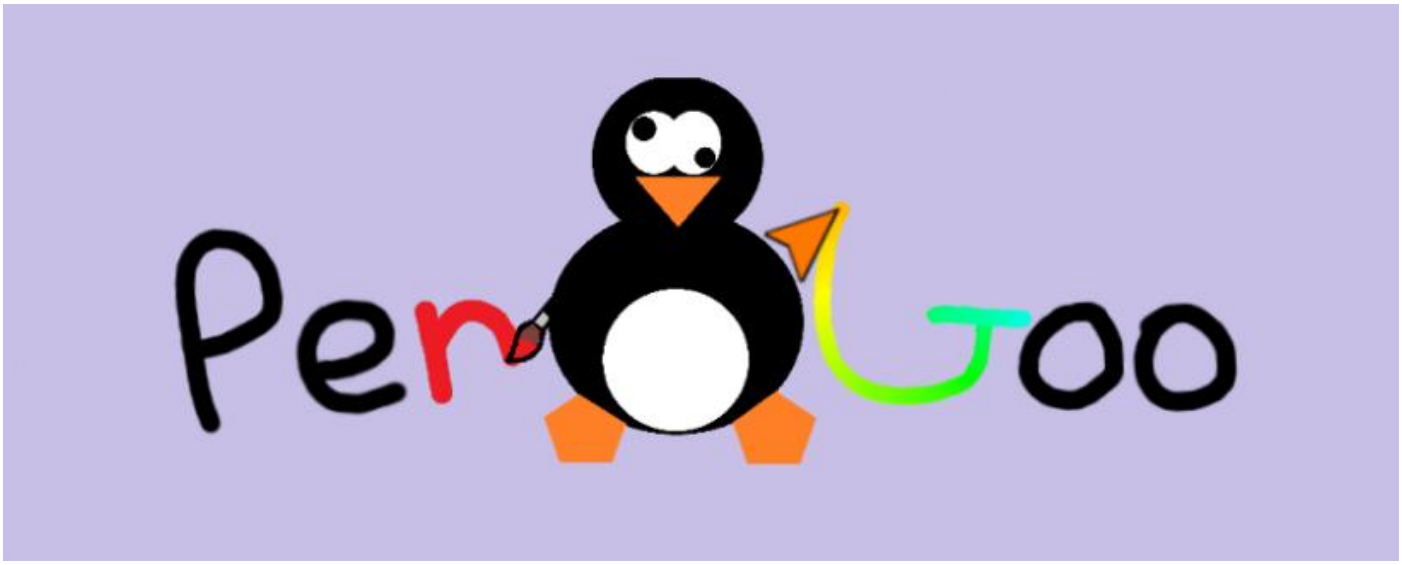
Laboratório de Computadores
2018/2019

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Pen-Goo



Grupo T4G10:

Daniel Ferreira Brandão, up201705812, up201705812@fe.up.pt

Pedro Miguel Moás, up201705208, up201705208@fe.up.pt

Índice

1.Instrucões de Utilização

1.1. Ecrã Inicial

Ao iniciar o jogo, é mostrado o ecrã inicial. Neste é possível ver as horas / data atual e avançar para o menu principal, pressionando a tecla “Enter”.

Figura 1- Ecrã inicial

1.2Menu Principal

A partir do menu principal, é possível aceder a todos os modos de jogo e respetivas instruções, assim como o botão de saída.

Figura 2 - Menu Principal

Para navegar neste menu, é utilizado o rato. Passar com o rato por cima de um dos botões faz com que esse fique “highlighted” , e clicar com o lado esquerdo do rato seleciona a opção correspondente.

Os modos de jogo disponíveis são: Play, Draw , Draw Together , Rainbow Snake e Flappy Rainbow.

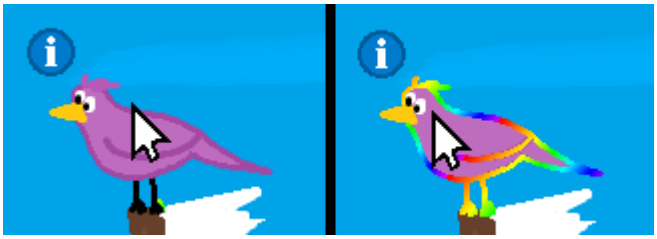


Figura 3- "hovering" com o rato

Figura 4- Instruções do modo Flappy Rainbow

1.3 Draw

Modo singleplayer, onde o utilizador pode criar os seus próprios desenhos.

Figura 5 - Modo Draw

Para além de começar um desenho em branco, é possível fazer load a ficheiros em formato bitmap e guardar as criações.

Figura 6 - Inserir nome para guardar ficheiro

Para criar o desenho, o utilizador tem acesso a múltiplas ferramentas de desenho:

- ***Slider* de grossura** – arrastar o cursor para aumentar ou diminuir a grossura do pincel.
- **Palete de cores –** abre uma palete de cores, de onde se pode escolher a cor principal (Botão esquerdo do rato) e secundária (Botão direito do rato)
- **Pincel –** ferramenta de desenho principal**.** Cor principal (botão esquerdo do rato) e secundária (botão direito do rato)
- **Arco-Íris –** Pincel com gradiente arco-íris
- **Balde –** Preenche uma área com a mesma cor
- **Borracha –** Usar o rato para apagar. Botão direito utiliza a cor principal e botão esquerdo a secundária.
- ***Color-Picker* –** Clicando em qualquer área do ecrã consegue-se obter qualquer cor.
- **Lixo –** Apaga o desenho.
- **Formas –** Permite desenhar formas com dois cliques do rato. Retângulo: Dois cliques representam dois cantos opostos. Círculo/Circunferência: Primeiro clique para o centro, segundo para o raio.

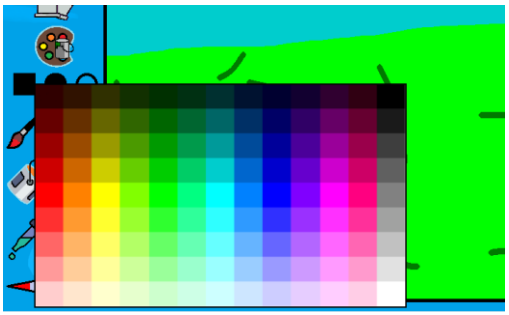


Figura 7 - Palete de cores

Para a maioria das ferramentas é possível utilizar Cor principal (botão esquerdo do rato) e secundária (botão direito do rato), exceto para a borracha, na qual acontece o inverso.

Para além das ferramentas de desenho é possível:

- Ver cor principal e secundária atual utilizando os guaches de cor
- Anular a última ação utilizando o botão de UNDO ou pressionar “Ctrl” + “Z”

Para selecionar a maioria destas ferramentas, utilizam-se botões clicáveis. A palete de cores abre um pop-up, onde se pode selecionar a cor desejada (botão direito para escolher a cor secundária) e o *slider* de grossura é um *slider* interativo.

