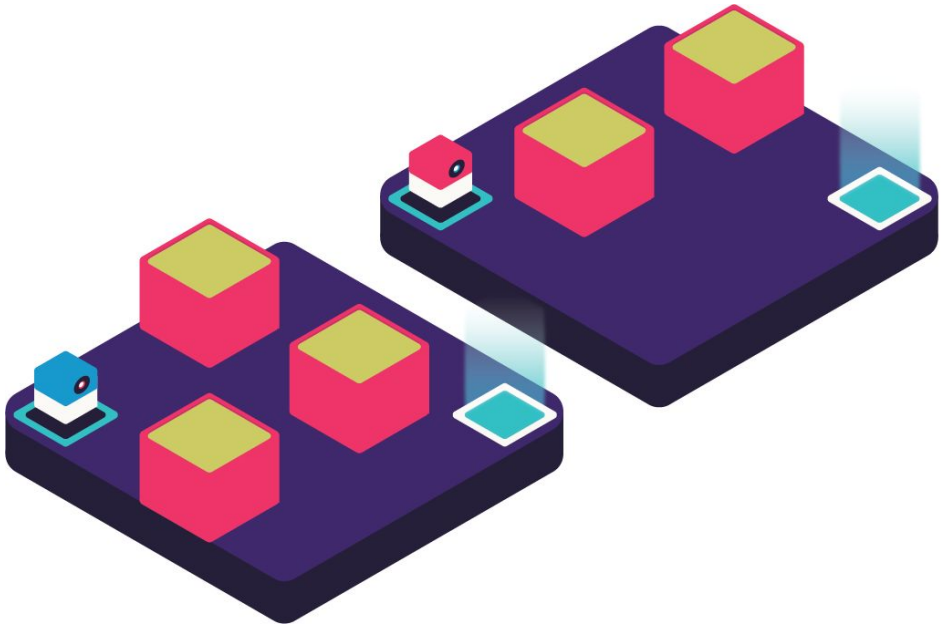
50.002 Computation Structures 1D Project

TWINS



Group 3-7

Ivan Chuang 1002744

Ryan Yu 1002769

Tan Zhao Tong 1003031

Tee Zhi Yao 1002845

50.002 Computation Structures 1D Project

TWINS **Group 3-7**

1. Introduction

Twins is a simple 1-player game, where the player has to move two particles to two end positions simultaneously. However, there is only one set of controls that controls both players. The challenge comes in guiding both particles through two different maps for player A and player B.

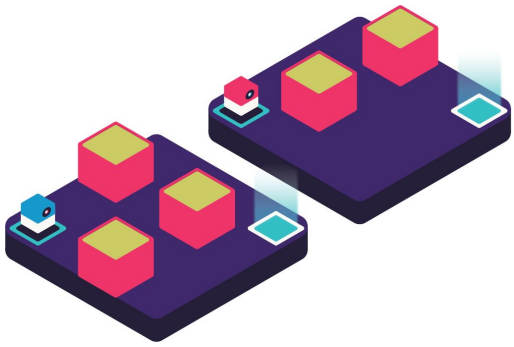


Figure 1: Artistic Vision of Twins

2. Game description, design, test scenarios

Twins is a maze-based logic game.The plot of the game is that there are 2 robots stuck in a maze. The robots can only move onto empty spaces in the maze. In order to escape, they have to reach their exits at the **same time**.

Using an LED matrix, two 4 by 4 maze maps will be displayed. The player's goal is to guide the robots to their exits. There will only be 1 set of controls which moves both robots in each map simultaneously. The player will only win when both robots reach their respective exits at the same time. The game touches on the player's cognitive flexibility, to solve 2 mazes at concurrently.

Figure 2 is an example of a typical game level.

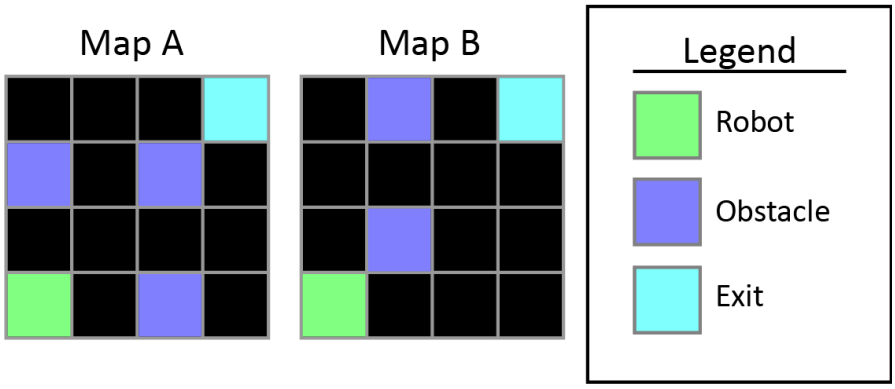


Figure 2: An example of a typical game level and its components.

