Estrutura de fila de prioridades na resolução do jogo labirinto

Daniel Cardozo Santos - 22152299, Nilton da Silva Nascimento - 22153821

Instituto de Computação – Universidade Federal do Amazonas(UFAM) Amazonas – AM – Brazil

daniel.santos@icomp.ufam.edu.br, nilton.nascimento@icomp.ufam.edu.br

1. Introdução

O conceito de fila, conhecido popularmente, segue a estrutura em que o primeiro a chegar é o primeiro a sair. Todavia, uma fila de prioridade organiza os elementos de acordo com a prioridade definida inicialmente. Desse modo, os elementos de alta prioridade são atendidos/retirados antes dos de baixa prioridade.

As filas de prioridade são utilizadas em diversas situações. Por exemplo, nas filas de um banco é um exemplo de fila de prioridade, pois existe uma regra em que pessoas grávidas ou idosas têm uma prioridade maior em relação às outras para ser atendida. Em filas de atendimento para análise de sinais em sistemas de controle, sensores prioritários são adicionados à fila de atendimento com uma maior prioridade que sensores secundários, o que garante o atendimento de sinais prioritários à frente dos demais.

Logo, esse trabalho tem como objetivo implementar a ideia de fila de prioridade para a resolução do jogo Labirinto, juntamente com a interação manual e automática em uma interface.

2. Problema Escolhido

Para fins de demonstrar os conhecimentos obtidos durante os estudos de fila de Prioridade/ heap mínima, o jogo labirinto foi selecionado.

Criado durante o século XV e conhecido mundialmente, o jogo é construído através de um conjunto de caminhos entrelaçados, que possuem entradas incertas e dificultosas. Tem como objetivo desorientar quem o percorre, fazendo assim, uma dinâmica mais desafiadora para seus jogadores encontrarem o caminho até a saída .

O labirinto pode ser modelado a partir de grafos. Os seus corredores remontam arestas, assim como cada ponto de decisão pode ser representado por um vértice. Os problemas de labirinto são baseados em encontrar a saída a partir de um ponto de origem, que pode ser interpretado como encontrar um caminho válido entre dois vértices de um grafo.

2. Procedimento de busca

Durante uma partida do Jogo Labirinto, o número de caminhos possíveis até o ponto final pode ser grande. Porém, esse percurso pode ter uma alta complexidade que vai requerer um tempo proporcional a sua complexidade , o que também pode levar a maiores erros e dúvidas durante o jogo.

Tendo em vista que tal problema pode ser resolvido com fila de prioridade, foi implementado a concepção do algoritmo de Dijkstra com o objetivo de calcular o caminho com a menor quantidade de movimento até o ponto final do labirinto.

Sendo assim, o procedimento seguiu os seguintes passos:

2.1) Definindo Estado Atual e demais variáveis

```
def caminho_ate_Fim(estado):
    fila_De_Prioridade = [] # heapMinima
    estados_Passados = set() # Pontos que já foram Escolhidos Anteriormente
    estado_Atual = estado # Começa com o Estado Inicial (Ponto Inicial)
    fila_De_Prioridade.append(estado_Atual)
    vetor = [] # vai receber os Possiveis Proximos estados a serem escolhidos
```

Na função Caminho_ate_Fim, recebe como parâmetro o Ponto Inicial, que será recebido pela variável estado_Atual. As seguintes variáveis são utilizadas para facilitar a escolha do próximo estado, que serão descritas posteriormente.

2.2) Etapa para a geração do caminho referente ao ponto inicial até o final.

```
while len(fila_De_Prioridade) > 0:
    if (estado_Atual.PontoAtual == estado_Atual.PontoFinal):
        return estado_Atual.caminho # caso ponto Atual == ponto Final
    estados_Passados.add(estado_Atual.PontoAtual)
    vetor = Estado.transicoes(estado_Atual, estados_Passados)
    fila_De_Prioridade = Heap.remover_Primeiro_Elemento(fila_De_Prioridade)
    fila_De_Prioridade = Heap.atualizar_Heap(fila_De_Prioridade, vetor) # reorganiza Heap Minima

    if len(fila_De_Prioridade) != 0:
        estado_Atual = fila_De_Prioridade[0] # Seleciona o Proximo Estado
return [] # retorna um vetor Vazio caso não tenha solução
```

No loop há duas possibilidades de parada, caso a fila de prioridade seja vazia ou o ponto que encontramos seja o ponto final. Se nenhuma dessas condições for verdadeira, o estado atual (ponto) será adicionado nos estados passados.

