

2 DE JUNHO DE 2024

## FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# COMPUTER LABS 23/24

## PROJECT REPORT: CLAUSTRO

Alexandre Morais - 201906049

Daniel Silva - 201909935

Pedro Plácido - 202107987

Tiago Simões - 202108857

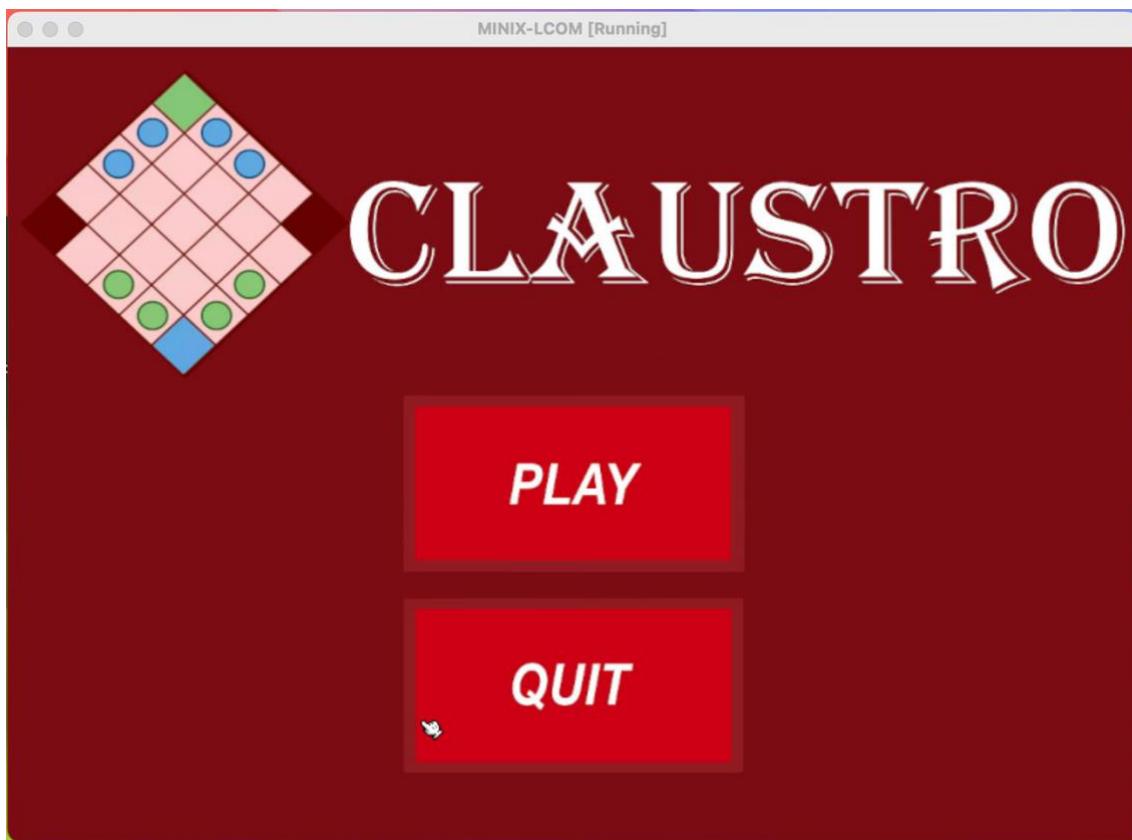
T17 – G03

## 1. User instructions

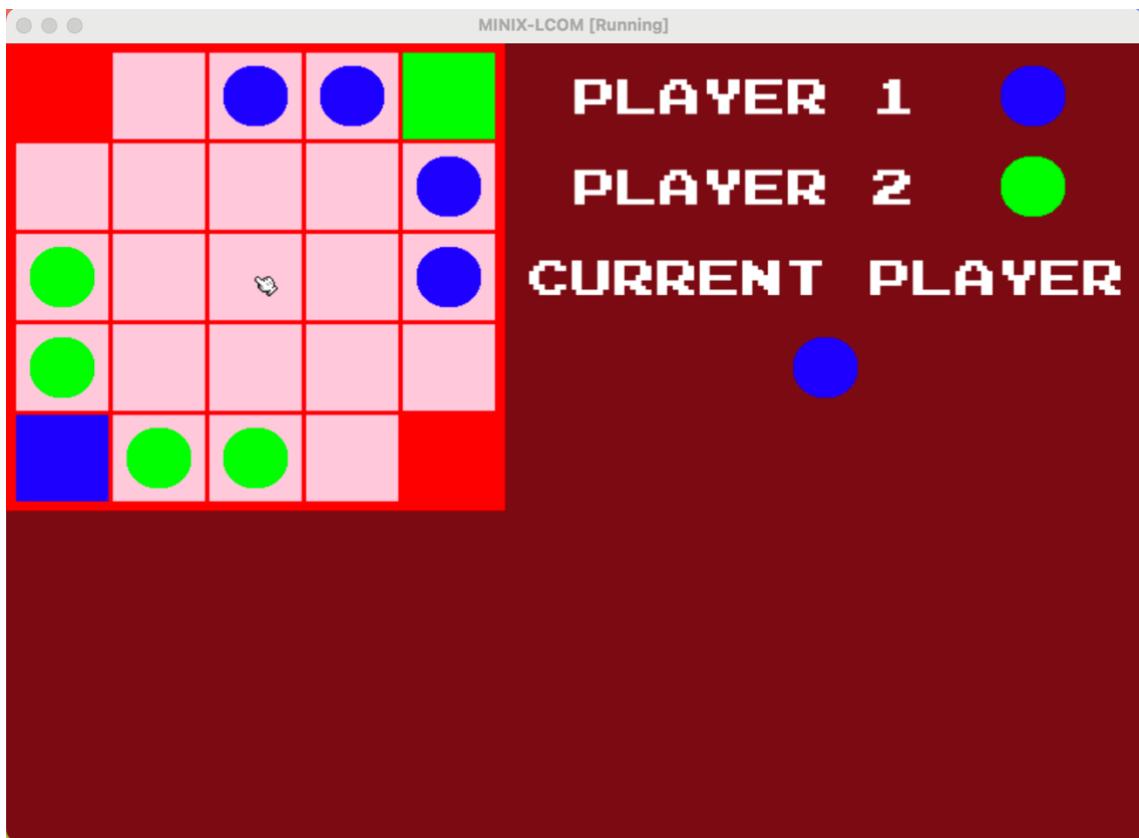
Claustro, é um jogo para 2 jogadores, jogado num tabuleiro quadrangular 5x5 e cada jogador tem 4 peças da mesma cor, ou seja, são 8 no total de 2 cores diferentes. Alternadamente, os 2 jogadores movem uma das suas peças com o objetivo de chegar ao seu *goal* - quadrado (situado num canto) da mesma cor das suas peças. Inicialmente