**UNIVERSIDADE TÉCNOLOGICA FEDERAL DO PARANÁ**

**GUSTAVO YUJI MEKARU**

**THIAGO RENAN DE LIMA**

**VINICIUS FONSECA NAVARRO**

**RESOLUÇÃO DE PROBLEMA**

**POR MEIO DE ALGORITMOS DE BUSCA:**

**IMPLEMENTAÇÃO DO JOGO PAC-MAN**

**PONTA GROSSA – PR**

**2023**

**GUSTAVO YUJI MEKARU RA: 2163306**

**THIAGO RENAN DE LIMA RA: 2165198**

**VINICIUS FONSECA NAVARRO RA: 2319314**

**RESOLUÇÃO DE PROBLEMA**

**POR MEIO DE ALGORITMOS DE BUSCA:**

**IMPLEMENTAÇÃO DO JOGO PAC-MAN**

Projeto apresentado como requisito para obtenção de nota referente a matéria de inteligência artificial.

Professora: Helyane Bronoski Borges

**PONTA GROSSA – PR**

**2023**

Sumário

[1. INTRODUÇÃO 3](#_Toc146476678)

[2. DESCRIÇÃO DO PROBLEMA 4](#_Toc146476679)

[2.1 Jogo 4](#_Toc146476680)

[2.2 Algoritmos de busca 5](#_Toc146476681)

[3. METODOLOGIA 6](#_Toc146476682)

[3.1. Ferramentas 6](#_Toc146476683)

[3.2. Algoritmos 6](#_Toc146476684)

[3.3. Heurísticas 7](#_Toc146476685)

[4. IMPLEMENTAÇÃO 8](#_Toc146476686)

[4.1 Códigos Gerais 8](#_Toc146476687)

[4.2 Implementação dos Algoritmos de Busca 9](#_Toc146476688)

[4.2.1 Algoritmo Guloso 9](#_Toc146476689)

[4.2.1 Algoritmo A\* 10](#_Toc146476690)

[4.3 Desenvolvimento no UNITY 12](#_Toc146476691)

[5. RESULTADOS 15](#_Toc146476692)

[6. CONCLUSÃO 17](#_Toc146476693)

[REFERÊNCIAS 19](#_Toc146476694)

# 1. INTRODUÇÃO

A utilização de algoritmos de busca não é um novo conceito dentro da área da ciência da computação, diversos tipos existem para encontrar um caminho em um labirinto ou achar um número especifico dentro de um vetor, estes podem ser tanto a busca linear que pode ser implementada em poucos segundos, até a utilização de buscas como A\* (Astar/ A estrela) onde diversas condições devem ser cumpridas para que um resultado seja obtido, dependendo de qual objetivo o desenvolvedor planejou para o programa, inúmeros algoritmos de busca podem ser utilizados.

PAC-MAN originalmente desenvolvido por Toru Iwatani, um designer de jogos empregado pela NAMCO. Em 1980, quando o jogo foi lançado, os computadores não possuíam uma grande capacidade de computação, considerando que o jogo possui oque parecem 4 agentes, estes sendo os fantasmas, isso pode parecer um grande desafio para época pensando de maneira tradicional onde cada fantasma é um objeto individual dentro do algoritmo, porém Toru utilizou em seu jogo uma técnica anti-objeto, onde o próprio labirinto possuía regras e os fantasmas apenas as seguiam. (NEAL FORD 2008).

Nos dias de hoje, os computadores apresentam uma capacidade de computação infinitamente maior do que em 1980, portanto é possível empregar a inteligência em cada fantasma individualmente, que é o caso desse projeto, onde foi empregado tanto o algoritmo guloso como o A\* nos fantasmas, com essa individualidade que os fantasmas apresentam ao utilizar diferentes algoritmos é possível criar diferentes comportamentos e dinâmicas que afetam a jogabilidade.

# 2. DESCRIÇÃO DO PROBLEMA

O projeto consiste no desenvolvimento do jogo PAC-MAN utilizando os algoritmos A\* e busca gulosa com diferentes heurísticas em relação a implementação do comportamento dos fantasmas.

## 2.1 Jogo

PAC-MAN lançado em 1980 e desenvolvido por Toru Iwatani é um jogo com um uma proposta simples, o usuário controla o personagem principal, que é chamado de PAC-MAN, ele se encontra em um labirinto 2D onde existem as *pellets*, *power pellets*, e os fantasmas.

As *pellets* são a base do sistema de pontuação do jogo, ao passar por cima delas PAC-MAN as “come” e uma quantidade de pontos é adicionada a sua pontuação.

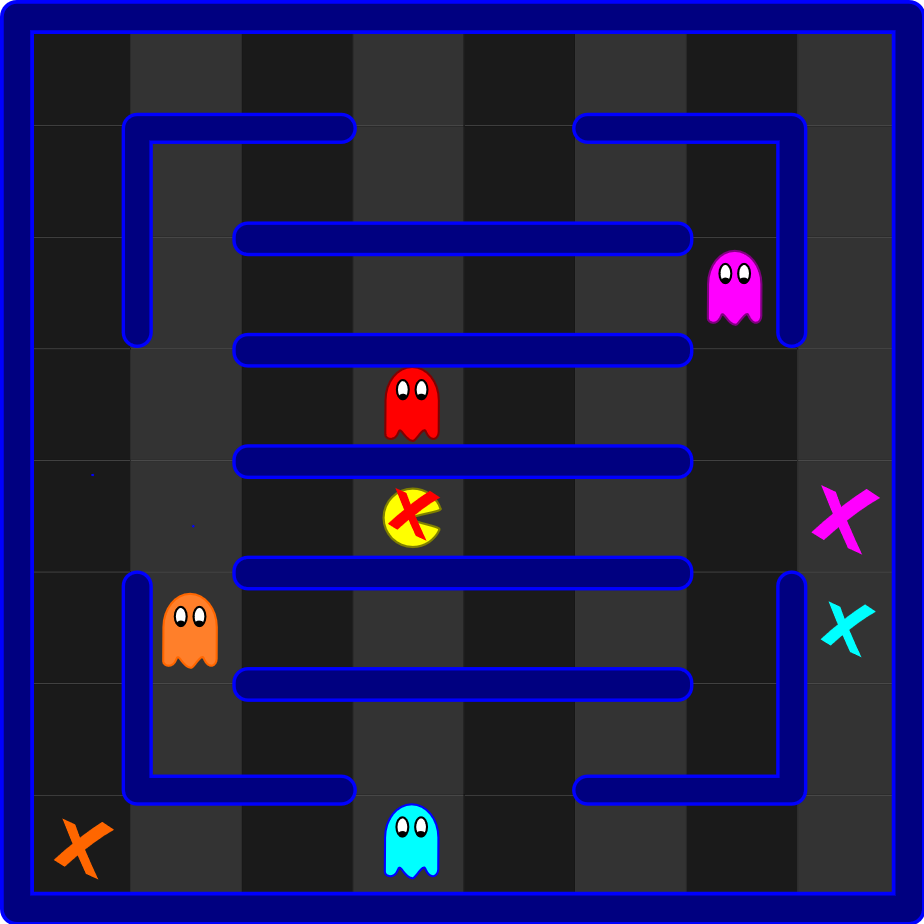
As *power pellets* além de fornecer uma quantidade maior de pontos ao jogador do que as *pellets* comuns, também vão deixar os fantasmas vulneráveis por um período.

