**Faculdade de Engenharia da Universidade do Porto**

**Mestrado Integrado em Engenharia Informática e Computação**


# Relatório do Projeto

# "Age Of War"

**Laboratório de Computadores – 2016/2017 – 1ºSemestre**

Turma 4, grupo 9

**Trabalho realizado por:**

Diogo Luís Rey Torres up201506428@fe.up.pt

Rui Emanuel Cabral de Almeida Quaresma up201503005@fe.up.pt

Data de entrega do Relatório: 2 de janeiro de 2017

# Índice

# 1. Introdução

O nosso projeto, realizado no âmbito da unidade curricular Laboratório de computadores do curso Mestrado Integrado em Engenharia Informática e Computação, baseia-se no jogo *"Defender a Torre"*, um jogo em *flash*, encontrado facilmente em *Max Games*, criado por *Louissi*.

Dado os objetivos desta unidade curricular, propusemo-nos a realizar uma versão do jogo, adaptando este de modo a utilizar os diferentes periféricos mencionados ao longo do semestre e o software de desenvolvimento da unidade curricular (como sistema operativo, Minix, e como linguagem de programação, C, e claro, adicionando novas funcionalidades.

O jogo em si, desenvolve-se ao longo de diversas eras, podendo ao longo destas o utilizador comprar soldados e também armamento de defesa, assim como usar ataques especiais. Assim, é permitido ao utilizador criar a sua própria estratégia de jogo invocando soldados e defesas de diferentes tipos, que vão evoluindo ao longo das eras, com o objetivo final de destruir a torre inimiga.

# 2. Age Of War

## *2.1 O jogo*

Na nossa versão do jogo, é permitido dois utilizadores jogarem um contra o outro, utilizando diversas teclas do teclado para invocar as diversas funções (Anexo II).

O objetivo do jogo é destruir a torre adversária e defender a sua.

Em todo o jogo, a criação de soldados e de defesas exigem um gasto de certa quantidade de dinheiro dependendo do tipo de cada um. Estes custos aumentam ao longo das eras, em que uma unidade é mais cara e mais forte do que na era anterior. Os jogadores com a morte das tropas inimigas ganham uma certa experiencia e dinheiro, que utilizam posteriormente para passar para a era seguinte e para adquirir novos elementos, respetivamente.

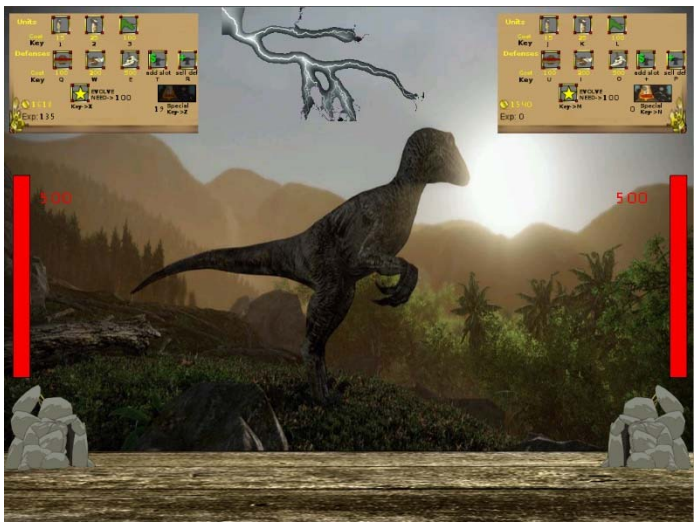
Cada jogador tem a tarefa de criar as unidades, defesas, espaços para construir mais defesas e também de usar o ataque especial, sendo que as unidades criadas irão avançar em direção à torre inimiga e batalhando com unidades inimigas de forma automática.

Os jogadores poderão implementar diferentes estratégias, dado que existem 3 tipos de unidades: um de curto alcance mais barato e consequentemente mais fraco, que poderá ser usado em situações de aperto; um de longo alcance, que dispara a partir do momento em que uma unidade inimiga entra dentro do seu alcance; um de curto alcance mas mais forte que os anteriores e também mais caro. Tendo em consideração que quando as tropas de longo alcance estiverem a disparar estas não se irão mover, recomenda-se aos jogadores que lancem de seguida unidades de curto alcance, para que estas passem a frente das de longo alcance e vão lutar para a frente de batalha.

Quanto às defesas, existe um máximo de 4 espaços para criar defesas. No começo do jogo inicia-se com um espaço destes, sendo que custará 1000 de ouro para a obtenção de cada um dos espaços seguintes. Os jogadores podem ainda vender defesas para permitir a criação de outras no seu lugar.

No total o jogador tem ao seu dispor de 15 diferentes soldados e 15 diferentes defesas, 3 por cada era. Perfazendo, um total de 5 eras(Anexo III).

O score final do jogador dependerá da experiência obtida durante o jogo.



## 2.2 Instruções de Utilização

Antes de iniciar o jogo, é necessário fazer algumas configurações primeiro que estão explicadas no Anexo I, deste relatório.