as 4 peças estão junto ao *goal* do adversário. Os outros 2 cantos preenchidos não são jogáveis.

Um jogador pode mover uma peça ortogonalmente em direção ao seu *goal* ou capturar uma peça do adversário na diagonal em qualquer direção. Após uma captura, o jogador que capturou a peça deve reposicionar a peça capturada (do adversário) em qualquer quadrado livre. Se um jogador chegar ao *goal* ou se encontrar numa posição em que não consegue executar um movimento ou uma captura, ganha automaticamente.

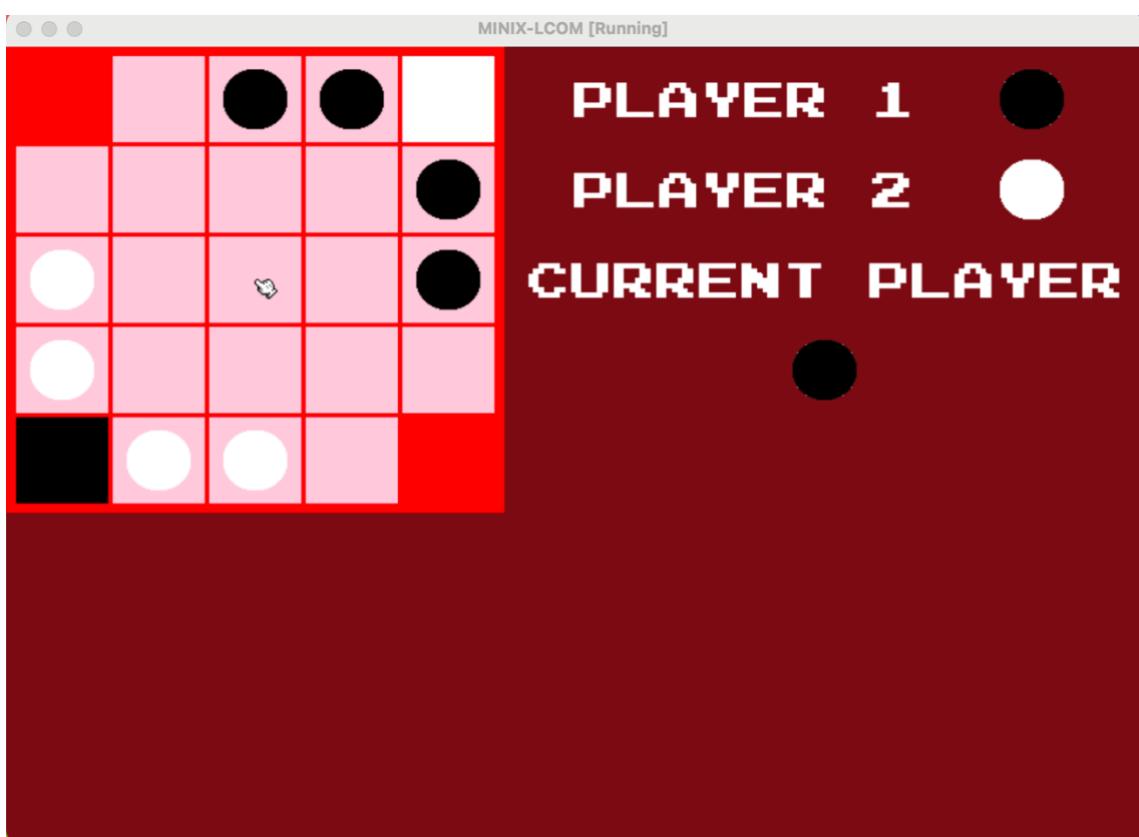
Ao iniciar o jogo, é apresentado o menu inicial com 2 botões *PLAY* e *QUIT*, tal como ilustra a captura de ecrã abaixo.



