

TRABALHO PRÁTICO FINAL

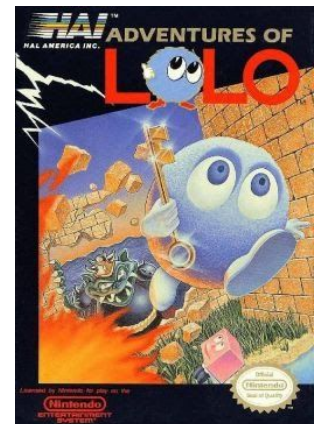
Objetivo

Exercitar as habilidades e conceitos de programação desenvolvidos ao longo da disciplina através da implementação de uma aplicação em C, desenvolvida por um grupo de 2 alunos da mesma turma ou de turmas diferentes (A e B). A coordenação de um trabalho em equipe faz parte da habilidade de programar.

O programa deve ser estruturado de forma a receber um conjunto de entradas (no início da execução ou durante o uso do programa), cuja consistência deve ser verificada, processá-lo, e fornecer uma ou mais saídas.

Contextualização

O trabalho a ser desenvolvido consiste na programação de um jogo, chamado [Lolo's Adventure](#), lançado em 1989 para a plataforma NES. Neste jogo, o jogador controla Lolo, que deve salvar Lala, presa no topo de uma torre cercada por monstros. Em cada fase do jogo, Lolo deve derrotar alguns destes inimigos e encontrar a chave que abre a porta para o estágio subsequente. Um vídeo demonstrando as fases do jogo pode ser encontrado neste [link](#). O objetivo do trabalho é implementar uma versão simplificada do jogo **Lolo's Adventure** em modo texto (*ASCII art*).



Interface do jogo

A implementação do espaço do jogo deve ser feita em modo texto e pode adotar o tamanho de janela padrão de execução do Code::Blocks (25 linhas por 80 colunas).

Esta interface deve conter:

- Menu principal: A tela inicial do programa deve apresentar ao usuário um menu com as seguintes opções:
 - Novo Jogo: Se esta opção for selecionada, o programa deve solicitar o nome do usuário e prosseguir para a primeira fase do jogo.
 - Carregar Jogo: Se esta opção for selecionada, o programa deve mostrar a lista de usuários (com nome e nível atingido) e permitir ao usuário continuar o jogo de onde parou.
 - Créditos: Deve apresentar o nome dos alunos que criaram o jogo.

- Sair: Opção deve encerrar o jogo.
- Área de jogo: Cada fase do jogo deve ser representada por um plano cartesiano discreto de duas dimensões. Cada coordenada pode estar em um dos seguintes estados:
 - Livre: permite Lolo ou um inimigo a andar para aquele espaço em um determinado momento do jogo;
 - Lolo: o espaço ocupado por lolo no momento;
 - Inimigo: indica que o espaço está ocupado por um inimigo;
 - Pedra: nem Lolo, nem os inimigos podem andar para esta posição;
 - Bloco Móvel: representam blocos que podem ser movidos por Lolo. Lolo deve mover estes blocos os empurrando com a cabeça;
 - Água: apenas Lolo pode mover para espaços marcados como água. Porém, se o fizer, Lolo se afogará e perderá uma vida;
 - Corações: Fazem com que Lolo ganhe super poderes e possa atingir UM inimigo, eliminando-o da fase;
 - Baú: quando Lolo encosta no baú, a porta de saída da fase se abre. O baú aparece apenas quando todos os inimigos da fase forem eliminados. Assuma que a porta de saída está localizada na mesma posição que o baú no mapa.

Os menus do jogo devem ser textuais. Quando solicitado para que o usuário escolha uma opção dentre várias, as opções devem aparecer na tela e estarem numeradas corretamente. Enquanto o menu estiver ativo, o jogo fica pausado, isto é, não é permitido movimentação do personagem. Quando uma fase estiver sendo mostrada (corresponde à parte jogável da aplicação), além dos elementos da área de jogos, os itens abaixo também devem ser mostrados:

- Marcador de vidas: mostra quantas vidas Lolo tem;
- Marcador de pontuação: mostra a pontuação obtida no jogo até então;
- Número da fase atual.

