Faculdade de Engenharia da Universidade do Porto

Sea Interrupter

Projeto de Laboratório de Computadores

Laboratório de Computadores (L.EIC018) 2023/2024

Licenciatura em Engenharia Informática e Computação

Pedro Alexandre Guimarães Lobo Ferreira Souto (Regente do curso)

Daniel Filipe da Rocha Teixeira (Professor das aulas laboratoriais)

Grupo T16G02:

Inês Ferreira de Almeida up202004513@up.pt

Guilherme Duarte Silva Matos up202208755@up.pt

João Vítor da Costa Ferreira up202208393@up.pt

Maria Eduarda Pacheco Mendes Araújo up202004473@up.pt

Índice

Introdução

No âmbito da unidade curricular de Laboratório de Computadores, foi desenvolvido na linguagem C o projeto “Sea Interrupter”, inspirado no jogo da batalha naval, que usa os controladores dos dispositivos construídos ao longo das aulas laboratoriais.

Cada jogador tem no seu tabuleiro 8 barcos diferentes em posições aleatórias:

- 3 *“destroyers”* (barcos de duas células de tamanho);
- 2 *“submarines”* (barcos de três células de tamanho);
- 2 *“battleships”* (barcos de quatro células de tamanho);
- 1 *“carrier”* (barco de cinco células de tamanho).

Este jogo entre dois jogadores tem como objetivo adivinhar as coordenadas de todos os barcos do adversário primeiro, nomeadamente através de palpites.

Instruções de utilização

Menu

O ponto de entrada do jogo. Aqui o utilizador é apresentado com este ecrã (Fig. 1), dando-lhe as boas-vindas ao jogo. Ao **clicar na tecla Enter** do teclado, o utilizador é redirecionado para a preparação do jogo: a escolha dos barcos.

Figura 1: Menu inicial do jogo

Seleção de barcos

Antes do jogo começar, é necessário posicionar os barcos no tabuleiro do jogador. Neste estágio, o programa seleciona aleatoriamente as posições de cada barco pelo jogador, sendo que aqui o mesmo **não precisa de fazer nada.**

Caso o modo de depuração do projeto esteja ligado, é possível observar no terminal as posições de todos os oito barcos.

Espera do adversário

Aqui, o utilizador é apresentado com este ecrã (Fig. 2), com a intenção de que fique a esperar que o adversário acabe de selecionar todos os seus barcos. Quando isso acontecer, ambos os computadores irão comunicar entre eles quem será o primeiro a jogar.

Tal como anteriormente, aqui o utilizador **apenas precisa de esperar** até que o adversário esteja pronto.

Figura 2: Tela de espera do adversário

Vez do jogador

O jogo começa neste ecrã (Fig. 3). Aqui, o jogador pode:

- Observar os seus barcos no seu tabuleiro e o estado de cada um (se foi acertado ou não);
- Observar no seu tabuleiro os ataques anteriores realizados pelo adversário;
- Observar no tabuleiro do adversário os seus ataques anteriores, incluindo os barcos danificados do oponente;
- Observar no relógio o tempo restante que tem para atacar;
- Atacar uma coordenada do tabuleiro adversário. Basta **arrastar o rato para a posição desejada e clicar com o botão esquerdo.** A seguir, o programa passa a vez ao oponente.

O jogador só tem disponível um minuto para realizar o seu ataque. Caso contrário, o jogo atacará automaticamente em coordenadas aleatórias e passará a vez ao adversário.

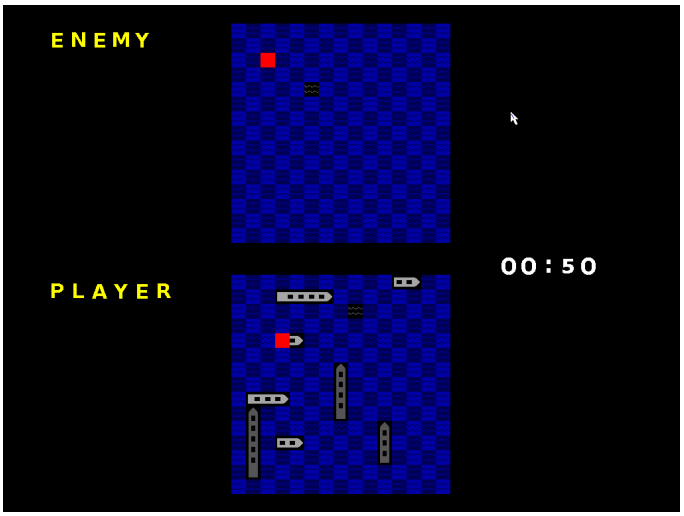


Figura 3: Vez do jogador

Espera da vez

Este jogo é baseado em turnos, portanto muitas das vezes o utilizador vai estar à espera da vez do oponente. Aqui o jogador pode observar tudo o que podia anteriormente, porém o relógio e o rato estão omitidos, sinalizando que não é a sua vez de jogar (Fig. 4). Portanto, o utilizador aqui só precisa de esperar que a vez do adversário acabe.

