Universidade Federal do Amazonas
Instituto de Computação - ICOMP

Trabalho prático

Trabalho referente à disciplina de
Algoritmos e Estrutura de dados II

Manaus - AM
Fevereiro de 2023

Universidade Federal do Amazonas
Instituto de Computação - ICOMP

Symon Cristhian Ramos Barreto - 22053260
Thales Lopes de Araujo - 22060127

Trabalho prático

Trabalho referente à disciplina de
Algoritmos e Estrutura de dados II

Manaus - AM
Fevereiro de 2023

1. Descrição do problema escolhido

Sliding Puzzle Game

O Sliding Puzzle Game é um tipo de jogo de quebra-cabeça em que o objetivo é mover peças em uma grade para formar uma imagem completa. O jogo é bastante popular e pode ser encontrado em diversas plataformas, como aplicativos para celular, jogos de navegador e até mesmo em consoles de videogame.

O jogo é baseado em uma grade, que pode variar em tamanho, dependendo do nível de dificuldade escolhido. A grade é dividida em várias peças que podem ser movidas pelo jogador. O objetivo é movê-las em uma sequência lógica, a fim de formar uma imagem completa.

O jogo pode parecer simples à primeira vista, mas, na verdade, é bastante desafiador. À medida que o jogador progride nos níveis de dificuldade, o número de peças na grade aumenta, tornando a resolução do quebra-cabeça mais difícil. Além disso, as peças podem ser movidas em várias direções, o que aumenta ainda mais a complexidade do jogo.

Para resolver um Sliding Puzzle Game, é necessário ter um pensamento lógico e estratégico. O jogador precisa analisar a posição atual das peças e pensar em qual movimento deve ser feito para alcançar o objetivo. À medida que o jogador avança no jogo, é possível aprender técnicas e truques para tornar a resolução mais fácil.

Além de ser um jogo divertido, o Sliding Puzzle Game também é benéfico para o desenvolvimento cognitivo. Ele ajuda a melhorar as habilidades de resolução de problemas, raciocínio lógico, memória e coordenação motora. Por esses motivos, o jogo é frequentemente recomendado por profissionais da área de educação e saúde mental.

Solver

Resolver o jogo pode ser um desafio, mas um botão de solver pode ajudar os jogadores a encontrar uma solução rapidamente.

No entanto, implementar o algoritmo A* em um botão de solver para um Sliding Puzzle pode ser um desafio significativo. Isso ocorre porque o A* é um algoritmo de busca em grafo que requer uma representação precisa do espaço de estados do problema, bem como uma heurística para estimar o custo restante para chegar ao objetivo.

Para o Sliding Puzzle, o espaço de estados é enorme, e pode ser necessário explorar muitos caminhos para encontrar a solução correta. Além disso, uma heurística imprecisa pode

levar o algoritmo a encontrar soluções subótimas ou até mesmo falhar em encontrar uma solução em um tempo razoável.

Outro desafio é que o A* pode ser sensível a erros de implementação, como a seleção inadequada de uma função de custo, a falta de gerenciamento de memória eficiente ou a escolha de uma estrutura de dados ineficiente para manter a lista aberta.

Em resumo, o desafio em usar o algoritmo A* em um Sliding Puzzle game é encontrar a representação correta do espaço de estados, implementar uma heurística precisa, gerenciar efetivamente a memória e a lista aberta e resolver problemas de implementação que possam surgir.

2. Procedimento de busca implementado

O algoritmo A* é um algoritmo de busca que encontra o caminho mais curto entre um nó inicial e um nó objetivo em um grafo ponderado. Ele usa uma função heurística para estimar o custo mínimo para chegar ao nó objetivo e uma lista de prioridade para escolher o próximo nó a ser explorado. O algoritmo expande os nós com menor custo total primeiro e mantém uma lista de nós visitados para evitar ciclos. O A* é considerado um dos algoritmos de busca mais eficientes e é amplamente utilizado em aplicações de inteligência artificial, como jogos, robótica e planejamento de trajetórias.

Para implementar o algoritmo A* nesse contexto, o primeiro passo é definir a heurística. A heurística é uma função que estima o custo mínimo para chegar ao objetivo. No caso do Sliding Puzzle Game, uma heurística comum é a soma das distâncias horizontais e verticais de cada peça até sua posição final.

Em seguida, o algoritmo A* é executado, explorando o espaço de busca e procurando uma solução ótima. O espaço de busca é definido por um grafo, em que cada estado representa uma configuração das peças do quebra-cabeça e cada transição representa um movimento de uma peça para uma posição adjacente vazia.

