MINIGOTCHI

Computer Laboratory 2023/2024
Class 10 - Group 5

Authors:
António Abílio (up202205469@fe.up.pt)
João Castro (up202206575@fe.up.pt)
Vanessa Queirós (up202207919@fe.up.pt)

Table of Contents

1. Introduction..... 3
2. User Instructions.....3
2.1. Main Menu.....3
2.2. “New Game” Screen.....4
2.3. Main Room.....5
2.4. Minigame Selection Screen.....6
2.5. Minigame: Tic Tac Toe.....7
2.6. Minigame: Rock, Paper, Scissors.....9
3. Project Status..... 10
3.1. Used I/O Devices..... 10
3.2. Timer..... 10
3.3. Keyboard..... 11
3.4. Mouse.....11
3.5. Video Card.....11
3.6. RTC..... 12
3.7. Serial Port.....12
4. Code Organization/Structure..... 13
4.1. Overall Code Structure.....13
4.2. Model.....13
4.2.1. minigotchi.c..... 13
4.2.2. bar.c.....13
4.2.3. button.c.....14
4.2.4. cursor.c.....14
4.2.5. hotbar.c.....14
4.2.6. item.c.....14
4.2.7. mainMenu.c.....14
4.2.8. mainRoom.c..... 14
4.2.9. minigameMenu.c..... 15
4.2.10. nameMinigotchi.c..... 15
4.2.11. tictactoe.c.....15
4.2.12. rockPaperScissors.c.....15
4.3. View.....15
4.3.1. guiDrawer.c..... 15
4.3.2. mainMenuViewer.c..... 15
4.3.3. mainRoomViewer.c.....16
4.3.4. minigameMenuViewer.c..... 16
4.3.5. nameMinigotchiViewer.c.....16
4.3.6. tictactoeViewer.c.....16
4.3.7. rockPaperScissorsViewer.c.....16
4.4. Controller.....16
4.4.1. mainMenuController.c..... 16

4.4.2. mainRoomController.c....17
4.4.3. minigameMenuController.c....17
4.4.4. nameMinigotchiController.c....17
4.4.5. tictactoeController.c....17
4.4.6. rockPaperScissorsController.c....17
4.5. Other Modules....17
4.5.1. proj.c....18
4.5.2. database.c....18
4.5.3. timer.c....18
4.5.4. kbc.c....18
4.5.5. keyboard.c....18
4.5.6. mouse.c....19
4.5.7. video.c....19
4.5.8. rtc.c....19
4.5.9. uart.c....19
4.5.10. vector.c....19
4.5.11. queue.c....20
4.5.12. utils.c....20
4.6. Code taken from the Internet....20
4.6.1 Vector Implementation in C....20
4.6.1.1 init_vector....20
4.6.1.2 resize_vector....20
4.6.1.3 push_back....20
4.6.1.4 remove_at....20
4.6.1.5 free_vector....21
4.6.2 Changes made....21
4.6.3 URL....21
4.6.4 Queue Implementation in C....21
4.6.4.1 createNode....21
4.6.4.2 createQueue....21
4.6.4.3 isEmpty....21
4.6.4.4 getSize....21
4.6.4.5 enqueue....22
4.6.4.6 dequeue....22
4.6.4.7 peek....22
4.6.5 Changes Made....22
4.6.6 URL....22
5. Implementation Details....23
5.1 Topics covered in the lectures....23
5.2 Topics not covered in the lectures....23
6. Conclusions....24
Appendix....25
How to run Minigotchi....25
How to reset the game....25

1. Introduction

Minigotchi is a virtual pet simulation game where players take care of a digital creature. Players start by naming their Minigotchi. The game revolves around fulfilling the Minigotchi's basic needs like feeding, playing and petting. By successfully managing these activities, players ensure their Minigotchi remains healthy and happy, creating an immersive and rewarding experience.

2. User Instructions

2.1. Main Menu

Figure 2.1: Main Menu

When starting the program, the main menu appears, where the user can select between two options: start a new game (or continue the game, in case a save file exists) and quit the game.

2.2. “New Game” Screen

Figure 2.2: Enter Name Screen

This screen appears if the user is playing for the first time and, therefore, doesn’t have any save file yet. This is where the program prompts the user to enter a name for their Minigotchi.

