INTRODUÇÃO À ARQUITETURA DE COMPUTADORES

LEIC

IST-TAGUSPARK

RELATÓRIO DO PROJETO

1. Introdução

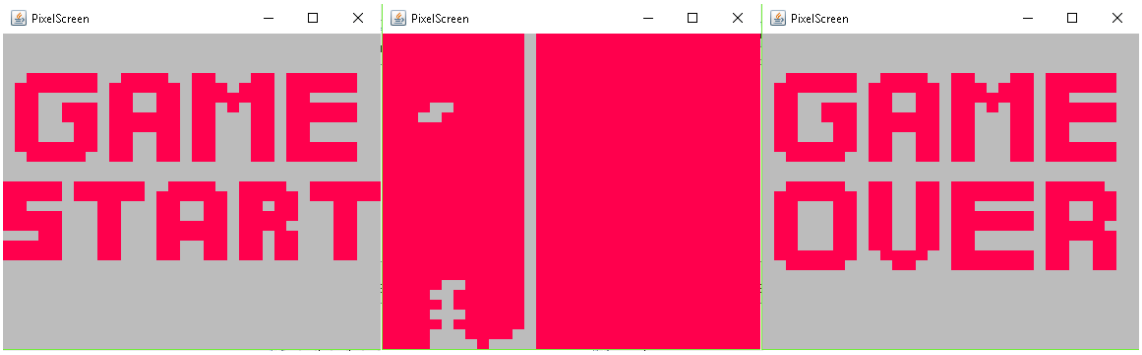
O Trabalho consiste num jogo clássico de Tetris com a diferença que existe um inimigo com o qual o jogador tem de se preocupar, vamos dar-lhe o nome de monstro. A meio do jogo normal de Tetris, irá aparecer o monstro do lado esquerdo do PixelScreen, tendo o jogador de impedir que este colida com as peças atualmente presentes no tabuleiro ou com o extremo direito do PixelScreen. Para eliminar o monstro, o jogador deve fazer colidir com o seu tetraminó a parte de cima do monstro. Objetivo é alcançar os máximos pontos possíveis antes que o monstro colida com os tetraminó ou com a parede esquerda do ecrã. Com cada monstro destruído o jogador ganha pontos com cada linha completa e por cada monstro destruído.

1.1. Convenções de registos

Do R0 a R7 são registos dedicados a cálculos e manipulações de informação, R8 a R10 são registosque terão valores não usados para cálculos. No âmbito deste projeto dedicou-se o R10 para ter o valor de posição do objeto a desenhar e R8 e R9 para parâmetros de rotinas, quer seja de input ou de output, só é usado R9 no caso em que será necessário mais que 1 registo, nesta entrega não foi necessária.

1.2. Estados de jogo

O jogo inclui ecrãs de game start e game over, como também um modo de pausa.



2. Conceção e Implementação

2.1. Gerador

Primeira instrução do gerador é o contador que será usado como parâmetro de aleatoriedade. Tem 2 funcionalidades, criar peças novas, inicialização de novo jogo. Também é usado um contador de peças em conjunto com o contador para decidir se o monstro é para ser criado. Só quando o bit de menor peso de ambos for igual a 1 que é activada a flag do monstro.

```
gerador:
        PUSH R0
        …
|       MOV R0, contador                    ; incrementacao do contador (RNG)
|       MOVB R1, [R0]                       ; que ira sempre correr durante o jogo
|       ADD R1, 1                           ;
        MOVB [R0], R1                       ;
        MOV R0, inicio_jogo                 ; verificacao da flag, inicio_jogo…
        MOVB R1, [R0]                       ;
        CMP R1, ACTIVO                      ; se o inicio_jogo estiver ativo, segue…
        JNZ criacao_peca
        …       Verificação de fim de jogo
|       criacao_peca:                              ; esta funcao verifica
|       MOV R0, criar_peca                  ; se no sitio onde
|       MOVB R1, [R0]                       ; se pretende
|       CMP R1, ACTIVO                      ; pintar a nova
|       JNZ nao_criar_peca                  ; peca ja tem algo
|       …       Activação de flags
|       CALL set_peca                       ; Instrução que preenche as words
        …       Verificação de poder criar a peça
nao_criar_peca:
        …
        POP R2
        …
        RET
```

2.1.1. Criar peça “set_peca”

Se a flag “criar_peca” estiver activa então irá criar uma peça na posição inicial defenida por uma constante.

