Projeto final das disciplinas

Estrutura de Dados Básicas **I** - 2015.1
Linguagem de Programação **I** - 2015.1

Maze RL/ Labirinto

Leno Pereira Ferreira
Raimundo Vítor de Godeiro Marques

Este projeto tem por objetivo apresentar o problema de geração de labirintos, gerar a sua solução (encontrar o caminho de entrada até a saída do labirinto). Para gerar o labirinto usamos a técnica de ***conjuntos disjuntos (disjoint sets)****, a solução do labirinto foi encontrada usando da técnica de* ***backtracking.***
Através desse projeto encontramos uma maneira de poder por em prática e aperfeiçoar os conhecimentos adquiridos nas disciplinas.

Sumário

15 de julho de 2015

1.Introdução

A motivação para a construção do **Maze RL** foi devido aos alunos gostarem de jogos de raciocínios lógicos e o **labirinto** é um jogo bastante usado com crianças, para desenvolver habilidades como coordenação motora, senso de lógica, lateralidade.

No jogo de labirinto encontramos basicamente dois problemas chaves, a geração do labirinto e como encontrar o caminho de saída do mesmo. Alem disso outro problema interessante foi a criação de uma inteligência artificial de busca de um caminho.

O labirinto é basicamente constituído de celulas, dispostas na forma de uma matriz. Cada célula possui quatro paredes. Ao criarmos um labirinto, inicialmente todas as paredes estão erguidas, as quais deverão ser “derrubadas” para gerar os caminhos do labirinto.

O objetivo na geração do labirinto é construir no minimo, um caminho da célula de entrada até a celula de saída formando um labirinto valido e criar um labirinto totalmente conexo, ou seja existe um caminho entre quaisquer duas células do labirinto.Para isso empregamos a ideia de **conjuntos disjuntos**.

A solução do labirinto é o caminho que devera ser percorrido para ir da célula atual até a célula final, o caminho de saída do labirinto pode ser encontrado a partir de algumas ideias de algoritmos como por exemplo: **força bruta, A*, backtracking**.

Escolhemos usar a técnica **backtracking** devido a mesma ser a mais eficiente para encontrar a saída do labirinto nessa situação, pois quando ela encontrar a saída final, o algoritmo será finalizado, enquanto o da **força bruta** ira percorrer todo o labirinto. Como só possuímos um único caminho de saída, não é necessário percorrer todo o labirinto. A técnica **A*** [1] é vantajosa quando há mais de uma possibilidade de caminho, na qual apresentará o caminho mais eficiente; se a usássemos, estaríamos perdendo em desempenho já que gastaríamos memória desnecessariamente para armazenar os valores dos melhores caminhos. Assim a técnica que mais se enquadra nas nossas necessidades foi a de backtracking.

Para criar novos desafios de jogo, construímos um modo em que o jogador irá jogar contra o computador. Para isso, desenvolvemos um algorítimo de inteligência artificial para o computador simular um jogador.

Na próxima seção, apresentaremos as principais ideias usadas para a solução dos problemas . Essas ideias tem todo o embasamento teórico que foram usadas para resolver os principais problemas desse projeto. Em seguida, é apresentado como o código foi organizado.

2. Técnicas

2.1 Conjuntos disjuntos

A ideia consiste basicamente em construirmos uma matriz na qual cada celula inicialmente é um conjunto de um elemento só; cada célula é um conjunto disjunto pois, inicialmente cada conjunto possui elementos diferentes, assim todos são disjuntos; a ideia é no final formamos um único conjunto contendo todas as células.

Para gerar o labirinto usamos conjuntos disjuntos, o livro do Weiss [2] sugere um algoritmo para geração de labirintos:

1. Comece com paredes em todas células (exceto na entrada e saída)
2. Continuamente escolha uma parede e uma célula randomicamente
3. Derrube a parede escolhida se as células compartilhadas por esta parede não estão conectadas entre si
4. Repita os passos 2 e 3 até que a entrada esteja conectada à saída.