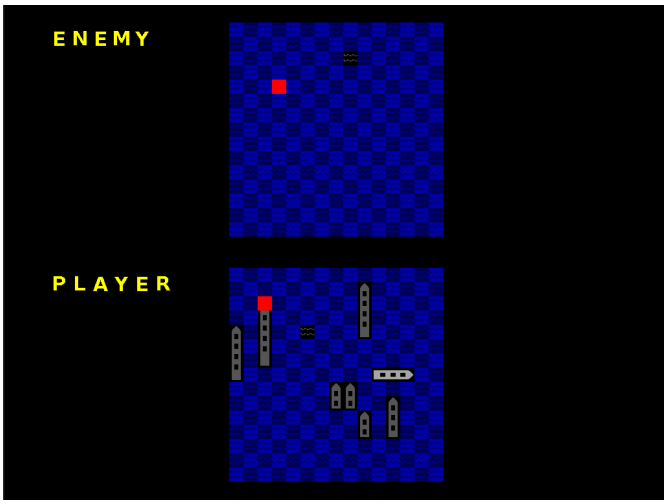


Figura 4: Utilizador à espera do adversário

Fim do jogo

O jogo termina quando um dos jogadores afunda todos os barcos do adversário, momento em que o ecrã de fim de jogo é exibido para indicar o resultado (Fig. 5 e 6).

Clique na **tecla Enter do teclado** para voltar ao menu inicial ou na **tecla Escape** para fechar o programa.

Figura 5: Ecrã de vitória

Figura 6: Ecrã de derrota

Estado do projeto

Funcionalidades implementadas

Funcionalidade	Dispositivos usados					Implementação
	Timer	Teclado	Rato	Placa Gráfica	*UART*	
Menu inicial		X		X		X
Sincronização entre jogadores				X	X	X
Escolha do tabuleiro						*
Atribuição de vez					X	X
Atacar	X		X	X	X	X
Receber ataque		X		X	X	X
Fim do jogo		X		X		X

*Foi implementada, mas não correspondia às expectativas iniciais. Veja as conclusões para mais informações.

Dispositivos I/O utilizados

Dispositivo	Utilidade	Interrupções
Timer	Cortar a vez ao jogador se demorar mais do que um minuto a atacar e controlar o *"framerate"*	X
Teclado	Confirmar a intenção do jogador para mudar de estados	X
Rato	Selecionar posições para atacar	X
Placa Gráfica	Apresentar todos os barcos, ataques e informações ao utilizador pelo ecrã	
UART	Sincronização e envio de informações ao longo do jogo sobre ataques e barcos descobertos.	

Timer

O timer 0 (o único utilizado) é responsável por limitar a frequência de desenho no ecrã por parte da gráfica (chamar as funções contidas na *function table* `STATE_FN[]`) e controlar a duração de um turno (Variável global `TIMEOUTCLOCK` e função `sm_timer()`).

Teclado

Utilizado para registar a intenção de iniciar e sair do jogo. Processa as teclas Enter e Escape, através das funções `sm_keyboard_menu()` e `sm_keyboard_end()` para RET, e `pressed_esc()`, para ESC.

Rato

Utilizado para selecionar a célula a atacar, usando a função `sm_mouse()`.

Regista o botão esquerdo e o movimento do rato.

Placa Gráfica

A placa gráfica desenha no ecrã conforme as funções na *function table* `STATE_FN[]` (`sm_menu()`, `sm_select_boats()`, `sm_waiting_select()`, `sm_my_turn()`, `sm_waiting_turn()`, `sm_end()`), que definem o que desenhar no ecrã com base no estado do jogo.

Fazemos "delta double buffering", onde num segundo "*buffer"* a nova "*frame"* é desenhada na sua integridade, mas esta não é desenhada na sua integridade no "*buffer"* da gráfica: apenas os pixeis que mudaram de valor é que voltam a ser desenhados.