Chama a rotina set_peca que vai buscar a tabela de tetraminós “*Mãe*” qual o tetraminó a ser criado.

```
tetra:         ;Tabela que contem todos os tetraminos possiveis
        WORD tetra_t
        WORD tetra_s
        WORD tetra_o
        WORD tetra_l
        WORD tetra_j
        WORD tetra_i
```

Adicionando a etiqueta os 3 bits de maior peso do contador, como o tamanho da tabela é de 5 elementos e o máximo valor é de 7, dividimos o valor por 4 e o resto dessa divisão será o índice da tabela, selecionando um dos primeiros valores, isto tem implicação que as peças no inicio da tabela tem maior chance de serem escolhidas.

```
set_peca:
        PUSH R0
        …
        MOV R0, tetra                       ;tetraminos,
        CMP R1, SIZE_TABELA_TETRA     ;caso o valor seja superior a 6
        JN set_peca_negativo          ;vamos aproveitar o seu bit de menor peso
        SHR R1, 2
set_peca_negativo:
        MOV R2, 2                                   ;Como e por words
        MUL R1, R2                                  ;vamos multiplicar o R1 para ter
indice par
        ADD R0, R1                                  ;aplicacao de indice
        MOV R1, [R0]
        …
RET
```

Escolhido o tetraminó teremos que preencher 2 words, para que as futuras rotinas saibam qual o tetraminó a ser movido e qual as suas transformações, também alteramos a string que serve como apontador da tabela de transformações (para a funcionalidade de rodar a peça).

```
        MOV R0, TETRATBL      ;Indicador da tabela de transformações da peca actual
        MOV [R0], R1
        MOV R2, [R1]          ;Vamos buscar a posicao inicial
        MOV R0, TETRASTR
        MOV [R0],R2           ;e vamos preencher essa peca
        MOV R0, TETRAROT      ;reiniciar o apontador da tabela de transformações
        MOV R1, 0
        MOVB [R0], R1
```

2.1.2. Inicio jogo (flag)

Quando esta é igual a:

- 1, significa que vamos fazer reset ao ecrã reinicializar os pontos e vamos activar a flag de criar_peca que pertence a própria rotina
- 0, quer dizer que estamos em jogo
- 2, significa que estamos em modo de GAME OVER e vamos pintar isso no ecrã

```
        MOV R0, inicio_jogo           ;Verificacao da flag
        MOVB R1, [R0]
        CMP R1, ACTIVO ;Activo é uma EQU que tem valor 1H
        JNZ game_over_gerador ;verificação se estamos em game over
inicio_jogo_gerador:
        CALL clear_pixel_screen
        MOV R8, criar_peca
        CALL activar_flag
        MOV R8, inicio_jogo
        CALL flip_flag
game_over_gerador:
        CMP R1, GAME_OVER
        JZ nao_criar_peca;Não criamos peça independentemente do valor da flag criar_peca.
```

