Preprint typeset using LaTeX style emulateapj v. 01/23/15

JOGO DA VELHA ESTENDIDO - COMPUTAÇÃO I

Mateus V. C. A. Santos (115062262)
Instituto de física, Universidade Federal do Rio de Janeiro, Rio de janeiro

Resumo

Com base no conhecido Jogo da Velha de três dimensões, construímos na linguagem C uma extensão do clássico em que é possível jogar em quatro ou até nove dimensões. Outra novidade é a possibilidade do jogador vencer o jogo com subdiagonais que tangenciam as bordas do tabuleiro. Apresentamos aqui a descrição do código bem como dificuldades encontradas ao longo do desenvolvimento.

1. INTRODUÇÃO

O Jogo da Velha é um passatempo popular com regras simples. Para vencer o jogo, é necessário que o jogador consiga assinalar seu símbolo em todas as células de uma linha, coluna, diagonal ou subdiagonal. A figura 1 mostra um exemplo de uma partida que terminou com um vencedor (o jogador X, que conseguiu preencher a diagonal principal). Caso nenhum jogador consiga satisfazer alguma condição acima, o jogo termina empatado. O tabuleiro do jogo é uma matriz 3x3 e dois símbolos distintos são usados pelos jogadores.

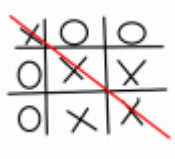

Figure 1. Preenchimento total da diagonal principal.

Neste trabalho, implementamos na linguagem C a versão estendida do jogo em que o número de dimensões do tabuleiro varia de quatro à nove podendo ter um até três jogadores humanos. Além das formas tradicionais, é possível vencer o jogo com subdiagonais que tangenciam as bordas do tabuleiro como ilustrado na figura 2.

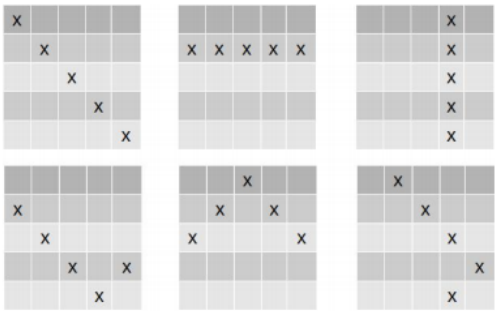

Figure 2. Exemplos de preenchimentos que vencem o jogo.

2. DESAFIOS E SOLUÇÕES PROPOSTAS

Para que o jogo funcione corretamente é preciso que haja alguns aspectos importantes, tais como, uma interface gráfica que sirva de comunicação com o jogador, um filtro de entrada para que o jogo não se encerre após o usuário digitar algo errado e uma validação do estado do tabuleiro para saber se algum jogador venceu ou se o jogo empatou.

A primeira parte é simples e pode ser feita com funções de entrada e saída padrões da biblioteca stdio.h. Para criar um filtro decente é necessário imaginar todas as possibilidades de entrada que o jogador pode inserir e permitir somente aquelas desejadas, é interessante colocar a função *scanf* dentro de um ciclo infinito como o *while* e encerrar o ciclo somente quando o usuário entrar com a informação correta, é importante usar a função *getchar* após o *scanf* para garantir que o caractere *enter* não seja capturado.

O principal desafio se encontra na validação do tabuleiro, verificar se um jogador venceu preenchendo uma linha ou uma coluna ou a diagonal principal/secundária é relativamente simples, basta percorrer a matriz e a cada linha/coluna/diagonal verificar se todos os símbolos são iguais e não nulos, mas verificar se este preencheu uma subdiagonal que pode tangenciar alguma borda requer alguns detalhes a mais. É possível se deslocar através de alguma subdiagonal de forma similar à diagonal principal (índices $[i][i]$), uma solução para um conjunto de subdiagonais, por exemplo, é percorrer a matriz através dos índices $[i][i+j]$ onde j incrementa uma unidade a cada subdiagonal.

Para simular a tangência com uma borda é possível criar uma variável auxiliar z que incrementa de duas unidades apenas quando $i+j >= N-1$ onde N é a dimensão do tabuleiro, desta forma percorrer a matriz com os índices $[i][i+j-z]$ simulará o 'rebate' em uma das bordas. Vale ressaltar que há vários conjuntos de subdiagonais possíveis como ilustrado na figura 3.

