



**Projeto Final  
Laboratório de Computadores  
2022/2023**



**Turma 13 Grupo 6:**

José Santos [up202108729@fe.up.pt](mailto:up202108729@fe.up.pt)

Luís Du [202105385@fe.up.pt](mailto:202105385@fe.up.pt)

Madalena Ye [up202108795@fe.up.pt](mailto:up202108795@fe.up.pt)

Rodrigo Moucho [202108855@fe.up.pt](mailto:202108855@fe.up.pt)

# Índice

<b>1. Introdução</b>	<b>4</b>
<b>2. Instruções de Utilização</b>	<b>5</b>
2.1 Menu Inicial	5
2.2 Instruções	6
2.3 Níveis	6
2.4 Tela de jogo	7
2.4 Game over	8
<b>3. Estado do Projeto</b>	<b>9</b>
3.1 Dispositivos utilizados	9
3.2 Timer	9
3.3 KBD	10
3.4 Mouse	10
3.5 Placa Gráfica	11
3.6 RTC	11
<b>4. Organização e Estrutura do Código</b>	<b>13</b>
4.1 Controller – 25%	13
• KBC.c – 1%	13
• keyboard.c – 4%	13
• rtc.c – 2%	13
• timer.c – 3%	13
• mouse.c – 4%	13
• graphics.c – 4%	13
• panda_controller.c – 2%	14
• map_controller.c – 2%	14
• item_controller.c – 2%	14
• utils.c – 1%	14
4.2 Model – 25%	14
• model.c – 1%	14
• sprite.c – 10%	14
• menu.c – 3%	14
• level_select.c – 3%	14
• instructions.c – 2%	14
• game.c – 4%	14
• game_over.c – 2%	14
4.3 Viewer – 25%	15
• viewer.c – 3%	15
• menu_viewer.c – 4%	15

• level_select_viewer.c – 4%	15
• instructions_viewer.c – 4%	15
• game_viewer.c – 6%	15
• game_over_viewer.c – 4%	15
4.4 States – 25%	15
• state.c – 3%	15
• menu_state.c – 4%	15
• level_selection_state.c – 4%	15
• instructions_state.c – 4%	15
• game_state.c – 6%	15
• game_over_state.c – 4%	16
4.5 Gráfico de chamadas de função	17
<b>5. Detalhes de Implementação</b>	<b>18</b>
<b>6. Conclusões</b>	<b>19</b>
<b>7. Soluções dos mapas</b>	<b>20</b>

## 1. Introdução

Bamboozlers consiste numa adaptação do jogo [Transformice](#), onde o utilizador assume o papel de um panda, em oposição ao rato. O objetivo é construir um caminho, recorrendo a tábuas e caixas de madeira, para que o panda consiga chegar ao alimento dele, o bambu. Após recolhê-lo, deverá regressar a casa e só assim completará um nível do jogo.

Cada nível apresenta uma combinação única de plataformas e armadilhas que exigem pensamento estratégico e que deverão ser superadas durante um tempo limite. Além disso, o número de tábuas e caixas que o panda pode usar também é limitado.

## 2. Instruções de Utilização

### 2.1 Menu Inicial

Ao iniciar o jogo, é apresentado o seguinte ecrã ao utilizador:



Figura 1 – Menu principal

São apresentadas 2 opções, que podem ser selecionadas através do botão esquerdo do rato:

- Levels, para o utilizador escolher o nível que pretende jogar. Também se pode ir diretamente para este menu ao pressionar “ENTER”;
- Instructions, que redireciona o utilizador para uma janela de instruções.

## 2.2 Instruções

Nesta janela, são apresentadas as regras do jogo. Para regressar ao menu principal, deve-se pressionar na tecla “ENTER” tal como indicado.

