UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MURILO LUSVARGHI GARCIA
PAULO HENRIQUE KOROVSKI GAVLAK

RESOLUÇÃO DE PROBLEMAS POR ALGORITMO DE BUSCA

PONTA GROSSA
2025

RESUMO

O presente trabalho apresenta o desenvolvimento de um clone do *videogame* Pac-Man, com o objetivo de atribuir aos fantasmas do jogo a implementação de algoritmos de busca de menor caminho auxiliado por heurísticas, sendo esta a responsável pela dinâmica e dificuldade das fases nele existentes, aproveitando os elementos principais do jogo como forma de adaptar a estrutura lógica empregada por esses algoritmos em um sistema real. Para isso, foram implementados os algoritmos de Busca Gulosa e A*, caracterizando o comportamento de cada fantasma unicamente e servindo como comparativo quando feita a análise empírica do desempenho da busca do fantasma ao jogador. O jogo conta com três fases para uma maior jogabilidade, diferenciando-as por meio da disposição do tabuleiro, quantidade de vidas que sinalizam as tentativas possíveis do jogador de vencer a fase ao obter todos os pontos de vitória disponíveis.

Palavras-chave: Algoritmos de Busca, A*, Busca Gulosa, Pac-Man.

ABSTRACT

This work presents the development of a Pac-Man video game clone, aiming to implement pathfinding algorithms guided by heuristics for the ghosts, enhancing the game's dynamics and difficulty. The main elements of the original game were preserved to adapt the logical structure used by these algorithms in a real system. For this purpose, the Greedy Search and A* algorithms were implemented, with each ghost exhibiting distinct behavior, allowing for empirical performance comparison in tracking the player. The game includes three stages to improve playability, each differentiated by board layout and the number of lives, which represent the player's attempts to win the stage by collecting all available victory points.

Keywords: Search Algorithms, A*, Greedy Search, Pac-Man.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

No contexto deste projeto acadêmico, desenvolvido para a disciplina de Inteligência Artificial (IA) do curso de Ciência da Computação da Universidade Tecnológica Federal do Paraná, foi proposta a recriação computacional do Pac-Man com foco especial na aplicação de algoritmos de busca e heurísticas para controlar o comportamento dos fantasmas. O jogador manipula o Pac-Man manualmente, enquanto os fantasmas agem como agentes inteligentes que utilizam algoritmos como busca gulosa e A* para decidir seus movimentos.

O projeto tem como objetivos principais: a implementação de múltiplos níveis com labirintos distintos, a definição de heurísticas que representem o comportamento individual e busca dos fantasmas.

Este relatório descreve o problema abordado, a metodologia utilizada, os algoritmos implementados, os resultados obtidos e as lições aprendidas durante o desenvolvimento, apresentando uma análise técnica e comparativa das soluções empregadas.

2 DESCRIÇÃO DO PROBLEMA

O desafio central deste projeto consiste em recriar o Pac-Man com ênfase na aplicação de conceitos de Inteligência Artificial para controlar o comportamento dos fantasmas adversários. Diferentemente de uma simples implementação gráfica ou mecânica do jogo, o foco está em construir fantasmas que atuem como agentes inteligentes, capazes de tomar decisões autônomas baseadas em algoritmos de busca.

No jogo original, cada fantasma possui uma “personalidade” distinta, traduzida por padrões de movimentação que aumentam progressivamente a dificuldade para o jogador. Para simular isso computacionalmente, é necessário implementar algoritmos que permitam aos fantasmas decidir qual direção seguir a partir de seu estado atual no labirinto, considerando tanto a posição do Pac-Man quanto os obstáculos ao redor.

Os algoritmos exigem que o mapa do jogo seja representado de forma adequada para permitir a exploração de caminhos possíveis, levando em conta as limitações físicas do labirinto, como paredes e portais. Além disso, é essencial lidar com aspectos como colisão, detecção de vitória (quando o jogador coleta todos os pontos) e derrota (quando o Pac-Man é capturado por um fantasma).

Portanto, o problema não envolve apenas a movimentação básica das entidades no jogo, mas também a aplicação prática de técnicas de IA que demonstrem como agentes podem navegar por ambientes complexos e dinâmicos, tomando decisões baseadas em heurísticas, objetivos e restrições.

3 IMPLEMENTAÇÃO

O projeto Pac-Man desenvolvido busca não apenas replicar o clássico jogo em termos visuais e mecânicos, mas também aplicar algoritmos de Inteligência Artificial para criar fantasmas que se comportem como agentes autônomos e desafiadores. Para isso, o código foi estruturado utilizando o padrão Strategy, o que permitiu desacoplar a lógica de movimentação dos fantasmas da implementação principal do jogo. Cada fantasma possui uma estratégia específica encapsulada em classes concretas que implementam a interface comum *SearchStrategy*, responsável por definir o método *nextDirection*.