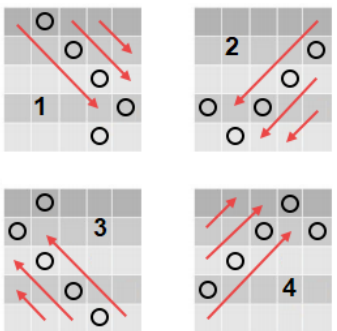

Figure 3. Conjuntos de subdiagonais.

Definimos diferentes conjuntos de subdiagonais, 1, 2, 3 e 4 como mostrado acima, o conjunto de subdiagonais 1 são todas as subdiagonais que começam na primeira linha e seguem o sentido do primeiro tabuleiro, as do conjunto 2 começam na última coluna e seguem o sentido do segundo tabuleiro e assim por diante. A ideia da variável auxiliar z para tangenciamento se aplica à todos os conjuntos de subdiagonais mudando apenas o sinal de z dependendo do conjunto.

3. FUNCIONALIDADES DO JOGO IMPLEMENTADO

A interface do jogo é bem trabalhada a fim de oferecer uma satisfação visual ao jogador. Textos bem colocados e espaçados, menus de opções bem legíveis e cores em textos importantes, o jogo conta ainda com um tabuleiro bem feito com espaçamento das células adequado, legenda das linhas e colunas e coloração da sequência vitoriosa para fácil identificação.

O jogo possui diferentes cenas (menu, em jogo, gameover, etc). Em cada cena é possível transitar para outras cenas, na cena inicial, por exemplo, pode-se escolher começar o jogo, aprender como jogar, mudar as configurações do jogo ou sair. Nas configurações da tela de opções é possível definir diversas características do jogo como número de jogadores, definir jogador 2 como máquina ou humano, alterar as dimensões do tabuleiro e ainda definir a dificuldade com que a máquina irá jogar o jogo.

A pseudo inteligência artificial da máquina é extremamente simples, há três opções de dificuldade: fácil, normal e difícil. No fácil a máquina preenche as células aleatoriamente, no normal existe uma probabilidade de 75% da máquina quando for preencher uma célula jogar no difícil e 25% de jogar no fácil, no modo difícil a máquina impede o jogador de ganhar preenchendo linhas/colunas/diagonais diretamente, é necessário preparar armadilhas para poder vencer a máquina.

4. ESTRUTURA DO CÓDIGO SOLUÇÃO

O código do jogo foi estruturado de forma modular, separamos todas as funções e estruturas por arquivos que representam as áreas descritas abaixo.

- Interface

  Funções que imprimem na tela criando a interface do jogo e a comunicação com o jogador.

- Ciclo de jogo

  Ciclos infinitos para cada estado de jogo (menu, em jogo, gameover, etc). Quando alguma condição é satisfeita um ciclo (exemplo: menu) é encerrado e um novo ciclo iniciado (exemplo: em jogo).

- Validação

  Funções que verificam se alguma condição foi satisfeita, por exemplo, se a entrada é válida, se o jogo terminou, etc.

- Utilidade

  Funções e estruturas de uso recorrente.

- Máquina

  Funções da máquina, por exemplo, funções que representam diferentes formas da máquina jogar contra o jogador humano.

- Constantes

  Diversas constantes globais que servem de fácil identificação e para evitar que o código fique 'hard-coded'.

Existem dois tipos de dados personalizados fundamentais para o controle do jogo, a *State* e a *Config*, a primeira é responsável por guardar todos os estados de jogo (tela de menu, em jogo, tela de gameover, etc), apenas um estado de jogo da *State* fica ativo durante o jogo e sempre que o jogador escolhe mudar de cena o estado de jogo se altera para o estado equivalente, a figura 4 ajuda a compreender como os estados de jogo ajudam a organizar as cenas de jogo. A segunda agrupa todas as informações referentes a partida em si, como quantidade de jogadores, número de dimensões do tabuleiro, marcações no tabuleiro, etc.