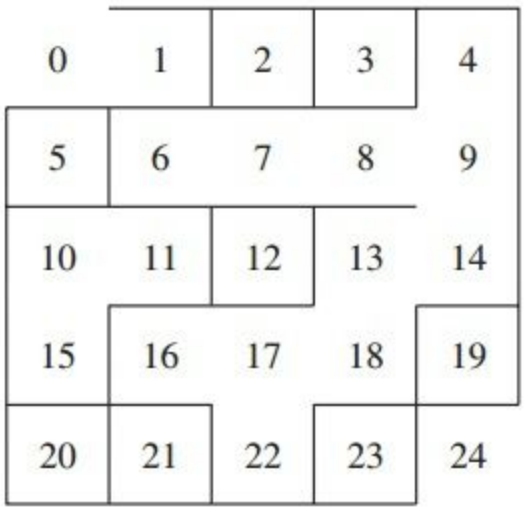
Por exemplo , a construção de um labirinto 5x5 :

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Inicialmente todas as paredes estão de erguidas(exceto a da entrada e saída). Assim teremos 25 conjuntos sendo todos disjuntos.

{0} {1} {2} {3} {4} {5} {6} {7} {8} {9} {10} {11} {12} {13} {14} {15} {16} {17} {18} {19} {20} {21} {22} {23} {24}

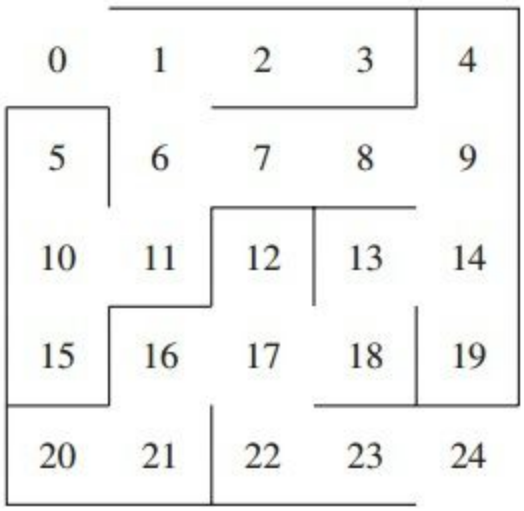
Em um estágio mais avançado teremos:



Neste casso a configuração de conjuntos disjuntos seria:

{0, 1} {2} {3} {4, 6, 7, 8, 9, 13, 14, 16, 17, 18, 22} {5} {10, 11, 15} {12} {19} {20} {21} {23} {24}

No final do algoritmo quando teremos um labirinto totalmente conexo, teremos um único conjunto contendo todas as células



{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24}

2.2 Backtracking

A solução do labirinto consiste em encontrar o caminho de onde o jogador se encontra até o final do labirinto. Para gerarmos esse caminho usamos da técnica de backtraking.

Caracteristícas do backtraking:

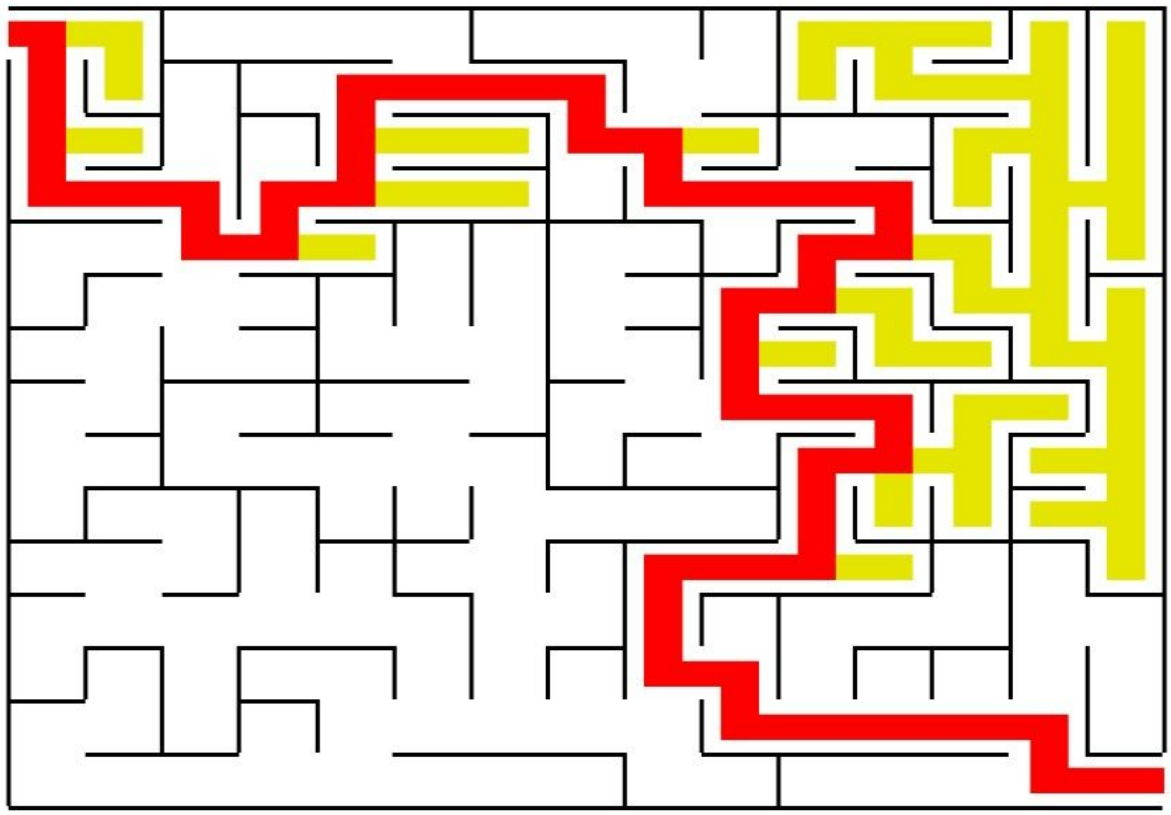
1. Algoritmo de tentativa e erro, não seguindo uma regra fixa
2. Os passos para encontrar a solução são tentados e armazenados
3. Caso os passos tomados não cheguem à solução final eles podem ser retirados dos registros.

A ideia é basicamente ir percorrendo o labirinto, armazenando o caminho em uma pilha e as células já visitadas vão sendo acrecidas em uma lista. Quando chegar no final do labirinto o algoritmo é encerrado.

Algoritmo 1: Backtracking (M, CelFinal,CelAtual, pilha, lista)

Entrada: Recebe a matriz M , CelFinal que é a celula final do labirinto, CelAtual que é a célula em que se encontra atualmente, recebe uma lista com todas as células visitadas e pilha que e a pilha contendo o caminho de saída do labirinto;
Saída: Retorna vazio
→ verifica se a célula em que se encontra é a célula de saída
→ retorna vazio e encerra o algoritmo;
→ se não;
→ verifica se não tem parede na direita e se a celula não foi visitada;
→muda a posição para a nova celula;
→adiciona a nova celula na pilha de caminhos;
→adiciona a nova celula na lista de celulas visitadas;
→ se não;
→ verifica se não tem parede em baixo e se a celula não foi visitada;
→muda a posição para a nova celula;
→adiciona a nova celula na pilha de caminhos;
→adiciona a nova celula na lista de celulas visitadas;
→ se não;
→ verifica se não tem parede na esquerda e se a celula não foi visitada;
→muda a posição para a nova celula;
→adiciona a nova celula na pilha de caminhos;
→adiciona a nova celula na lista de celulas visitadas;
→ se não;
→ verifica se não tem parede em cima e se a celula não foi visitada;
→muda a posição para a nova celula;
→adiciona a nova celula na pilha de caminhos;
→adiciona a nova celula na lista de celulas visitadas;
→ se não;
→ retira um elemento da pilha de caminhos;
→ chama novamente a função;

A seguir um exemplo da técnica de backtracking em um labirinto 15x15. Em vermelho temos a solução do labirinto (“pilha que contem o caminho de solução do labirinto”) e em amarelo as tentativas para encontrar a saída do labirinto (“lista que contem todos os caminhos percorridos”). Podemos observar que a técnica de backtracking é melhor na prática que a da força bruta .