2.3. Main Room

Figure 2.3: Main Room

The main room is where the player can interact with his Minigotchi by feeding him, playing with him or petting him. The more the player takes care of his Minigotchi, the happier it will be.
The two bars in the upper left corner are the hunger (orange) and happiness (yellow) bars, which update according to the minigotchi's hunger and happiness.
Below the two bars, there are two buttons: minigames and quit. The minigames button leads the player to a new screen presenting the available minigames.
The hotbar can be toggled on and off by pressing "E" on the keyboard. The player can select items in the hotbar by pressing the arrow keys (left or right) and finally, use the item pressing Enter.
There is also a window that is sunny during the day and dark at night and even shows the sunset. The minigotchi's name is displayed in the upper right corner.

2.4. Minigame Selection Screen

Figure 2.4: Minigame Selection Screen

The Minigame Selection Screen is a very simple screen where the user can select between two minigames: Tic Tac Toe (which uses serial port) and Rock Paper Scissors (which is a singleplayer game). These are two ways to win items for the minigotchi.

2.5. Minigame: Tic Tac Toe

Figure 2.5: Tic Tac Toe Selection Screen

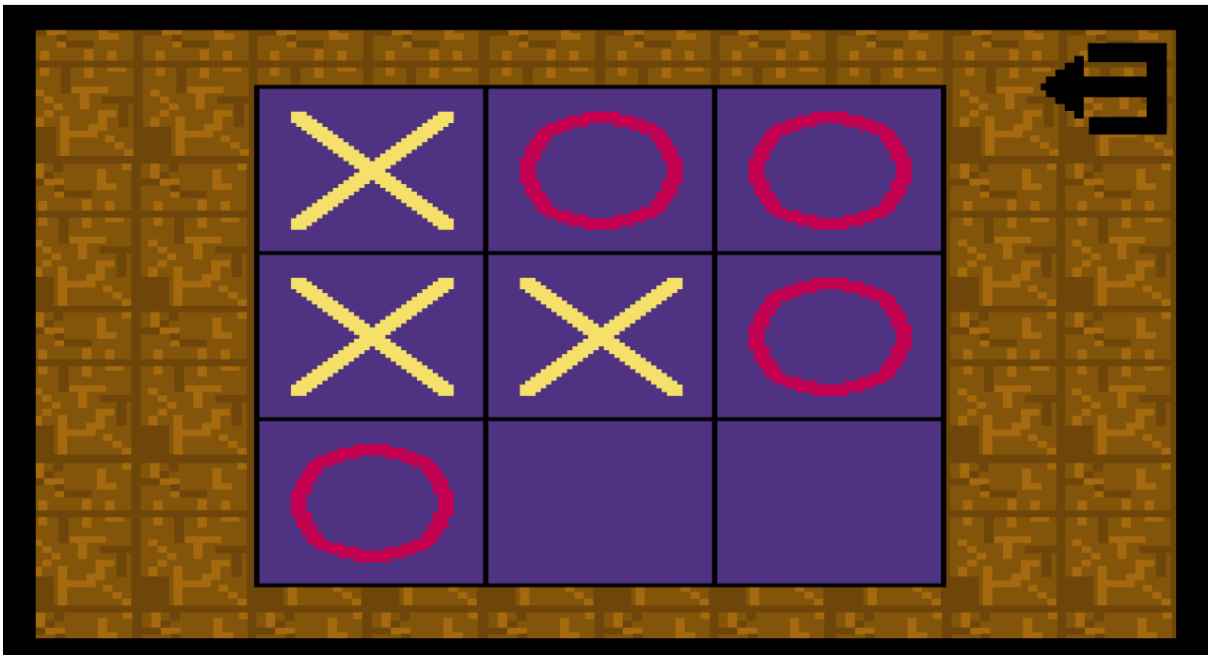

Figure 2.6: Minigame: Tic Tac Toe

Figure 2.7: Tic Tac Toe Winning Screen

This minigame is implemented like the very known Tic Tac Toe game, where two players compete against each other. The game is played on a 3x3 grid, and each player takes turns marking the spaces in the grid with their symbol, either X or O. The objective is to be the first to align three of their marks in a row, which can be achieved horizontally, vertically or diagonally. If a player successfully aligns three marks, they win the game and are rewarded with one food item, which can then be used to feed their Minigotchi. If the game ends in a draw, neither player receives a reward.

