Hugo Pitorro, nº 92478
Cristiano Clemente, nº 92440
Grupo: 34

Relatório do projeto

No âmbito da disciplina de IAC (Introdução à Arquitetura de Computadores), foi-nos proposto o desenvolvimento de uma versão minimalista do jogo “Gorillas”, programado na linguagem assembly do processador P3. Para tal, concebemos uma aplicação interativa e dinâmica com recurso aos dispositivos de I/O disponíveis, nomeadamente: o display de 7 segmentos, o teclado numérico e a janela de texto.

- **Abordagem Geral e Estruturas de Dados**

  Numa primeira fase, para desenvolver uma solução capaz de calcular a trajetória de um projétil considerando as limitações do P3, optámos por uma representação em vírgula fixa (Q8.8) que nos permite utilizar números decimais: os 8 bits mais significativos representam a parte inteira do número, enquanto que os últimos 8 bits representam a parte decimal.

  Ainda, considerando que o processador não suporta funções trigonométricas, optámos pela utilização de uma tabela que contém valores aproximados (em Q8.8) do seno dos ângulos entre 0 e 90º para cálculo da trajetória, simplificando o problema através de uma prática mais eficiente que a utilização de uma função aproximada (também foi considerada a série de Taylor).

  A partir deste momento prosseguimos para o desenho da arquitetura do programa. Em primeiro lugar, construímos, através de abstração procedimental, um esboço rudimentar onde considerámos as funcionalidades básicas do jogo. Ora, necessariamente, o cálculo da trajetória seria encapsulado numa função que devolvesse as coordenadas do projétil consoante a velocidade inicial, o ângulo e o tempo decorrido: daqui surge a necessidade de receber “inputs” do utilizador, bem como representar o projétil na janela de texto; o tempo seria recebido através do temporizador interno ao processador.

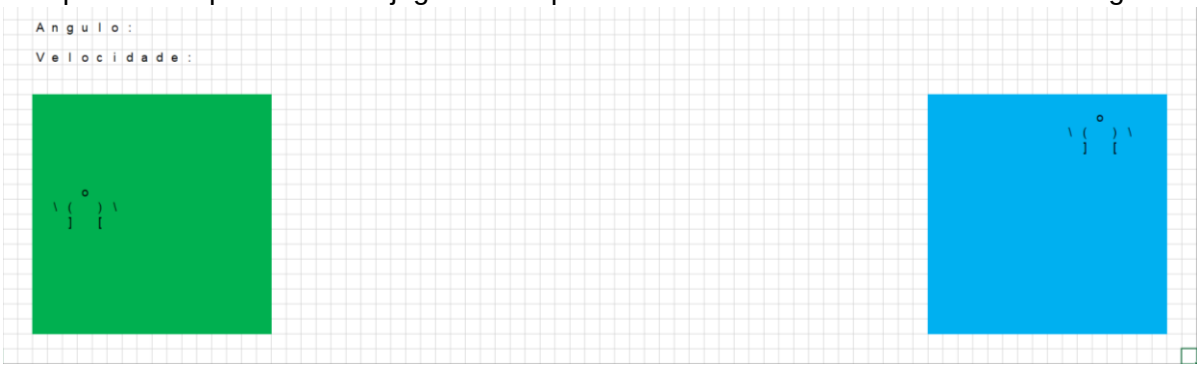
  Para que o jogo oferecesse alguma competitividade, implementamos um sistema de vidas: o jogador começa com 5 vidas e perde uma por cada lançamento em que falha o alvo ou acerta no próprio jogador (self-hit). Contudo, como esta abordagem limita o tempo de jogo, concluímos que seria necessário poder reiniciar o jogo após a sua conclusão. A condição para a vitória será, claro, a colisão entre o projétil (banana) e o alvo (gorila adversário).

  Finalmente, durante o desenvolvimento do jogo, apercebemo-nos que criar um jogo funcional para singleplayer e multiplayer, revelou-se um objetivo demasiado complexo para o espaço de tempo à nossa disposição. Deste modo, face ao problema que nos era proposto e ao tempo que utilizámos a desenvolver uma solução “ambígua”, concluímos que a melhor abordagem seria otimizar o modo singleplayer para que funcionasse a pleno e de forma a oferecer uma boa experiência ao utilizador.

Hugo Pitorro, nº 92478
Cristiano Clemente, nº 92440
Grupo: 34

- **Campo de jogo**

Já na segunda fase de desenvolvimento, ao considerar as dimensões que o P3 nos oferecia para a realização do jogo (24 linhas x 80 colunas), desenvolvemos a abordagem que nos parecia melhor para conseguir um jogo competitivo e com uma boa jogabilidade: considerámos que os gorilas deveriam estar contidos em subquadrados (16x16) e que a sua posição seria gerada aleatoriamente no interior dos mesmos. Esta metodologia permite-nos garantir simultaneamente que: as posições geradas para os gorilas são imprevisíveis - algo vital para a competividade do jogo – mas que existe uma distância saudável entre os gorilas.



