INTRODUÇÃO À ARQUITETURA DE COMPUTADORES

IST – LEIC

GRUPO 10

ist1106379 - Rafael Pedro Augusto Garcia
ist1106642 - Pedro Ruano Pinto Malta da Silveira
ist1106937 - André Hasse de Oliveira Vital Melo

Manual de utilizador

- A – Começar (e recomeçar) jogo;
- C – Pausar / retomar o jogo;
- E – Terminar o jogo;
- 0 – Lançar uma sonda para a esquerda;
- 1 - Lançar uma sonda em frente;
- 2 - Lançar uma sonda para a direita

Como o código funciona:

Para executar as várias atividades concorrentes (Ex: A evolução dos asteroides no ecrã, o movimento das sondas…) no projeto implementamos processos cooperativos, que funcionam de forma de forma autónoma, mas interagem entre si, criando assim vida à aplicação. Assim para o decorrer do jogo, criamos um processo por cada asteroide e sonda, um processo responsável por monitorizar e alterar a energia da nave, um para animar o painel de instrumentos da nave, um responsável por ler o input do utilizador (teclado) e por fim o próprio programa principal.

- ## Como os processos interagem entre si

A iteração entre processos dá-se através de variáveis globais, mais precisamente estados de jogo, que não passam de constantes que representam estados e são movidas para a memória, no endereço representado por GAME_STATE. Cada processo na sua execução vai periodicamente verificando o estado de jogo e de acordo com o mesmo realiza uma

certa ação (a ação dependo do processo em si e do estado de jogo). A título de exemplo, definimos um estado de jogo LOSS_COLLISION, que quando atualizado na memória indica aos outros processos que o jogo acabou por colisão da nave com um asteroide.

Para este projeto definimos 5 estados de jogo:

- IN_GAME, indica que o jogo está em execução

- PAUSED_GAME, indica que o jogo está pausado

-LOSS_COLLISION, indica que o jogo terminou por colisão da nave com um asteroide

- NO_ENERGY, indica que o jogo terminou porque a nave perdeu a energia

- ENDED_GAME, indica, apenas, que o jogo terminou

Para além dos estados de jogo, os processos também comunicam através da tecla_carregada, como por exemplo, quando o processo dos asteroides deteta uma colisão ele "força" a tecla para terminar o jogo.

• Criação dos processos para os asteroides e sondas

Neste projeto, optou-se por criar um processo por cada asteroide e por cada sonda, de modo, a facilitar a análise do comportamento de cada instância individualmente. Para isso reservou-se um lugar na memória com o tamanho genérica pilha a multiplicar pelo número de instâncias de cada boneco (sonda ou asteroide). Para atribuir o devido espaço a cada instância, na criação de cada processo individualmente, diminui-se o endereço respetivo à pilha de cada instância

Relativamente ao processo dos asteroides:

Para este processo definimos:

Asteroides -> Inicializa a pilha de cada instância, reserva um lugar na memória para guardar a posição de cada instância e a posição genérica inicial dum asteroide

move_asteroide -> verifica se o jogo está a decorrer e se estiver baralha a posição e direção inicial do asteroide

desenha_asteroide -> responsável por animar o asteroide (cada animação vai de acordo com um timer de 400ms). Esta função tem em conta se o asteroide pode colidir com a sonda ou com a nave, através do testa_colisao, e pode forçar o jogo a terminar caso a colisão seja com a nave

direciona_explosao / explosão / consome -> responsáveis pela explosão dum asteroide

reseta_asteroide -> responsável por reiniciar a posição do asteroide

Relativamente ao processo das sondas:

Para este processo definimos:

sondas -> Inicializa a pilha de cada instância, reserva um lugar na memória para guardar a posição de cada instância e a posição inicial respetiva de cada instância de sonda, guarda o comando de disparo respetivo a cada sonda e as direções respetivas de cada sonda

espera_disparo -> verifica se o jogo está a decorrer e espera que o utilizador prima o comando para disparar

desenha_sonda -> responsável por animar a sonda (cada animação vai de acordo com um timer de 200ms)

reinicia_sonda -> reinicia a posição de cada sonda

- **Criação dos processos teclado, energia_nave, desenha_nave**

Os seguintes processos são de instância única, logo mais fácil de inicializar.

Relativamente ao processo do teclado

teclado -> inicializa os periféricos, a máscara (usada para "limpar" bits gerados aleatoriamente do periférico de entrada) e a última linha

espera_tecla / ha_tecla -> funções responsáveis por ler a tecla carregada, atualiza-la numa variável global (espera_tecla) e depois espera que tecla seja desprimida (ha_tecla)

proxima_linha_teclado -> muda a linha do teclado a varrer

Relativamente ao processo energia_nave

energia_nave -> inicializamos os periféricos dos displays

espera_interrupcao / diminuir_unidade -> verificamos se o jogo está a decorrer e a de acordo com o timer de 3000ms atualiza a energia.

set_energia_zero_tres / call_no_energy / change_no_energy -> funções responsáveis por terminar o jogo caso se perca por falta de energia

waits_for_refuell -> Caso se tenha perdido, espera-se que o utilizador recomece o jogo

Relativamente ao processo desenha_nave

desenha_nave / loop_painel / painel_instrumentos -> responsáveis pelas animações do painel de instruções da nave, cada animação vai de acordo com o timer de 300 ms

apaga_nave -> apaga a nave no caso de o jogo terminar

- ## Cenários e sons

Por fim, aumentamos a iteratividade e melhoramos a jogabilidade, colocando-se diferentes cenários e sons para cada situação especifica, tal como, o cenário e sons quando se força o jogo, ou quando se perde por colisão com a nave, ou quando se perde por falta de energia são todos diferentes.

Sugestões:

Uma sugestão para o programa seria um relógio para ver quanto tempo o jogador conseguiu durar no jogo e um possível movimento da nave para a esquerda e para a direita.