# Simon Says

## Relatório Final


**Mestrado Integrado em Engenharia Informática e Computação**

**Laboratório de Computadores**

Grupo 3 Turma 4

**Mário Fernandes up201201705**

**Nelson Almeida up201505394**

Faculdade Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

**2 Janeiro 2017**

# Contents

# Section 1: Project background and Instructions

## 1. A – User's Instructions

Simon Says is a game for children which is both fun and challenging. The original game consisted on one person (Simon) to instruct others to do what Simon says.

Sometime later, table top versions of Simon Says started to appear, but a bit different from the original. These new versions had the major difference of the number of people that could play at the same time which was reduced to either one or two players. The original table top version of Simon Says had a round system: each round, a different sequence of colours would light up and the player guessing had to follow the sequence. If the player managed to correctly follow the sequence, he would proceed into the next round which would be even harder.

## 1. B – Our Project

On our project, we've decided to adapt a version of the original table top game, which instead of a sequence of colours lighting up, a player will enter a sequence of either keyboard or mouse inputs and the opposing player has to follow the correctly. These inputs range from pressing the keyboard key's to clicking on different parts of the screen and even from making certain gestures with the mouse.

I – Main Menu

When the program starts, the users will see the main menu.

On the main menu, you can access either the game by clicking with the mouse on the button "START" or exit the game by clicking "EXIT".


*Fig 1 – Main Menu*

II – Game

When the user clicks “START” the game will start and it will be player one’s turn. Player one after a short while will see the game board and will need to make an input in order for the game to continue. These inputs can either be pressing the W, A, S or D key on their keyboard, pressing the square, the triangle, the circle or the star that they can see on the game board, with their mouse, or make either an ascending or descending positive slope.

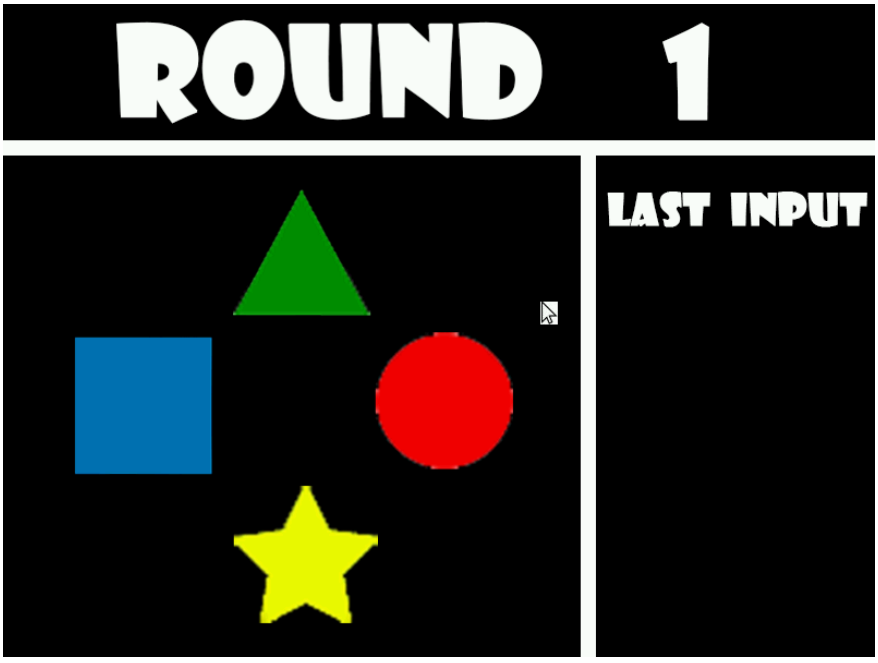



*Fig 2 – Game Board*

During the two seconds after he makes an input, the same input will be shown at the right side of the screen so the second player is sure of the input he will have to copy. The inputs will either be simple static figures like the circle or in the case of a gesture, an animated image of a mouse cursor will appear making the correct movement.

