Faculdade de Engenharia da Universidade do Porto

Relatório Final

Projeto Prático

Laboratório de Computadores
MIEIC 2º Ano

Professores:
Pedro Alexandre Guimarães Lobo Ferreira Souto

Estudantes & Autores:
Bruno Madeira up201306619@fe.up.pt
João de Almeida ei12053@fe.up.pt

2014/2015

Exposição do trabalho prático desenvolvido.

Conteúdo

Instruções de Utilização

Iniciar a aplicação

O jogo pode ser iniciado em 3 modos diferentes: “um jogador”, “host e “client”.

Os dois últimos modos devem ser usados para partidas com dois jogadores em simultâneo. Para jogar uma partida multijogador deve chamar o programa com um primeiro argumento com o valor de 1 para modo “host” ou de 2 para modo “client”*. Para correr em modo “host”, por exemplo, deve ser dado o *input*: <<service run `pwd`/proj –args “1”>> . O jogador com o modo “client” deve iniciar a sua aplicação primeiro, antes que o jogador 2 inicie a sua para que o jogo inicie correctamente.

*Os nomes associados aos modos podem sugerir erradamente a maneira como funcionam.

O modo “um jogador” será executado chamando o programa sem argumentos ou com argumentos que não correspondam à utilização dos modos referidos anteriormente.

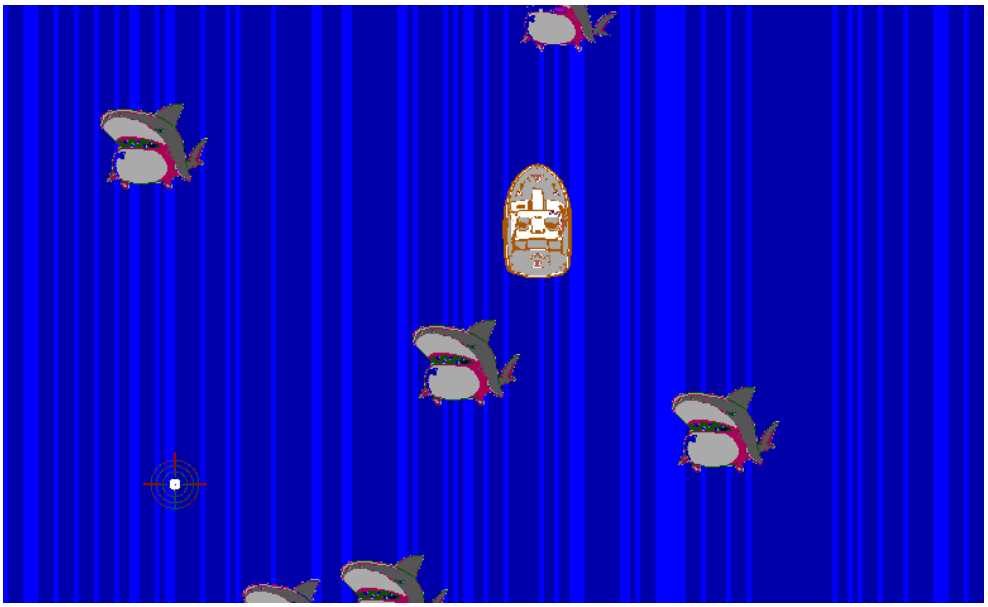
Modo Jogo

No modo jogo o utilizador tem controlo sobre um barco que deve manter fora do contacto dos tubarões o maior tempo possível. No modo multijogador é possível observar a posição do barco do outro jogador. Este barco aparece como sendo um “fantasma” não fazendo alterações no jogo. Seguidamente são descritos os diversos componentes do jogo.

Tubarões: aparecem com o passar do tempo no cenário de jogo. Movimentação pelo mapa de jogo.

Barco: pode ser deslocado em 4 direções básicas fazendo uso das teclas W, A, S, D para efectuar respectivamente uma movimentação para cima, esquerda, baixo ou direita. Também podem ser realizados movimentos em diagonal combinando movimentação horizontal com vertical. Movimentos diagonais não conferem maior velocidade ao barco, isto é, o módulo do vector de deslocamento permanece sempre constante.