Os fantasmas podem apresentar comportamentos diferentes dependendo de como eles foram programados, mas originalmente os 4 fantasmas foram conceptualizados de formas distintas, estas sendo (Imagem 1 – Células alvo):

* Blinky: Fantasma vermelho, utiliza como alvo a célula do mapa em que PAC-MAN está, ou seja, seguindo-o diretamente;
* Pinky: Fantasma rosa, utiliza como alvo a célula que está 4 células em frente PAC-MAN caso ele esteja indo em direção a esquerda, direita ou para baixo, caso ele esteja indo para cima, o alvo será a célula 4 células acima e 4 células a esquerda de PAC-MAN, mas isso pode ser considerado uma anomalia causada por overflow ao se calcular a posição (DON HODGES 2008);
* Inky: Fantasma azul, depende tanto da posição de PAC-MAN e de Blinky, o alvo é a célula encontrada no cálculo dá distancia da 2ª célula em frente a PAC-MAN a distância de Blinky rotacionada em 180º, apresenta a mesma anomalia que Pinky quando PAC-MAN está virado para cima;
* Clyde: Fantasma laranja, utiliza como alvo a célula do mapa em que PAC-MAN está, similar a Blinky, porém ao chegar a 8 células de distância de PAC-MAN, ele vai se redirecionar a esquerda inferior do labirinto;

Além desse comportamento enquanto os fantasmas estão caçando PAC-MAN, também existe seu comportamento vulnerável, que ocorre quando PAC-MAN come uma *power pellet*, durante este estado, os fantasmas tentaram fugir de PAC-MAN indo para direções contrarias de maneira aleatória.

Quando PAC-MAN come algum dos fantasmas eles entraram em um novo estado onde ficaram invisíveis e retornaram para o centro da tela, durante este estado eles não podem interagir com PAC-MAN, após um tempo retornaram ao seu comportamento normal.



**Figura 1 – Células alvo (Autoria própria 2023)**

## 2.2 Algoritmos de busca

Os algoritmos de busca A\* e de busca gulosa podem ser utilizados para implementar essas diversas personalidades dos fantasmas presentes no jogo, enquanto o A\* é geralmente eficiente e encontra soluções ótimas quando a heurística é bem escolhida, a busca gulosa não garante a um resultado ótimo, ou seja, não necessariamente encontrará o caminho mais curto. No entanto, pode ser computacionalmente eficiente, especialmente em espaços de busca muito grandes, devido à sua abordagem de escolher o próximo nó com base apenas na heurística.

O funcionamento desses 2 algoritmos pode ser descrito das seguintes formas:

* Algoritmo A\*: Avalia os nós do grafo com base em duas métricas: O custo acumulado do ponto de partida até o nó atual (o custo real, denotado como ). Uma estimativa do custo do nó atual até o ponto de destino (uma heurística, denotada como ). Utiliza uma função de avaliação, tal . O nó com o menor valor de é escolhido para expansão;
* Busca gulosa: Avalia os nós com base apenas na heurística, ou seja, usa apenas a estimativa do custo do nó até o destino, ou seja, o nó que possui o menor valor em determinado momento é escolhido para expansão.

# 3. METODOLOGIA

Nas seguintes sessões serão descritas as ferramentas utilizadas para implementar o jogo, os algoritmos de busca aplicados ao comportamento dos fantasmas e as heurísticas que orientaram suas decisões.

## 3.1. Ferramentas

O jogo foi desenvolvido com uso da Unity, que é uma *engine* (motor de jogos) grátis para uso pessoal ou pequenas empresas, essa escolha foi dada por ela ser uma *engine* acessível e possuir um *framework* direcionado a desenvolvimento de jogos.

Também foi utilizado o IDE (Ambiente de desenvolvimento integrado) Visual Studio para a implementação dos algoritmos, heurísticas do fantasma e toda parte programável do jogo, Visual Studio é suportado oficialmente pela Unity e é a IDE escolhida por padrão durante a instalação, fazendo com que seja a escolha mais lógica para a criação do jogo.

Para o desenvolvimento dos sprites do jogo, que são os personagens, *pellets*, fantasmas e outros elementos visuais, foram utilizadas as ferramentas GIMP (Programa de Manipulação de Imagem do GNU) uma ferramenta grátis de código aberto, e *microsoft paint* que é um software que vem com a instalação padrão do sistema operacional Windows. A escolha dessas ferramentas foi motivada pela sua acessibilidade e eficácia na criação dos elementos gráficos.

## 3.2. Algoritmos

Busca gulosa foi utilizada no fantasma Blinky e A\* foi utilizada em Pinky, que consequentemente foram os algoritmos escolhidos para a implementação do comportamento, ambos foram escolhidos de acordo com o enunciado disponibilizado pela professora, mas é possível citar algumas vantagens que esses algoritmos apresentam, ambos são algoritmos versáteis, bastante utilizados em jogos por possuírem uma proposta simples de encontrar o menor caminho e serem rápidos, outra vantagem é que podem ser adaptados para configurar diversos comportamentos para os diferentes fantasmas nesse caso, ou outras entidades em outros jogos, entre diversas outras qualidades.

## 3.3. Heurísticas

Os fantasmas Blinky e Pinky utilizam a heurística de distância euclidiana, o objetivo principal é orientar os fantasmas em direção ao PAC-MAN. Ela é calculada usando a seguinte fórmula que mede a distância em linha reta entre dois pontos em um espaço bidimensional. Supondo que um fantasma esteja posicionado nas coordenadas (x1, y1) e PAC-MAN esteja em (x2, y2).

No contexto do jogo, essa heurística determina a distância direta entre a posição atual de um fantasma e a posição do PAC-MAN no labirinto, quando um fantasma decide qual direção seguir para alcançar o PAC-MAN, ele calcula a distância euclidiana e escolhe a menor direção para alcança-lo. Essa heurística foi escolhida por possuir uma grande eficácia em achar um caminho direto entre os fantasmas e PAC-MAN, deixando os fantasmas de certa forma desafiadores.

Inky e Clyde não possuem heurística e andam em direções aleatórias, já que os dois fantasmas prévios compartilham a mesma heurística, mesmo que baseados em algoritmos diferentes, eles podem apresentar padrões que os jogadores podem abusar, a aleatoriedade aplicada em Inky e Clyde pode pegar jogadores despercebidos, isso adiciona uma dinâmica maior ao jogo.

