UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
ENGENHARIA DE COMPUTAÇÃO

LABORATÓRIO DE INTRODUÇÃO À CIÊNCIA DA COMPUTAÇÃO I - SSC 0601

Prof. Jó Ueyama, PhD

TETRIS

Rafael Carvalho 14604271

SÃO CARLOS

2023

SUMÁRIO

1. Introdução

Como avaliação na disciplina laboratório de introdução à ciência da computação I, foi requisitado aos alunos o desenvolvimento de um certo projeto arbitrário, exclusivamente na linguagem C. Sobretudo o objetivo era claro, a exploração da linguagem de programação com intuito de autodesenvolvimento e aplicação dos conceitos aprendidos em sala de aula.

Após certa consideração, foi decidido ao meu grupo, que o presente projeto consistiria na produção de uma adaptação do jogo Tetris, de Alexey Pajitnov, lançado pela primeira vez no ano de 1984.

O Tetris é um passatempo com regras simples e intuitivas. Resume-se em organizar peças aleatórias, chamadas de tetraminó, as quais caem sucessivamente, uma a uma, numa grade de 20 blocos de altura por 10 blocos de largura. Uma vez que completa-se uma linha com peças, essa linha é deletada e as peças superiores caem sob ação da gravidade, e assim sucessivamente, até formar-se uma coluna com peças além do limite do mapa, caracterizando a derrota do jogador.

2. Visão geral

Apesar de parecer trivial, a complexidade do código escalou significativamente conforme o nível de detalhe que precisou ser implementado no jogo. Para atender às necessidades do Tetris, foram criadas *structs* que representam as peças e suas rotações. Tais structs, ao longo do jogo, são escolhidas aleatoriamente, e o jogador pode movimentá-las conforme achar necessário, e caso complete uma linha, soma-se um nível de dificuldade e as peças caem mais rápido.

2.1. Ambiente

O desenvolvimento do projeto foi feito inteiramente no sistema operacional "Ubuntu 22.04.2 LTS (Jammy Jellyfish)", o qual dispõe do compilador "gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0". Como editor de texto, optou-se pelo "NVIM v.0.9.0".

2.2. Bibliotecas

A princípio seria possível que todo o código fosse feito usando apenas bibliotecas padrão, entretanto, o projeto foi fundamentado na biblioteca externa *ncurses.h*, com a qual, estruturou-se o código de maneira mais prática e compreensível.

2.3. Divisões do código

Uma vez que o projeto demonstrou-se extenso, optou-se por fazer a modularização do código em vários arquivos, agrupando-os em *headers*, da seguinte forma:

- ***game.c*:**

  Responsável pelo controle da lógica de jogo e das estruturas de impressão gráfica.

- ***boot.c*:**

  Responsável pela inicialização de funções necessárias em todo o programa, contendo principalmente as funções de inicialização dos atributos da biblioteca *ncurses.h*.

- ***hiscore.c*:**

  Responsável pelo controle de impressão, salvamento e ordenação decrescente de recorde.

- **tetris.c:**

  Abrange todas as seções anteriores, agrupando as funções principais na *main( )*.

- ***makefile*:**

Organização de todas as seções do código em headers e as bibliotecas externas num comando, simplificando o processo de compilação.

2.4. Gráficos

A representação gráfica do jogo foi feita com base nas funções da biblioteca ncurses.h, sendo as mais essenciais:

- ***mvprintw(int y, int x, char *texto, ….)*:**

Serve como alternativa da função *printf( )* da biblioteca *stdio.h*. A diferença é que inclui os parâmetros de coordenadas de localização no terminal, e imprime no buffer da tela.

- ***int refresh( )*:**

Serve para imprimir o *buffer* de tela, no caso, o *buffer* "*stdscr*", no console.

- ***int clear( )*:**

Responsável por limpar a tela.