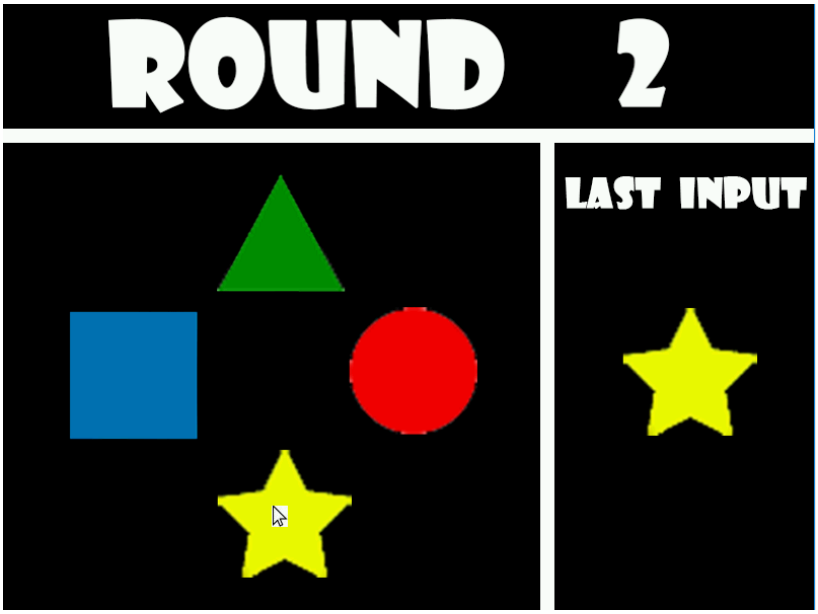



*Fig 3 – Game Board with Input*

After player one makes his first input, his turn will be over and player two will have his moment to shine, by having the task of copying the input he just saw. If he succeeds he will earn points based on the round number, in this case, he will gain one point. After the player two makes his input, a message will be shown saying that he either made the correct guess or had it wrong.


*Fig 4 – Failed Sequence Screen*


*Fig 5 – Correct Sequence screen*

From here on out, the game will go back and forth between player one and player two, until ten rounds have passed. Each round that passes, player one will have to make one more input than the previous round, and player two will also have to guess that extra input, thus, gaining more points if he succeeds, but having more difficulty while doing it.

Upon reaching the end of the last round, player two will be able to see his total score, a “thanks for playing” message will appear afterwards and finally the player will be brought to the main menu again. He will be able to choose to play again, starting the game with 0 score, from round 1, or exiting the game.


*Fig 6 – Final Score*

It is important to state that during the transition of rounds, player turns, and even input turns, the players can make inputs but all will be discarded apart from the escape key, which will terminate the game completely, at any given time.

# Section 2 – Project Status

## 2. A – Summarized table

| Device | Purpose | Mode |
|---|---|---|
| **Timer** | -Counting time<br>-Controlling pause times<br>- Controlling video card interruptions | Interrupt |
| **Keyboard** | -Inputs during the game<br>-Exiting the program | Interrupt |
| **Mouse** | -Main menu selection<br>-Inputs during the game | Interrupt |
| **Video Card** | -Game's display | Interrupt |

## 2. B – Timer

On our project, we're using the timer 0 and all of the functions related to this module are in timer.h, timer.c and needs access to some variables in gamevariables.h. This module essentially controls when a pause will occur and the display of the game.

Since the game starts, with each interruption, the timer will draw the state of the game, be it on the main menu or on the game itself. This is an extremely important issue since the functionality of a cursor is implemented and this permits a smooth, constant movement of that cursor. This would not happen if, for instance, this device was implemented with a polling mode, or at any given time during the program's activity, the timer would stop calling drawing functions.

As up to the effort put into the timer, Nelson created the timer handler that treats its interruptions and deals with the time counting, while Mário decided how could we implement timeouts that allows the game to keep the same image on the screen for a certain period of time.