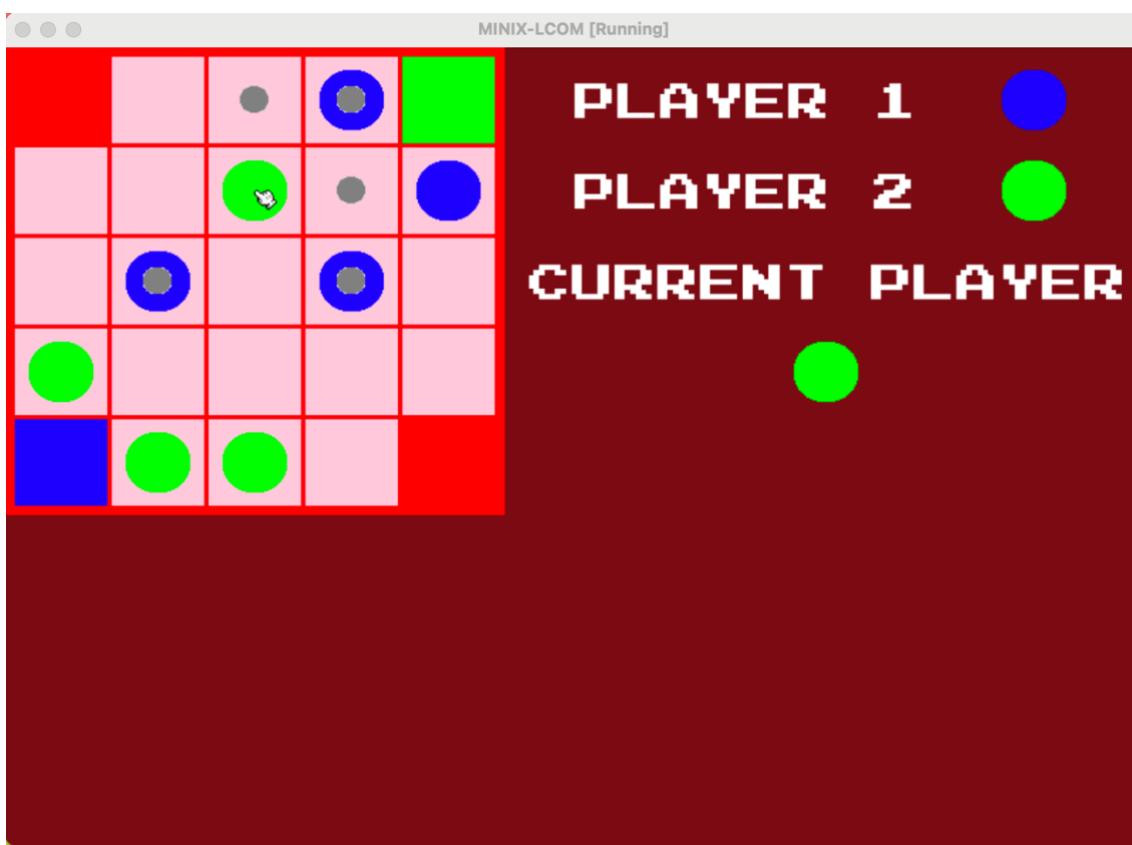
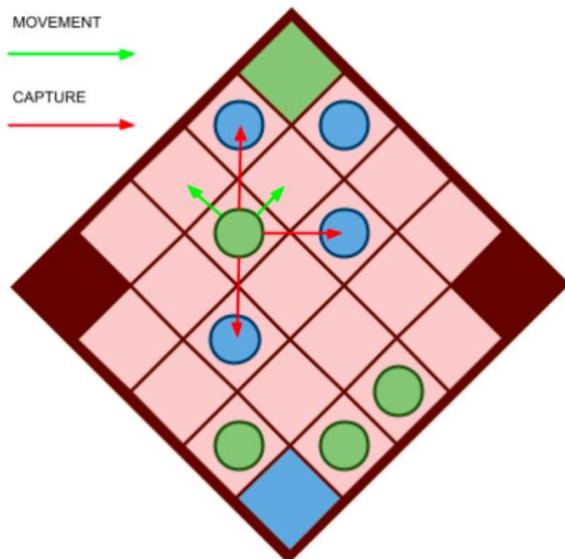
Para começar um novo jogo é necessário clicar no botão esquerdo do rato sob a área do botão *PLAY*, ou apenas clicar na tecla *ENTER* do teclado. Para sair (voltar ao modo texto do Minix), deve utilizar-se o botão *QUIT* ou clicar em qualquer momento do jogo na tecla *ESC*. Durante uma partida é possível visualizar o tabuleiro (e disposição das peças) à esquerda, enquanto à direita tem a indicação do jogador atual.