The implementation of this game relies heavily on the use of the serial port / UART interface. Upon boot up of the game one can choose to be the host or the guest. The order of which should be: Guest enters the game and waits for the host. If done in the wrong way the game will not crash but due to synchronization issues the game may become unplayable and force the user to restart the minigame.

2.6. Minigame: Rock, Paper, Scissors

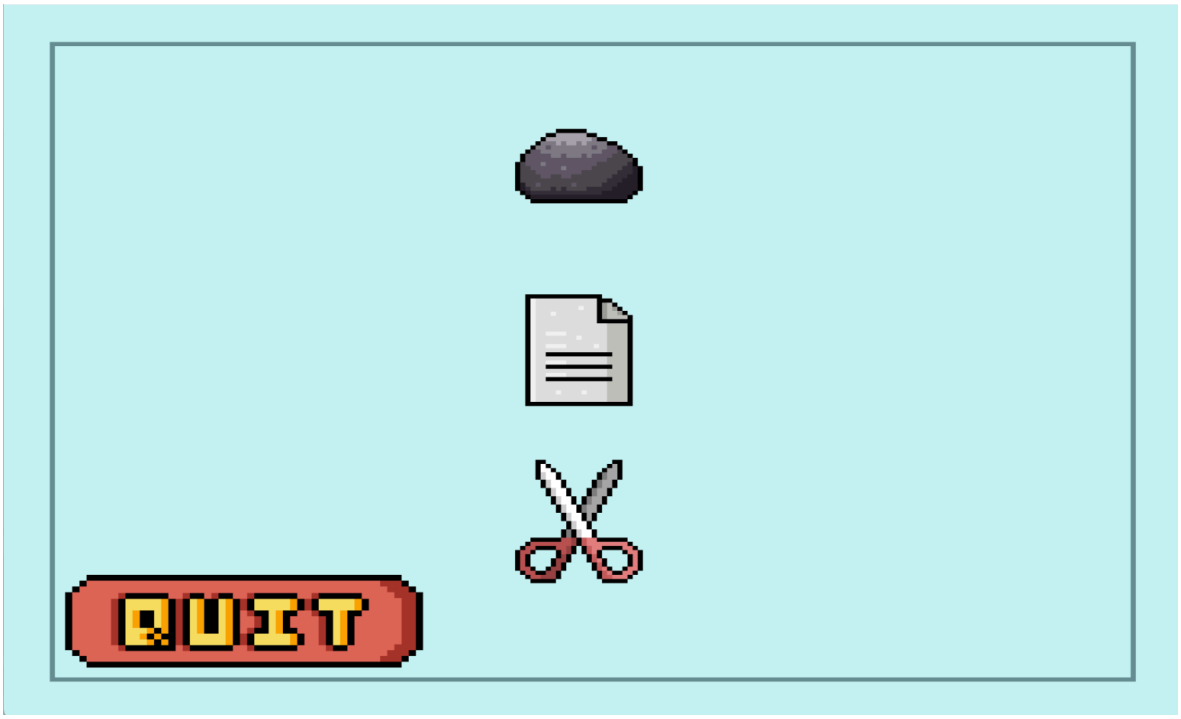

Figure 2.8: Minigame: Rock, Paper, Scissors

Figure 2.9: Minigame: Rock, Paper, Scissors Winning Screen

This minigame is implemented like the classic rock, paper, scissors game, where the player plays against a bot, by choosing one of the three options (rock, paper or scissors). If the bot chooses an item that beats the one chosen by the player, the player is not rewarded. However, if the player wins, he gets one food item as a reward, which he can then feed to his Minigotchi.

3. Project Status

3.1. Used I/O Devices

Device	What for	Interrupts
Timer	- Controlling frame rate - Timing health and happiness bar updates	Y
Keyboard	- Writing Minigotchi's name - Controlling hotbar	Y
Mouse	- Petting Minigotchi - Cursor - Clicking buttons	Y
Video Card	- Overall screen and menu display	N
RTC	- Switching between day, evening and night in main room window	Y
Serial Port	- Establish communication between the two players. - Play the Tic-Tac-Toe Minigame.	Y

3.2. Timer

The timer is used to control the frame rate by setting a fixed rate on which it updates the game loop. Our project currently runs at 60 frames per second by executing the actual game loop when this condition

```
if (((getTimerCounter() - (sys_hz() / FPS)) == 0))
```

is met.
The low level functions that handle the timer functionality are implemented in the device_controllers folder more specifically the timer.c and utils.c files. The interrupts are handled in the proj.c main loop.