Entre os algoritmos implementados, destacam-se dois obrigatórios para o trabalho: o algoritmo guloso (*Greedy*) e o algoritmo A* (A-estrela). Ambos foram aplicados para definir o comportamento de alguns dos fantasmas no labirinto, enquanto outros foram configurados com estratégias alternativas, como movimento aleatório (*RandomMovementStrategy*) é um modelo híbrido com comportamento semi-inteligente (*SemiSmartStrategy*) — esses últimos adicionados por iniciativa própria da equipe, com objetivo de testar diferentes padrões e balancear a dificuldade do jogo.

3.1 ALGORITMO GULOSO (GREEDY)

O algoritmo guloso é caracterizado por sempre tomar a decisão local que parece mais promissora no momento, buscando minimizar a distância entre o fantasma e o Pac-Man a cada movimento, sem considerar o impacto das escolhas a longo prazo. Na prática, a estratégia gulosa calcula a distância heurística (normalmente a distância de Manhattan) entre as posições e decide a direção que mais rapidamente aproxima o fantasma do jogador. Esse comportamento torna o fantasma razoavelmente eficiente, mas vulnerável a armadilhas no labirinto, já que ele não prevê obstáculos futuros ou caminhos fechados.

3.2 ALGORITMO A* (A-estrela)

O A* foi adicionado como uma melhoria significativa, já que combina os pontos fortes da busca gulosa com uma abordagem mais informada e completa. Ele

utiliza uma função de custo composta pela soma do caminho percorrido (g) e da estimativa heurística para o destino (h), garantindo que o fantasma escolha o caminho ótimo até o Pac-Man considerando paredes e obstáculos. Essa estratégia torna os fantasmas equipados com A* muito mais desafiadores, já que eles são capazes de navegar por rotas complexas e evitar becos sem saída, aumentando o realismo e a pressão sobre o jogador.

3.3 ESTRATÉGIAS ADICIONAIS

Além das estratégias obrigatórias, foram incluídas outras abordagens para diversificar o comportamento dos fantasmas. A primeira delas foi a busca em largura, que explora o labirinto expandindo todos os caminhos possíveis a partir do ponto atual até encontrar o Pac-Man, garantindo encontrar a solução mínima em número de passos. Embora seja eficiente em encontrar caminhos curtos, pode ser mais custosa computacionalmente em labirintos grandes ou com muitos nós, já que não utiliza uma heurística de proximidade como o A* ou o Greedy. Foi adicionada também, a “SemiSmartStrategy”**,** que combina uma porcentagem de decisões inteligentes (baseadas na heurística de decisões locais guiadas pela proximidade imediata ao Pac-Man, sem aplicação de algoritmos de busca global) com decisões aleatórias, criando um padrão de movimento que alterna entre perseguir diretamente e se desviar, aumentando a dificuldade para jogadores que tentam explorar previsibilidade.

3.4 MECÂNICA ADICIONAL

À medida que o jogador avança na fase, coletando pontos, o jogo verifica se determinadas metas de progresso foram atingidas. Quando isso ocorre, um dos fantasmas é removido do jogo, aliviando a pressão sobre o jogador e recompensando seu avanço.

3.5 FUNCIONAMENTO GERAL DO CÓDIGO

O centro do jogo é gerenciado pela classe *Game,* que atua como painel principal (JPanel) e coordena os elementos do jogo dentro de um loop de atualização (game loop), que roda continuamente.

O personagem Pac-Man é controlado diretamente pelo jogador, utilizando as teclas direcionais (setas) do teclado. A classe PacMan estende a classe base Block, herdando atributos como posição, tamanho, imagem e velocidade. Ao receber eventos de teclado, o jogo atualiza a direção atual de movimento do Pac-Man, permitindo que ele avance no labirinto, colete pontos (representados como blocos de comida) e interaja com os fantasmas.

Já os fantasmas (instâncias da classe Ghost) também herdam de Block e possuem atributos adicionais, como uma referência à estratégia de movimento atual (*SearchStrategy*). Essa estratégia é implementada usando o padrão de projeto Strategy, que define uma interface comum (*SearchStrategy*) e permite criar múltiplas classes concretas que especificam como o próximo movimento deve ser escolhido. Isso garante que o comportamento de cada fantasma possa ser trocado facilmente apenas substituindo a instância da estratégia, sem necessidade de reescrever a lógica interna do fantasma ou do jogo.