Ao selecionar uma ferramenta, o cursor muda de imagem para a ferramenta escolhida e ao mudar de cor principal, a cor deste também muda.

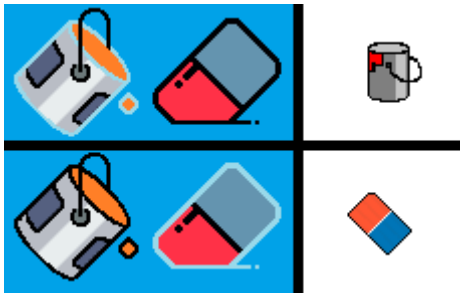


Figura 8- Mudança da forma do cursor

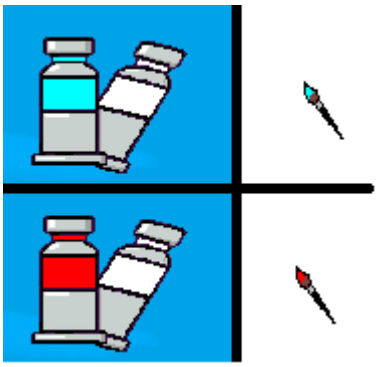


Figura 9-Mudança da cor do cursor

1.4 Play (Pen-Goo)

Modo principal do projeto, trata-se de um jogo de adivinhar cooperativo para dois jogadores.

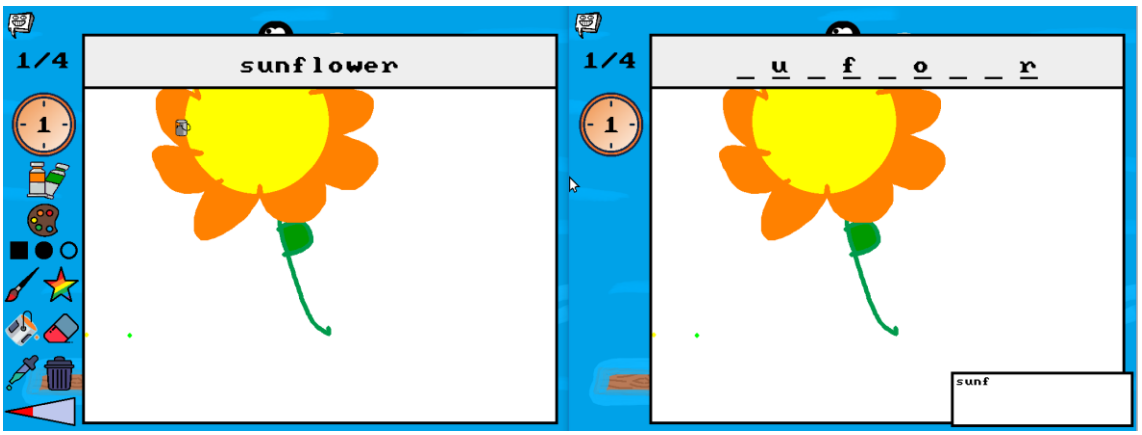


Figura 10 - Pen-Goo (vista dos 2 jogadores)

O objetivo do jogo é adivinhar a palavra que o parceiro está a desenhar o mais rápido possível. Cada sessão tem 4 rondas, nas quais ambos os jogadores terão oportunidade de desenhar. O papel de cada jogador é escolhido aleatoriamente, antes da primeira ronda começar. Cada ronda desenvolve-se da seguinte forma: É dado ao jogador que vai desenhar uma palavra aleatória do dicionário, o jogador tenta representar o que desenha, e a mesmo tempo, noutro computador, utilizando a porta de série, o outro jogador tenta adivinhar o que o parceiro está a tentar representar , usando o teclado, o mais rapidamente possível. O desenho é transmitido em tempo real para o computador do parceiro. O jogador que desenha tem acesso a todas as ferramentas de desenho do modo “Draw”.

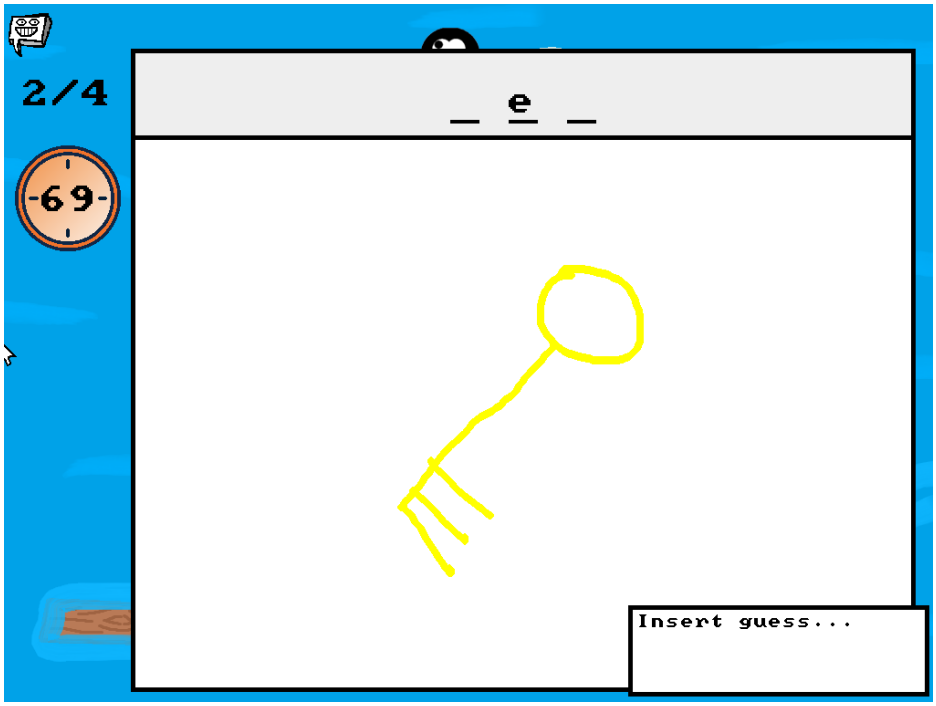


Figura 11 - Caixa de texto para tentar adivinhar a palavra

Para além do desenho do seu colega, o jogador que tenta adivinhar também saberá quantas letras tem a palavra que tenta adivinhar, e com o passar do tempo algumas delas vão sendo reveladas.

