U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Pool Game 

Report, LCOM 2023/24

Authored by
Beatriz Sonnemberg up202206098
Gonçalo Marques up202206205
Nuno Rios up202206272
Tomás Teixeira up202208041

Index

Game Instructions 4
Main menu 4
About screen 5
Local game 6
Project Status 7
Timer 7
Keyboard 8
Mouse 8
Video Card 9
Real Time Counter 9
Code Organization/Structure 10
Mouse Module - 5% 10
Keyboard Module - 5% 10
Real Time Counter Module - 5% 10
Video Card Module - 5% 10
Timer Module - 5% 11
Utils Module - 2% 11
Sprite Manager Module- 2% 11
Module sprite - 3% 11
View Module - 15% 11
Controller Module- 15% 12
Physics Module - 20% 12
Pool Module- 10% 12
Vector2 module- 3% 12
Function call graph 13
Implementation Details 14
Real Time Counter 14
Physics: 14
Video Graphics: 15
View: 15
Conclusion 16

Concept

Pool game recreates the classic real-life 8 Ball pool experience. As the name suggests, each player aims to pocket 8 balls in total—either the solid-colored balls or the striped balls—always finishing by pocketing the 8 ball last.

This game was developed in C for the Minix operating system.

Game Instructions

Main menu

Upon launching the game, the home screen offers two selections: 'Local Game' and 'About'. Players can navigate through these options using the UP and DOWN arrow keys and press ENTER to select the desired choice. The ESC key can be used to exit the game at any time.

About screen

This screen contains information about the game, including the names of the authors. Users can return to the main menu at any time by pressing the 'B' key.

Local game

In this mode, the user and a friend can engage in a fully functional snooker game controlled via the computer mouse. To execute a shot, the player positions the cursor over the white ball and clicks the left mouse button. Without releasing the button, the player then drags the mouse to angle the cue in the desired direction. By dragging the mouse forward or backward, the player adjusts the power of the shot. Each player is allotted 20 seconds to complete their move. The ball next to the player's name indicates whose turn it is.

Project Status

Device	What for	Interruptions
Timer	Limit the player turn	Yes
Keyboard	Menu navigation	Yes
Mouse	Cue stick control	Yes
Video Card	Application menus and screens display	No
RTC	Show actual date	No

Timer

The timer is widely used in the project. The main features are the physics and frame updates at a 60Hz rate, and the counter used in the game to limit the player turn time.

Relevant Functions:

The timer is used in the init_controller() in controller.c and its utils are in io/timer. There when there is a timer interrupt the on_timer_tick() in controller.c is called and then, depending on the game state it updates the view (physics and/or frames) and potentially the “Too long to hit” game event (when a turn takes longer than 20 seconds).

Keyboard

The keyboard is used for game controls, i.e. to enable the user to choose the menu they want to go to and quit the game.

Relevant Functions:

The keyboard is used in the init_controller() in controller.c and its utils are in io/keyboard and io/kbc. There when there is a keyboard interrupt the on_kbd_event() in controller.c is called and then, depending on the game state (if it is in the main menu, in the about menu or in the actual game) it handles that event accordingly. Enabling users to navigate through the menus and quit the game.

Mouse

The mouse is used for the cue stick controls. It enables the user to aim to the pocket and manage the force they put into the cue ball.

Relevant Functions:

The mouse is used in the init_controller() in controller.c and its utils are in io/mouse and io/kbc. There when there is a mouse interrupt the on_mouse_event() in controller.c is called and then, depending on the game state(if we are waiting for a hit) it enables the view of the cue stick and according to the mouse position it puts more or less force into the shot.

Video Card

The video card is used to draw the user interface to the screen. For optimization purposes, we use video mode 0x113, which uses indexed color. We also take advantage of double buffering as well as local buffers to avoid visual artifacts, namely when rendering motion physics.

We developed a physics engine capable of handling moving balls with friction in 2d, as well as a robust collision detection system, which leverages the laws of physics and math to handle circle-circle and circle-wall collisions.

Real Time Counter

The real time counter is used to display the current time for the users.

Relevant Functions:

The real time counter is used in the render_game_view() to display the current time in view.c and its utils are in io/rtc.

Code Organization/Structure

Mouse Module - 5%

The mouse module is the same as the one implemented the labs. We then handled the bytes accordingly in the controller.

Keyboard Module - 5%

The keyboard module is the same as the one implemented the labs. We then handled the bytes accordingly in the controller.

Real Time Counter Module - 5%

The real time counter (RTC) module provides a basic interface to get the current date and time. No interrupts were needed.

Video Card Module - 5%

The video card module is pretty much like the one implemented the labs. However, there is a small tweak. We implemented a function that enables us to draw a sprite rotated by a certain angle. This was particularly useful to handle the rotation of the cue stick and draw it properly.

Timer Module - 5%

The timer module is the same as the one implemented the labs. We then handled its interrupts in the controller to update the physics and the frames.

Utils Module - 2%

The utils module is the same as the one implemented the labs.

Sprite Manager Module- 2%

The sprite manager only loads the sprites and frees the memory at the end. Consequently, we only load the sprites once.

Module sprite - 3%

The sprite saves the sprite xpm, its width and height. A sprite is created in sprite manager and also destroyed there.

View Module - 15%

The view module handles all the UI logic. It has methods to render the various menus, according to the game state and other variables.

Controller Module- 15%

The controller module handles all the interrupts. When it receives an interrupt, it calls the proper handler, which in turn, (potentially) calls the event handler to act accordingly.