## 2. C – Keyboard

The keyboard is controlled by the kbc.h and kbc.c, the keyboard.h and keyboard.c and the variables such as the game's input code for the W key are stored in gamevariables.h. This device is essentially used for in-game inputs with the W, A, S and D keys. It also has the option to close the game if the "ESCAPE" key is pressed at any given moment.

Mário created the function that reads the keys pressed by the player every time a keyboard interruptions appears, while Nelson was in charge on when should this inputs be considered or not.

## 2. D – Mouse

The mouse has all of the mouse related functions saved in mouse.c and mouse.h, while also needing access to some functions in kbc.h and kbc.c and to some variables in gamevariables.h

The mouse has the essential task of selecting inputs or options shown in the screen. To do this, mouse coordinates are stored based on the current image of the cursor and the mouse simply gives the "impulse" to that image. In other words, the real position of the mouse is irrelevant, since the image's coordinates are the one's taken into account while pressing the left mouse button. There are also inputs which involve pressing the right mouse button and making a positive slope for a short distance.

About the work we've put into this device, Nelson was responsible for the code that is able to detect a positive slope mouse movement, as Mário was the one in charge with synchronizing the mouse movement with the cursor that appeared on screen, we both worked on the function that allows the game to recognize which type of mouse input was given the player.

### 2. E – Video Card

The video card has many modules which are the video_gr (.h and .c), the vbe (.h and .c) and the bitmap (.h and .c). The video card is used to display what's happening in the game, be it the .bmp images of the board and inputs or the mouse cursor, which is refreshed as fast as possible. All of the display of the game is used in conjunction with a double buffer so that the game never "blinks". With the usage of the double buffer, we prevent the game from showing images that are not yet ready.

It was up to Nelson to create the used images in .bmp format using photoshop and the draw functions that allows the images to appear on screen, while Mário was responsible for the implementation of the mouse cursor on the screen, as well as the double buffer idea behind it.

## Section 3: Code Organization/Structure

On this section, we will talk about each module individually, specifying their most important functions.

### 3. A – bitmap (.h and .c) – 7%

Adapted code from Henrique Ferrolho which involved loading, reading and drawing of .bmp images.

- "loadBitmap" reads a .bmp file and returns a struct containing the file's information;
- "drawBitmap" draws the .bmp onto a received buffer, on our project's case, it will always draw to the double buffer which, when ready, will give the final output to the screen.
- "get_bitmaps" defines previously declared Bitmap's structs.

Most of the rest of the functions in this module are fully made by us, which involve calling drawBitmap in order to have something prepared on the double_buffer.

### 3. B – game (.h and .c) – 30%

Main section of our game, having two state machines, one for drawing and one for changing the game state.

- simonsays is in charge of the game cycle and dealing with the hardware interruptions.
- state_machine is in charge of changing the logical changes to the game.
- timer_sate_machine is in charge of changing what the users are seeing on the screen at any given moment.

This module is constantly drawing on screen so that the user is always permitted to see where his mouse cursor is. In order to do this, there was a need to implement various cycles in which every

round has many pause times. In each pause time, the game state and input at that moment will always be the same and when the pause time ends (controlled by the timer), the game state will change and the control over the inputs will be given again to the user. During the pause times, each input is discarded apart from the escape key and the way that that is done, is by using an input buffer. This input buffer is analysed and if it's a valid input and there is no timeout at the moment, then that will be transmitted to the actual input and to the actual game itself.

During the game cycle, when a round is over, two arrays will be compared, one for player one and another for player two. These arrays were filled beforehand with the valid inputs from the players. If the arrays are the same, then the score of the player two will be incremented.

## 3. C – gamevariables.h – 1%

Header file with variables used throughout the whole game, mostly "defines".

## 3. D – kbc (.h and .c) – 2%

This module only contains two functions that analyse the state of the status buffer in order to read or write successfully. If by any chance one of the read fails, it will be tried again for as many tries as declared on the TRIES variable.