A placa gráfica está no modo indexado com cores padrão do *"Minix"* a uma resolução de 1024x768.

UART / Porta Série

A porta série (`COM1`) é utilizada em varrimento *(polling)* durante o programa. (`sp_poll()`).

A mesma é utilizada apenas para comunicar um *byte* durante as transições de estado. (Envio - `sp_push_byte()` seguido de `sp_poll()`; Receção - `sp_poll()` seguido de `sp_pop_byte()`).

Inicialmente, é utilizada para notificar a presença de outro jogador. A seguir, ambos os lados geram um número ("*byte"*) aleatório, enviam ao outro lado, e cada lado compara o número recebido com o seu próprio: o menor começa primeiro. Caso os números sejam iguais, repetem até poderem decidir quem começa primeiro (`sm_serial_waiting_select()`).

Quando a partida começar, após selecionada uma coordenada, é enviado um "*byte*" que representa a mesma. Este mesmo sinal desencadeia uma mudança de estado em ambos os lados (`sm_serial_my_turn()` e `sm_serial_waiting_turn()`).

Em baixo contém as configurações da porta série: são as *padrão* à execução do programa.

- Bit-rate: 115200;
- 8 bits por pacote;
- 2 stop bits por pacote;
- Sem paridade.

Estrutura de código

Model (20%)

Este módulo é responsável por modelar (agrupar em estruturas de dados e funções relevantes à funcionalidade unitária de cada unidade) a informação respectiva ao jogo.

Inclui funções para inicializar um novo jogo (**`new_game`**), eliminar um jogo (**`delete_game`**), recuperar coordenadas no tabuleiro (**`get_coord`**), adicionar barcos ao tabuleiro do jogo (**`add_boat`**), executar ataques no tabuleiro do inimigo (**`attack`**), lidar com ataques recebidos (**`receive`**) e verificar se o jogo terminou (**`is_over`**). Além disso, uma função de "*debug*" (**`debug_game`**) está incluída para imprimir o estado atual dos tabuleiros e barcos do jogo para fins de depuração.

As principais estruturas de dados definidas no código são:

➔ `Coord:`

Descrição: Representa um quadrado individual no tabuleiro do jogo

Fields:

- **`bool has_boat:`** Indica se o quadrado contém uma parte de um barco.
- **`bool is_hit:`** Indica se o quadrado foi atingido por um ataque.

➔ `Board:`

Descrição: Representa o tabuleiro inteiro do jogo para um jogador ou inimigo.

Fields

- **`Coord board[BOARD_SIZE]:`** Um array de estruturas Coord que representa os quadrados do tabuleiro. O tabuleiro é um array unidimensional, mas conceitualmente representa uma grade bidimensional.

➔ `BoatType (enum):`

Descrição: Enumera os diferentes tipos de barcos com base nos seus tamanhos.

Variants:

- **`carrier = 5:`** Um barco que ocupa 5 quadrados do tabuleiro.
- **`battleship = 4:`** Um barco que ocupa 4 quadrados do tabuleiro.

- **`submarine = 3:`** Um barco que ocupa 3 quadrados do tabuleiro (também referido como *cruiser*)
- **`destroyer = 2:`** Um barco que ocupa 2 quadrados do tabuleiro (também referido como *petrol boat*).

➔ `Boat`

Descrição: Representa um barco no tabuleiro.

Fields:

- **`enum BoatType type:`** O tipo/tamanho do barco.
- **`uint8_t x:`** A coordenada x do canto superior esquerdo do barco.
- **`uint8_t y:`** A coordenada y do canto superior esquerdo do barco.
- **`bool horizontal:`** Indica se o barco está colocado horizontalmente (`true`) ou verticalmente (`false`).

➔ `Game`

Descrição: Contém todas as informações sobre o estado atual do jogo.

Fields:

- **`Board player:`** O tabuleiro do jogo representando os quadrados do jogador.
- **`Board enemy:`** O tabuleiro do jogo representando os quadrados inimigo, incluindo apenas as informações conhecidas pelo jogador (ex: `hits`).
- **`Boat boats[BOAT_COUNT]:`** Uma lista dos barcos do jogador, usado para gerir e desenhar os barcos no tabuleiro.
- **`uint8_t boat_count:`** O número de barcos atualmente colocados no tabuleiro do jogador.

Estas estruturas de dados geram o estado e as interações dentro do jogo, permitindo a inicialização, atualização e consulta do estado do mesmo, bem como a execução da lógica do jogo, como adicionar barcos e realizar ataques.