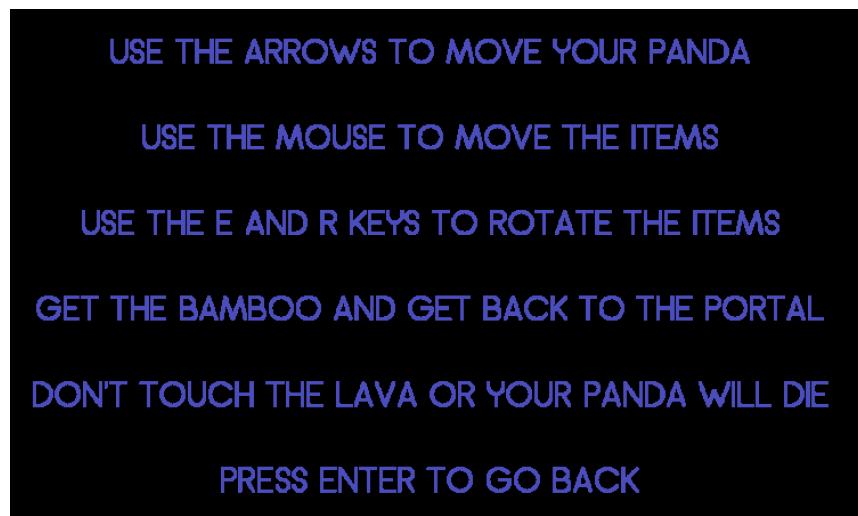


Figura 2 – Instruções

## 2.3 Níveis

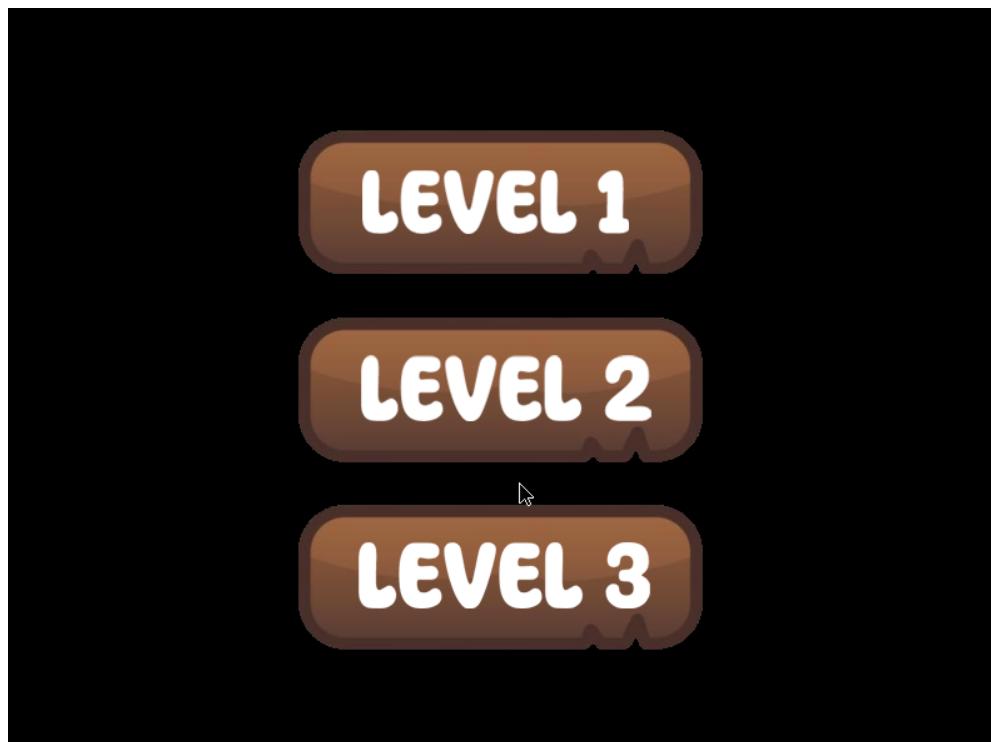


Figura 3 – Menu para selecionar níveis

São apresentados 3 níveis ao utilizador e este só consegue desbloquear o próximo se tiver completado o anterior. Por exemplo, só se pode jogar o nível 2 se o nível 1 for completado.

## 2.4 Tela de jogo

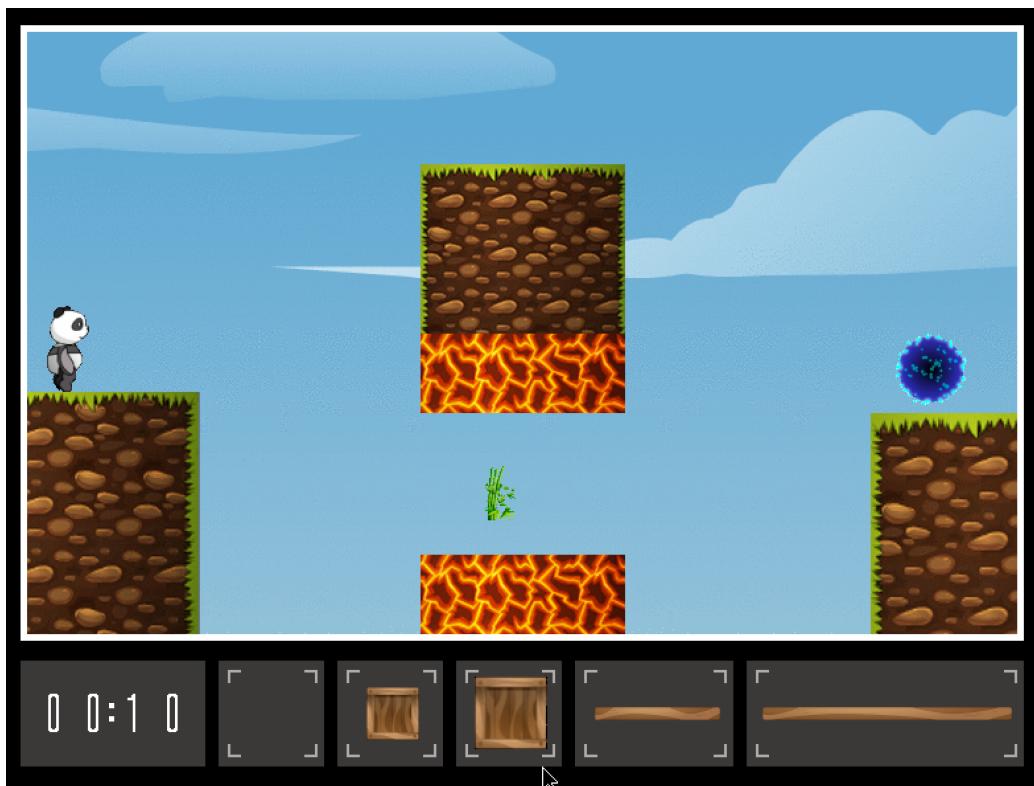


Figura 4 – Tela de jogo

Na parte superior da tela, encontra-se o mapa que o panda precisa de completar para passar de nível. Na parte inferior, está disponibilizado um inventário de materiais que permitem ao panda construir um caminho até ao bambu e depois regressar a casa através do portal. Também se encontra um temporizador com o tempo restante disponível para completar a ronda.