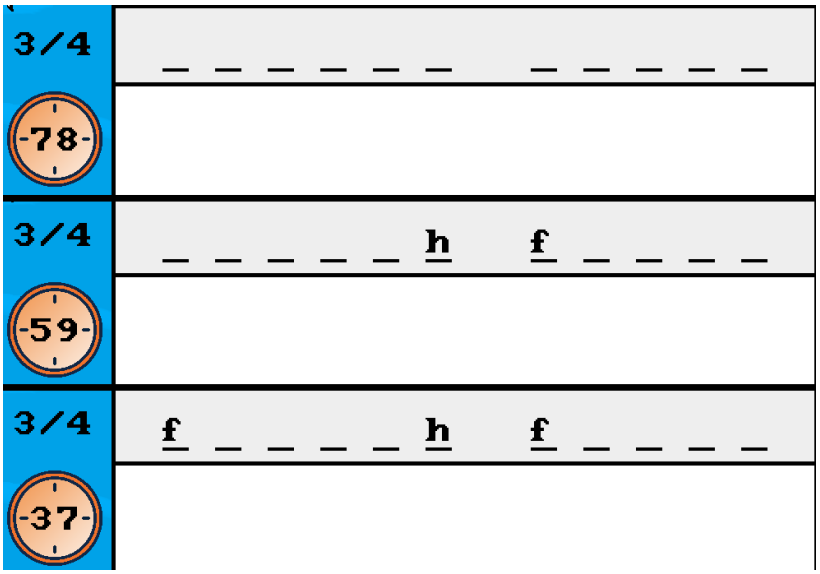


Figura 12 - Letras a serem reveladas à medida que o tempo passa

Quanto mais rápido o jogador adivinhar, mais pontos o grupo ganha na ronda.

Figura 13 - Ecrã de palavra adivinhada com sucesso

Os jogadores podem comunicar entre si utilizando emotes.

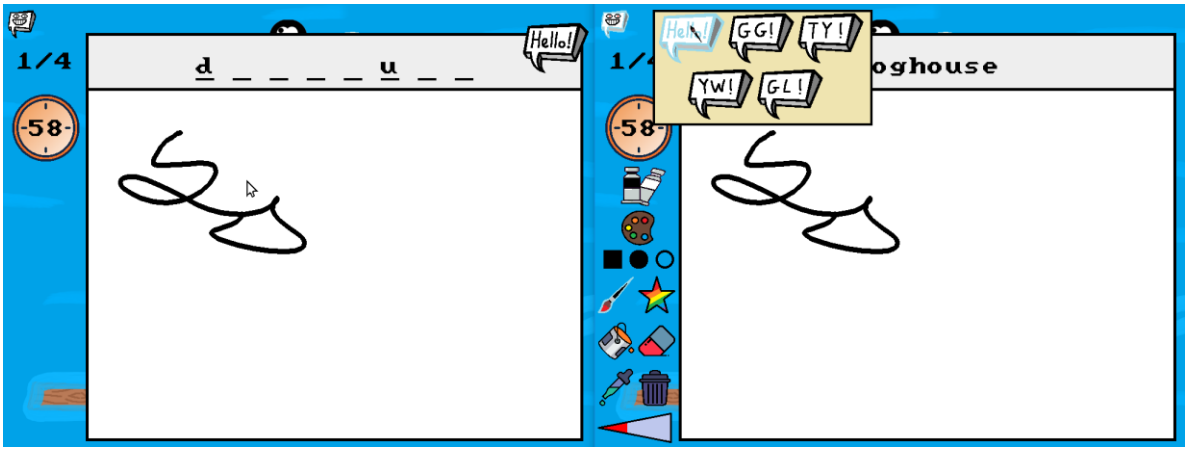


Figura 14 - Sistema de emotes (visto da perspetiva dos dois jogadores)

1.5 Draw Together

Modo de jogo igual ao "*Draw*", mas com um parceiro, através da porta de série. Os desenhos estão sincronizados em tempo real. Todas as ferramentas de desenho estão disponiveis para ambos os utilizadores, e ainda têm a possibilidade de guardar o desenho, tal como no modo "Draw".