Ao iniciar o jogo, é apresentado um menu inicial com o nome do projeto Age Of War, e quatro opções que permite ao utilizador selecionar com o uso do botão esquerdo do rato.

### Main menu

Este menu disponibiliza o acesso ao jogo, assim como aos High Scores, aos Créditos e ao botão de saída. Como apresentado na seguinte imagem:

### High Scores menu

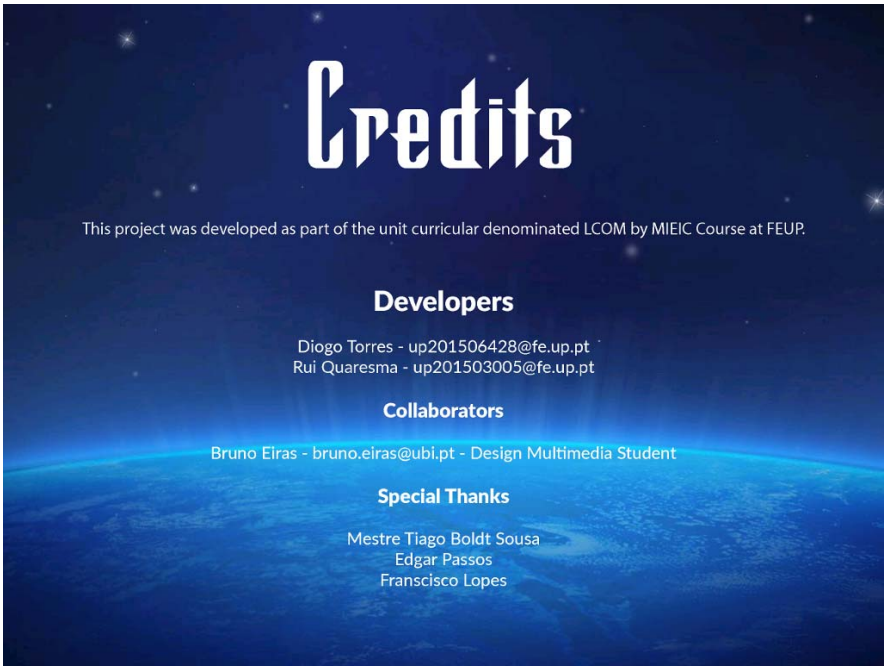
Neste menu são apresentados os resultados, bem como o nome do jogador e a respetiva pontuação.

Para sair deste menu basta clicar na tecla “ESC”.

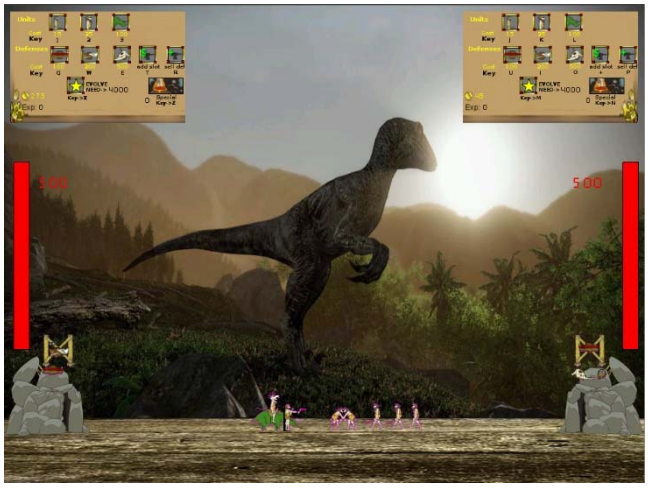

### Credits menu

Neste menu são apresentados os créditos, onde é possível destacar o âmbito do projeto, assim como os seus intervenientes de forma direta ou indireta.

Para sair deste menu basta clicar na tecla “ESC”.



## Modo de jogo



No canto superior esquerdo e direito são apresentados os menus de jogo, em que informa ao utilizador as ferramentas necessárias (teclas necessárias) para poder jogar. Podemos também neste menu, observar a quantidade de dinheiro e experiência adquirida ao longo do tempo assim e se o ataque especial está disponível ou não para utilizar.

No canto inferior esquerdo e direito encontram-se as torres de defesa com a respetiva vida na barra a vermelho acima destas, e entre elas será a zona de movimentação das tropas. Cada jogador poderá ter no máximo 10 tropas no terreno.

Ao longo do tempo cada jogador ganha uma certa quantidade de dinheiro.

Com a morte das unidades inimigas o jogador ganha dinheiro e experiencia que lhe permitirá criar novas unidades, defesas… e evoluir de nível (após acumular suficiente experiencia para isso).

Passando de nível o jogador terá acesso a novas unidades e defesas, para além do aumento da vida da sua torre.

Cada jogador tem ainda um ataque especial para usar, que se recarrega em 30 segundos.

Por fim, temos um fundo temático para cada era(nível).

## Save score menu

Neste menu os jogadores após termino do jogo podem registar o seu nome e score.

