Space Invaders

Eliane Isadora Faveron Maciel

[1]Centro de Ciências Exatas e Tecnologias (CCET) – Universidade de Caxias do Sul (UCS)
Rua Francisco Getúlio Vargas, 1130 – 95.070-560 – Caxias do Sul – RS – Brasil

{eifmaciel@ucs.br}

1. Problema

Desenvolver um programa em *Assembly* do 8086 que implemente o jogo *Space Invaders*. O objetivo do jogador é defender seu planeta natal de uma invasão de seres alienígenas. O jogador deve destruir as aeronaves inimigas.

2. Solução

O jogo implementado possui 6 modelos de aeronaves inimigas, estas estão dispostas em matriz, sendo que cada linha possui 10 aeronaves de cada modelo. Na execução do jogo as aeronaves inimigas se movimentam lateralmente e ao atingir uma lateral movimentam-se verticalmente.O jogador pode disparar um tiro por vez, se alguma aeronave for atingida, ela some da tela. As naves inimigas também atiram contra o jogador, uma de cada vez, aleatoriamente. Se o jogador for atingido perde vida, sendo 3 vidas no total, ao zerar as vidas o jogo se encerra e o jogador perde. Quando as naves inimigas atingem o solo o jogador perde. O jogador vence apenas quando consegue atingir todas as aeronaves inimigas.

O programa inicia com a configuração para vídeo. Em seguida é apresentado a Tela Inicial, pode ser observada na figura 1, possui os itens de menu, Recordes, Jogar e Sair, o programa fica na espera de um *click* de uma tecla. Ao clicar “j”, é apresentado a Tela jogo, ao clicar “r” é apresentado a Tela Recordes, caso seja clicado outras teclas o programa finaliza a execução.

Figura 1. Tela Inicial

Tela Recordes apresenta as últimas pontuações vencedoras, estas são guardadas em memória. Podemos ver a tela na figura 2.

Figura 2. Tela Recordes

A tela de jogo possui os Scores que é a pontuação do jogo e é atualizado a cada nova pontuação, nesta tela possui também a marcação de vidas. Esta tela é atualizada

constantemente, alterando as posições das aeronaves inimigas, nave amiga pode ser movimentada usando as teclas de setas do teclado.

O Jogador pode também atirar contra as aeronaves inimigas, pressionando a tecla ”espaço”. Podemos ver a organização desta tela na figura 3.

