INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

ARQUITETURA DE COMPUTADORES

LETI/LEE

IST-TAGUSPARK

81950 João Freitas

81968 Diogo Mesquita

82015 José Gomes

MODELO DO RELATÓRIO DO PROJETO

1. Introdução

Este é o relatório do projeto feito para a disciplina de Arquitetura de Computadores. O dito projeto tratou-se da criação de um jogo cujo objetivo era abater aviões através de um canhão. O jogador ganha pontos por cada avião abatido e tem de ter cuidado para não ficar sem munições, abatendo também cartuchos de munição para tal não acontecer.

Para construir o jogo foi necessário implementar:

- Um ecrã de 32x32 píxeis;
- Figuras no ecrã que representem o canhão, os aviões, os cartuchos e as balas;
- Um teclado para mover o canhão, disparar as balas, terminar e reiniciar o jogo;
- Dois contadores compostos por dois displays de 7 segmentos cada: um contador serve para apresentar a pontuação e o outro para apresentar as balas restantes;
- Interrupções que se servem dos relógios, um para mover as balas e o outro para mover os aviões e os cartuchos;
- Colisões entre as balas e os aviões e entre as balas e os cartuchos, sendo que a primeira aumenta a pontuação e a segunda as balas do jogador;

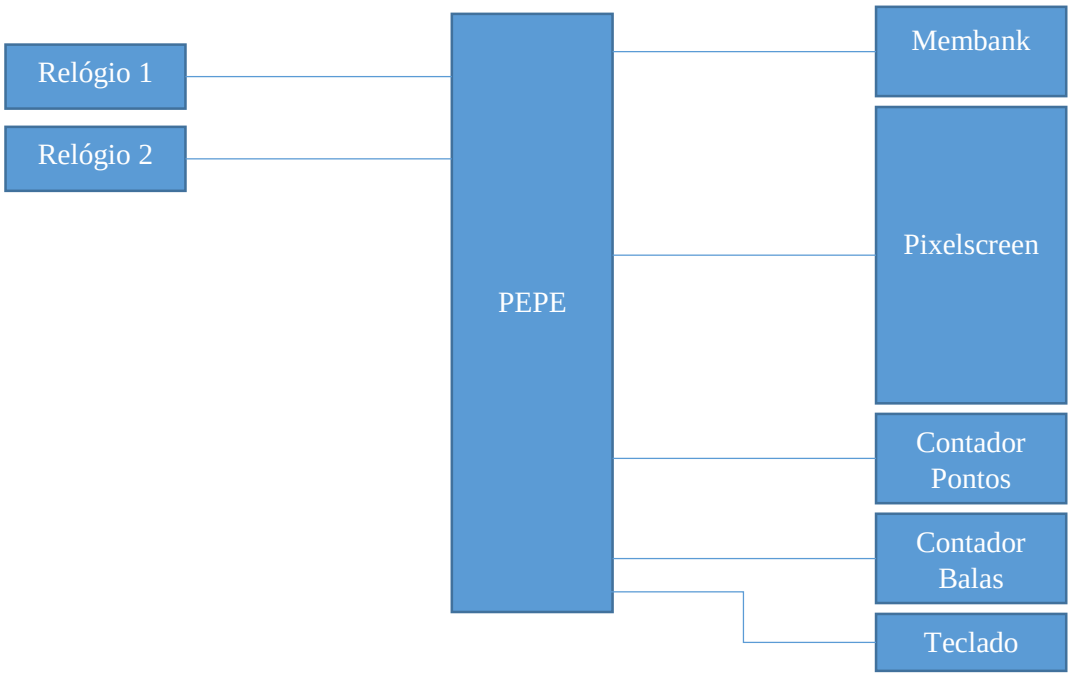
- Ecrã e condição de fim de jogo.

Na secção 2 descreve-se a estrutura do *hardware* com que se trabalhou e do *software* concebido, bem como as decisões do grupo no que toca ao mapa de endereçamento, à comunicação entre processos, variáveis de estado, interrupções e rotinas. Na secção 3 apresentam-se as conclusões retiradas sobre o projeto.