2.3 Inteligência Artificial

Foi preciso criar uma inteligência artificial que satisfizesse os diferentes níveis de dificuldade do jogo. A IA é responsável por guiar o player adversário de seu ponto de partida até sua saída. Para a modelagem, os seguintes itens foram levados em consideração:

1. Tempo para tomar uma decisão: o tempo de espera para ir para a próxima celula.
2. Capacidade de tomar a decisão certa: a probabilidade de a próxima célula ser aquela que vai levar a saída.
3. Memória: quantidade máxima de células já visitadas que se pode lembrar.

O 'poder de raciocínio' da IA foi definido através de um número, chamado de coeficiente de inteligência, que pode variar de 1 a 200. Quanto maior esse número, mais eficiente e rápida é a busca pela saída em um labirinto. Vários testes e ajustes foram realizados para determinar os coeficientes de inteligência adequados para cada nível de dificuldade do jogo.

A fim de proporcionar certo grau de comportamento humano à IA, tanto o tempo de tomada de decisão quanto a probabilidade de tomar a decisão certa variam dinamicamente em função do coeficiente de inteligência e do tamanho do caminho restante até a saída. Dessa forma, quanto mais a IA se aproxima da saída, maior a chance de se escolher a celula certa e menor é o tempo para se fazer essa escolha.

3.Organização do código

A tabela a seguir descreve resumidamente os arquivos de código-fonte mais importantes do projeto:

Arquivo	**Descrição**
labirinto.h labirinto.cpp	Define a classe Labirinto, que gerencia a construção do labirinto
player.h player.cpp	Define a classe Player, que representa um jogador
game.h game.cpp	Define a classe Game, que fornece a interface gráfica e gerencia o andamento do jogo
mainwindow.h mainwindow.cpp	Define a funcionalidades relacionadas à interface do usuário, como menus de opções e tabela de recordes
pathfinderai.h pathifinderai.cpp	Define a classe PathFinderAI, que gerencia a inteligência artificial
digitalclock.h digitalclock.cpp	Define a classe DigitalClock, que gerencia o cronometro

4. Ferramentas e bibliotecas usadas

Esse projeto foi desenvolvido em **C++**[3]**,** usando como IDE o QT Creator [4] e suas bibliotecas. Outras ferramentas usadas foram as seguintes:

- ➪ API**:** OpenGL [5]
- ➪ Controle de versão: GitHub
- ➪ Documentação: Doxygen

5.Conclusão

O projeto **Maze RL** foi de extrema importância para o amadurecimento do nosso conhecimento em programação e melhorar o nosso trabalho em equipe. Utilizamos poucas bibliotecas de alto nível, isso nos proporcionou uma grande experiência e um maior aprofundamento em programação **C++.**

Nesse trabalho realizamos a construção do labirinto usando a idéia de **conjuntos disjuntos**, encontramos o caminho do labirinto através do algoritmo de **backtracking**, implementamos três níveis de dificuldade no jogo (fácil, normal e difícil); além de implementarmos quatro modos de jogo (carreira, multiplayer, competir, treinar).

Se estivéssemos começando o labirinto hoje, faríamos o modo de jogo multiplayer de uma forma diferente. Criaríamos o modo multiplayer via rede, sendo possível jogar de computadores diferentes, fazendo com que os desafiantes pudessem jogar sem um ver os passos do outro e assim gerar uma maior competitividade para saber quem ira acabar primeiro.

Nesse projeto gostaríamos de implementar o labirinto em modo 3D, devido a complexidade e o curto espaço de tempo disponível para realizar o projeto, essa tarefa se tornou impossível.

6.Referência

[1] A * Pathfinding para Iniciantes ,Por Patrick Lester (Atualizado em 21 de abril de 2004) (Traduzido por Reynaldo N. Gayoso, rngayoso@msn.com em 04 de Abril de 2005) (Atualizado em 04 de junho de 2005)

http://www.policyalmanac.org/games/aStarTutorial_port.htm

[2] M. A. Weiss. Data Structures and Algorithm Analysis in C++, Third Edition, Cap. 8.

[3] http://en.cppreference.com/w/

[4] https://www.qt.io/

[5] http://nehe.gamedev.net/