Inspired by the idea of a gameboy, our hardware was designed to be intuitive, hence resembling the controls of a gameboy.

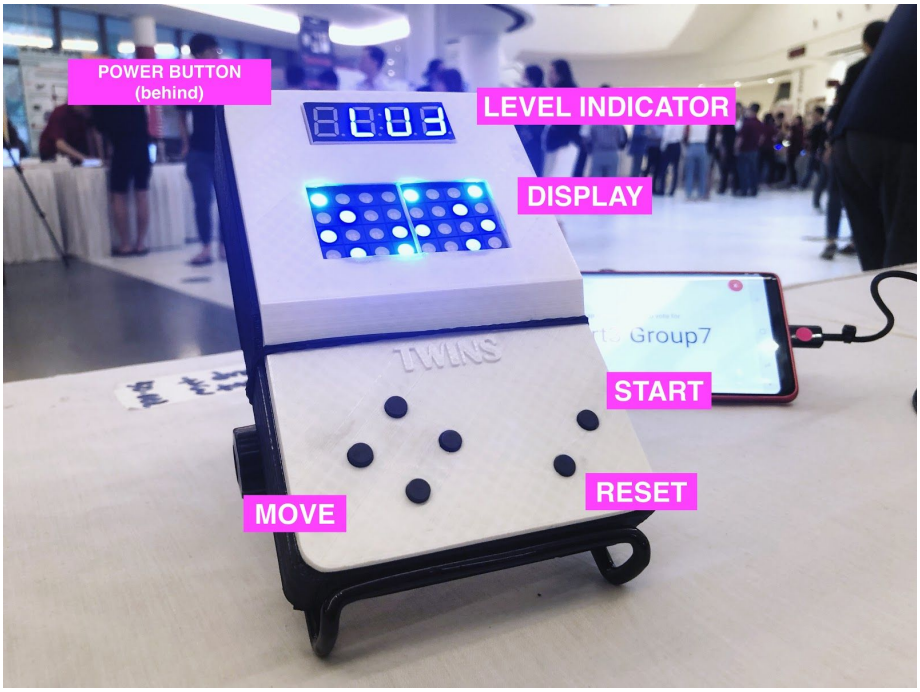


Figure 3: Final Build

3. User manual

Rules:

- Controls will move both robots on the maze
- Robots can only move onto empty spots
- Robots cannot move obstacles
- Both robots have the reach their exits together to win the level.
- When only 1 robot reaches the exit, it is a lose and the level will restart.

How to play:

1. On the map selection screen, use the direction buttons to scroll through the different levels.
2. Press the ‘select’ button to choose a level. Now you’re ready to play the game!
3. Use the direction buttons to move the robots to the exit. Remember the rules of the game.
4. If you’re stuck, press reset to move the robots back to their starting positions.

Hint: Obstacles aren’t there to just block you. Use them to your advantage!

4. Building the prototype - Hardware

4.1 Outer Casing

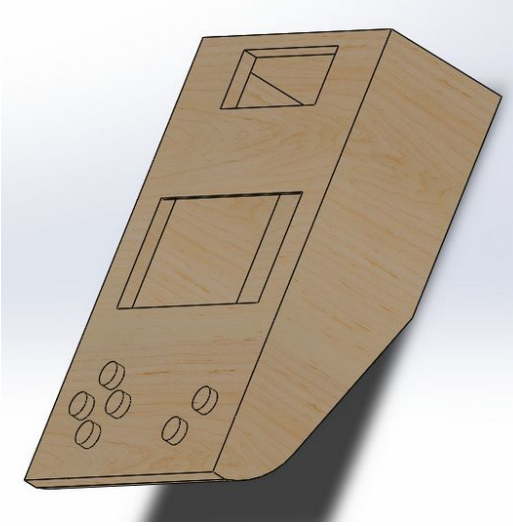


Figure 4: Initial Design of CAD

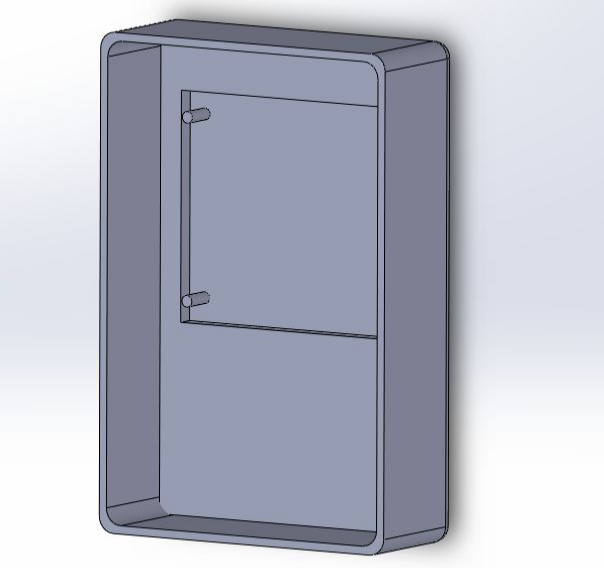


Figure 5: Improved Design

Initial Design of CAD. Inspired by design of a Gameboy

Problems faced:

- A nice curvature will be hard to achieve regardless of using 3D printing or laser cutting.