Para registar o nome, o jogador apenas necessita de escrever o seu nome no máximo de 10 letras e terminar com o numero 0. Para depois o segundo jogador, poder fazer o mesmo.

### Pause menu

A partir deste menu, o jogador pode voltar ao jogo clicando em “Resume” ou terminar o jogo atual e voltar ao menu principal clicando em “Back To Main Menu”.

## 2.3 Estado do projeto

Na especificação do projeto, pretendíamos implementar a serial porter, mas infelizmente, por questões técnicas não foi possível a sua implementação.

Por outro lado, os restantes periféricos foram implementados com sucesso. Por isso, o projeto encontra-se concluído e com todas as funcionalidades propostas testadas e funcionais.

| Periférico | Utilização | Interrupções | Implementado |
|---|---|---|---|
| Timer | Controlar os frames<br>Tempo de criação de unidades e sua movimentação<br>Tempo de carregamento do ataque de especial<br>Atualizar todo o jogo. | Sim | Sim |
| Keyboard | Criação de unidades, defesas, ataque especial, etc<br>Gravar o nome nos scores no final de cada jogo<br>Saída de certos menus | Sim | Sim |
| Mouse | Navegar no menu inicial e no menu pausa. | Sim | Sim |
| Graphics | Apresentação de toda a estrutura do jogo, desde fundos, torres, unidades defesas… | Não | Sim |
| RTC | Atualizar a data e hora no menu Créditos. | Não | Sim |
| Serial Porter | ---------- | ---------- | Não |

**Timer** – O timer 0 é utilizado maioritariamente para controlar a taxa de refrescamento do ecrã, contando o numero de vezes que este envia uma interrupção. No entanto, também é utilizado para o tempo de criação de unidades e a sua animação assim como para o tempo de carregar o ataque especial. Sendo que para isso recorre ao ciclo de interrupções presente no ficheiro AgeOfWar.c, na função interruptHandler. Este periférico tem as suas funções implementadas em timer.c e timer.h.

**Teclado** – O teclado é usado maioritariamente como leitor de scancodes, e durante o projeto este apresenta-se em duas situações cruciais. A primeira diz respeito ao jogo em si, em que o utilizador controla toda a dinâmica do jogo(Instruções de utilização – Anexo II). A segunda é o input de texto utilizado para registar o nome do jogador. Em ambas as situações a receção do input é feita em AgeOfWar.c na função interruptHandler, sendo posteriormente

essa informação processada no game.c, na função kbdInterruptHandler, e nos menus.c na parte relativa ao Menu do Score. Este periférico tem as suas funções implementadas em keyboard.c e keyboard.h.

**Rato** – O rato é utilizado para movimentar o cursor no menu principal e no menu pausa, dependendo assim da sua posição; para selecionar opções nos menus. Com a receção por interrupção dos packets do rato em AgeOfWar.c na função interruptHandler, torna possível a sua utilização, sendo feita posteriormente o devido tratamento dos dados por este recebido em mouse.h e mouse.c. A sua utilização está presente no ficheiro menu.c.

**Placa gráfica** – A placa gráfica utilizada em modo 0x117 permite fazer o display de todos os menus, da data, de texto, e de todos os elementos do jogo em si. Este modo refere-se à utilização da resolução 1024*768 com 2 bytes por pixel. É importante referir que toda a informação é primeiro passada para um double buffer e apenas depois é que é feita a transição para a memória principal. Com o uso de unidades também se torna crucial saber a posição destas para o devido tratamento das suas colisões, como também das respetivas animações. Os ficheiros deste periférico são video_gr.h, video_gr.c, vbe.c e vbe.h.

**RTC** – O rtc é utilizado para ler a data e hora atual do computador, estando presente no menu Créditos, em menu.c. Este periférico tem as suas funções no ficheiro rtc.h e rtc.c

Com a utilização da máquina de estados na estrutura principal do jogo, apenas necessitamos de uma função que utilizasse a chamada de driver_receive().

A função interruptionHandler em AgeOfWar.c é responsável pela chamada desssa função, sendo que conforme a interrupção recebida é ativada uma flag a identificar o periférico de onde a interrupção foi originada, e depois de acordo com essa flag e de acordo com o estado atual da estrutura principal são feitas as atualizações necessárias ao jogo.

## 2.4 Estrutura e modulação do projeto

O projeto foi desenvolvido tendo por base módulos bem estruturados e com a reutilização dos diferentes módulos já realizados nos labs, cujas características são apresentadas de seguida, assim como o seu peso relativo no projeto.

- AgeOfWar.c – Este módulo permite, através da máquina de estados e do ciclo de interrupções, gerir a informação recebida e utilizá-la conforme o estado atual do jogo. Assim este módulo é crucial para relacionar todos os outros módulos, na medida que serve para coordenar todas as operações que resultam de cada um dos mesmos.

  Achamos que no geral o Diogo fez 60% deste módulo e o Rui 40%.
- Animation.c – Este módulo permite a criação de animação das unidades com a utilização de um array de bitmaps, permitindo assim a alteração dos bitmaps de acordo com o estado da unidade.