A seguir, chamamos a função transições() que retorna um vetor contendo os possíveis estados a partir de uma ponto, referente às posições que o jogador pode se mover . Em seguida Atualizamos a fila de Prioridade (heap Mínima), removendo o primeiro elemento da fila e adicionando os novos estados com a função atualiza_Heap.

Após esse procedimento, o Estado que estiver mais próximo do ponto final (saída), estará no início da fila de prioridade, onde será selecionado e colocado na variável estado atual. Em seguida, ele retorna para o início do loop.

2.3) função transições

```
def transicoes(self, estados_Passados):
    saida = []# Retorna As possiveis Transições (Estados) a partir de um Estado

    for i in range(1,5):
        proxEstado, direcao = Ponto.RetornaPonto(self.PontoAtual, i)
        if (self.matriz[proxEstado.y][proxEstado.x] == 1) and ((proxEstado.x, proxEstado.y) not in estados_Passados):
            caminho = self.caminho + [direcao] # adicionando movimento no caminho
            saida.append(Estado(self.matriz, proxEstado, self.PontoFinal, self.g + 1, caminho))
    return saida
```

Função responsável por retornar os possíveis estados que podem ser escolhidos. A função RetornaPonto, devolve o estado (ponto) e as 4 direções. Todavia, antes de adicionar no vetor que será retornado, é realizada a verificação se esse estado já não foi escolhido anteriormente ou não é um ponto proibido.

2.4) Heurística

```
def calcula_Heurística(self):  # calcula_Heurística = |Ponto Atual - Ponto Final|
    return int(Ponto.DistanciaEntreDoisPonto(self.PontoAtual, self.PontoFinal))
```

Função que retorna o cálculo do estado atual até o ponto final (h).

Essa função é importante pois é realizada a soma do número de passos até o estado Atual (g) com a heurística(h), formando o (f), que é a regra de prioridade.

$$f(x) = g(x) + h(x)$$

Logo, quanto menor o valor de f(x), mais próximo do início da fila de prioridade (heap mínima) esse estado se encontra.

De forma geral, a busca pela saída começa no ponto inicial. Ao passar pela função caminho_ate_fim(), é obtido um vetor com os possíveis movimentos (estados candidatos) que podem ser escolhidos para ser o próximo estado. Em seguida, esses estados candidatos são inseridos na fila de prioridade (heap mínima). Logo após, o

primeiro elemento da fila é selecionado para ser o estado atual. Esse processo se repete até o estado atual ser igual ao ponto final(saída).

4. Elaboração da interface

Inicialmente, foi realizado o processo de decisão referente às telas que seriam implementadas na interface. Onde o usuário tem a possibilidade de decidir o modo de criação do labirinto e resolução.

Após a realização dos moldes de todas as telas. Levando em consideração todos os procedimentos que seriam realizados para a criação da área compartilhada. A biblioteca pygame se apresentou como a mais viável a ser utilizada.

Ao todo, foram criadas três classes responsáveis por diferentes áreas, que são:

a) **Gera_labirinto_manual**

Onde o usuário irá realizar a criação do labirinto na interface , escolhendo o ponto inicial e final.

b) **interface_Resolucao**

Realiza a plotagem do labirinto já existente, atualizando a posição do usuário a cada movimento, seja ele manual ou automático

c) **interface_Menu**

Responsável pela interação do usuário com o jogo, conectando todas as etapas, desde a criação do labirinto até o resultado final de sua tentativa.

Além disso, as quarenta e oito imagens utilizadas para a animação foram criadas manualmente, com o intuito de criar uma interação maior com o usuário. Também apresentando instruções para facilitar a compreensão do que deve ser realizado.

Sendo assim, apesar das dificuldades durante o desenvolvimento, seja com bugs ou a resolução da imagem ser de acordo com o desejado. Foi criada uma interface de fácil manuseio e compreensão.