# 4. IMPLEMENTAÇÃO

Como evidenciado anteriormente, os scripts foram feitos utilizando o Visual Studio, e a linguagem foi o C#. Para que todas as funcionalidades do projeto fossem atendidas, foram utilizados diversos arquivos de código. Dessa forma, cada função dentro do game recebeu um arquivo separadamente para sua funcionalidade. A seguir, serão comentadas as funções mais simples:

## 4.1 Códigos Gerais

A seguir, serão demonstrados os códigos gerais implementados na funcionalidade do jogo, apresentando também uma breve descrição de seu funcionamento:

Quadro 1 – Códigos Gerais

|  |  |
| --- | --- |
| **Nome do Script** | **Descrição** |
| 1. GameManager.cs | Central principal do código, serve principalmente para atualizar a tela e checar se o jogador ganhou a fase ou perdeu a fase. |
| 2. ChangeScene.cs | Muda as fases, ou seja, muda para cada tela dependendo do botão clicado ou no caso de vitória ou derrota |
| 3. Pellet.cs | Basicamente trabalha com o sistema de colisão, aonde caso um objeto encoste nele e esse objeto seja o Pacman, ele faz com que a pellet suma da tela e o jogador pontue. |
| 4. PowerPellet.cs | Caso seja “comido” pelo Pacman, ele some e muda a cor dos fantasmas para branco e permite que eles sejam derrotados. |
| 5. Pacman.cs | Código principal do Pacman, anexa os controles com a direção de movimento do personagem, além de fazer as rotações de Sprite (Aplicado na fase 3) |
| 6. Ghost.cs | Central de estados do fantasma, define como ele está agindo, e faz a ligação entre os códigos do fantasma |
| 7. GhostHome.cs | Estado onde os fantasmas estão dentro da casa. Ele deixa o fantasma em espera, e após um tempo ele move o fantasma para o meio da tela e então o retira da casa. |
| 8. GhostScatter.cs | Serve para espalhar os fantasmas, é o movimento inicial que ele faz após sair da casa, da qual vai até o node mais próximo e após isso inicia a execução da busca. |
| 9 .GhostFrightened.cs | Funciona em conjunto com a PowerPellet, muda a cor do fantasma e o permite ser derrotado. A duração desse estado está definida para durar 8 segundos após pegar o PowePellet. |
| 10. GhostChase.cs | Será melhor explicado a seguir, local onde fica implementado os algoritmos de busca |
| 11. Movement.cs | É o script que implementa a movimentação dos objetos na tela. Quando o Pacman está se movimentando pelas setas, ele envia a direção atual do pacman e um buffer que recebe a próxima direção que ele deve se mover. Assim que ele não tiver uma parede impedindo esse movimento, ele realiza o movimento do buffer e começa a rodar novamente. Quando ele esbarra em uma parede, ele ficará parado esperando o próximo comando de movimento |

**Fonte: Autoria Própria (2023)**

## 4.2 Implementação dos Algoritmos de Busca

O código “GhostChase.cs” serve como a central de busca dos fantasmas. Nele, são chamados os algoritmos de busca que definem qual caminho o fantasma deve seguir, utilizando a heurística da menor distancias entre dois pontos (Calculada através de uma função própria do Unity). Portanto, o método “OnTriggerEnter2D()” recebe o retorno das funções de busca que se trata de uma lista de posições próximas que o fantasma deve assumir para chegar no alvo (pacman), ele envia esses comandos uma a um para o fantasma até que ele chegue no próximo node. Ao chegar nesse novo node, todos os cálculos são refeitos e o algoritmo de busca roda novamente, enviando as novas direções e repetindo essas ações até que ele encontre o alvo.

Vale comentar que as posições dos personagens são feitas utilizando valores de x e y no plano cartesiano. Portanto, para que o algoritmo fixe o alvo no pacman, ele utiliza de outro método “GetTargetNodePosition()”, que faz a identificação do node mais próximo à posição do pacman.

Após adquirir esses dados do alvo, quando o fantasma chegar no node (estrutura invisível que fica representada em vários pontos da tela), ele rodará um dos algoritmos de busca que fica de para cada fantasma e será explicado a seguir. Para ambos os algoritmos, foi utilizado a mesma heurisca da menor distância entre dois pontos. O que faz esse cálculo é um método chamado “CalculateEuclidianDistanceCost()”, que retorna a distância euclidiana.

### 4.2.1 Algoritmo Guloso

Para implementar o algoritmo guloso, utilizamos um método chamado *“GreedySearch()”.* O primeiro passo é descobrir se foi possível achar o pacman e retornar a lista de movimentos. Após isso, ele pega as posições de todos os vizinhos possíveis usando o método *“GetPossibleStates()”,* função que pega e descobre quais direções o fantasma poderá se mover sem bater em nenhuma parede e retorna uma lista de direções. Depois disso, checa caso não tenha nenhum vizinho.Se caso tiver, ele retorna. Após isso, ele adiciona todos os vizinhos com suas distancias já calculadas dentro da lista. Para isso, ele usa o método de cálculo de distância Eucladiana, passando os vizinhos de parâmetro. Com a distância em mãos, ele ordena os vizinhos em ordem de ascendente por custo. Se todos os vizinhos forem custo maior que a posição inicial, ele retorna a lista de movimentos que devem ser realizados.

Como só é possível passar um comando de movimento por vez para o fantasma, e isso não seria considerado guloso, foi implementado um sistema aonde a cada node, é demonstrado um símbolo de atenção para demonstrar qual é o objetivo final do fantasma depois dele realizar todos os comandos de movimentos propostos ou chegar a um novo node onde um novo cálculo de busca gulosa sera feita, dando o próximo resultado de caminho até o pacman.

Figura 2 – Código-fonte Algoritmo Guloso



****Fonte: Autoria Propria (2023)****

### 4.2.1 Algoritmo A\*

Ambos os algoritmos de busca apresentam algum grau de similaridade entre eles, como por exemplo uma lógica de programação similar. Porem, o algoritmo “A\*” apresenta uma vantagem em relação ao seu antecessor. Com esse algoritmo, é possível guardar custos anteriores e não apenas o atual e seus vizinhos. Assim ele consegue lidar com alguns erros que o guloso possa cometer, já que o mesmo é um algoritmo cego às possibilidades anteriores.

Como dito anteriormente, ambos apresentam uma estruturação parecida onde o método principal nesse caso é o *“AstarSearch()”,* que serve de central para a os métodos do algoritmo. A principal funcionalidade do “A\*” é o seu cálculo de custo que, diferente do guloso, não utiliza apenas o custo da heurisca, como também guarda o custo do início até a posição que ele esta atualmente. Somando então esses dois custos, é possível ter resultados mais satisfatórios em relação ao custo final para chegar até o objetivo. O cálculo do custo então fica realizado da seguinte forma:

H = Custo da distância Heuristica (Mesmo custo utilizado no guloso).

G = Custo do início do caminho até a posição atual.

O custo G inicial será de 0. Como estamos no começo do caminho, são criados duas listas, uma fechada, e umaa aberta. Um elemento entra na lista fechada a partir do ponto em que todos os seus vizinhos estejam listados, assim como na lista aberta estão os elementos que ainda não tiveram todos os seus vizinhos listados.