2.2. Leitura de teclado “keyboardcheck”

Lê o input do teclado e guarda a ultima tecla pressa em memória, tendo um modo em “*Idle*” guardando em memória a tecla FFH se nenhuma tecla estiver a ser pressa.

```
keyboardcheck:                              ;Função que verifica todos os botoes
        PUSH R1
        …
        MOV R1,         LINHA
        MOV R8,         0
        MOV R7,         0
nextline:                                           ;Cico para correr todas as linhas

        MOV R2, POUT
        MOVB [R2], R1                       ;Envia codigo R1 para as linhas do teclado
        MOV R2, PIN
        MOVB R3, [R2]                       ;R3 recebe resposta do teclado que varia
entre 8H a 1H
        MOV R2, mask_3_0                    ;Mascara que isola os bits de 0 a 3
        AND R3, R2
        AND R3, R3
        JZ no_column
        CALL button_conversion        ;Função que devolve o valor da tecla para R9

no_column:
        SHR R1, 1                                   ;Proxima linha do teclado
        JNZ nextline                        ;R6 tem valor unico do teclado

        CALL wrdisp ;Função que actualiza as strings caso seja novo valor
        …
        POP R1
        RET
```

2.2.1. Actualização da tecla (wrdisp)

A rotina wrdisp actualiza a string LASTKREAD e essa é a será usada em outras rotinas para comunicação entre o teclado essas rotinas, como será um valor numérico e não simbólico, terá a versatilidade. O valor de nenhuma tecla pressa, essa sim é que será simbólica, tendo que ser feita a verificação dessa em outras rotinas que usem o teclado.

```
wrdisp:
        PUSH R0
        PUSH R1
        PUSH R5
        AND R7,R7                                   ;Verifica se alguma tecla foi pressa
        JNZ zero_keyboardcheck
        MOV R8, NOKEY  ;se for zero entao R8 = FFH e é o que mostra no ecra

zero_keyboardcheck:
        MOV R0, LASTKDISP      ;Verificação de qual a ultima tecla displayada
        MOVB R1, [R0]
        CMP R8, R1             ;Compara se o valor ja foi mostrado no ecra
        JZ fim_display         ;Se ja foi mostrado entao passa para o fim da função
        MOVB [R0], R8  ;Senão, entao actualiza a variavel de ultima tecla mostrada
        MOV R0, LASTKREAD
        MOVB [R0], R8
        ;MOV R5,
        ;MOVB [R5], R8                ;E escreve para onde os displays estão a ler.
fim_display:
        POP R5
        POP R1
        POP R0
        RET
```

2.3. Executar funções teclado “keyboardfunc”

Usa uma tabela de words gravada em memoria e consoante o botão carregado faz “CALL“ da primeira word e passa como parâmetro o conteúdo da segunda word como paramtero no Registo8. Tendo a versabilidade de se quisermos só activar uma flag assim o fazer, ou chamar mesmo rotinas inteiras. Caso teclado tenha mais teclas e queremos endereça-las, então basta adicionar mais elementos a tabela. Usamos a constante EMPTY_WORD para indicar que essas posições estão livres.

```
teclado_func: ;Isto sera lido de 2 a 2 words onde a primeira simboliza a funcao e a
segunda o paramtero para essa funcao
        WORD activar_flag                   ;0 Func
        WORD inicio_jogo                    ;0 Param
        WORD mov_tetramino                  ;1 Func
        WORD LEFT                           ;1 Param
        …
        WORD EMPTY_WORD                             ;8 Func
        WORD EMPTY_WORD                             ;8 Param
```

2.3.1. Pintar um pixel (paintxy)

Esta função tem 2+1 modos:

- Flag phatom_paint, decide se estamos em modo de verificação, importante notar que esta flag sobrepõe-se a segunda flag, do tetraminó, neste modo activamos a flag phatom_paint_found em 2 situações:
  - caso no pixel dado está a 1
    - Também é guardado na WORD phatom_point o valor da coordenada de colisão para detecção de colisão com o monstro.
  - caso o pixel dado é o ultimo, isto é defenido apartir de uma EQU

```
MOV R4, flag_phantom_paint    ; Flag para verificar se é para pintar ou nao
        MOVB R2, [R4]
        CMP R2, ACTIVO        ; Verifica se esta activa
        JNZ paintxy_check

        MOVB R2, [R0]         ;Ler celula
        MOV R1, ENDLINE
        MOV R4, R0
        SUB R4, R1
        JNN activar_phantom

        AND R2, R3            ;Verifica se esta vazia
        JZ paintxy_fim        ;Se nao entao passamos para o fim sem executar qualquer
alteração
activar_phantom:
        MOV R4, flag_phantom_found
        MOV R2, ACTIVO
        MOVB [R4], R2
        MOV R4, phantom_point
        MOV [R4], R10
        JMP paintxy_fim
```