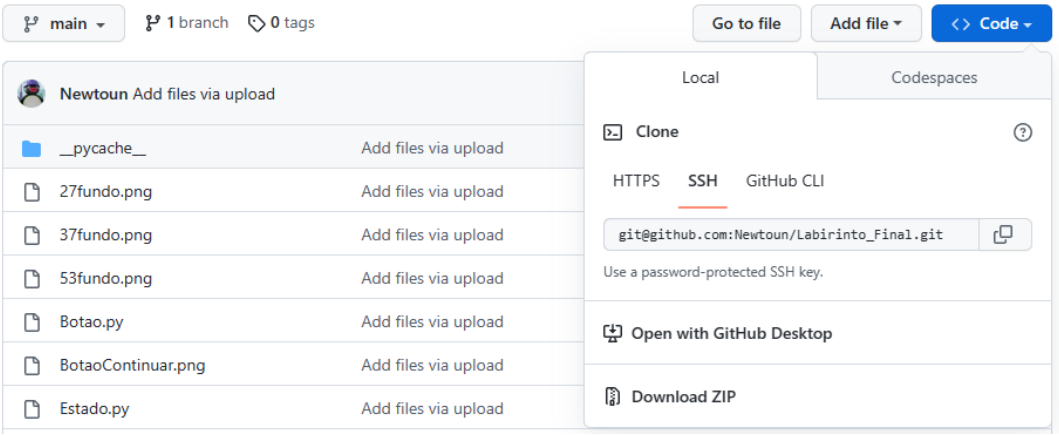
5. Instrução de instalação

Já levando em consideração que o usuário já possui o compilador python e pygame.
Para instalar pygame no windows pelo terminal : **“$pip install pygame”.**
Para instalar pygame no linux pelo terminal: **“$sudo apt-get install python3-pygame”**

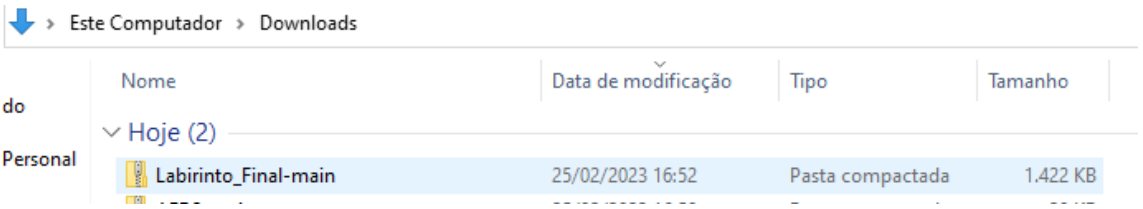
5.1 Windows

5.1.1 Baixando Arquivo Zip:

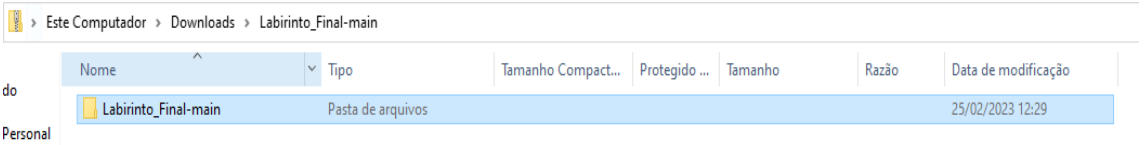
Após acessar o link , clique no botão “Code” e faça o Download do arquivo zip.



5.1.2 - Abra o arquivo .zip na pasta onde o arquivo foi baixado.

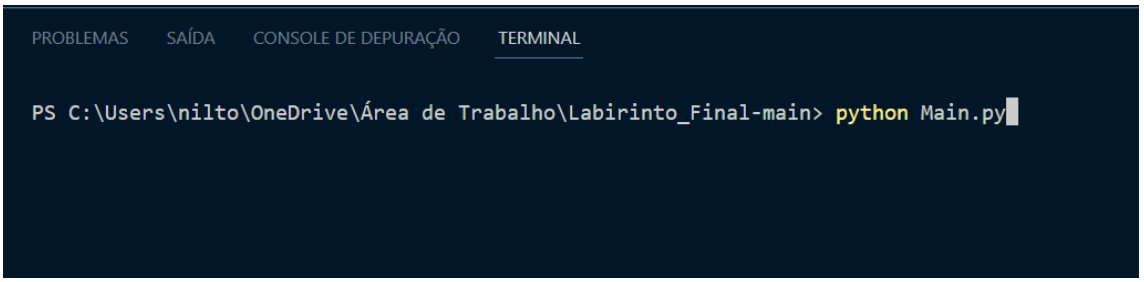


5.1.3 - Copie a pasta interna para a sua área de trabalho ou área de sua preferência