Funcionamento do jogo

Lolo vence a fase se conseguir eliminar todos os inimigos e abrir o baú que contém a chave da saída. Para eliminar um inimigo, Lolo precisa encostar em algum dos corações espalhados pela fase. Assim, cada fase deve possuir um número de corações igual ao número de inimigos, assim Lolo poderá eliminar cada inimigo consumindo exatamente um coração. Ao encostar em um inimigo sem ter encostado em um coração, Lolo será eliminado da fase. Porém, se encostou em um coração primeiro, Lolo ficará forte, e poderá encostar no inimigo, eliminando-o. Caso encoste em um coração enquanto sob o efeito de outro, o efeito deve acumular.

O objetivo de cada fase é encontrar os corações, eliminar os inimigos e prosseguir para a próxima fase. A dificuldade do jogo se dá pelo número de inimigos presentes em cada fase

(crescente na medida com que o jogo progride) e no posicionamento dos corações (com acesso mais difícil na medida com que o jogo progride).

- Pontuação e Vidas: Para cada inimigo eliminado, Lolo contabiliza 1 ponto. A pontuação é persistente, isto é, acumula entre as fases. Lolo ganha uma vida por alcançar 10 pontos, isto é, quando elimina o décimo inimigo consecutivo. Se for eliminado, a fase recomeça com a pontuação zerada, além de Lolo perder uma vida.
- Controles: Lolo é capaz de se mover nos eixos X e Y do plano. Os movimentos possíveis são:
 - Mover para um espaço vazio. Lolo assume o espaço vazio e libera o espaço onde estava.
 - Mover para um espaço onde há um coração. Lolo consome o coração e assume o lugar onde o coração estava.
 - Mover para onde há um inimigo. Se Lolo consumiu algum coração anteriormente, então aquele inimigo é eliminado do jogo e Lolo assume seu lugar. Caso contrário, Lolo é eliminado.

Para a movimentação, é recomendado que se usem as setas do teclado.

Além destes, deve ser permitido ao usuário sair de uma fase em andamento e voltar para a tela inicial do jogo, pressionando a tecla ESC.

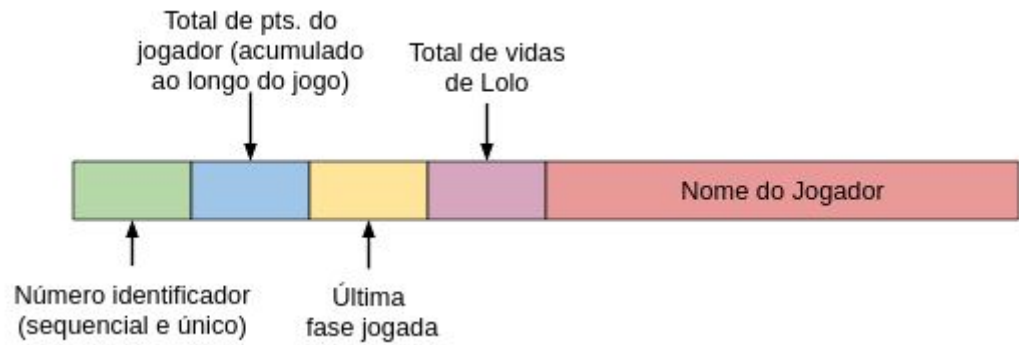
- Inimigos: Cada vez que Lolo se move, os inimigos também se movimentarão aleatoriamente, ocupando uma nova posição "válida" (posição livre no cenário). Nesta versão do jogo, apenas um tipo de inimigo deve ser implementado.

Implementação

Seguem as regras de como o programa deve ser estruturado.