Mira: de modo a auxiliar a tarefa, o jogador pode ainda controlar uma mira utilizando o rato. Com esta pode disparar contra os tubarões. O tubarão que for apanhado na mira pode ser eliminado com um clique do botão esquerdo do rato.



Modo GameOver

Quando o barco entrar em contacto com um tubarão o jogo termina e é mostrado o ecrã de fim de jogo. Neste ecrã pode seleccionar com o rato a opção de reiniciar o jogo disparando com a mira no símbolo de refresh (símbolo mais acima na imagem) ou sair do jogo disparando no botão de desligar (mais abaixo na imagem). No modo multijogador não é possível reiniciar o jogo directamente, sendo necessário voltar a correr o programa.

Abortar

O utilizador pode abortar a aplicação a aplicação a qualquer momento fazendo uso da tecla escape.

Estado final do projeto

Funcionalidades não implementadas

Sistema de pontuação e complexidade do jogo

Priorizamos a implementação e o melhoramento do modo multiplayer à criação de alguns detalhes da aplicação. Assim sendo, não chegou a ser definido nenhum sistema de pontuação nem foram implementadas as sardinhas e os tiros referidos nas especificações. Estes pontos não são particularmente difíceis de implementar com a estrutura do projeto actual e consideramos que seriam menos valorizados e muito menos interessantes de implementar que o modo multijogador com a porta de série. Algumas funcionalidades do modo multijogador também ficaram por implementar, como o eliminar dos tubarões, sendo apenas passada a posição do barco. Apesar de serem relativamente fáceis de adicionar no estado actual do projeto não foram consideradas prioritárias.

Actualização do jogo

Por razões semelhantes à referida no parágrafo anterior não exploramos o uso do RTC ao máximo. Este é usado apenas para verificar a data sendo que para controlar a frequência das actualizações do jogo utilizam-se interrupções do timer. Como por vezes o desenho da imagem é demorada, o que faz com que se ignore uma interrupção, aumentamos a frequência do timer para tentar assegurar que se desenhe na tela à frequência pretendida. Como nos foi referido pelo docente esta implementação não é correta sendo que o uso do RTC seria preferível.

Robustez do modo multijogador

Funcionalidades relativas à uart não foram bem concebidas. Apesar do protocolo de comunicação entre as aplicações ter sido bem definido existem alguns pontos na aplicação que não são muito robustos. Isso é visível no facto do modo "client" ter que ser obrigatoriamente o primeiro a ser iniciado. Um dos problemas detectados, ao qual não se chegou a implementar solução, foi que ao correr o jogo uma segunda vez o seu início podia ocorrer erradamente quando o "client" termina o programa antes do "host".

Dispositivos de entrada e saída usados

DISPOSITIVO	UTILIZAÇÃO	MODO DE UTILIZAÇÃO
TIMER	Controlar frequência da lógica de jogo e do desenho	INTERRUPÇÃO
MOUSE	Movimentação e disparo da mira (cursor)	INTERRUPÇÃO
KBD	Movimentação do barco no jogo	INTERRUPÇÃO
VIDEO CARD	Desenhar o ambiente de jogo	"POLLING" (sincronizado com interrupções do timer)
RTC	Gerar a "seed" (aleatoriedade do jogo) através da data	SINGLE CALL
UART	Modo multi-jogador	POLLING

TIMER

Altera-se a sua frequência para o dobro e subscreve-se a interrupções periódicas (por tick). A alteração da frequência visa solucionar o possível atraso no desenho do jogo por uma via que não enveredasse o uso de RTC. O seu uso, tal como referido pelo docente, não

é por isso o mais correto mas permite obter o comportamento desejado na aplicação. Implementado no módulo timer.c.

