LICENCIATURA EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Relatório do Projecto de Introdução à Arquitectura de Computadores

TRON

Grupo 68
Turno de 2ª feira às 12h30
81115 Rodrigo Lousada
79740 Miguel Pereira
46829 Cristina Pereira

2014 / 2015

INSTITUTO SUPERIOR TÉCNICO

Índice

1. **Introdução** 3
2. **Instruções de Jogo** 3
3. **Descrição e Explicação da Programação** 4
   - 3.1. **Rotina Principal** 4
   - 3.2. **Rotinas MenuInicial, Modo_Grafico, EscString e EscCar** 5
   - 3.3. **Rotinas EscreveLCD, EscStrLCD e EscLCD** 5
   - 3.4. **Tabela de Colisão e a Rotina ConverteCoordenadas** 6
   - 3.5. **Rotinas MenuDeJogo, LimpaJanela e EscStringTabela** 7
   - 3.6. **Rotina IniciaJogo e associadas** 7
   - 3.7. **Virar para o lado** 9
   - 3.8. **Rotina MenuTerminado** 9
4. **Principais Dificuldades** 10
5. **Características Avançadas** 12
6. **Conclusão** 13

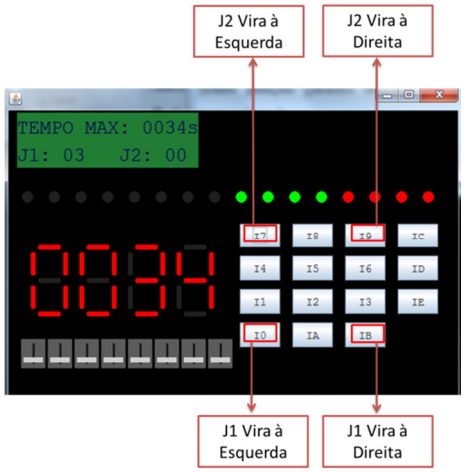
1. Introdução

Este relatório surge com intuito de ajudar quem quiser perceber melhor o código concebido pelo grupo, para a execução do projecto da cadeira de Introdução à Arquitectura de Computadores. O projecto consiste no desenvolvimento do jogo TRON na linguagem de programação *assembley* para o processador P3. Este jogo possui 2 jogadores e decorre na janela de texto do P3, onde cada jogador deverá conduzir a sua partícula escolhendo mudanças de direcção de modo a limitar o espaço ao adversário e a evitar colisões com os limites e o seu rasto. Neste documento são descritos os detalhes da programação utilizados no jogo.
O jogo utiliza ainda os diversos recursos disponibilizados, como são exemplo, a janela de texto, os *displays* de 7 segmentos, os LED's, o *display* LCD e os interruptores.
Aconselhamos a leitura do relatório com a visualização simultânea do código.

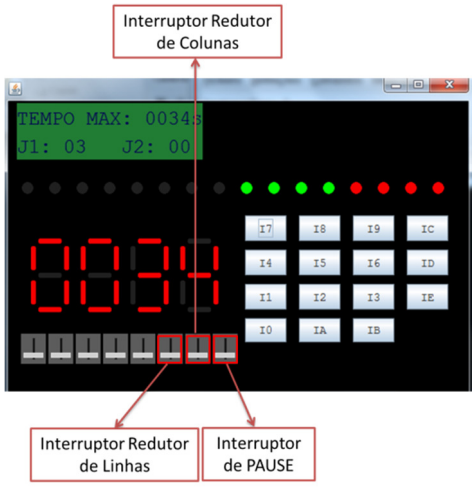
2. Instruções de Jogo

Como já referido no enunciado, o TRON é um jogo *multiplayer* bastante simples, sendo o único objectivo bloquear o avanço do adversário e evitar colidir com qualquer dos rastos deixados pelos jogadores ou com a moldura de jogo.

