U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

SNOOK8

LCOM 2023/2024

Sérgio Rego Nossa (up202206856@fe.up.pt)
Pedro Marinho Silva Roleira Marinho (up202206854@fe.up.pt)
Eduardo Castro Portugal Ferreira (up202206828@fe.up.pt)
Xavier dos Santos Portugal Martins (up202206632@fe.up.pt)

Índice

1. Instruções de utilização..... 3
1.1. Menus.....3
1.1.1. Menu Principal.....3
1.1.2. Menu de instruções.....4
1.1.3 Menu de conectividade online.....5
1.1.4 Menu de opções.....6
1.2 Durante o jogo.....7
2. Estado do projeto.....8
2.1. Estado das funcionalidades..... 8
2.2 Dispositivos utilizados..... 9
2.2.1. Timer..... 9
2.2.2 Teclado..... 9
2.2.3 Rato.....10
2.2.4 Placa gráfica.....10
2.2.5 RTC..... 10
3. Estrutura do Código..... 12
3.1 Geral.....12
3.2 Viewers.....12
3.3 Model.....12
3.4 Physics..... 14
3.4 Controllers..... 15
3.5 Gráfico de function calls..... 15
4. Detalhes da implementação.....16
5. Conclusões.....17
5.1 Problemas..... 17
5.2 Like to haves..... 17
5.3 Lições aprendidas..... 18

1. Instruções de utilização

1.1. Menus

Seguem as instruções de utilização para os menus do programa "**Snook8**".

1.1.1. Menu Principal

Após a inicialização do programa, o menu principal é apresentado ao utilizador. Este contém 4 botões que podem ser clicados com o botão esquerdo do rato.

- O botão "**PLAY**" redireciona o utilizador para o menu de conectividade online.
- O botão "**INFO**" redireciona o utilizador para o menu de instruções.
- O botão "**OPTIONS**" redireciona o utilizador para o menu de opções.
- O botão "**EXIT**" termina o programa.

1.1.2. Menu de instruções

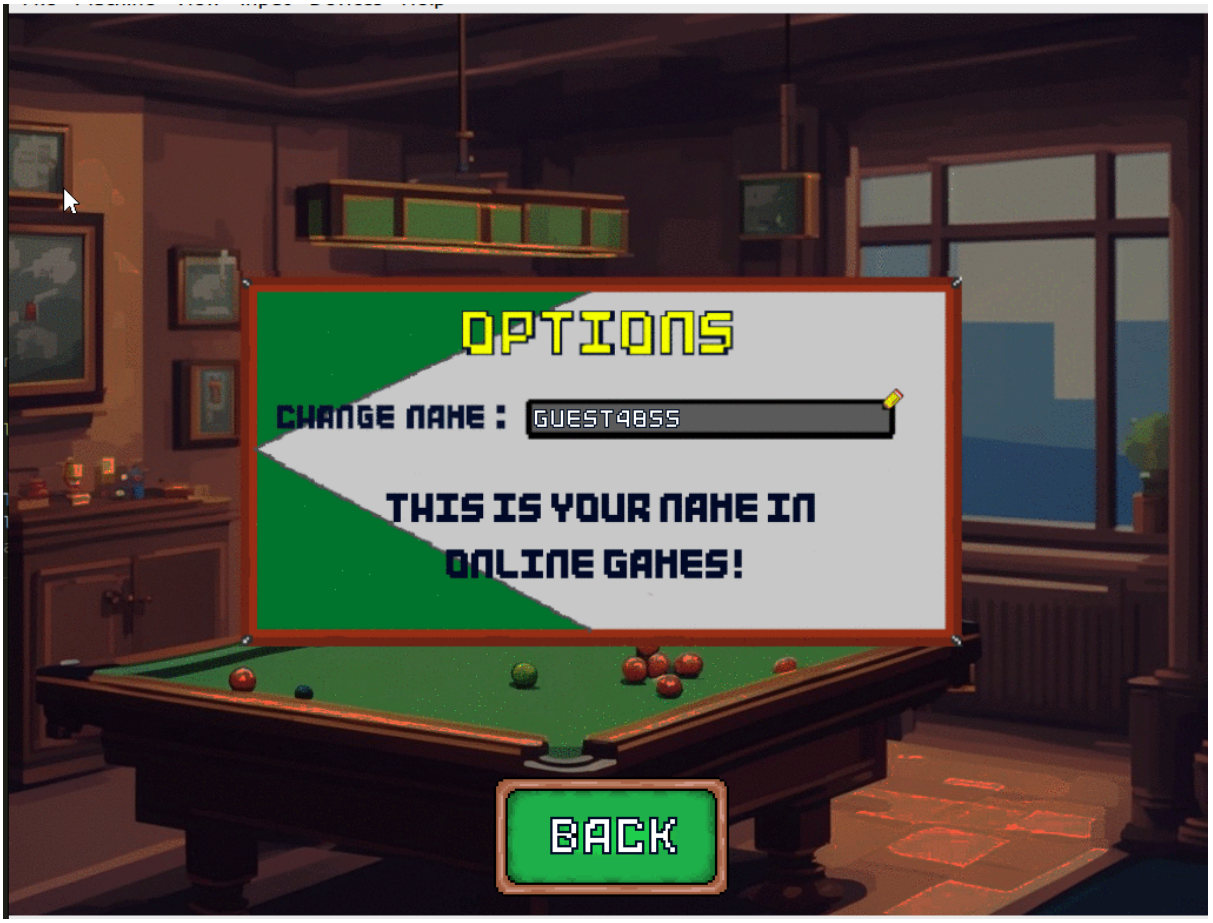
Ao entrar neste menu, é apresentado ao utilizador um menu que ensina ao utilizador como jogar. Ao clicar no botão “**BACK**”, o utilizador volta ao menu principal.