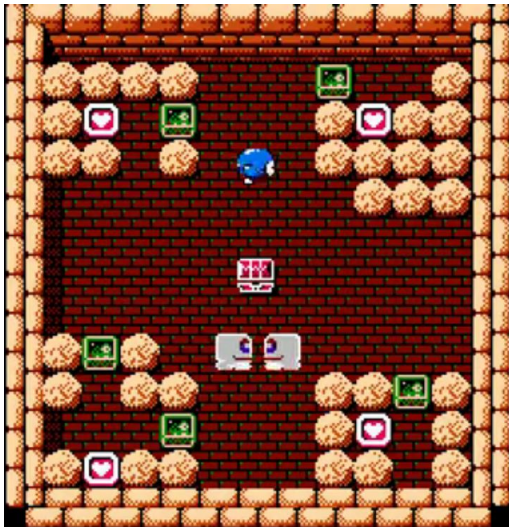
1. Os menus devem ser implementados conforme os exercícios praticados ao longo da disciplina.
2. Aos jogadores é permitido gravar seu progresso. Cada gravação de progresso deve ser guardado em um arquivo binário. A estrutura a ser gravada é composta por cinco elementos, nesta ordem:
 - o número da gravação (deve ser sequencial e único),
 - o total de pontos daquele jogador até o momento,
 - um inteiro indicando a última fase jogada,
 - A quantidade de vidas de Lolo.
 - o nome do jogador, com limite de 8 caracteres.

A estrutura do registro é ilustrada abaixo.



3. Os mapas do jogo devem ser armazenados em arquivos de texto e codificados como a seguir:
- '@' denota a posição inicial de Lolo
 - 'P' denota pedra
 - 'A' denota água
 - 'C' denota coração
 - 'B' denota bloco movível. Lolo pode movimentar este bloco ao encostar no mesmo. A movimentação ocorre no mesmo sentido em que Lolo se desloca.
 - 'E' denota inimigo
 - 'T' denota baú.
 - Espaços em branco denotam espaços não ocupados na fase, nos quais Lolo e os inimigos podem se movimentar.

Um exemplo de mapa é mostrado abaixo.

 <p>Mapa original</p>	<pre> PPPPPPPPPPPPPP PPPPP B PP PPC B PCPPP PPP P @ PPPPP P PPPP P P P B P P P PPBP E E P PP PP PPBPP P B PC PP PPCPP PP PP PPPPPPPPPPPPPP </pre> <p>Versão a ser utilizada no jogo, gravada em um arquivo de texto.</p>
--	---

Diferentes grupos de trabalho podem compartilhar mapas entre si com a finalidade de testar suas implementações. Porém, cada grupo deve entregar 3 mapas

juntamente com o código do programa. Os mapas podem ser imitações dos mapas originais do jogo (consulte o Google) ou criados pelos componentes do grupo. O mapa mostrado acima pode ser considerado a primeira fase do jogo.

Sugestões para implementação

Seguem algumas sugestões de estruturas de dados para serem utilizadas no jogo:

Estrutura para armazenar o registro de gravação de progresso.	<pre>struct gravacao { int identificador; int totalpts; int ultimafase; int vidas; char nomejogador[9]; };</pre>
Estrutura para representar uma fase	<pre>struct fase { int tamanhox; int tamanhoy; char elementos[x][y]; }</pre>
Estrutura para representar posições no mapa	<pre>struct ponto { char x; char y; }</pre>
Laço principal do jogo	<pre>repetir enquanto não encontrar o baú { Mostrar mapa; Ler ação do jogador; Tratar ação do jogador; }</pre>

Produto do trabalho e datas de entrega:

O trabalho será entregue em 2 etapas:

- A. Indicação das duplas de trabalho: 09 de outubro (sexta-feira).** A indicação será feita através de um documento online compartilhado pela professora, onde deverão ser informados os nomes dos integrantes da dupla.
- B. Entrega do código documentado e do vídeo de apresentação: 23 de novembro (segunda-feira) até às 9 horas pelo Moodle.** Upload no Moodle em tarefa própria de um ÚNICO arquivo compactado cujo nome do arquivo é o nome dos alunos contendo:
 - a. Código documentado (arquivos .c). Inclua o nome dos autores no cabeçalho do programa. Bibliotecas extras devem ser incluídas no código e se necessário,

adicionadas ao arquivo compactado. O programa deve rodar em Codeblocks ou compiladores online.

