

INF 01202 – ALGORITMOS E PROGRAMAÇÃO

Turmas A, B, C e D

Trabalho Prático Final – Semestre 2014/2

MOTIVAÇÃO E OBJETIVOS

No decorrer da disciplina de algoritmos e programação, foram apresentados diversos conceitos e técnicas de programação, visando expor os alunos às principais ferramentas lógicas e práticas para construção de algoritmos e solução de problemas diversos. Como passo fundamental, este trabalho vem a colocar a prova o aproveitamento dos alunos frente ao que foi exposto nas aulas teóricas e práticas, na forma de um jogo que deverá ser implementado na linguagem de programação C, em duplas de alunos.

O objetivo é implementar uma versão simplificada do jogo *Mr Do!* em modo texto (*ASCII art*). A implementação deve conter toda a interação do usuário (jogador) com o cenário (labirinto).

ESPECIFICAÇÃO

OBJETIVO DO JOGO

Mr Do deve colher todas as frutas do cenário ou eliminar todos os fantasmas nas duas fases do jogo. O jogo acaba quando:

- i) Não existem mais frutas para colher
- ii) Não existem mais fantasmas para eliminar
- iii) O Mr Do é eliminado por um dos fantasmas.

Os casos i) e ii) representam vitórias no jogo. O caso iii) representa derrotas.

LAYOUT DO GAME

Um exemplo do cenário do jogo pode ser visto na Figura 1. Além disso, pode-se entender melhor o jogo através de uma versão online. Esta versão pode ser acessada em: <http://www.letsplaysnes.com/play-mr-do-online/>. Entretanto, a nossa versão é mais simples que a versão online.

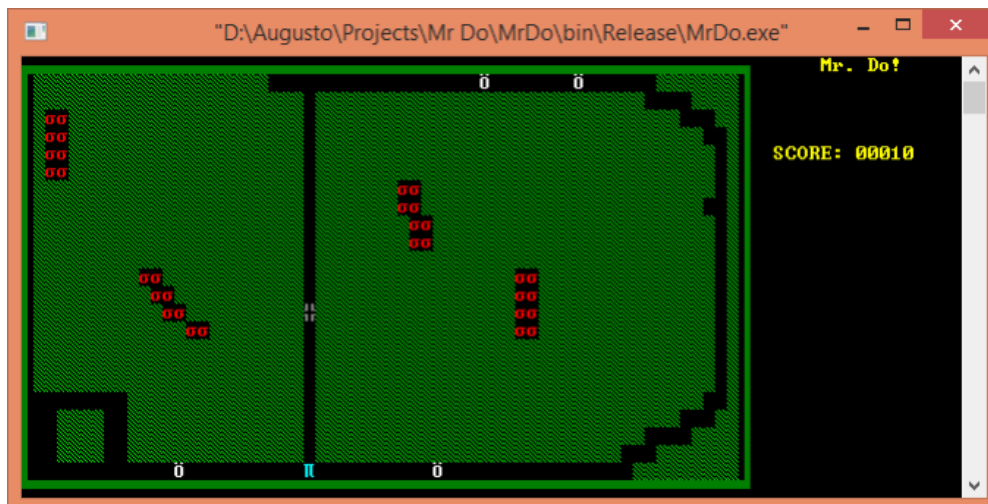









Figura 1: Exemplo de tela do jogo.

Nossa versão de *Mr Do!* é composta por:

- Um personagem (mr. Do) controlado pelo jogador.
- Um ninho de fantasmas.
- Um conjunto de fantasmas.
- Um conjunto de frutas.
- Um labirinto.

Personagem	Símbolo	Caracter ASCII
Mr Do!		227
Fantasmas		153
Ninho		206
Parede		176
Frutas		229
Tiros		250
Caminho		32

REGRAS DO JOGO

- **Mr Do!:**
 - Inicialmente localiza-se na base do cenário.
 - Pode mover-se para direita, esquerda, para cima ou para baixo, de acordo com as setas do teclado, respectivamente.
 - Efetua disparos para 'frente' quando o jogador pressiona 'espaço'.
 - Abre novos caminhos no labirinto quando passa por cima de uma parede.
 - Para colher frutas, basta colidir com elas.
 - Morre quando ocorre uma colisão com um fantasma.