O panda só pode ir para o portal se recolher o bambu, que, uma vez pegue, se encontrará no seu inventário.



Figura 5 – Inventário

O movimento do panda é possibilitado pelas setas do teclado e com recurso à tecla “E” e “R” é possível também rodar as tábuas de madeira.



Figura 6 – Rotação da tábuas

## 2.4 Game over

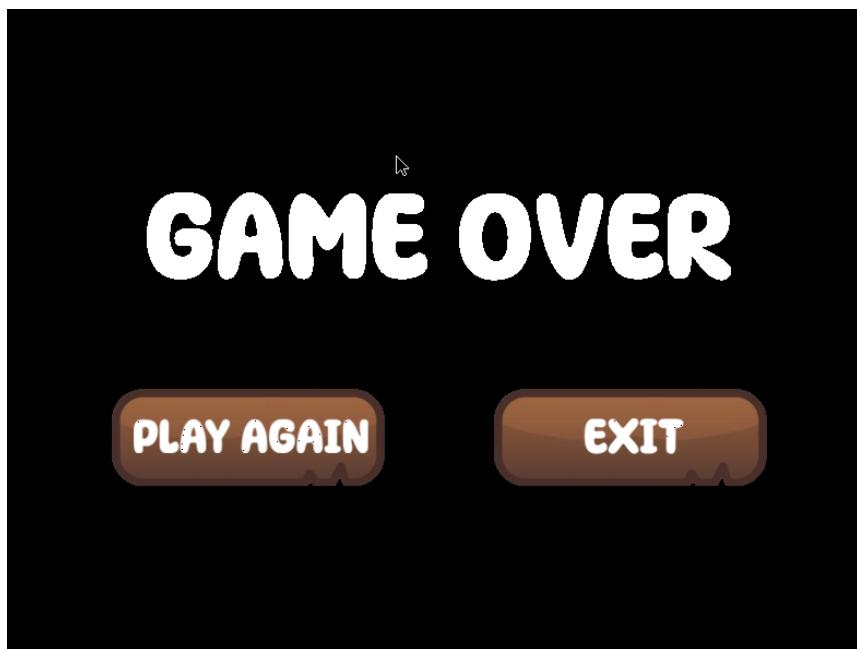


Figura 7 – Menu de game over

É apresentada esta janela quando se sucede um dos seguintes casos:

- O utilizador caiu da plataforma em que está apoiado;
- O tempo terminou e o panda não conseguiu regressar a casa;
- O utilizador tropeçou na lava.

Após perder, o utilizador pode escolher jogar outra vez ou sair do jogo.

### 3. Estado do Projeto

#### 3.1 Dispositivos utilizados

Dispositivo	Funcionalidade	Interrupções?
Timer	Controla a <i>frame-rate</i> e limita o tempo de jogo	Sim
KBD	Controla o movimento do panda e a direção das tábuas	Sim
Mouse	Permite selecionar as opções no menu inicial, no menu de <i>game-over</i> e escolher materiais (tábuas e caixas) – arrastando-os e largando-os para construir caminhos.	Sim
Placa gráfica	Faz o <i>display</i> dos gráficos e de toda a UI do jogo	N/A
RTC	Permite que o jogo alterne entre <i>light mode</i> e <i>dark mode</i> , consoante as horas atuais	Sim

#### 3.2 Timer

O timer é utilizado para controlar a *frame-rate* da placa gráfica, sendo o ecrã atualizado 30 vezes a cada segundo. Também é responsável por limitar o tempo de jogo a cada nível. Além disso, a atualização do *display* engloba diferentes aspetos do jogo. Como tal, o timer acaba por controlar a chamada de várias funções, que realizam diversas tarefas, como a deteção de colisões e a atualização dos *sprites*.

A implementação do timer pode ser encontrada no ficheiro “`../src/controller/timer/timer.c`”. Neste, foram criadas funções para subscrever e cancelar interrupções, para definir a frequência do timer e foi ainda implementado um *interrupt handler* que incrementa um contador global.

### 3.3 KBD

O teclado é utilizado principalmente para controlar o movimento do panda (`update_keyboard_game()` no ficheiro `../src/states/game_state.c`).

Ao receber um *make code* relevante ao jogo (neste caso, os *make codes* das setas do teclado), o sprite do panda movimentar-se-á na direção desejada. No caso do “arrow-right” e “arrow-left”, quando a tecla se encontra pressionada, o panda passa ao estado “RUN”. Ao receber o *break code*, o movimento é interrompido e o estado do panda passa para “INACTIVE”.

Além do mais, também se utiliza o teclado para decidir em que direção queremos colocar as tábuas. Ao pressionar na tecla “E”, roda-se a tábuha no sentido anti-horário e a tecla “R” roda a tábuha no sentido horário.

A tecla “ESC” pode ser usada para sair do programa a qualquer altura do jogo e a tecla “ENTER” pode ser utilizada no menu inicial para selecionar os níveis.

No `“../src/controller/keyboard/keyboard.c”`, foram implementadas as funções para subscrever e cancelar as interrupções do teclado e a função para ler os bytes gerados pelo KBC (`read_scancode()`).