Cada jogador pode virar-se para a esquerda e para a direita, mas (ATENÇÃO!) tal como uma pessoa normal, se se virar duas vezes para o mesmo lado antes de andar, acaba virado para trás, o que não será benéfico neste caso uma vez que chocamos com o nosso próprio rasto. Os jogadores têm os seguintes botões para se movimentar:



Podem ainda ser utilizados 3 interruptores:



3. Descrição e Explicação da Programação

3.1. Rotina Principal

O programa tem origem na posição 0000h começando com um *jump* (JMP) para a etiqueta Inicio, que corresponde ao programa principal. O grupo após analisar diversos códigos oferecidos pelo corpo docente na página da cadeira, achou por bem seguir uma metodologia idêntica, deixando assim a rotina principal no final do código.
Esta rotina corresponde ao TRON do fluxograma apresentado. Começando por colocar o SP na sua posição inicial através da constante SP_INICIAL (FDFFh), é chamada a rotina Reinicia que irá colocar muitas das variáveis como estavam definidas inicialmente, e colocar a zero os dispositivos necessários para a inicialização do jogo. A rotina Reinicia embora desenvolvida para o final do programa foi a solução para não ser necessário fechar e voltar a abrir o simulador do P3, uma vez que quando se limpa a memória este não coloca a zero nem os leds nem os *displays* de 7 segmentos.
Após garantir que está tudo pronto para correr sem problemas, o programa chama, respectivamente, as rotinas MenuInicial e EscreveLCD que correspondem à escrita do menu de início de jogo e do LCD, aguardando então a instrução dos jogadores para começar o jogo. Através da instrução ENI (*enable interrupts*) as interrupções permitidas pela máscara (falado mais a frente) passam a ficar activas.
Sendo a interrupção I1 responsável por incrementar a varável FLAG, o programa irá ficar "preso" na condição ConfereFlag até que alguém prima o botão I1, permitindo assim iniciar o jogo. Uma vez que iremos necessitar novamente da interrupção I1 para jogar novamente basta-nos colocar a FLAG novamente a zero.

Entrando no Jogo chamamos a rotina MenuDeJogo para escrever a moldura de jogo e a rotina IniciaJogo para começar a jogar.
Uma vez terminado um jogo, será chamada a rotina MenuTerminado que irá escrever que o jogo terminou e perguntar se queremos jogar novamente, e a rotina ActualizaLCD para actualizar as pontuações dos jogadores o tempo máximo decorrido num jogo.
Após o final do jogo, o programa irá ficar novamente retido na condição ConfereFlag1 que funciona exactamente da mesma forma que a condição ConfereFlag, aguardando assim a utilização do botão I1 para iniciar um novo jogo.
Após ser dada a ordem de iniciar um novo jogo, o programa chamará a rotina Reinicia (já utilizada anteriormente), permitindo assim o *jump* (JMP) para a etiqueta Jogo onde, apenas se verifica uma alteração quando ali passamos pela primeira vez: os valores no LCD.

3.2. Rotinas MenuInicial, Modo_Grafico, EscString e EscCar

Ao ser chamada a rotina MenuInicial o programa escreve nas linhas 12 e 13, as mensagens "Bem-vindo ao TRON" e " Prima o interruptor I1 para começar".

Esta rotina começa por colocar a constante INT_MASK_Inicio no endereço da Máscara de Interrupções (FFFAh), permitindo assim apenas a utilização da interrupção I1.
Para iniciar o Modo Gráfico da janela de texto do P3, é chamada a rotina Modo_Grafico que envia a constante MODO_GRAFICO (FFFFh) para a posição de memória do IO_CURSOR (FFFCh).

Para a escrita das mensagens são utilizadas as rotinas EscString e EscCar introduzidas em aulas anteriores, utilizando a pilha de forma a reduzir o tamanho de código e generalizar assim a rotina de forma a que possa ser utilizada em mais do que uma situação.