- flag paint_clear tem faz com que a rotina faça 1 de 2:
  - Pintar o pixel
  - Limpar o pixel

```
paintxy_check:
        MOV R4, flag_paint_clear
        MOVB R2, [R4]  ;Como é uma string tera que ser lida com movb
        CMP R2, ACTIVO
        JNZ pintar_off


        MOVB R2, [R0]  ; Vamos buscar o valor que lá tem na coluna
        OR R3, R2              ; Adicionamos o pixel que queremos adicionar
        MOVB [R0],R3   ; Vamos escrever na coluna com o pixel adicionado
        JMP paintxy_fim
pintar_off:
        NOT R3                 ; Vamos inverter ficando por exemplo (0010) (1101)
        MOVB R2, [R0]  ; Vamos ler o que esta escrito
        AND R2, R3             ; e por exemplo temos (0011) e fazendo o AND so vamos
negar o que queremos apagar: (1101) and (0011) = (0001)
        MOVB [R0],R2   ; e vamos escrever esse valor na coluna
```

2.3.2. Pintar objecto “paint_tetramino”

Rotina que dada tabela objecto apontada pela WORD TETRASTR e coordenadas(valor guardado em R10), pinta esse objecto. A tabela é composta por:

- Primeira linha indica o numero de movimentos a percorrer, no exemplo em baixo serão 4 linhas
- Irá pintar na posição actual depois da transformação de cada linha, cada linha dentro da tabela será dividida em 2 bytes:
  - O primeiro byte é o que adicionar as coordenadas do Y
  - O segundo byte é o que adicionar as coordenadas do X

```
tetra_j_d:
        WORD 0004H              ;Numero de linhas a percorrer
        WORD 0000H
        WORD 0001H
        WORD 01FFH
        WORD 0100H
```

Olhando para a linha “WORD 01FFH” que significa (+1)(-1)
vamos supor que o R10 = 0A0B >> 0A(y) 0B(x)
A rotinha irá transformar o R10 em >> 0B(y+1) 0A(x-1)H

```
paint_tetramino:
        PUSH R0
        …
        PUSH R10 ;Para repor a posição orignal

        MOV R1, TETRASTR        ;Receber em memoria o tetramino a ser pintado
        MOV R2, [R1]            ;passagem do endereço na memoria onde ele se encontra
                                        ;R2 sera pontador na tabela
                                        ;A tabela e composta pelas transformações a
ser feitas ao ponteiro
        MOV R6, [R2]
        ADD R2, NEXTWORD
        MOV R4, 0                       ;inicio da contagem  sera sempre 4 blocos a
proximo_pixel:
        MOV R0, R10                     ;Ter em R0 a copia de R10
        SHR R0, 8                       ;Cortar o bits de menor peso
        MOV R3, LOWBYTE
        AND R10, R3                     ;cortar os bits de maior peso
        MOV R1, [R2]            ;R1 agora tem o valor da primeira posição da tabela
        ADD R2, NEXTWORD
        MOV R5, R1                      ;Ter copia do valor de R1 para separar em X e Y por
vamos aplicar transformações
        SHR R1, 8
        ADD R0, R1
        SHL R0, 8
        MOV R3, LOWBYTE
        AND R5, R3
        ADD R10, R5
        AND R10, R3
        ADD R10, R0
        CALL paintxy
        ADD R4, 01H
        CMP R4, R6      ;Compara basicamente com 4 para ver se ja foram pintados todos
        JNZ proximo_pixel

        POP R10
        …
        POP R0
        RET
```

