UCS UNIVERSIDADE DE CAXIAS DO SUL

# TDE B - Scramble

# Descrição do Trabalho

## 1. Introdução

O presente trabalho tem como objetivo desenvolver um programa em Assembly 8086 inspirado no jogo *Scramble*, desenvolvido pela Konami em 1981, conforme as especificações apresentadas neste documento.

O desenvolvimento deverá ser realizado em grupos de até dois integrantes. As duplas devem se inscrever previamente em um dos 15 grupos disponíveis na tarefa abaixo. Para participar desta atividade você precisa se inscrever em um grupo.

A apresentação do projeto ocorrerá no último dia de aula do semestre (11 de dezembro) e deverá ser realizada exclusivamente por meio do emulador GUI Turbo Assembler (Windows). Recomenda-se que o programa seja testado frequentemente nesse ambiente. Após a apresentação, as equipes estarão autorizadas a submeter a versão final do trabalho. **Submissões sem apresentação não serão avaliadas!**

## 2. Scramble

*Scramble* é um clássico jogo de arcade de tiro com nave e rolagem horizontal, desenvolvido pela Konami e lançado em 1981.

No jogo, o jogador controla uma nave (apelidada “Jet”) que avança automaticamente por um cenário contínuo, enfrentando obstáculos e inimigos. A nave dispõe de dois tipos de ataque: tiros para a frente e bombas para alvos terrestres. Um desafio extra é o consumo de combustível — se ele se esgotar, a nave cai. Para reabastecer, é necessário destruir tanques de combustível distribuídos no terreno.

O jogo é composto por seis fases distintas, cada uma com estilo de cenário e obstáculos próprios. Na última fase, o objetivo é destruir uma base inimiga. Uma vez destruída, aparece uma bandeira indicando a missão concluída, e o jogo recomeça com um nível de dificuldade maior.

*Scramble* foi um dos primeiros jogos a introduzir o conceito de rolagem forçada de cenário e etapas distintas, servindo de base para muitos shooters futuros.

Uma amostra do jogo pode ser encontrada no Scramble no YouTube .

# 3. Requisitos do Jogo

Para atender aos objetivos da disciplina e ao tempo disponível para a tarefa, o jogo passou por algumas adaptações em seu funcionamento. Assim, o jogo a ser implementado será uma versão reduzida do *Scramble* original. As principais adaptações são as seguintes:

- A nave (“jet”) contará com somente um tipo de ataque: disparos frontais.
- A mecânica de combustível não será implementada (e, consequentemente, não haverá reabastecimento).
- Foguetes terrestres não serão incluídos no jogo.
- O tempo de jogo será controlado por meio de um cronômetro.
- O jogo contará com três fases distintas.
- Não haverá base final na última fase.

Esta seção descreve os requisitos para o desenvolvimento do jogo.

## 3.1 Configuração do Vídeo

O programa deve utilizar o modo de vídeo 320 × 200 (código do modo de vídeo = 13h). Nesse modo, a tela é composta por 320 colunas e 200 linhas, conforme ilustrado na Figura 1. Cada pixel ocupa 1 byte na memória de vídeo, o que permite representar 256 cores distintas ($2^8$ = 256 combinações possíveis). Assim, a memória total utilizada pelo vídeo é de 320 × 200 = 64.000 bytes.

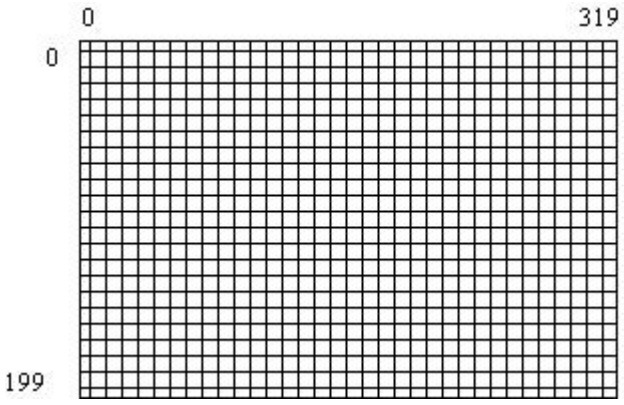


Figura 1. Estrutura do modo de vídeo 320 x 200

O modo 13h utiliza mapeamento de E/S em memória, o que significa que uma região da memória principal é destinada exclusivamente ao vídeo. Esse espaço de memória inicia no segmento A000h, onde cada posição representa um pixel na tela. Para desenhar um pixel, basta escrever o valor da