- **Ninho de fantasmas:**
 - Localizam-se no centro do labirinto.
 - Não pode se mover nem ser destruído.
 - Todos os fantasmas nascem na posição do ninho.
- **Conjunto de fantasmas:**
 - Inicialmente localizam-se no 'ninho'.
 - Movem-se aleatoriamente pelo labirinto.
 - Não podem abrir novos caminhos no labirinto.
 - Se um fantasma é destruído por um tiro do Mr Do!, o jogador ganha 10 pontos.
 - Pode haver no máximo 10 fantasmas vivos ao mesmo tempo.
 - Nasce um fantasma a cada 3 segundos.
 - Nascem no máximo 10 fantasmas por fase.
 - Podem comer as frutas do cenário, sem dar pontos ao jogador.
 - Elimina o Mr Do! ao colidir com ele.
- **Um conjunto de frutas:**
 - Deve haver 32 frutas no início de cada fase espalhadas pelo labirinto.
 - Quando o jogador pega uma fruta, ele ganha 50 pontos.
 - Quando um fantasma pega uma fruta, a fruta é destruída.
- **Tiros:**
 - É disparado pelo Mr Do! quando o jogador pressiona 'espaço'.
 - O tiro explode quando colide com fantasmas, frutas ou paredes do labirinto.
 - A direção do tiro será a última direção pressionada pelo jogador.
 - Um novo tiro só pode ser dado se o anterior já atingiu algum alvo.
 - Se o tiro colidir com um fantasma, ambos são destruídos.
 - Não há número máximo de tiros.

DETALHES TÉCNICOS

- **Cenário:**
 - Será composto por uma matriz M de **char** de tamanho fixo em 23x60.
 - Toda a lógica do jogo será controlada na matriz M.
 - Essa matriz M deverá ser lida de um arquivo texto, com a configuração inicial da fase. O arquivo fase1.txt deve configurar a primeira fase. O arquivo fase2.txt deve configurar a segunda fase do jogo. O formato do arquivo é descrito no tópico '**Arquivo de Configuração Inicial**'.
- **Labirinto:**
 - É o conjunto de caracteres vazios (32 em ASCII) que formam um caminho no cenário.
- **Arquivo de Configuração Inicial:**
 - Um arquivo texto com o mesmo número de linhas e colunas da matriz M, configura o início de cada fase.
 - Cada caractere do arquivo será um **char** que indica a posição e o tipo de um elemento do jogo que será carregado na matriz M.
 - Você deve mapear os elementos do jogo de acordo com a seguinte estratégia:
 - 'p' = parede
 - 'i' = fantasma
 - 'f' = fruta
 - 'd' = Mr Do!
 - 't' = tiro
 - 'n' = ninho dos fantasmas

- 'v' = vazio (forma o labirinto)
- A Figura 2 ilustra um arquivo de configuração e o resultado carregado no jogo:

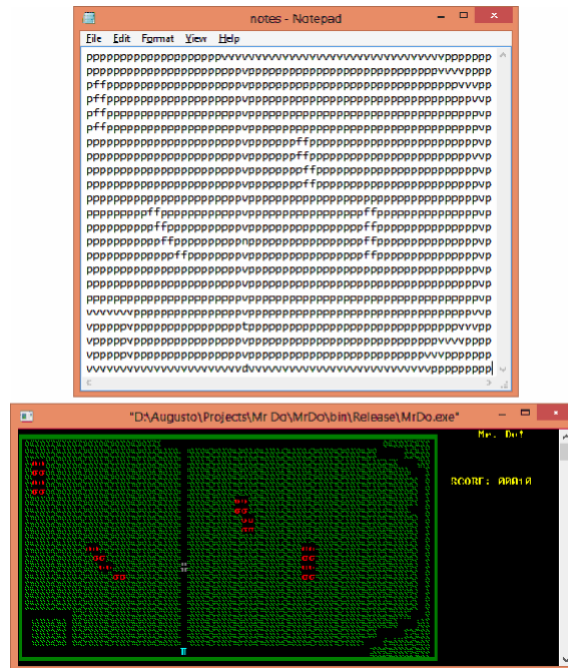


Figura 2: Exemplo de arquivo de inicialização da 1ª fase do jogo.

- **Menu Inicial:**
 - O jogo deve mostrar inicialmente um **menu** com as opções:
 - Novo Jogo
 - Continuar
 - High Scores
 - Sair
 - Um exemplo de **menu** pode ser observado na Figura 3.