View (30%)

É responsável por desenhar os elementos gráficos do jogo na tela.

Utiliza várias funções para carregar e desenhar *"sprites"* (imagens) que compõem a *"interface"* do utilizador, incluindo o menu inicial (**`draw_menu`**), o ecrã de espera (**`draw_waiting_start`**), o estado do jogo durante a jogada (**`draw_turn`**), o ecrã de fim

de jogo (**`draw_end`**) e o cursor do rato (**`draw_mouse`**). Além disso, este módulo desenha o tabuleiro do jogo (**`draw_board`**), as legendas dos jogadores (**`draw_player_string`** e **`draw_enemy_string`**), o relógio (**`draw_clock`**) e os barcos (**`draw_boat`**), além de indicar acertos (**`draw_hit`**) e erros (**`draw_miss`**).

Controller (50%)

Implementa a lógica de controlo e estado do jogo, gerindo a interação entre o teclado, o rato, o *timer* e a porta série para coordenar as jogadas dos jogadores.

O *Controller* define várias funções de máquina de estados para gerir as diferentes fases do jogo:

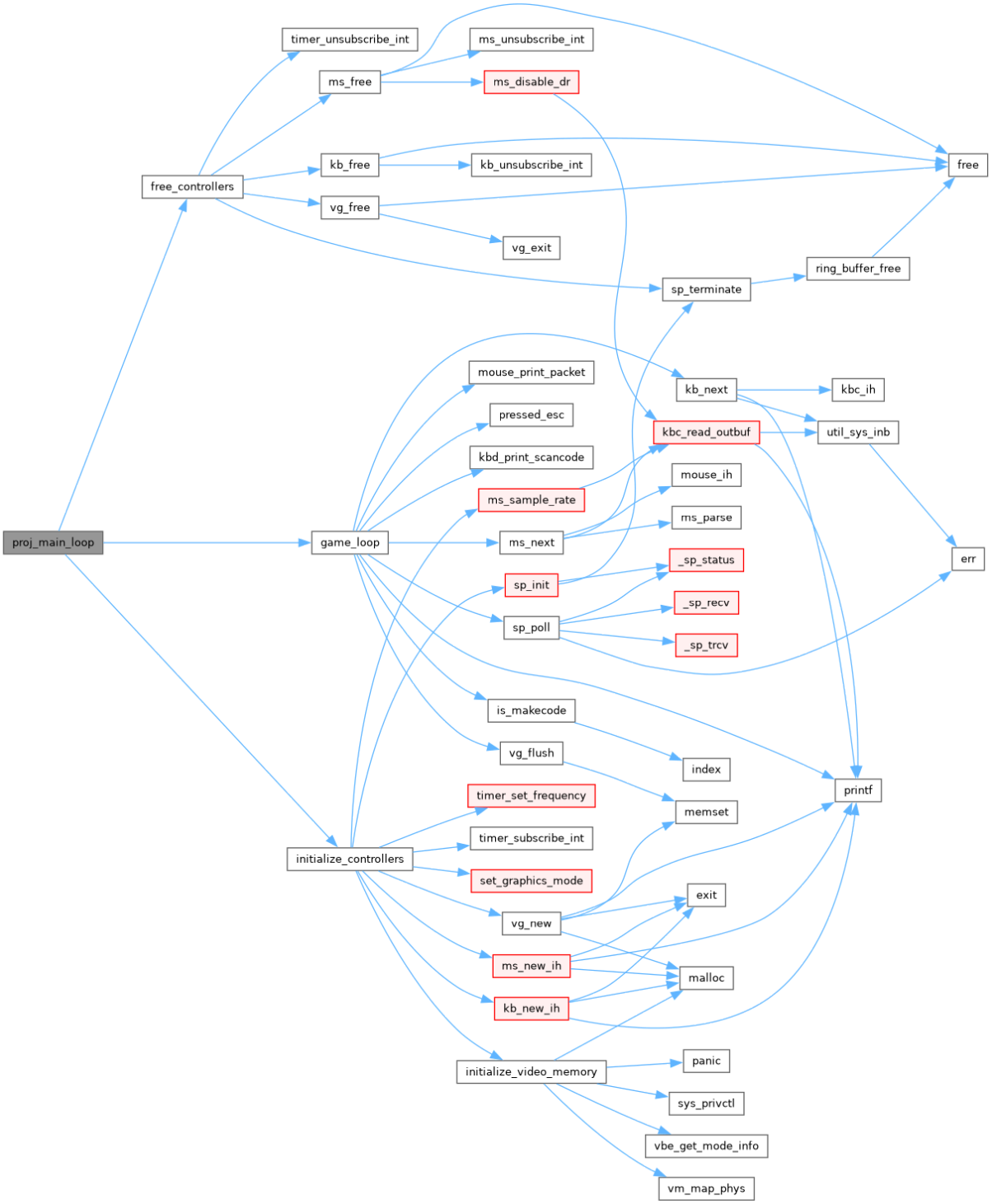
- Menu
- Seleção de barcos
- Espera do adversário
- Vez do jogador
- Espera pela vez do adversário
- Fim de jogo