Solution:

- Next iteration of CAD drawing takes the shape of a Box or Tin (Figure 5).

Final Prototype:

Design Considerations:

- Holes for button inputs are lofted with exact dimensions to create a fitting for the buttons.
- Edges are rounded to prevent sharp edges around the casing.
- Buttons used are now of the same size for standardization.

Figure 6: Final Prototype Design

4.2 Wiring & Prototype board

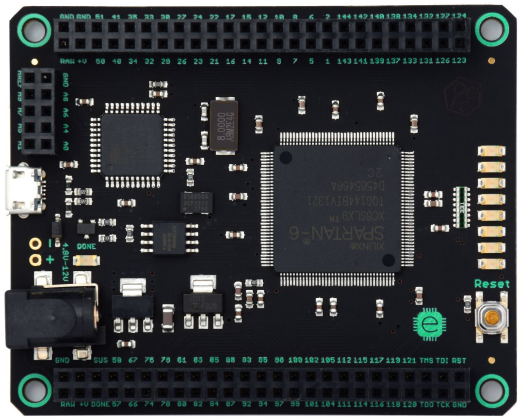


Figure 7: MOJO

Pins were defined such that we are able to reduce the complexity in the circuit design. The pinouts on the Mojo board were divided into four rows. We mainly used the 2 inner rows(row 2 and row 4) to configure our pinout.

5. Building the prototype - Software

For efficiency, we tried to modularise the software as much as possible, allowing each of us to work on a specific module. We improved and simplified the beta diagram from Checkoff 3 (Appendix B) and use it as a reference to create the building blocks of our game. The following, figure 7, shows the summary of our different modules and how they interact with one another.

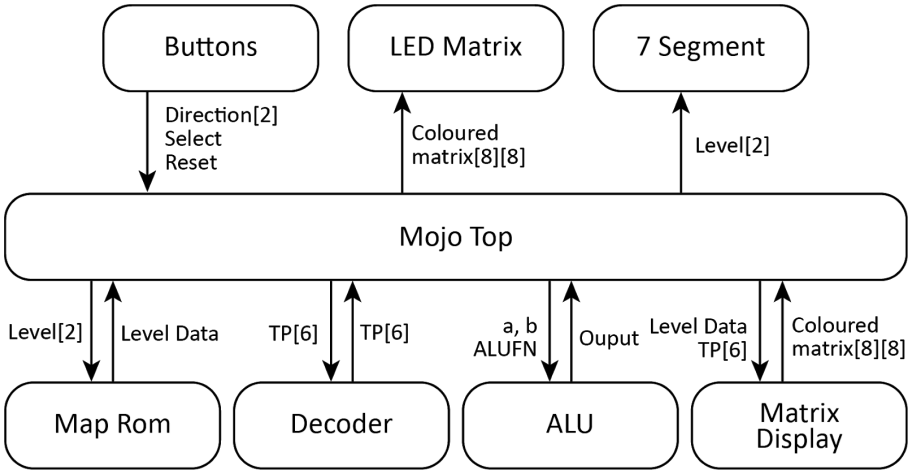


Figure 8: Summary of mojo modules with hardware

Being modular has allowed us to debug the codes easily. For example when the LED Matrix was not displaying the maps in the map rom, we could check if it was a hardware problem by connecting map rom directly to the LED Matrix. Then slowly work our way up by adding Matrix Display to the system to fix the positions of the maps on the LED Matrix, so on and forth.

The following explains the properties and components of each module.

The source code can be found here.

Buttons

The *down, up, left and right* buttons will set the direction register to 0, 1, 2 and 3 respectively when pressed.

The *select* button sets the select register to 1 when pressed.

The *reset* button initialise reset register to 1 when pressed.

LED Matrix

An 8 by 8 LED matrix is used to display two 4 by 4 maze maps of the game.

The remaining 32 leds are hidden above and below the two 4 by 4 maps. Being hidden by the case, these LEDs are still used to add simple graphical effects to the game, such as when the player selects a new level, or when the player is able to finish a level. This, along with the 7 segment display, is used to add some personality to the game, to better engage players to continue trying and playing the game.

7 Segment

The 7 segment is mainly used to display the status of the game. On the map select screen and the main game loop, the 7 segment displays the currently selected level.