3.3. Keyboard

The keyboard can be utilized for various purposes such as:

- Game Control: the keyboard is used to open the inventory, select food items, and give food to the Minigotchi. These controls facilitate smooth interaction within the game, allowing players to efficiently manage and care for their pet.

The functions that are used for the keyboard's game control are multiple but the main ones reside in the game state switch case.

- Text Input: the keyboard is also used for naming the Minigotchi. This allows players to personalize their virtual pet by entering a name.

For text input

The low level functions that handle the keyboard functionality are implemented in the device_controllers folder more specifically the keyboard.c, kbc.c and utils.c files. The interrupts are handled in the proj.c main loop.

3.4. Mouse

In our project, the mouse is used for the following purposes:

- Position: the mouse position is tracked to navigate through the game interface and menus. It is used to hover buttons, icons, and other interactive elements, providing a visual cue for the selection.
- Buttons: the mouse buttons are used for various interactions within the game. Players click to select options in menus along the game, select if you want to host/guest a game,... We can also use the mouse to left click on the Minigotchi, allowing us to pet it.

The low level functions that handle the mouse functionality are implemented in the device_controllers folder more specifically the mouse.c, kbc.c and utils.c files. The interrupts are handled in the proj.c main loop.

3.5. Video Card

In our project, the graphics card plays a crucial role in ensuring a seamless and visually appealing user experience by performing the following functions:

- Overall Screen and Menu Display: the graphics card is responsible for rendering all visual elements on the screen, including the game's main interface and various menus. This involves setting the video mode, …
- Display moving elements, such as the cursor, which also has collision detection with the buttons.
- Display static elements, such as all the GUI elements, the items or the backgrounds.
- Display our custom font

We use video mode 0x14C, which has a resolution of 1152x864 (bits per pixel: 32 ((8:)8:8:8)) and uses direct color. Furthermore, we use triple buffering with page flipping, which helps in reducing flickering and tearing by ensuring smooth transitions between the game frames. While the first buffer updates the screen, the second buffer is used to build the next frame. The third buffer is used to load static elements, and with this we avoid any loss of performance.
With that, we also used sprites to create several visual elements of our game.
The low level functions that handle the video card functionality are implemented in the device_controllers folder more specifically the video.c file. This device controller is set up in the proj.c main loop.

3.6. RTC

We use the Real-Time Clock (RTC) in our project for the following purpose:

- Switching Between Day, Evening, and Night: the RTC is used to read the current time and switch between different times of the day in the main room window, by creating an alarm at three specific times of the day. This feature enhances the game's realism, allowing players to experience different times of the day.

The low level functions that handle the real-time clock functionality of the game are implemented in the device_controllers folder more specifically the rtc.c and utils.c files.
The interrupts are handled in the proj.c main loop.

3.7. Serial Port

The UART Interface uses interrupts along with a queue (FIFO) that is used as a first in first out data structure. When the TX sends a byte over the serial port, the RX side will have a new interrupt from the serial port prompting it to go ahead and process it. The byte that is received is placed inside the queue and the interrupt is cleared.

We decided to use COM1 and 115200 Hz as the frequency of the serial communication.
All low level functions that handle the functionality of the UART interface are present in the device_controllers folder, more specifically the uart.c and queue.c. The interrupts are handled in the proj.c main loop.