- b. Programa executável, gerado a partir do seu código.
- c. Link para vídeo de apresentação do trabalho. Este vídeo pode ser depositado no Youtube como 'não listado' ou em outro repositório online, desde que acessível via link para a professora. O vídeo deve demonstrar o jogo em execução, com suas principais funcionalidades (acesso a menus, jogo em ação para algumas fases, salvar e carregar jogo em andamento, etc.), e descrição das estruturas gerais utilizadas para implementação (se diferentes das sugestões dadas). A **duração máxima deve ser de 5 minutos**. O vídeo deve ter a participação de ambos os alunos, entretanto, não é necessário que os alunos apareçam no vídeo se não desejarem. Pode ser realizada apenas a gravação da tela do jogo, com narração dos alunos. Atenção: não deve ser depositado no Moodle o arquivo do vídeo, apenas o link!

Havendo exceções aos requisitos listados neste enunciado, descreva como comentário no início do arquivo fonte.

Você pode fazer upload de diferentes versões e ir aperfeiçoando o programa. Faça o upload assim que tiver uma versão executável, de modo a garantir a entrega e precaver-se de problemas com servidor, redes, internet, etc.

Avaliação

O programa deve atender todos os requisitos listados neste enunciado e não deve apresentar erros de compilação. Deve executar normalmente. Pontos serão reduzidos caso contrário.

A aplicação desenvolvida deverá demonstrar os seguintes conteúdos que serão avaliados:

- (2 pontos) Habilidade em estruturar programas pela decomposição da tarefa em subtarefas, utilizando subprogramas para implementá-las.
- (1 pontos) Documentação de programas (indentação, utilização de nomes de variáveis, abstração dos procedimentos para obter maior clareza, uso de comentários no código).
- (2 pontos) Domínio na utilização de tipos de dados simples e estruturados (arranjos, estruturas) e passagem de parâmetros. Uso exclusivo de variáveis locais.
- (1 ponto) Formatação e controle de entrada e saída, com construção de interfaces que orientem corretamente o usuário sem instruções ou intervenção adicional do programador.
- (1 ponto) Utilização de arquivos binários e de texto.
- (2 pontos) Atendimento aos requisitos do enunciado do programa: modelo de estrutura de dados, de interação e de relatórios, ordenação dos dados, opções do programa, etc..

- (1 ponto) Apresentação do trabalho no formato de vídeo pré-gravado, demonstrando a funcionalidade do jogo implementado.

A nota do trabalho prático corresponderá a 20% da nota final. A apresentação do trabalho prático, mesmo que rodando parcialmente, é pré-requisito para realizar a recuperação.

Dicas gerais

1. Organize o código em diferentes arquivos, separando as partes que implementam o comportamento do jogo das partes que implementam a interface. Isso facilita a modificação de uma parte, sem afetar o resto (por exemplo, a troca de uma interface inicial, mais simples, para uma mais complexa).
2. Implemente “**cheat codes**” para facilitar os testes do seu programa, por exemplo: teclas que aumentam/diminuem vidas, etc.. Com isso, é possível testar essas situações sem ter que esperar as condições para elas;
3. Quanto à implementação da interface via terminal: Para montar uma interface é possível manipular o terminal de forma avançada com algumas bibliotecas externas (ou seja, que não são padrão da linguagem C). As bibliotecas mais utilizadas para isso são a `conio2` (para Windows), `ncurses` (para Linux) e `rlutil` (para Windows e Linux). Todas disponibilizam funções básicas para captura de entrada sem bloqueio (permitindo interação em tempo real), troca de cores e a inclusão de caracteres em lugares arbitrários na tela. Veja alguns exemplos ou documentação das bibliotecas. Bibliotecas externas devem ser incluídas junto ao arquivo compactado do trabalho.
 - a. Biblioteca `conio.h`: <http://www.programmingsimplified.com/c/conio.h>
 - b. Biblioteca `ncurses.h`:
<https://www.viget.com/articles/game-programming-in-c-with-the-ncurses-library>
 - c. Biblioteca `rlutil.h`: <https://github.com/tapio/rlutil>