Cada função de estado é responsável por desenhar os elementos visuais adequados no ecrã e processar eventos específicos, como entradas do teclado, movimentos do rato e atualizações do temporizador.

Para mais informações sobre a máquina de estados e a sua implementação em C, veja o capítulo “Detalhes de implementação”.

Além disso, o módulo contém funções para inicializar (**`initialize_controllers`**) e libertar (**`free_controllers`***)* os controladores necessários para o funcionamento do jogo, e um ciclo principal (**`game_loop`**) que executa continuamente até que o utilizador saia, ou o jogo acabe, processando interrupções e mudanças de estado conforme estas ocorrem.

Grafo de chamada de funções



Apesar de omitido no grafo, a função `game_loop()` é a única função que chama `driver_receive()`, isto é, que está encarregue de tratar de todas as interrupções.

Detalhes de implementação

Máquina de estados do programa

O controlo do projeto inteiro está em torno da máquina de estados presente no `controller.c`. Esta máquina contém 6 estados, tal como descrito nas instruções de utilização:

```C/C++
typedef enum {
  Menu,
  SelectBoats,
  WaitingSelect,
  MyTurn,
  WaitingTurn,
  End
} GameState;
```

A implementação modular desta máquina de estados consiste em associar a cada dispositivo uma lista de funções e executar a função adequada conforme o estado atual. Por exemplo, no `controller.c` está presente esta lista:

```C/C++
void (*const TIMER_FN[])(void) = {
  [Menu] = no_op,
  [SelectBoats] = no_op,
  [WaitingSelect] = no_op,
  [MyTurn] = sm_timer,
  [WaitingTurn] = no_op,
  [End] = no_op,
};
```

Esta lista está associada ao relógio, que no nosso projeto, só é útil quando é a vez do utilizador a jogar para decrementar o cronómetro, pelo que chama a função `sm_timer()` nesse cenário e a função vazia `no_op()` em todos os outros estados.

Esta modularidade torna extremamente fácil modificar a máquina de estados a qualquer momento, bem como executar as funções desejadas no momento certo: basta “chamar” a lista de funções, sendo o `GameState` como o índice. No projeto, realizamos estas ações após a interrupção do respetivo dispositivo.

Modo de depuração

O projeto disponibiliza a *“flag”* de compilação `DEBUG` que ativa muitas mensagens no terminal relacionadas com a lógica da máquina de estados e do projeto em si, nomeadamente:

- Mudanças de estado da máquina;
- Pacotes do rato e do teclado;
- Posição dos 8 barcos no estado `SelectBoats`;
- Conteúdo do pacote recebido em cada rodada do jogo;
- Erros de funções que falharam.

Por padrão, o projeto tem esta funcionalidade desativada, porém basta retirar o “`#`” da seguinte linha no `Makefile`:

```
Unset
# CFLAGS += -DDEBUG                # Extra debug messages from the project
```

Conclusões

Este projeto teve alguns atrasos que resultaram em algumas funcionalidades não serem implementadas por completo ou serem substituídas por outras mais fracas.

Um exemplo disso é o estado `SelectBoats`. A ideia original apresentada na proposta do projeto foi dar a possibilidade ao utilizador de arrastar com o rato os oito barcos para as posições do tabuleiro que desejasse. O utilizador poderia até rodar os barcos usando o teclado para que o tabuleiro inicial refletisse a mente do jogador. Infelizmente, devido à falta de tempo e de recursos, essa funcionalidade teve de ser cancelada e substituída por um simples algoritmo que decide pelo jogador as posições dos barcos.

Apesar disto tudo, o projeto tornou-se num simples e conciso exemplo de uma aplicação real das bibliotecas desenvolvidas nos laboratórios, com uma programação em C estruturada e de baixo nível.