### 3.4 Mouse

O mouse é maioritariamente utilizado para selecionar as opções no menu (quer seja o menu inicial, o menu para selecionar os níveis ou o menu de game over) e para escolher o material a usar, arrastando-o e largando-o para construir caminhos.

As diferentes posições do rato são transformadas em coordenadas cartesianas, permitindo então, por exemplo, distinguir os botões do menu que o jogador quer selecionar. O botão esquerdo do mouse é usado para selecionar a opção que o utilizador quer.

Por outro lado, com o botão esquerdo podemos também escolher no inventário do panda que material queremos usar para construir o seu caminho até ao bambu. Depois de o selecionar, sem largar o botão esquerdo, podemos arrastar a tábuha ou a caixa até à posição pretendida e aí soltar o botão.

No ficheiro `“../src/controller/mouse/mouse.c”`, estão implementadas as funções para subscrever e cancelar as subscrições do rato e para permitir o *data reporting* do mesmo. Também está implementado um *interrupt handler* (`mouse_ih()`) que processa os bytes enviados pelo rato, sendo em seguida construídos os *mouse packets*.

### 3.5 Placa Gráfica

A placa gráfica foi inicializada no modo VBE 0x115, com uma resolução de 800x600 pixels e com cores de 3 bytes por pixel no modo direto – possibilitando  $2^{24}$  cores no total. Esta permite fazer *display* de todos os gráficos e UI relacionados ao jogo.

Uma vez que o projeto foi realizado no modo vídeo, todos os sprites foram animados via XPMs – gerados por meio de imagens PNGs e recorrendo ao software [GIMP](#). Os sprites são desenhados com a função `create_sprite_xpm()` (“`../src/model/sprite.c`”) e posteriormente “destruídos” quando o jogo termina (`destroy_sprite()`), de modo a libertar a memória alocada.

Foi utilizada a técnica de *double buffering*, de forma a tornar o jogo mais fluido e evitar a lentidão no processo de desenho de sprites no ecrã. Foram criados 2 buffers (`main_frame_buffer` e `drawing_frame_buffer`), sendo que o primeiro está na VRAM e o segundo está na memória. O buffer secundário guarda informação temporária e, quando estiver pronto, é chamada a função `swap_buffers()` que copia a informação para o buffer principal, através do `memcpy()`, que é então desenhado.

Um sistema de deteção de colisões foi elaborado também, recorrendo ao *overlap* de pixels, como pode ser comprovado na função `collide()` (“`../src/controller/game/map_controller.c`”).

A fonte de texto utilizada foi a [Soopafresh](#) e os mapas foram criados com base em elementos/sprites originais do Transformice.

### 3.6 RTC

O RTC é utilizado para que o jogo possa alternar entre *light mode* e *dark mode*. São utilizados 2 dos 3 interrupts – para ler as horas e para gerar um alarme.

O RTC permite mudar o background do jogo para de dia se as horas atuais estiverem entre as 6:00 e as 19:59 e para um background de noite se as horas estiverem entre as 20:00 e 5:59.

De um modo simples, ao começar o jogo e através da função `rtc_upd()` e do *time register*, fazemos a leitura das horas atuais e consoante estas escolhemos um background para o jogo. Depois o alarme será programado para “tocar”/gerar um interrupt quando houver a alteração de dia/noite ou vice-versa.