  O Diogo fez 60% deste módulo e o Rui 40%.
- Bullet.c – Este módulo contém a abstração de uma bala, qual a velocidade desta dependendo da inclinação e a deteção de colisão entre a bala e o “chão”.

  O Diogo fez 40% deste módulo e o Rui 60%.
- Const.h – Este módulo serviu para compilar grande parte das constantes necessárias para o jogo, para uma maior facilidade de acesso.

  O Diogo fez 50% deste módulo e o Rui 50%.
- Defense.c – Este módulo contém a parte relativa ao armamento de defesa das torres, que permite também quer a verificação se tem o array de bullets cheio ou vazio, quer a adição de uma bala a esse array.

  O Diogo fez 50% deste módulo e o Rui 50%.
- Font.c – Este módulo contém a parte relativa de escrita no buffer na parte dos Scores e da data/hora do RTC.

  O Diogo fez 50% deste módulo e o Rui 50%.
- Game.c – Este módulo contém toda a lógica de jogo desde a sua inicialização, às mudanças de nível, interação com o teclado, ao controlo dos dois jogadores, verificação de colisões e batalha entre unidades.

  O Diogo fez 40% deste módulo e o Rui 60%.
- Keyboard.c – Este módulo contém todo o material necessário para lidar com este hardware, assim como uma enumeração com os diferentes scancodes do teclado(retirado do blog do Ferrolho) e a sua conversão para ASCII.

  O Diogo fez 50% deste módulo e o Rui 50%.
- Main.c – Este módulo contém a parte de dar permissões ao programa para

correr no Minix, assim como inicializar o modo gráfico e chamar o módulo principal do jogo.

O Diogo fez 50% deste módulo e o Rui 50%.

- Menu.c – Este módulo contém toda a informação relativa a todos os menus, e a sua interação com eles.

  O Diogo fez 60% deste módulo e o Rui 40%.
- Mouse.c – Este módulo contém uma struct de dados para o rato, assim como todas as funções necessárias para este funcionar corretamente. Foram implementadas duas funções mais simples para o tratamento de packets.

  O Diogo fez 60% deste módulo e o Rui 40%.
- Number.c – Este módulo foi necessário, sendo criado independente da font.c, pois requeria acesso a determinadas funcionalidades do jogo. A funcionalidade é desenhar a experiência e outros números durante o jogo.

  O Diogo fez 40% deste módulo e o Rui 60%.
- Player.c – Este módulo permite gerir a informação do jogador, desde o nível em que se encontra à gestão de quantidade de unidades, experiencia, dinheiro…

  O Diogo fez 40% deste módulo e o Rui 60%.
- Rtc.c – Este módulo gere toda a informação deste periférico e permite a atualização e desenhar a data e hora respetiva do sistema.

  O Diogo fez 60% deste módulo e o Rui 40%.
- Slot.c – Este módulo permite gerir a quantidade de slots disponíveis para armamento de defesa nas torres.

  O Diogo fez 40% deste módulo e o Rui 60%.
- Special.c – Este módulo gere o ataque especial durante o jogo.

  O Diogo fez 50% deste módulo e o Rui 50%.
- Terrain.c – Este módulo gere os diferentes fundos correspondentes a cada nível do jogo.

  O Diogo fez 60% deste módulo e o Rui 40%.
- Timer.c – Este módulo permite a utilização e tratamento de dados deste periférico.

  O Diogo fez 50% deste módulo e o Rui 50%.
- Tower.c – Este módulo contém a informação de cada torre, incluindo as funções para gerir os dados desta.

  O Diogo fez 40% deste módulo e o Rui 60%.
- Unit.c – Este módulo contém toda a informação relativa as diferentes unidades e aos seus estados.

  O Diogo fez 40% deste módulo e o Rui 60%.