- ***int getmaxx( )* e *int getmaxy( )*:**

Funções responsáveis por retornar as dimensões máximas do console. Foi aplicada na centralização das impressões usando coordenadas relativas.

Com essas funções, estruturou-se o sistema gráfico do programa. Por motivos de otimização, a função *refresh()* só foi usada quando realmente era necessária. E com as funções de obtenção de tamanho da tela, organizou-se a impressão conforme as coordenadas relativas, de modo que, independente do tamanho do console, o jogo sempre estará centralizado.

2.5. Algoritmos

Foi necessário a implementação de vários algoritmos no desenvolvimento do programa. Por exemplo, as sub-rotinas de impressão geralmente tinham o seguinte algoritmo:

1 - Impressão no buffer;
2 - Descarga de buffer de tela no console;
3 - Pausa momentânea.

Nos mecanismos de lógica de jogo, um algoritmo interessante é o utilizado para limpar a linha de bloco, o qual usa recursão:

1 - Verifica se a linha está completa, se não estiver, continua o fluxo do programa;

2 - Caso esteja, apaga a linha e desce as peças superiores;

3 - Retorna ao passo inicial, verificando se após a limpeza, novas linhas se fecharam.

Vale também a menção do algoritmo de organização de recordes, que escalona os resultados das partidas em um arquivo em ordem decrescente de pontuação. Para tanto, foi implementado o algoritmo "*bubble sort*", que, apesar da complexidade de tempo ser $O(n^2)$, funciona bem para poucas amostras.

2.6. Arquivos

O armazenamento de recordes foi feito com o uso de arquivos, primeiramente salvando as pontuações sem ordenação, mas, em seguida, o arquivo é organizado com um "*bubble sort*", para que sejam impressas apenas uma quantidade limitada de recordes decrescentemente na seção do menu.

2.7. A biblioteca *ncurses.h*

É válido ressaltar alguns detalhes da biblioteca ncurses.h, e como ela funciona.

Os processos de inicialização usados foram os seguintes:

- ***int initscr( )*:**

É a função inicial de quase todo arquivo que usa a biblioteca *ncurses.h*, atua inicializando um *buffer*, que para o programa serve como tela.

- ***int cbreak( )*:**

Inicializa o processo de leitura de teclado no modo "*cbreak"*.

- ***int nodelay(WINDOW, bool)*:**

Faz com que a função de leitura *getch( )* não trave a execução do programa.

- ***int noecho( )*:**

Desabilita a impressão de teclas digitadas.

- ***void boot_sequence( )*:**

É uma sub-rotina própria que agrega toda inicialização dos processos da ncurses e mais alguns outros, como temporizadores.

Toda a parte gráfica foi trabalhada com funções externas da padrão, mas, manipulando caracteres ASCII. Para tanto, o processo do uso de cores customizáveis na ncurses.h consiste no seguinte:

1 - inicializam-se as cores:

- ***int start_color( )*;**

2 - criam-se os pares de cores desejados:

- ***init_pair(int, COLOR_CUSTOM, COLOR_CUSTOM_1)*;**

3 - as cores são ligadas e desligadas conforme desejado:

- ***attron(COLOR_PAIR(int));***
- ***attroff(COLOR_PAIR(int));***

Além dessas, uma função muito relevante da *ncurses.h* é a *flushinp( ):*

- ***int flushinp( )*:**

Limpa a fila de comandos a serem executados no *getch( )*, de modo que sejam apenas utilizados os comandos pressionados pelo jogador no momento. Sem ela, o *delay* nos comandos torna o programa inutilizável.

3. Documentação

Pela complexidade do programa, as rotinas foram divididas em funções, de modo que o código pudesse ser facilmente reutilizado.

3.1. Controle de menu

A primeira seção desenvolvida no projeto foi o menu, a princípio, por sua baixa complexidade. É implementado diretamente na *main( )* através da função *menu( ),* localizada na seção *boot.c*.