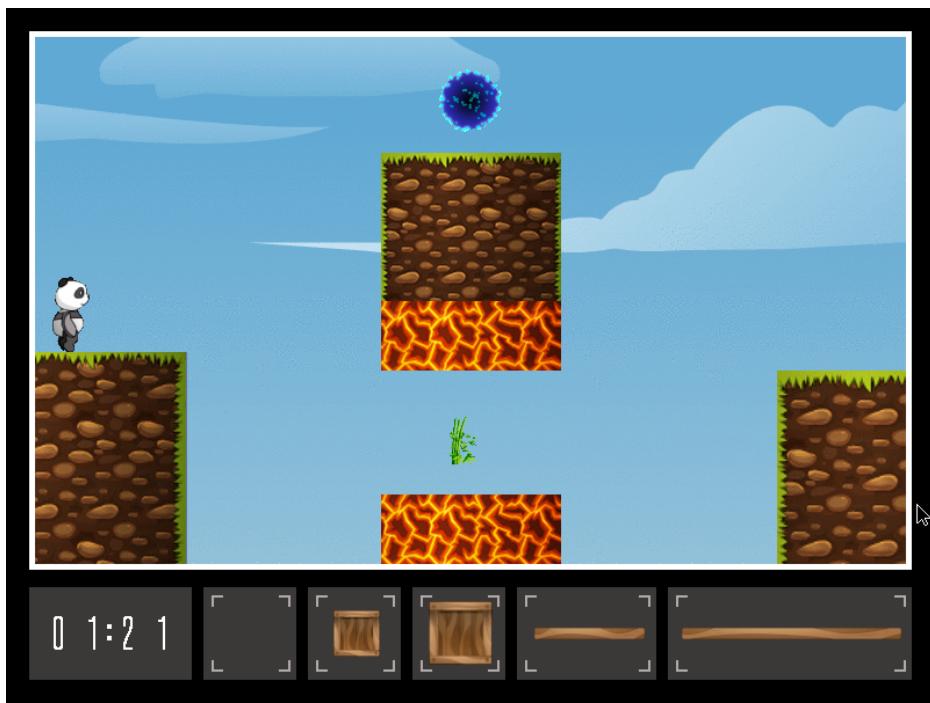


Figura 8 – Light Mode

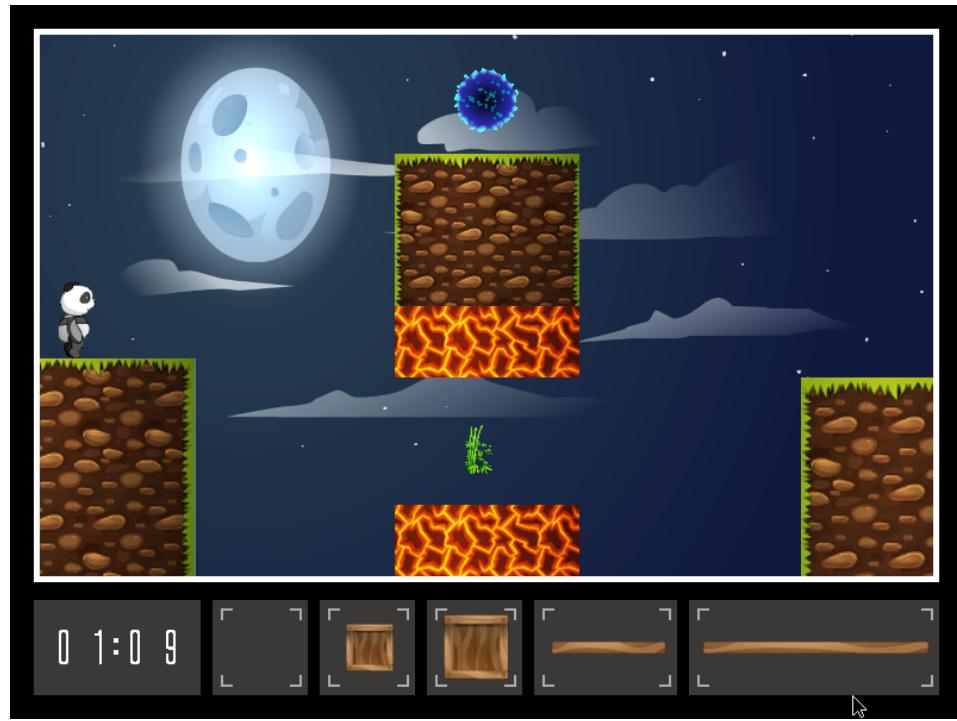


Figura 9 – Dark mode

## 4. Organização e Estrutura do Código

O nosso projeto segue o padrão de arquitetura MVC (Model-View-Controller). O princípio desta arquitetura é dividir o jogo em três camadas: a camada de interação com o utilizador (view), a camada de manipulação de dados (model) e a camada de controlo (controller). Temos um módulo adicional (states) que distingue os diferentes tipos de estado em que o jogo se pode encontrar.

### 4.1 Controller – 25%

- KBC.c – 1%  
Implementa funções encarregues de ler e escrever comandos para o KBC. É útil para o teclado e para o rato.
- keyboard.c – 4%  
Ficheiro constituído por algumas funções utilizadas no Lab 3 e é responsável por tratar de configurar, ativar e cancelar as interrupções do teclado e ler os scancodes recebidos.
- rtc.c – 2%  
Ficheiro constituído pelas funções que controlam o *Real Time Clock* – encarregue de configurar, ativar e desativar as interrupções, gerir os eventos relacionados com o rtc e fazer com que jogo alterne entre *light mode* e *dark mode*.
- timer.c – 3%  
Ficheiro responsável por tratar das funções responsáveis por subscrever e desativar as interrupções do timer, alterar a frequência do timer e tratar dos eventos relacionados com o timer (*timer\_int\_handler*).
- mouse.c – 4%  
Responsável por subscrever e desativar as interrupções do rato, tratar dos eventos relacionados com o mesmo (*mouse\_ih*), criar e sincronizar os pacotes.
- graphics.c – 4%  
Encarregue de tratar de operações com a placa gráfica, como, por exemplo: dar *set* ao modo gráfico, voltar ao modo de texto e criar o *main buffer* e o buffer secundário. Também contém todas as funções responsáveis por desenharem elementos no ecrã.

- `panda_controller.c` – 2%  
Contém as funções que controlam o panda, desde o seu movimento, à sua animação e deteção de colisão com outros objetos.
- `map_controller.c` – 2%  
Contém as funções responsáveis por dar *setup* ao jogo, por gerir os mapas e detetar a colisão destes com outros elementos.
- `item_controller.c` – 2%  
Funções que controlam e a atualizam os objetos do jogo, desde a sua posição à sua animação.
- `utils.c` – 1%  
Funções disponibilizadas do Lab 2, úteis para o nosso projeto e, portanto, decidimos reutilizá-las.

## 4.2 Model – 25%

- `model.c` – 1%  
Encarregue de criar e “destruir” os sprites de todo o jogo.
- `sprite.c` – 10%  
Contém a struct `Sprite` e as funções responsáveis por criar e “destruir” os sprites a partir dos XPMs.
- `menu.c` – 3%  
Funções que criam os sprites do menu principal.
- `level_select.c` – 3%  
Funções que criam os sprites do menu que permite selecionar os níveis.
- `instructions.c` – 2%  
Funções que criam os sprites do menu de instruções.
- `game.c` – 4%  
Funções que criam os sprites do menu de jogo.
- `game_over.c` – 2%  
Funções que criam os sprites do menu de game over.