Com as listas em mãos, checamos qual a direção apresenta o menor custo utilizando o *“GetLowestFCostNode()”,*  que identifica qual o menor custo da lista aberta e a define como “currentNode”. Após isso, ele realiza a mesma operação de obtenção dos custos dos vizinhos usado no método guloso com o “*GetPossibleStates()”* e no laço de iteração de vizinho a vizinho, onde é feito o cálculo dos custos H e G. O custo H é calculado do mesmo método do guloso, o “CalculateEuclidianDistanceCost()” . Já o custo G é feito utilizando o custo atual de G, somado a mais 1(Pois o custo dos vizinhos sempre vai ser o custo até aqui mais um), caso o nó analisado já esteja na lista fechado, ele ignora e passa para o próximo. Caso ele ainda não esteja na lista aberta, ele é adicionado ao final da execução do laço. O nó atual que já está com seus vizinhos adicionados à lista é removido da lista aberta e adicionado a lista fechado. Por fim, o código retorna a lista de direções que o fantasma deve seguir (note que esse cálculo é feito utilizando o método “CalculatePath()”, onde ele retorna a lista de trás para frente. E,m seguida, é utilizado uma função de “reverse” para colocar os caminhos na ordem correta).

Figura 3 – Codigo-Fonte A\*

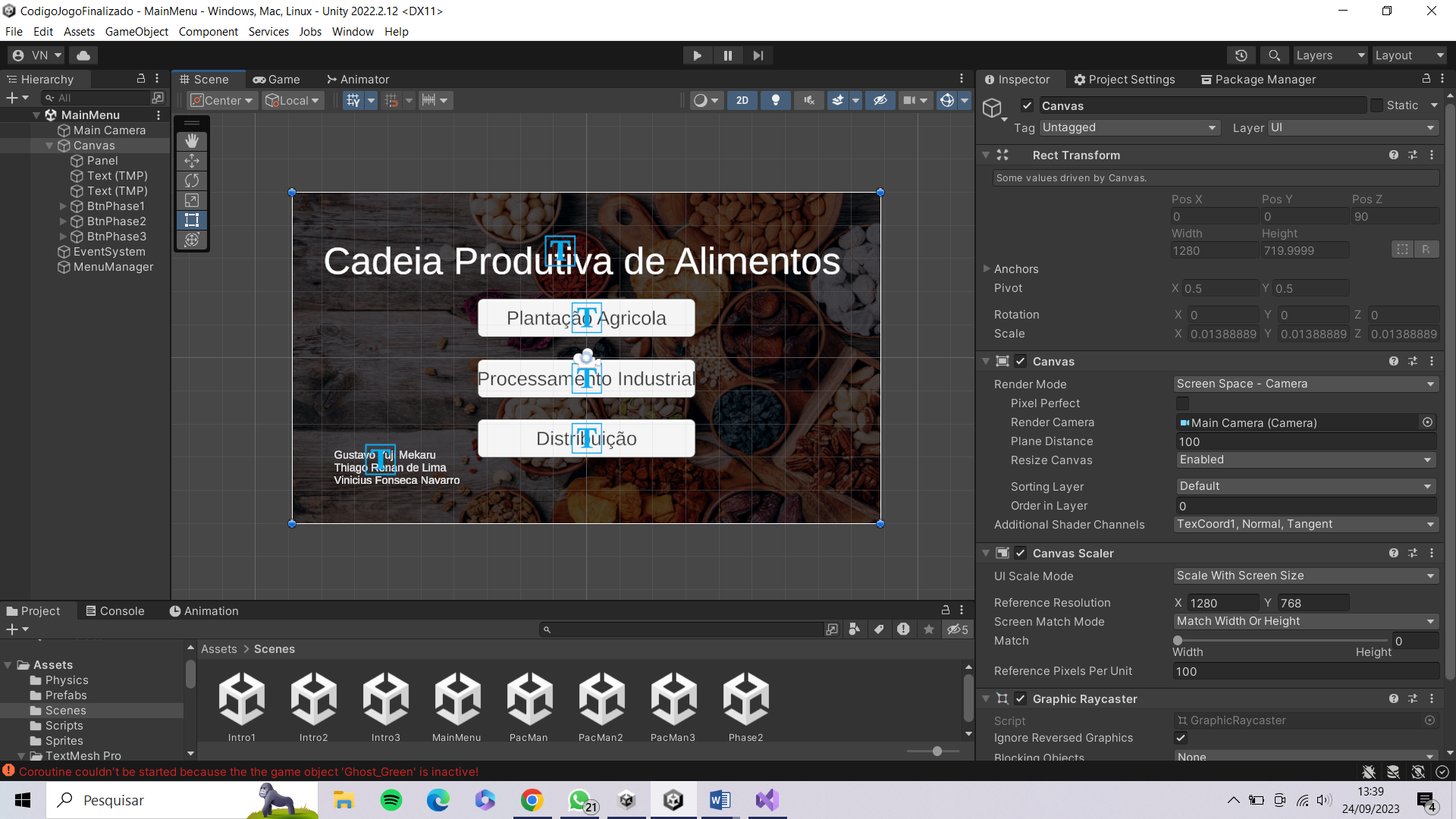


****Fonte: Autoria Propria (2023)****

## 4.3 Desenvolvimento no UNITY

Para poder demonstrar de forma efetiva o funcionamento dos algoritmos de busca e também validar as especificações do projeto como demonstra a figura 4, utilizamos a plataforma Unity para criar um jogo de computador funcional, rodando ambos os algoritmos de busca.

