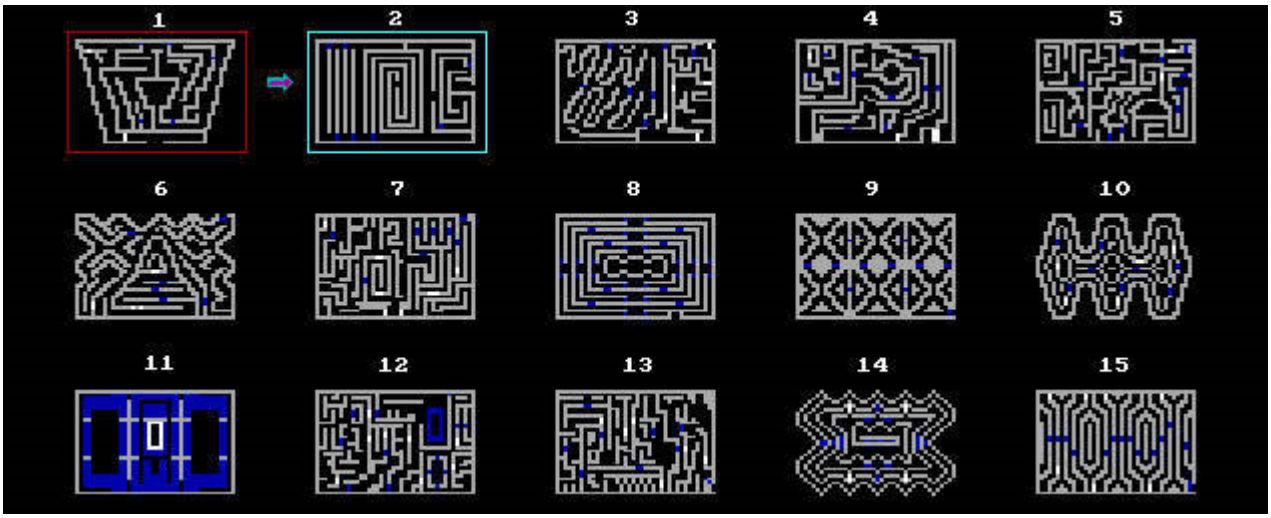
Turmas A e B
Trabalho Prático Final - Semestre 2023/1
Os Labirintos do INF



MOTIVAÇÃO E OBJETIVOS

O objetivo deste trabalho é exercitar as habilidades e conceitos de programação desenvolvidos ao longo da disciplina pela implementação de um jogo em C, em grupos de 2 alunos.

Deverá ser desenvolvido um jogo inspirado no ***"Laberinto del Saber"***, jogo criado em Havana/Cuba no ano 1988 e relançado em 2013 como ***"Maze of Knowledge"***. Nesse jogo, o jogador deve passar por diferentes labirintos, tendo um tempo limite para cada um deles. Em caso de não finalizar um labirinto no tempo dado, o jogador perde uma vida e re-inicia no labirinto que ficou. Em cada labirinto o jogador deve coletar chaves que lhe permitirão abrir as portas de saída, enquanto escapa ou enfrenta guardiões que se movimentam pelo labirinto. Ao enfrentar um guardião, uma pergunta de cultura geral com múltipla escolha é feita, se o jogador acertar, vence o confronto e o guardião desaparece, em caso contrário perde uma vida. Se o jogador perde todas as vidas, então o jogo finaliza em derrota. Se o jogador finaliza todos os labirintos então ganha. Ademais dos guardiões, o jogador pode encontrar estátuas no labirinto, estas representam jogadores anteriores que perderam naquela posição. Quando o jogador encontra uma estátua, pode tentar salvá-la, solucionando uma questão (semelhante ao confronto com os guardiões). Se o jogador não soluciona corretamente a questão, a estátua se destrói e desaparece, em caso contrário ela retorna à vida e (antes de desaparecer) deixa um item aleatório para o jogador que pode ser tempo extra, uma vida extra ou uma bomba. O tempo extra é adicionado imediatamente ao tempo restante que o jogador possui para completar o labirinto. As bombas, podem ser usadas (uma única vez) para explodir paredes do labirinto (criando atalhos).