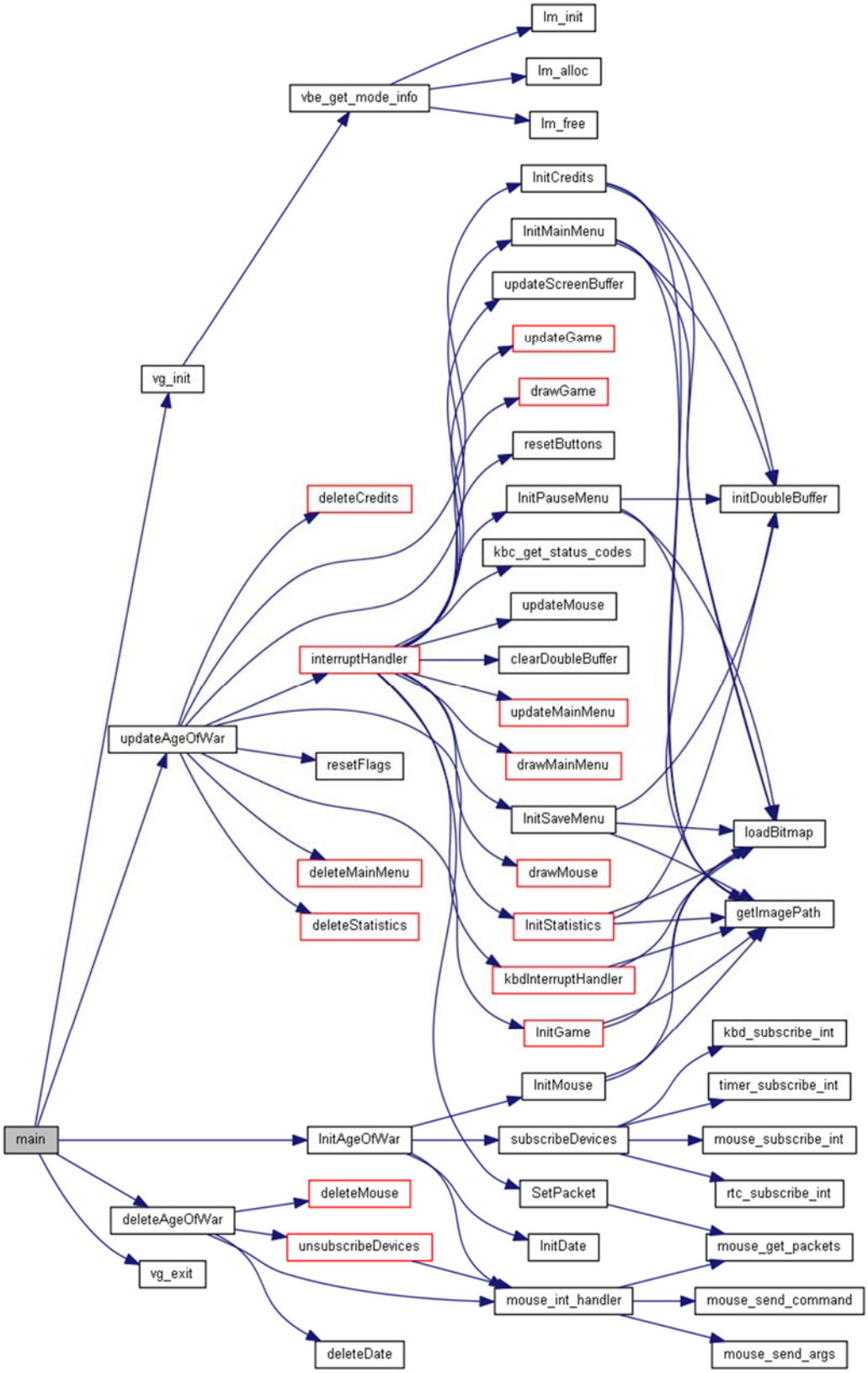
- Vbe.c – Este módulo contém as funções para obter a informação sobre o modo gráfico utilizado.

  O Diogo fez 50% deste módulo e o Rui 50%.
- Video_gr.c – Este módulo é responsável pela interação com a placa gráfica assim como a inicialização do double buffer. Ainda foram acrescentadas duas funções respetivas a alteração da cor de 1 pixel com 2 bytes, e a criação de um retângulo.

  O Diogo fez 50% deste módulo e o Rui 50%.
- Bitmap.c – Código desenvolvido por Henrique Ferrolho, estudante do MIEIC, com base num post do fórum fedoraforum.org. Este módulo é utilizado para fazer o load dos bitmaps e para fazer o seu draw no buffer. Foram feitas alterações na função de draw para poder ser desenhada para respetivos buffer's e ainda foi criada uma nova função de desenhar bitmaps em que lida com transparências.

  O Diogo fez 60% deste módulo e o Rui 40%.

### 2.4.1 Diagrama de chamada de funções



## 2.5 Detalhes de Implementação

### 2.5.1 Máquinas de estado

Ao longo do projeto, tentamos implementar máquinas de estado, visto que é uma forma de tornar o código mais legível e estruturado.

Então, a primeira máquina de estados faz parte da estrutura AgeOfWar. A partir desta conseguimos controlar as várias transições entre menus e o jogo. Além disso facilitou o processamento das interrupções, pois sabendo o estado atual, tornou-se mais fácil aplicar a resposta do programa face a essas interrupções.

As outras máquinas de estado encontram-se implementadas em vários módulos desde o módulo das unidades ao módulo da animação destas. Sabendo assim, o estado atual das unidades é possível saber qual o sprite adequado a desenhar.

### 2.5.2 Desenvolvimento por "layers"

De forma a tornar o projeto mais estruturado, tentamos criar vários ficheiros que se relacionassem entre si, mantendo ao mesmo tempo uma estrutura hierarquizada. Como exemplo disto temos, o ficheiro AgeOfWar sendo o principal deste projeto, que por sua vez, conforme o estado, interage com menu.c ou game.c. Posto isto, no estado de jogo, game.c interage com os outros módulos necessários para o correto funcionamento do mesmo. Com isto, as estruturas principais vão se relacionando com as mais secundárias criando uma enorme abstração para o programador conseguindo testar o programa por partes, em vez de implementar tudo de uma única vez.