1.1.3 Menu de conectividade online

Após entrar neste menu, o utilizador pode ver três botões:

- O botão “**LOCAL**” inicia um jogo local, onde os jogadores jogam no mesmo computador, alternando turnos.
- O botão “**ONLINE**” inicia um jogo online utilizando a serial port.
- O botão “**BACK**” redireciona o utilizador para o menu principal.

1.1.4 Menu de opções



Após entrar neste menu, o utilizador tem a opção de mudar o seu nome utilizando o teclado. O programa guarda todas as mudanças feitas no nome automaticamente e o utilizador pode sair com o botão “**BACK**”.

1.2 Durante o jogo

O objetivo do jogo é vencer o oponente numa partida de bilhar. Todas as regras deste jogo são baseadas nas regras de bilhar.

Ao iniciar um jogo, é apresentado ao utilizador o ecrã do jogo. No centro do ecrã encontra-se a mesa onde o jogo de bilhar é jogado. Na parte superior, o utilizador consegue ver quantas bolas faltam a cada jogador para poder colocar a bola preta e acabar o jogo, bem como um indicador da rotação a ser aplicada na bola para a próxima jogada. O botão no canto superior direito abre um menu que permite ao utilizador sair do jogo.

Quando é o turno do utilizador, este pode fazer uma jogada ao utilizar o rato para escolher o ângulo da jogada e, ao manter o botão esquerdo do rato clicado, pode arrastar o taco para trás, escolhendo assim a força da tacada. Caso necessário, o utilizador pode escolher a rotação da bola ao clicar no indicador de rotação.

Quando uma partida acaba, é apresentado ao utilizador um ecrã que indica o vencedor da partida com um botão “**BACK**” que redireciona o utilizador para o menu principal.

2. Estado do projeto

2.1. Estado das funcionalidades

Foram implementadas todas as funcionalidades referidas no manual de utilização com exceção de:

- Mudar a rotação da bola utilizando as setas do teclado
- Jogar um jogo multiplayer (o jogo faz a sincronização entre VMs mas a implementação do multiplayer em si está a falhar)

As funcionalidades implementadas que não foram referidas no manual de utilização são:

- **Double buffering através de page flipping:** A utilização da função 0x07 permite utilização de double buffering sem problemas de flickering.
- **Escrita de texto:** O texto escrito para o nome dos utilizadores é criado dinamicamente com a combinação de um alfabeto de XPMs.
- **Cálculo de físicas contínuas:** As físicas do jogo são calculadas de forma semi-contínua permitindo mais precisão nos cálculos das posições das bolas.