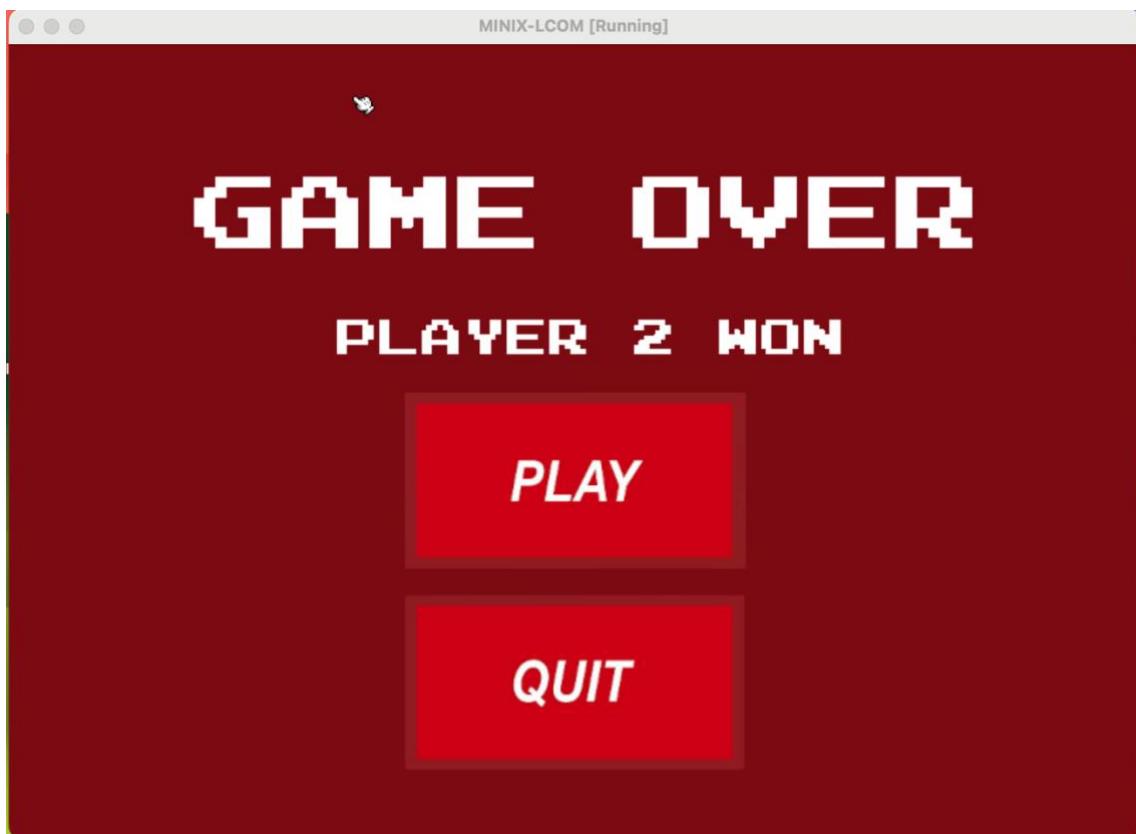
Num determinado horário, por defeito entre as 20H e as 8H, é ativado o modo escuro - *dark mode* – permitindo alterar a cor do tabuleiro e das peças.



Uma jogada, isto é, o movimento de uma peça, pode ser executada utilizando os periféricos rato e teclado em simultâneo. Com o teclado, pressionando as teclas W, A, S e D, é possível movimentar o cursor pelos quadrados do tabuleiro e com a tecla ENTER selecionar a peça e o quadrado do destino. Utilizando o rato é possível movimentar o cursor livremente e a seleção da peça/destino é feita com o botão esquerdo. Sempre que o cursor está por cima de uma peça do atual jogador é mostrado, através de círculos cinzentos, os possíveis destinos da mesma. Segue, um estado do jogo (retirado do manual de instruções) que exemplifica as possíveis movimentações de uma determinada peça e implementação do mesmo estado tal como descrito acima.



Por último, quando o jogo termina é apresentado no ecrã o vencedor e os mesmos botões do menu inicial.



## 2. Project status

Dispositivo	Funcionalidade	Int.
Timer	Definir a <i>frame rate</i> .	Sim
KBD	Navegar nos menus. Executar uma jogada.	Sim
Mouse	Navegar nos menus. Executar uma jogada.	Sim
Video card	Representar o estado do jogo.	Não
RTC	Implementar o modo escuro.	Sim

### Timer

As interrupções do *timer 0* foram utilizadas para definir a *frame rate*, ou dito de outra forma, o número de vezes por segundo em que é feita a cópia do segundo *buffer* - *video\_second\_buffer* – para o principal – *video\_mem* - que tem tradução física na VRAM, na técnica *double buffering*. Note-se que não foi necessário alterar a frequência do timer, utilizando a função *timer\_set\_frequency()* e por consequência *timer\_get\_conf()*, que tinham sido desenvolvidas no lab2, porque a *frame rate* utilizada é 30. Por defeito, o *timer 0* gera 60 interrupções por segundo, e uma vez que 30 é múltiplo e inferior a 60, a cada 2 interrupções é realizada a cópia. Valores acima de 30 não apresentaram ganhos significativos, dado que a carga de processamento maior tem a ver com geração de *frames*. As interrupções do *timer* são tratadas em *timer\_action()*, sendo que as funções relacionadas com o *timer* estão em *timer.c*.