- **Funcionamento do programa**

No decorrer do programa recorremos a várias rotinas que tratam de interrupções geradas pelo utilizador ou pelo temporizador interno ao processador: este método permite-nos oferecer ao jogador um controlo dinâmico sobre o que está a acontecer durante o jogo (ecrã com mensagem inicial, jogo em si, e a possibilidade de reiniciar o jogo ou terminá-lo).

Já na rotina em curso durante o decorrer do jogo, recorremos à abordagem modular onde cada função trata de aspetos estanques e fundamentais; construímos um ciclo distinguindo o início de uma nova ronda do “voo” regular. Ainda, definimos que a colisão do projétil com o alvo encerra o ciclo, oferecendo ao utilizador uma escolha: reiniciar o jogo ou terminá-lo.

Assim, temos várias funções que tratam, nomeadamente: do desenho dos gorilas, do tratamento das vidas (escrita e possível conclusão do jogo), de receber os inputs do utilizador, iniciar uma novo ciclo através do cronómetro, o cálculo das coordenadas do projétil, o cumprimento das condições terminais do jogo e o desenho da banana.

- **Funções complexas**

  - Gorilas → Gera posições aleatórias para os gorilas e desenha-os

Através da função random, que gera um número aleatório de 16 bits, conseguimos obter as coordenadas para ambos os gorilas (4 números de 4 bits cada um), definidas no interior dos quadrados. A partir deste momento, com as coordenadas nas variáveis (randoml, randomr), é chamada a função gorila e aí

são desenhados os gorilas do jogador e do alvo. O desenho recorre a uma função auxiliar (write) que recebe as coordenadas de escrita, a string a escrever e o seu comprimento e as escreve na janela de texto.

- lives → Processa a vida do jogador com o passar das rondas

  Ao início do jogo, o utilizador detém 5 vidas que são mantidas numa variável (player1); a cada ronda que passa (ronda – lançamento de um novo projétil) este valor é decrementado. Ainda, esta função é responsável pela deteção da derrota: quando o número de vidas é nulo, é solicitado ao utilizador que escolha entre reiniciar o jogo e encerrá-lo.

- coord → Efetua o cálculo da trajetória

  Com a atualização do tempo, é necessário calcular novas coordenadas para o projétil e devolvê-las sob uma forma que simplifique as restantes funções. Ora, daqui decorre a necessidade de aplicar transformações aos valores de (x,y) que esta função devolve, nomeadamente: apresentar y como (23 - y) para que a escrita na janela de texto decorra como esperado e, ainda, agregar ambas as coordenadas num único número de 16 bits. Assim, será este o número devolvido para a representação do projétil e a verificação de condições finais.

- addtime → Adiciona uma constante à variável que mantém o tempo

  Primeiramente, é necessário explictiar que o nosso jogo mantém uma representação do tempo distinta do tempo real: utilizamos o temporizador interno ao processador apenas para deter a execução do programa em pontos críticos. Esta abordagem simplifica o programa, já que o processamento da interrupção do cronómetro só atualiza uma variável de controlo (update). É a função addtime que controla o tempo interno (adiciona uma constante (0.3125s) à variável que detém o tempo (t) e nada mais).

- compare → compara a posição do projétil com as condições terminais

  Durante o voo do projétil é necessário verificar se este colide com determinadas fronteiras: assim, para que o programa tenha o comportamento pretendido, convencionamos que a banana nunca pode ultrapassar (x = 80 e y = 0). Além disso, o voo da banana também é interrompido se ela colidir com qualquer dos gorilas. Deste modo, existe tanto a possibilidade de encerrar o programa com a colisão sobre o gorila adversário, como o começo de uma nova ronda quando a colisão acontece sobre o gorila do jogador. Assim, esta função processa a posição do projétil e conclui se é necessário passar a uma nova ronda ou se o joga termina.

Hugo Pitorro, nº 92478
Cristiano Clemente, nº 92440
Grupo: 34

- **Conclusão**

Durante o desenvolvimento do projeto, a principal conclusão foi a complexidade que reter várias funcionalidades constitui, especialmente numa linguagem low-level como o assembly. A nossa ambição de construir uma solução que oferecesse tanto o modo singleplayer como multiplayer, rapidamente se tornou mais complexa do que esperávamos, especialmente quando o código divergia em duas situações diferentes para cada gamemode. Contudo, esta tentativa não deixou de ser positiva, já que nos alertou para aspetos que ainda não tínhamos sequer considerado.

Por outro lado, a solução singleplayer pela qual optámos não demonstra nenhuns problemas críticos que possam prejudicar a experiência do jogador. Aliás, todas as funcionalidades que decidimos implementar aparentam estar a funcionar perfeitamente pelos testes que realizámos.

Finalmente, temos que a abordagem modular que implementámos oferece uma melhor experiência, especialmente quando nos reportamos a aspetos como a manutenção e a adição de novas funcionalidades.