2.2 Dispositivos utilizados

Dispositivo	Utilização	Interrupções
Timer	Controlar a frame rate do jogo	SIM
Teclado	Escrever o nome do jogador	SIM
Rato	Interagir com menus, controlar o taco no jogo	SIM
Placa gráfica	Mostrar os menus e jogo no ecrã	NÃO
RTC	Controlar o tempo de cada ronda e tempo total de jogo	SIM
Serial Port	Multiplayer entre duas VMs	SIM

2.2.1. Timer

A implementação do timer é semelhante à do lab 2. A única mudança feita a esta driver é na variável *elapsed*, que passou a ser estática, sendo necessário a utilização de um getter para a obter.
No programa, o timer é utilizado de várias formas diferentes:

- A cada duas interrupções do timer (que está configurado para gerar 60 interrupções por segundo), é desenhado um novo frame no buffer secundário e é feita a mudança de página.
- O menu principal utiliza a variável *elapsed* para controlar a altura do logo, que varia consoante o tempo utilizando uma função sinusoidal.
- Após o cálculo de um evento, o estado do jogo é atualizado em todos os frames até o próximo evento.

2.2.2 Teclado

A implementação do teclado é parecida com a do lab 3.

No programa, o teclado é utilizado de várias formas diferentes:

- O utilizador pode pressionar na tecla esc para terminar o programa.
- No menu de opções, o utilizador pode usar o teclado para escrever o seu nome.

2.2.3 Rato

A implementação do rato tem poucas mudanças relativamente à versão do lab 4.
No programa, o rato é utilizado de várias formas diferentes:

- Nos menus, o rato é utilizado para clicar em botões e navegar pelos menus.
- No jogo, o rato controla o taco e a força da tacada.

2.2.4 Placa gráfica

A implementação da placa gráfica apresenta algumas mudanças relativamente à versão do lab 5. O ficheiro graphics.h apresenta agora, para além das funcionalidades já implementadas:

- **Double buffering através de page flipping:** Foi adicionada a variável *frame_buffer_secondary* e a função *swap_buffers()* que invoca a função 0x07 do VBE e muda o canto superior do ecrã para ser o início do buffer secundário.
- **Escrita de texto:** Foi adicionada a função *drawText()* que aceita uma string e uma fonte (implementada como um array de imagens XPM). Esta função é extremamente limitada (só desenha números e letras minúsculas) e é apenas utilizável para a funcionalidade deste programa.

No programa, a placa gráfica é utilizada para apresentar os menu e jogo ao utilizador. É utilizado o modo 0x118 com resolução 1024x768 e 16.8 milhões de cores (apesar do

2.2.5 RTC

No nosso projeto, utilizamos o RTC de forma a controlar o tempo de cada ronda e o tempo total gasto no jogo de uma forma mais precisa.
Este é um dispositivo que apesar de não ter sido abordado nas aulas práticas, através dos documentos disponibilizados sobre o mesmo, revelou-se simples de compreender e implementar.
A nível de implementação, conta com ativação de interrupções, e ativação do modo alarme, no modo segundo a segundo, para que o tempo seja perfeitamente controlado. Também estão ativas as interrupções periódicas e uma função de modo a saber a hora no momento que seria para mostrar no caso de uma resolução maior.

2.2.6 Serial Port

No projeto, tentamos implementar a Serial Port de forma a promover o modo multiplayer, através da comunicação entre as VMs. Revelou-se complicado estabelecer uma boa ligação, como tal ficou implementado a conexão e entrada no jogo, no entanto o modo multiplayer em si não é totalmente funcional.

Nesta implementação são utilizadas duas interrupções de entre as da Serial Port, no caso Received Data e Transmitter Holding Register Empty. Além disso, esta é configurada com uma bitrate de 115200, 8 bytes por caractere e paridade ímpar.
Para estabelecer a conexão inicial entre as duas VMs, sempre que um jogador entra no modo multiplayer manda um byte, de forma a perceber se algum outro jogador já está à espera, caso se verifique há uma troca de bytes entre os jogadores de forma a sincronizar e iniciar o jogo em ambas as VMs.

