Snake: The Game

Final Report
LC 2021/22

T06G04

Diogo Babo (up202004950@up.pt)

João Oliveira (up202004407@up.pt)

João Teixeira (up202005437@up.pt)

Rui Andrade (up202007539@up.pt)

Index

1 - Introduction

We decided to recreate the classic “Snake” game but with a few tweaks to include the mouse and other functionalities. The player’s goal is to grow the snake to the largest size possible without dying. The snake grows when it eats the fruits that appear during the game. In order to give some dynamic to the game, the player can destroy the blocks (2 clicks in order to fully be destroyed) with the left-click, and can also spawn fruits with the right click. Therefore, the fruits can spawn inside blocks but the user can see them, so in order to reach them he needs to destroy the blocks.

2 - User Instructions

2.1 - How to play

To play the game, the controls are:

- **WASD**: to move up, left, down or right, respectively (the arrow keys can also be used to move the snake);
- **Mouse left-click**: to destroy the blocks every 1 second (must click twice to fully destroy a block);
- **Mouse right-click**: to spawn a fruit in the game;
- **ESC**: to pause the game.

2.2 - Main Menu

When the user runs the program, the main menu appears on the screen.

Figure 1: Main Menu

The user can then use the mouse to select one of the available options:

- **Play**: starts the game;
- **Leaderboard**: shows the top 5 scores;
- **Exit**: exits the game.

2.3 - Play

Upon clicking the **Play** option in the main menu, the game starts. The goal is to destroy all the blocks in the map, using the mouse left click, and grow the snake to the biggest size possible by eating the fruits.

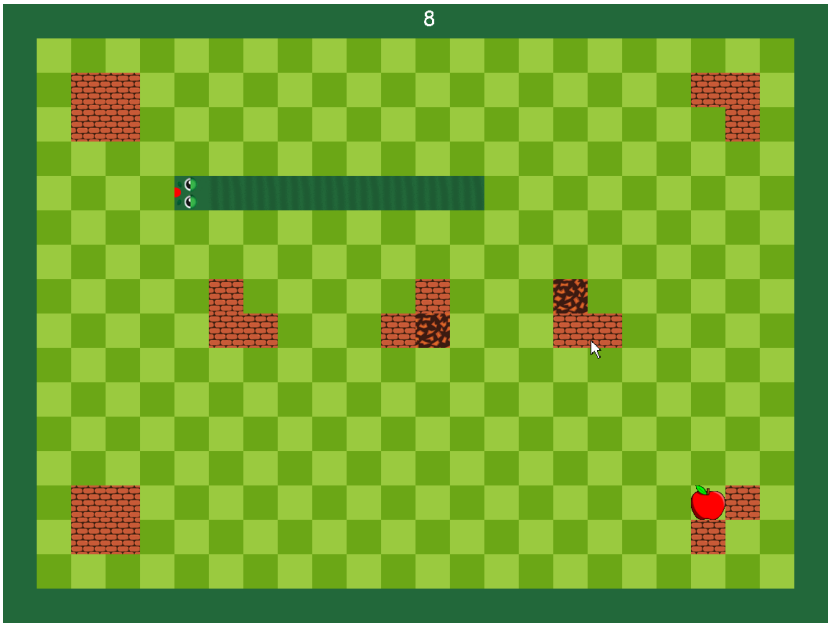

Figure 2: The Game

At any time, the user can click the **ESC** key to pause the game and show the pause menu, which allows him to resume the game or exit it. Unlike the Main Menu, the user can use the **Arrow Keys** to navigate the pause menu.

Figure 3: Pause Menu

2.4 - Leaderboard

Upon clicking the **Leaderboard** option in the main menu, the user is presented with a list of the top 5 highest scores of the game.

At any time, the user can click the **ESC** key to return to the main menu.

Figure 4: Leaderboard

3 - Project Status

All functionalities planned were fully implemented, apart from the **Multiplayer mode** (Serial Port**)**.

Device	Usage	Interrupt (I) / Polling (P)
Timer	Frame rate, cooldown to destroy an obstacle and to spawn fruit.	I
Keyboard	Snake movement and selecting options in the pause menu.	I
Mouse	Selecting options in the menus and destroying the blocks in-game and spawn fruits.	I
Video Card	Menus, screen display and in-game drawings.	N/A

3.1 - Timer

The timer is used to control the frame-rate, in that it allows 60 interrupts per second. In the **Game State** it is responsible not only for drawing all the objects and the animations but also for checking the collisions with the game objects such as the blocks and the borders, as well as updating the movement. This is done in the **InterruptHandlerTimer()** function in **“snake.c”**.