2. Conceção e Implementação

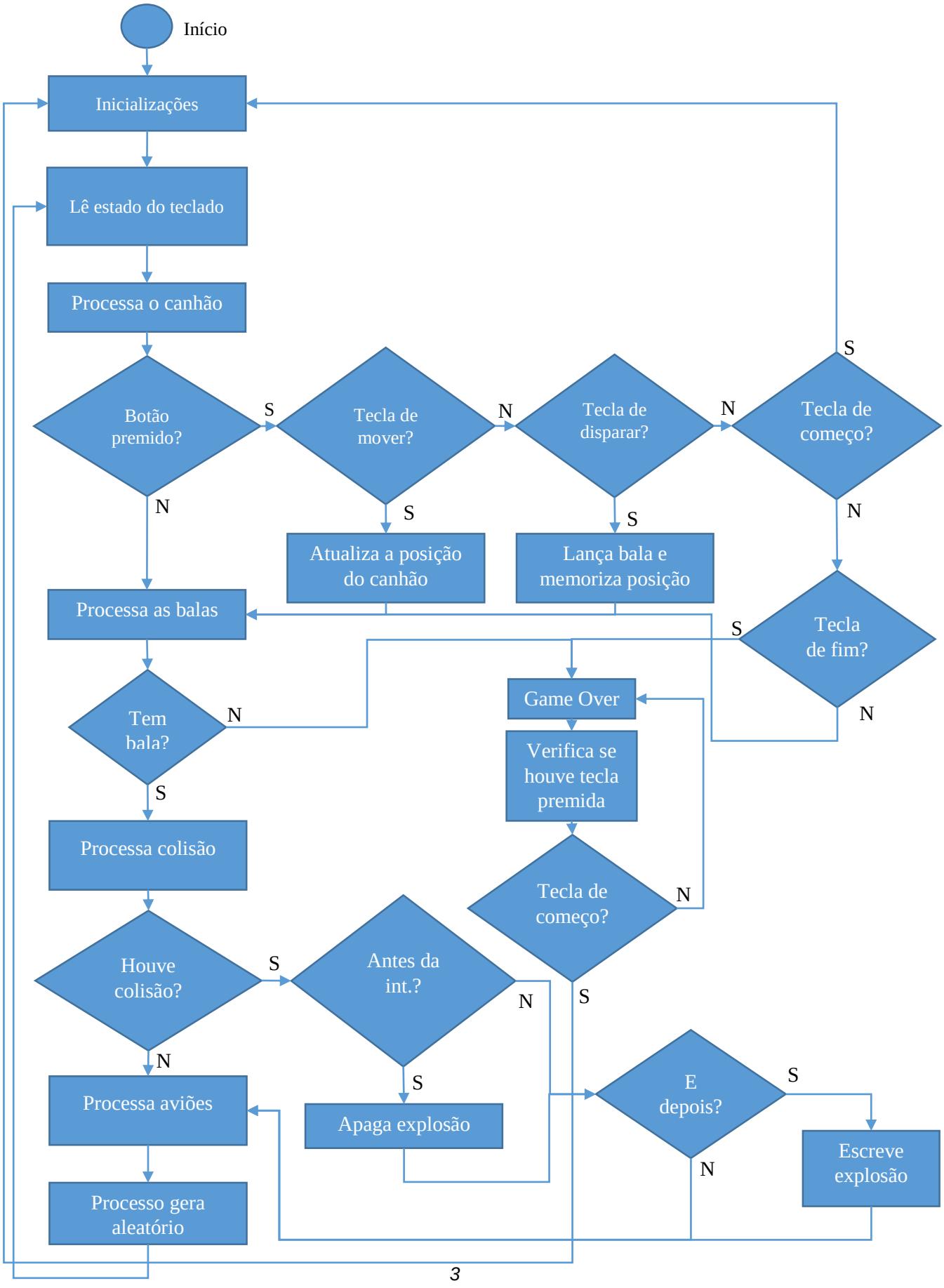
2.1. Estrutura Geral

Apresenta-se em baixo um diagrama de blocos a representar a estrutura do *hardware* utilizado no simulador:



O relógio 1 é usado para temporizar o movimento das balas, enquanto o relógio 2 é usado para o movimento dos aviões. O PEPE gere o avanço do programa, as interrupções dos relógios e guarda valores no Membank conforme necessário. Cada um dos contadores é composto por dois displays de sete segmentos, um para mostrar a pontuação do jogador e o outro contador para mostrar o número de balas que este ainda pode disparar, ambos em base decimal.

A seguir temos um fluxograma que demonstra a lógica do programa:


Início
Inicializações
Lê estado do teclado
Processa o canhão
Botão premido?
S
Tecla de mover?
N
Tecla de disparar?
N
Tecla de começo?
S
N
S
S
N
Atualiza a posição do canhão
Lança bala e memoriza posição
Processa as balas
Tecla de fim?
S
N
Tem bala?
N
S
Game Over
Verifica se houve tecla premida
Processa colisão
Tecla de começo?
N
S
Houve colisão?
S
Antes da int.?
N
N
S
Processa aviões
Apaga explosão
E depois?
S
N
Escreve explosão
Processo gera aleatório

2.1.1. Mapa de endereçamento escolhido

Em relação aos endereços escolhidos, além de posicionar o programa principal no endereço 0, era necessário posicionar a pilha, a tabela de exceções, as figuras, as máscaras, as variáveis de estado e restantes inicializações em geral num lugar onde o código não chegaria à medida que este se fosse estendendo. Por isso, escolheu-se o endereço 2000H.

2.1.2. Comunicação entre processos

Para a comunicação entre processos funcionar, era preciso algo mais que simplesmente guardar valores em registos, por isso recorreu-se ao uso de WORDs para guardar estes valores em memória, por exemplo:

```
balas_pos:     WORD -1        ; bala 1

               WORD -1
```

Aqui está a primeira entrada da tabela que guarda o posicionamento das balas. Esta tabela é usada em vários processos, por exemplo, no processo bala e no processo colisão, dessa forma é possível a comunicação entre os dois processos.

2.1.3. Variáveis de Estado

Para gerir as variáveis de estado, também se fez recurso às WORDs:

```
move_bala: WORD 0
```

Esta variável de estado é usada para indicar se ocorreu ou não a interrupção da bala:

```
next:

       MOV R0, move_bala      ; R0 com o endereco da indicação de interrupção

       MOV R8, [R0]           ; R8 com a indicacao se a interrupcao ocorreu ou nao

       CMP R8, 0

       JZ out                 ; se não ocorreu, saio
```

2.1.4. Interrupções

Tanto a rotina de interrupção das balas como dos aviões é usada para mudar as suas variáveis para 1 para estas mais tarde serem verificadas pelos processos que tratam das respetivas funções:

```
aviao_int0:

       PUSH R0

       PUSH R1

       MOV R1, 1

       MOV R0, move_aviao     ; R0 com o endereco da flag que indica se
```

```
                        ;houve interrup
        MOV [R0], R1            ; colocar flag a 1
        POP R1
        POP R0
        RFE
```