Durante a busca, o algoritmo mantém uma lista de nós abertos, que representa os estados a serem explorados, e uma lista de nós fechados, que representa os estados já explorados. O algoritmo escolhe o próximo nó a ser explorado com base em sua função de custo $f(n) = g(n) + h(n)$, onde $g(n)$ é o custo real para chegar ao nó e $h(n)$ é a heurística.

Uma vez que a solução é encontrada, o algoritmo retorna uma sequência de movimentos que levam do estado inicial ao estado objetivo. Essa sequência pode ser usada para animar o quebra-cabeça e mostrar ao jogador a solução ótima.

3. O Projeto

A ideia do nosso projeto seria que o nosso programa ao ser executado tentasse realizar uma série de resoluções com um número máximo de tentativas, onde cada tentativa seria um nó da fila de prioridades que foi implementada, do 8-puzzle game, por padrão escolhemos dez tentativas mas esse número pode ser alterado facilmente no código, e de acordo com as tentativas ele mostra no console os casos que ele conseguiu resolver com quantas tentativas e em quanto tempo e os casos que ele não conseguiu resolver dado um número máximo de tentativas, por padrão nós passamos o valor de três mil tentativas entretanto esse valor pode também pode ser facilmente alterado dentro do código.

Após isso, nosso programa irá abrir uma interface com 8-puzzle game para que o jogador possa tentar resolver alguns casos e após isso comparar seus tempos com os tempos de resolução do programa. O jogo se inicia ao clicar no botão “Shuffle” onde irá gerar um estado aleatório e irá dar start no cronômetro. O cronômetro para quando o 8-puzzle for resolvido e caso o tempo do jogador seja o menor que o menor valor de tempo já registrado esse novo valor é armazenado em um txt chamado high score e o valor do high score que é exibido na interface do jogo e atualizado. Ao clicar reset o jogo volta para o estado inicial.

A ideia inicial era que na interface interativa também fosse exibidas as tentativas realizadas pelo computador entretanto pela falta de experiência com a biblioteca pygame muitas vezes acabamos quebrando a aplicação quando tentamos rodar a função que realiza a resolução por si própria dentro do jogo então optamos pela abordagem citada anteriormente

4. Decisões de projeto

4.1 Pygame

O Pygame é uma biblioteca em Python que fornece funcionalidades para a criação de jogos 2D. Ela é amplamente utilizada por programadores de jogos devido à sua facilidade de uso e flexibilidade. O Pygame fornece uma série de ferramentas para simplificar o processo de criação de jogos, incluindo módulos de som, gráficos e entrada de teclado/mouse.

Uma das principais razões para escolher o Pygame para o desenvolvimento de um Sliding Puzzle Game é sua capacidade de manipular gráficos em tempo real. O jogo em questão requer uma interface gráfica com o usuário para que o jogador possa mover as peças do quebra-cabeça, e o Pygame permite criar facilmente essa interface.

Outra razão é que o Pygame é uma biblioteca multiplataforma, ou seja, o jogo pode ser executado em diferentes sistemas operacionais, incluindo Windows, Linux e Mac OS. Isso é importante para desenvolvedores que desejam que seu jogo seja acessível a um público mais amplo.

Além disso, o Pygame é uma biblioteca bem documentada e possui uma grande comunidade de desenvolvedores, o que significa que é fácil encontrar exemplos de código e soluções para problemas comuns. Isso pode economizar muito tempo e esforço no desenvolvimento do jogo.

Por fim, o Pygame é uma escolha popular entre desenvolvedores de jogos independentes, e seu uso pode facilitar a participação em comunidades de desenvolvimento e publicação de jogos.

Em resumo, a escolha do Pygame para o desenvolvimento de um Sliding Puzzle Game é justificada pela sua capacidade de manipular gráficos em tempo real, ser multiplataforma, ter uma documentação abrangente e uma grande comunidade de desenvolvedores e ser popular entre desenvolvedores de jogos independentes.

4.2 Fila de Prioridades

Uma fila de prioridades é uma estrutura de dados que armazena elementos e os organiza com base em suas prioridades, permitindo que o elemento com a maior prioridade seja acessado primeiro.

Essa estrutura de dados é semelhante a uma fila, na qual os elementos são adicionados no final e removidos do início. No entanto, na fila de prioridades, cada elemento é atribuído a uma prioridade, e o elemento com a maior prioridade é colocado na frente da fila.

Existem várias formas de implementar uma fila de prioridades, incluindo o uso de uma árvore binária, um heap binário ou uma lista encadeada ordenada.