## 4.3 Viewer – 25%

- viewer.c – 3%  
Responsável por chamar as funções que fazem display de todos os elementos do jogo no ecrã.
- menu\_viewer.c – 4%  
Desenha o menu principal.
- level\_select\_viewer.c – 4%  
Desenha o menu que apresenta os níveis do jogo.
- instructions\_viewer.c – 4%  
Desenha o menu de instruções do jogo.
- game\_viewer.c – 6%  
Desenha a tela de jogo.
- game\_over\_viewer.c – 4%  
Desenha o ecrã de game\_over.

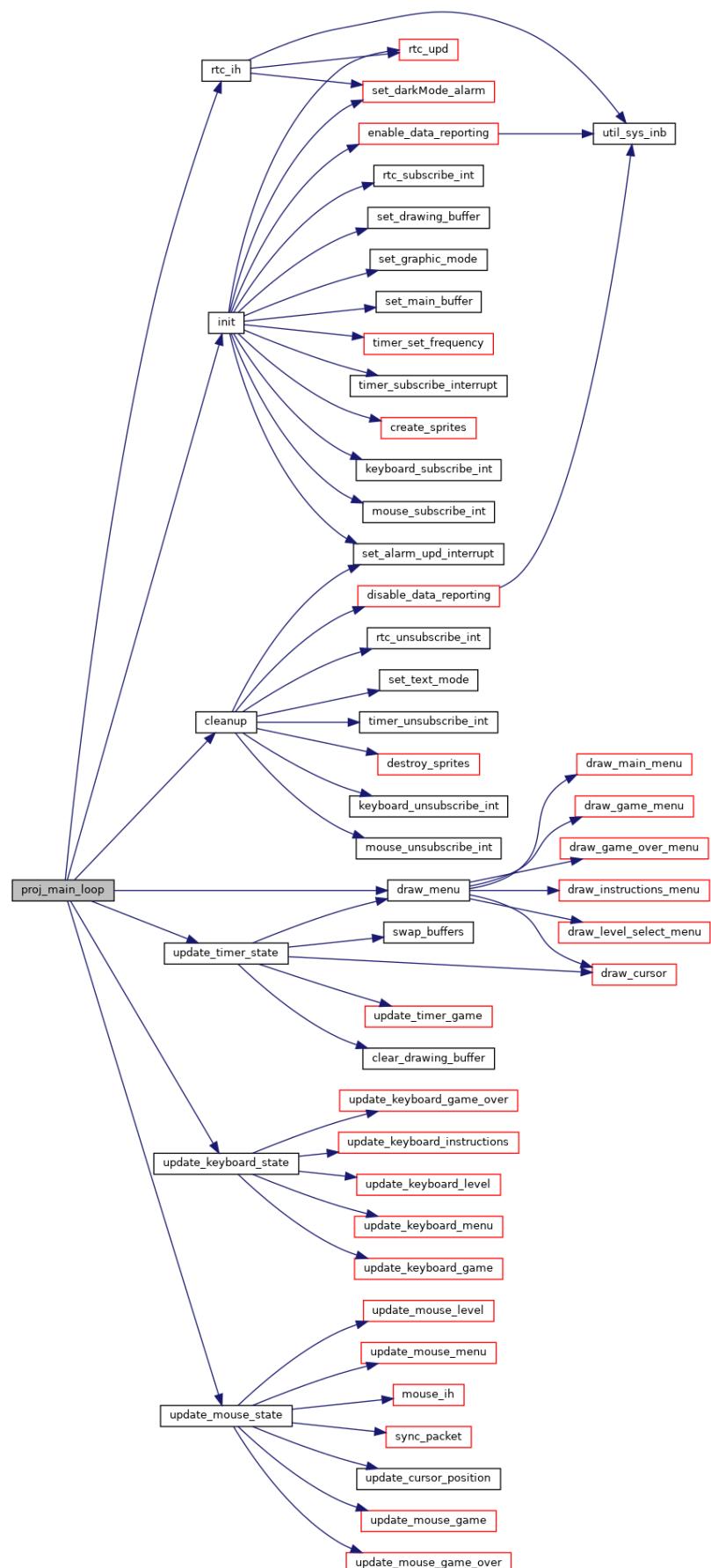
## 4.4 States – 25%

- state.c – 3%  
Responsável por atualizar os estados de todos os dispositivos utilizados (rato, teclado, timer...).
- menu\_state.c – 4%  
Encarregue de atualizar o estado dos elementos que pertencem ao menu principal.
- level\_selection\_state.c – 4%  
Responsável por atualizar o estado do nível do jogo.
- instructions\_state.c – 4%  
Responsável por atualizar o estado do menu de instruções.
- game\_state.c – 6%  
Encarregue de atualizar o estado de todos os elementos da tela de jogo.

- game\_over\_state.c – 4%

Encarregue de atualizar o estado do menu de game over.