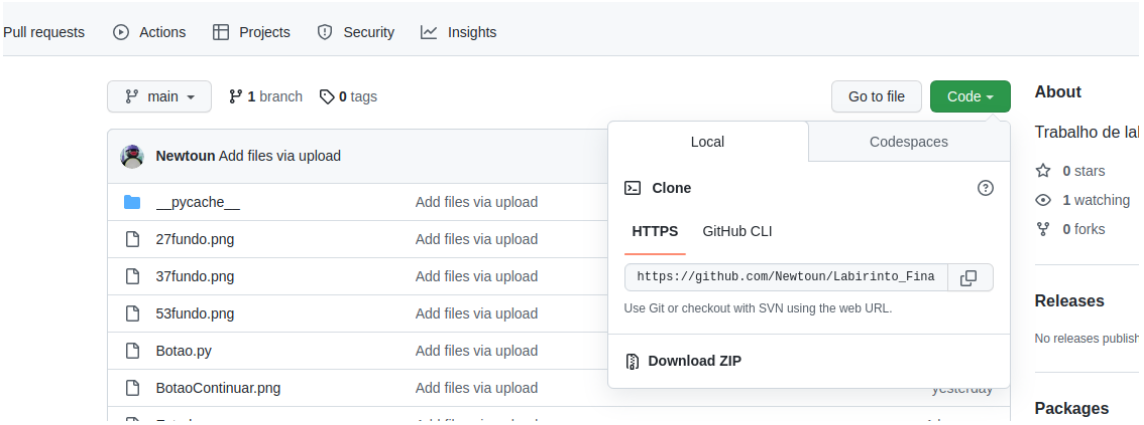
5.1.4 - Escolha uma ide de sua preferência para abrir a pasta

5.1.5 - compile o arquivo Main.py com o comando “Python Main.py” no terminal

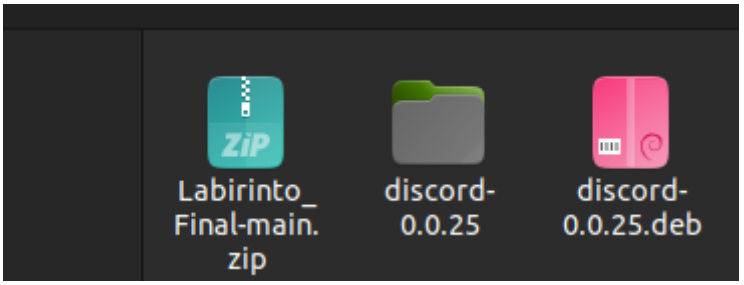


5.2 Linux-Ubuntu

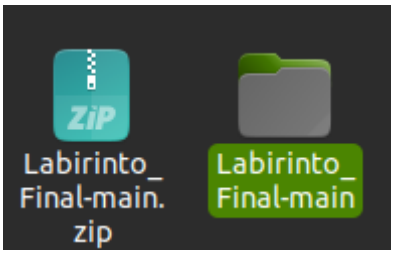
5.2.1 - Após acessar o link baixe o arquivo zip



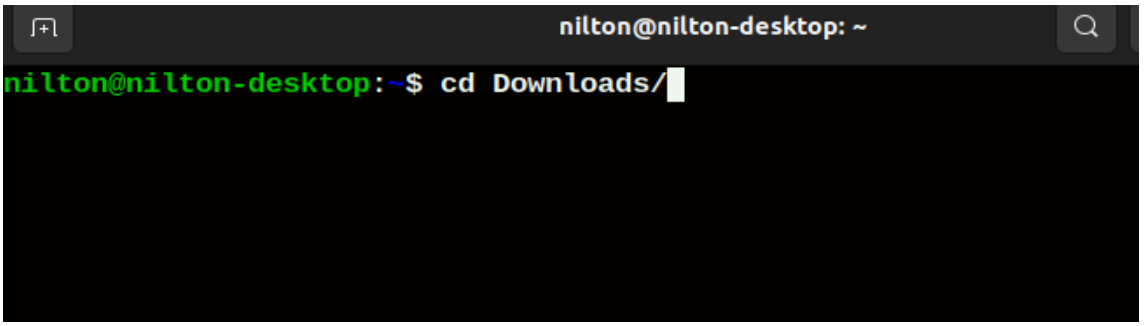
5.2.2 -Localize o arquivo após ser baixado