2.3.3. Movimento tetraminó “mov_tetramino”

Função de movimento de tetraminó aceita 3 parametros de movimento, mais 2 caso a flag de rotação esteja ativa, todos paramteros menos 1 em especifico são valores numéricos que são aplicados diretamente. É segue as seguintes etapas:

- Limpa tetraminó
- Muda valor da posição de acordo com parâmetro.
    - Caso flag de rotação esteja activa irá só selecionar o próximo ou anterior valor da tabela de transformações.
- Verifica se pode pintar nesse ponto
    - Caso não possa, reverte o valor da posição
    - Caso o movimento seja vertical e a flag do phatom_found está activa então iremos verificar se a colisão foi com o monstro, agarrando na posição de phatom_point e verificar se pertence ao monstro com a rotina “pertence”.
- Pinta tetraminó
- Se a função receber como valor simbólico “DROP_IT” (0FF0H) significa que queremos um ciclo da peça cair até colidir com algo.

Esta é uma das rotinas que é bloqueada caso a flag inicio jogo não está com o valor 0.

```
mov_tetramino:
        PUSH R0
        PUSH R1
        PUSH R2
        PUSH R3
        PUSH R4

        MOV R0, inicio_jogo                  ; qualquer situacao anormal no jogo
        MOVB R1, [R0]                        ; desabilita a funcao
        CMP R1, DESACTIVO                    ; (pausar, fim do jogo)
        JNZ ret_mov_tetramino         ;
        …
        MOV R8, flag_paint_clear ;instrucoes que vao limpar
        CALL desactivar_flag ;a peca na posicao
        CALL paint_tetramino ;actual, como flag_paint_clear e 0, paint_tetramino limpa a
peca
        …
        ADD R10, R3                                  ; move a peca para a esquerda
        … verificação

        MOV R8, flag_phantom_paint    ;
        CALL desactivar_flag          ; desativa a flag_phantom_paint

        MOV R8, flag_paint_clear
        CALL activar_flag                    ; activa a flag_paint_clear para pintar

        CALL paint_tetramino          ; chama a funcao para pintar o tetramino
        …
        RET
```

2.3.4. Rodar tetramino (rotate_tetra)

Rotina que é chamada pela mov_tetramino que recebe como paramtero R8 a indicar qual o sentido que vamos percorrer a tabela de transformações. Actualizando a word TETRASTR que comunica com a “paint_tetramino”

```
rotate_tetra:
        PUSH R0
        …
        MOV R0, TETRAROT        ;Buscar o indice da tabela de rotacoes
        MOVB R1, [R0]           ;colocar o valor em R1


        CMP R8, 1                               ;verificacao se nao estamos no fim da
tabela e reciniacilizar para o proximo elemento
        JNZ rotate_tetra_check_negative
        CMP R1, TAMANHO_TABELA_ROT
        JNZ rotate_tetra_check_negative
        MOV R1, 0
        JMP rotate_tetra_noadd

rotate_tetra_check_negative:
        MOV R0, 0FFFFH                  ;Verificacao se
        CMP R8, R0                              ;nao estamos no primeiro
        JNZ rotate_tetra_add   ;indice da tabela
        CMP R1, 0                               ;para passar para o ultimo elemento
        JNZ rotate_tetra_add
        MOV R1, TAMANHO_TABELA_ROT      ;Colocar o indice no fim da tabela
        JMP rotate_tetra_noadd

rotate_tetra_add:
        ADD R1, R8                      ;aplicar a transformacao de R8
rotate_tetra_noadd:
        MOV R0, TETRAROT
        MOVB [R0], R1
        MOV R0, 2                       ;como e por words
        MUL R1, R0                      ;teremos de multiplicar o indice por 2
        MOV R0, TETRATBL        ;word onde esta guardada a tabela de rot
        MOV R2, [R0]            ;buscar a tabela de rotacoes actual
        MOV R0, R2
        ADD R0, R1                      ;Aplicar o indice
        MOV R1, [R0]
        MOV R2, TETRASTR
        MOV [R2], R1
        …
        POP R0
        RET
```