## 4.5 Gráfico de chamadas de função



## 5. Detalhes de Implementação

Utilizámos uma estrutura *event driven*, na medida em que a função `driver_receive()` (apenas chamada no ficheiro “`main.c`”) fica encarregue de “apanhar” as interrupções recebidas dos diferentes dispositivos e traduz essas mesmas para os eventos correspondentes. Posteriormente, estes eventos são tratados pelos respetivos *event handlers*.

Em relação a máquinas de estado, o nosso projeto foi desenvolvido à volta de diferentes estados. A título de exemplo, o enum `Menu_State` definido no ficheiro (`../src/states/state.h`) – `{MENU, LEVEL_SELECTION, INSTRUCTIONS, GAME, GAME_OVER, EXIT}`.

Dependendo do estado, o comportamento das nossas funções podem mudar, o que faz com que, efetivamente, o projeto funcione como uma máquina de estados.

Como já referido anteriormente, foi implementado um sistema de colisões que deteta as colisões entre o panda e outros elementos do mapa (`collide()`), através de *pixel overlap*. Isto foi necessário, uma vez que um simples sistema de deteção de colisões através das bordas do objeto não permitiria que, após a rotação das tábuas, se assegurasse um bom contacto entre a tábuza e o panda e isso dificultaria também o movimento deste último.

## 6. Conclusões

Este projeto, misturado com o conhecimento previamente adquirido dos labs, permitiu-nos uma melhor compreensão e visão sobre os dispositivos do computador. Assegurou-nos a possibilidade de explorar as suas funcionalidades e configurações, sendo seguro dizer que foi um trabalho enriquecedor e vantajoso para nós.

Conseguimos implementar todas as funcionalidades a que nos tínhamos proposto inicialmente. A única alteração que decorreu foi a escolher os sprites – mediante a dificuldade em encontrar sprites de ratos, optou-se por escolher um de pandas.

Considerando que nos propusemos a fazer um jogo, tivemos que pensar em novas formas de como o poderíamos implementar, enquanto tentávamos aplicar os conhecimentos aprendidos nas aulas teóricas. Uma solução foi utilizar então a máquina de estados.

Desde o início do projeto, foi estipulado que talvez não teríamos tempo suficiente para implementar a porta série, no entanto seria uma ferramenta interessante a considerar no futuro. Também gostaríamos de considerar a possibilidade de adicionar mais mapas e níveis.

Apesar de não termos a porta série, consideramos que desenvolvemos um projeto do qual podemos sentir orgulho.