cor correspondente, conforme a Tabela 1, no endereço de memória que representa sua posição (coordenadas x, y). A posição de memória é calculada pela fórmula:

$$\text{endereço} = A000h : (x + y \times 320)$$

Por exemplo, para exibir um pixel na posição **(100, 10)** — ou seja, **coluna 100** e **linha 10** (ambas iniciando em 0) — o deslocamento é:

$$100 + 10 \times 320 = 0CE4h$$

Logo, o endereço completo será A000h:0CE4h.

| Hexadecimal | Cor | Hexadecimal | Cor |
|---|---|---|---|
| 0 | Preto | 8 | Cinza escuro |
| 1 | Azul | 9 | Azul claro |
| 2 | Verde | A | Verde claro |
| 3 | Ciano | B | Ciano claro |
| 4 | Vermelho | C | Vermelho claro |
| 5 | Magenta | D | Magenta claro |
| 6 | Marrom | E | Amarelo |
| 7 | Cinza claro | F | Branco |

**Tabela 1 - Código das cores.**

Embora seja possível utilizar a interrupção 10h (serviços de vídeo BIOS) para desenhar pixels, o método de endereçamento direto na memória de vídeo é consideravelmente mais eficiente. Isso ocorre porque o uso de interrupções envolve uma espera pelo processamento da CPU — uma espécie de "fila de atendimento" do sistema. Já o acesso direto à memória permite escrita imediata, sem dependência da BIOS. Além disso, podem ser utilizadas instruções de manipulação de strings (como `MOVSB`, `STOSB` ou `REP`) para acelerar a escrita de múltiplos pixels. Dessa forma, recomenda-se utilizar endereçamento indireto ou instruções de manipulação de strings para a escrita de pixels. O uso de interrupções deve ser reservado apenas para a escrita de texto ou operações não gráficas.

## 3.2 Tela Inicial

A tela inicial deve conter três elementos principais, conforme ilustrado na Figura 2:

- **Nome do jogo**: o nome "Scramble" deve ser exibido na parte superior da tela, utilizando arte em ASCII gerada pelo site patorjk.com/software/taag. O texto deve ser impresso na cor verde-claro (0Ah), garantindo boa legibilidade e destaque visual.
- **Elementos do Jogo**: entre o título e o menu, deve ocorrer uma animação cíclica dos elementos principais do jogo: nave aliada, meteoro e nave alienígena:
  - **Nave aliada**: move-se da esquerda para a direita, desaparecendo ao atingir o limite direito da tela e reaparecendo novamente à esquerda.

- **Meteoro**: realiza o movimento inverso, deslocando-se da direita para a esquerda, desaparecendo no limite esquerdo e reaparecendo no lado direito.
- **Nave alienígena**: deve iniciar em uma posição aleatória da tela, abaixo da trajetória do meteoro, deslocando-se inicialmente para a esquerda. Ao atingir o limite esquerdo, deve inverter o movimento e seguir para a direita.

- **Menu de opções**: O menu deve conter duas opções: **"Jogar"** e **"Sair"**, cada uma delimitada por uma **caixa** formada pelos seguintes caracteres ASCII:

```
218 ("┌")   191 ("┐")
192 ("└")   217 ("┘")
196 ("─")   179 ("│")
```

O usuário poderá navegar entre as opções utilizando as setas para cima e para baixo. A opção selecionada deve ser destacada em vermelho-claro (0Ch).As demais opções permanecem na cor branca (0F). Ao pressionar a tecla Enter, a ação correspondente à opção destacada será executada.


Figura 2 - Exemplo de tela inicial do jogo

## 3.3 Fase

O jogo será composto por três fases, cada uma introduzida por uma tela de apresentação exibida por 4 segundos (valor configurável), conforme ilustrado na Figura 3. Os textos de apresentação

devem ser exibidos em cores variadas, gerados por meio do gerador de arte em ASCII, garantindo uma identidade visual distinta para cada fase. Cada fase terá duração de 60 segundos, com o tempo restante exibido na barra de status. Esse tempo poderá ser ajustado por meio de uma constante pré-definida no código-fonte.