3. Estrutura do Código

3.1 Geral

- **main.c** - O módulo main invoca initGame() que inicializa o jogo.
- **Game.c** - Este módulo cria os resources para o jogo e é onde se localiza o driver receive loop para este programa. *(peso relativo de 4%)*
- **resources.c** - Resources é um struct que contém toda a informação necessária para o funcionamento do programa. *(peso relativo de 3%)*

3.2 Viewers

- **menuViewer.c** - O menuViewer contém todas as funções necessárias para desenhar o menu, incluindo o fundo, os botões, o rato, o logo, o nome do jogador. drawEllipses() e drawMenuLogo() desenham imagens dinamicamente, utilizando o tempo do temporizador como variável para alterar a posição e quantidade de imagens a ser mostradas. *(peso relativo de 2%)*
- **lineViewer.c** - O lineViewer contém as funções necessárias para desenhar linhas, que são utilizadas no jogo para a mira. O código da internet mas perdemos a fonte. *(peso relativo de 1%)*
- **cueViewer.c** - O cueViewer contém a função drawCue() que faz os cálculos de raytracing necessários para a mira e desenha o taco e a mira. *(peso relativo de 2%)*

3.3 Model

- **table.c -** O módulo table representa a mesa de bilhar contendo a informação relevante à gameplay do jogo e funções que utilizam os atributos desta. table.c contém o enum GAME_STATE que representa o estado da mesa:
  Aiming - Quando o jogador está a mirar o taco.
  Shooting - Quando o jogador larga o taco.
  Waiting - Quando jogo espera por ação de outro módulo
  Simulating - Quando a jogada está a ser apresentada ao utilizador.
  Advantage - Quando o jogador pode mudar a posição da bola branca utilizando o rato.
  Em geral o struct Table contém informação útil para o funcionamento do jogo, como as bolas, as paredes, os jogadores, as constantes físicas bem como as imagens que são necessárias para desenhar o ecrã de jogo. *(peso relativo de 20%)*
- **pocket.c** - O struct Pocket representa um buraco na mesa, tendo uma posição e um raio. *(peso relativo de 1%)*

- **player.c** - No módulo player.c, está definido um enum chamado playerBall que representa as bolas do jogador:
  PLAYERSOLID - bolas lisas.
  PLAYERSTRIPED - bolas riscadas.
  PLAYERBALLNONE - cada jogador é inicializado com o tipo de bola PLAYERBALLNONE que é atualizado quando um jogador coloca uma bola no buraco. *(peso relativo de 1%)*
  O struct Player representa um jogador, tendo um nome, tipo de bola e se este está a jogar. player.c contém uma função para desenhar o nome de um jogador no ecrã.
- **mouseModel.c** - mouseModel contém o struct Mouse que representa o rato no programa, tem os atributos pos, savedPos e delta. *(peso relativo de 1%)*

- **menu.c** - O módulo menu contém o código necessário para criar e alterar o estado interno dos menus no jogo. Este ficheiro contém o enum MenuType que representa o menu a ser mostrado no momento (Ex. Main Menu, Options Menu, etc.). O struct menu contém os botões, o rato, a opção que está a ser selecionada e as imagens necessárias para desenhar os menus. *(peso relativo de 10%)*
- **event.c** - No sistema de física continua, um evento representa uma ação que muda o movimento da bola. O módulo contém o enum EVENT_TYPE que representa os diferentes eventos que podem ocorrer:
  BALL_BALL - Colisão entre bolas.
  BALL_LINEAR_CUSHION - Colisão entre uma bola e uma parede reta.
  BALL_CIRCULAR_CUSHION - Colisão entre entre uma bola e uma parede curva.
  BALL_POCKET - Colisão entre uma bola e um buraco.
  STICK_BALL - Colisão entre o taco e a bola.
  SPINNING_STATIONARY - Transição entre o movimento da bola a girar para a bola parada.
  ROLLING_SPINNING - Transição entre o movimento da bola a rolar para a bola a girar.
  SLIDING_ROLLING - Transição entre o movimento da bola a deslizar para a bola a rolar.
  O struct Event contém o tempo do evento, o tipo de evento e os elementos que se estão a interagir (bolas, paredes e buracos). *(peso relativo de 5%)*
- **cushion.c** - Uma cushion é uma parte das paredes que cercam o campo de jogo. O módulo contém dois structs LinearCushion, as paredes retas e CircularCushion, as paredes curvas. Estes structs contém a informação necessária para representar estas paredes, dois pontos e um vetor normal (a direção para qual a parede aponta) para as retas e um ponto e um raio para as curvas. *(peso relativo de 2%)*
- **cue.c** - O módulo cue contém o struct Cue que tem as variáveis necessárias para representar o taco e fazer cálculos sobre as suas colisões com a bola branca. *(peso relativo de 1%)*
- **button.c** - button representa um botão que pode ser clicado com o rato durante o funcionamento do programa. Button contém duas imagens, uma quando o botão está *idle* e outra quando o rato encontra-se sobre o botão e uma posição. Para a detecção do rato também é necessário a largura e comprimento do botão, guardado no vetor size. *(peso relativo de 2%)*