A rotina EscString é a rotina que efectua a escrita de uma cadeia de caracteres, terminada pelo caracter FIM_TEXTO (dai todas as nossas *strings,* STR, possuírem este caracter no final), na janela de texto numa posição especificada. Pode-se definir como terminador qualquer caracter em ASCII.
Após colocar o IO_CURSOR na posição desejada e seleccionar o caracter a escrever, a rotina EscString irá chamar a rotina EscCar até encontrar o caracter FIM_TEXTO, situação na qual irá dar como terminada a escrita da *string* e consequentemente sair da rotina.

A rotina EscCar limita-se a efectuar a escrita de um caracter para o ecrã, enviando o mesmo para a posição de memório do IO_WRITE (FFFEh), podendo este ser visualizado na janela de texto.

3.3. Rotinas EscreveLCD, EscStrLCD e EscLCD

A rotina Escreve LCD tem o intuito básico de escrever a mensagem "TEMPO_MAX: 0000s" na linha 1, e as mensagens "J1: 00" e "J2: 00" na linha 2 do LCD.

Para isto é utilizada a rotina auxiliar EscStrLCD que consiste numa versão adaptada para o LCD da rotina já conhecida EscString. As únicas diferenças são a substituição das constantes IO_CURSOR e IO_WRITE pelas constantes LCD_CURSOR e LCD_WRITE, deixando de escrever na janela de texto e passando a fazê-lo no LCD. Existe também o cuidado de que ao enviar para a pilha a localização do cursor para o primeiro caracter de cada *string* o endereço de posição tem de ser criado com base no que é descrito no Manual do P3 (pág.28).

A rotina EscLCD, dada pelo corpo docente num dos seus programas de demonstração, é criada já, para utilizar mais tarde quando for necessário converter algum número em caracter ASCII, ou seja fazer desse número uma *string.*

3.4. Tabela de Colisão e a Rotina ConverteCoordenadas

De forma a idealizar um método eficaz para o programa reconhecer quando é que um jogador colide ou não, percebemos de imediato que o programa antes de escrever o movimento do jogador teria de ir à posição a ser escrita e ver se já lá estava algo, se estivesse era porque o jogador estaria a colidir. Mas se o P3 não consegue ler directamente da janela de texto, como faríamos isso?
A resposta foi reservar um dado número de posições de memória a partir da origem 8000h (de forma a facilitar os cálculos), criando uma tabela que registava todos os espaços já ocupados. Sempre que algo era escrito numa dada posição da janela de texto, a posição correspondente na tabela de colisão seria incrementada.

De seguida pensámos que se existe um vasto espaço de posições não ocupadas na janela de texto, porque não restringirmo-nos às necessárias, ou seja, 50*22 posições? Teríamos então de criar uma operação matemática que convertesse a coordenada da posição a escrever na janela de texto para a coordenada correspondente na tabela de colisão (composta por 1100 posições).

Com isto, criamos a rotina ConverteCoordenadas, que recebe uma coordenada, da posição a escrever na janela de texto, contida na pilha. A subtracção de 0110h faz um deslocamento segundo um vector com essas coordenadas, colocando o canto superior esquerdo da moldura (primeira posição que é escrita) na posição 0000h.
A operação seguinte consiste em ter num registo a linha e noutro a coluna da coordenada. A execução faz-se tendo a coordenada num registo e dividir o mesmo por outro registo com o valor 0100h.
De forma a minimizar as posições necessárias para a tabela de colisão, sabemos que o último caracter da primeira linha será seguido do primeiro caracter da segunda. Sabendo isto multiplicamos a linha pelo número máximo de colunas possíveis (50) e adicionamos esse valor à coluna. Originando uma sequência como no exemplo seguinte:

1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0

Que equivale a:

+	-	-	-	-	-	-	-
-	-	-	+	|			

Para esta rotina a utilização do R1 é fulcral sendo o único registo que não salvaguardamos pois necessitaremos que este mantenha o seu valor para uma utilização na rotina que chamou o mesmo. O R1 irá conter a coordenada convertida para a tabela de colisão e será utilizada na rotina que chamou a sub-rotina ConverteCoordenadas.