Figura 3 - Tela de apresentação da fase.

## 3.4 O Jogo

A tela do jogo é composta por três áreas principais: o **status do jogo** na parte superior, o **espaço do planeta** no centro, e a **superfície** do planeta na parte inferior. Cada um desses elementos será detalhado nas seções a seguir.

### 3.4.1. Status do Jogo

O status do jogo é composto pelo score, naves representando a quantidade de vidas e o tempo restante da fase, como ilustrado na Figura 4. Ele deve ser exibido no topo da tela, com o score alinhado à esquerda, naves representando a quantidade de vidas no centro e o tempo restante alinhado à direita. A dimensão das naves representando as vidas devem ter 7 pixels de altura e 19 pixels de largura. O tempo deve ser atualizado em intervalos de um segundo. Ao término do tempo, o jogo deverá exibir a tela da próxima fase e reiniciar o jogo, mantendo o score inalterado.


Figura 4 - Status do Jogo

A pontuação será explicada em cada fase.

### 3.4.2 Fase 1

Na Fase 1, a nave deve percorrer o trajeto sem colidir com a superfície ou nave alienígena. As naves alienígenas podem ser desctruídas com tiros.  Cada nave eliminada vale 100 pontos, e o jogador precisa se movimentar estrategicamente para maximizar a pontuação, e a cada segundo de jogo, a pontuação do jogador aumenta em 10 pontos, incentivando a permanência ativa na fase. Um exemplo do jogo na fase 1 é mostrado na Figura 5.

A **nave aliada** pode se deslocar para frente, para trás, para cima e para baixo utilizando as teclas de seta, e sua velocidade é maior do que a das naves alienígenas, permitindo manobras rápidas. Ela também pode atirar utilizando a barra de espaço, e cada tiro é representado por um pixel branco. Tanto a nave aliada quanto as naves alienígenas estão limitadas verticalmente pela barra de status no topo da tela e pela superfície do planeta abaixo, garantindo que não ultrapassem os limites visuais da fase. Se a nave aliada colidir com uma nave alienígena, o jogador perde uma vida. O número de vidas é limitado a três, e cada perda deve ser refletida na barra de status, removendo uma das naves que representam as vidas restantes.

As **naves alienígenas** se movem da direita para a esquerda da tela. Elas surgem em quantidade aleatória na vertical a partir da lateral direita da tela, nunca se sobrepondo, e respeitam os limites da barra de status e da superfície do planeta. Caso não sejam destruídas pelo jogador, desaparecem ao atingir o limite esquerdo da tela, mantendo a dinâmica contínua da fase.

O **planeta** apresenta uma superfície montanhosa, com desenho irregular que representa um terreno rochoso. Para transmitir a sensação de movimento, a superfície se desloca continuamente para a esquerda. Quando a superfície termina, ela deve reiniciar pelo início, garantindo que o início e o fim permitam uma transição contínua. Sua largura mínima é de 480 pixels, equivalente a 1,5 vezes a largura da tela, e a altura máxima é de 100 pixels, permitindo a criação de obstáculos que tornam a navegação mais estratégica.

A dimensão da nave aliada e das naves alienígenas é de 13 pixels de altura por 29 pixels de largura, proporcionando consistência visual e facilitando o cálculo de colisões e movimentação.


Figura 5 - Exemplo de tela da fase 1.

### 3.4.2 Fase 2

Na Fase 2, o jogador tem como objetivo desviar dos meteoros. Estes surgem em quantidade aleatória, nunca se sobrepondo, e respeitam os limites da barra de status e da superfície do planeta. Como os meteoros são indestrutíveis, deve haver uma distância mínima suficiente entre eles para que a nave aliada consiga passar, garantindo pelo menos 5 pixels de espaço além da altura da nave. Os meteoros se movem da direita para a esquerda da tela e, caso não sejam desviados pelo jogador, desaparecem ao atingir o limite esquerdo, mantendo a dinâmica contínua da fase.