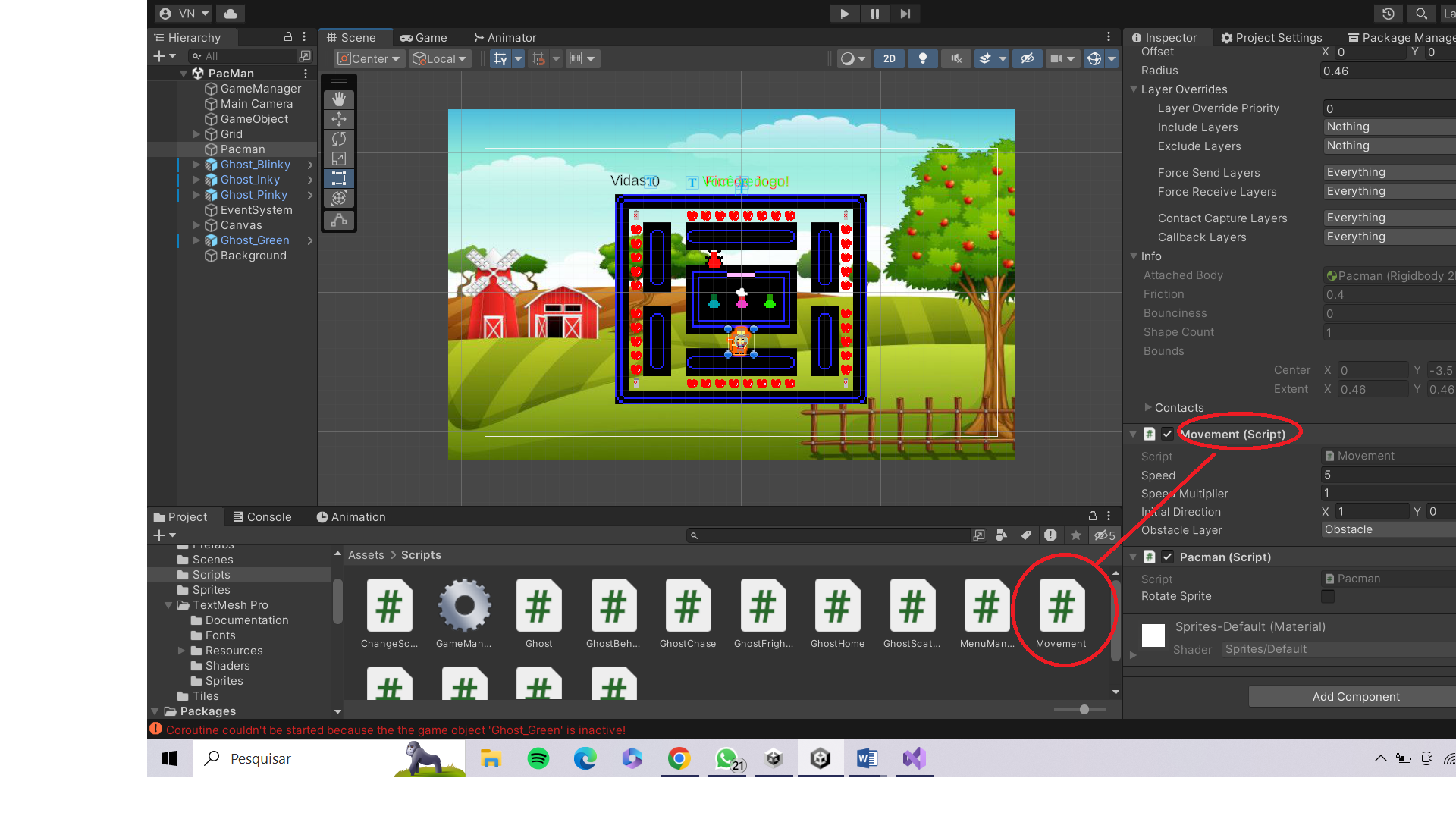
Figura 4 – Print da tela do Unity



**Fonte: Autoria Propria (2023)**

O jogo é divido em 3 fases com os layouts do labirinto alterando em cada fase. Os inimigos são os mesmos em todas as fases apenas alterando o Sprite em cada fase. Cada fantasma apresenta um comportamento distinto: o vermelho é o que está a implementar o algoritmo guloso. Já o rosa utiliza o A\*. Os outros dois (verde e azul) tem uma programação que faz com que eles andem aleatoriamente no mapa sem parar, ou sejam não utilizam os algoritmos de busca. Para fazer a programação no unity, é assumido cada elemento da tela como um objeto único que tem funções associadas a ele. Você pode adicionar esses scripts apenas arrastando eles para dentro do objeto e também mudar algumas de suas configurações (se uma edição dentro do código for definida como publica, será possível alterar aquele campo dentro do unity sem a necessidade de editar o código em si), como mostra a figura 5.

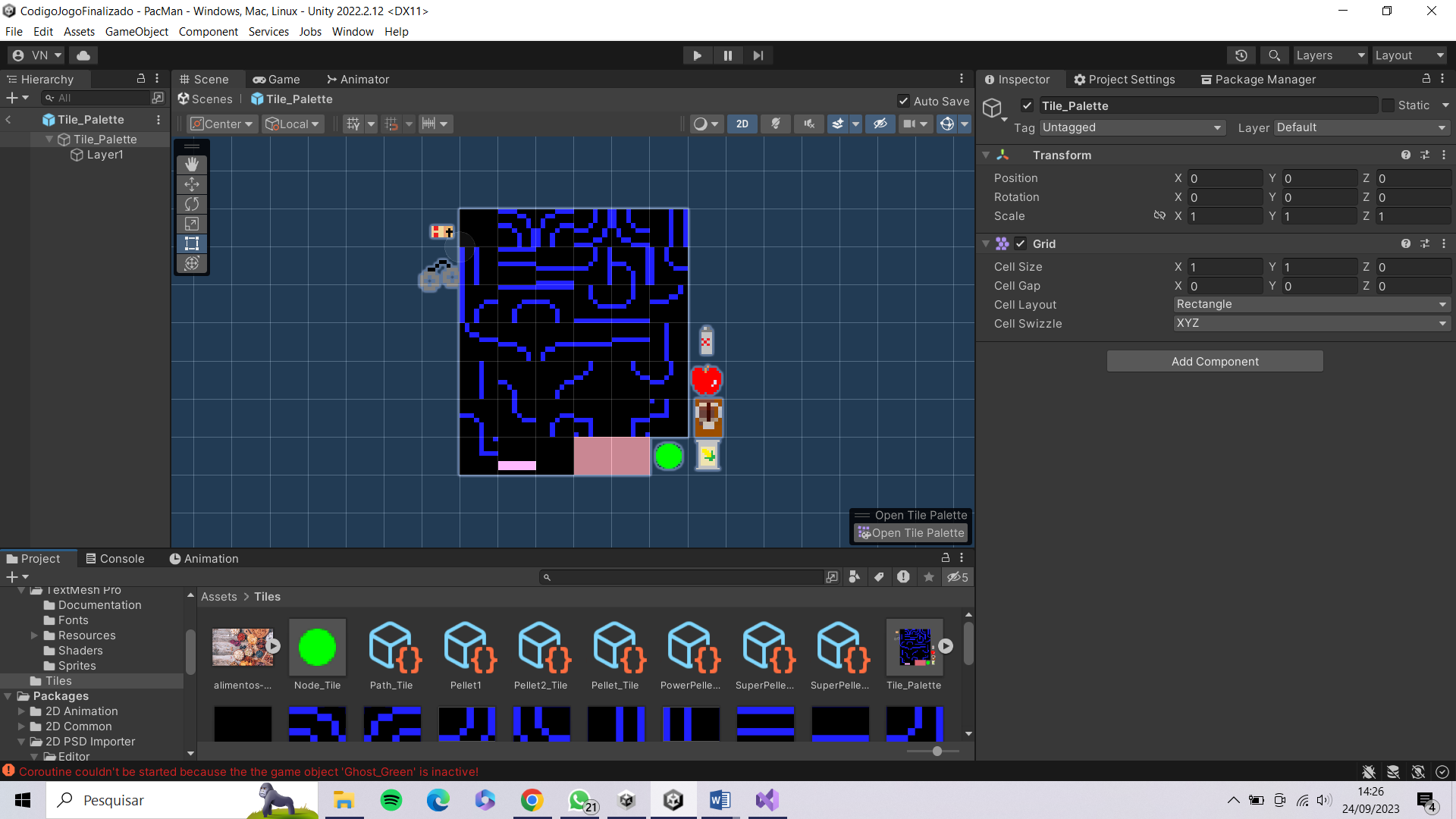
Figura 5 – Adicionar scripts a objetos



**Fonte: Autoria Propria (2023)**

Para adicionarmos os elementos como as paredes e os items que o pacman recolhe, foi utilizado um “Tile Pallet” que usa “prefabs” (Elementos que são pré-fabricados para receberem um script) e servia como uma central de elementos que interagem com jogador. Os obstáculos são montados em vários pedaços de Sprite diferentes e não permitem que o pacman passe por eles .Podeomos observar a paleta escolhida a seguir na figura 6.