4. Code Organization/Structure

4.1. Overall Code Structure

We decided to use a previously studied architecture pattern called Model-View-Controller (studied in LDTS), which essentially separates the code into three different components. As an example, code that is responsible for displaying game elements on the screen is in the view section, code that sends the data to be displayed by the viewer and that controls the game through user input, timed events, etc., is in the controller section, and lastly, the data structures that represent the elements to be displayed and controlled during the game are in the model section. We generated the following graph using doxygen (can be seen with more detail in the doxygen documentation):

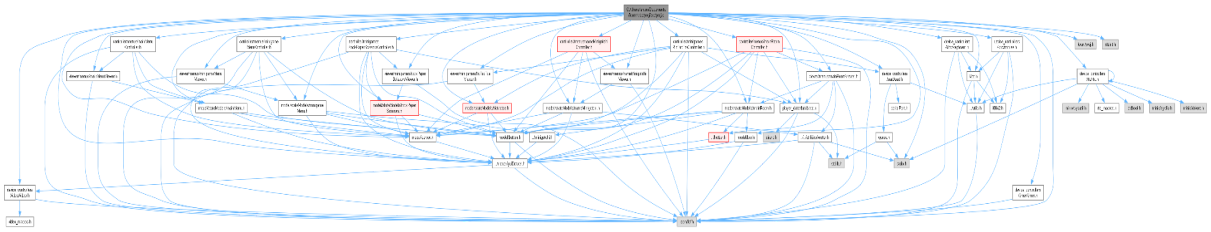

Figure 4.1: Overall project graph

Our project only has one main function, which is **proj_main_loop**. This function subscribes the device controllers to interrupts, calls the corresponding interrupt handlers and sets up the graphics mode. Furthermore, this function has a state machine that manages the current game's state (window to be displayed). In the end, the function unsubscribes all interrupts and sends the necessary commands to the controllers in order to terminate the program correctly.

4.2. Model

4.2.1. minigotchi.c

Contains the structure representing the Minigotchi, including its health, happiness, name and some other variables which are verified and updated during the game loop.

Relative weight in project: 2.7%

4.2.2. bar.c

Contains the structure representing the bars, such as the health and happiness bars, which holds a level that can be specified. This level can then be read and updated during the game loop.

Relative weight in project: 2.7%

4.2.3. button.c

Contains the structure representing the buttons, which can be of several times, ranging from the start button to the rock, paper and scissors options in one of the minigames. Buttons can be activated and deactivated during the game loop.

Relative weight in project: 2.7%

4.2.4. cursor.c

Contains the structure representing the cursor, which can be used in all screens. This structure has a value that can be used to check if the player is currently clicking the mouse, which can be accessed during the game loop.

Relative weight in project: 2.7%

4.2.5. hotbar.c

Contains the structure representing the hotbar, which is one of the main room elements. It stores and manages every item it contains during the game loop. It also has a value that tells which item is currently selected, so that the game can check if the player can use the item in the highlighted slot.

Relative weight in project: 2.7%

4.2.6. item.c

Contains the structure representing an item (in this case food), which holds a feed level that indicates how much it satiates the Minigotchi’s hunger.

Relative weight in project: 2.7%

4.2.7. mainMenu.c

Contains the structure representing the main menu, with all its buttons and the cursor. This structure can be used to construct the main menu as a template and behaves like a singleton.

Relative weight in project: 2.7%

4.2.8. mainRoom.c

Contains the structure representing the main room, with all its buttons, bars, its hotbar, Minigotchi and the cursor. This structure can be used to construct the main room as a template and behaves like a singleton.

Relative weight in project: 2.7%

4.2.9. minigameMenu.c

Contains the structure representing the minigame menu, with all its buttons and the cursor. This structure can be used to construct the minigame menu as a template and behaves like a singleton.

Relative weight in project: 2.7%

4.2.10. nameMinigotchi.c

Contains the structure representing the screen where the player can name his Minigotchi, with all its buttons and the cursor. This structure can be used to construct the naming screen as a template and behaves like a singleton.

Relative weight in project: 2.7%

4.2.11. tictactoe.c

Contains the structure of the tic tac toe game and the functions that execute operations on it (getters and setters). This structure can be used to construct the tic tac toe game as a template and behaves like a singleton.

Relative weight in project: *2.7%*

4.2.12. rockPaperScissors.c

Contains the structure of the rock paper scissors game and the functions that execute operations on it (getters and setters). This structure can be used to construct the rock paper scissors game as a template and behaves like a singleton.

Relative weight in project: *2.7%*

4.3. View

4.3.1. guiDrawer.c

This module sets up all the sprites so that they can be drawn during the game loop. It also manages everything related to sprites and images to draw.