O menu é dividido em: jogo, recordes, instruções, créditos e uma opção de saída do programa.

- ***int menu( )*:**

Retorna um valor correspondente à seleção. Além disso, também faz a impressão das partes principais do menu.

Menu inicial

3.2. Controle gráfico de interface

- ***void borders( )*:**

  Impressão de bordas ao redor do console, com intuito puramente estético.

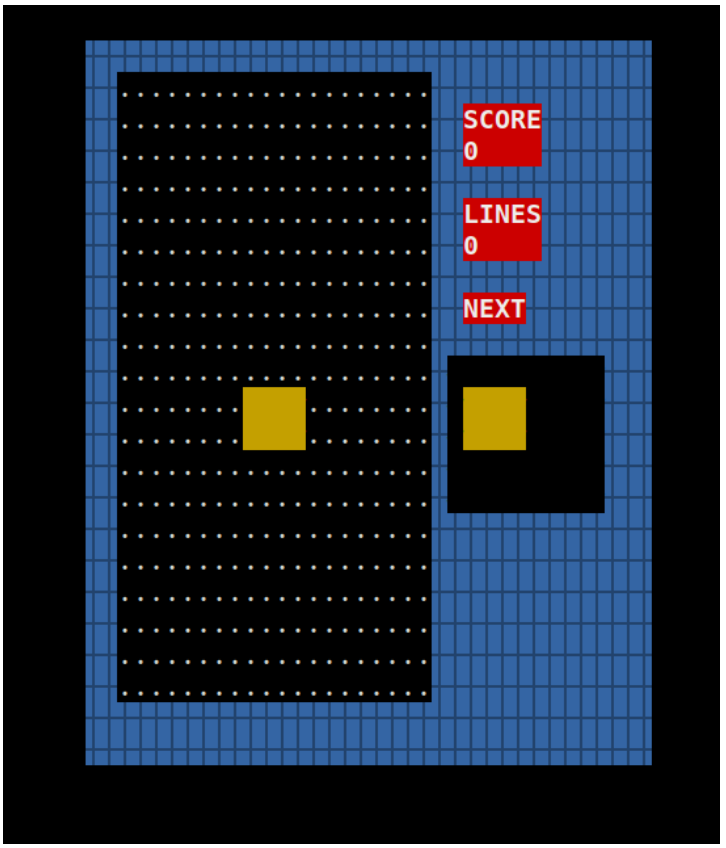
- ***void logoprint(int x):***

  Impressão do logo escrito tetris na tela inicial.

Logo que pisca em cores aleatórias

- ***void gameborders( ):***

  Impressão das bordas da tabela no jogo.



Interface do jogo

- ***void instrucoes( ), void creditos( ), void recordes( ):***

  Sub-rotinas de impressão de seleções do menu.

3.3. *Struct Tetromino* e matriz *gridvar[22][22]*

São duas variáveis amplamente utilizadas durante o jogo.

- ***Struct Tetromino:***

Consiste em uma *struct* que contém as características das peças do jogo, sendo essas, uma matriz de inteiros contendo cada célula não-vazia do tetraminó, e um inteiro correspondente ao "*id"* da peça.

- ***int gridvar[22][22]:***

É um vetor responsável por armazenar o estado de cada célula do jogo. Uma vez que um tetraminó atinge a colisão, as coordenadas das células em que ele parou são alteradas.

3.4. Controle gráfico de jogo

A parte gráfica se resume à descarga do buffer de tela no console entre as iterações do programa. Porém, para o jogo, é preciso que seja apagada a tela anterior e impressa uma nova, com o estado atualizado das peças.

Para isso, usam-se, principalmente, as seguintes funções:

- ***void printTetr(int y, int x, tetromino falling)*:**

Impressão da peça (tetraminó) na posição vertical "y" e horizontal "x".

