Sérgio Rosendo da Silva Júnior – 6508702
MAC 0121 – Algoritmos e Estruturas de Dados I - 28/11/2016

EP 4 – Hex

1. Decisões de Projeto

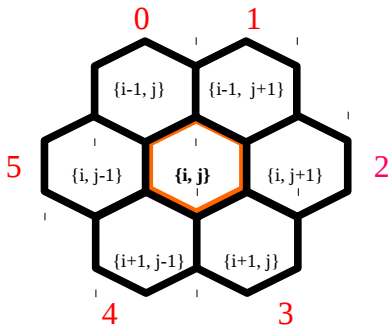
O programa faz uso de diversas variáveis globais a fim de torná-lo mais modular e de fácil alteração. Por exemplo:

BOARD_LINES => definido em board_management.h
BOARD_COLUMNS => definido em board_management.h

Indicam o tamanho do tabuleiro de hex a ser criado, todas funções estão preparadas para funcionar sobre qualquer tamanho de tabuleiro.

index neighbors [6] = {{-1, 0}, {-1, 1}, {0, 1}, {1, 0}, {1, -1}, {0, -1}}

Indica a posição relativa de cada vizinho de uma célula no tabuleiro de hex.



Em toda função para se obter os índices de um vizinho ‘k’ calcula-se [line+neighbors[k].lin] e [column+neighbors[k].col].

Outras variáveis globais são definidas em ai_strategy.c, elas são usadas pela IA do programa para julgar qual a melhor jogada a cada turno.

```
int _cell_position_priority_multiplier = 3;
int _cell_is_dead_or_filled = 0;
int _cell_I_win = 900;
int _cell_enemy_win = 300;
int _cell_is_a_threatened_two_bridge = 12;
int _cell_can_generate_two_bridges = 5;
int _cell_creates_wall = 6;
int _cell_creates_rope = 3;
```

Elas são melhores explicadas mais adiante.

1.1 - Sobre a detecção do fim do jogo

O algoritmo *upper_connected_to_start (char **board, int line, int column, char c)* é responsável por tornar maiusculo o caractere referente a toda peça que está conectado a sua borda de partida (linha 0 para peças brancas e coluna 0 para peças pretas) através de outras peças da

mesma cor. Desta forma, se uma posição do tabuleiro contém ‘B’, isso indica que a partir desta posição é possível fazer um caminho até a linha 0 através de outras peças ‘B’.

O algoritmo check_victory confirma a vitória do jogador branco se na última linha do tabuleiro houver uma peça ‘B’ ou confirma a vitŕoa do jogador preto se a na última coluna houver uma peça ‘P’.

A impressão do tabuleiro é mantida com as peças em letras minusculas e ‘-’ para posição vazia.

1.2 – Arquivos C presentes:

hex.c – função main

auxiliar_ep5.c e ***auxiliar_ep5.h*** – funções auxiliares como create_matrix e destroy_matrix.

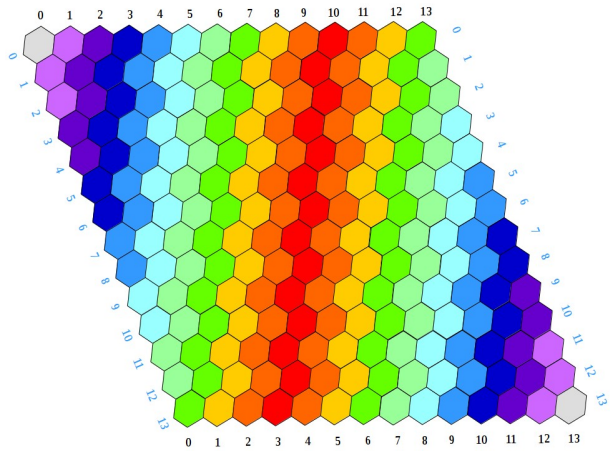
board_management.c e **board_management.h** – funções que manejam o estado do tabuleiro.

ai_strategy.c e ***ai_strategy.h*** – funções responsáveis pela inteligência artificial e pelos movimentos no jogo.

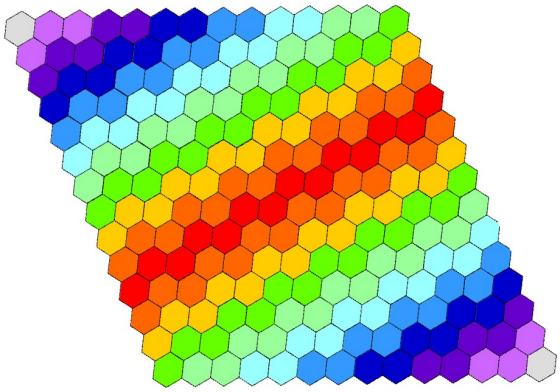
2. Estratégia da IA

Para cada posição no tabuleiro, considerou-se que ela tem uma prioridade diferente para cada jogador. Esta prioridade é tão maior quanto mais fácil for, para aquele, conectar seus lados opostos do tabuleiro passando por dada posição. Observe a seguir, o vermelho no centro indica a máxima prioridade para cada jogador, a prioridade diminui a medida que se aproxima das células cinzas.