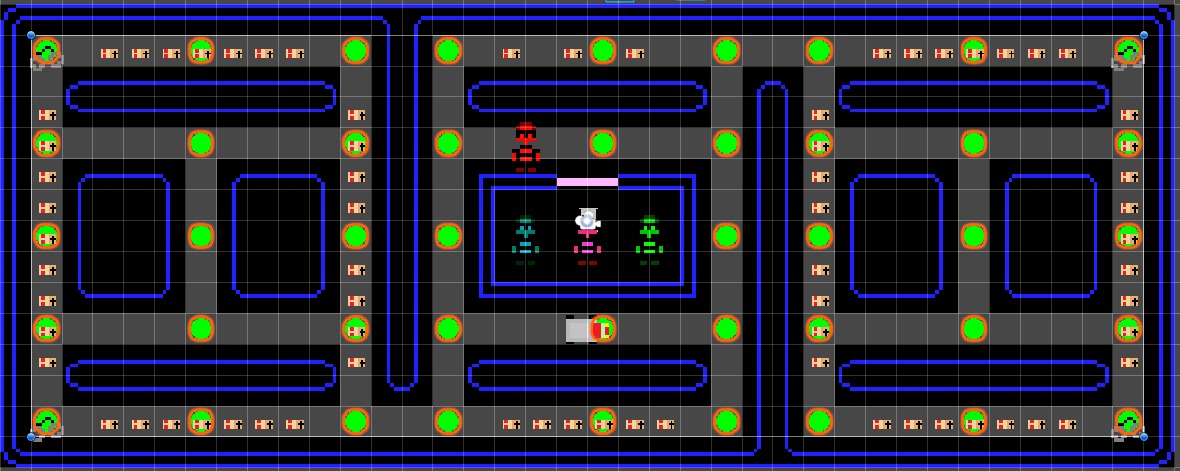
Figura 6 – Tile Pallet



**Fonte: Autoria Própria (2023)**

Como os fantamas devem fazer os cálculos dos algoritmos de busca de forma contínua pelo fato de que o alvo o pacman se move pelo cenário, foi necessário usar os nodes, que são objetos grandes e da cor verde que ficam invisíveis para o jogador. Porem, ao colidir com um fantasma, é ativado o script da busca e diz para o inimigo qual a direção ele deve tomar até que ele colida com outro nó e repita os passos.

Figura 7 – Representação dos Nodes espalhados pelo labirinto



**Fonte: Autoria Própria (2023)**

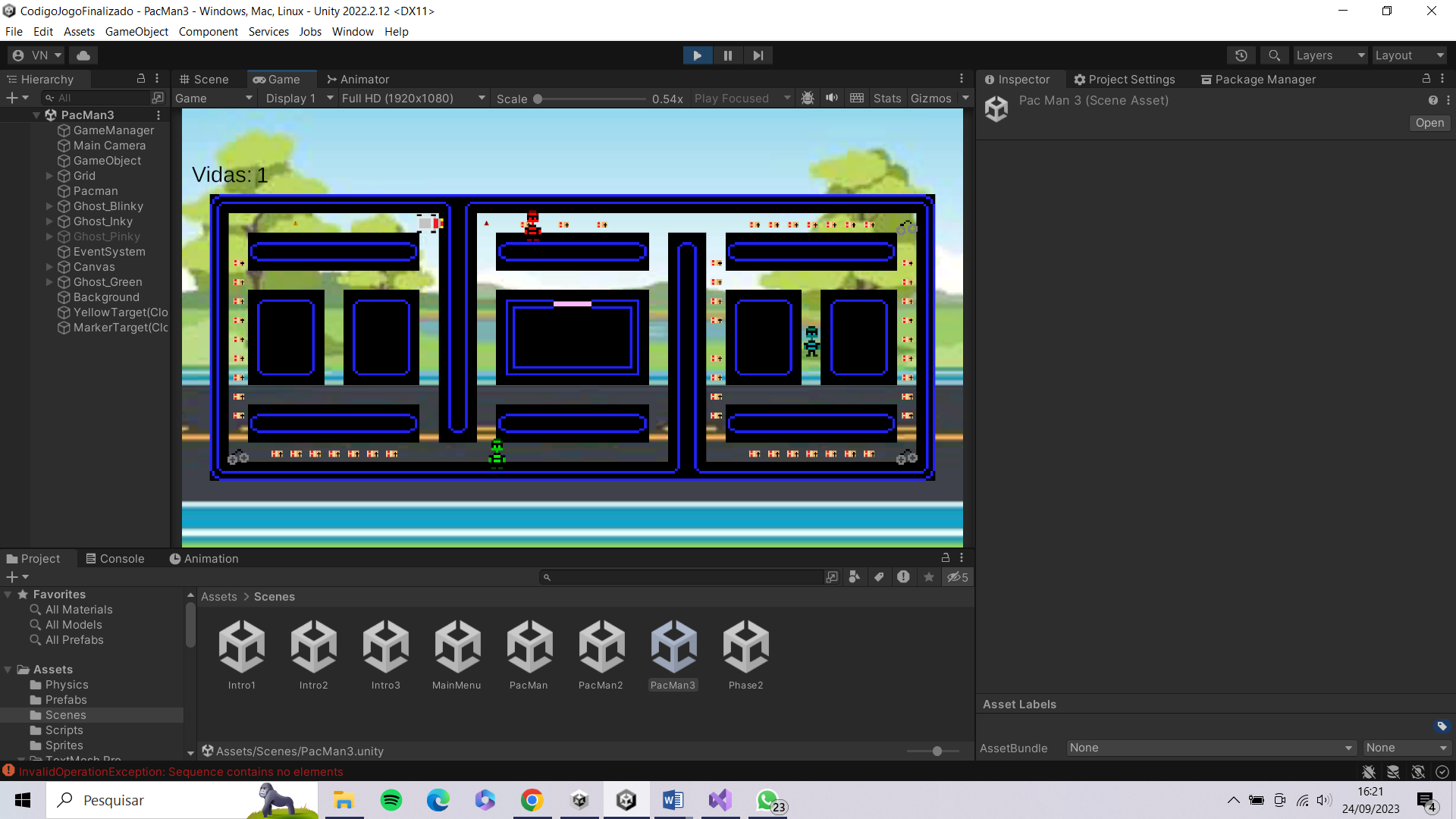
# 5. RESULTADOS

Como foi possível demonstrar ao longo do projeto, o objetivo principal é demonstrar o funcionamento dos algoritmos de busca dentro de um jogo que utiliza inteligência artificial. Com isso em mãos e funcionando, começou a etapa de testes do jogo, onde logo de início ficou claro a diferença na implementação do guloso para o A\*, e como o segundo é mais completo. Já que o algoritmo A\* consegue guardar os custos do início até aquela posição atual e também não descarta os caminhos analisados anteriormente, ele consegue medir de forma mais precisa qual a melhor escolha de caminhos até o objetivo.