Figure 4. Funções de estado de jogo são chamadas dependendo do estado ativo do jogo.

5. VALIDAÇÃO E TESTES REALIZADOS

Para processar os dados de entrada do jogador optou-se por fazer como mencionado na seção 2, o caractere esperado na função *scanf* é do tipo char, isto por que este tipo acolhe inteiros e símbolos convencionais da tabela ASCII, caso o caractere esperado fosse do tipo inteiro e o usuário digitasse uma letra poderia causar erros de execução do programa. Durantes as possíveis entradas de dados no jogo como a escolha de uma opção no menu ou a entrada da linha e da coluna da marcação no tabuleiro nenhum erro ou *bug* foi encontrado, inúmeros símbolos da tabela ASCII foram digitados e o programa respondeu conforme o esperado enviando uma mensagem de erro ao jogador.

Na validação do tabuleiro o algorítimo implementado segue a lógica descrita da seção 2, inicialmente para saber se o jogo terminou as linhas/colunas/diagonais e subdiagonais eram verificadas simultaneamente por estarem dentro do mesmo ciclo infinito *for* como descrito no arquivo *src/valid.old.c*, entretanto percebemos que este método necessitava de mais variáveis para guardar

as pontuações e os símbolos de cada direção, também era feito mais cálculos computacionais já que a função só encerrava após todas as possibilidades $(6(N-1))$ serem calculadas.

Atualmente cada conjunto de direção (linhas ou colunas ou subdiagonais) é calculado separadamente, a vantagem deste método é que se houver uma linha vencedora a função já será encerrada e o programa não perderá tempo computando as colunas e cada conjunto de subdiagonais, há também a vantagem de criar apenas três variáveis auxiliares para todo o processo. Foram testadas todas as $6(N-1)$ possibilidades no tabuleiro 4x4 e 5x5 e o programa respondeu conforme o esperado, nenhum erro ou *bug* foi encontrado.

É durante este processo de validação do tabuleiro que a sequência de índices da última direção é armazenada em um vetor, se há um vencedor este vetor é utilizado para colorir as letras da jogada vencedora como ilustrado na figura 5.

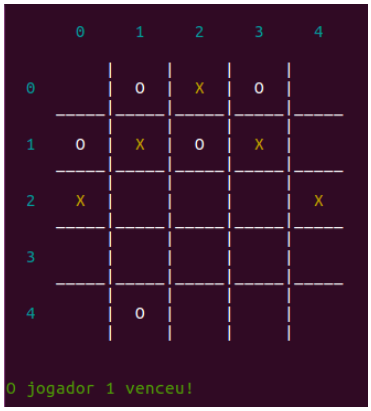

Figure 5. Sequência vitoriosa é colorida em amarelo.

6. DIFICULDADES E CONCLUSÃO

A principal dificuldade de criar o jogo foi a própria linguagem de programação escolhida. O fato de C não ser orientado à objetos fez com que o código ficasse maior e mais desorganizado do que seria se tivesse sido escrito em uma linguagem com suporte à orientação à objetos. C é uma linguagem muito mais rígida com certos detalhes, há muito trabalho para se lidar com coisas simples como, por exemplo, strings. A sensação que temos é de que o jogo poderia ter sido desenvolvido muito mais rapidamente se tivesse sido escrito em outras linguagens como Python, Java ou Javascript. C apesar de dificultar certas tarefas permite ter um controle total da memória do sistema bem como evitar computações desnecessárias ao lidar de forma básica com certos elementos (ex: strings), perfeitamente indicado para dispositivos com capacidades limitadas e aplicações que necessitam de alta performance como simulações e jogos mais complexos.

Com este trabalho percebemos que até para criar programas relativamente simples como um Jogo da Velha requer organização, estruturação e resolver problemas frequentemente, requisitos que são necessários para se criar aplicações maiores e que possam ser divididas em tarefas individuais por cada membro da equipe desenvolvedora.

REFERENCES

Introdução à programação: uma nova abordagem usando C, Flávio Varejão, Campus/Elsevier, 2015.
A Linguagem C, Brian W. Kernighan e Dennis M. Ritchie, Editora Campus
O jogo da velha, Wikipedia Reference