Na nossa versão, o jogo se chamará **Os labirintos do INF**. Nesta versão, o jogador representará um aluno tentando vencer cada semestre (os labirintos). Os guardiões serão professores caminhando pelos labirintos e as estátuas representam colegas (outros alunos) que precisam ajuda. Antes de poder sair de cada labirinto, ao invés de recolher chaves para abrir portas, o aluno deverá acumular uma quantidade mínima de créditos, que estarão espalhados pelos labirintos e também poderão ser obtidos ao responder corretamente as perguntas dos professores.

REQUISITOS DO PROGRAMA

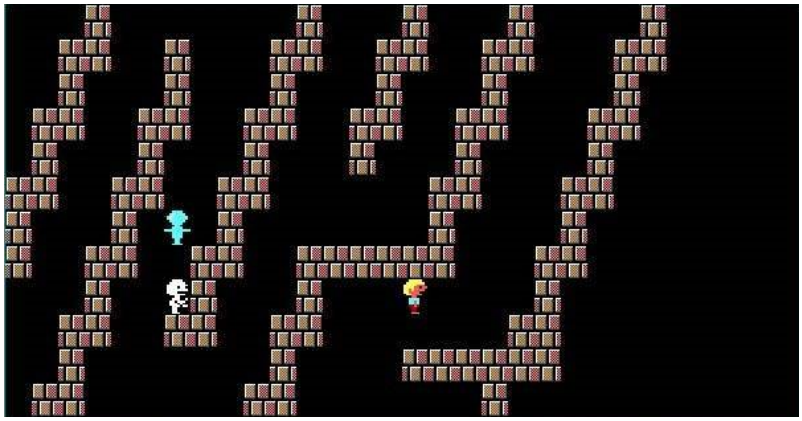
As duplas terão liberdade para a implementação do programa. Entretanto, os seguintes requisitos devem ser respeitados:

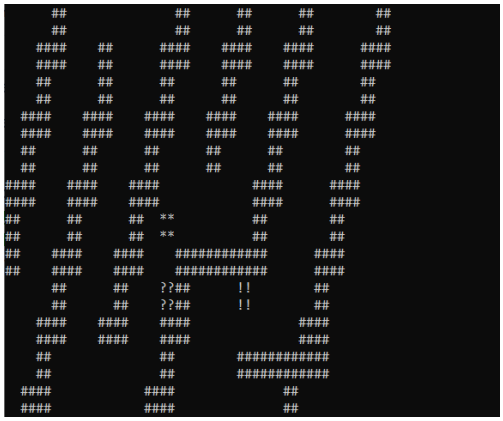
- Cada fase do jogo será representada por um plano cartesiano de duas dimensões, segmentado em quadrantes. Cada quadrante deve representar um dos seguintes elementos:
  - **Livre**. Representa um área livre que pode ser ocupada posteriormente pelo aluno (jogador) ou por um professor.
  - **Parede**. Representa um área que não pode ser ocupada pelo aluno nem pelos professores.
  - **Aluno**. Representa a posição em que está o jogador (só poderá ocupar um quadrante por vez).
  - **Professor**. Representa a posição em que está um professor (cada professor só pode ocupar um quadrante por vez). Quando o aluno estiver em um bloco adjacente ao de um professor, deve solucionar a questão do professor. Se soluciona corretamente, o professor desaparece e deixa nesse bloco um item de tipo crédito, em caso contrário o professor desaparece e o aluno perde uma vida.
  - **Colega**. Representa a posição em que está um outro aluno que precisa ajuda (estes não se movimentam). Quando o aluno estiver em um bloco adjacente ao de um colega, deve solucionar a questão de ajudar ao colega. Se soluciona corretamente, o colega desaparece e deixa nesse bloco um item aleatório. em caso contrário o colega só desaparece. Os itens que o colega pode deixar são: bomba inativa, relógio ou coração.
  - **Bomba inativa**. Representa a posição em que está um item de tipo bomba inativa. Se o aluno entra nesse bloco, a bomba desaparece e forma parte do inventário do aluno.
  - **Bomba ativa**. Representa a posição em que está um item de tipo bomba ativa. A bomba e os blocos adjacentes a ela, irão ficar livres após a explosão. Elimina paredes, professores, colegas e itens. Se o aluno for pego pela explosão, perde uma vida e deve re-iniciar o labirinto atual.
  - **Relógio**. Representa a posição em que está um item de tempo extra. Se o aluno entra nesse bloco, o relógio desaparece e incrementa o tempo do aluno para finalizar o labirinto.
  - **Crédito**. Representa a posição em que está um item de tipo crédito. Se o aluno entra nesse bloco, o crédito desaparece e incrementa o total de créditos do aluno nesse labirinto.
  - **Coração**. Representa a posição em que está um item de tipo vida extra. Se o aluno entra nesse bloco, o coração desaparece e incrementa o número de vidas do aluno.
  - **Entrada**. Representa a posição em que começa o labirinto atual. Ao inciar ou re-iniciar o labirinto, o aluno sempre aparece nesta posição.
  - **Saída**. Representa a posição em que está o umbral de saída para o próximo labirinto. Se aluno tiver a quantidade mínima de créditos para finalizar o labirinto, então passa ao próximo labirinto ao entrar neste bloco.
- O item de tipo **bomba ativa** tem o tempo de explosão fixo, já os itens **relógio** e **crédito** devem ter um valor que é selecionado aleatoriamente dentre um conjunto, quando o item é criado. Deve ser possível identificar os valores dos relógios e dos créditos.
- Durante as fases, os seguintes informações devem aparecer juntamente a área de jogo:
  - **Marcador de vida**. Indicando quantas vidas ainda tem o aluno.
  - **Marcador de tempo**. Indicando o tempo restante do aluno para completar o labirinto atual;

  - **Marcador de créditos**. Indicando o número de créditos que faltam para poder sair do labirinto atual.
  - **Número do labirinto**. Indicando qual o labirinto em que o aluno está atualmente.
  - **Pontuação**. Indicando a pontuação total acumulada pelo aluno até o momento. A pontuação é calculada como segue: $10 \cdot \max \cdot L + \frac{C}{T}$, onde: max é um número maior que a quantidade de créditos necessários para passar qualquer labirinto, L é a quantidade de labirintos terminados, C o total de créditos acumulados ao longo do jogo e T o tempo total consumido durante o jogo.