**5.1 Problema do Guloso**

Fica possível de evidenciar isso com um exemplo dentro do jogo observando na figura 8, quando o fantasma vermelho (Guloso) chega a uma parede, mas do outro lado da parede está o pacman. Ele não calcula a distância para dar a volta e conseguir chegar no alvo, então percebemos que o sinal de atenção está no local errado e ele não consegue chegar ao objetivo, pois essa distância de dar a volta teoricamente para ele é “maior”, ou seja, pior do que ficar preso a parede. Esse é o principal problema do fato do guloso ser “cego”. Ele não analisa outros tipos de dados e pode acabar assumindo uma postura errada e no fim nunca concluir seu objetivo.

Figura 8 – Exemplo do problema do algoritmo guloso

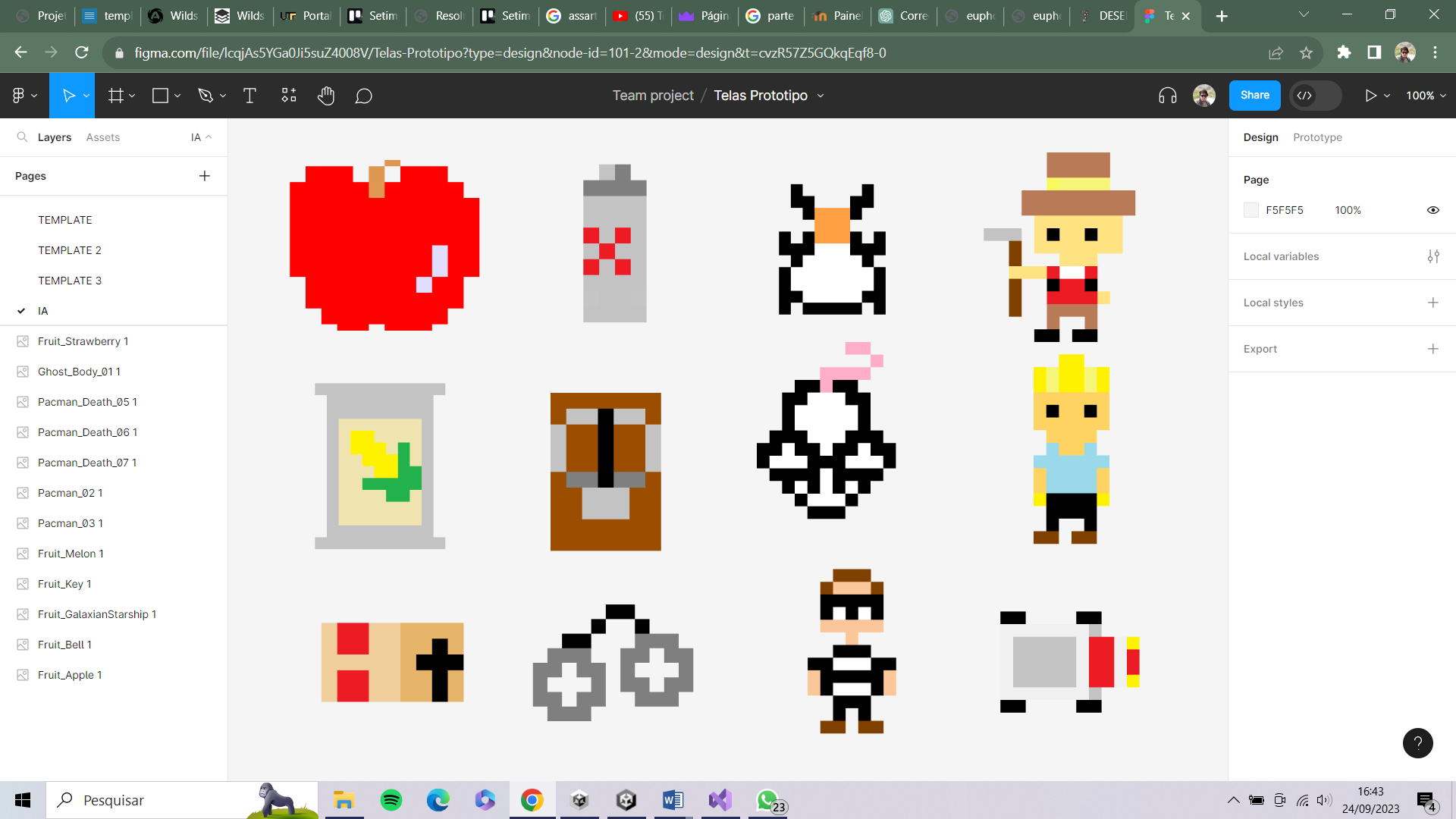


**Fonte: Autoria Própria (2023)**

**5.2 Criando um jogo educacional**

Para realizar o objetivo bônus do projeto, era necessário transformar o design do pacman para um jogo educacional com o tema de cadeia produtiva de alimentos. Para que esse objetivo fosse atingido, foram utilizados sprites (Figura 9) desenhados representando as 3 etapas principais da cadeia moderna de alimentos. A primeira fase se trata de uma fazenda, onde o fazendeiro (pacman) deve coletar as frutas enquanto lida com os besouros (Fantasmas). Esse comportamento se repete para as outras duas fases, sendo a segunda localizada em uma indústria e a terceira é a etapa de transporte dos alimentos.