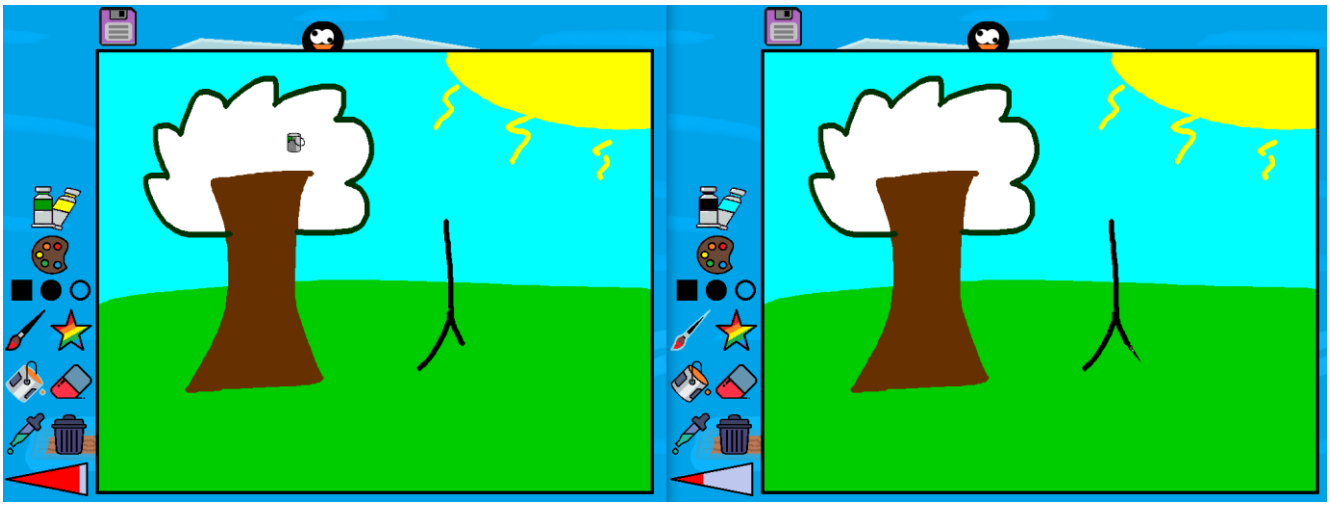


Figura 15 - Modo Draw Together

1.6 Rainbow Snake

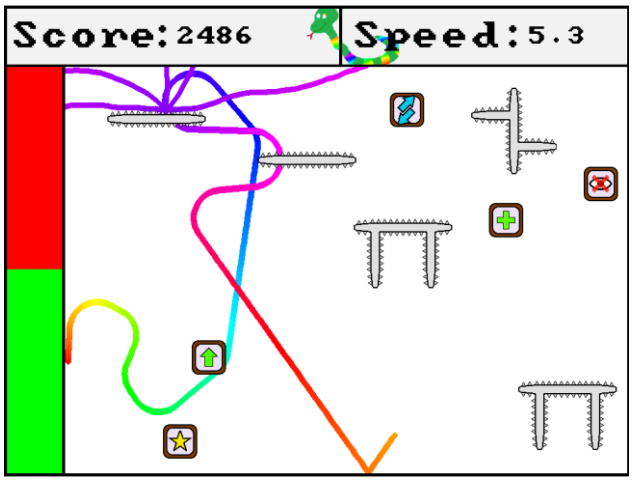


Figura 16- Rainbow Snake

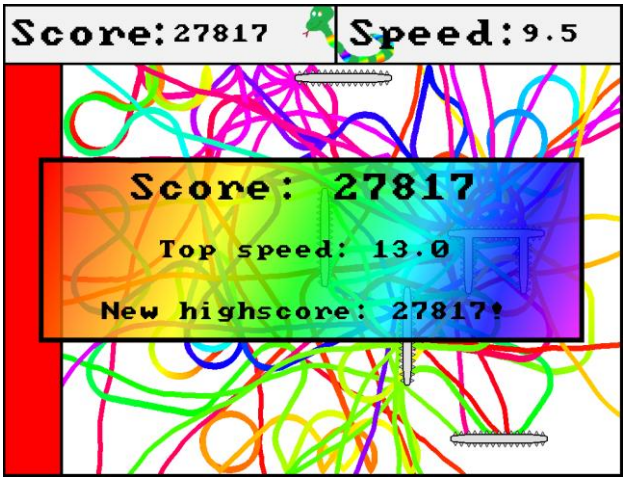


Figura 17-Ecrã de resultados do rainbow snake

O modo de jogo *rainbow snake* trata-se de um modo de jogo secundário, que se diferencia bastante do resto do projeto. O objetivo é sobreviver o maior tempo possível, controlando a *rainbow snake*. Com o passar do tempo a *snake* vai perdendo vida e ganhando velocidade, por isso é necessário utilizar os *power-ups* disponíveis para sobreviver e maximizar a pontuação. A *snake* é controlada com as setas do teclado, podendo apenas virar para a esquerda ou direita.

Atingir os obstáculos faz com que a *snake* perca vida e atingir uma parede faz com que seja refletida e que perca velocidade.

Figura 18 – Power-ups de rainbow snake, visualizados nas instruções do jogo

O highscore é gravado num ficheiro de texto.

1.7 Flappy Rainbow

Outro modo de jogo secundário, *Flappy Rainbow*, baseado no conhecido jogo mobile, *Flappy Bird*, consiste em sobreviver o máximo de tempo possível, evitando obstáculos, apenas com salto (tecla "space bar") e conhecimento da física do jogo. O *flappy rainbow* tem velocidade e aceleração (para baixo), para tornar o jogo mais consistente com a realidade. O jogador perde quando o *flappy rainbow* colidir com um dos obstáculos. Tal como o *rainbow snake*, o highscore é gravado num ficheiro de texto, para registo futuro.

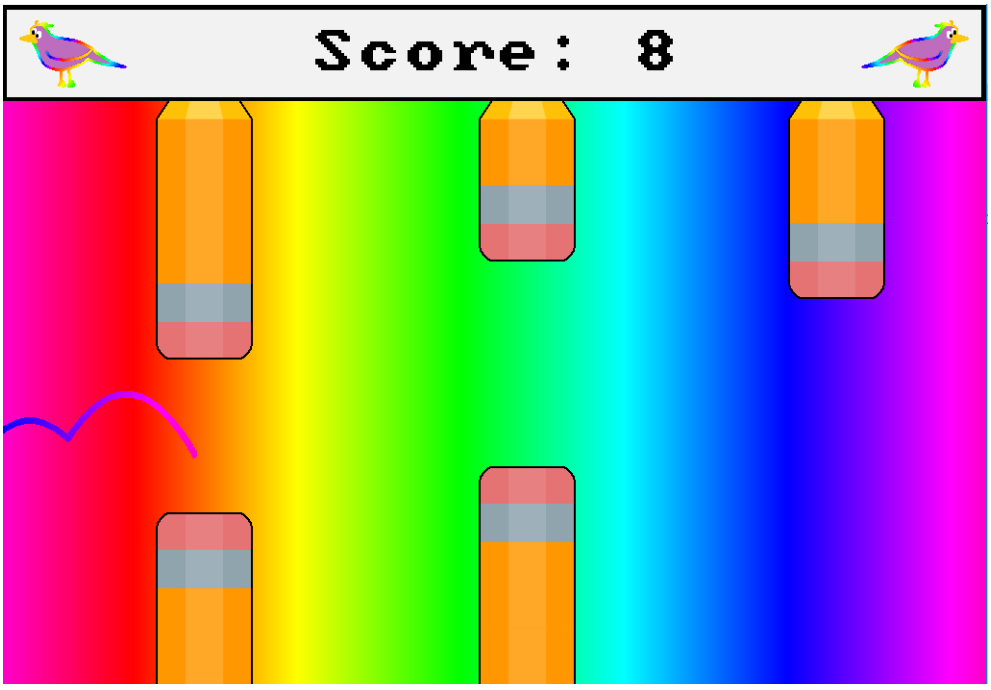


Figura 19 - Flappy Rainbow

2. Estado do Projeto

Dispositivo	Utilização	Interrupção?
Timer	Controlo da frame rate nos minijogos, revelar letras periodicamente, animações a mostrar scores.	Sim
Teclado	Escrever texto no ecrã, controlar *rainbow snake* e *flappy rainbow*, alguns *shortcuts.*	Sim
Rato	Navegação nos menus, desenhar, escolha de ferramentas de desenho.	Sim
Placa Gráfica	Visualização de imagens, desenho e menus.	Não
RTC	Mostrar data e hora, atualização do relógio de jogo.	Sim
Porta de série	Modos de jogo multiplayer "Play" e "Draw Together"	Sim

- **Timer**: O timer 0 é usado para controlar a frame rate, atualizando o desenvolvimento do jogo nos modos "*Rainbow Snake*" (*snake.c*) e "*Flappy Rainbow*" (*flappy.c*), para controlar o tempo entre letras reveladas (*wordpicker.c – reveal_letter()*), e para mostrar a pontuação final não espontaneamente.
  Está implementado em**:** timer.c

- **Teclado:** O teclado é utilizado para escrever no ecrã, a partir de caixas de texto (*textbox.c*) (adivinhar palavra, guardar nome de ficheiro, fazer load a ficheiro), para sair de certos modos de jogo e fazer *undo* no desenho ("Ctrl" + "Z") , mas também para controlar a *snake* (setas) (*snake.c – snake_turn_left/right()*) e o *flappy* ("Space bar") (*flappy.c - flappy_jump()*).
  Está implementado em: keyboard.c

- **Rato:** Usado para controlar o cursor, que é utilizado para navegar nos menus, selecionar ferramentas, a partir de botões, e desenhar, não só para escolher onde desenhar, mas também para escolher se se utiliza a cor principal (botão esquerdo do rato) ou secundária (botão direito do rato).
  Está implementado em: mouse.c