### 2.5.3 Programação orientada a objetos

Apesar de C não ser uma linguagem orientada a objetos, tentamos replicar essa orientação no nosso projeto, criando para isso estruturas como AgeOfWar, Unit, Tower, etc abstraindo assim estes "objetos" e atribuindo-lhes membros função como Init, update, draw e delete (além de outras funções necessárias, claro). Este método de desenvolvimento facilitou bastante tanto o acesso como a modificação desses objetos.

### 2.5.4 Deteção de colisões

No decorrer do jogo são detetadas colisões entre objetos "Bullet" e "Unit", "Unit " e "Unit", "Unit" e "Tower" e também "Bullet" e "Tower".

Para todas estas colisões, a sua deteção é realizada verificando se as posições dos objetos se sobrepõem (se a sua posição no eixo dos x somada da largura do seu bitmap, sobrepõe a posição no eixo dos x do outro objeto somada da largura do bitmap do outro objeto).

Para cada colisão existe um tratamento específico, que inclui a retirada de “vida” de um dos objetos ou até dos dois.

### 2.5.5 Ficheiros Bitmap

Todas as imagens e sprites incluídas estão em formato bitmap, bem como os carateres que formam o texto. Usando como base o já referido código desenvolvido por Henrique Ferrolho, tendo sido feito duas alterações referentes a possibilidade de escolher o buffer para desenhar e criada uma função que lida com transparência.

De modo a sobrepor bitmaps, como por exemplo o fundo do nível com as unidades e torres nela implementadas, decidimos usar como cor transparente o tom de rosa que equivale ao código hexadecimal RGB de #FF00FF. Assim, quando o bitmap está a ser desenhado, se a cor do atual pixel a ser lido for igual à cor referida, esse pixel é ignorado, e o iterador passa para o pixel seguinte.

Todos as imagens têm resolução par, devido ao facto de existir problemas nesta biblioteca que distorciam as imagens.

Ainda foi implementada um módulo que trata da animação das unidades de acordo com o seu estado (ataque, andar, parado, disparar e morrer).

### 2.5.7 Frame rate

A partir das interrupções do timer 0, temos a possibilidade de ter um qualquer número de fps, de 0 a 60 (valor por defeito do minix), sem este ser necessariamente múltiplo de 60. Isto deve-se ao facto de existir um contador de interrupções, que faz com que realizando o calculo do resto entre o numero de interrupções e o frame rate desejado permite atualizar o jogo num valor diferente sem se alterar a frequência do timer.

### 2.5.8 Código em assembly

No projeto, foram implementadas funções em assembly na parte do RTC para desativar e ativar interrupções enquanto os registos estão a ser lidos. E no teclado para leitura de *scancodes*.

# 3. Conclusões

Em relação ao que se poderia melhorar nesta unidade curricular, achamos que quanto à porta de serie se deveria realizar pelo menos uma aula de apoio centrada nesse periférico de forma a ajudar a ultrapassar as dificuldades existentes em relação a este. E a entrega dos labs deveria ser até 24 horas após o término da aula prática, pois por vezes os estudantes levam dúvidas a esclarecer nessa aula prática e sobrecarregam bastante o professor para conseguirem terminar o lab. No entanto, se fosse dado mais 24 horas seria possível os alunos tirarem as duvidas e corrigir alguns erros durante a aula prática e depois fora da aula poderem acabar o lab e submeter. E por fim, deveria ser dado um lab mais de introdução ao Minix e à linha de comandos porque no inicio a demora em realizar os lab's deve-se ao facto de a maior parte dos estudantes nunca terem tido contato com a linha de comandos UNIX e até esta familiarização não estar completa provoca bastantes distúrbios no arranque do semestre.

Quanto aos aspetos positivos, acho que a matéria abordada é bastante abrangente levando a um melhor conhecimento do hardware presente nas maquinas de hoje em dia.

Por último, apesar de esta ser uma unidade curricular bastante exigente devido às labs, ao projeto e à introdução ao funcionamento dos periféricos quer em termos de tempo quer em termos de raciocínio, ambos concordamos que, sem dúvida, a forma como unidade curricular está estruturada permite aprofundar bastante os conhecimentos nos diversos periféricos lecionados ao longo do semestre bem como melhorar o conhecimento da linguagem C, sendo o projeto final o desafio ideal para relacionar esses periféricos e colocar à prova os conhecimentos adquiridos.