Relative weight in project: 2.7%

4.3.2. mainMenuViewer.c

This module is responsible for drawing all the sprites that belong to the main menu, such as buttons and the background.

Relative weight in project: 2.7%

4.3.3. mainRoomViewer.c

This module is responsible for drawing all the sprites that belong to the main room, such as buttons, bars, the Minigotchi, the hotbar and the background.

Relative weight in project: 2.7%

4.3.4. minigameMenuViewer.c

This module is responsible for drawing all the sprites that belong to the minigame selection menu, such as buttons and the background.

Relative weight in project: 2.7%

4.3.5. nameMinigotchiViewer.c

This module is responsible for drawing all the sprites that belong to the minigame selection menu, such as buttons and the background.

Relative weight in project: 2.7%

4.3.6. tictactoeViewer.c

This module is responsible for drawing all the sprites that belong to the Tic Tac Toe minigame, such as buttons, the board and the background.

Relative weight in project: 2.7%

4.3.7. rockPaperScissorsViewer.c

This module is responsible for drawing all the sprites that belong to the Rock Paper Scissors minigame, such as buttons, the board and the background.

Relative weight in project: 2.7%

4.4. Controller

4.4.1. mainMenuController.c

This module is responsible for controlling all the actions, updates and checks that have to be done every frame, in the main menu. In this case, the only action it has to check is whether the player is clicking any buttons.

Relative weight in project: 2.7%

4.4.2. mainRoomController.c

This module is responsible for controlling all the actions, updates and checks that have to be done every frame, in the main room. In this case, the controller is responsible for checking whether the player is petting or feeding the Minigotchi, clicking any buttons, or toggling the hotbar.

Relative weight in project: 2.7%

4.4.3. minigameMenuController.c

This module is responsible for controlling all the actions, updates and checks that have to be done every frame, in the minigame selection menu. In this case, the only action it has to check is whether the player is clicking any buttons.

Relative weight in project: 2.7%

4.4.4. nameMinigotchiController.c

This module is responsible for controlling all the actions, updates and checks that have to be done every frame, in the Minigotchi naming window. In this case, the only action it has to check is whether the player is typing or clicking any buttons.

Relative weight in project: 2.7%

4.4.5. tictactoeController.c

This module is responsible for controlling all the actions, updates and checks that have to be done every frame, in the Tic Tac Toe minigame. In this case, the controller is responsible for managing the player turns and checking if the player is clicking any buttons.

Relative weight in project: 2.7%

4.4.6. rockPaperScissorsController.c

This module is responsible for controlling all the actions, updates and checks that have to be done every frame, in the Rock Paper Scissors minigame. In this case, the controller is responsible for managing the player turns and checking if the player is clicking any buttons.

Relative weight in project: 2.7%

4.5. Other Modules

There are some modules that we considered unsuitable for the MVC pattern, so we decided to created separate folders for them.

4.5.1. proj.c

This module contains the main loop (which is also the game loop). It is responsible for subscribing all the necessary device controller interrupts and unsubscribing them, and handling the interrupts. Furthermore, the game loop behaves like a state machine to manage all the possible game windows.

Relative weight in project: 2.7%

4.5.2. database.c

This module is responsible for managing the player's data in the game, including saving, loading and handling various aspects of the game state. The database functionality ensures that the player's progress and game state are preserved across sessions.

Relative weight in project: 2.7%

4.5.3. timer.c

This module is responsible for configuring and handling the system timer, which is crucial for managing time-based events and ensuring the game's smooth operation. The function within this module allows for setting the timer frequency, subscribing to timer interrupts, and managing timer-related configurations.

Relative weight in project: 2.7%

4.5.4. kbc.c

This module is responsible for managing interactions with the Keyboard Controller (KBC), which is important for handling keyboard interrupts and ensuring smooth communication between the keyboard and the system. The functions within this module allow for enabling keyboard interrupts, reading the KBC status, reading from the output buffer, and writing to the KBC ports.

Relative weight in project: 2.7%

4.5.5. keyboard.c

This module is responsible for managing keyboard inputs by handling keyboard interrupts. It provides functions to subscribe and unsubscribe from keyboard interrupts, process interrupt requests, and retrieve the latest scan from the keyboard.