MOUSE

Usado com o *stream mode enabled* à semelhança do lab4 é usado para leitura de movimentos do rato e para verificar quando o botão esquerdo é pressionado.
Implementado no módulo mouse.c que vai passando para o código de jogo os valores de movimento do rato a serem somados consecutivamente até à próxima actualização da lógica jogo. Ocorrendo a actualização da lógica o somatório dos deltas volta a zero.
O pressionar do botão esquerdo é um pouco menos preciso pois não se realiza nenhuma contagem do número de vezes em que o botão alterna entre pressionado e não pressionado, entre actualizações da lógica de jogo. É teoricamente possível que um ou múltiplos cliques passem despercebidos no jogo. Como na prática a aplicação sempre realizou o que era esperado consideramos que a solução implementada era aceitável, contudo estamos cientes que não é a implementação mais correta.

KBD

Responsável pela leitura de *scan codes*, implementado no módulo kbd.c. Permite à aplicação verificar o último *scan code* lido mas o pressionar e largar de teclas não é verificado directamente a partir do módulo implementado. Os *scan codes* são usados para actualizar o estado de teclado guardado em keyStates que é depois usado para esse fim.

VIDEO CARD

Implementada nos módulos graphics.c, vídeo_gr e vbe.c realizados para o *lab5*. À semelhança do *lab* é usado o modo 0x105.

RTC

Implementado no módulo rtc.c. Vai buscar a data de sistema e utiliza-a para gerar a "seed" da função *random* da *stdlib*.

UART

Foram implementadas as funções básicas necessárias para o envio/recepção de dados através da porta de série e sua alteração de configuração. Optamos por não usar o sistema FIFO e o modo de utilização por interrupção pois estes exigiam maior tempo dispensado e não traziam melhorias significativas ao nosso projeto. Uma vez que a porta de série é simulada por sockets, apenas o número de bits por *char* é relevante e tem que ser igual. De qualquer das maneiras a configuração usada é: 8 bits por char, 2 stop bits, sem controlo de paridade e uma taxa de 9600 bit/s .

Protocolo de baixo nível

A recepção da informação para o jogo na porta de série inicia-se ao reconhecer o caractere "S" que indica o início de um "pacote" (conjunto de dados) a ser recebido. São guardados todos os dados recebidos até se encontrar o caractere "." duas vezes seguidas (que indica que se leu duas vezes o mesmo valor pois não há nenhum valor que possa ser representado com dois pontos no protocolo usado).

Comprimir informação

No modo jogo a posição do barco em *x* e em *y* nunca tem valores maiores que 12 bits apesar de serem do tipo integer. Assim para partilhar a posição são enviados somente três bytes seguidos em que um deles apresenta 4 bits para x e 4 bits para y em vez de serem enviados 4 bytes.

Organização de código

Apresentamos seguidamente os módulos implementados numa tabela com a contribuição de cada elemento na sua realização. Seguidamente à tabela é dado um resumo dos módulos que ainda não foram sido referidos na secção de dispositivos. Informação adicional sobre estruturas, funções, macros e variáveis pode ser consultado na pasta html que contem a documentação gerada pelo Doxygen.

Tabela de módulos implementados e contribuição de cada elemento

MODULO	RESPONSAVÉL(EIS)	CONTRIBUIÇÃO DE CADA ELEMENTO (%)		PESO NO PROJETO (%)
		BRUNO	JOAO	
boat.c	BRUNO	90	10	2
buffer.c	feito no lab 5	pequeno hack para random background	uso de memcopy	2
collision.c	BRUNO	100	0	1
data_pixmap.c	JOAO	0	100	1
EndScreen.c	BRUNO	0	100	1
GameController.c	BRUNO	50	50	9
graphics.c	feito no lab 5	0	0	10
kbd.c	feito no lab 3	modificações mínimas	0	10
keyStates.c	JOAO	100	0	1
lab5.c	BRUNO/JOAO	50	50	1
mouse.c	feito no lab 4	0	0	7
multiplayerSync.c	BRUNO	100	0	5
pixmap.c	feito no lab 5	modificações mínimas	0	1
rtc.c	BRUNO/JOAO	20	80	1
scope.c	BRUNO	0	100	2
shark.c	BRUNO	100	0	3
smart_vector.h	BRUNO	100	0	4
sprite.c	JOAO	100	0	2
test7.c	BRUNO/JOAO	25	75	10
timer.c	feito no lab 2	0	modificações mínimas	10
uart.h	JOAO	0	100	5
Utilities.h	BRUNO/JOAO	60	40	4
vbe.c	feito no lab 5	0	0	3
vector2.c	BRUNO	100	0	2
video_gr.c	feito no lab 5	0	0	3

