

Lista de Figuras

Figura 1 - Diretória própria.....	14
Figura 2 - Abrindo terminal.	14
Figura 3 - Terminal.....	15
Figura 4 - Menu.....	15
Figura 5 - Simulação do jogo.	16

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.

1. Introdução

Serve o presente documento para fazer a síntese o trabalho prático número 3 da cadeira de Multimédia e Computação Gráfica no curso de Engenharia Informática, na Escola Superior de Tecnologia e Gestão de Oliveira do Hospital.

Conforme solicitado foi implementado um jogo em Python na qual foi desenvolvido várias funcionalidades.

Este por sua vez tinha apenas como restrição o uso da biblioteca PyGame para o desenvolvimento de jogos 2D

Em seguida irá ser exposto como o jogo foi desenvolvido, a sua arquitetura e o seu respetivo funcionamento.

2. 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.

2.1. Ferramentas e Tecnologias

2.1.1. 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>

2.1.2. 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 usá-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
```

2.2. Planeamento

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

- a. Adaptação Git/GitHub – 6 horas.
- b. Criação das classes – 3 horas.
- c. Gerir NPC's, Threads – 13 horas.
- d. Correção de desafios encontrados. – 8 horas.
- e. Animações. – 4 horas.
- f. Menu / Teclas de atalhos. – 3 horas.
- g. Implementar servidor. – 15 horas.
- h. Introduzir sons. – 2 horas.
- i. Dados permanentes. – 4 horas.
- j. Elaborar relatório – 10 horas.
- k. Elaborar manual do utilizador – 4 horas.

3. Trabalho Desenvolvido

Neste capítulo iremos explicar como o trabalho foi desenvolvido, estruturando o mesmo segundo um desenvolvimento Orientado a Objetos e os algoritmos mais importantes do jogo.

3.1. Classes

3.1.1. 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.

3.1.2. 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.

3.1.3. 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.

3.1.4. 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 tiro de acordo com o valor recebido e atualiza a animação.

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

3.1.5. 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.

3.1.6. Server

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

Atributos:

__players – conjunto de dicionários com as informações dos jogadores.

__connections – número de conexões no servidor.

__playerID – número que representa o ID do próximo jogador a conectar-se.

__npcs – conjunto de dicionários com as informações dos npcs.

__shots – conjunto de dicionários com as informações dos tiros do jogo.

__waves – número que representa a horda no jogo.

__npc_speed – número que representa a velocidade do npc.

__score_per_kill – número que representa o número de pontos por morte.

__display – tuple que representa a largura e altura da janela do jogo.

_socket – representa o socket para a conexão com o cliente.

_server – representa o endereço do servidor.

_port – representa a porta do servidor.

Métodos:

incrementConnections() – aumenta o número de conexões em 1.

decrementConnections () – diminui o número de conexões em 1.

incrementPlayerID () – aumenta o próximo ID de um jogador em 1.

getConnections(): int – obtém o número de conexões no servidor.

getNPCS() – retorna o conjunto dos npcs.

send(connection, data): boolean – envia ao cliente o tamanho dos dados a enviar e só depois envia os dados.

recv(connection): pickle – recebe o tamanho dos dados a receber e só depois recebe os dados a receber.

generateNPCS(INTERVAL) – gera npcs para a horda seguinte.

serverLog(log): boolean – função responsável por escrever num ficheiro os logs do servidor (não inclui exceções).

setupClient(connection): pickle – função responsável por manipular as informações de modo a serem enviadas e recebidas pelo cliente.

3.2. Algoritmos

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

3.2.1. game_single

Ficheiro de execução do jogo para singleplayer.

Recolhe do utilizador seu nome e 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, além de atualizar o ficheiro de dados persistentes com a maior pontuação do jogador com a função **saveScore**. 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**.

Outras funções existentes são usadas para a adaptação da trilha sonora no jogo, sendo elas: **operateSounds** que tem objetivo de parar, pausar ou recomeçar a trilha sonora; **setVolumeSound** que configura o volume da trilha sonora no jogo; **createChannels** adiciona um som a todos os sons que configuram a trilha sonora.

3.2.2. 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.

3.2.3. app

Ficheiro para execução do servidor. Inicializa e gere o servidor. Quando o número de conexões é igual a 2, recusa conexões futuras.

3.2.4. 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.

3.2.5. globals

Armazena variáveis globais da aplicação.

3.3. Procedimentos de Teste

Foram adotados diversos testes unitários não automatizados a vários componentes isolados da aplicação para a identificação de erros. Houve 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.

4. Discussão

O plano traçado inicialmente era de um jogo multiplayer de cooperação na qual dois jogadores teriam de matar zombies de modo a estes não chegassem ao final do ecrã ou de modo que estes não tocassem nos jogadores.

No entanto a versão final e jogável aponta para um jogo single-player, isto porque, devido a pressões de outras cadeiras, optamos por desenvolver um jogo single-player e caso tivéssemos tempo suficiente, migrá-lo para um jogo multiplayer.

Apesar de apenas a versão single-player estar finalizada e ser jogável, poderá observar que de facto existiu algum desenvolvimento na migração do jogo para multiplayer, contudo, devido à complexidade que isso acarreta, gerando muitos erros de software e consequentemente muitas horas de depuração, tivemos desistir de o fazer visto que o prazo de entrega estava próximo.

5. Conclusões

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

O facto de nos termos proposto para um jogo multiplayer de cooperação não saíram propriamente de acordo com o planejado, refletindo-se ao invés em um jogo singleplayer.

Ainda assim considerados que, tendo em conta fatores externos, conseguimos produzir um bom jogo num curto espaço de tempo.

Em suma, acreditamos que fizemos um bom trabalho em equipe, algo pouco explorado por nós antes.

5.1. Forças

As forças deste projeto são sobretudo o menu, a lógica por de trás do jogo, a trilha sonora, as animações dos personagens e a pontuação.

5.2. Limitações

As limitações do jogo são maioritariamente o facto de não ter sido possível implementar em tempo útil o multiplayer no jogo, no qual desbloquearia ao jogador uma nova forma de jogar, ou seja, o modo cooperação.

5.3. Trabalho Futuro

Para trabalho futuro é essencial concluir a integração do jogo no multiplayer, algo que já foi começado durante esta jornada. Isto inclui 2 jogadores conseguirem em simultâneo jogar na qual npcs são gerados pelo servidor e ações de um jogador seja espelhado no jogo do outro jogador e também ainda a sincronização da pontuação com outros jogadores.

6. Referências

Página da Internet:

- I. 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>
- II. Pygame Front Page — pygame v2.1.4 documentation [Internet]. [citado 15 de dezembro de 2022]. Disponível em: <https://www.pygame.org/docs/>
- III. Git - Documentation [Internet]. [citado 15 de dezembro de 2022]. Disponível em: <https://git-scm.com/doc>
- IV. Documentação de ajuda do GitHub.com [Internet]. GitHub Docs. [citado 15 de dezembro de 2022]. Disponível em: <https://ghdocs-prod.azurewebsites.net/pt>
- V. 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

7. Anexos

7.1. Manual do Utilizador

7.1.1. Execução do jogo

Para a execução do jogo é necessário ter previamente instalado a biblioteca Pygame (ver capítulo Desenvolvimento). Em seguida deve 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_single.py
```