- Se o jogo acaba porque acabaram as vidas, o jogador deve ser armazenado em um arquivo binário como colega a ser ajudado por próximos jogos (deve aparecer sempre na posição em que perdeu o jogo). Se durante o jogo o aluno consegue ajudar um colega, esse colega deve ser retirado do arquivo binário.
- Sempre que se inicia ou re-inicia um labirinto, deve:
  - Sortear aleatoriamente itens de crédito que somem a totalidade requerida para passar. Tais itens devem ser distribuídos de forma aleatória pelo labirinto.
  - Sortear e distribuir pelo labirinto de forma aleatória os professores.
  - Carregar os colegas nas posições em que perderam e se não houver ou o número for muito baixo (menor que três), distribuir aleatoriamente pelo menos três colegas pelo labirinto.
  - Colocar o aluno (jogador) na posição de entrada.
- Se o aluno estiver na linha de visão de um professor (se não houver paredes entre eles) o professor deve perseguir o aluno. Em caso contrário, o professor deve definir uma direção aleatória e se movimentar até ela.
- A movimentação do aluno deve ser usando as teclas de direção do teclado.
- O bloco em que o aluno está sempre deve ficar visível.
- Durante as fases, também deve ser possível pausar o jogo com as opções de: salvar, sair, voltar, reiniciar labirinto atual ou novo jogo.
- O jogo deverá possuir uma tabela de pontuação que deverá ser carregada através de um arquivo binário. Nesse arquivo serão salvos a pontuação e o nome do jogador em ordem decrescente, no mínimo 10.
- Quando o jogador salva o estado atual do jogo, este deve ficar armazenado em um arquivo binário. Deve ser possível carregar o jogo salvo e continuar pelo ponto em que ficou.
- Deve haver pelo menos 10 labirintos, com formatos diferentes. Estes devem ser armazenados e carregados de um arquivo binário. Cada labirinto deve ter um nível de dificuldade maior que o anterior. Cada labirinto deve ter pelo menos 100×100 blocos e a seção visível em cada momento do jogo deve ser no máximo $\frac{1}{4}$ do menor dos labirintos.
- Deve haver pelo menos 100 questões de múltipla escolha que serão carregadas de um arquivo de texto. A formatação do arquivo é como segue, por cada questão: haverá uma linha com o texto da pergunta, seguida por outra linha com dois inteiros n e k, onde n é o total de escolhas e k a escolha correta; seguem então n linhas, cada uma com o texto da escolha. O total de questões não é informado no arquivo, a leitura das perguntas finaliza quando chega no **EOF**. Sua implementação deve permitir que se no arquivo são adicionadas até 10.000 questões, então todas elas possam ser usadas no jogo. As questões a serem resolvidas devem ser selecionadas aleatoriamente com probabilidade uniforme do conjunto de questões.

- A dificuldade dos labirintos não é só pelo formato, mas também pela quantidade de professores que vão ter, a velocidade destes, a quantidade de créditos e o tempo dado para finalizá-los. Tendo isto em conta, a cada nível aumente a dificuldade e quando o jogador começar um novo jogo pergunte se deseja: fácil, médio ou difícil.
- O jogo deve possuir uma tela que mostre um menu inicial com as seguintes opções: novo jogo, carregar jogo, exibir ganhadores, informações e sair. a opção informações deve explicar a representação de cada bloco e no caso dos itens bomba ativa, relógio e crédito como são os valores destes.
- O jogo deve reconhecer quando o jogo acabou, ou seja, quando a peça do jogador foi consumida ou explodida;
- Ao final do jogo, o programa deverá verificar se a pontuação do jogador entra na tabela dos *Ganhadores*. Caso afirmativo, o programa deve solicitar a leitura do nome do jogador e atualizar o arquivo binário com os ganhadores.
- O jogo pode ser desenvolvido usando bibliotecas visuais como a **raylib** ou no console em formato de texto. Se for desenvolvido no console, segue um exemplo visual de como uma parcela do labirinto (jogo original na esquerda) poderia ser representada com texto (na direita). Nesse exemplo, se considera que cada bloco é um quadrado de quatro (4) caracteres, onde # é usado para compor paredes, ? para definir professores, $*$ para definir colegas e ! para definir o aluno.





ENTREGA E AVALIAÇÃO

- O trabalho deverá ser feito em duplas. Informar os integrantes da dupla até o 04/08/2023.
- Até o 28/08/2023, cada a dupla deverá submeter via Moodle um arquivo zip cujo nome deve conter o(s) nome(s) do(s) aluno(s). O arquivo zip deve conter:
  - Uma descrição do trabalho realizado contendo a especificação completa das estruturas usadas e uma explicação de como usar o programa;
  - Os programas-fonte devidamente organizados e documentados (arquivos.c e arquivos.h);
  - Executável do programa.
- O trabalho será obrigatoriamente apresentado durante a aula prática do 01/09/2023. Ambos membros da dupla deverão saber responder perguntas sobre qualquer trecho do código.
- No dia da apresentação serão realizados diferentes testes de condições de jogo para testar o programa.

- Os seguintes itens serão considerados na avaliação do trabalho: estruturação do código em módulos, documentação geral do código (comentários, indentação), jogabilidade e atendimento aos requisitos definidos.
- **Importante**: trabalhos copiados não serão considerados. Serão considerados copiados, trabalhos cujo códigos códigos apresentem com alto índice de similaridade.