Como autoavaliação concordamos com o seguinte:

Participação: Diogo – 50% Rui – 50%

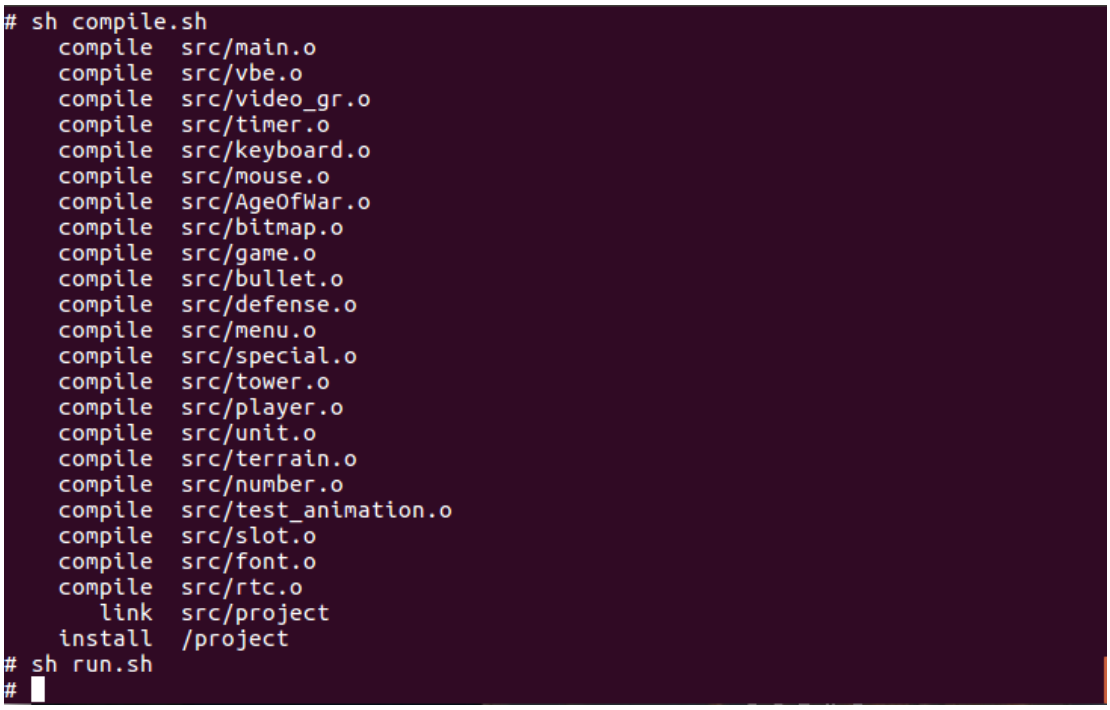
Contribuição relatório: Diogo – 50% Rui – 50%

Contribuição total: Diogo – 50% Rui – 50%

# Anexo 1 – Instruções de Instalação

Para poder correr o jogo é necessário estar na pasta do projeto e executar os seguintes comandos em modo root (su):

- sh install.sh
- sh compile.sh
- sh run.sh

# Anexo 2 – Instruções de Jogo

Cada jogador começa com 150 de ouro, o que lhe permite criar de início algumas unidades, bem como instalar uma defesa na torre.

À medida que vai ganhando mais ouro poderá criar novas unidades, defesas e espaços para novas defesas. Poderá também vender defesas.

Terão disponível um ataque especial que irá retirar “vida” às unidades inimigas e que demora 30 segundos a recarregar.

Com a morte de tropas inimigas ganha-se experiência necessária para evoluir de nível. Ao evoluir de nível passa-se a ter acesso a novas unidades e defesas, para além disso, a vida da torre aumenta.