2.4. Interrupções “interrupt_exec_func”

Usa uma tabela de words no mesmo género que da tabela funções que teclado para decidir quais a funções a executar para cada interrupção. Estas rotinas não serão executadas se a flag inicio_jogo estiver diferente de 0.

```
interrupt_func:
        WORD mov_tetramino              ;0 Func
        WORD NEXTLINE                   ;0 Param
        WORD mov_monstro                ;0 Func
        WORD EMPTY_WORD                         ;0 Param
```

2.4.1. Interrupção 0 “descer_peca”

Interrupção que do movimento vertical do objeto, a interrupção activa a flag e o interrupt_exec_func usa a função mov_tetramino para descer a peca e desliga a flag.

2.4.2. Interrupção 1 “mov_monstro”

Interrupção que verifica primeiro se o gerador ativou a flag de monstro e caso sim activa a flag movimento do monstro. A rotina de interrupções verifica se está ligada e executa a função mov_monstro que cria o monstro e move o monstro para a esquerda, esse movimento inclui partir a parede e verificação de colisão.

2.4.2.1. Movimento monstro “mov_monstro”

Esta rotina usa tem como primeira parte usar as funções de tetraminó (pintar, mover) ou seja, esta rotina é construída a volta do funcionamento destas. Por isso, sabendo que a função movimento não permite que o um tetramino mover através da parede, supomos que se a posição do monstro não se alterou apos movimento, que chocou contra a parede, usaremos a função limpa_parede, que usando como constante o tamanho vertical do monstro, limpa a parede nessa secção, a posição do monstro é enviada como paramtero, a segunda vez que o movimento não se alterar significa que colidiu com um tetramino e que acabou o jogo, caso também o a posição X do monstro seja igual a 0 significa que estamos na situação que o monstro colidiu com a parede esquerda. Isto não acontece caso a flag_colisão_monstro esteja acesa pois significa que conseguimos acertar o monstro com um dos nossos tetraminós.

```
mov_monstro:
        PUSH R0
        …
        PUSH R10
        MOV R0, TETRA_POS_MONSTRO
        MOV R10, [R0]

        MOV R1, monstro
        MOV R0, TETRASTR
        MOV R4, [R0]
        MOV [R0], R1
        ;Validacao de esquerda do ecra
        MOV R1, R10
        MOV R0, LOWBYTE
        AND R1, R0
        CMP R1, 0
        JNZ nao_fim_jogo_monstro
        ;
        MOV R8, flag_phantom_paint
        CALL desactivar_flag
        ;
        MOV R8, flag_paint_clear
        CALL desactivar_flag
        CALL paint_tetramino
        MOV R8, R10
        CALL reset_monstro
        ;************************************************
        MOV R8, game_over
        CALL pinta_ecra
        MOV R0, inicio_jogo
        MOV R1, GAME_OVER
        MOVB [R0], R1
        ;************************************************
        ;
        JMP fim_mov_monstro

nao_fim_jogo_monstro:
        MOV R0, flag_colisao_monstro
        MOVB R5, [R0]
        CMP R5, ACTIVO
        JNZ naocolisao_mov_monstro
        MOV R8, flag_phantom_paint
        CALL desactivar_flag

        MOV R8, flag_paint_clear
        CALL desactivar_flag
        CALL paint_tetramino
        CALL reset_monstro
        MOV R8, flag_colisao_monstro
        CALL desactivar_flag
        JMP fim_mov_monstro

naocolisao_mov_monstro:
        MOV R8, LEFT
        CALL mov_tetramino
        MOV R0, TETRA_POS_FINAL_MONSTRO
        MOV R2,[R0]
        CMP R10, R2
        JNZ nao_limpar_coluna
        ;Primeira tentative calha sempre na coluna
        MOV R8, R10
        CALL limpar_coluna
        MOV R8, LEFT
        CALL mov_tetramino
        ;Segunda tentativa sera numa peca nossa ou no "monte"
        CMP R10, R2
        JNZ nao_limpar_coluna
        ;
        MOV R8, flag_paint_clear
        CALL desactivar_flag
        CALL paint_tetramino
        MOV R8, R10
        CALL reset_monstro
        ;************************************************
        MOV R8, game_over
        CALL pinta_ecra
```

```
        MOV R0, inicio_jogo
        MOV R1, GAME_OVER
        MOVB [R0], R1
        ;*********************************************
        JMP fim_mov_monstro
        ;
nao_limpar_coluna:
        ;
        MOV R0, TETRA_POS_FINAL_MONSTRO
        MOV [R0], R10
        MOV R0, TETRA_POS_MONSTRO
        MOV [R0], R10
        ;
fim_mov_monstro:
        MOV R0, TETRASTR
        MOV [R0], R4
        POP R10
        …
        POP R0
        RET
```