3.5. Rotinas MenuDeJogo, LimpaJanela e EscStringTabela

A rotina MenuDeJogo tem como objectivo escrever a moldura do jogo, tanto na janela de texto como na tabela de colisão.
Começando por limpar toda a janela de texto, é chamada a rotina LimpaJanela, que embora para o simulador bastasse ser executada a rotina Modo_Grafico, no caso da placa é necessário enviar espaços para todas a posições que queremos limpar.
Com isto, queremos limpar todas as posições desde a posição de coordenadas (0,0) até à posição de coordenas (24,80). De forma a facilitar os cálculos enviamos uma *string* de 80 espaços, sendo apenas necessário ir aumentando a linha que é escrita, desde a linha 0 até estar na linha 25.

Para escrever a moldura não nos bastava utilizar a rotina EscString pois tínhamos de escrever não só na janela de texto como na tabela de colisão. Criámos então a rotina EscString_Tabela, que é a rotina EscString alterada para também escrever na tabela de colisão. Os limites, superior e inferior, da moldura escreveram-se com uma *string*, enquanto os limites laterais são escritos através duma alteração na EscString_Tabela, fazendo com que esta escreva um caracter ("|") de uma posição até outra, descendo sempre de linha.

3.6. Rotina IniciaJogo e associadas

A rotina IniciaJogo é onde toda a acção de jogo acontece. Começa por actualizar a Máscara de Interrupções para interrupções necessárias ao jogo, ou seja, interrupções 15 (tempo), 0, B, 7 e 9. De seguida, envia para o porto o valor 1 para o TIMER_VALUE (FFF6h), dizendo que tem de contar de uma em uma décima de segundo, e dá início ao relógio enviando o valor 1 para o porto TIME_CONTROL (FFF7h).

Coloca a FLAG_Vencedor a 0, caso esta já seja uma repetição do jogo, envia o NIVEL_1_VEL para a Vel_Actual, ou seja, coloca a velocidade 7 décimas de segundo por movimento para a variável que tem a velocidade actual, e faz o mesmo com os leds.
Ao entrar no ciclo de contagem, cria um ciclo de contagem que apenas sai caso algum jogador tenha colidido. Criámos então as rotinas ContHex, Confere_ Nivel e Movimento, que, respectivamente, actualiza o *display* de 7 segmentos, confere o nível de jogo em que estamos, e movimenta as partículas.
A FLAG_Vencedor tem 4 valores possíveis, 0 caso o jogo ainda deva correr, 1 caso o jogador 1 tenha vencido, 2 caso o jogador 2 tenha vencido, e 3 caso seja um empate.

A rotina ContHex é responsável por dividir por diferentes registos todos os dígitos no Contador_ Nivel, caso o valor de Contador_Nivel ultrapasse os mil segundos, o Contador_Nivel é posto a 0. A rotina EscCont envia para os *displays* de 7 segmentos o dígito dos segundos para o porto FFF0h, o dígito das dezenas de segundo para

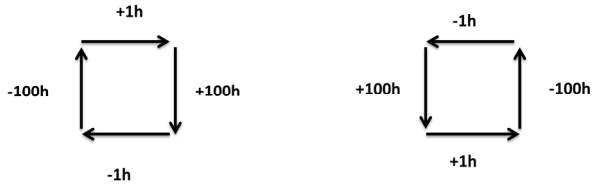
FFF1h, o dígito das centenas de segundo para FFF2h e o dígito dos milhares de segundo FFF3h.

A rotina Confere_Nivel serve para verifica em que nível de jogo nos encontramos e actualiza as características do jogo dependentes desse nível.
Passando o Contador_Nivel para um registo, comparamos esse registo com os TIME_NIVEL dos diferentes níveis, se a diferença for não negativa passamos ao nível seguinte, se for negativa é porque o Contador_Nivel ainda se encontra dentro deste nível, logo irão ser atribuídos os leds a ser ligados e a velocidade actual do nível.