## 7. Soluções dos mapas

Estão aqui apresentados possíveis soluções para completar os mapas do jogo

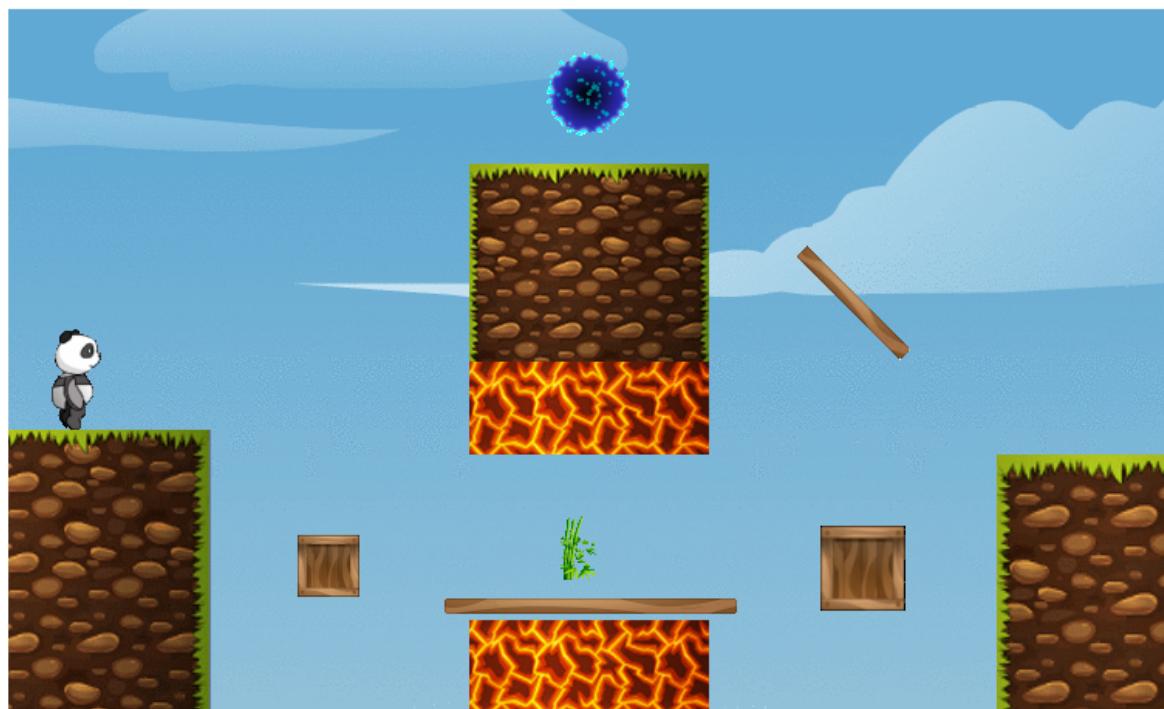


Figura 10 – Mapa 1

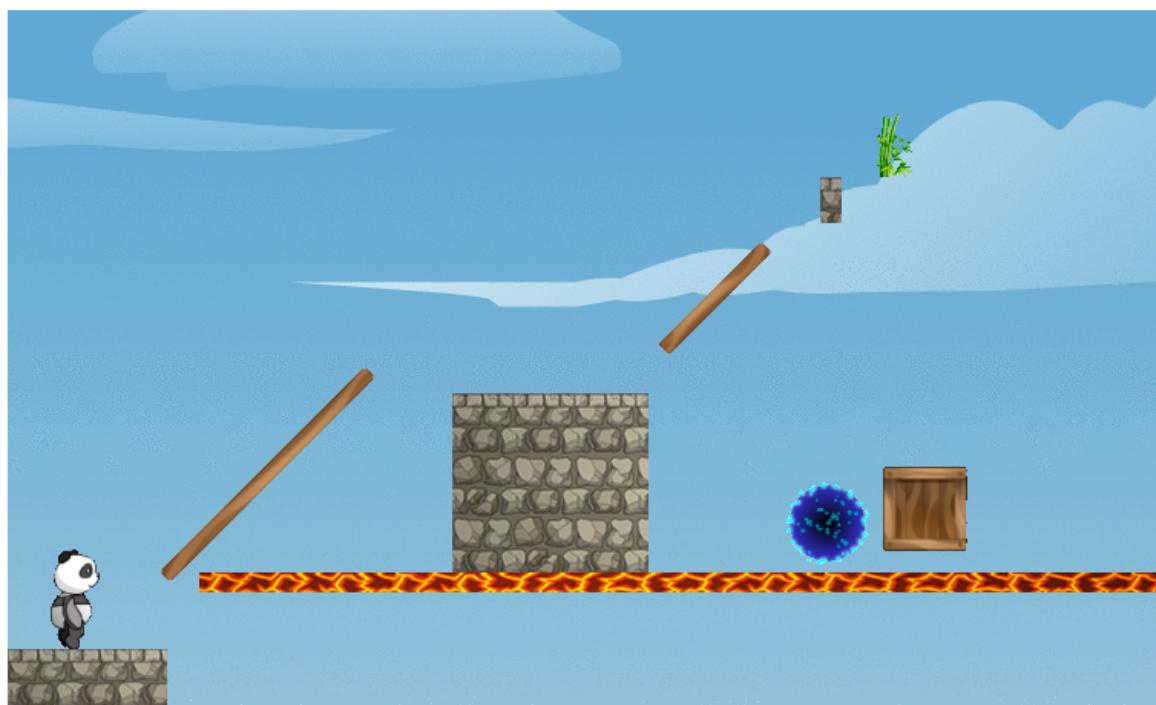


Figura 11 – Mapa 2

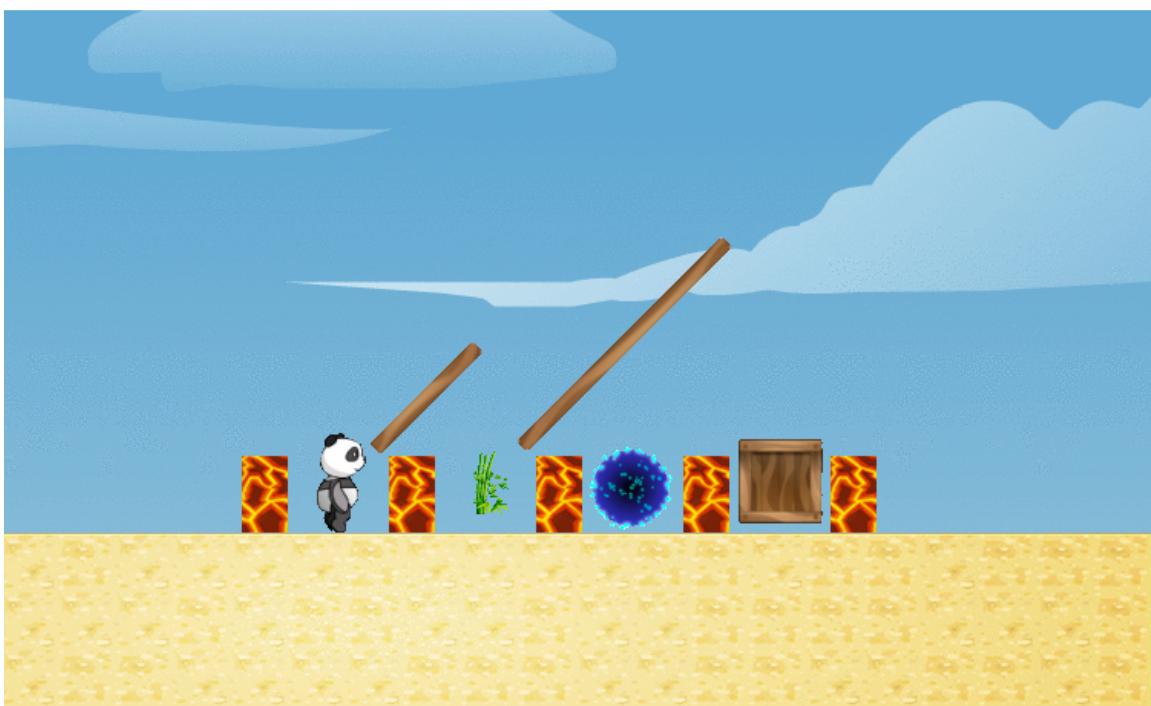


Figura 12 – Mapa 3