- **Placa Gráfica:** Este dispositivo é utilizado para mostrar todas as imagens usadas no programa (bitmap, RGB(8;8;8)). O modo de vídeo utilizado é 118 (1,024×768, 24 bits por píxel)), sendo que o número de cores disponíveis é 16 777 216. (*proj.c*)
  Double buffering está implementado no código do programa, mas não é utilizado, pois devido ao facto de que se trata principalmente de uma ferramenta de desenho, não muito focada em atualização frequente do ecrã inteiro, não faria muita diferença, tornando a experiência mais lenta até. (*video.h*)
  Foi utilizada colisão de *sprites* para o cursor e botões, e hitboxes para os modos de jogo "Rainbow Snake" e "Flappy Rainbow". (*sprite.c - is_on_button()*) (*sprite.c- check_hitbox_collision()*) Um sistema de *layers* está a ser utilizado de modo a ser possível desenhar facilmente janelas por cima de outras sem perder a informação que está "por baixo". (*layer.c*)
  A fonte utilizada para a textbox, para mostrar as pontuações dos jogadores, tempo do relógio, etc., é retirada de um bitmap que contém todos os carateres, que depois são retiradas individualmente a partir de um algoritmo. (*textbox.c*)
  Está implementado em: video.c vbe.c

- **RTC :** O Real-Time Clock utiliza alarmes de 1 segundo para atualizar a hora / data atual periodicamente no Ecrã inicial (*clock.c – update_big_ben()*) e para atualizar o relógio de jogo (90s por ronda no máximo). (*wordpicker.c – wordgame_tick_clock()*)

  Está implementado em: rtc.c

- **Porta de série:** A porta de série utiliza FIFO, Interrupções e, por vezes, *polling*, para transferir informação entre dois computadores diferentes. A porta de série é um dispositivo crucial para o funcionamento dos modos "Play" (Adivinhar) e "Draw Together", nos quais se transferirão até cerca de 1.5 KBytes por segundo, potencialmente ao mesmo tempo, de um computador para outro. Através desta são enviadas várias informações, como quem será o primeiro a desenhar, que um jogador está pronto, qual a palavra a adivinhar, se o jogador adivinhou a palavra, que o outro computador desenhou uma linha, numa certa posição, com uma certa cor, entre outras. Para facilitar a gestão da informação recebida/enviada, para ambos os computadores saberem o significado de cada byte recebido, e para não haver dessincronizações que prejudiquem muito o desenvolvimento do jogo, criou-se um protocolo relativamente simples, para a comunicação neste projeto, ilustrado nas imagens abaixo. As mensagens são então constituídas por um prefixo, que contém também o tipo de mensagem e o seu tamanho, e por um sufixo. Mesmo que haja informação que por vezes seja redundante, considerou-se que por segurança guardar-se-ia na mesma.

  Implementado em uart.c, uart_protocol.c, e uart_wordgame.c (este último para aplicar o protocolo a casos do jogo em concreto).

Protocolo de Comunicação utilizado

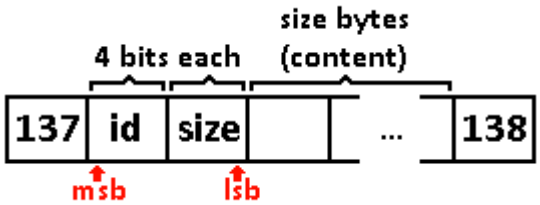


Figura 20 – Protocolo de comunicação usado para o UART, byte a byte

ID	Significado	Size	Content
0x0	Chamar draw_line() com os argumentos recebidos	12	Por ordem: xi, yi, xf, yf, cor, grossura
0x1	Chamar draw_line2() com os argumentos recebidos	12	Por ordem: xi, yi, xf, yf, cor, grossura
0x2	Limpar a tela.	0	----
0x3	Utilizar balde com na posição indicada	7	Por ordem: x, y, cor
0x4	Desenhar forma indicada na tela, nas posições indicadas	13	Por ordem: forma, x1, y1, x2, y2, cor, grossura
0x5	O jogador a desenhar está pronto.	?	String com a solução da ronda.
0x6	O jogador a adivinhar está pronto.	0	----
0x7	O jogador a adivinhar acertou na palavra.	0	----
0x8	O jogador a adivinhar não acertou na palavra a tempo.	0	----
0x9	Alterar o tempo restante.	1	Tempo restante
0xA	Desenhar um emote no ecrã.	1	Emote (enum)
0xB	Apenas um número, escolhe o primeiro a desenhar.	4	1º Byte: LSB 4º Byte: MSB
0xC	O jogador está pronto para um desenho colaborativo	0	----

*Nota: Todas as informações constituídas por mais do que um byte são enviadas desde o LSB até ao MSB, por ordem.
**Nota: As posições são constituídas por 2 Bytes e as cores por 3 Bytes. Todos os outros argumentos correspondem a 1 Byte do conteúdo da mensagem.

Figura 21 – Tipos de mensagens enviadas pelo UART, no programa

3. Organização / Estrutura de código

Timer

Código importado do lab2. Utiliza-se para subscrever as interrupções e interrupt handler do timer.

Keyboard

Código importado do lab3. Utiliza-se para subscrever as interrupções e interrupt handler do teclado.

Mouse

Código importado do lab4. Utiliza-se para subscrever as interrupções e interrupt handler do rato.

Vbe

Código importado do lab5. Utiliza-se para obter informações do modo vbe utilizado (aquando da inicialização do Minix em modo gráfico).

Video

Código importado do lab5. Contém funções para inicializar o Minix em modo gráfico, e também funções básicas de manipulação de píxeis (desenhar um píxel, obter a cor de um píxel, pintar um retângulo ou uma linha horizontal, etc.)

Rtc

Lê a informação dos registos do Real-Time Clock.

Subscreve alarmes periódicos de 1 segundo, para ler a data/horas em todos os segundos. Uso das struct Date / Mini_Date para obter as datas a partir das funções rtc_get_date/mini_date();

Uart

Utiliza-se para subscrever as interrupções e interrupt handler do UART, mas também outras funções, como ler o seu status register, ler ou enviar um byte através da porta de série.

Uart_protocol

Código utilizado para definir, criar e enviar mensagens pela porta de série, através de um protocolo definido (descrito na secção anterior).

Uart_wordgame

Utiliza-se para enviar mensagens e processar mensagens recebidas, no contexto do projeto, por exemplo, permite enviar uma mensagem indicando que deve ser desenhada uma linha desde uma posição até outra, com uma certa cor e espessura, como também processar essa mesma mensagem. O formato foi descrito na secção 2.

Interrupts

Módulo onde existem funções para dar subscribe e unsubscribe e receber notificações de todos os periféricos utilizados de forma mais limpa e intuitiva. Uso da struct Notification para, a partir da função GetNotification() obter todas as informações convenientes da notificação recebida.