A rotina Movimento foi a mais complicada de desenvolver. Começa por criar uma condição em que só irá ser feito o movimento quando o Contador_Vel atingir o valor de Vel_Actual, enquanto isto não acontece a rotina tem de permitir que o ciclo CicloCont continue a circular, saindo então da rotina. Quando o Contador_Vel chegar ao valor de Vel_Actual significa que a partícula se deve mover, permitindo assim um movimento para cada jogador. Colocamos o Contador_Vel a 0 para que seja possível continuar a fazer esta comparação para os movimentos restantes.
As rotinas de interrupção relativas a mudar a direcção do jogador (ViraJ1Esquerda. ViraJ2Esquerda, ViraJ1Direita e ViraJ2Direita) apenas alteram o valor de MOVI_J1 e MOVI_J2, ou seja, altera-nos o valor a adicionar à posição actual de cada partícula (se quisermos andar para a esquerda o valor é -1h, se quisermos andar para a direita o valor é 1h, se quisermos andar para cima o valor é 100h, e se quisermos andar para baixo o valor é -100h). O resultado será a posição a ser ocupada a seguir, escrevendo-a apenas caso esta não origine uma colisão.
Para determinar as colisões temos também de conseguir determinar quem é que ganha ou se é um empate. Comparamos, então, se as posições a ser ocupadas são iguais, caso isso se verifique, é um empate. Depois vemos se a posição a ocupar pelo jogador 1 está ocupada na tabela de colisão, sendo então necessário chamar a função ConverteCoordenada. Caso a posição esteja ocupada significa que o J1 perdeu, teremos de ver se o jogador 2 também perdeu, originando um empate (FLAG_Vencedor = 3), ou se foi só o jogador 1 a perder e ganhou o jogador 2 (FLAG_Vencedor = 2). Se o jogador 1 não perdeu, temos de ver se o jogador 2 colidiu, caso tenha colidido, o jogador 1 ganhou (FLAG_Vencedor = 1), caso não tenha é porque o jogo ainda está a decorrer (FLAG_Vencedor = 0). Se o jogo continuar a decorrer vamos proceder à escrita dos jogadores nas respectivas posições, e por segurança colocamos a FLAG_Vencedor a 0.

3.7. Virar para o lado

As rotinas de interrupção responsáveis pelas mudanças de direcção foram idealizadas de forma a que fossem separadas do movimento em si, dando assim duas capacidades "motoras" aos jogadores: avanço e rotação. As interrupções seriam então capazes de, consoante a direcção actual, determinar a direcção para qual mudar, segundo os dois ciclos ilustrados na imagem seguinte.



3.8. Rotina MenuTerminado

Após sair da rotina IniciaJogo, significa que o jogo acabou (FLAG_Vencedor diferente de 0), a rotina principal vai então chamar a rotina MenuTerminado, que limita-se a parar o relógio enviando 0 para o porto TIME_CONTROL (FFF7h). Faz ainda uma comparação entre o TEMPO_MAX e o tempo jogado no jogo que acabou, se o tempo jogado for superior significa que temos um novo recorde e desta forma o TEMPO_MAX vai ser assumir o valor do tempo jogado no último jogo, se for inferior o TEMPO_MAX mantém-se.
A máscara é actualizada para estar apenas disponível a interrupção 1, e escreve as mensagens "Jogo terminou" e "Pressione I1 para recomeçar", respectivamente nas linhas 12 e 13.