### Graphics card

A placa de vídeo foi utilizada para representar o estado do jogo, como os menus inicial e final, bem como o resultado do processamento do *input* dado pelo utilizador via teclado e rato.

O projeto utiliza o modo gráfico, inicializado em *vg\_init()*, no final retorna ao modo de texto por defeito do Minix através de *vg\_exit()*. Utilizou-se o modo VBE 0x115, com uma resolução de 800x600 pixéis, com 24 bits (3 bytes) por pixel (8:8:8) em modo direto, sendo possível representar  $2^{24} = 16777216$  cores. Apesar de não ser possível o utilizador escolher durante a execução do jogo outro modo, o código suporta múltiplos modos diretos com 8 bits por cor primária, sendo necessário alterar o argumento *mode* de *vg\_init()*. Por exemplo, o modo 0x118 com uma resolução de 1024x768 pixéis.

A principal otimização utilizada no projeto foi a implementação da técnica *double buffering* via cópia, tal como já foi explicado na subsecção do *timer*. A cópia é realizada em *vg\_double\_buffering()*, que é invocada em *timer\_action()*, com a taxa descrita anteriormente. Notou-se que a carga de processamento maior tem a ver com a geração de *frames*, principalmente quando se optou por pintar o *background*. Uma forma de reduzir essa carga seria usar *double buffering* via *page-flipping*. Outra otimização seria reduzir o tamanho do *frame-*

*buffer*, reduzindo a resolução e/ou o número de bytes por pixel, por exemplo, utilizando o modo 0x112.

Utilizou-se *sprites* para representar todos os elementos do jogo, como o tabuleiro, as peças ou o cursor. Os *sprites* foram criados, utilizando *create\_sprite()* a partir de um *pixmap*, uma definição textual, correspondendo um carácter a um pixel, e estando cada tipo de carácter associado a uma determinada cor. Os *pixmaps* são representados através de XPMs, que são guardados num *array* de *strings*, sendo todos da autoria do grupo (utilizando o software GIMP), à exceção do cursor que foi obtido na internet. No entanto, tal como referido numa aula teórica, os XPMs, não são considerados código, daí não estar na secção seguinte como código que não é da autoria do grupo. Os *sprites* são criados no arranque do programa em *create\_sprites()* e destruídos no final em *destroy\_sprites()*, através de *destroy\_sprite()*.

As funções relacionadas com a gráfica estão nos ficheiros *video.c* e *sprite.c*.

## Keyboard

O teclado permite ao utilizador controlar o jogo, isto é, navegar nos menus e executar uma jogada tal como já foi descrito na primeira secção. Sempre que o utilizador prime/liberta uma tecla, o teclado envia *scancodes*. Depois da leitura do *scancode* invocando *kbd\_ih()*, é feito o seu processamento em *kbd\_action()*. Caso o *scancode* gerado quando o utilizador liberta uma tecla (*break code*) for reconhecido pelo programa, isto é, trata-se de um *break code* gerado pelas teclas W, A, S, D, o cursor move-se pelos quadrados do tabuleiro. Na prática, premir uma dessas teclas corresponde a alterar (chamando *update\_pawn\_move()*) os campos da *struct Move* que representa uma jogada, ou seja, as coordenadas iniciais (*xi*, *yi*) e finais (*xf*, *yf*) de uma peça. O comportamento destas teclas é o habitual em jogos, por exemplo, premir W avança o cursor um quadrado para cima, isto significa, subtrai-se uma unidade a *xi* ou *xf*, dependendo se está a ser feita a escolha da peça (*xi*) ou o seu destino (*xf*). Numa primeira fase, o ENTER é utilizado para confirmar as coordenadas iniciais (a peça a mover) e é verificado se a peça pertence ao jogador atual através de *can\_pick()* (de modo a facilitar a jogabilidade em caso de erro). Numa segunda fase, o ENTER é utilizado para confirmar as coordenadas finais (o destino), e é chamada função *move()*, que valida e executa a jogada de acordo com a lógica do jogo.

As funções relacionadas com o teclado estão nos ficheiros *kbd.c* e *kbc.c*.

## Mouse

O rato também permite ao utilizador controlar o jogo, isto é, navegar nos menus e executar uma jogada tal como já foi descrito na primeira secção. Pode realizar as mesmas tarefas que o teclado, permitindo ao utilizador alternar entre os 2 periféricos. Por exemplo, escolher uma peça com teclado e o seu destino com o rato, ou vice-versa.

Sempre que existe um evento no rato, dito de outra forma, o utilizador prime/liberta um botão ou realiza um movimento, o rato envia um pacote de dados. Depois da leitura correta do pacote (verificando o *bit* 3 do 1º *byte*) invocando *mouse\_ih()*, é feito o seu processamento em *mouse\_action()*. Calcula-

se a posição ( $x, y$ ) do rato no ecrã e ativa-se a *flag mouse\_mv*, guardando estes dados na *struct MouseState*. Esta *flag* permite identificar a proveniência (rato/teclado) do novo evento/interrupção ao atualizar a máquina de estados ou gerar uma nova *frame*. Esta *flag* é desativada em *kbd\_action()*.