Relative weight in project: 2.7%

4.5.6. mouse.c

This module is responsible for managing mouse inputs by handling mouse interrupts. It provides functions to subscribe and unsubscribe from mouse interrupts, read status and data from the KBC (Keyboard Controller), issue commands to the KBC, and handle mouse-specific interrupt requests.

Relative weight in project: 2.7%

4.5.7. video.c

This module is responsible for managing video output in the game, including setting up video modes, handling frame buffers for double and triple buffering, drawing pixels and pixmaps, and performing page flips. The functions within this module ensure smooth and efficient rendering of graphics.

Relative weight in project: 2.7%

4.5.8. rtc.c

This module is responsible for managing the Real-Time Clock (RTC) and handling RTC interrupts. It provides functions to subscribe and unsubscribe from RTC interrupts, read and write RTC registers, convert between BCD and binary, and handle different types of RTC interrupts (Alarm, Update, Periodic).

Relative weight in project: 2.7%

4.5.9. uart.c

This module is responsible for managing the Universal Asynchronous Receiver-Transmitter (UART) for serial communication. It provides functions to initialize the UART, handle interrupts, read and send bytes, and manage the receive queue.

Relative weight in project: 2.7%

4.5.10. vector.c

This module is responsible for managing dynamic arrays (vectors) in the game. It provides functions to initialize a vector, resize it, add elements, remove elements, and free the allocated memory. The dynamic vector allows for efficient management of collections of data that can grow and shrink in size.

Relative weight in project: 2.7%

4.5.11. queue.c

This module is responsible for managing an implementation of a FIFO that is used in the serial port. The implementation of the queue uses Nodes (Linked List) to actually reproduce the effect of a FIFO. The file also includes functions to initialize the queue, nodes, add more nodes, remove nodes from the queue and other functions that are usually present in a data structure of a higher programming language.

Relative weight in project: 2.7%

4.5.12. utils.c

This module is responsible for only 3 functions. Two of them perform operations to extract the MSByte and LSByte of an uint16_t. The other function is a wrapper function, it allows us to use the sys_inb system call with uint8_t instead of having to use an uint32_t.

Relative weight in project: 2.7%

4.6. Code taken from the Internet

4.6.1 Vector Implementation in C

4.6.1.1 init_vector

This function initializes a new vector with a specified initial capacity. It uses malloc to allocate memory for the vector structure and its data array. If the memory allocation fails at any point, the function will handle the error by printing a message and exiting the project.

4.6.1.2 resize_vector

This function allows the vector to be resized to accommodate new elements. It reallocates memory for the data array and adds more space for the new items that are going to be added. If memory reallocation fails, it handles the error by printing a message, freeing the vector, and exiting the project.

4.6.1.3 push_back

This function adds a new element to the end of the vector. If the vector is already at its capacity, it calls resize_vector to increase the capacity by one. It then appends the new element to the data array and increments the size of the vector.

4.6.1.4 remove_at

The objective of this function is to allow an item that is specified at a given index to be removed then, it shifts the elements after the specified index to the left to fill the gap. If the

vector size is reduced to half of its capacity, it calls resize_vector to shrink the capacity. If the index is out of bounds, it returns an error message.

4.6.1.5 free_vector

This function frees the memory allocated for the vector that was previously created.

4.6.2 Changes made

We changed the implementation that we found on the internet to accommodate the fact that the vector would now hold Sprites instead of integers.

4.6.3 URL

We have taken inspiration from this implementation:

- https://codereview.stackexchange.com/questions/269299/vector-implementation-in-c

4.6.4 Queue Implementation in C

4.6.4.1 createNode

This function creates a new node for the queue. It uses malloc to allocate memory for the node, assigns the provided data to the node, and sets the next pointer to NULL. If memory allocation fails, it returns NULL.

4.6.4.2 createQueue

This function initializes a new queue. It allocates memory for the queue structure, sets both the front and rear pointers to NULL, and initializes the size to 0. If memory allocation fails, it returns NULL.

4.6.4.3 isEmpty

This function checks if the queue is empty by comparing the size of the queue to 0. It returns 1 if the queue is empty and 0 otherwise.

4.6.4.4 getSize

This function returns the size of the queue, which is the number of elements currently in the queue.