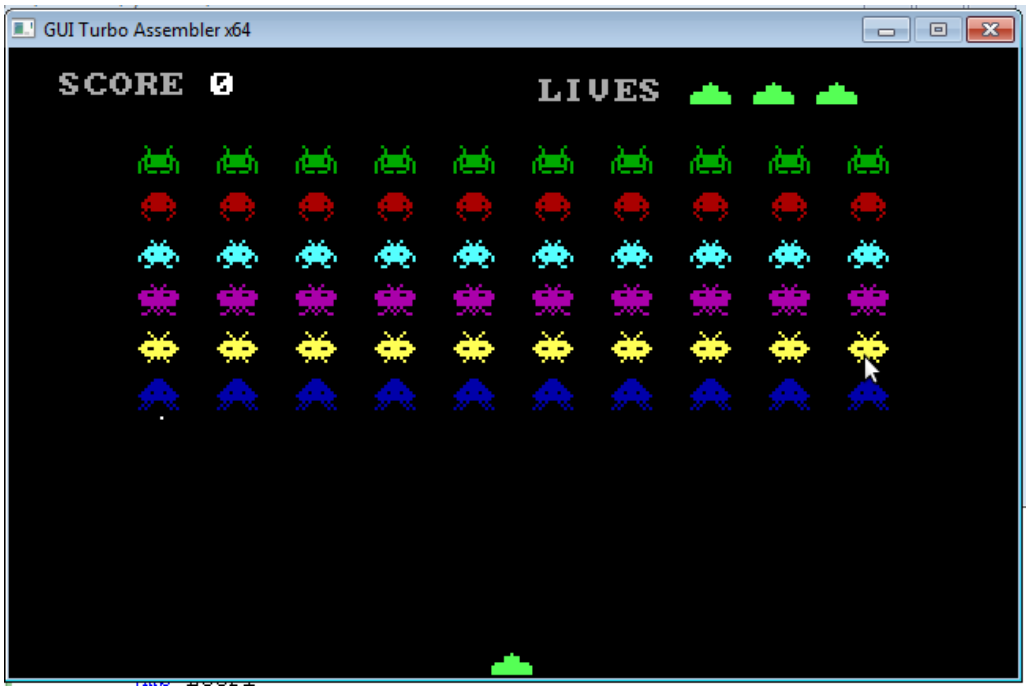

Figura 3. Tela de Jogo

O programa possui a Tela Final para quando o jogador perde e a Tela Final para quando o jogador ganha, nestas telas é apresentado a pontuação final, conforme pode ser observado na figura 4.