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

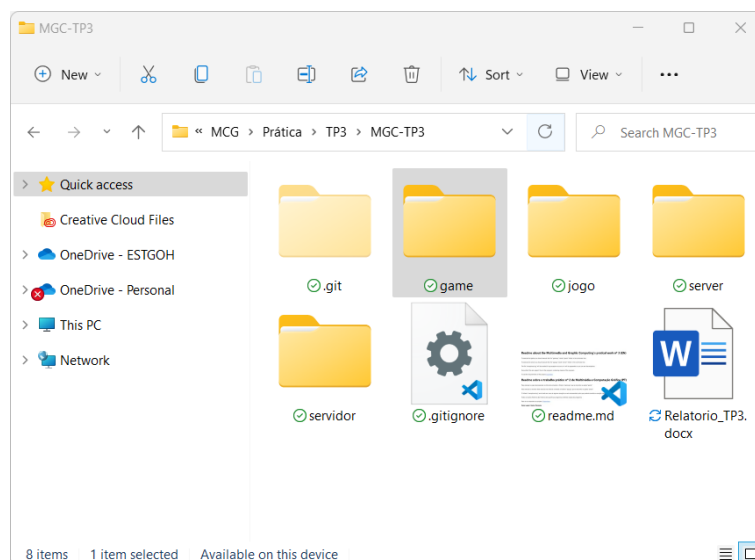


Figura 1 - Diredória própria.