Durante cada ciclo do jogo, os fantasmas verificam se estão alinhados à grade para chamar o método `nextDirection()` de sua estratégia, decidindo a próxima direção antes de executar o movimento e verificar colisões com o Pac-Man. O gerenciamento geral é feito pelo `GameStateManager`, que controla vidas, pontuação, condições de vitória ou derrota, e gerencia a eliminação de fantasmas conforme a porcentagem de comida coletada, garantindo que o progresso do jogador influencie a dificuldade.

O `BoardLoader` carrega o labirinto a partir de uma matriz textual, instanciando paredes, comidas, Pac-Man e fantasmas, permitindo alterar os layouts sem mudar o código. O método central `updateGame()` atualiza Pac-Man e fantasmas, verifica colisões, acumula pontos e redesenha os elementos com o `WindowManager`, garantindo a sincronização entre lógica e gráficos.

Além disso, o código conta com mecânicas refinadas, como a travessia por portais nas bordas e um estado de invulnerabilidade temporária após perder uma vida, evitando que o jogador perca todas rapidamente por causa do processamento veloz. Variáveis de controle foram adicionadas para balancear a frequência de recálculo dos algoritmos de movimento, garantindo ajustes finos na dificuldade.

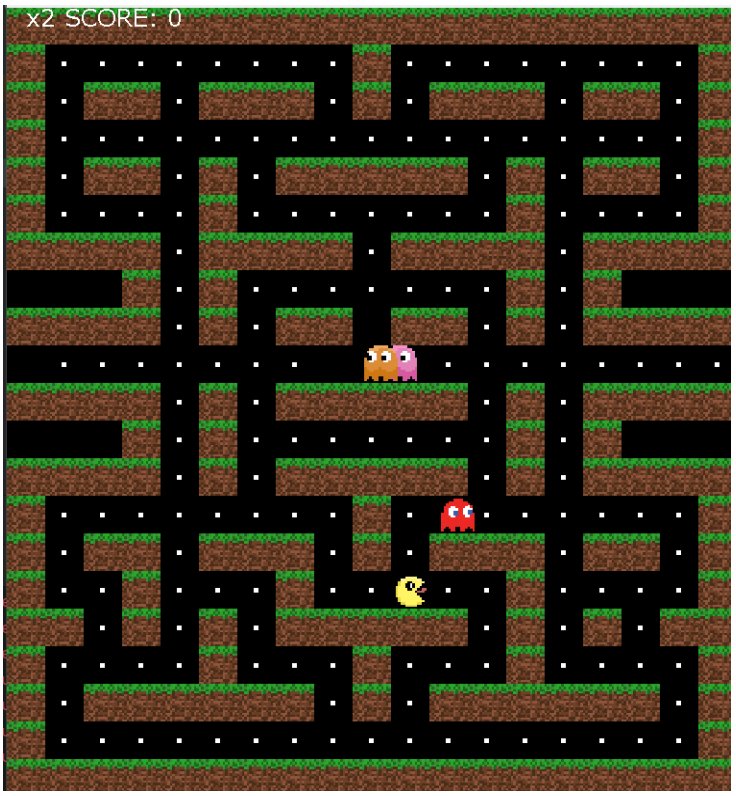
4 RESULTADOS

Como resultado, desenvolvemos um simples menu que permite a escolha entre três níveis de dificuldade, alterando não apenas a velocidade de recálculo do caminho ou a estratégia dos fantasmas, mas também o layout do mapa. Dados como a quantidade de vidas e a pontuação atual podem ser visualizados na parte superior esquerda da tela.