## 3. E – keyboard (.h and .c) – 15%

This module is in charge of handling the keyboard interruptions.

- KBD_subscribe_int subscribes the keyboard's interruptions
- KBD_unsubscribe_int unsubscribes the keyboard's interruptions
- KBD_handler reads an interruption and if it's the ESCAPE, W, A, S or D returns a code to be interpreted by the game module.

## 3. F – mouse (.h and .c) – 20%

This module is in charge of handling the mouse interruptions.

- MOUSE_subscribe_int subscribes the mouse's interruptions
- MOUSE_unsubscribe_int unsubscribes the mouse's interruptions
- MOUSE_handler reading and synchronization of the mouse packets; when the packets are ready to be interpreted, calls MOUSE_analyze_input.
- MOUSE_analyze_input analyses the given input and returns a code to be interpreted by the game cycle; even if the code is invalid, this function also will update the mouse coordinates in order for the cursor to be drawn successfully.

## 3. G – simonsays.c – 3%

This module is the first to be called when the user runs the program.

- main analyses the written input before the program starts, and if it's correctly written, it starts the game

## 3. H – timer (.h and .c) – 15%

This module is in charge of handling the timer interruptions.

- TIMER_subscribe_int subscribes the timer's interruptions
- TIMER_unsubscribe_int unsubscribes the timer's interruptions
- timer_handler handles the timer's interruptions, incrementing by 1 unity a counter with each interruption and by 1 unity another counter for each second that passes.