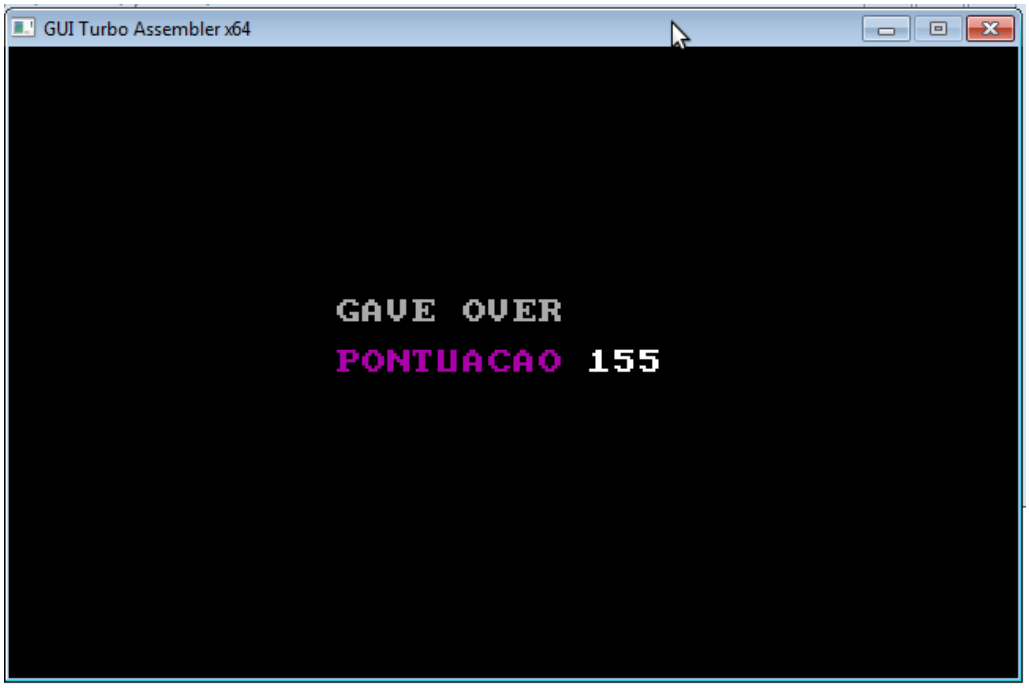

Figura 4. Tela Final

2.1. Algoritmo

O algoritmo principal do Jogo é feito pela *proc* GAME, código pode ser visto abaixo. Inicialmente é setado o valor das "variáveis" que guardam a posição x e y das aeronaves inimigas, e o jogo possui um *loop* onde é feito movimento das naves, limpa a tela e desenha a tela, o programa fica a "espera" de um *click* no teclado para movimentar a aeronave amiga ou disparar um tiro contra as naves inimigas, a cada duas repetições do *loop* é chamado a *proc* que faz a movimentação das naves alienígenas. Verifica-se as vidas e o "fim perdeu", se não possui mais vidas ou as naves inimigas atingem o "solo" , o *loop* se encerra e mostra a tela de fim de jogo. É comparado também se o jogador ganhou acertando todas as aeronaves, se sim mostra a tela de fim de jogo e armazena o recorde. Caso não entre nestas condições o loop se repete.

```
GAME proc
    EMPILHATUDO

    mov [pos_y], 30
    mov [pos_x], 40
    call LIMPA_TELA
    call TELA_JOGO
    mov BX, 0
LOOP1:
    call DELAY
    call LIMPA_TELA
    call TELA_JOGO

    call LE_TECLADO
```

```
    cmp BX, 2
    jne CONTINUE
    mov BX, 0
    call MOVE_ALIENS

CONTINUE:
    inc BX

    cmp [vidas], 0
    jz PERDEU1
    cmp fim_perdeu, 1
    je PERDEU1
    jmp LOOP1

GANHOU:
    call LIMPA_TELA
    call TELA_FINAL_GANHOU
    xor AX, AX
    int 16h
    jmp END_GAME

PERDEU1:
    call LIMPA_TELA
    call TELA_FINAL
    xor AX, AX
    int 16h

    cmp ax, 6A00h

END_GAME:
    DESEMPILHATUDO
    ret
endp
```