Event

Módulo similar a interrupts, mas com ainda mais abstração do funcionamento do Hardware, contendo então ainda mais informação sobre o estado do hardware, como o estado dos botões do rato, de alguns botões do teclado. Um evento também indica que tipo de interrupção/evento ocorreu, e para o próprio evento, que tipo de evento ocorreu (por exemplo, um KeyboardEvent pode ter mais de 20 tipos, como CTRL_PRESS, ARROW_DOWN_RELEASE, CHARACTER_PRESS – neste caso é indicado também o carater que foi premido).

PenGoo

Este modulo é a base do projeto, contem a lógica dos menus do projetor, de conectar jogadores em modos multiplayer – select_drawer(), wait_for_collab(), wait_for_drawer(), wait_for_guesser(), de mudar as ferramentas / grossura e cor que está a ser utilizada - change_tool().

Bitmap

Este ficheiro é utilizado para fazer load / save a ficheiros em formato bitmap, mas também para desenhá-los. Ao serem desenhados, os bitmaps podem ter transparência – representado pela cor ROSA (rgb (255,0,255)) e uma cor mudável – representado pela cor VERDE (rgb(0,255,0)). Esta permite mudar uma certa parte do bitmap, para qualquer cor desejada (Utilizado para mudar a cor do cursor para a cor primária, por exemplo). Neste módulo estão também implementadas funções que rodam e recortam bitmaps - draw_bitmap_rotate() e ainda uma função que desenha o bitmap translúcido - draw_bitmap_transp().

Bitmaps

Módulo onde todos os bitmaps utilizados no projeto são lidos dos seus ficheiros e guardados em variáveis.

Sprite

Módulo onde estão implementadas as Sprites e todas as suas “Classes derivadas”. A *sprite* pode ser desenhada (tendo em atenção a transparência e a sua cor alterável), apagada e movida. Existem também funções que atuam em sprites, mas destinadas a um *Sprite* que representa o cursor do rato, como funções para atualizar a posição, mudar de bitmap e cor (para poder representar a ferramenta e cor primária que estão a ser utilizadas). Existem também outras, como a struct Hitbox (utilizada para o jogo da snake e Flappy), com as suas funções para mover e para detetar colisões, a struct Button (usada para navegação nos menus e escolha de ferramentas), com funções para detetar se o cursor esta a fazer hover, fazer highlight ao botão e pressioná-lo, a struct IdleSprite, que se trata de um sprite não interativo que pode mudar de cor (utilizado para mostrar cor primária e secundária atual) e a struct slider (utilizado para o slider dinâmico de grossura), com funções para mover e atualizar, que estão implementadas neste ficheiro.

Layer

Módulo onde está implementado o sistema de *layers* que permite desenhar píxeis ou *sprites* numa layer - draw_on_layer(), layer_draw_image() - em cima de outra layer, sem perder informação sobre o que está "por baixo". Graças a este sistema, é possível abrir e fechar janelas por cima de outras, ao utilizar o programa, sem perder a informação que está por baixo, usando também funções que verificam qual é a layer que está em cima – get_highest_layer() e se certa layer é a do topo - is_top_layer().

Canvas

Implementação da struct canvas que é utilizada como espaço de desenho para o utilizador. O canvas tem uma layer e limites. É aqui que se encontram as funções de desenho principais como canvas_draw_line(), canvas_draw_circle(), etc.. As ferramentas de desenho como o rainbow, balde e formas estão aqui definidas.

O sistema de undo é feito utilizando as funcões canvas_save_drawing() e canvas_undo().

Textbox

A textbox é utilizada para escrever texto, quer pelo utilizador, ou para mostrar informações. É implementada utilizando uma struct que contém o bitmap da imagem da textbox, layer, tamanho da fonte, coordenadas da textbox e do cursor utilizado para escrever texto.

A fonte utilizada é lida a partir de um ficheiro bitmap com a função loadLetterMap().

Para escrever numa caixa de texto são utilizadas a funções textbox_put() e textbox_write(), e textbox_backspace() para apagar.

Este módulo também contém duas funções que não utilizam a struct Textbox, draw_char() e draw_word(), mas são relevantes para o módulo pois permitem, respetivamente, desenhar no ecrã um carater e uma string, numa posição especificada.

Clock

Neste ficheiro está implementado o relógio utilizado para mostrar as horas/data no ecrã inicial, que utiliza o rtc.

Emote

Implementação do sistema de emotes que são enviados entre jogadores nos modos multiplayer a partir da serial port. Permite desenhar emotes no ecrã. Também tem funções para abrir ou fechar a "emote wheel", através da função toggle_emote_wheel().

Wordpicker

Este módulo trata de ler as palavras que vão ser adivinhadas / desenhadas de um ficheiro de texto (dicionário). Também contém o mecanismo de mostrar a palavra escondida que está a ser adivinhada utilizando as funções wordgame_draw_hidden_word() para mostrar a palavra escondida (só com os '_' em vez de letras) e reveal_letter() para revelar letras à medida que o tempo passa. Este módulo também contém a função verify_guess() que verifica se a tentativa de adivinhar estava correta, ignorando diferença entre letras maiúsculas/minúsculas.

Snake

Módulo que contém todo o modo de jogo secundário "Rainbow Snake". As structs principais utilizadas neste modo de jogo são:

- Snake – contém as coordenadas, velocidade (e velocidade máxima), ângulo, vida, tempo restante para alguns powerups terminarem, etc. Para controlar a snake as funções snake_move() e snake_turn_right/left() são essenciais.
- Powerup – contém uma hitbox, bitmap , tipo de power , etc.
- Obstacle - contém uma hitbox, coordenadas e bitmap.
- SnakeArena – contém os limites da arena.

A função snake_move() verifica colisões com obstáculos, powerups e parede. Ao apanhar um powerup ou houver colisão com obstáculos a barra de vida da snake vai ser atualizada snake_update_hp_bar().

No inicio do jogo diferentes obstáculos são colocados na arena de forma aleatória, mas a evitar sobreposições - arena_add_obstacle(). Durante o jogo são adicionados à arena powerups - arena_add_powerup().

No final do jogo é mostrado os stats aos jogadores - snake_showstats() que utiliza um bitmap translúcido. O highscore é guardado num ficheiro de texto - saveSnakeHighscore() , loadSnakeHighscore().

Flappy

Módulo onde esta implementado o modo de jogo "Flappy Rainbow". As structs utilizadas são:

- Bird – contém coordenadas , velocidade , aceleração , hitbox, raio do círculo, etc. do pássaro
- FlappyTube – contém bitmap, hitbox e um booleano que indica se o obstáculo já foi pássaro.

Para o jogo avançar, é utilizada a função flappy_tick() que avança o ecrã para a esquerda, utiliza a função flappy_move() que atualiza a posição do pássaro com base na sua aceleração e velocidade e a função flappy_collided() verifica se houve colisões com os obstáculos.

Para controlar o pássaro, a função flappy_jump() é utilizada para fazer com que ele salte.

Os obstáculos são adicionados com a função flappy_add_tube().