Se a nave aliada colidir com um meteoro, o jogador perde uma vida. A fase apresenta um terreno rochoso diferente, com dimensões similares às da Fase 1, e a superfície se move continuamente para a esquerda, da mesma forma que na Fase 1, garantindo transição contínua. A dimensão da nave aliada e dos meteoros segue o mesmo padrão da fase anterior: 13 pixels de altura por 29 pixels de largura.

A cada segundo de permanência na fase, o jogador acumula 15 pontos, incentivando a navegação contínua e cuidadosa.

Um exemplo do jogo na Fase 2 é mostrado na Figura 6.

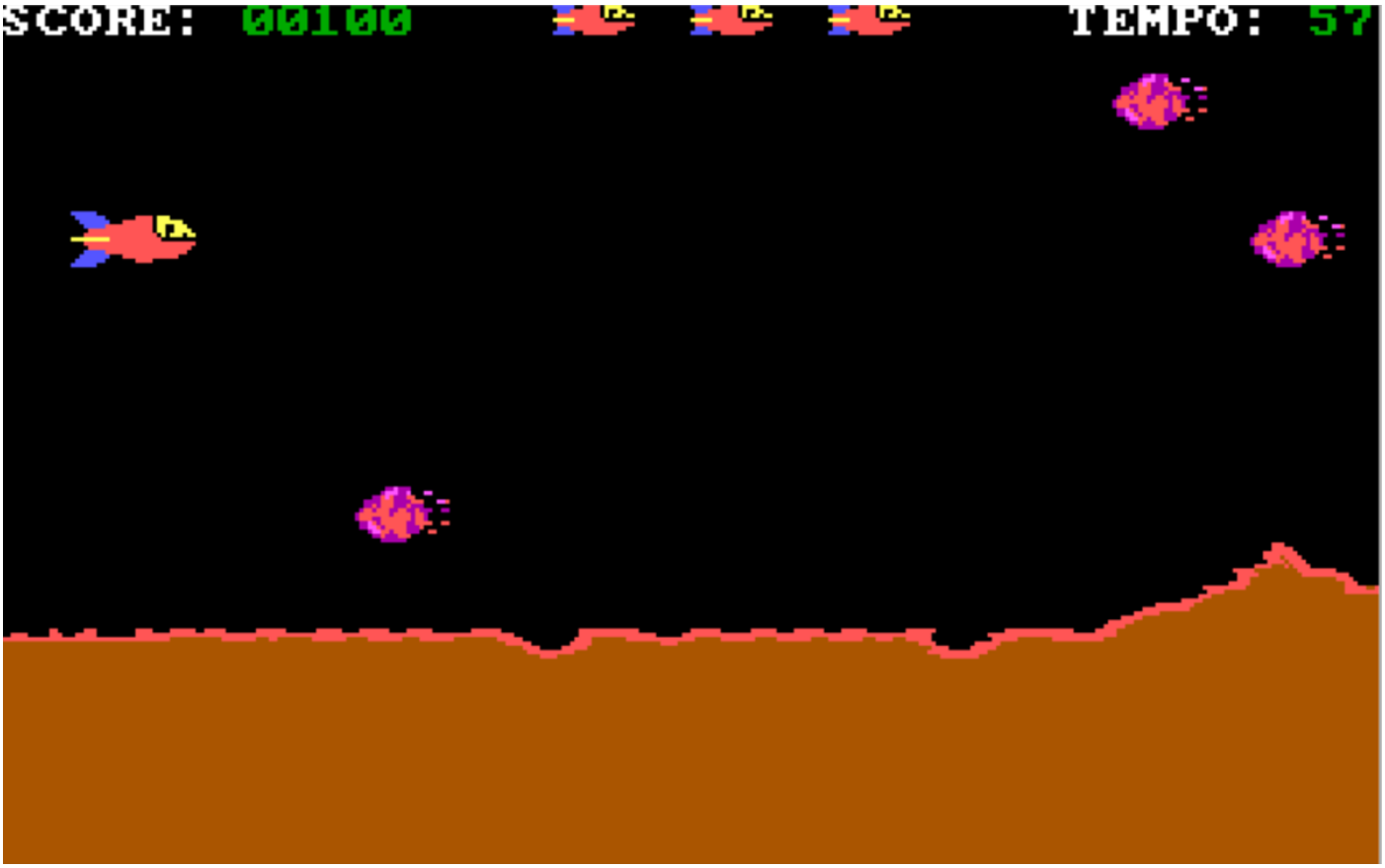


Figura 6 - Exemplo de tela da fase 2.