Breve Descrição de cada módulo

boat.c: Todas as relacionadas com o barco que aparece no jogo.

buffer.c: Todas as relacionadas com o uso de double buffering.

collision.c: Detecção de colisão usando círculos.

data_pixmap.c: Contem *xpm*s a usados nos sprites do jogo.

EndScreen.c: "Menu" de fim de jogo.

GameController.c: Elemento central responsável pela gestão de recursos, sincronização, actualização e desenho do jogo a mais alto nível. Faz uso das diversas funcionalidades implementadas nos outros módulos para implementar o jogo.

keyStates.c: Responsável por guardar o estado actual e anterior do teclado. Ajuda a abstrair dos dados do kbd.c permitindo verificar o pressionar/largar de teclas de interesse mais facilmente.

lab5.c: Inclui função (*main*) principal e verifica os argumentos recebidos de modo a iniciar o modo de jogo correto.

multiplayerSync.c: Sincronização do jogo para o modo multi-jogador. Responsável por estabelecer e usar o protocolo de comunicação entre as duas aplicações de jogo.

pixmap.c: Criação e gestão de pixmaps.

scope.c: Todas as relacionadas com a mira/cursor do jogo.

shark.c: Todas as relacionadas com os tubarões que aparecem no jogo.

smart_vector.h: Não é um ficheiro (.c) uma vez que a implementação foi realizada com base em macros. Implementa uma estrutura semelhante aos vectores genéricos usados em c++. Usado em diversos módulos do projeto.

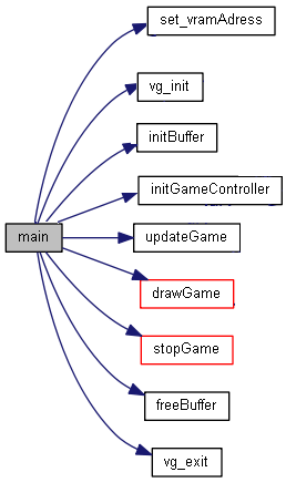
sprite.c: Todas as relacionadas com o uso de sprites. Contem as estruturas Sprite, StaticSprite e AnimatedSprite sendo que em termos práticos Sprite funciona como se fosse uma classe abstracta pai (analogamente a c++) das outras duas.

Utilities.c: Funções de genéricas, funções de debug e handler genérico.

vector2.c: Estrutura e funções para o uso de vectores de duas dimensões. Facilitam a implementação e leitura da lógica de jogo.

Diagrama de chamadas de funções

Foram gerados gráficos de chamadas de funções fazendo uso do Doxygen e do GraphViz que estão disponíveis na documentação gerada pelo Doxygen na pasta html. Dada a complexidade dos gráficos é preferível a sua visualização em conjunto com o resto da documentação gerada uma vez que esta permite consultar as diversas funções e módulos de forma contextualizada. Expõe-se seguidamente uma versão bastante simplificada da função *main* para o modo de um jogador apenas para ilustrar como está organizado o projeto a um nível mais abstracto.



Através deste modelo simplificado pretende-se ilustrar como a aplicação foi separada essencialmente em três segmentos integrais. A criação do jogo, o executar do jogo e o fim do jogo. O primeiro visa subscrever aos dispositivos usados, carregar imagens e alocar memória necessária. O segundo trata das interrupções, lógica de jogo e sincronização do desenho no ecrã pela ordem mencionada. Por fim cancelam-se as subscrições e liberta-se a memória usada pela aplicação. Na documentação gerada pode ser verificada o diagrama completo para a função *main*.

