Departamento de Ciência da Computação – Universidade de Brasília
CIC 113956 Programação Sistemática – 1/2016
Data de entrega: 13/06/2016, 23:59h
Trabalho Prático - Parte 2

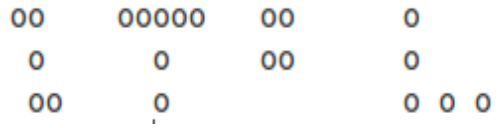
Introdução

Considerando as variadas versões de um Tetris, o TP1 da disciplina de Programação Sistemática em 2016/1 foi uma versão simplificada do mesmo. O objetivo da segunda parte do trabalho é aprimorar as funcionalidades do jogo implementado na primeira etapa. Para isso, é imprescindível que a nova versão satisfaça a especificação a seguir.

Regras e Funcionamento do Jogo

Todas as regras estabelecidas na primeira etapa devem ser respeitadas e incluídas nesta segunda. Regras como Tela Inicial, Tela de encerramento (Game Over), dimensão da matriz, cores das peças, exclusão de linhas preenchidas, limite superior demarcado, pontuação, bordas da tela feita com ncurses, boa exploração da ncurses, aleatoriedade das peças, etc. As regras abaixo devem ser adicionadas ou modificadas no jogo criado no TP1:

1. Peças: Estas serão as novas peças que atuarão juntamente com as anteriores, do trabalho 1:



2. Movimentação: Durante a execução do jogo, a peça deverá, obrigatoriamente, cair na direção vertical (simulando efeito de gravidade, de forma **automática**). O padrão inicial do tempo de queda, para cada peça nova que surgir na parte superior da matriz, deverá ser de 1 segundo por posição no eixo Y. Durante a queda, o jogador poderá:

1. Direcional para cima: Rotacionar a peça;
2. Direcional para baixo: Aumentar velocidade de queda (2x mais rápido a cada vez que o usuário pressionar o direcional para baixo);
3. Direcional para esquerda: Mover a peça para a esquerda;
4. Direcional para direita : Mover a peça para a direita.

3. Recordes e Sistema de Registro: Sempre que um jogador encerrar uma partida, sua pontuação deve ser comparada com as maiores pontuações já alcançadas por outros jogadores em partidas anteriores. Para isso, o programa deverá conter um sistema de armazenamento que salva as 5 maiores pontuações já atingidas, sem que o programa seja compilado outra vez. O programa deverá armazenar os dados em um arquivo.txt . Obs.: *As 5 primeiras partidas terão sua pontuação salva automaticamente.*

Se em uma nova partida a pontuação do jogador ultrapassar uma pontuação registrada no arquivo .txt, o arquivo deve ser atualizado. O jogador deve inserir seu nome/apelido antes ou

depois do término da partida. Finalmente, na tela de Fim de Jogo, o placar atualizado deve ser mostrado ao jogador.

Composição de cada pontuação:

[APELIDO DO JOGADOR] [PONTUAÇÃO] [DATA/TEMPO DE EXECUÇÃO]

Modularização:

Todos os módulos utilizados na etapa anterior serão reutilizados aqui e, assim como na primeira parte, para cada módulo de implementação criado (.c), deve ser criado também um módulo de definição (.h). Esta etapa conterá apenas um módulo exclusivo, os outros serão apenas alterados, se necessário. Não se esqueçam de utilizar diretivas de controle para evitar múltiplas inclusões!

Módulo 1: Tela

O módulo 1 será responsável por criar a matriz que servirá como tela principal e dinâmica do jogo, onde as peças se movimentarão.

A matriz deve ter o tamanho 15x25 ([y][x]).

As telas de início e fim de jogo podem ser criadas onde for mais usual.

Funções:

```
/* A função "cria_tela" deve criar a interface (bordas),
lugar onde ocorrerá toda a dinâmica do jogo. */
• Tela* cria_tela();

/* A função "mostra_tela" será chamada sempre que o jogo for
iniciado ou alguma mudança ocorrer dentro da matriz principal,
pois tal mudança só irá ser percebida pelo jogador quando ela for
mostrada na saída. */
• void mostra_tela(Tela* t);
```

Módulo 2: Peças (com alterações)

O módulo 2 será responsável por criar as peças do jogo e realizar seus movimentos dentro da tela(matriz). A lógica de funcionamento do trabalho anterior pode ser alterada, conforme for necessário.

As peças:

- As peças retas terão tamanho variável entre 3 e 5 caracteres.
- As demais peças deve seguir o exemplo dado na ***imagem 1***.