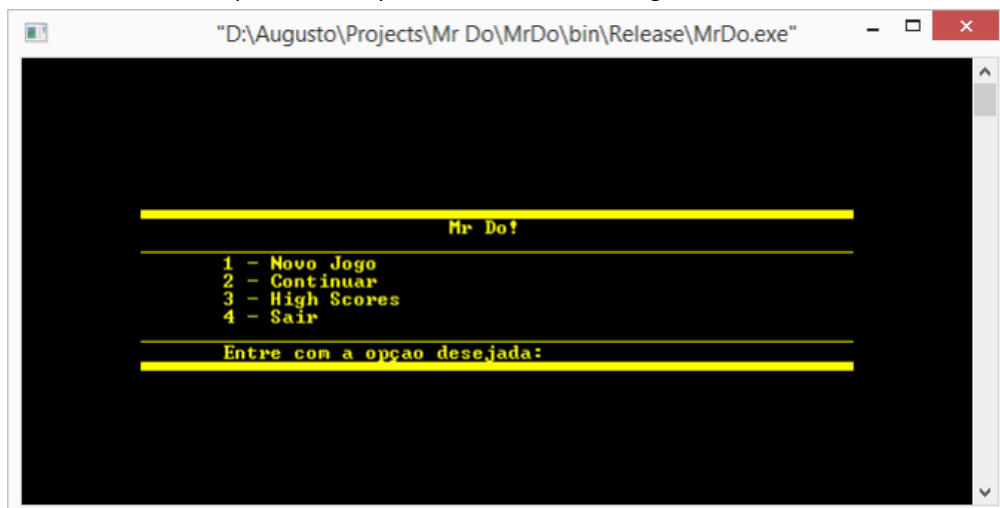


Figura 3: Menu inicial do jogo.

- **High Scores:**
 - O jogo deverá ter uma lista com os 5 melhores pontuadores.
 - Essa lista será atualizada ao final de cada jogo com a pontuação e o nome do jogador,

- caso ele tenha entrada na lista de 5 melhores pontuações.
- A lista deverá ser armazenada num arquivo binário chamado 'highscores.bin', definida como uma **struct** com os elementos 'nome' e 'pontos'.
- Uma visualização da lista deverá estar disponível no **menu** inicial do jogo (a exibição da lista deve ser ordenada).
- **Sair do Jogo**
 - Quando o jogador digitar a tecla 'ESC', o jogo deve ser encerrado.
 - O encerramento do jogo com 'ESC' deve também salvar todo o estado atual do cenário. Mais detalhes na seção '**Salvar Jogo**'.
 - Ao encerrar o jogo, o programa deve voltar ao menu inicial.
- **Salvar Jogo**
 - Quando o jogador decidir sair do jogo, o programa deve salvar o cenário com todos os seus **elementos** e o **estado** do jogo.
 - Os **elementos** do cenário devem ser salvos num arquivo texto chamado '**continuar.txt**', no mesmo formato especificado para a configuração das fases do jogo (na seção **Arquivo de Configuração Inicial**).
 - O **estado** do jogo será gravado num arquivo binário chamado 'estado.bin'. Uma **struct** deve ser criada para gravar as seguintes informações:
 - Pontuação do jogador
 - Quantidade de fantasmas que já nasceram
- **Movimentação dos Fantasmas**
 - Cada fantasma terá movimentações independentes.
 - O movimento de um fantasma é aleatório, porém obedece as seguintes regras:
 - Se fantasma está num cruzamento, sorteia nova direção que não seja a de onde ele veio e que não tenha obstáculo (parede).
 - Se não, se a próxima posição não for obstáculo (parede), continua caminhando no sentido da última direção.
 - Se não for nenhum dos casos anteriores, sorteia uma nova direção que não seja obstáculo (parede). Neste caso, pode escolher também a direção de onde o fantasma veio.
 - Na seção **Dicas** está disponível um pseudo-código que implementa as regras acima.

JOGABILIDADE

O jogo do Mr Do! que será criado pela dupla de alunos deve ser interativo, ou seja, será um programa que recebe interações do jogador via teclado, e em tempo real, interage com os personagens do jogo.

O **fluxo normal** do jogo deve iniciar na apresentação do menu, em seguida iniciar a fase 1, e caso o jogador completar a tarefa, apresentar a fase 2. Terminada a fase 2 com sucesso, o jogo é finalizado e retorna ao menu inicial.

REQUISITOS MÍNIMOS

O QUE SE ESPERA DO PROGRAMA

- O programa deve implementar todas as regras descritas até aqui.
- O código fonte deve compilar sem apresentar erros de compilação.
- O executável deve rodar sem mostrar erros de execução.
- Manipulação dos arquivos texto e binários:
 - fase1.txt
 - fase2.txt

- continuar.txt
- estado.bin
- highscores.bin
- Uso de **struct**.
- Carregamento de arquivo para configurar as fases.
- Duas fases para o jogo.
- Salvar e continuar um jogo.
- Atualização e exibição do High Scores.
- Algoritmo de movimentação dos fantasmas.
- Identação e documentação (comentários) do código fonte .
- Modularização do código (uso de funções e passagem de parâmetros).