In the **Menu State** it is used to highlight the selected option when the mouse cursor is over it. This is done in the **MenuTimerHandler()** function in **“menu.c”**.

In the **Pause State** it is used to highlight the selected option when the user changes the option selected with the keyboard. This is done in the **PauseTimerHandler()** function in **“pause.c”**.

3.2 - Keyboard

The keyboard is used to control the snake in the **Game State** and in the **Pause Menu** to choose between the options. We created a global variable **KEY** that has an **ENUM** for the most important keyboard keys, and we update it every time the user presses a key. We receive the scancode and we use the **updateKBC()** to update the **KEY,** which is passed as an argument in the functions that control the keyboard interrupts for each **State**.

3.3 - Mouse

The mouse is used to destroy blocks and spawn fruits while the snake is in the **Game State**. We created a global variable called **mouseInfo** that is updated every time the player uses the mouse. We receive the packets and use the **mouse_set_packet()** to update the struct, which then tells us where the mouse is and if any button is pressed. We pass this information to the respective mouse handler functions that work differently for each **State**. We check the mouse position, and also add functionalities for the **Left** & **Right Click**. (**StateMouseHandler()).**

3.4 - Video Card

The video card is used to draw the menus and the other elements in the game, such as the snake, fruits and obstacles, in all **States**. We used **double buffering** to draw the different images of the game. The mode used was the **0x14C**, with a resolution of **1152x864**, direct color mode and 4 294 967 296 different colors. We receive the xpm's and draw them using the function **draw_pix_map()**. In order to optimize the game and avoid drawing the background every other interrupt we draw the specific part of the background that was changed using the function **cleanBG().** We do this in most **States,** for example in the Menu instead of drawing another Menu XPM with one of the options selected, filling all the screen, we just clear the options that are not selected and draw a smaller XPM with the option highlighted. We created a custom **font**, which consists of a unique XPM with all the alphabet and the digits, equally spaced so it's easier to manipulate each character **initLetters()**. We could also have a XPM for each character but in order to optimize the program as much as possible, we opted for this implementation. (This XPM was used to draw the user score). We also check collisions for the snake with every object in **CheckColisions()**, and handle the snake movement and its tail in **moveSnake().**

4 - Code Organization/Structure

4.1 - game.c

Responsible for the game main loop. Handles the different Game States and calls different functions depending on it. Inside the **gameLoop()** function, the function **driver_receive()** is called.

4.2 - leaderboard.c

Displays the leaderboard, saves it to a file and updates it according to the player score.

4.3 - menu.c

Displays the menu and highlights different options depending on which one is selected.

4.4 - pause.c

Displays the pause menu. Functions similarly to the main menu.

4.5 - proj.c

Subscribes to the interrupts of all the devices used before the game main loop and unsubscribes after the game ends.

4.6 - snake.c

Responsible for all the game mechanics available.

4.7 - graphics.c

Has all the functions that are responsible for handling the video mode and drawing on screen.

4.8 - keyboard.c

Responsible for checking for errors on the **KBC** and subscribing and unsubscribing keyboard interrupts.

4.9 - mouse.c

Responsible for checking for errors on the **KBC** and subscribing and unsubscribing mouse interrupts.

4.10 - timer.c

Sets the timer frequency and subscribes and unsubscribes timer interrupts.

4.11 - utils.c

Has functions that get the most and least significant byte from a 2 byte unsigned int and creates a **sys_inb()** that works with 1 byte values.

DB - Diogo Babo's Contribution
JO - João Oliveira's Contribution
JT - João Teixeira's Contribution
RA - Rui Andrade's Contribution

C File	Weight	DB	JO	JT	RA
game.c	20%	20%	30%	20%	30%
leaderboard.c	5%	20%	20%	30%	30%
menu.c	9%	30%	30%	20%	20%
pause.c	6%	25%	25%	25%	25%
proj.c	3%	25%	25%	25%	25%
snake.c	20%	30%	20%	30%	20%
graphics.c	11%	25%	25%	25%	25%
keyboard.c	4%	25%	25%	25%	25%
mouse.c	14%	25%	25%	25%	25%
timer.c	5%	25%	25%	25%	25%
utils.c	3%	25%	25%	25%	25%