Os campos da *struct Move* ( $xi, yi, xf, yf$ ) que representa uma jogada, são calculados usando a função *find\_board\_cell()*, que recebe a posição do rato e calcula a posição no tabuleiro. O botão esquerdo é utilizado da mesma forma que a tecla ENTER, que já foi explicado na subseção do teclado.

As funções relacionadas com o rato estão nos ficheiros *mouse.c* e *kbc.c*. Note-se que o *stream mode* foi ativado usando a função *mouse\_data\_reporting()*, desenvolvida no lab4 (não fornecida pela LCF).

## RTC

O RTC é utilizado para obter a hora atual e ativar o modo escuro, por defeito entre as 20H e as 8H, permitindo alterar a cor do tabuleiro e das peças. Sempre que ocorre uma interrupção do RTC (*update interrupt*), invoca-se *rtc\_action()*, que por sua vez chama *rtc\_ih()*, onde é feita a leitura dos registos das horas, minutos e segundos. De seguida, verifica-se se a hora pertence ao horário definido de modo a ativar a *flag dark\_mode*.

As funções relacionadas com o RTC estão no ficheiro *rtc.c* e é possível encontrar os detalhes da sua implementação na secção 4.

### **3. Code organization/structure**

#### **Action Module – 10%**

Este módulo contém o código que processa as interrupções dos dispositivos utilizados no projeto – *timer*, teclado, rato e RTC. Contém as funções que são chamadas no "ciclo *driver-receive*", que por sua vez, chamam os *interrupt handlers* dos dispositivos e processam a informação/eventos de modo a alterar o estado do jogo. Neste módulo está presente um tipo enumerado *GameState* que agrupa todos os estados possíveis do jogo. A máquina de estados que representa o jogo será apresentada na secção seguinte.

#### **Claustro Module – 40%**

Este é módulo principal do projeto, concentra todas as funções relacionadas com o jogo Claustro, nomeadamente a implementação da lógica e regras do jogo, como por exemplo validação e execução de jogadas, bem como deteção de *game over*. Para além disso, também agrupa todas as funções relacionadas com a criação de *frames* – desenho do tabuleiro, peças, cursor, menus, etc. Neste ficheiro encontram-se a matriz *board* e a *struct Move*, que representam o tabuleiro e uma jogada respetivamente.

#### **KBC Module – 6%**

Este módulo foi criado durante a realização do lab4, quando se verificou que o rato utiliza o mesmo controlador que o teclado. São funções genéricas que permitem ler o *status register*, ler o *output buffer* e enviar comandos.

#### **KBD Module – 3%**

O módulo do teclado, que foi desenvolvido na realização do lab3, agrupa funções relacionadas com subscrição de interrupções e o *interrupt handler* que lê um *scancode*.

#### **Mouse Module – 8%**

O módulo do rato, que foi desenvolvido na realização do lab4, agrupa funções relacionadas com subscrição de interrupções e o *interrupt handler* que lê um *byte* do pacote de dados e verifica se o *bit 3* do 1º *byte* está ativo permitindo atenuar problemas de sincronização. Existe uma função que faz *parsing* do pacote de dados e outra que ativa/desativa o *data reporting*. Por último, para além da *struct packet* declarada em lab4.h, existe a *struct MouseState* que guarda a posição (x,y) do rato e uma *flag*, cujo propósito já foi explicado anteriormente.

## **Proj Module – 9 %**

Este módulo contém o *loop* principal do jogo que é o "ciclo *driver-receive*", cuja condição de saída é o estado de jogo ser QUIT. Este ciclo é invocado apenas uma única vez.

## **RTC Module – 8%**

O módulo do *Real Time Clock*, que foi desenvolvido no âmbito do projeto, agrupa funções relacionadas com subscrição/ativação das *update interrupts* (UI) e o *interrupt handler* que lê a hora do dia, escrevendo os valores nos campos da *struct RTC\_Time*. Também foram criadas 2 funções genéricas (*rtc\_read()*) e (*rtc\_write()*) que permitem a leitura/escrita direta de registos. Podem ser encontrados mais detalhes sobre a implementação deste módulo na secção seguinte.

## **Sprite Module – 3%**

Este módulo foi adaptado dos *slides* "Lab 5: Sprites" e contém código que permite a criação, desenho e destruição de *sprites*. A *struct Sprite* armazena a posição atual, as dimensões e o *pixmap*.

## **Timer Module – 3%**

O módulo do *timer*, que foi desenvolvido na realização do lab2, agrupa funções relacionadas com subscrição de interrupções e o *interrupt handler* que incrementa um contador. As outras funções implementadas no lab2 não foram utilizadas no projeto pelas razões que já foram mencionadas.