3.9. Rotinas ActualizaLCD, QuemGanhou, ContHexLCD e EscContLCD

A rotina ActualizaLCD serve apenas para actualizar os valores de TEMPO_MAX, P_J1 e P_J2 (pontuações de jogador 1 e 2) e actualizá-los no LCD.
Para isso começa por chamar a rotina QuemGanhou, que determina se o valor da FLAG_Vencedor é 1, 2, ou 3 (0 não pode ser uma vez que só chegamos a esta rotina através de uma condição que diz que FLAG_Vencedor não é 0). Se for um, P_J1 é incrementada e se for dois, P_J2 é incrementada. Se não for nenhuma das duas é porque é 3 e assim nenhuma alteração é feita às pontuações.
Para actualizar o TEMPO_MAX no LCD, chamamos a rotina ContHexLCD, que resulta de uma alteração da rotina ContHex para que em vez de separarmos os dígitos do Contador, separamos os do TEMPO_MAX actualizado. A rotina EscContLCD, também uma derivação da rotina EscCont que em vez de enviarmos os dígitos das unidades, dezenas, centenas e milhares de segundos para os *displays* de 7 segmentos, enviamos para as coordenadas atribuídas a cada dígito no LCD.
Finalmente, a rotina vai dividir as pontuações dos jogadores por 10d, separando-as em unidades e dezenas. De seguida irá escrever os dígitos de cada um nas respectivas coordenadas do LCD.

3.10. Novo Jogo

A condição de Novo Jogo (ConfereFlag1) pergunta se os jogadores pretendem voltar a jogar. Tendo sido posta a FLAG a 0, basta voltar a carregar I1 para podermos jogar. Se os jogadores não pretenderem voltar a jogar basta não carregar botão e o fim de jogo é considerado desta forma, ficando preso nesta condição.
Caso seja carregado I1, colocamos novamente a FLAG a 0, para a vês seguinte. Chamamos ainda a rotina Reinicia para "zerar" o jogo, ou seja, repor todas as variáveis e dispositivos nas suas características de origem, como por exemplo, FLAG_Vencedor a 0, XY_I_J1 e XY_I_J2 com as posições iniciais dos jogadores, e os MOVI_J1 e MOVI_J2 com as direcções com que se começa o jogo.

4. Principais Dificuldades

Sendo a linguagem de programação *assembley* difícil, o grupo demorou a conseguir entender verdadeiramente como funcionar com a mesma, necessitando de alguns dias de observação dos programas exemplo disponibilizados pelo corpo docente e revisão das aulas práticas para conseguir pôr o projecto em marcha.
A principal fonte de *bugs* foi sem dúvida a referência à posição de memória de uma variável ou constante, como por exemplo a utilização de XY_I_J1 quando se devia utilizar M[XY_I_J1].
A nível de idealização apresentámos dificuldades na idealização da Tabela de Colisão e a melhor forma de a utilizar poupando o máximo de espaço possível, concluindo no final que conseguimos uma fácil e perceptível forma de o executar.
Relativamente à alteração das direcções poderia ou não ser considerado um *bug* o facto de se carregarmos duas vezes seguidas numa interrupção de mudança de direcção, o jogador voltaria para trás, colidindo com o seu rasto e perdendo o jogo. Tendo conhecimento de métodos para o evitar, tal como flags que permitissem que um jogador fizesse apenas uma alteração de direcção, cada vez que a partícula se deslocasse, acabámos por manter esta opção pois retracta melhor a realidade.
Após a execução das características obrigatórias do jogo, o grupo consegui desenvolver as características avançadas com uma relativa facilidade devido à maior familiarização com linguagem.

O problema com a solução mais tardia foi, então, no teste com a placa do P3 utilizada nos laboratórios onde se pôde observar os dois símbolos representativos dos jogadores num extremo da janela de texto, verificámos que isso se devia a uma rotina chamada JPosicoesI, criada para a demonstração de dia 10, que tinha o objectivo escrever os jogadores nas suas posições iniciais. Ao retirar a rotina, que era vista como uma medida de segurança o problema viu-se resolvido e com o projecto a trabalhar sem problemas.