- **ball.c** - ball representa uma bola na mesa de bilhar. A bola tem estado, BALL_STATE que indica que movimento a bola apresenta. A bola também tem um tipo (branca, preta, lisa e riscada) bem como uma posição, uma velocidade, uma velocidade angular, um raio, uma imagem e um evento. *(peso relativo de 2%)*

3.4 Physics

- **utilities.c** - este módulo tem diversos métodos úteis para o cálculo das diferentes transformações matemáticas servindo como suporte a todos os outros módulos da física. Aqui é criada uma struct QuarticCoeff, responsável por guardar os coeficientes de uma determinada equação. O boleano *valid* permite uma pequena otimização, evitando o cálculo da solução da equação se os valores forem inválidos *(peso relativo de 6%)*

- **simulate.c** - o módulo simulate é aquele responsável pela simulação das físicas, sendo este que controla o início e fim da mesma. É a partir desta que o controller inicia o processo de evolução. *(peso relativo de 8%)*

- **quartic.c** - quartic consiste numa função resolvente de uma equação de quarto grau. É neste módulo que definimos as equações necessárias para a resolução das mesmas. Código de Daniel Klostermann. (https://web.archive.org/web/20210727073153/http://www.iowahills.com/P51RootFinder.html) *(peso relativo de 4%)*

- **evolve.c** - este módulo consiste em funções responsáveis por evoluir o estado dos diferentes elementos por um determinado intervalo de tempo. Com base em fórmulas físicas, a velocidade, posição e velocidade angular de cada bola são alteradas ao longo do tempo. *(peso relativo de 7%)*

- **events.c -** é neste módulo que a verificação da existência de novos eventos é realizada. São considerados todos os possíveis e depois determinado aquele que acontecerá primeiro.*(peso relativo de 7%)*

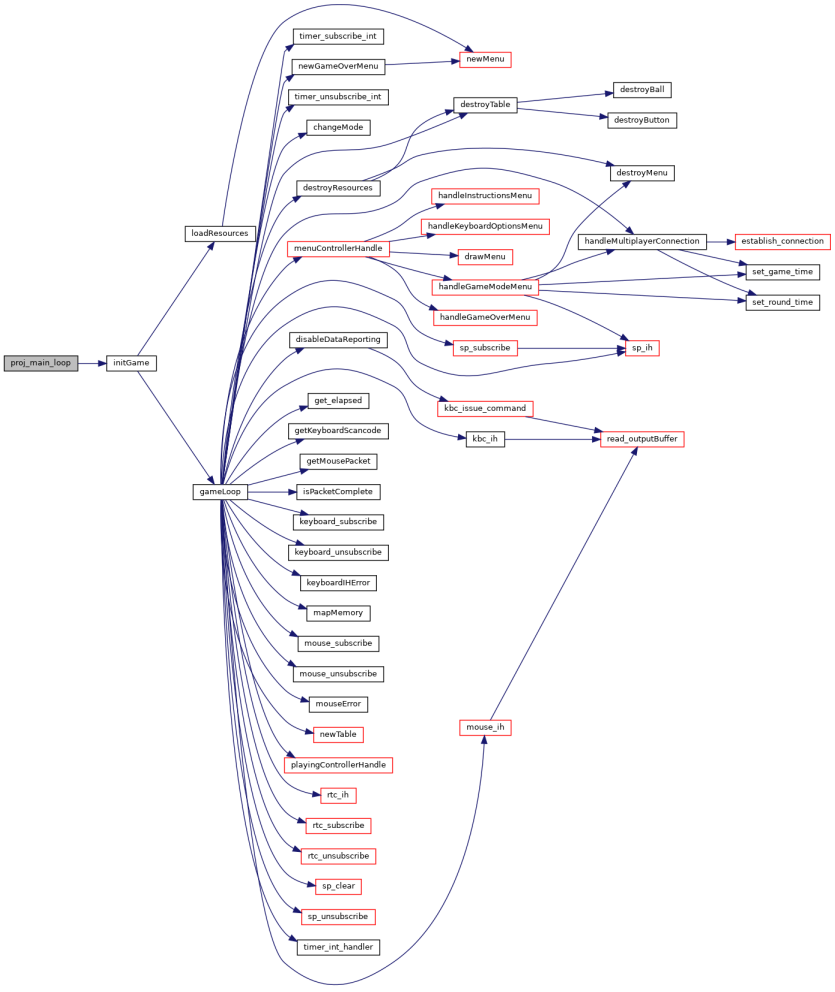
- **resolve.c -** aqui é realizado o processamento dos eventos. Uma vez detetados, é neste módulo que o resultado dos mesmos se reflete em mudanças no estado das diferentes bolas. *(peso relativo de 7%)*