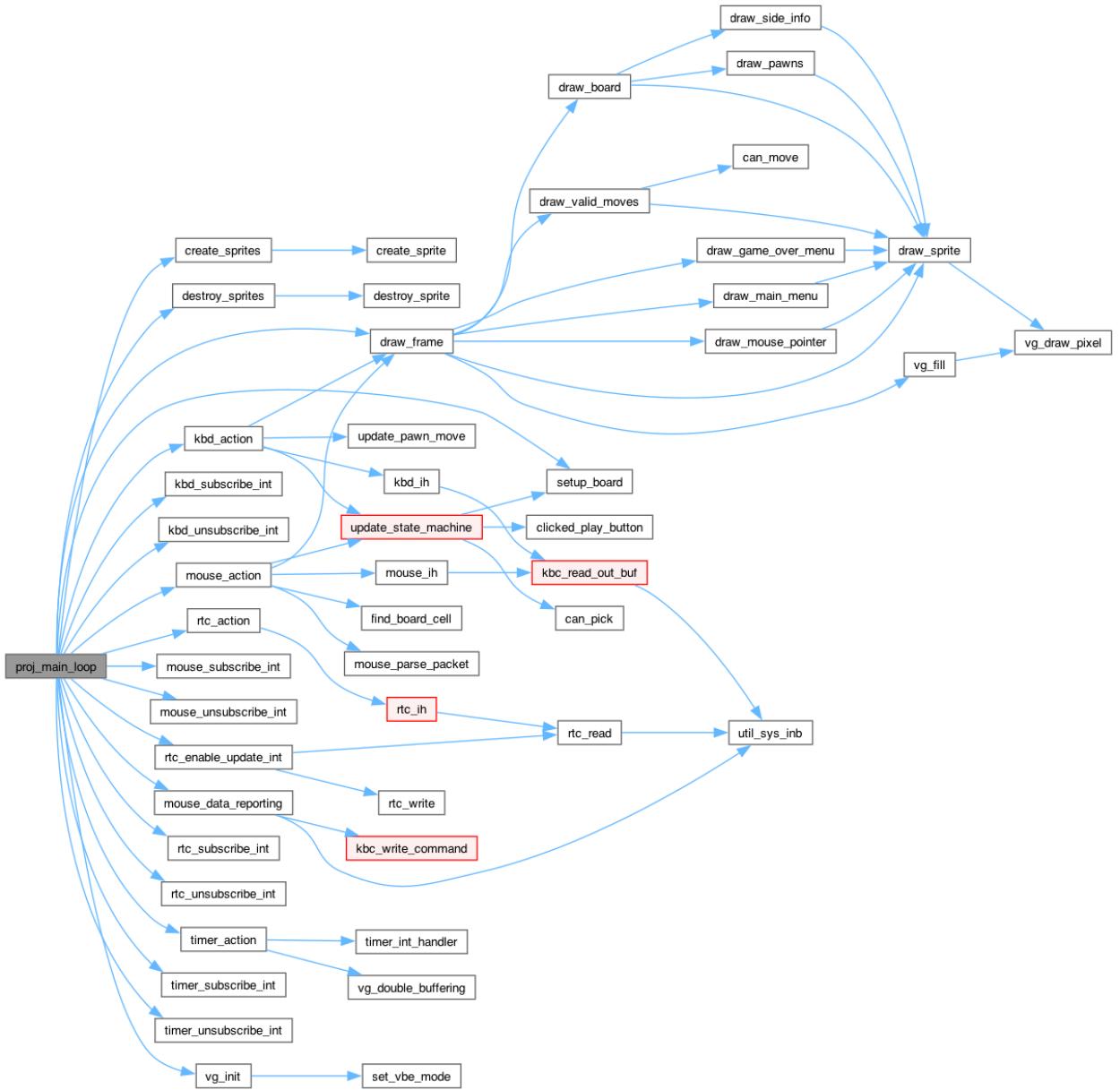
## **Utils Module – 1%**

Neste ficheiro, existe apenas a função *util\_sys\_inb()* (desenvolvida no lab2), que permite invocar a *system call sys\_inb()* lendo o valor para uma variável do tipo *uint8\_t*.

## **Video Module – 9%**

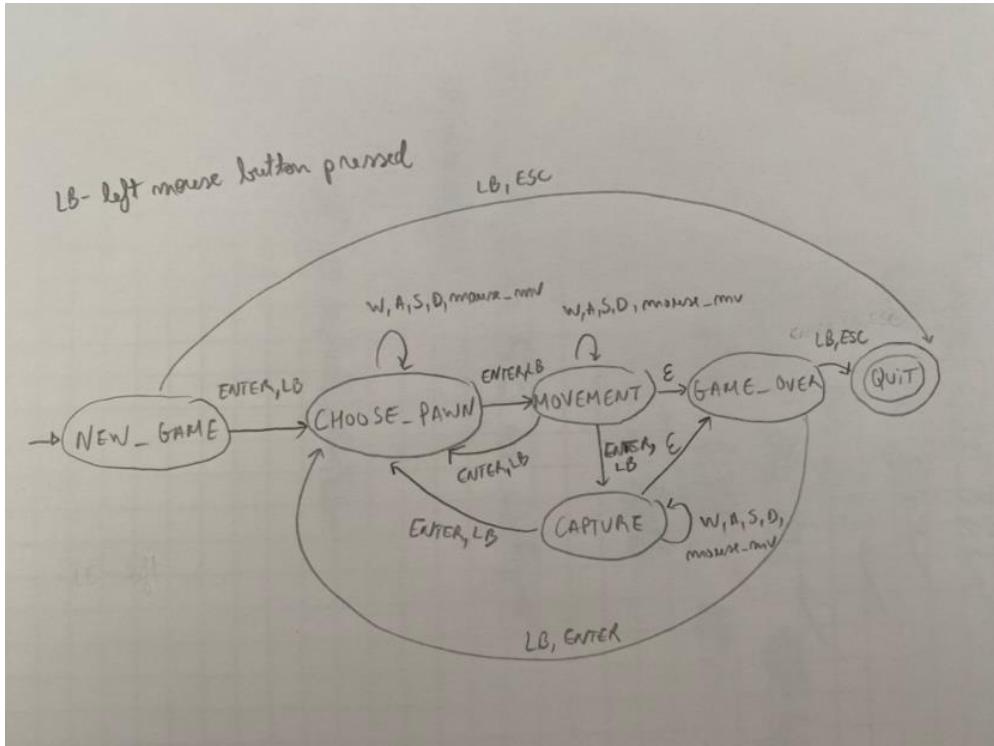
O módulo da placa de vídeo, que foi desenvolvido na realização do lab5, agrupa funções relacionadas com inicialização do modo gráfico, alteração da cor de um pixel e implementação da técnica *double-buffering* via cópia. As outras funções implementadas no lab5 não foram utilizadas no projeto.

