RELATÓRIO IF684-EC

Sistemas Inteligentes

Arthur Henrique Tavares de Lyra Costa

Bezaliel da Conceição Silva

Luis Felipe Batista Pereira

Jose Maycon Lima Cunegundes da Silva

Marcos Heitor Carvalho de Oliveira

Teresina

09/03/2022

Sumário

1 Identificação

Esse exercício teve os seguintes integrante:

- Arthur Henrique Tavares de Lyra Costa (ahtlc).
- Bezaliel da Conceição Silva (bcs5@cin.ufpe.br).
- Luis Felipe Batista Pereira (lfbp@cin.ufpe.br).
- Jose Maycon Lima Cunegundes da Silva (jmlcs@cin.ufpe.br).
- Marcos Heitor Carvalho de Oliveira (mhco@cin.ufpe.br).

2 Processo

Inicialmente dividimos o trabalho para os integrantes do grupo. Cada integrante ficou com uma parte específica.

Implementamos a geração do mapa utilizando Perlin Noise, onde o agente e a comida são gerados de forma aleatória no mapa em pontos que não são uma barreira. Para cada intervalo numérico de geração do Perlin Noise definimos um terreno específico, seguindo a orientação do exercício para definir o peso de cada terreno.

Os algoritmos foram implementados seguindo o que foi proposto pelo professor da disciplina e com base na literatura. Foram feitas apenas algumas alterações nos algoritmos para deixar mais visível o passo-a-passo da execução. As estruturas usadas de cada algoritmo seguem o padrão da figura 1. A função de heurística usada foi a distância de Manhattan.

BUSCA	ESTRUTURA	FUNÇÃO
LARGURA	FILA	-
PROFUNDIDADE	PILHA	-
CUSTO UNIFORME	LISTA PRIORIDADE	$f(n) = g(n)$
GULOSA	LISTA PRIORIDADE	$f(n) = h(n)$
A*	LISTA PRIORIDADE	$f(n) = g(n) + h(n)$

Figure 1: Resumo das estruturas de dados usadas em cada busca

A GUI foi implementada logo em seguida. Toda interface foi feita pelos integrantes do grupo já que o suporte para bibliotecas gráficas em Python do P5 não é tão abrangente.

Por fim, implementamos a colisão entre o agente com a comida e integramos toda a parte lógica com a interface do usuário.

2.1 Desafios

- Escolher algoritmo de geração de mapas, de início pensamos em cellular automata, mas depois vimos que Perlin Noise era o ideal.
- Fazer com que o código dos algoritmos de busca fossem reutilizáveis, conseguimos que Greedy, Dijkstra e A* usassem a mesma função de passo.
- Achar uma boa biblioteca de interface para Jython.
- Manter qualidade do código, Python dá muita liberdade mas pode tender a ficar bagunçado

com muita coisa espalhada. Um exemplo é a classe Grid que mantém todo o contexto do mundo gerado, tanto quem foi visitado, visto, distância e antecessor no caminho.

2.2 Erros

- Comida surge em lugares inalcançáveis pelo agente.
- Bugs na FSM(finite state machine) por conta das transições de contexto no processo de computar a cada frame.

2.3 Aprendizado

Perlin Noise 2D: parametrização com ajustes no nível de detalhes e camadas. Tentamos ajustar de forma a ter uma aparência de terreno interessante, água do lado de lama, lama do lado de areia e áreas altas(walls) centralmente. Além disso, Jython não é tão amigável.

3 Divisão do Trabalho

De início, todos fizeram pesquisas sobre o funcionamento dos algoritmos que seriam utilizados, e como seria a aplicação de tais. Assim, começamos a nos dividir entre áreas sendo elas: geração de mapa, algoritmos, interface e movimentação do agente. Entretanto caso ocorresse algum problema ou trava durante o desenvolvimento chamávamos outro participante para atuar na mesma tarefa.

4 Arquitetura e Tecnologias usadas

4.1 Arquitetura

Na parte de arquitetura utilizamos hierarquia para modularizar algumas buscas, também dividimos a parte da lógica e a parte da visualização. Além disso, criamos nossa própria interface,

porque as bibliotecas gráficas de processos não se encaixavam tão bem. Dividimos entre GRID e Pathfinding para facilitar nosso trabalho. Sendo GRID responsável por visto, visto, distância de veículo até a célula e o peso/tipo. Enquanto Pathfinding tem todos os algoritmos de encontrar caminho.

4.2 Tecnologias

Na parte de tecnologias utilizamos Python e Processing 3, além de algumas libs embutidas.