Funções:

```
/* A função “nova_peça” mostra uma nova peça na parte superior do
tetris. */
```
- `void nova_peça(Tela* tela);`

```
/* A função “move_peça_x” move a peça para a direita e esquerda (
eixo x ). O parâmetro inteiro x é a coordenada foco, para onde a
peça deverá se mover. */
```
- `void move_peça_x(peça* peça, int x);`

```
/* A função “move_peça_y” move a peça para baixo ( eixo y ). O
parâmetro inteiro y é a coordenada foco, para onde a peça deverá
se mover. */
```
- `void move_peça_y(peça* peça, int y);`

```
/* A função “rotaciona_peca” gira a peça.*/
```
- `void rotaciona_peça(peça* peça);`

```
/* A função “speed_up” aumenta a velocidade de queda da peça.*/
```
- `void speed_up (peça* peça, int y);`

Módulo 3: Engine

O módulo 3 será responsável por ler o input do teclado a cada jogada e associar com uma ação, para então, mostrar a movimentação na tela, além de inicializar e finalizar o uso da ncurses. Portanto, este será o módulo com maior acoplamento do trabalho, sendo cliente dos outros dois módulos anteriores.

Funções:

```
/* Todos os recursos da biblioteca ncurses deverão ser
declarados/usados entre as duas funções a seguir. */
```
- `void inicia_ncurses();`
- `void finaliza_ncurses();`

```
/* A função “pega_input”, do tipo inteiro, é a função que será
responsável por ler a entrada do teclado. Seu retorno é o valor da
leitura que poderá ser usado em outras funções, quando necessário.
*/
```
- `int pega_input(int input);`

Módulo 4: Placar

O módulo 4 será o módulo exclusivo do trabalho 2. Ele será responsável por criar o sistema de registro de pontuações.

Funções:

- `void cria_placar();`
- `void atualiza_placar( int pontuacao );`
- `void mostra_placar();`

Análise Estática via Ferramenta Splint:

Utilizem o **Splint** para avaliar as faltas de interface, dados, entrada/saída, controle e armazenamento, conforme a aula de análise estática e o artigo "C traps and pitfalls" disponível no Moodle da disciplina. Seu programa deve compilar sem Warnings com os comandos ***-weak +compdef +infloops +sysdirerros.***

Padrão de Documentação

1- Para tornar o código mais organizado e legível, o grupo deve adotar um *padrão de documentação*.

2- Elaborem um arquivo .pdf que contenha explicações e exemplos do padrão adotado pelo seu grupo.

3- Faça a documentação e comentários do código (funções, variáveis, TADs, módulos e suas interfaces) utilizando a ferramenta Doxygen e gere em formato .pdf ou .html.

Controle de qualidade das funcionalidades

Deve-se criar um módulo controlador de teste (disciplinado) usando o CUnit (C) para testar se as principais funcionalidades e restrições dos módulos atendem a especificação. O teste disciplinado deve seguir os seguintes passos:

1. Antes de testar: produzir um roteiro de teste;

2. Antes de iniciar o teste: estabelecer o cenário do teste;

3. Criar um módulo controlador de teste, usando a ferramenta CUnit (C) para testar as principais funcionalidades de cada módulo;

4. Ao testar: produzir um laudo em que todas as discrepâncias encontradas são registradas. Esse laudo pode ser uma saída da execução do CUnit (C) . Somente termine o teste antes de completar o roteiro, caso observe que não vale mais a pena continuar executando o roteiro, uma vez que o contexto para o resto está danificado;

5. Após a correção: repetir o teste a partir de 2 até o roteiro passar sem encontrar falhas.

Parte Escrita

Visando facilitar o trabalho em grupo, vocês devem utilizar o repositório **GitHub** para interagir e atualizar o código a cada etapa do desenvolvimento. Vide slides introdutórios sobre o GitHub no Aprender da disciplina de Programação Sistemática.

O arquivo .zip de submissão deve conter:

- Todos os arquivos necessários para a compilação e execução do programa, incluindo os módulos controladores de teste.
- Um ReadMe.txt contendo:
  - As instruções detalhadas para a compilação e execução correta do trabalho. Lembrem-se que o programa será avaliado em uma distribuição Linux.
  - O link do projeto no GitHub.
- Um documento textual explicando a arquitetura utilizada contendo os modelos conceituais e físicos
- Os gráficos gerados pelo **GitHub** com as estatísticas do desenvolvimento do projeto contendo as horas trabalhadas **por cada membro do grupo** e o relatório de descrição em formato 'pdf' ou 'txt' contendo as tarefas detalhadas que **cada membro realizou**.

Observações importantes

1. Organizem o trabalho em partes onde a equipe produza os artefatos solicitados incrementalmente e organizado em pequenas, mas constantes etapas. Deixar para fazer o trabalho na última hora comprometerá consideravelmente a qualidade final do trabalho entregue.
2. Não haverá prorrogação da data de entrega em hipótese alguma!
3. O método de correção aborda todas as funcionalidades pedidas na especificação. É importante que cumpram toda a especificação. Trabalhos que não sigam a especificação das funcionalidades serão desconsiderados, mesmo que estejam funcionando de forma correta.
4. Apenas códigos, modelos, descrições, desenvolvidos pelo grupo, são avaliados. Caso seja detectado que qualquer parte do trabalho tenha sido plagiado (seja de colegas de outros grupos ou de fontes encontradas na Internet) a nota do trabalho será **ZERO**.