### 3.4.3 Fase 3

Na Fase 3, o objetivo é iagual ao da fase 1, mas as naves alienígenas possuem uma velocidade maior e o espaço vertical de voo é mais estreito. Estes surgem em quantidade aleatória, nunca se sobrepondo, e respeitam os limites da barra de status e da superfície do planeta. As mesmas regras sobre o comportamento das naves se aplicam nesta fase. Cada nave eliminada vale 150 pontos e a cada segundo de permanência na fase, o jogador acumula 20 pontos.

Um exemplo do jogo na Fase 3 é mostrado na Figura 8.

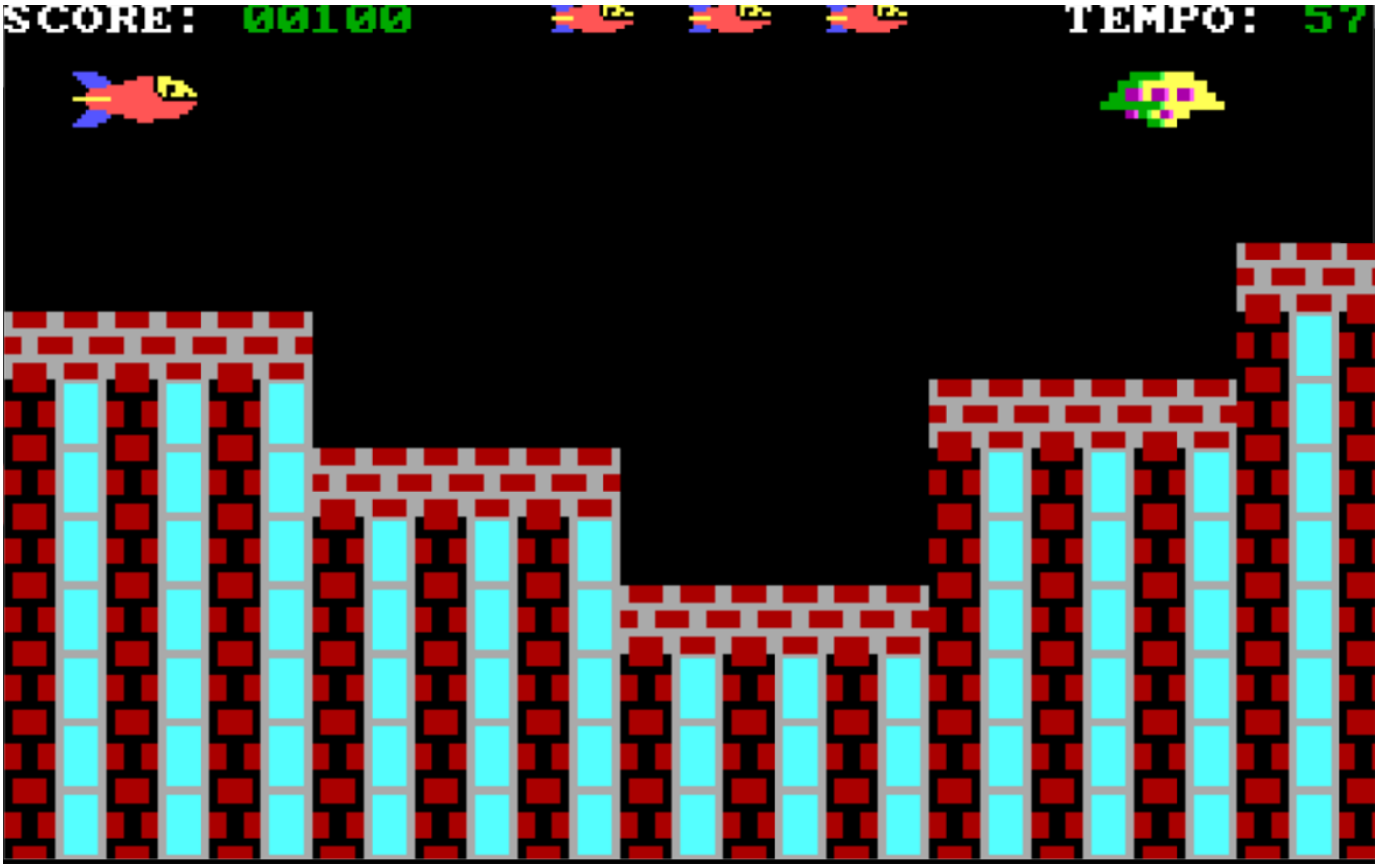


Figura 7 - Exemplo de tela da fase 3

A superfície do planeta é formada por plataformas de tijolos e colunas, que se repetem de forma contínua, com diferentes alturas. Cada plataforma é composta por 2 a 4 colunas, conforme o modelo apresentado na Figura 8, com dimensões de 16 pixels de altura por 24 pixels de largura. Cada bloco, ao ser posicionado sobre outro bloco, mantém a continuidade visual do terreno, criando uma aparência uniforme e sem quebras na superfície.

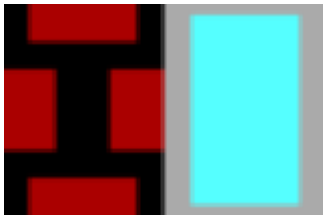

Figura 8 - Coluna

A coluna é finalizada com um bloco de tijolos, conforme ilustrado na Figura 9, cujas dimensões são iguais as da coluna.

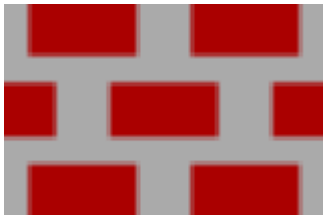

Figura 9 - Topo

### 3.5 Tela de Fim de Jogo

A tela de fim de jogo pode ser exibida em duas situações distintas:

- Perda de todas as vidas: nesse caso, a tela apresenta a mensagem “GAME OVER”, centralizada em duas linhas, na cor vermelha, gerada por meio de arte em ASCII.
- Conclusão das três fases: quando o jogador completa todas as fases, é exibida a mensagem “VENCEDOR!”, na cor verde, também utilizando arte em ASCII. Abaixo da mensagem, o score final é mostrado em branco.

Em ambos os casos, o jogador deve pressionar qualquer tecla para retornar à tela inicial.