3.4 Controllers

- **menuController.c** - A função presente neste módulo permite interpretar os diferentes interrupts gerados pelas drivers, alterando o estado do menu.
- **playingController.c** - A função presente neste módulo permite interpretar os diferentes interrupts gerados pelas drivers, alterando o estado da table.
- multiplayerController.c - A função presente neste módulo permite interpretar os diferentes interrupts gerados pelas drivers, alterando o estado da table durante o modo multiplayer.

3.5 Gráfico de function calls

A única main function neste programa é a função *gameloop()* que chama os diferentes controllers baseado no estado do programa.



4. Detalhes da implementação

Neste programa, a máquina de estados é extremamente simples visto que a maior parte dos objetos tem o seu próprio estado. No header states.h, tem definido os estados do programa:

- **Playing** - A interpretação dos interrupts é feita pelo playingController.
- **Menu** - A interpretação dos interrupts é feita pelo menuController.
- **Over** - O programa é terminado.

Sendo que a máquina de estados é tão simples, as transições são poucas. O jogo começa em menu e quando é escolhida uma opção no menu de conectividade online, o estado muda para Playing. Quando um jogo é terminado ou o jogador sai da partida, o estado volta para menu. Quando o utilizador clica em EXIT no menu principal ou clica em ESC no teclado, o estado muda para Over e o programa termina.

A simulação física utiliza um método contínuo para os cálculos, que se mostra mais eficiente e preciso. Em vez de realizar todos os cálculos de uma só vez, optamos por evoluir o sistema em partes, distribuindo os cálculos ao longo do tempo para aumentar a eficiência do programa. Assim, sempre que um novo evento é identificado, o sistema evolui até alcançá-lo. Nesse momento, o evento é resolvido e o próximo evento é detectado. Este processo é repetido até que não seja possível encontrar um novo evento, indicando que o sistema está em repouso. O sistema realizado baseou-se na implementação desenvolvida e documentada aqui.

5. Conclusões

5.1 Problemas

O maior problema que tivemos neste projeto foi certamente a implementação da função que resolve uma equação quártica, que é necessária para o cálculo das física contínua. Como os valores que introduzimos nesta função apresentam um grande intervalo, está sujeita a graves erros de aritmética de valores flutuantes. Para resolver a equação quártica, precisamos de resolver uma equação cúbica. No entanto, a nossa implementação da função resolvente da equação cúbica baseada neste artigo: William Kahan, "To solve a real cubic equation." PAM-352, Center for Pure and Applied Mathematics, University of California, Berkely. November 10, 1986. (https://people.eecs.berkeley.edu/~wkahan/Math128/Cubic.pdf) era imprecisa. A solução da internet que encontramos é robusta o suficiente para a maioria dos cálculos feitos no nosso programa, apenas em raras ocasiões é possível observar certas imprecisões. No entanto, a pesquisa por esta função abrandou consideravelmente o desenvolvimento do nosso projeto.

5.2 Like to haves

- Opções de customização dos visuais do jogo
- Opções de resolução

5.3 Lições aprendidas

Ao longo do projeto, aprendemos bastante sobre o desenvolvimento de projetos em C e sobre modelos de física continua. Semelhantemente, aprendemos sobre as graves consequências de erros de precisão em funções críticas.