- ***void delPrev(int y, int x, tetromino falling)*:**

Remoção da peça (tetraminó) na posição vertical "y" e horizontal "x". Funciona imprimindo uma peça normalmente, mas com as cores da grade, de modo que apague qualquer peça nessa posição.

- ***void grid(int gridvar[][22])*:**

Imprime a grade com as posições das peças que já cairam.

3.5. Leitura de dados durante o jogo

O trabalho de leitura de dados é feito pelas seguintes funções:

- ***int readInput():***

Retorna um valor correspondente a tecla selecionada pelo jogador. A validação da tecla é excludente, ou seja, qualquer tecla que não seja um comando é ignorada.

- ***int kbhit():***

Como na biblioteca *ncurses.h* não existe uma adaptação do *kbhit( ) da conio.h*, essa função faz esse papel, e retorna 1 caso uma tecla seja pressionada.

3.6. Lógica de jogo

É a parte mais complexa do projeto, e relaciona as *structs tetromino* com o ambiente do jogo e com as movimentações feitas pelo jogador. Engloba as seguintes funções:

- ***tetromino chooseTet(int choose):***

  Escolhe o tetraminó correspondente a variável *choose*, e retorna uma *struct* de tetraminó.

- ***tetromino rotate(int rotation, tetromino falling):***

  Rotaciona a peça que está caindo. Funciona retornando a *struct* do tetraminó correspondente a rotação feita.

- ***void saveGrid(int y, int x, tetromino falling, int gridvar[][22]):***

  Salva o estado da grade com todas as peças no local em que estão, funciona alterando a matriz de inteiros "*gridvar*" atribuída, de modo que, salva-se um inteiro correspondente ao "*id*" da peça que caiu, para posteriormente obter informações a respeito da cor da célula.

- ***int checkCollision(int y, int x, tetromino falling, int gridvar[][22]):***

  Checa a colisão inferior a todos os pixels da peça que está caindo, e retorna 1 se verdadeiro.

- ***int checkCollision_X(int y, int x, tetromino falling,int gridvar[][22], int move):***

  Checa a colisão horizontal da peça caindo com relação a um movimento feito, retorna 1 caso não haja colisão. Com esse valor é possível validar ou invalidar a ação.

- ***int validateRotation(int y, int x, int rotation, tetromino falling):***

  Verifica a colisão na rotação da peça. Caso haja colisão, retorna 1 e o movimento pode ser invalidado.

- ***int hardDrop(int y, int x, tetromino falling, int gridvar[][22]):***

  É um movimento clássico do Tetris que funciona para agilizar o jogo. Funciona movimentando a peça que está caindo diretamente para o primeiro lugar onde houver colisão. Retorna a posição válida para a ação.

- ***int clearLine(int gridvar[][22], int *contador):***

  Limpa uma linha caso ela esteja completa, e, por recursão, verifica se, após a queda dos blocos, outra linha completa será formada, e assim sucessivamente até retornar falso. O inteiro retornado por essa função corresponde a quantidade de linhas zeradas durante o processo de recursão.

- ***int checkEnd(int gridvar[][22]):***

  Verifica se a grade já atingiu o limite, e retorna verdadeiro, finalizando o jogo.

- ***int runGame():***

  É apenas uma função na qual todas as outras são agregadas.

3.7. Recordes

Para gerenciamento da pontuação, foram feitas as seguintes funções:

- ***"void hiscore(int score)":***

  Salva a pontuação e as iniciais do jogador num arquivo ".txt".

- ***"void sortHiscore( )":***

  Aplica o algoritmo *bubble-sort* para a organização da pontuação em um novo arquivo ".txt", alocando os vetores temporários dinamicamente, por meio de um *malloc( )*.

- ***"void printHiscore( )":***

  Imprime, centralizado na tela, uma quantidade arbitrária das melhores pontuações em ordem decrescente.