## 4. Restrições do Trabalho

- **Impressão e Ocultação**: devem ser realizadas exclusivamente por meio de endereçamento indireto ou instruções de manipulação de strings, garantindo eficiência e consistência na renderização dos elementos gráficos (naves, terreno e tiros).
- **Evitar Duplicação de Rotinas**: o código deve ser estruturado para evitar redundâncias. Sempre que possível, deve-se implementar rotinas genéricas reutilizáveis. Por exemplo, uma única rotina de movimentação pode atender a diferentes elementos do jogo — como naves alienígenas e nave aliada — eliminando a necessidade de funções separadas para cada tipo.
- **Configuração de Tempo**: os intervalos de movimentação dos elementos e a duração de cada fase devem ser parametrizáveis, permitindo ajustes de desempenho e dificuldade sem necessidade de alterar o código principal.
- **Compatibilidade com o Ambiente de Montagem**: o programa deve ser plenamente compatível com os montadores GUI Turbo Assembler, assegurando comportamento idêntico em ambas as plataformas.

## 5. Mínimo Entregável

Para garantir um padrão mínimo de implementação, cada trabalho deve atender e operar corretamente de acordo com os seguintes requisitos:

- **Tela Inicial**: todos os elementos devem estar visíveis e funcionais, incluindo a movimentação dos componentes do jogo e a operação completa do menu de opções.
- **Jogo**: após a tela de apresentação do setor, o jogo deve exibir todos os elementos essenciais — status, naves e superfície. A nave aliada (jet) deve mover-se vertical e horizontalmente por meio das setas do teclado. As naves alienígenas devem surgir de forma aleatória e deslocar-se para a esquerda. A superfície deve estar visível, sendo a movimentação opcional, mas a construção do terreno da fase 3 deve estar implementada e funcional. A implementação do

tiro e da detecção de colisão é opcional. O temporizador de 60 segundos para a mudança de setor deve estar ativo e funcional, promovendo a transição automática ao término do tempo. O score deve estar funcional.