Detalhes de Implementação

Estruturação de código

No projeto, um dos pontos que achamos que ficou bem concebido foi a estruturação do código.

Inicialmente demorou-se um pouco mais a desenvolver código "auxiliar" ao projeto principal como o caso dos módulos vector2, smart_vector, sprites e Utilities o que levou a uma maior facilidade no desenvolvimento da aplicação mais tarde por permitir abstrairmo-nos de certos tipos de operações/estruturas e por facilitar a organização do código. Alguns destes módulos são bastante genéricos e/ou independentes do projeto podendo ser usados no desenvolvimento de outras aplicações pelo que se tentou deixá-los bem documentados.

Ainda relativamente à estruturação de código a abordagem que tivemos face aos diversos "objectos" (barco, tubarões e mira) que aparecem no jogo assemelha-se um pouco à programação orientada a objectos. Cada "objecto" é implementado num módulo que tem as funcionalidades de criar o objecto e de libertar a memória usada por este á semelhança dos construtores e destrutores de classe em c++. Esta abordagem facilita a gestão de memória que pode ser problemática em códigos mais complexos. Também facilita o desenvolvimento por não existirem dependências indirectas com outros "objectos" que possam causar comportamentos não desejado. Isto é, as funções implementadas no módulo de "objecto" não acedem a parâmetros ou funções de um módulo de outro "objecto", semelhante ao encapsulamento que se efectua nas linguagens orientadas a objectos.

Como sugerido pelo docente, a implementação do código da aplicação está a numa "camada de abstração" separado da de interacção com os dispositivos, sendo por vezes usada uma camada intermédia como é o caso do módulo keyStates e do multiplayerSync.

Protocolo de comunicação da aplicação no modo multijogador

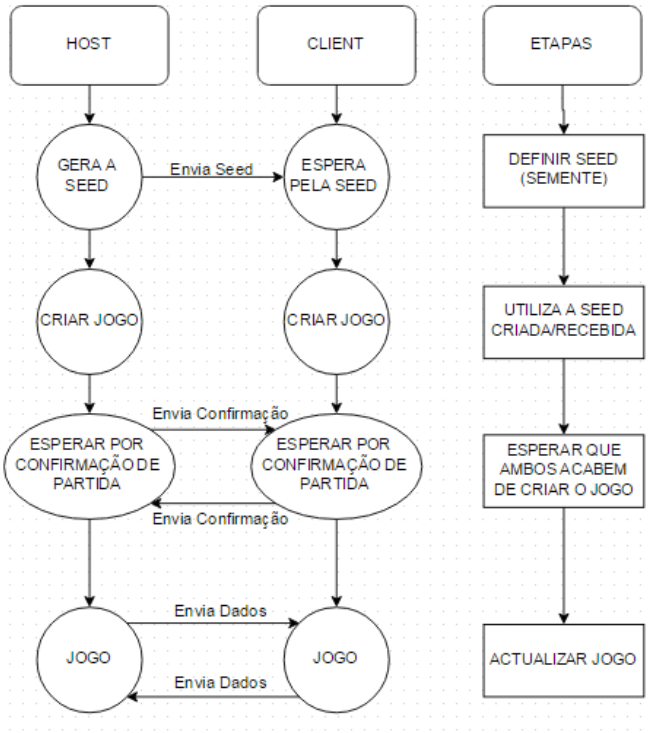
O tipo de comunicação estabelecida é do tipo *Half-duplex*. No modo jogo tanto o "client" como o "host" esperam por informação sobre o estado do outro jogo e respondem seguidamente com informação sobre o seu estado seguindo um padrão de "recepção-envio" de mensagens. Este padrão é iniciado pelo "host" que envia a primeira mensagem. Por vezes o "host" poderá sair do padrão "recepção-envio" quando não obtiver resposta do "client" durante o período tempo prolongado, de modo a tentar reiniciar o padrão.