Figura 1 - Tela Inicial

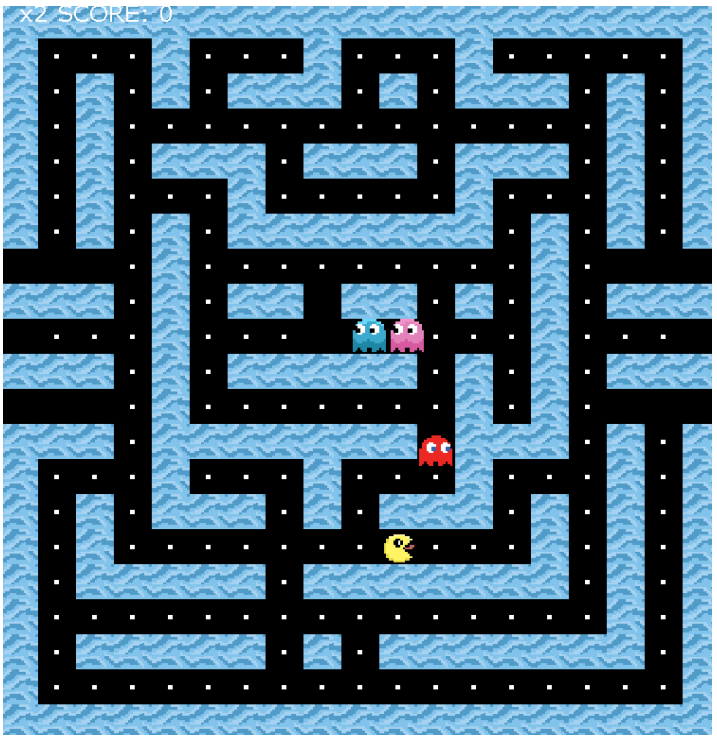
Fonte: autoria própria

Figura 2 - Fase 1



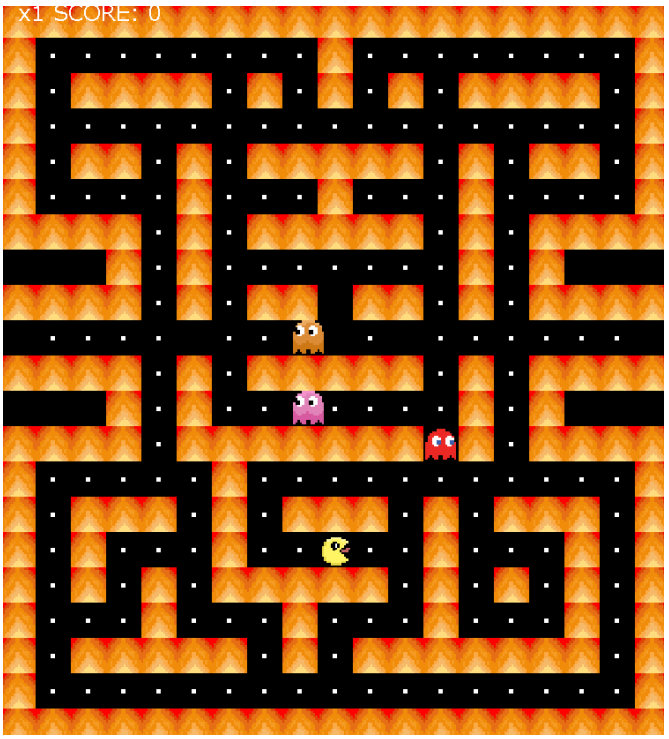
Fonte: autoria própria

Figura 3 - Fase 2



Fonte: autoria própria

Figura 4 - Fase 3



Fonte: autoria própria

Ao atingir a pontuação máxima, é exibido um pop-up indicando que o jogador concluiu a fase. Caso não consiga (ou seja, perca todas as vidas disponíveis), basta pressionar “R” para reiniciá-la.

Figura 5 - Tela de Vitória

Fonte: autoria própria

5 CONCLUSÃO

Ao longo deste projeto, obtivemos resultados satisfatórios, considerando o conhecimento adquirido, os desafios enfrentados durante a implementação e as novas percepções geradas. Foi possível perceber não apenas a complexidade envolvida no desenvolvimento de um jogo, mas também o grande potencial da aplicação de técnicas de inteligência artificial nesse contexto, ampliando nossa visão sobre o tema.

Como produto final, desenvolvemos um simples menu que permite ao jogador escolher entre três níveis de dificuldade, alterando não apenas a velocidade de recálculo do caminho ou a heurística utilizada pelos fantasmas, mas também o layout do mapa. Além disso, foram implementadas mecânicas adicionais: ao atingir a pontuação máxima, um pop-up é exibido informando a conclusão da fase; caso o jogador perca todas as vidas disponíveis, a fase pode ser reiniciada pressionando a tecla “R”.

Esses resultados demonstram que, apesar das dificuldades naturais do processo, a aplicação prática de IA em jogos não só é desafiadora, mas também oferece oportunidades muito interessantes para explorar estratégias inteligentes e aprimorar a experiência do usuário. O projeto serviu, portanto, como um valioso exercício para consolidar conceitos e ampliar nossa compreensão tanto de desenvolvimento de software quanto de inteligência artificial aplicada.

REFERÊNCIAS

RUSSELL, Stuart J.; NORVIG, Peter. *Inteligência Artificial*. 3. ed. São Paulo: Pearson, 2013.

LAVALLE, Steven M. *Planning Algorithms*. Cambridge: Cambridge University Press, 2006. Disponível em: http://planning.cs.uiuc.edu/. Acesso em: 08 maio 2025.

MILLINGTON, Ian; FUNGE, John. *Artificial Intelligence for Games*. 2. ed. Burlington: CRC Press, 2009.