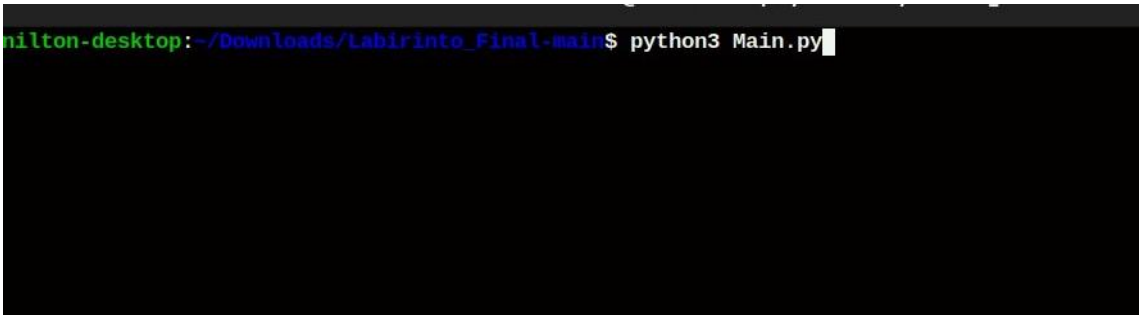
5.2.3 - Extraia a pasta para o local de sua preferência



5.2.4 Localize a pasta pelo seu Terminal



5.2.5 - Compile o arquivo Main.py com o comando "python3 Main.py" no terminal.



6. Modo de uso do Aplicativo desenvolvimento

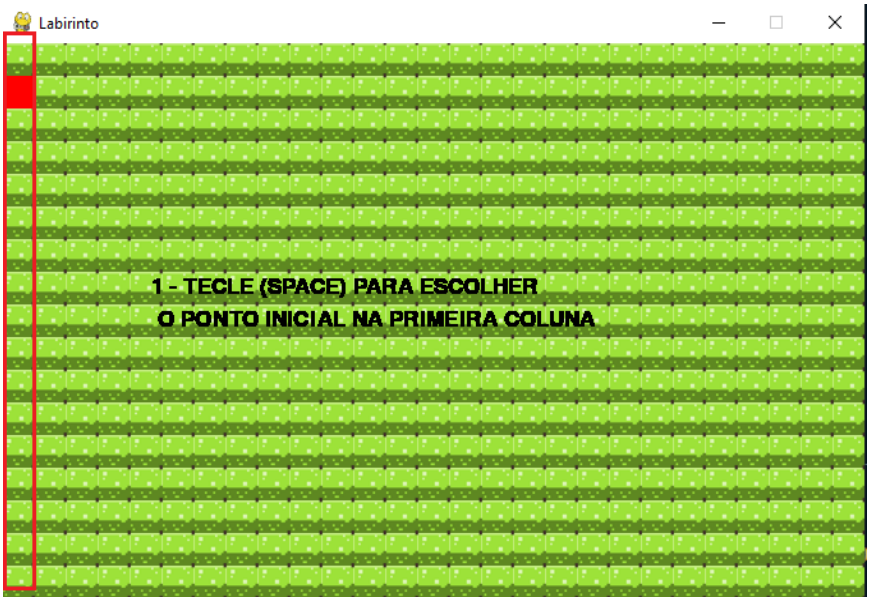
6.1 Seleção do nível de dificuldade

Selecione o nível que deseja jogar. Os níveis se referem ao tamanho do labirinto que irá ser criado.