BIBLIOTECAS

A seguir são relacionadas algumas constantes, estruturas e funções por bibliotecas que poderão ser de utilidade no desenvolvimento do projeto.

- **time.h**: (empregada para medir o tempo)
  - `clock_t`: tipo para representar número de *clocks* em dado momento (recomendado realizar *casting* para *int* ou *double* na hora de realizar os cálculos).
  - `CLOCKS_PER_SEC`: constante que indica quantos *clokc* acontecem em um segundo.
  - `clock()`: função que retorna a quantidade de *clocks* atuais.

  Exemplo:

  clock_t inicio, fim;

  inicio = clocks();

  //Algum código a ser executado

  fim = clocks();

  printf("Demorou %d segundos", (int) ((fim - inicio) / CLOCKS_PER_SEC));
- **conio.h**: (empregada para algumas operações de I/O)
  - `kbhit()`: função que retorna verdadeiro se alguma tecla foi apertada. Deve ser usada em conjunto com `getch()` para obter a tecla em questão.
  -

  Exemplo:

```
if(kbhit()) {
  switch(getch()) {
    case 'a' ... 'z':
    case 'A' ... 'Z':
      printf("Apertou uma letra!");
      break;
    case 13:
      printf("Apertou o ENTER!");
      break;
    case 72:
      printf("Apertou o UP!");
      break;
}
```

- **windows.h**: (empregada para algumas operações de visualização)
  - `HANDLE`: tipo usado para controlar o *prompt* de comandos.

- `GetStdHandle(int)`: função que permite obter o controlador do *prompt* de comandos.
- `STD_OUTPUT_HANDLE`: constante que permite indicar que se deseja obter o controlador do *prompt* de comandos.
- `COORD`: tipo que permite representar coordenadas no *prompt*.
- `SetConsoleCursorPosition(HANDLE, COORD)`: função que coloca o cursor do *prompt* na coordenada indicada.
- `SetConsoleTextAttribute(HANDLE, int)`: função que permite trocar atributos de texto do *prompt* antes de imprimir (ex. cor da fonte ou cor do fundo).
- cores principais: (combinando-as com o operador | se podem obter outras cores)
  * `BACKGROUND_RED`, fundo vermelho
  * `BACKGROUND_GREEN`, fundo verde
  * `BACKGROUND_BLUE`, fundo azul
  * `FOREGROUND_RED`, texto vermelho
  * `FOREGROUND_GREEN`, texto verde
  * `FOREGROUND_BLUE`, texto azul

Exemplo:

HANDLE prompt = GetStdHandle(STD_OUTPUT_HANDLE);

COORD coordenada;

coordenada.X = 20;

coordenada.Y = 5;

//Posiciona o cursor do prompt na linha 5, coluna 20:

SetConsoleCursorPosition(prompt, coordenada);

//Troca a cor para escrever com letra vermelha

SetConsoleTextAttribute(prompt, FOREGROUND_RED);

printf("Texto vermelho");

coordenada.Y = 2;

//Posiciona o cursor do prompt na linha 2, coluna 20:

SetConsoleCursorPosition(prompt, coordenada);

//Troca a cor para escrever com letra amarela (se obtém juntando o vermelho e o verde)

SetConsoleTextAttribute(prompt, FOREGROUND_RED | FOREGROUND_GREEN);

printf("Texto amarelo");

coordenada.Y = 15;

//Posiciona o cursor do prompt na linha 15, coluna 20:

SetConsoleCursorPosition(prompt, coordenada);

//Troca a cor para escrever com letra branca (se obtém juntando o vermelho, o verde e o azul) e fundo roxo (se obtém juntando vermelho e azul)

SetConsoleTextAttribute(prompt, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE | BACKGROUND_RED | BACKGROUND_BLUE);

printf("Texto branco fundo roxo");