O desenho da Tela de Jogo é feito pela *proc* TELA JOGO. Ao iniciar a *proc* é chamada outra *proc* desenhar a TELA SCORS, está tela fica sempre na mesma posição e altera os valores da pontuação e vidas do jogador. Após isso, começa a desenhar as aeronaves inimigas, das superiores de maior pontuação às inferiores, é feito um *loop* por linha, sendo cada linha um tipo de nave. Inicia-se atualizando o valor de cx(coluna x) e dx (linha y). Verifica-se se dx já está no "solo" se sim, será ativado o fim do programa.Caso não é verificado se a nave está "viva" move-se a matriz da estrutura da nave para o registrador e desenha ela, após isso, é atualizado as posições de cx e dx. Verifica-se se há uma colisão com um tiro, se há a nave ficará "morta" e não aparecerá mais na tela, e soma a pontuação conforme valores salvos em memória. Verifica-se também se o tiro inimigo, é ativado. O procedimento se repete para as seis linhas da matriz de naves. Após isso é desenhado o tiro inimigo, o tiro amigo (se houver) e a nave amiga. Os desenhos obedecem sempre a mesma lógica, atualizando o cx e dx e escrevendo o pixel na tela. O código pode ser visto abaixo

```
TELA_JOGO proc
    EMPILHATUDO
    xor cx, cx
    xor dx, dx
    xor DI, DI

    call TELA_SCORS

    mov cx, [pos_x]
    mov bx, loop_nav
    mov DI, offset naveAtiva6 ; Nave
    mov dx, [pos_y] ; Posicao da linha no eixo y
    call FIM_PERDEU_PROC ; Verfica colisão com o solo

LINHA1:
    cmp [DI], 00
    je CONTINUA1

    mov SI, offset NAVE_6 ; Nave
    call DESENHA_NAVE_AMIGA ; Desenha a nave na posicao (dx, cx)
    mov ax, 6
    ; Verifica se foi atingida pelo tiro, se foi o [DI] é setado em 0
    call VERIFICA
    call TIRO_INIMIGO_TESTE
CONTINUA1:
    inc DI
    dec bx
    add cx, 25
    cmp bx, 0
    jnz LINHA1

    ; segunda linha de naves
    mov cx, [pos_x]
    mov bx, loop_nav
    add dx, 15
    mov DI, offset naveAtiva5 ; Nave
    call FIM_PERDEU_PROC ; Verfica colisão com o solo
LINHA2:
    cmp [DI], 00
    je CONTINUA2
    mov SI, offset NAVE_5
    call DESENHA_NAVE_AMIGA
    mov ax, 5
    ; Verifica se foi atingida pelo tiro, se foi o [DI] é setado em 0
    call VERIFICA
    call TIRO_INIMIGO_TESTE
CONTINUA2:
    inc DI
    dec bx
```

```
    add cx, 25
    cmp bx, 0
    jnz LINHA2

; terceira linha de naves
    mov cx, [pos_x]
    mov bx, loop_nav
    add dx, 15
    mov DI, offset naveAtiva4 ; Nave
    call FIM_PERDEU_PROC ; Verfica colisão com o solo
LINHA3:
    cmp [DI], 00
    je CONTINUA3
    mov SI, offset NAVE_4
    call DESENHA_NAVE_AMIGA
    mov ax, 4
    ; Verifica se foi atingida pelo tiro, se foi o [DI] é setado em 0
    call VERIFICA
    call TIRO_INIMIGO_TESTE
CONTINUA3:
    inc DI
    dec bx
    add cx, 25
    cmp bx, 0
    jnz LINHA3

; Quarta linha de naves
    mov cx, [pos_x]
    mov bx, loop_nav
    add dx, 15
    mov DI, offset naveAtiva3 ; Nave
    call FIM_PERDEU_PROC ; Verfica colisão com o solo
LINHA4:
    cmp [DI], 00
    je CONTINUA4
    mov SI, offset NAVE_3
    call DESENHA_NAVE_AMIGA
    mov ax, 3
    ; Verifica se foi atingida pelo tiro, se foi o [DI] é setado em 0
    call VERIFICA
    call TIRO_INIMIGO_TESTE
CONTINUA4:
    inc DI
    dec bx
    add cx, 25
    cmp bx, 0
    jnz LINHA4

; Quinta linha de naves
```

```
    mov cx, [pos_x]
    mov bx, loop_nav
    add dx, 15
    mov DI, offset naveAtiva2 ; Nave
    call FIM_PERDEU_PROC ; Verfica colisão com o solo
LINHA5:
    cmp [DI], 00
    je CONTINUA5
    mov SI, offset NAVE_2
    call DESENHA_NAVE_AMIGA
    mov ax, 2
    ; Verifica se foi atingida pelo tiro, se foi o [DI] é setado em 0
    call VERIFICA
    call TIRO_INIMIGO_TESTE
CONTINUA5:
    inc DI
    dec bx
    add cx, 25
    cmp bx, 0
    jnz LINHA5

; ULtima linha de naves
    mov cx, [pos_x]
    mov bx, loop_nav
    add dx, 15
    mov DI, offset naveAtiva1 ; Nave
    call FIM_PERDEU_PROC ; Verfica colisão com o solo

LINHA6:
    cmp [DI], 00
    je CONTINUA6
    mov SI, offset NAVE_1
    call DESENHA_NAVE_AMIGA
    mov ax, 1
    ; Verifica se foi atingida pelo tiro, se foi o [DI] é setado em 0
    call VERIFICA
    call TIRO_INIMIGO_TESTE
CONTINUA6:
    inc DI
    dec bx
    add cx, 25
    cmp bx, 0
    jnz LINHA6

    call DESENHA_TIRO_INIMIGO
    call DESENHAR_TIRO
    xor cx, cx
    xor dx, dx
    mov cx, pos_x_naveamiga
```

```
    mov dx, pos_y_naveamiga
    mov SI, offset NAVE_AMIGA
    call DESENHA_NAVE_AMIGA
ACABA:
    DESEMPILHATUDO
    ret
endp
```