Physics Module - 20%

The physics module handles all the collisions and movement of the game objects.

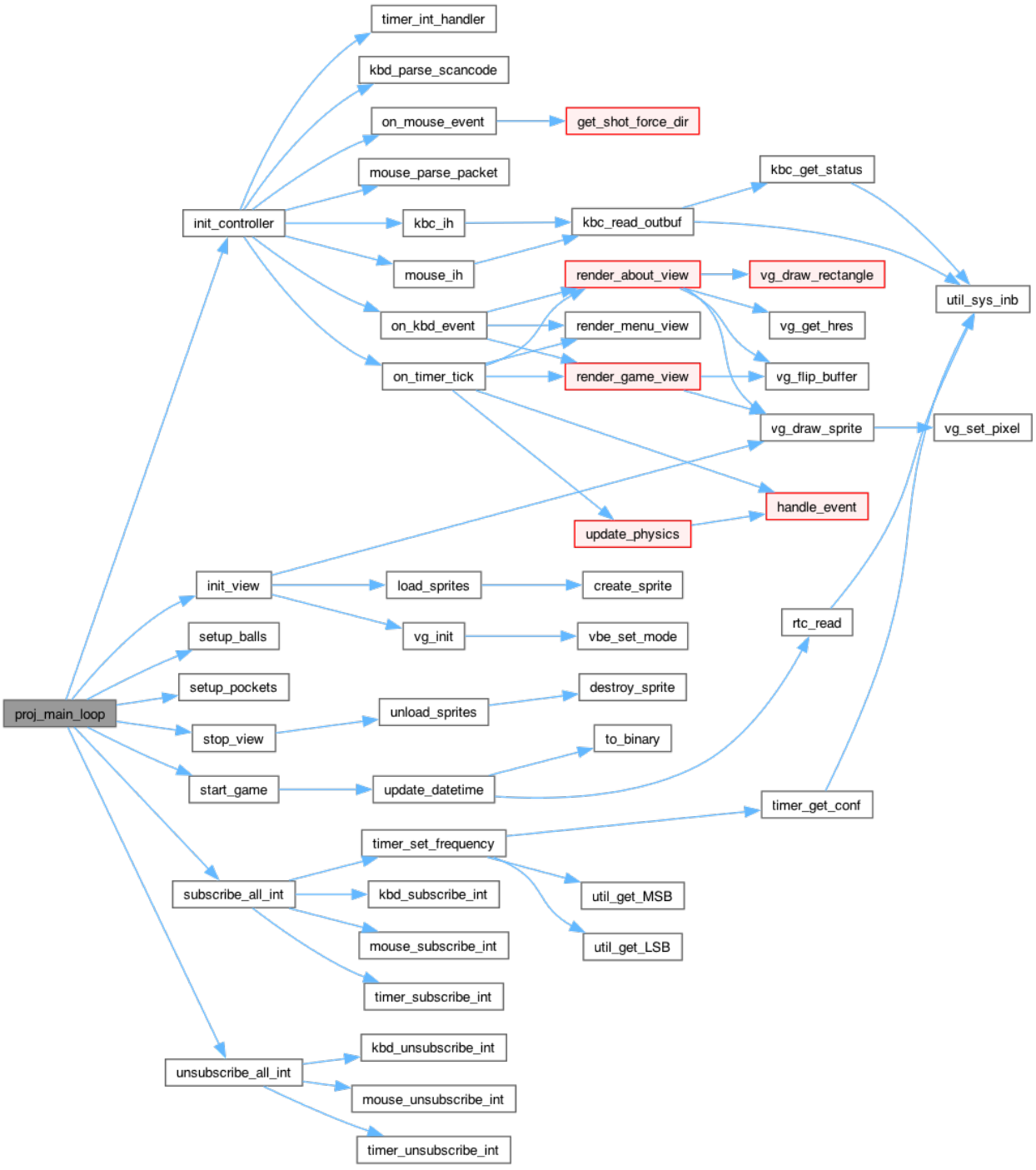
Pool Module- 10%

The pool module handles all the logic related to the game such as updating the objects every tick.

Vector2 module- 3%

The vector2 module is a utils module which we use in physics to represent the 2D movement of the objects.

Function call graph



Implementation Details

Real Time Counter

The RTC was implemented using the resources made available in the Moodle and the reference sheets. It was not particularly challenging as we only used it to display the current time. A future use of RTC would be in the communication protocol, to make sure the bytes were sent with a non-significant delay.

Physics:

- Collisions:

Collision detection played a crucial role in our project, given that a game of snooker inherently involves frequent collisions between the balls, and between the balls and both the walls and the cue. Ensuring accurate and realistic simulation of these interactions was essential for maintaining the authenticity of the game. It was quite difficult to make it realistic and accurate and we had a lot of work searching for possible approaches due to not having squared objects.

Video Graphics:

- Rotated Sprite:

One of the biggest challenges in video graphics was the method to rotate a sprite by a certain angle. It involved some math, and we had a bad time debugging it.

View:

- Double Buffer and Flipping:

Double buffering and flipping played a big role in making our game more fluid. It was quite challenging because we were not getting why it was taking so long while using the VBE flip function. When we used memcpy it improved the time the rendering was taking.

Conclusion

Throughout the development of the project, there were some delays, particularly in the physics. Consequently, we were not able to implement the multiplayer with the serial port. Additionally, the MINIX poor performance in MacOS computers impacted the way we tested our code, and consequently delayed everything else. The main achievements of our project are the physics which greatly simulates collisions between non-square objects and the event-driven approach in which our project was develop onto.