No final do jogo é mostrado os stats aos jogadores - flappy_showstats(). O highscore é guardado num ficheiro de texto - saveFlappyHighscore(), loadFlappyHighscore().

Game_info

Módulo que contém as instruções de cada modo de jogo, que estão disponíveis no menu principal, incluindo funções para as mostrar/retirar do ecrã – toggle_instructions().

Game_State

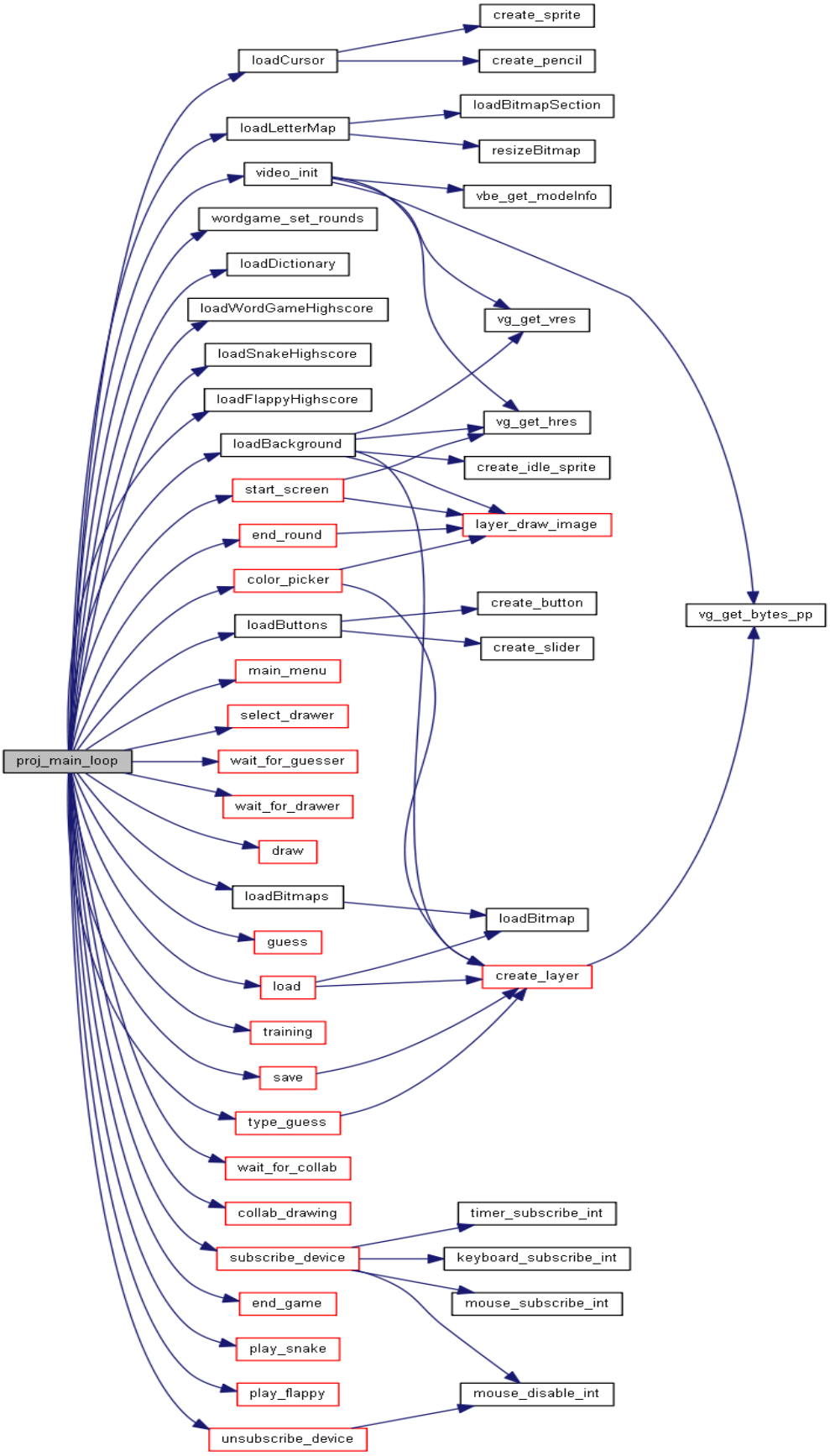
Este módulo contém as definições das structs com o estado do programa assim como do estado do "pincel" (inclui cor, ferramenta, grossura, etc.). Através da função changeState(), consegue-se mudar o estado do jogo e também conservando o atual numa variável lastGameState. Para além disso, contém também algumas funções que possam gerir o estado do programa. Por exemplo, create_pencil() que inicializa o estado do "pincel", ou useTextbox(), que analisa um evento e altera uma caixa de texto, podendo ainda regressar para um estado anterior ou progredir para um seguinte, com as teclas ESC ou ENTER.

Distribuição dos diferentes Módulos pelo programa

MÓDULO	PESO RELATIVO (%)	FEITO POR
MOUSE	5%	Daniel/Pedro
RTC	1%	Daniel
TIMER	2%	Daniel/Pedro
KEYBOARD	3%	Daniel/Pedro
VBE	0.5%	Daniel/Pedro
VIDEO	3%	Daniel/Pedro
UART	3%	Pedro
UART_PROTOCOL	2%	Pedro
UART_WORDGAME	2%	Pedro
INTERRUPTS	5%	Daniel/Pedro
EVENT	10%	Daniel/Pedro
PENGOO	20%	Daniel/Pedro
BITMAP	5%	Pedro
BITMAPS	1%	Daniel/Pedro
CANVAS	8%	Daniel/Pedro
SPRITE	7%	Daniel/Pedro
TEXTBOX	3%	Pedro
LAYER	2%	Pedro
CLOCK	1%	Daniel
EMOTE	2%	Daniel
WORDPICKER	3%	Pedro
SNAKE	5%	Pedro
FLAPPY	5%	Daniel
GAME_INFO	0.5%	Pedro
GAME_STATE	1%	Daniel/Pedro

Call Graph

Nota: Devido a ser demasiado fundo, omitiu-se grande parte do gráfico, estando a restante informação presente na documentação gerada por doxygen.