Como referido de passagem no capítulo de relativo à utilização do programa o nome "client" e "host" podem sugerir um protocolo diferente do implementado. O "host" e o "client" recebem e enviam exactamente o mesmo tipo de informação no modo jogo. Não é usado o padrão em que o jogo é processado somente no "host" (hospedeiro), sendo devolvido o estado actual ao cliente. O jogo corre em ambas aplicações de igual forma. Diferem apenas no facto do "host" ser responsável por reiniciar o padrão (referido no paragrafo anterior) e por gerar e enviar a *seed* (semente) antes do início do jogo.

A *seed* responsabiliza-se por tornar os dois jogos "iguais", mas atenção, o saltar de um ciclo de actualização ou erros ocasionais podem quebrar o suposto sincronismo. Tal problema verificou-se quando se fazia uso da função *rand()* no desenho do background. Ocorreu devido a ocasionalmente uma das aplicações saltar um desenho, consequência do computador usado para testar não conseguir correr duas máquinas virtuais à mesma velocidade constantemente. Este problema podia ser agravado se tivessem de ser utilizadas decisões específicas ao longo do decorrer do jogo, como o uso de *AI* (inteligência artificial) por exemplo. Como neste projeto em particular o jogo é simples podemos usar o sistema simplificado que implementamos.

Após a criação e partilha da *seed* (semente) e antes de começar o jogo é necessário sincronizar o início do jogo para ambos os jogadores.

A imagem abaixo resume o processo referido.



Avaliações

Avaliação da Cadeira

Uma vez que este grupo contém um elemento que já participou na cadeira do ano passado torna-se mais fácil uma avaliação por comparação. A cadeira este ano está bastante mais acessível no que diz respeito ao limite de tempo que temos para realizar os sucessivos trabalhos de laboratório comparativamente ao ano passado, bem como, os laboratórios mais difíceis, nomeadamente o UART, são agora de caráter opcional e não obrigatório como em anos anteriores. Este último penso que é benéfico para todos, os alunos que a quem suscitam mais dificuldades ao decorrer da cadeira podem agora "terminar" a cadeira não sendo penalizados pela nota de um trabalho de laboratório e mesmo assim tirarem uma nota razoavelmente boa no fim. Já os alunos mais ambiciosos e dedicados podem tirar uma melhor nota à cadeira tendo apenas que realizar os laboratórios opcionais e utilizá-los. Em suma, a cadeira está bastante bem mais acessível que os anos anteriores (nomeadamente o ano passado) mas não deixa de ter a sua dificuldade e custo de tempo associado.

Auto-Avaliação

Devido à dinâmica de equipa algumas funcionalidades acabaram por ser implementadas pelo elemento que não era responsável por elas. Assim equilibrou-se o trabalho "re-delegando" tarefas dinamicamente à medida que o projeto ia evoluindo.

Bruno: "Penso que em o trabalho foi relativamente bem dividido sendo que trabalhei mais especificamente nas funcionalidades do jogo e menos na implementação dos periféricos a "baixo nível" no caso do *rtc* e *uart*. Acho que ambos contribuímos um pouco em quase todos os segmentos de código apesar de nem sempre estarem à nossa responsabilidade. Mesmo quando não se escreveu código foram analisados segmentos de código em busca de possíveis erros e ocasionalmente feitas pequenas correcções e melhoramentos."

João: "O trabalho foi bem distribuído por ambos os membros do grupo, tendo eu ficado mais focado na implementação e adaptação dos periféricos para o projeto. Ao longo do projecto foram feitas sucessivas alterações ao plano que inicialmente estava definido e por vezes até entrei em áreas que no início não estava designado a entrar e o mesmo aconteceu com o meu colega. Este último ponto prova que eu e o meu colega de grupo tivemos a capacidade de nos adaptar à medida que eram necessárias alterações no plano de desenvolvimento do projeto."

Instruções de Instalação/Execução

Para a instalação apenas é necessário incluir a liblm.a, à semelhança do *lab5*, antes de realizar o comando *make*.

A execução deve ser realizada em modo privilegiado (no root) após colocar as permissões do programa na pasta confd.