2.1.5. Rotinas

Existem 6 rotinas principais neste programa: a do teclado, do canhão, da bala, da colisão, do avião e da geração do número aleatório:

```
teclado:
        PUSH    registos
        testa as linhas para ver a tecla premida
        converte a linha para hexadecimal
        converte a coluna para hexadecimal
        obtém o valor da tecla a partir dos valores de linha e coluna
        POP     registos
        RET


canhao:
        PUSH    registos
        MOV     registo, linha do canhão
        MOV     registo, coluna do canhão
        obter valores do movimento
        verifica tecla premida ;se é inativa, de reiniciar, terminar ou disparar e agir
                        ;em conformidade
        POP     registos
        RET


bala:
        PUSH    registos
        verifica se alguma tecla foi premida e se sim, se foi a de disparar
        caso tenha sido a de disparar, cria bala e salta para o próximo passo
        caso contrário, salta logo para o próximo passo
        CMP     estado da interrupção, 0
```

```
        JZ      out
        estado posto a zero
repeat: CMP número de balas movimentadas, balas no ecrã
        JGE     out
        altera a linha da bala
        verifica se a bala chegou ao fim ou não
        se chegou, a bala fica inativa
        se não, atualiza a bala no ecrã
        JMP     repeat
out:    POP     registos
        RET
colisao:
        PUSH    registos
        verifica se a interrupção está ativa ou não
        se estiver apaga as colisões que estiverem no ecrã
        faz um ciclo para as balas, neste ciclo vai ver se a bala está ativa
        se não estiver passa à próxima
        verifica se a bala chegou há zona dos aviões
        se não estiver passa à próxima
        vai buscar a coluna da bala
        compara as suas coordenadas com as dos aviões ativos
        se as coordenadas coincidirem:
        escreve explosão
        mete a bala e o avião inativos
        e passa à próxima bala.
        se já verificou todas as balas sai do ciclo
        POP     registos
        RET
aviao:
        PUSH    registos
        verifica se a interrupção do avião está ativa
        se estiver mete a interrupção inativa
        vê quantas interrupções já passaram desde que o último avião entrou
        se já tiverem passado as suficientes vai gerar um novo avião
```

```
        gera um número aleatório para determinar a distância ao último avião criado
        escreve 0 no número de interrupções que passaram desde o último avião
        compara o número de aviões no ecrã com o máximo de aviões
        se for igual sai
        gera um número aleatório para escolher a linha onde vai escrever o objeto
        gera um número aleatório para decidir se vai escrever um avião ou um cartuxo
        se for um cartuxo verifica se já atingiu o número máximo de cartuxos
        se sim vai escrever um avião
        se não atualiza o número de cartuxos no ecrã
        escreve ou o cartuxo ou o avião no ecrã
        incrementa o número de aviões ou cartuxos ativos
        escreve na memória as coordenadas do mesmo
        faz um loop dos aviões
        se já viu todos os objectos salta para os POPs da rotina
        verifica se o objeto está ativo
        apago o objecto
        verifico se é um cartuxo, para decidir que objecto vai escrever
        subtrai 1 há coluna do objecto
        compara a coluna com o limite esquerdo
        se passar do limite objecto fica inactivo
        e decrementa o número de aviões que o jogador deixou passar se for um avião
        se não passar escreve o objecto e actualiza as suas coordenadas
        volta ao ciclo
        POP     registos
        RET

gerador:
        PUSH    registos
        adiciona 1 ao valor em n_gerador
        POP     registos
        RET
```

3. Conclusões

Do objetivo inicial, que era fazer um jogo de abate de aviões, conseguiu-se fazer o que estava estipulado, adicionando ainda um ecrã de boas-vindas, níveis de dificuldade, diferentes aviões para cada nível, ecrãs de transição de nível que apresentam o número máximo de aviões que podem passar para fora do ecrã antes de o jogo ser considerado perdido, bem como ecrã de vitória, ecrã de derrota diferente, diferentes parâmetros de aparecimento e movimento dos aviões assim como dos cartuchos, conforme o nível.

O programa funciona bem demonstrando que as decisões tomadas foram as acertadas, pois ao testar o projeto depois de este estar concluído, verificou-se que fazia o que era suposto sem grandes problemas, sendo estes mesmos problemas corrigidos após testar. No entanto, o desenvolvimento deste foi um pouco turbulento de forma desnecessária devido ao simulador, que ocasionalmente não funcionava como desejado, induzindo por vezes em erro, tendo sido no entanto possível contornar estes obstáculos.

Para o melhor funcionamento do jogo, recomenda-se um período de 600 para o primeiro relógio e 350 para o segundo porque se os valores forem mais baixos, a leitura das teclas não é feita corretamente.