# Anexo 3 – Informação sobre as diferentes eras

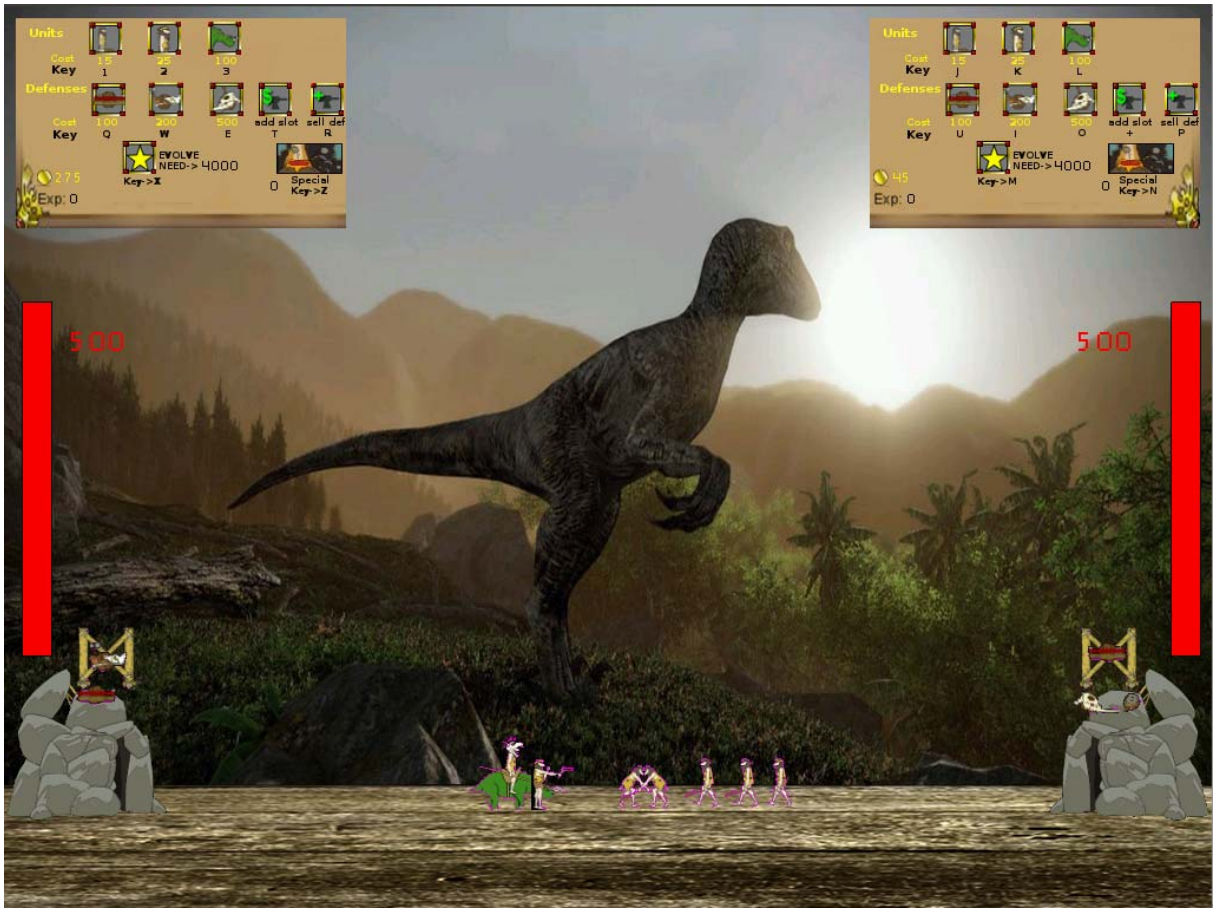
## *Stone Age*

A “Stone Age” é a primeira era apresentada neste jogo. A vida máxima da torre nesta era é 500.

| Soldiers | Cost | Life | Strength |
|---|---|---|---|
| Clube Man | 15 | 45 | 9 |
| Slingshot Man | 25 | 75 | 19 |
| Dino Rider | 100 | 150 | 44 |

| Turrets | Cost | Strength |
|---|---|---|
| Rock Slingshot | 100 | 10 |
| Egg Automatic | 200 | 15 |
| Primitive Catapult | 500 | 20 |

**Evolução**: Precisa de 4000 exp. para evoluir para a “Medieval Age”.



## *Medieval Age*

A “Medieval Age” é a segunda era apresentada neste jogo. É baseada no período medieval entre o quinto e o decimo quinto século. A vida máxima da torre nesta era é 1000.

| Soldiers | Cost | Life | Strength |
|---|---|---|---|
| Swords Man | 50 | 90 | 18 |
| Archer | 75 | 150 | 38 |
| Knight | 500 | 300 | 88 |

| Turrets | Cost | Strength |
|---|---|---|
| Catapult | 500 | 20 |
| Fire Catapult | 750 | 30 |
| Heated Tar | 1000 | 40 |

**Evolução**: Precisa de 14000 exp. para evoluir para a “Renaissance Age”.

## *Rennaissance Age*

A “Medieval Age” é a terceira era apresentada neste jogo. Esta era é baseada nos soldados franceses e alemães. A vida máxima da torre nesta era é 2000.

| Soldiers | Cost | Life | Strength |
|---|---|---|---|
| Dueler | 200 | 180 | 38 |
| Musketeer | 400 | 300 | 76 |
| Cannoneer | 1000 | 600 | 176 |

| Turrets | Cost | Strength |
|---|---|---|
| Small Turret | 1500 | 40 |
| Large Turret | 3000 | 60 |
| Explosives Turret | 6000 | 80 |

**Evolução**: Precisa de 45000 exp. para evoluir para a “Modern Age”.

## *Modern Age*

A “Modern Age” é a quarta era apresentada neste jogo. A vida máxima da torre nesta era é 3200.

| Soldiers | Cost | Life | Strength |
|---|---|---|---|
| Melee Infantry | 1500 | 540 | 108 |
| Rifle Infantry | 2000 | 900 | 228 |
| Tank | 7000 | 1800 | 528 |

| Turrets | Cost | Strength |
|---|---|---|
| Modern Turret | 7000 | 120 |
| Rocket Launcher | 9000 | 180 |
| Advanced Turret | 14000 | 240 |

**Evolução**: Precisa de 200000 exp. para evoluir para a “Future Age”.

## *Future Age*

A “Future Age” é a última era apresentada neste jogo. A vida máxima da torre nesta era é 4700.

| Soldiers | Cost | Life | Strength |
|---|---|---|---|
| God's Blade | 5000 | 1620 | 324 |
| Blaster | 6000 | 2700 | 684 |
| War Machine | 20000 | 5400 | 1584 |

| Turrets | Cost | Strength |
|---|---|---|
| Green plasma cannon | 24000 | 360 |
| Red plasma cannon | 40000 | 540 |
| Blue plasma cannon | 100000 | 720 |

**Evolução:** Não existem mais níveis.