4.6.4.5 enqueue

This function adds a new element to the rear of the queue. It creates a new node using createNode and, if the queue is empty, sets both the front and rear pointers to this new node. Otherwise, it links the new node to the current rear and updates the rear pointer. The size of the queue is incremented.

4.6.4.6 dequeue

This function removes an element from the front of the queue. If the queue is empty, it returns a null character. Otherwise, it retrieves the data from the front node, updates the front pointer to the next node, and frees the old front node. If the front becomes NULL, it also sets the rear to NULL. The size of the queue is decremented, and the removed data is returned.

4.6.4.7 peek

This function returns the data at the front of the queue without removing it. If the queue is empty, it returns a null character.

4.6.5 Changes Made

We adjusted the implementation to better handle specific error conditions and to incorporate the changes we needed to make so that it would work with the serial port.

4.6.6 URL

We have taken inspiration from this implementation:

- https://www.geeksforgeeks.org/queue-linked-list-implementation/

5. Implementation Details

Throughout this project, we focused on dividing our code into three different components: model, view and controller. There are, however, some other modules that we decided to separate from this implementation by creating other directories for them, making them more maintainable. For example, the device controllers created during the semester or the database.

5.1 Topics covered in the lectures

We managed to implement all the device controllers, including the RTC and the UART.
For the RTC we ended up using alarms for controlling the main room window, which changes between day, sunset or night, depending on the time (day starting at 8 AM, sunrise starting at 6 PM and night starting at 9 PM). This is done via interrupts so we only change the window state upon an RTC interrupt.
For the UART we decided to make one multiplayer minigame, where two players can play against each other in a tic tac toe game. However, this game has to be played in a specific order for it to work. First, the guest should enter the game and then the host. Sometimes there is also an extra step to be considered, for when the package loss is high (for example, when the game is played through the internet): the game should be played slowly for the serial port to process the bytes correctly.

5.2 Topics not covered in the lectures

Some of the most relevant topics that we ended up implementing were also not covered in the lectures and we had to either learn or search our old courses. For example, in the class we did not talk about a way to save files using C. This was not challenging but it would be nice if this was explained in the lectures.
One more thing we consider rather important is the way we structured our code. Although not perfect we ended up using MVC (Model View Controller) but this was not explained in the lectures nor was it explained how we should structure the code itself.

6. Conclusions

Now that everything is done we found some things that we would like to address.

The problems:

Although not related to the code / implementation of features we would like to give a paragraph to reference some issues that we have encountered. Originally we had planned to do everything in two weeks but due to other projects, tests, exams we did most of this work in just 1 week. Besides that, our team is only constituted of three elements, which made things harder, due to the close deadline.

Plans for the future:

Due to the limited time that we had, during the actual project we thought about many new ideas that could have been added, for example, more minigames and a way to trade food or items with other players using the serial port. These may end up being added in the future as we are planning to keep exploring the possibilities of the Minix's LCOM image and its code base.

Main achievements:

We are really impressed with the amount of content we have managed to add to the game in just one week while still adding all the features we talked about in the Theoretical Classes, even the ones that were considered not so relevant (RTC and UART). Not only did we build a fully fledged Tamagotchi but we also added minigames and multiplayer support. We also think that the design, while not being important, shows that we have managed to make the game pretty but still efficient.

Lessons learned:

Finally, analyzing all this we have concluded that we need to learn more about time management.

Appendix

How to run Minigotchi

To run Minigotchi do the following:

- Make sure you're inside the folder that contains the Makefile.
- Run the following commands:
- make
- lcom_run proj

Note that there is a possibility of compilation failing with the error "unexpected num of chars xpm_load", the cause of which is still unknown to us. Just try to recompile it. You should also consider playing with the Virtual Box in scaled mode, because it works best.

How to reset the game

Please, before you reset the game, close it. If this is not done, it can lead to confusion on why the save file still exists. Deleting the file while the game is running and pressing quit will result in the Minigotchi process writing the current save data to the save file thus creating one if deleted.

- Make sure you're inside the folder that contains the Makefile.
- Run the following commands:
- su (input your password for superuser after)
- ./delete_save_file.sh
- exit

If for some reason this did not work you should do this instead:

- su (input your password for superuser after)
- cd /usr/
- rm ./save.txt
- exit