RODADA DE TESTE NA APRESENTAÇÃO

Na apresentação do trabalho, além da demonstração do programa funcionando com os arquivos auxiliares do próprio aluno, haverá um teste no qual os arquivos auxiliares (fase1.txt, fase2.txt, continuar.txt, estado.bin, highscores.bin) serão fornecidos pelos professores.

FUNCIONALIDADES OPCIONAIS

Pontos extras serão atribuídos a quem desenvolver soluções para os desafios propostos. Entretanto, o aluno pode receber nota máxima se implementar de forma correta todos os itens obrigatórios. Outros desafios poderão ser considerados conforme sugestão dos alunos e avaliação pelo professor:

- Uso de cores diferentes para cada personagem.
- Um algoritmo mais sofisticado para a movimentação dos fantasmas.
- Controle da velocidade de movimentação dos fantasmas de acordo com uma rotina de tempo.
- Na segunda fase, tornar os fantasmas mais rápidos para aumentar a dificuldade em completar a fase.
- Implementar mais elementos no cenário, inspirados no jogo original (como as maçãs que caem sob o labirinto).

DICAS DE IMPLEMENTAÇÃO

Para escrever o jogo na tela é preciso controlar o cursor e definir onde devem ser escritos os dados da matriz. Pode-se utilizar a função `gotoxy()` para definir a posição de impressão na tela.

Para Windows:

```
#include <conio.h>
#include <Windows.h>

void gotoxy (int x, int y)
{
    COORD coord = {0, 0};
    coord.X = x; coord.Y = y; // X and Y coordinates
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}
```

Para Linux:

```
#include<stdio.h>

// posiciona cursor na posicao i j
void gotoxy(int x,int y)
{
    printf("%c[%d;%df",0x1B,y,x);
}
```

O programa funcionará em um laço principal e periodicamente deverá ler os comandos do jogador via teclado. Uma maneira de ler o teclado para analisar se o jogador pressionou alguma tecla é utilizar a função `kbhit()`. Para ler a tecla pressionada sem mostrá-la na tela, pode-se utilizar a função `getch()`.

```
#include <conio.h>

int main() {
    char c;

    if( kbhit() ) c = getch(); // lê a entrada do teclado somente quando há algo
    para ser lido

    return 0;
}
```

Usar estrutura para representar um componente do jogo (que guarde posição *i* e *j* e estado e ultima direção).

```
typedef struct
{
    int i;
    int j;
    int lastDirection;
    char status;
} Component;
```

O algoritmo de movimentação dos fantasmas foi descrito na seção **Detalhes Técnicos**. Uma sugestão de pseudo-código é apresentada abaixo:

```
if(isCross(grid, ghost)==1)
    newDirection(grid, ghost, 1);
else if(isNextWall(grid, ghost)==0)
    nextStep(ghost);
else
    newDirection(grid, ghost, 0);
```

Os protótipos para as funções acima:

```
int isCross(char grid[23][60], Component *ghost);

void newDirection(char grid[23][60], Component* ghost, int ignoreLast);

int isNextWall(char grid[23][60], Component* ghost);

void nextStep(Component* ghost);
```

OUTRAS INFORMAÇÕES

- O trabalho deverá ser realizado em **duplas**. Informar os componentes da dupla.
- Até **23:59** do dia **03 de dezembro**, a dupla deverá submeter via Moodle um arquivo zip cujo nome **deve** conter os nomes dos alunos. O arquivo zip deve conter:
 - Um arquivo com uma descrição do trabalho realizado contendo a especificação completa das estruturas utilizadas e uma explicação de como usar o programa;
 - os códigos-fonte devidamente organizados e documentados (arquivos .c);
 - o executável do programa.
- O trabalho será **obrigatoriamente** apresentado durante a aula prática do dia **04 de dezembro**. Ambos membros da dupla deverão saber responder perguntas sobre **qualquer** trecho do código;
- Os seguintes itens serão considerados na avaliação do trabalho:
 - estruturação do código em módulos;
 - documentação geral do código (comentários, indentação);
 - “jogabilidade” do jogo;
 - atendimento aos requisitos definidos;
- **Importante:** trabalhos copiados não serão considerados. Saibam que há ferramentas que possibilitam a detecção automática de plágio.