A única questão por resolver foi mesmo o facto de quando a moldura se apresenta reduzida para 32 colunas, o jogador 1 está inicialmente encostado à moldura de jogo, e, executando o P3 as interrupções anteriormente carregadas serão activas quando a máscaras as permitir. Permitam-nos ser mais claros, se pararmos o jogo

na jogada antes de uma colisão e carregarmos I1, mal perdermos o jogo, outro será reiniciado. No movimento das partículas acontece o mesmo, se o jogador 2 perder e o jogador 1, antes de se aperceber, carregar duas vezes I0, o mais provável de acontecer é que vai começar o próximo jogo numa direcção inicial diferente da que lhe programamos na rotina Reinicia, ou seja, estando a moldura reduzida e o jogador começar na direcção oposta, o jogador 1 irá perder automaticamente.
Pensamos em deixar as interrupções activas para não ficarem pendentes, no entanto ao correr no simulador este muitas vezes retornava ao início do programa.

5. Características Avançadas

As características adicionais ao programa foram implementadas posteriormente ao Capitulo 3 deste relatório sendo adicionada a utilização dos primeiros três interruptores, para opções como PAUSE, redução das colunas e/ou linhas da Moldura de Jogo.

Para o PAUSE fomos ao CicloCont, na rotina IniciaJogo, e adicionámos uma condição relativa ao mesmo. Lendo pelo porto IO_SWITCH (FFF9h), fizemos um AND sobre o bit correspondente ao interruptor que nos interessa verificar, para determinar se esse interruptor está ligado ou não, caso esteja, enviamos 0 para o TIME_CONTROL (FFF7h) de forma a parar o contador. Enquanto não decidimos voltar a jogar, o programa fica preso no ciclo Parado, perguntando sempre se o interruptor já está ligado, quando estiver, voltamos a pôr o contador a trabalhar da mesma forma de quando começámos o jogo, enviando o valor 1 para o TIMER_VALUE (FFF6h) e o TIME_CONTROL (FFF7h). No caso de não ter sido ligado o interruptor, o ciclo continua a correr normalmente.

Para as alterações das colunas e linhas na moldura de texto é criada uma condição no início do ciclo Jogo, no Programa Principal. Prosseguindo a mesma ideia do que fizemos com o interruptor anterior, lemos pelo porto IO_SWITCH (FFF9h), e fizemos um AND sobre os dois bits relativos aos dois interruptores responsáveis pela redução das linhas e colunas. Caso nenhum interruptor esteja ligado, é feito o percurso normal, sendo chamada a rotina MenuDeJogo, se o interruptor da redução das linhas estiver ligado, é chamada a rotina MenuDeJogo2, reduzindo o número de linhas de 20 para 15 enquanto se o interruptor da redução das colunas estiver ligado, é chamada a rotina MenuDeJogo3, reduzindo o número de colunas de 48 para 32. Se os dois interruptores estiverem ligados em simultâneo, as duas reduções também serão feitas em simultâneo, através da rotina MenuDeJogo4.
Todas estas rotinas são semelhantes à rotina MenuDeJogo tendo apenas pequenas alterações nas coordenadas de escrita e na *string* MolduraLimite1 que é substituída pela MolduraLimite2 que possui menos colunas.
Caso algum destes interruptores seja ligado durante um jogo, as alterações apenas serão feitas no jogo seguinte.

6. Conclusão

O grupo atribui um balanço positivo ao resultado final do projecto, não só cumprindo com tudo o que o enunciado pedia, como o que o enunciado sugeria. A originalidade faltou um bocado para sermos capazes de criar alguma funcionalidade de cunho pessoal. Preferimos desenvolver um relatório maior, pois, não querendo massacrar o corpo docente, esquematizamos a informação para que sejam possíveis duas situações: ler apenas o essencial, reconhecendo através da leitura do índice, e ao mesmo tempo, para quem quiser, ler todo o nosso relatório com informação detalhada do nosso percurso durante a sua execução.
Serve também como um complemento da explicação presente nos comentários.