4. Considerações finais

O processo de desenvolvimento foi desafiador, inúmeras dificuldades quase me fizeram desistir do projeto, mas, com certo planejamento essas dificuldades foram, aos poucos, sendo eliminadas. Caso fosse para me avaliar, afirmo que uma nota maior ou igual a 9,8 é válida.

4.1. Aprendizado

Quanto à questão de conhecimento técnico, tenho certeza que desenvolvi bastante. Após algumas dezenas de horas desenvolvendo o programa, encontrei de tudo. Acredito que o que mais foi desenvolvido em mim nesse processo foi a estruturação de algoritmos, ou seja, a elaboração de uma sequência lógica de etapas bem definida sem sombra para dúvidas. Percebi que a negligência dessa etapa pode resultar em uma "bola de neve", desestruturando o sistema em alguma etapa, e potencialmente, invalidando todo o programa por inconsistências.

4.2. Dificuldades

As dificuldades foram inúmeras, começando pelo próprio relatório. A princípio considerei apenas algumas das dificuldades que encontrei, e as classifiquei como razoáveis, mas, com o desenrolar dos algoritmos percebi a escalada dos problemas. Vale notar as seguintes dificuldades:

- **Rotação de peças:**

Inicialmente, com intuito de resolver o problema de uma maneira elegante, minha ideia era a de criar um algoritmo para a rotação das peças, entretanto, após um certo tempo sem sucesso, decidi pesquisar uma solução pronta, porém, as recomendações que encontrei não usavam um algoritmo, e sim, criavam as peças em todas as rotações, de modo que fossem impressas diretamente.

Feito isso, encontrei outro problema, a rotação das peças deveria ser barrada por outro bloco. Adaptei esse problema com uma função que verifica se a peça rotacionada teria colisão, funciona bem, mas deve haver uma solução mais elegante.

- **Limpeza de linhas completas:**

Rapidamente fui capaz de solucionar o problema para uma linha, mas ao tentar implementar uma solução que fosse capaz de limpar as outras linhas tive dificuldades. Intuitivamente achei que teria como adaptar a recursão nessa solução, mas não sabia como, até que, numa tentativa sem muita esperança, inclui a função de limpar a linha dentro dela mesma e o problema foi solucionado.

- **Coordenadas relativas:**

Esse foi, sem sombra de dúvidas, o maior dos problemas. Como comecei o projeto imprimindo no console as informações centralizadas, após um avanço considerável no código, as referências às coordenadas estavam extremamente confusas, o que me fez reescrever toda a seção do código relativa ao jogo, mas, usando o conceito de coordenadas relativas.

- **Controle de tempo:**

Para o controle da velocidade das peças, foi necessário um tratamento da contagem do tempo. As dificuldades foram reveladas quando houve o conflito entre a função *napms( )* e a função *clock( )*. Para tanto, tive de procurar outra alternativa para a função *clock( )*, no caso, a solução foi a *clock_gettime( )*, entretanto, a contagem do tempo tinha de ser feita na precisão dos milissegundos, pelo menos. Como a *clock_gettime( )*, registra ou segundos ou nanosegundos, tive de optar pela segunda opção, mas com um detalhe, a contagem rapidamente estourava o valor de um inteiro. Entretanto, com certo trabalho algorítmico de controle de intervalo, foi possível a resolução do problema.

- ***Buffer* de teclado:**

Uma outra dificuldade que encontrei foi o tratamento com o *buffer* de teclado. Caso o jogador segurasse uma tecla, esses comandos seriam registrados e estariam em fila para serem executados no programa, mas, por sorte, encontrei a solução rapidamente, executar a função *flushinp( )* antes da execução do *getch( )*.

Referências

Tetris Wiki - Fandom. c2023. Página inicial. Disponível em <https://tetris.fandom.com/wiki/Tetris_Wiki> Acesso em: 05 de jun de 2023.

Tetris Wiki. c2023. Página inicial. Disponível em <https://tetris.wiki/Tetris.wiki> Acesso em: 05 de jun de 2023.