2.5. Contar pontos (add_points, clean_points)

Temos a rotina soma pontos aos displays que recebe como parâmetro o valor a ser adicionado e uma outra que limpa os pontos, caso seja estado GAME_OVER.

```
add_points:
        PUSH R0
        …
        MOV R0, pontos_screen_decimal
        MOVB R1, [R0]
        ADD R8, R1
        CALL convert_hx_to_dec
        CALL writedisplay
        MOV R0, pontos_screen_decimal
        MOVB [R0], R8
        …
        POP R0
        RET

convert_hx_to_dec:    ; converte o resultado de
        PUSH R0
        …
        MOV R0, R8                                          ;
        MOV R1, R0

        MOV R2, mask_3_0
        AND R0, R2
        MOV R2, DEZ
        CMP R0, R2
        JN negativo_chxdec

        SUB R0, R2
        MOV R2, DEZ_DEC
        ADD R1, R2

negativo_chxdec:
        ;ADD R1, R0
        MOV R2, mask_7_4
        AND R1, R2
        MOV R2, CEM
        CMP R1, R2
        JN nao_cem
        MOV R1, 0
nao_cem:
        ADD R1, R0
        MOV R8, R1
        …
        POP R0
        RET
```

```
writedisplay:
        PUSH R0
        MOV R0, DISPLAYS
        MOVB [R0], R8
        POP R0
        RET

clean_points:
        PUSH R0
        PUSH R1

        MOV R0, pontos_screen_decimal
        MOV R1, 0
        MOVB [R0], R1
        MOV R0, DISPLAYS
        MOVB [R0], R1

        POP R1
        POP R0
        RET
```

2.6. Modo Pausa

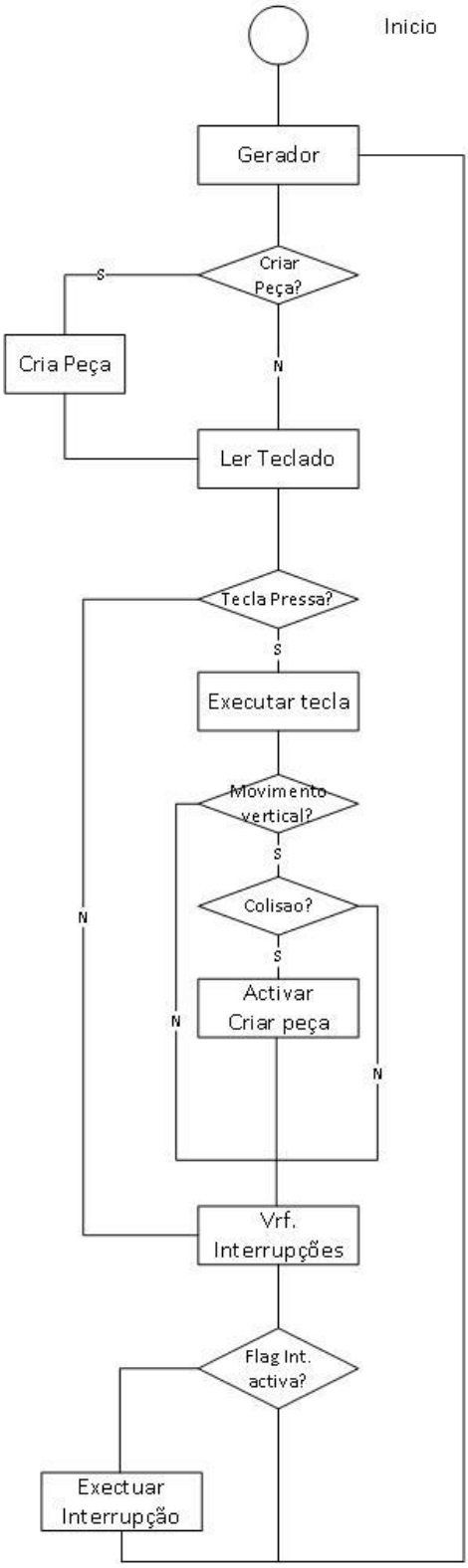
Um modo que altera a flag inicio_jogo que bloqueia processos de movimento no ecrã e as interrupções e troca as cores.