## Function call graph



## 4. Implementation details

### Event Driven Design and State Machines



O jogo pode ser visto como uma implementação deste Epsilon-NFA, um autómato finito não determinístico, que pode ser convertido para um DFA. É constituído por 6 estados que fazem parte do tipo enumerado *GameState*.

- NEW\_GAME: estado inicial, onde é apresentado o menu inicial;
- CHOOSE\_PAWN: utilizador escolhe a peça a movimentar;
- MOVEMENT: utilizador escolhe o destino da peça escolhida no estado CHOOSE\_PAWN;
- CAPTURE: utilizador escolhe a posição para repor a peça capturada;
- GAME\_OVER: apresentação do vencedor e menu final;
- QUIT: estado final, o jogo é encerrado (voltando ao modo texto do Minix);

Os detalhes desta solução, na sua maioria, já foram explicados na secção 2, mas observando o desenho do autómato, fica claro que os eventos responsáveis pela transição de estados são, principalmente, premir a tecla ENTER e o botão esquerdo do rato. Note-se que em qualquer estado (para além do NEW\_GAME) é possível transitar para o estado final pressionando a tecla ESC, mas essas transições foram omitidas. As transições épsilon existem, uma vez que, não é necessário consumir qualquer *input* do utilizador quando é feito um movimento ou uma captura para existir um vencedor. O não determinismo verifica-se, por exemplo, depois de um movimento, haver a possibilidade de tratar-se de uma captura ou trocar de jogador e regressar ao estado CHOOSE\_PAWN.

## RTC

Para fazer a leitura da hora é necessário ler 3 registos sequencialmente (registos 4, 2 e 0) o que pode resultar em inconsistência nos dados, independentemente da ordem dos registos em que é feita a leitura. Para atenuar este problema o RTC oferece 3 mecanismos - *update in progress flag* (UIP), *update-ended interrupt* e *periodic interrupt*. Utilizou-se o primeiro, em que o RTC ativa o UIP (bit 7) do registo A 244 microssegundos antes de começar um *update* e desativa o bit no final.

Sendo assim, para além de verificar se existe uma interrupção de *update* pendente através do UF (bit 4) do registo C, também se verifica o UIP fazendo “*polling*” neste registo durante 3 tentativas e esperando 244 microssegundos entre elas. No entanto, o problema não é totalmente resolvido, uma vez que, pode ocorrer preempção do processo e ainda assim existir inconsistências. Para resolver isso seria necessário inibir as interrupções e voltar a ativar depois da leitura.

## 5. Conclusions

A principal dificuldade na realização deste projeto tem a ver com a incompatibilidade entre Minix + VirtualBox e macOS. Durante a realização dos labs não houve qualquer problema, no entanto, com a integração de vários dispositivos tornou-se muito difícil desenvolver o projeto em macOS. Apesar de existirem computadores na FEUP, a maior parte do trabalho acaba sendo feito em casa.

Em relação ao projeto, existem pontos a melhorar e funcionalidades que poderiam ser implementadas. Por um lado, melhorar a técnica de *buffering* para *double buffering* via *page flipping* ou *triple buffering*, bem como a leitura das horas devido ao problema mencionado acima. Por outro lado, podiam ser adicionadas animações (quando é feita uma captura por exemplo), um temporizador para limitar o tempo de cada ronda, inserção de texto via teclado (para alterar o nome do jogador) ou até mesmo a porta série.

De qualquer forma, de modo geral os objetivos da unidade da curricular foram atingidos, nomeadamente:

- usar a interface de "hardware" de periféricos comuns;
- desenvolver "software" de baixo nível, p.ex. "device drivers", e de "software" embebido;
- usar a linguagem de programação C de modo estruturado;
- fazer “debugging” de forma sistemática (com base no método experimental científico);
- utilizar várias ferramentas de desenvolvimento de software.