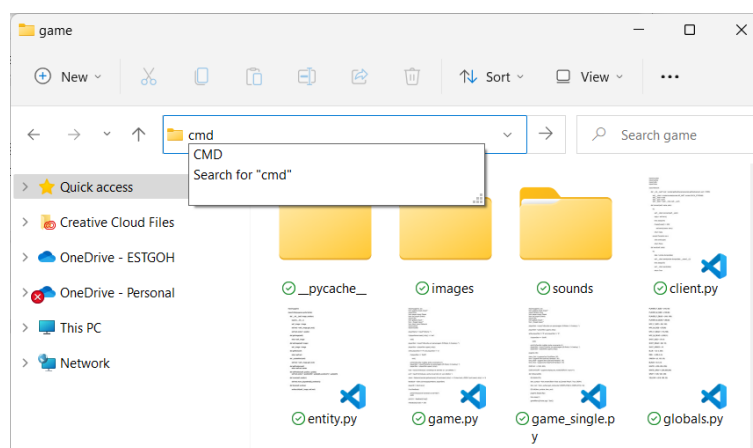
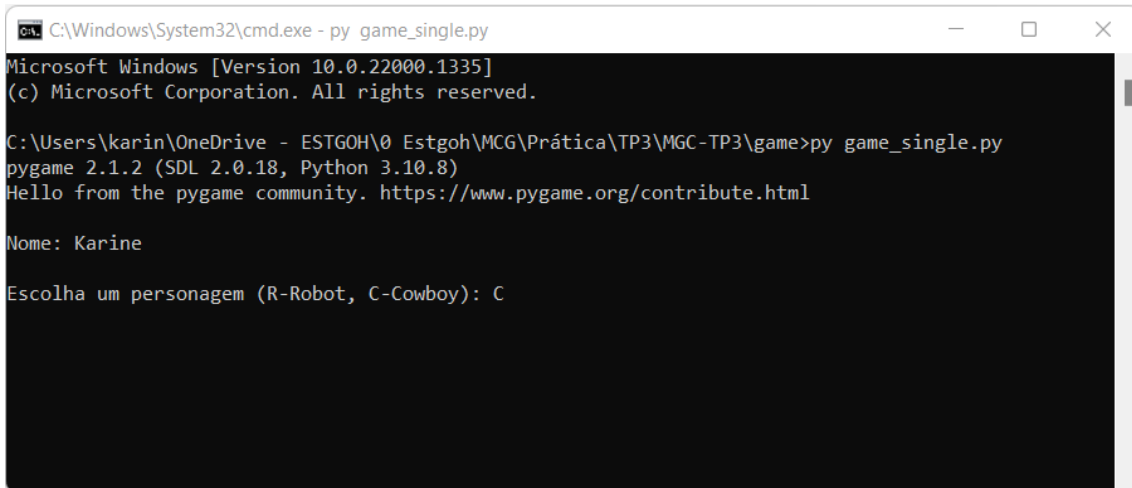


Figura 2 - Abrindo terminal.



```
C:\Windows\System32\cmd.exe - py game_single.py
Microsoft Windows [Version 10.0.22000.1335]
(c) Microsoft Corporation. All rights reserved.

C:\Users\karin\OneDrive - ESTGOH\0 Estgoh\MCG\Prática\TP3\MGC-TP3\game>py game_single.py
pygame 2.1.2 (SDL 2.0.18, Python 3.10.8)
Hello from the pygame community. https://www.pygame.org/contribute.html

Nome: Karine

Escolha um personagem (R-Robot, C-Cowboy): C
```

Figura 3 - Terminal.

7.1.2. Menu

No início 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 seleccionar uma das opções e a tecla 'ENTER' para confirmar a escolha.



Figura 4 - Menu.

7.1.3. 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 por dois, quanto mais hordas finalizar 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.

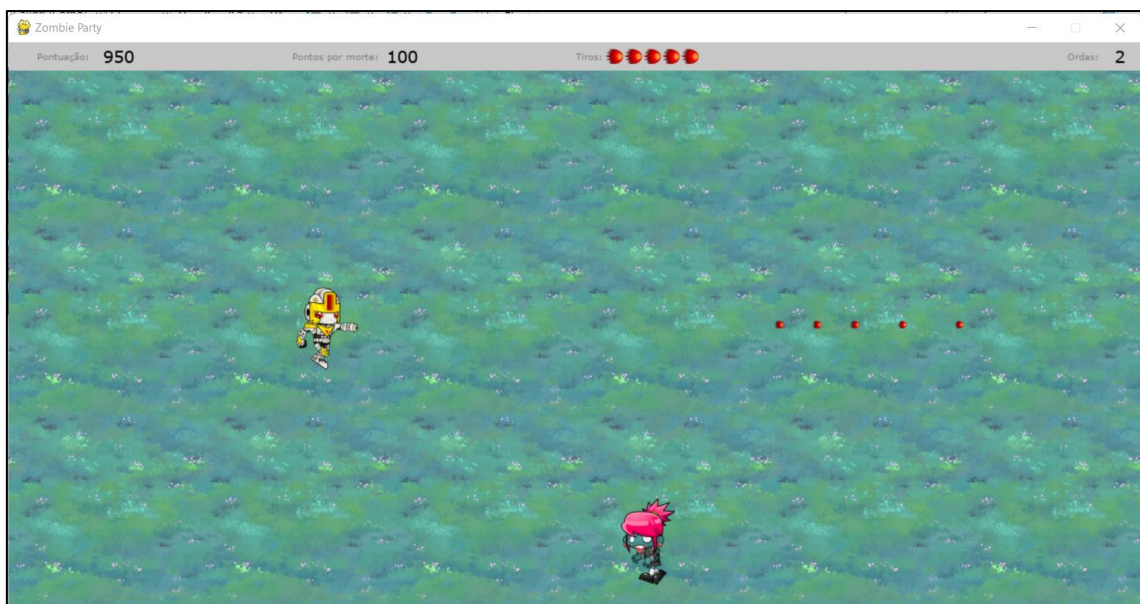


Figura 5 - Simulação do jogo.