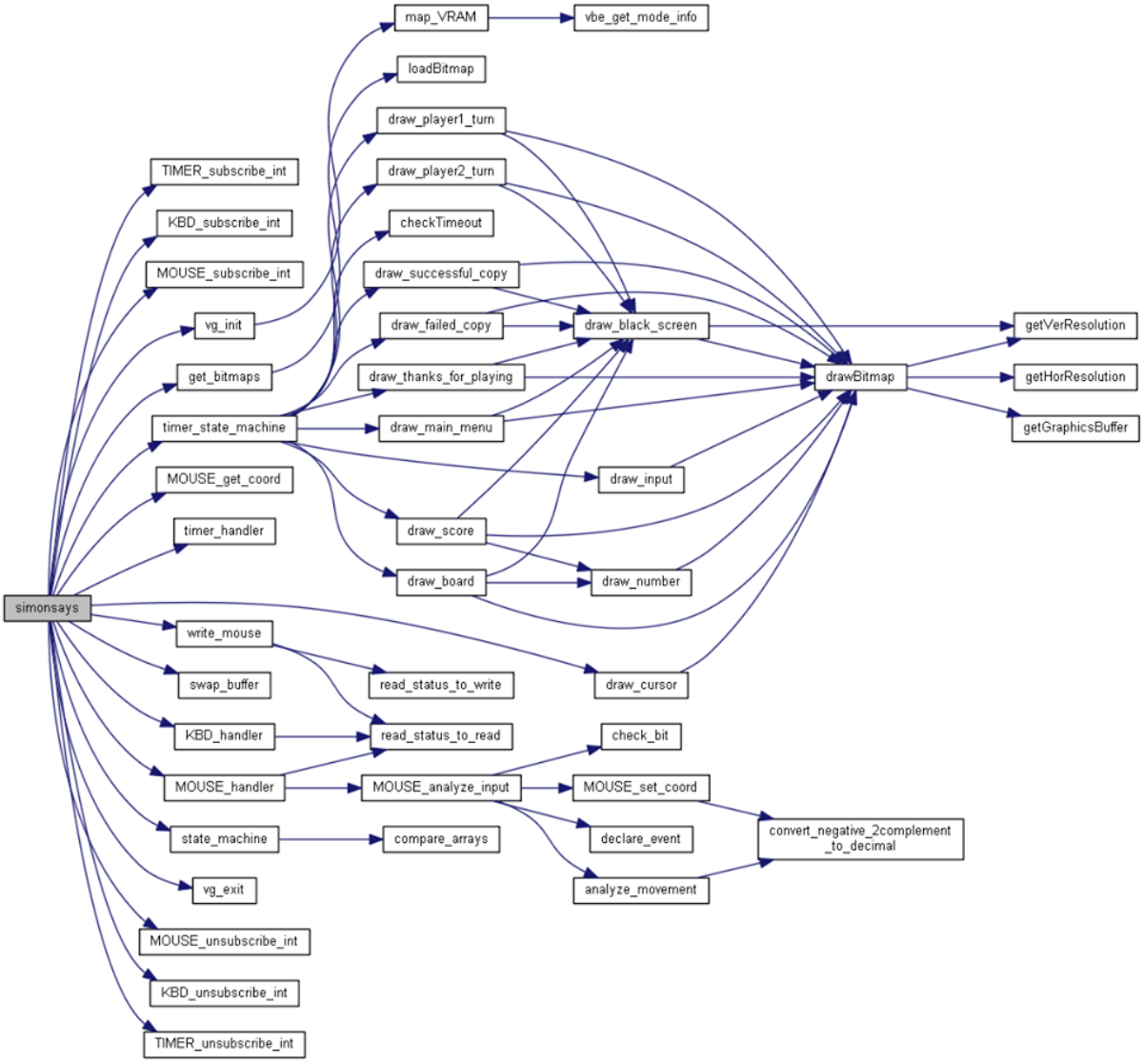
## 3. I – vbe (.h and .c) – 2%

This module is in charge of accessing the BIOS in order to get valuable information such as the resolution that is currently being used in the game.

## 3. J – video_gr (.h and .c) – 5%

This module is in charge of handling the graphics card's functions. It uses a double buffer and has the swap_buffer function to swap between that buffer and the game's video address.

The following graphic describes the process of function calls.



# Section 4 – Implementation Details

The project was a bit hard to make considering we had very little guidelines on what we were supposed to do; that's what made it a lot more enjoyable to do compared to previously labs done.

## 4. I – State machines

Since it was our first opportunity to implement real state machines, we had some initial issues while trying to adapt to this way of coding. After we got used to it we even decided to make two different state machines: one for handling the logic of the game and the other to handle what's being shown on the screen.

We've had some trouble synchronizing everything well and passed through some different stages. Some of these involved having only one image showing at a given pause time which would work fine apart from the mouse movement and the cursor animation on the right side of the board. Another example of a significant change that had to be done during the project's development was shifting every visual modification to the timer's interruption section; before this change, the drawing functions were spread throughout the timer's, keyboard's and mouse's interruption sections.

## 4. II – Mouse coordinates

One of the big challenges on this project was the implementation of a mouse cursor which in theory we figured very quickly how it had to be done, but in practice, it was quite troublesome.

There was a need to create new variables, the X and Y mouse coordinates, and with each mouse interruption they would be updated with how much the mouse moved. Upon reaching the edge of the screen the mouse coordinates would maintain their values, forcing the cursor to stay inbounds of the screen.

One of the most important aspects that we've learned while doing this was the fact that our own mouse cursor's coordinates are irrelevant to the game, only the impulses we give with our mouse matter in terms of moving the cursor's coordinates.

## 4. III – Interruption synchronization

Another challenge that we've found doing this project was the need to have a well-done synchronization between each interruption.

In order to analyse our interruptions, we've decided to simply doubt of every interruption we receive. Every interruption, no matter if it's valid to our game or not, count. Therefore, every interruption goes into a variable called "input_buffer". First off, this variable is checked after it is defined by an interruption, so that we know if the interruption made was a valid one or not. After that, we analyse the state of the game in order to either take the input into account or discard it completely.