O mais importante no uso da `conio.h` ou `ncurses.h` é a entrada e saída sem bloqueio (ou seja, ler os comandos do usuário sem bloquear o programa). Para fazer isso com a `conio.h`, basta usar o par de funções `kbhit()` e `getch()`. A função `kbhit()` retorna 1 quando existem caracteres para ler e `getch()` lê o caractere:

```
#include <stdio.h>
#include <conio.h>
main() {
    // repete multiplas vezes, sem bloquear, ate que o
    // usuario digite uma tecla
    while (!kbhit()) {
        printf("Voce ainda nao pressionou uma tecla.\n");
    }
    char t = getch();
    printf("A tecla que voce apertou foi %c", t);
    return 0;
}
```

Para fazer isso em ncurses.h é preciso definir a função kbhit() - veja abaixo. Após feito isto, é só usar normalmente, fazendo a chamada à função como no exemplo acima para a conio.h.

```
int kbhit(void){
    int ch = getch();
    if (ch != ERR) {
        ungetch(ch);
        return 1;
    } else {
        return 0;
    }
}
```

Outra funcionalidade importante é a possibilidade de inserir caracteres em posições específicas, sem precisar redesenhar toda a tela a cada ciclo do jogo. Basta desenhar os elementos que mudam (por exemplo, as margens do jogo nunca mudam). As funções que permitem fazer isso e outras que adicionam outros recursos úteis (como alteração das cores dos caracteres e fundo), estão descritas a seguir:

```
/* FUNCOES DE POSICIONAMENTO EM TELA E CORES */
void clrscr(); // limpa a tela
void gotoxy (int x, int y); // posiciona o cursor em (x,y)
void cputsxy (int x, int y, char* str); //imprime str na posicao x, y
void putchxy (int x, int y, char ch); //imprime char na posicao x, y
void textbackground (int cor); // altera cor de fundo ao escrever na tela
void textcolor (int cor); // altera cor dos caracteres ao escrever na tela
```

```
/* FUNCOES E DEFINICOES UTEIS DA CONIO2 */
/** * Colors which you can use in your application. */
typedef enum {
    BLACK,      /**< black color */
    BLUE,       /**< blue color */
    GREEN,      /**< green color */
    CYAN,       /**< cyan color */
    RED,        /**< red color */
    MAGENTA,    /**< magenta color */
    BROWN,      /**< brown color */
    LIGHTGRAY,  /**< light gray color */
    DARKGRAY,   /**< dark gray color */
    LIGHTBLUE,  /**< light blue color */
    LIGHTGREEN, /**< light green color */
    LIGHTCYAN,  /**< light cyan color */
    LIGHTRED,   /**< light red color */
    LIGHTMAGENTA, /**< light magenta color */
    YELLOW,     /**< yellow color */
    WHITE       /**< white color */
} COLORS;
```

```
/* FUNCOES E DEFINICOES UTEIS DA CONIO.H */
char getch(); // devolve um caractere lido do teclado, sem eco na tela
int kbhit(); // devolve true se alguma tecla foi pressionada, sem eco na tela
```

```
/* FUNCOES E DEFINICOES UTEIS DA WINDOWS.H */
void Sleep (int t); // paralisa o programa por t milissegundos
void GetKeyState (int tecla); // pode ser usada para ler as setas do teclado
```


Exemplo de uso (trecho):

```
if (GetKeyState (VK_RIGHT) & 0x80) // se a tecla “seta para direita” está pressionada
```

```
/* FUNCOES E DEFINICOES UTEIS DA TIME.H */
```

```
clock();
```

Exemplo de uso (trecho de código):

```
double tempo;  
clock_t inicio, fim;  
inicio = clock(); // salva tempo ao iniciar  
(...) // trecho do programa  
fim = clock(); // salva tempo ao terminar  
tempo_total_segundos = (int) ((fim - inicio) / CLOCKS_PER_SEC); /* calcula o  
número de segundos decorridos durante a execução do trecho do programa*/
```