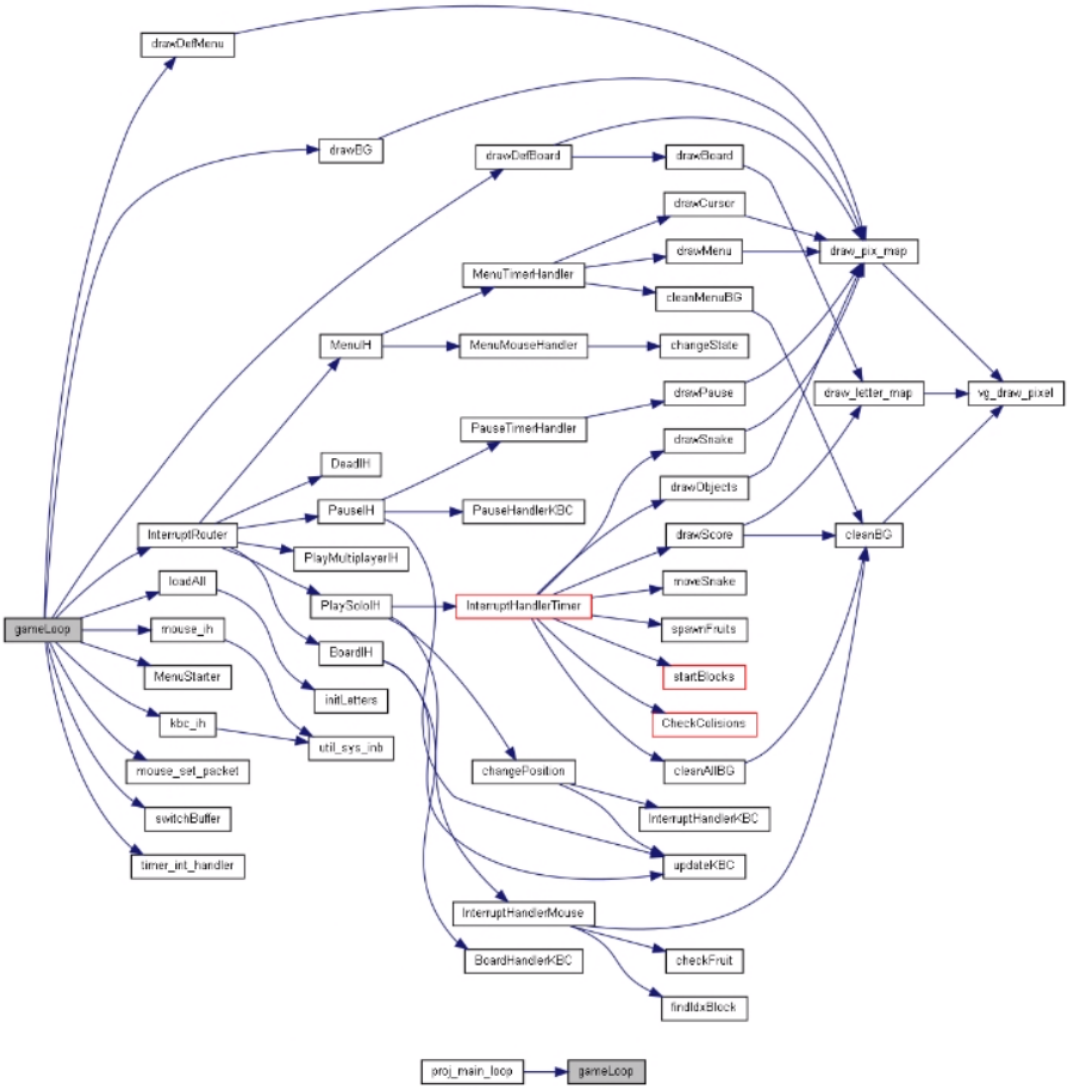

Figure 5: Function call graph

5 - Implementation Details

Our game is based on a **State Machine**, making it easier to implement different interrupt handlers for the different **States**.

In our Snake Game, the snake can only move in “blocks”, in specific 48x48 pixel blocks. So we verify **collisions** by checking if 2 or more object coordinates are in the same block.

In order to add some dynamic, in our two different **Menus** (**Pause** and the **Initial One**) we handle the different options differently. While on the **Start Menu**, we check if the **mouse** is over any of the buttons highlighting it if that’s the case, on the **Pause State** we use the **keyboard** to choose one of the available options (highlighting it when selected).

Last but not least, to optimize the program we decided to avoid drawing the same background every time, for example in **Play_Solo State** we just draw the game background once, and for every timer interrupts we clear the parts that changed, for example the snake movement. This also happens in the **Menu State**, we draw the “default” background, and then only clear and draw the buttons that are/aren’t selected.

6 - Conclusion

In spite of the “bumps in the road” that we as a group faced in the early stages of the development such as lack of expertise in low level languages like **C** we managed to create a good project and overcome those problems. We believe in our project to be a good replica of the snake game with some additional twists that give a unique sense to the game.

We also had some fun in the development of the game and every group member contributed equally to it. It was a great challenge thrown at us and we did our best to make every detail look authentic and special.