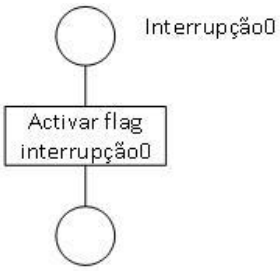
```
pausa_jogo:
        PUSH R0
        PUSH R1
        PUSH R2

        MOV R0, inicio_jogo
        MOVB R1, [R0]
        CMP R1, GAME_PAUSE
        JZ continuar_jogo
        MOV R1, GAME_PAUSE
        MOVB [R0], R1
        CALL flip_screen
        JMP fim_pausa_jogo
continuar_jogo:
        CALL flip_screen
        MOV R1, DESACTIVO
        MOVB [R0], R1
fim_pausa_jogo:
        POP R2
        POP R1
        POP R0
        RET
```

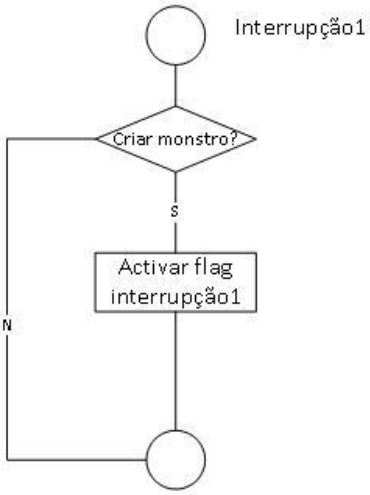
2.7. Fazer troca de peças (flip_tetra)

Criamos opção de jogo de conseguir segurar num tetraminó e gerar peça nova na primeira volta, e nas seguintes troca a actual com a do backup.

2.8. Fluxograma







3. Conclusões

O Trabalho foi concretizado na sua totalidade, com a adição das mensagens de “GAME START” e “GAME OVER”.

Consideramos o funcionamento e desempenho do nosso projecto bastante satisfatório, na medida em que, á exceção dos problemas do Simulador, o nosso projecto corre na perfeição, com todas as funcionalidades a funcionarem a uma velocidade bastante satisfatória.

Na execução do projecto optamos pela implementação de um teclado reprogramável á base de uma tabela de Words, o que resultou numa maior facilidade em testar certas funcionalidades específicas do projecto, devido a nos ser possível programar o teclado para ativar qualquer flag que nos seja conveniente. Também nos facilitou, na reta final do projecto, quando tivemos de mudar as funcionalidades de cada tecla do Teclado, para que os controles sejam mais intuitivos.

No entanto, consideramos que existem alguns aspetos a melhorar, como o fato que a colisão horizontal do tetraminó com o monstro termina o jogo, não devendo isto acontecer. Outro aspecto em que podíamos melhorar o projeto seria a implementação de uma secção no ecrã que mostrasse as próximas peças a serem evocadas.