4.Detalhes de implementação

De um modo geral, achamos que maioria de o que foi utilizado no projeto foi bem abordado nas aulas laboratoriais de LCOM. No entanto, mesmo que alguns conceitos sejam tenham sido dados nas aulas teóricas, requereram mais algum cuidado e estudo, para que fossem corretamente implementados:

- ***Layering* / Eventos**: Tratar das interrupções de todos os dispositivos ao mesmo tempo foi desafiante no início pois utilizávamos individualmente as funções de cada dispositivo, quando necessário. Para tornar o código mais limpo, e fácil de utilizar, decidimos criar um módulo de interrupções para utilizar as funções de cada dispositivo de uma forma mais universal. Mais tarde, acabámos por ainda criar um módulo de eventos que está ainda mais abstraído do que o módulo de interrupções, sendo bastante mais fácil de utilizar e necessita de quase nenhum conhecimento sobre o funcionamento do hardware, como foi descrito na secção 2.
- **Máquinas de Estado:** No projeto, utilizou-se uma implementação de uma "state machine", que foi essencial para controlar o estado atual do programa em proj.c, no loop principal do programa. Os estados utilizados estão definidos em game_state.c, e facilmente se consegue alterar o estado através de funções definidas também nesse ficheiro.
- **Programação Orientada a Objetos:** Em grande parte dos módulos aplicámos conceitos de OOP em C, utilizando structs como classes e funções como métodos das classes.
- **RTC** – apesar de não diferir muito dos outros dispositivos dados nas aulas, não nos foi óbvio qual era a maneira mais eficaz de atualizar o relógio. Acabamos por definir alarmes de segundo a segundo para saber quando atualizar o tempo, já que essa funcionalidade era necessária para outra parte do projeto.
- **Serial Port** – Embora não consideremos que seja o dispositivo mais difícil por uma grande margem, é certamente o mais problemático para dar debug, por várias razões: 1. Dificuldade em analisar o trace.txt / output.txt pois não indicam qual dos computadores originou as mensagens, 2. tanto a transmissão como a receção podem estar a causar erros, 3. na maioria dos erros que foram ocorrendo, apenas um *poweroff* permitia a UART regressar a um estado funcional, 4. Os erros que ocorriam normalmente impediam a ocorrência de novas interrupções, entre outras. Na implementação também foi necessário criar um protocolo de comunicação que pudesse facilmente ser aplicado no programa (descrito na secção 2).
- **Layers** – Desde o início do projeto achamos que era necessário implementar um sistema de layers, de modo a ser possível termos janelas "pop-up" para podermos escrever texto, escolher cores de forma menos limitada, mostrar scores, etc, por cima do que já está a ser mostrado.
- **Colisões e hitboxes** – Para detetar se o cursor estava a fazer hovering com os botões, foi necessário aplicar um método de colisões com as sprites simples, que verifica se o cursor se encontra por cima de algum pixel não transparente. Para além disso foram aplicadas hitboxes (retangulares) nos modos de jogo secundário "Rainbow Snake" e "Flappy Rainbow" para detetar colisões, uma forma bastante mais eficiente, para sprites aproximadamente retangulares.

Para além dos módulos, algumas funções necessitaram de algoritmos mais complexos, o que requereu alguma investigação da nossa parte:

- **Algoritmos de círculos , linhas** – Para poder desenhar círculos e linhas não horizontais/verticais tivemos de utilizar algoritmos pré-existentes (https://en.wikipedia.org/wiki/Midpoint_circle_algorithm , https://en.wikipedia.org/wiki/Bresenhams_line_algorithm)
- **Balde** - A ferramenta de desenho balde foi alvo de muitas alterações ao longo do projeto. Para encontrar o algoritmo mais otimizado experimentamos várias versões (https://en.wikipedia.org/wiki/Flood_fill) até encontramos a mais fluida. Inicialmente pensámos num algoritmo recursivo que verificava os 4 píxeis adjacentes, chamando a função recursivamente por cada píxel que tivesse a mesma cor inicial. O algoritmo foi simples e direto, mas (não muito surpreendentemente) causava *stack overflow* para regiões muito grandes. De seguida tentámos um algoritmo que teria um efeito semelhante ao anterior, mas de forma iterativa, utilizando uma *queue*, que conteria os píxeis a tratar, sendo que por cada píxel da *queue* os 4 adjacentes seriam verificados e os que teriam a sua cor igual à inicial seriam colocados na *queue*. O algoritmo era bastante mais complexo, mas mesmo com um enorme conjunto de otimizações à implementação do algoritmo e da *queue* (havendo até uma tentativa de implementar a *queue* através de um *array* com capacidade de 1500000, sendo o máximo de eficiência **temporal** que atingimos), demorava quase 0.25s para encher o ecrã inteiro (embora pareça curto, estraga bastante a experiência do jogador). No final, tentámos um terceiro algoritmo, que analisa os píxeis numa linha horizontal, e para cada linha analisa as suas duas adjacentes recursivamente, o que refletiu-se ser quase instantâneo em qualquer situação.

5.Conclusão

Avaliação da unidade curricular

Na nossa opinião, LCOM foi uma unidade curricular que se começou de forma brusca devido a introdução repentina de vários mecanismos de trabalho simultaneamente. A introdução do SVN, Minix, programação em C e matéria de LCOM tudo ao mesmo tempo foi um choque desencorajador. No entanto, à medida que os labs passavam, a matéria ficava mais esclarecida, e a unidade curricular foi-se tornando mais interessante.

Devido ao certo cuidado e trabalho necessário para fazer os labs, LCOM tornou-se por uma grande margem a unidade curricular que tivemos de investir mais horas semanais, o que não se reflete de todo nos ECTS definidos para a disciplina.

Outro ponto a destacar é o atraso na avaliação dos labs e do miniteste, tendo em conta que o primeiro lab (lab2) foi entregue no dia 14 de Outubro, e até à entrega do projeto não houve uma única avaliação entregue, o que prejudica os alunos, pois não só não conseguem ter uma ideia clara do seu desempenho até ao momento, como também haverão certos erros nos labs que serão repetidos nos labs seguintes, o que poderia ser evitado.

Por outro lado, LCOM ajudou-nos a ser mais autónomos na aprendizagem de certos assuntos, e aumentou o nosso conhecimento sobre a linguagem C. O ponto alto da unidade curricular foi sem dúvida este projeto.

6. Apêndice

Instruções de utilização

Para além de invocar *make*, é necessário garantir que a porta de série está corretamente configurada, e que os **seis** *paths* declarados em proj.c sejam definidos corretamente:

- bitmap_folder – *path* para a localização da pasta bitmaps, que contém todos os ficheiros .bmp necessários. O programa não funcionará sem a sua definição correta.
- Words_path - *path* para a localização do ficheiro .txt que contém o dicionário do jogo de adivinhar. O jogo não funcionará sem a sua definição correta.
- Saved_img_folder – *path* para a localização da pasta onde se guardarão as imagens desenhadas. Sem a sua definição correta não será possível guardar nem carregar imagens já desenhadas.

Recomenda-se que os 3 seguintes ficheiros se guardem na mesma pasta, para melhor organização:

- Snake_hs_path – *path* para a localização do ficheiro .txt que contém o highscore do jogo "Rainbow Snake". Sem a sua definição correta não será possível guardar o highscore deste jogo.
- Flappy_hs_path – *path* para a localização do ficheiro .txt que contém o highscore do jogo "Flappy Rainbow". Sem a sua definição correta não será possível guardar o highscore deste jogo.
- Wordgame_hs_path – *path* para a localização do ficheiro .txt que contém o highscore do jogo principal. Sem a sua definição correta não será possível guardar o highscore deste jogo.