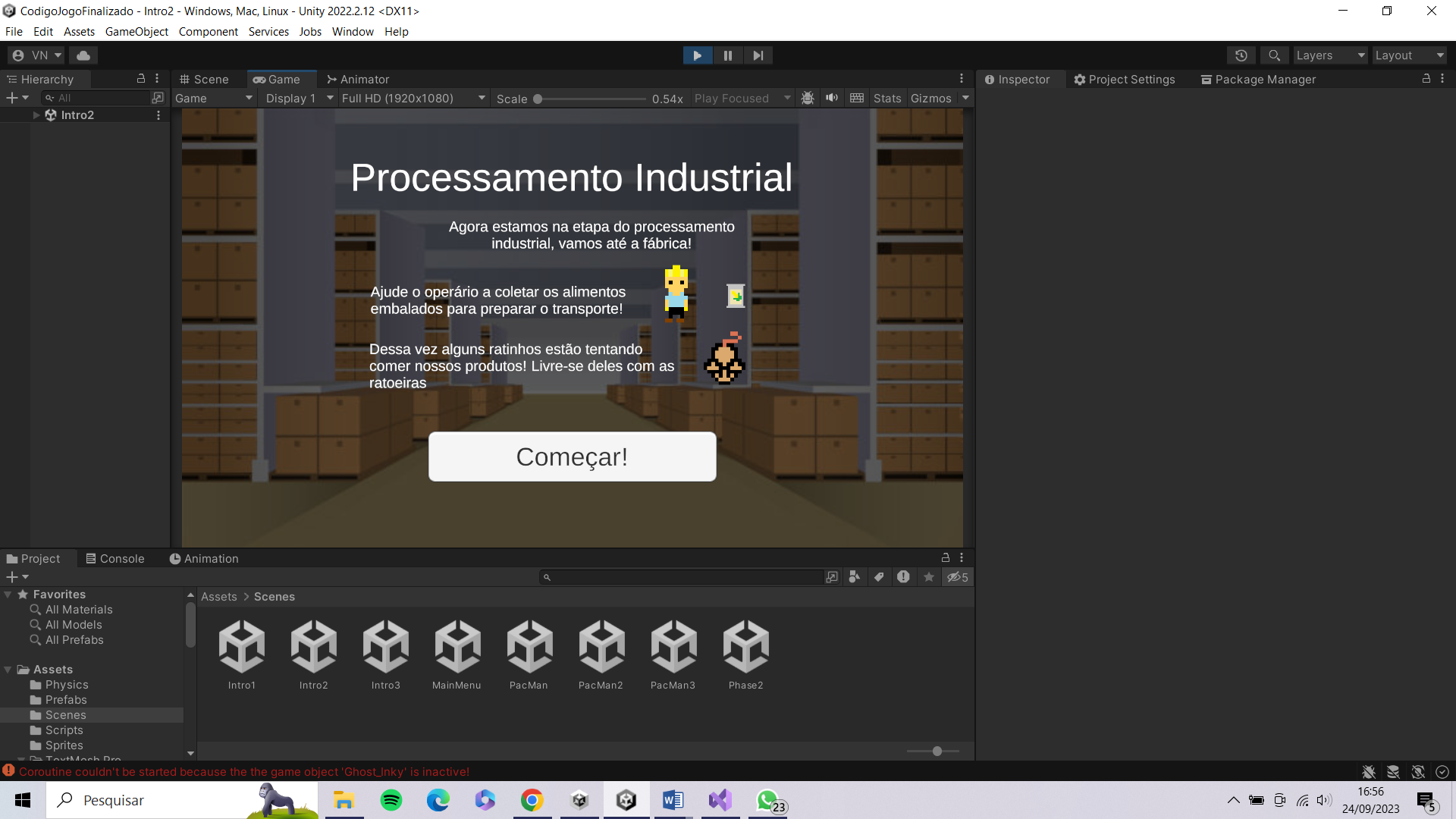
Figura 9 – Sprites utilizados na adaptação



**Fonte: Autoria Própria (2023)**

Com os sprites em mãos, foram criados telas iniciais antes de cada fase para explicar qual processo a fase esta relacionada e falando um pouco sobre os objetivos daquela tela. A Figura 10 demonstra a fase 2 onde se passa em uma indústria onde tem ratos de inimigos e alimentos embalados para recolher.

Figura 10 – Tela de instruções da fase



**Fonte: Autoria Própria (2023)**

Portanto, foi observado que foi possível obter resultados condizentes com os problemas propostos, as fases estão estruturadas dentro da temática e a tela de informação cuida da parte educacional do projeto.

# 6. CONCLUSÃO

Com a implementação do jogo concluída, foi possível identificar diversas dificuldades durante seu desenvolvimento. Especificamente, a parte dos algoritmos de busca foi uma das tarefas mais árduas do projeto. Diversos vídeos de base teórica tiveram que ser estudados para que tanto os algoritmos de busca gulosa e o A\* fossem aplicados de forma satisfatória no comportamento dos fantasmas, mas o resultado final foi suficiente.

Porém, o jogo apresenta alguns problemas, mesmo que mínimos, como a questão de Blinky tentar atravessar paredes, visto que esse é o caminho de menor custo. Isso se dá pela utilização do algoritmo guloso, e não pela implementação no jogo em si. Por mais que foram utilizados alguns ajustes no código para reduzir esse efeito, talvez exista uma maneira melhor de tratar esse comportamento do algoritmo. Outra coisa que pode ser considerado um ponto negativo é a questão dos dois fantasmas apresentarem comportamento aleatório. Essa imprevisibilidade pode gerar uma dinâmica maior ao jogo como descrito durante o trabalho, mas ao mesmo tempo eles podem ficar no canto da janela e/ou serem irrelevantes durante a experiencia do usuário. Talvez se essa aleatoriedade fosse intermediada por outros algoritmos, a experiência seria melhor.

O sistema de nodes atribuído foi bastante efetivo para a configuração dos mapas e movimentação dos fantasmas. Algumas modificações possíveis seriam a remoção de nodos no meio de um corredor, fazendo com que os fantasmas não tenham a possibilidade de alterar sua rota após decidirem um caminho e não terem chegado em uma esquina ainda. Isso possivelmente simplificaria a criação de mapas, tornando possível até a automatização desse processo.

Mas como forma de pesquisa em relação aos algoritmos A\* e busca gulosa, o jogo cumpriu seu objetivo. É possível claramente observar as rotas dos fantasmas com o auxílio da interface gráfica e com uma breve explicação dos algoritmos, é possível identificar visualmente seu comportamento.

# REFERÊNCIAS

HODGES D. Why do Pinky and Inky have different behaviors when Pac-Man is facing up?. 30 dez. 2008. Disponível em: <http://donhodges.com/pacman_pinky_explanation.htm>.

Acesso em: 18 set. 2023.

FORD N. The Productive Programmer. Califórnia: O’Reilly Media, Inc. 2008.

LOSEGO, Alex. Pac-Man Ghost AI Explained. Youtube, 13 jul. 2019. Disponível em: <https://www.youtube.com/watch?v=ataGotQ7ir8>. Acesso em: 18 set. 2023.

RED BLOB GAMES. Introduction to the A\* Algorithm. 26 maio 2014. Disponível em: <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html>.

Acesso em: 18 set. 2023.

RHODES A. A.I.: Informed Search Algorithms. Chapter III: Part Deux. Sem data. Disponível em: <https://web.pdx.edu/~arhodes/ai8.pdf>. Acesso em: 18 set. 2023.

RED BLOB GAMES. Heuristics. Sem data. Disponível em:

<http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>.

Acesso em: 18 set. 2023.

MINGOTO Carlos. Explicação do Algoritmo A\* (A Star). Youtube. 11 dez. 2017. Disponível em: <https://www.youtube.com/watch?v=o5_mqZKhTvw>. Acesso em: 20 set. 2023.

GRAHAM Adam. How to make Pacman in Unity (Complete Tutorial). 21 jun. 2021Disponível em: <https://www.youtube.com/watch?v=TKt_VlMn_aA>.

Acesso em: 6 set. 2023.