We started with this logic in order to have a very easy time synchronizing with a multiplayer game using a serial port, but due to the lack of time we could not complete that functionality.

### 4. IV – Drawing synchronization

With each timer's interruption, a new image is shown. We've decided to implement this section like this to have a very smooth control over the mouse with a very high frame rate. Although the higher the frame rate, the more the cpu consumes, we've decided to implement it this way on purpose since we consider frame rate a very important factor, especially to see the mouse cursor not "flash".

This synchronization is done on the timer's state machine which has a round starting at "START" and ending with "ROUND_TRANSITION2", and every time a background image changes, a new state has just started. The fact that this function has both 1 and 0 as possible returns indicates if there is a new pause time pending or not, respectively.

### 4. V – Usage of Bitmaps

Although on the labs we've were taught how an image responds in minix, there was no coverage of loading and using actual images (apart from xpm).

This was also a hard task to do but fortunately we encountered Henrique Ferrolho's code on how to load and use bitmaps and with the permission of our teachers we used it on our project. This made things a lot easier although we had to read and understand every single line of his code in order to know what was happing and to make the changes to adapt to our program (such as making the bitmap's draw function draw to our double buffer).

## Section 5: Conclusions

Although we found "Laboratorio de Computadores" to be an extremely demanding and sometimes even frustrating course compared to the one's we've had so far, it was by far the most rewarding at the end. The fact that we had many difficulties during this semester with this course proved that it was hard, but the fact that in the end, we had overcome every problem along the way made us somewhat proud. In short, LCOM is a life experience, unlike other regular courses.

As far as our group connection goes, we found no problems. We always got along and no complications existed about unequal contribution. We've always put the same and maximum effort either throughout the labs as well as in the final project, and that's one crucial point for a group to work properly: team work.

Besides what we just referred we consider the following aspects to be positive on this course:

- Usage of C language
- First “real” application of our programming skills.
- *Almost* direct communication with the hardware, possibly the closest many people will have on their life.
- The teachers have a good grading policy, even sometimes giving some margin to make mistakes.

We consider the following, negative aspects of the course, although some of them are completely unavoidable and have to take part in an Engineer’s life:

- By far, the ratio of work to do to ECT’s given for the completion of the course is extremely low.
- The fact that the course seems to never be updated, information wise, even having previous years mistakes on some labs like the mouse one. This lab has had wrong information for at least 3 years in terms of the mouse button configurations, and (at least this year) it had another mistake in which the gesture had no “tolerance” parameter but on the powerpoints had. These aspects lead to a lot of confusion making us waste valuable hours searching for problems that weren’t there.
- The fact that deliveries are made right at the end of the class. This topic, to us, was infuriating since it made us have a bad mark on a lab we’ve managed to correct 2 hours after the class was over. The fact that labs are meant to be done outside of classes (since it’s literally impossible to start and finish labs during the classes) defeat the purpose of having them be delivered during a class. On our opinion, practical lessons should not even exist or at least have them be optional since the only purpose we’ve had for them the entire year was to have some questions answered. This would easily be done either by moodle, e-mail or going to the attending ours of any teacher. Not only that, the teachers themselves have a very hard time dealing with an entire classroom filled with students and their specific problems.
- Finally, the fact that there is an overload of work for this course, makes people who have courses from different years very uncomfortable, for instance, one element of our group was a transferred student from another course which had 2 second grade courses and 4 third grade courses to do, which proved impossible for him to work on LCOM during the third week of December, which made it impossible in terms of time to work on the RTC and Serial Port modules, although we tried to complete both of those labs. We found this to be very demotivating since we had high hopes to have a very complete project.

# Section 6: Appendix

Our program needs to be on the directory /home/lcom/lcom1617-t4g03/proj/src on minix to work properly. To start simply use the make command to compile it and then write “service run `pwd`/simonsays -args “simonsays”.

There is also the need to have the file provided by the course lmlib.a on the src of the program which proved to be impossible for us to upload that file through minix.