4.3 Interface Interativa

O jogo é baseado em uma grade quadrada de tamanho GAME_SIZE (definido no arquivo de configurações "settings.py") que contém peças numeradas em ordem sequencial, sendo a última peça vazia. O objetivo do jogo é deslizar as peças na grade para reorganizá-las em ordem numérica crescente, deixando a última célula vazia no canto inferior direito.

O jogo usa vários arquivos: "sprite.py" contém as classes para os sprites que aparecem na tela, como o botão, o título e as peças do quebra-cabeça; "settings.py" contém as configurações do jogo, como o tamanho da tela, as cores usadas e a taxa de quadros; e "high_score.txt" armazena os tempos mais rápidos de conclusão do quebra-cabeça.

A classe Game tem um construtor init() que inicializa várias variáveis e configurações necessárias para o jogo, como a janela do jogo, a taxa de atualização e o título da janela. Também tem um método get_high_scores() que lê os scores de um arquivo de texto e retorna uma lista.

O método new() é responsável por inicializar o jogo, incluindo o embaralhamento das peças e a criação dos botões.

O método shuffle() é chamado quando o jogador clica no botão shuffle e é responsável por embaralhar as peças do jogo.

O método draw_tiles() é responsável por desenhar as peças na tela.

O método create_game() é responsável por criar o jogo com as peças na ordem correta.

O método run() é responsável por iniciar o jogo e limitar a taxa de atualização.

O método update() é responsável por atualizar a lógica do jogo.

Por fim, o método draw_grid() é responsável por desenhar as linhas do grid na tela e o método draw() é responsável por desenhar todas as peças e botões na tela.

No geral, o código cria um jogo divertido e desafiador de quebra-cabeça deslizante com várias funcionalidades, como embaralhar as peças, verificar o tempo mais rápido e armazenar os tempos mais rápidos em um arquivo de texto.

4.4 Fila de Prioridades

Foi implementada uma fila de prioridades que é utilizada pelo nosso programa para tentativa de resolução dos casos que são gerados aleatoriamente. Para isso temos os seguintes componentes:

- **META:** variável que guarda o estado de resolução do jogo (*[[1,2,3],[4,5,6],[7,8,0]]*)
- **class NO():** responsável por guardar o estado e também qual seu estado pai e o custo até chegar naquele estado, a ideia é que cada nó fosse uma tentativa que gera um estado.
- **def solucionavel(lista):** responsável por verificar se a partir de um estado se é possível chegar a uma solução.
- **def geraInicial(estado=META[:]):** responsável por gerar uma configuração aleatória que sera o estado inicial de onde iremos partir até chegar ao estado final
- **def localizar(estado,elemento=0):** responsável por buscar no grid o bloco com valor passado por parâmetro
- **def distanciaQuarteirao(estado1,estado2):** responsavel por calcular a distância de um bloco,número de linhas e colunas, no estado passado para o estado de resolução
- **def criarNo(estado,pai,gx=):** função chamada ao gerar uma tentativa para que essa possa entrar na fila de prioridades
- **def inserirNoFilaPrioridades(no,fila):** realizar a inserção de uma tentativa na fila de prioridades obedecendo às regras da fila de prioridades
- **def moverAbaixo(), def moverAcima() ,def moverEsquerda(), def moverDireita():** responsável por realizar a movimentação dos blocos
- **def possiveisMovimentos(no):** Dada uma tentativa essa função verifica quais possíveis novos estados podem ser atingidos e retorna uma lista desses estados
- **def buscaResolucao(numMaxTentativas,noInicial):** função que utilizando as funções anteriores parte do Inicial, que contém o estado inicial, e tenta chegar a um estado final em numero maximo de tentativas, proveniente do parâmetro numa Tentativas.
- **def puzz8(maxProfundidade,numAmostras):** para um determinado numero de amostras essa função executa a resolução do 8-puzzle game e ao final ela mostra os casos solucionados dentro do número de tentativas e os que não foram selecionados pois estouraram os números máximos de tentativas

- **solucao = puzz8(3000,10)** *//exemplo de como fazer a chamada para o programa tenta fazer as resoluções do 8-puzzle*

5. Instruções de instalação e uso

Clone o repositório: https://github.com/s4imu/AED2_8Puzzle para o seu computador.

Certifique-se de ter o Python e o Pygame instalados no seu computador. Caso não tenha, você pode instalá-los executando os seguintes comandos no terminal:

```
sudo apt-get update
sudo apt-get install python3
sudo apt-get install python3-pygame
```

Abra o terminal e navegue até o diretório que contem o arquivo main.py

Execute o seguinte comando para iniciar o jogo:

```
python3 main.py
```