2.2. Variáveis

O blocos de memória principal são as matrizes com os desenhos das aeronaves, existe um bloco para cada tipo de aeronave de tamanho 15 x 10 onde cada byte representa a cor do pixel. O código pode ser visto abaixo.

```
NAVE_AMIGA  db 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
            db 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
            db 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
            db 00, 00, 00, 00, 00, 00, 00, 10, 00, 00, 00, 00, 00, 00, 00
            db 00, 00, 00, 00, 00, 10, 10, 10, 10, 10, 00, 00, 00, 00, 00
            db 00, 00, 00, 00, 00, 10, 10, 10, 10, 10, 00, 00, 00, 00, 00
            db 00, 00, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 00, 00
            db 00, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 00
            db 00, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 00
            db 00, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 00

NAVE_1      db 00, 00, 00, 00, 00, 00, 01, 01, 01, 00, 00, 00, 00, 00, 00
            db 00, 00, 00, 00, 00, 01, 01, 01, 01, 01, 00, 00, 00, 00, 00
            db 00, 00, 00, 00, 01, 01, 01, 01, 01, 01, 01, 00, 00, 00, 00
            db 00, 00, 00, 01, 01, 01, 01, 01, 01, 01, 01, 01, 00, 00, 00
            db 00, 00, 01, 01, 01, 00, 01, 01, 01, 00, 01, 01, 01, 00, 00
            db 00, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 00
            db 00, 00, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 00, 00
            db 00, 00, 00, 01, 00, 01, 00, 00, 00, 01, 00, 01, 00, 00, 00
            db 00, 00, 01, 00, 01, 00, 00, 00, 00, 00, 01, 00, 01, 00, 00
            db 00, 01, 00, 01, 00, 00, 00, 00, 00, 00, 00, 01, 00, 01, 00
```

É guardado em memória também os valores de cada nave, a pontuação, os recordes e as strings que são usadas nas telas.Abaixo as demais variáveis e suas descrições :

- TIRO SYMBOL - Pixel da cor do tiro;
- tiro - Guarda se tem tiro ativo.
- tiro inimigo - Guarda se tem tiro inimigo ativo.
- 'pos tiro x1' , 'pos tiro y1' - Guarda a posição do tiro amigo, coluna e linha.
- 'pos tiro inimigo x', 'pos tiro inimigo y' - Guarda a posição do tiro inimigo, coluna e linha.
- 'nave atira', Guarda qual nave que atira,

- 'cont tiro x' Guarda qual a linha que atira.
- 'naves count' - Total de naves
- naveAtiva1, naveAtiva2, naveAtiva3, naveAtiva4, naveAtiva5, naveAtiva6 - Vetor de flags, que guardam se a nave inimiga está viva.
- 'pos x', 'pos y' Guarda a posição das naves inimigas.
- 'pos x naveamiga', 'pos y naveamiga' - Guarda a posição das nave amiga.

2.3. Descrição das Procs

O programa foi desenvolvido fazendo sempre a divisão de funcionalidades em *procs*, que servem como funções e são chamadas sempre que necessário. Em cada *proc* a uma chamada para a macro que Empilha e Desempilha todos os registradores. Abaixo segue a descrição das *procs* exceto das *procs* principais Game e Tela Jogo que já foram descritas acima.

- VIDEO MODE - Seta com a interrupção 10h e al= 13h, o modo de vídeo. É feito apenas no inicio do programa.
- ESC CHAR - Escreve um carácter, int 21h ah = 2;
- LIMPA TELA - Faz um *scroll* da tela, AH= 07h int 10h
- DESENHA NAVE AMIGA - Desenha cada pixel da nave que está na matriz. Usa ah = 0Ch int 10h
- DESENHA TIRO - Desenha o pixel de tiro, ah = 0Ch int 10h
- FIM PERDEU PROC - Verifica se as naves inimigas atingiram o limite da tela, e retorna no 'fim perdeu'.
- DESENHAR TIRO, DESENHA TIRO INIMIGO - Posiciona o tiro, se atingir um limite remove o tiro.
- VERIFICA COLISÃO - Verifica se o tiro atingiu a nave amiga, se atingiu decrementa as vidas.
- VERIFICA - Verifica se o tiro da nave amiga atingiu alguma base da nave inimiga, se atingiu insere 0 no vetor de flags.
- CALCULA PONTOS - É chamando quando a proc verifica detecta colisão de uma nave, é feito vários desvios condicionais para incrementar o valor da pontuação.
- TIRO INIMIGO TESTE - Ativa o tiro inimigo e atualiza a posição do tiro.
- LE TECLADO - Lê uma tecla do teclado, verifica se a tecla clicada movimenta a nave ou para atirar. Usa AH = 01h da int 16h.
- DELAY - Faz o processador "ficar ocupado"por um tempo.
- MOVE ALIENS - Verifica a posição das aeronaves e atualiza as posições.
- ESC UINT16 - Para escrever um inteiro.