- **Fim de Jogo**: o sistema deve detectar corretamente o encerramento da partida quando o jogador completar os três setores ou perder todas as vidas.
- **Código**: o código-fonte deve estar claramente documentado por meio de comentários explicativos. Cada rotina deve ser precedida por uma descrição que contenha:
  - Função: breve explicação do propósito da rotina;
  - Parâmetros de entrada: identificação e descrição de cada parâmetro utilizado;
  - Parâmetros de saída: valores ou resultados retornados pela rotina.

Todos os itens acima devem estar implementados e operacionais. Caso contrário, o projeto não poderá ser avaliado. Elementos não descritos ou definidos como opcionais, bem como o relatório do projeto, também serão considerados para pontuação complementar.

# 6 . Entrega

O código-fonte do programa deve ser submetido somente após a apresentação do trabalho. Projetos não apresentados não serão avaliados nem considerados para pontuação.

# 7. Dicas

Esta seção tem por objetivo passar algumas dicas que visam facilitar a implementação do trabalho.

# 7.1. Escrita de Strings com Cor

Para escrever uma string, pode-se utilizar a função 13H da interrupção 10H:

```
AH = 13h
AL = modo
      bit 0: 1- atualiza a posição do cursor após a escrita
      bit 1: 1- string contém caracteres e atributos
BL = atributo do caractere se o bit 1 em AL for 0
BH = número da página de vídeo
DH,DL = linha, coluna da posição de impressão
CX = tamanho, em caracteres, da string
ES:BP = endereço do início da string
```

Nota: esta interrupção reconhece os códigos CR, LF, e backspace.

## 7.2. Suspensão da Execução por um Determinado Intervalo de Tempo

A interrupção 15h oferece uma variedade de serviços, incluindo a função AH=86H, que permite suspender a execução do programa por um intervalo de tempo especificado. O intervalo é definido

em microssegundos (1/1.000.000 de segundo) e é especificado pelos registradores CX e DX, onde CX contém os 16 bits mais significativos e DX contém os 16 bits menos significativos.

Para converter o intervalo de tempo desejado em microssegundos, é necessário configurar CX e DX de acordo. Por exemplo, para um intervalo de 2 segundos (2.000.000 microssegundos), CX deve ser configurado com o valor 1EH e DX com o valor 8480H, pois 1E8480H é a representação hexadecimal de 2.000.000.

## 7.3. Leitura de Teclado sem Aguardar

A interrupção 16h oferece várias funções relacionadas ao teclado. Para ler uma tecla sem que o programa precise esperar, você pode usar a interrupção 16h para verificar o status do teclado. Especificamente, o serviço AH=01H permite verificar se há teclas no buffer do teclado. Se o buffer estiver vazio, o Zero Flag (ZF) será setado.

Para ler uma tecla do buffer, você deve utilizar o serviço AH=00H da mesma interrupção. Este serviço não apenas retorna o código da tecla pressionada, mas também remove a tecla do buffer.

## 7.4. Definir constante com o tamanho da memória

No Assembly, o símbolo $ é uma referência ao deslocamento atual (ou posição atual) dentro de um segmento de código ou de dados. Ele é útil principalmente para cálculos que envolvem endereços e offsets durante o tempo de montagem. O valor de $ pode ser usado para calcular comprimentos ou distâncias entre diferentes partes do código ou dados, principalmente em tabelas, blocos de memória ou instruções que requerem referência a endereços relativos.

Imagine que você está criando um buffer de dados e quer armazenar o tamanho do buffer em outro local. Você pode usar $ para calcular o tamanho do buffer diretamente no momento da montagem:

```
buffer db "Exemplo de texto"
tamanho equ $ - buffer;
```

Neste exemplo:

- buffer é o rótulo que marca o início dos dados.
- O valor de $ - buffer calcula o número de bytes entre o deslocamento atual (posição $) e o início de buffer. Assim, o valor de tamanho armazenará o tamanho de buffer de forma automática.