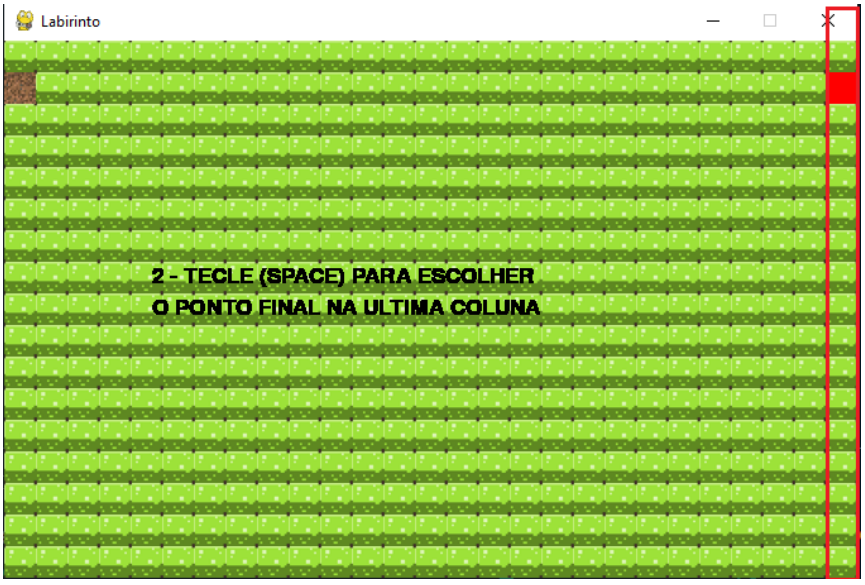
6.2 Seleção de como criar labirinto

Selecione o modo como deseja que o labirinto seja criado.

6.3 Como criar o labirinto manual:



Seleciona a linha da primeira coluna apertando seta para cima ou para baixo em seu teclado e aperte “Espaço” para confirmar sua escolha.



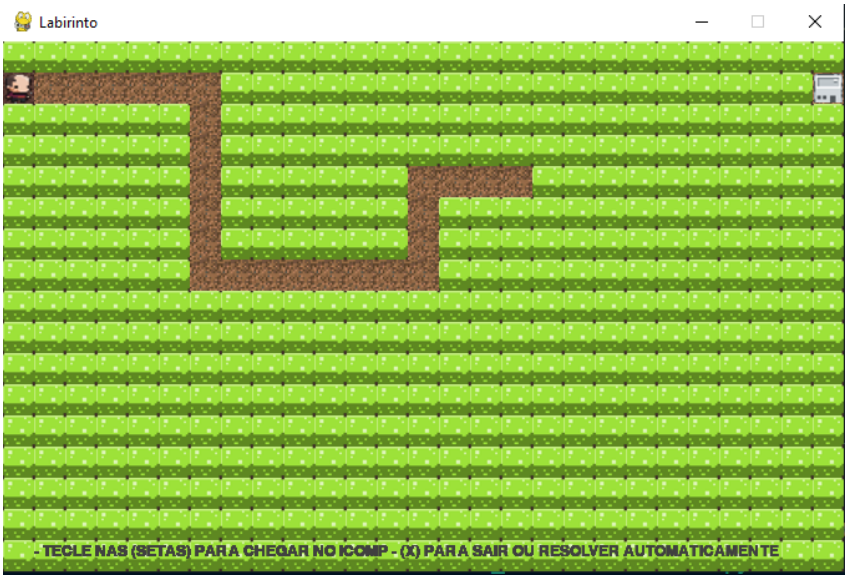
Da mesma forma, selecione a linha da última coluna que será o ponto final do labirinto.

Ande pela interface utilizando as setas, e pressione “C” para a posição que você deseja transformar em caminho. Após as suas escolhas, pressione “S” para salvar seu labirinto.

6.4 Resolver o labirinto de forma manual ou automática:

Selecione o modo na qual você deseja resolver o labirinto gerado.

6.5 Tela de jogo:



Caso o usuário escolha andar manualmente pelo labirinto utilizando as Setas

6.6 Em caso de desistência de resolução manual:

Caso o usuário não deseje mais realizar o processo de resolução manual, apertando a tecla “X” ou clicando em fechar janela , ele terá a opção de resolver o labirinto de forma automática ou sair do jogo.

7. Conclusão

Em virtude de todo o processo realizado perante a execução do trabalho. A fila de prioridade se apresenta de grande relevância para a resolução de problemas da busca do melhor caminho. Vale destacar que sua manipulação deve ser realizada com extremo cuidado para que não ocorra frustrações e erros com os resultados obtidos. Logo, foi de

muita importância e aquisição de conhecimento, a realização prática da utilização da estrutura da fila de prioridade na resolução do jogo labirinto

8. link para acessar o arquivo no github

https://github.com/Newtoun/Labirinto_Final

9. Referências

https://github.com/russs123/pygame_tutorials/tree/main/Button

https://acervolima.com/python-exibir-texto-na-janela-pygame/

https://stackoverflow.com/questions/65887274/collision-for-pygame-game-map/65888081#65888081