Nas *procs* de Telas o funcionamento é o mesmo para todas, é posicionado o local da *string* nos registradores, carrega o valor da *string* no registrador, e manda escrever, são

elas:

- ESCREVE STRING - Utilizada para escrever as strings de títulos das telas. Usa ah = 13h da int 10h.
- TELA INICIAL - *Proc* que prepara a Tela inicial do programa.
- TELA FINAL - *Proc* que prepara a Tela final do programa, quando o jogador perde.
- TELA FINAL GANHOU - *Proc* que prepara a Tela final do programa quando o jogador ganha.
- TELA RECORDES - *Proc* que prepara a Tela de Recordes do programa.
- TELA SCORS - *Proc* que prepara a tela de scores está que fica posicionada na tela de jogo, e é chamada sempre na tela de jogo.

Podemos analisar no fluxograma os detalhes do algoritmo, figura 5.

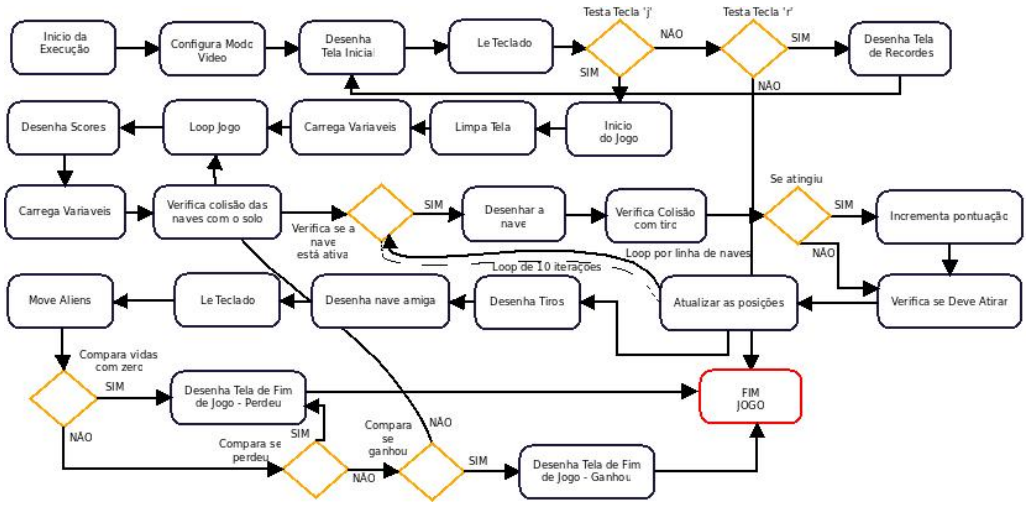

Figura 5. Fluxograma

3. Conclusão

O trabalho teve um grau elevado de dificuldade devido a linguagem de programação utilizada, e ao fato de este método de desenvolvimento não ser tão utilizado. Posso dizer que dentre todas as dificuldades encontradas, a maior foi a parte de desenho da tela de jogo, e funções de tiro. Além disso, tive uma demora para o entendimento do funcionamento da linguagem e adaptação para o problema.

Com este trabalho pude entender melhor o funcionamento memória, registradores e a linguagem *assembly*, assim como auxílio no entendimento para as demais linguagens de programação. O programa feito pode ser melhorado ou adaptado para formas melhores de programação.