Jogador Branco

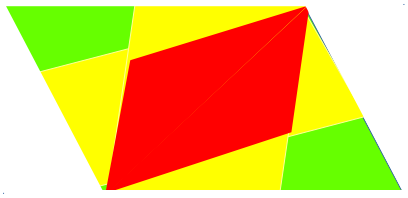


Jogador Preto



Embora a distância mínima entre os lados opostos seja a mesma para algumas das “linhas” de cores distintas, algumas tem prioridade maior do que as outras porque a partir dessas existem mais opções para desvio de trajeto, quando o desvio se faz necessário, a fim de atingir-se o objetivo com o menor número de peças possível.

No hex a defesa é também o ataque, a medida em que bloqueia a passagem do adversário um jogador é capaz de formar sua própria conexão entre as bordas. Por isso é interessante somar a prioridade de ambos os jogadores e utilizar o resultado como sua prioridade própria. Desta forma dá-se enfá-se as posições que são úteis a ambos, facilitando sua vitória e dificultando a vitória do inimigo. Observe no esboço (muito simplificado) a seguir o resultado da soma das prioridades num só gráfico, novamente, a região vermelha é a de maior prioridade.



Como é fato conhecido sobre o jogo Hex, o centro do tabuleiro é uma região valiosa.

Este programa faz uso desse raciocínio, ele utiliza uma grade com a soma das prioridades de cada jogador para cada posição. Além disso, a IA aplica a pie rule se o adversário começar o jogo em uma posição com prioridade alta.

2.1 Outros valores de prioridade calculados

Para cada celula, após cada jogada, é recalculado o valor de sua prioridade considerando a condição atual do tabuleiro. Isso é feito pela função ***enhance_priority_board (char **board, int **priorityBoard, char my_color, char enemy_color)***, esta função checa diferentes configurações locais e globais do tabuleiro a fim de aumentar a prioridade das celulas mais importantes para a IA. Ela também leva em consideração as melhores posições para o adversário (ou seja, considera o "esboço" apresentado acima, em mais detalhes) a fim de ocupar as melhores posições do tabuleiro.

A IA jogará sua peça naquela posição em que a soma (prioridade (branco) + prioridade (preto)) é máxima, no intuito de ligar suas bordas enquanto impede o adversário de fazer uma de suas melhores jogadas.

Para o calculo da nova prioridade de uma celula é levado em consideração, qual a nova configuração do tabuleiro se adicionada uma peça a mesma, além da posição relativa que já foi discutida acima.

As variaveis globais definidas em *ai_strategy.c* indicam o peso de cada estrutura formada quando se adiciona uma peça a uma celula.

int _cell_position_priority_multiplier = 4;
=> Quanto mais alto, mais o algoritmo dá enfase ao centro do tabuleiro.

int _cell_is_dead_or_filled = 0;
=> Uma celula 'dead' nunca altera o resultado do jogo e portanto recebe prioridade 0. Assim como celulas já prenchidas que obviamente não podem ser alteradas.

int _cell_I_win = 900;
=> Uma celula em que se jogada, é ganha pela IA em sua jogada, recebe prioridade máxima.

int _cell_enemy_win = 300;
=> Uma celula em que, na próxima jogada do adversario, permite que eloe ganhe o jogo recebe a segunda maior prioridade, a fim de evitar ou postergar enquanto possível a vitória do adversário.

int _cell_is_a_threatened_two_bridge = 15;
=> Seja y = cell e sejam as celula x, y parte de uma two bridge. Se x foi preenchida pelo adversário, y tem aumento de prioridade a fim de salvar a conexão entre as peças da IA.

int _cell_can_generate_two_bridges = 8;

=> Esta celula se ocupada, pode formar uma ou mais two_bridges, facilitando a conexão entre diferentes sets de celula, portanto ela também tem maior prioridade.

int _cell_creates_wall = 6;

=> uma celula que cria uma wall é aquela que cria conexão no sentido perpendicular a sua borda, sendo mais util para atingir o objetivo do a celula que cria rope, ela recebe prioridade maior.

int _cell_creates_rope = 3;

=> uma celula que cria uma rope é aquela que cria conexão no sentido paralelo a sua borda, ela recebe prioridade maior que outras que não criam conexão alguma.

Diferentes combinações dos valores das variáveis globais descritas acima foram testados a fim de se conseguir a vitória com o menor número de jogadas contra um jogador que só faz jogadas aleaorias. Mesmo que este adversário tenha sido extremamente fácil, ele ajuda a testar combinações tais que evitem jogadas desnecessárias da IA. Isto é, se desempedida, que a IA ganhe o mais rápido possível.

3. Considerações Finais:

O algoritmo difere dos mais famosos para o jogo hex por não usar uma *parse tree* em que se analisa as possíveis respostas a cada jogada, sua simplicidade garante melhor performance e menor consumo de memória.

Obviamente, o grau de sucesso há de ser menor do que as implementações mais complexas. Mas a adição de novas funções de reconhecimento de padrões locais que alterem a prioridade de uma célula podem ajudar o algoritmo a tomar decisões ainda melhores em cada jogada sem torná-lo extremamente complexo.