The 7 segment is also used during transition states to help signpost what exactly is happening in the game. For example, when the player chooses a new level or restarts the current one, it displays ‘play’. If the player loses, it displays ‘ded’, indicating that something is wrong, and it shows ‘yay’ if the player completes the level.

Decoder

Decoder takes in current map data (Map Data) and the new current position (TP) of the player. Using the coordinates from TP, the decoder will decide if the position of the player should be updated. If the output is 0, it means the spot is empty, so new position

of the player will be updated to TP. If the output is 1, it means there is an obstacle, hence the player's position will not be updated.

Map Rom

Map Rom stores all the data of every level. These data include an 6 by 6 array for the maze (Map Data), start position (SP) and end position (EP) for each map.

Map Data contains the position of walls and obstacles. 0 indicates empty spots. 1 indicates wall/ obstacle. Refer to figure 8 for more clarity.

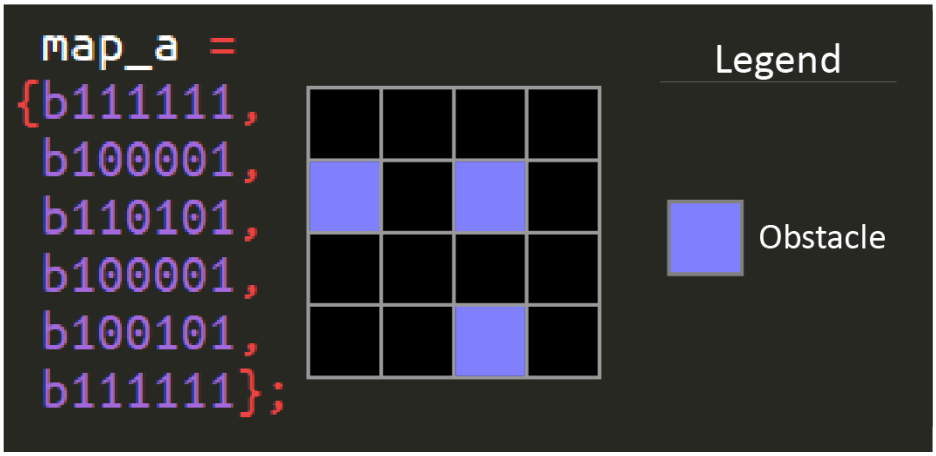


Figure 9: Illustration of Map

Map Data is passed to the decoder to update the current position (TP) of the robot.

SP, EP and TP are all 6 bits. First 3 bits indicates the x position while the last 3 bits indicates the y position.

EP is passed to the ALU to compare if TP equals EP.

All level data is passed to Matrix Display to convert them to a 8 by 8 matrix for display on LED Matrix.

ALU

During state 4 and 5, ALU takes in EP and TP and **compare** if the values are equal. The outputs will be stored in registers R1 and R2 for map a and map b respectively.

If output is 1, it means the robot is the exit.

If output is 0, it means the robot is not at the exit.

During state 6, ALU will do a boolean **AND** check of R1 and R2. Output R0 is returned to mojo top.

During state 7, ALU will do a boolean **OR** check of R1 and R2. Output R0 is returned to mojo top.

Matrix Display

Matrix Display takes in Map Data, SP, EP of the level and TPs of the robots. It creates a new 8 by 8 array to display the Map Data, TP and EP. Map A will take the left side while Map B will take the right. Only the playing area, 4 by 8, will be visible to the players. The remaining LEDs of the matrix is hidden by the casing, and its use is described above.

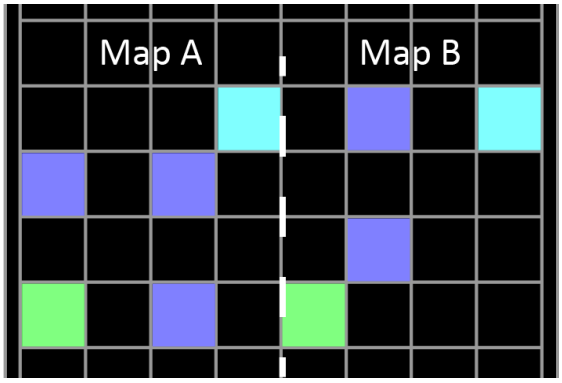


Figure 10: Overview of Matrix Display

Game Logic

We used a finite state machine with nine states to implement the main logic of our game. Each state determines the control signals (Appendix C), inputs and output sent to each module. Figure 10 shows the state transitions.

This state diagram was concurrently with the software. We realised that the registers(flip flops) will only update at the next clock cycle, hence we had to introduce more states to ensure that the game was functional. This was significant for refreshing the LED Matrix, State 1 is used solely to display the maps while State 2 is used to update the levels.

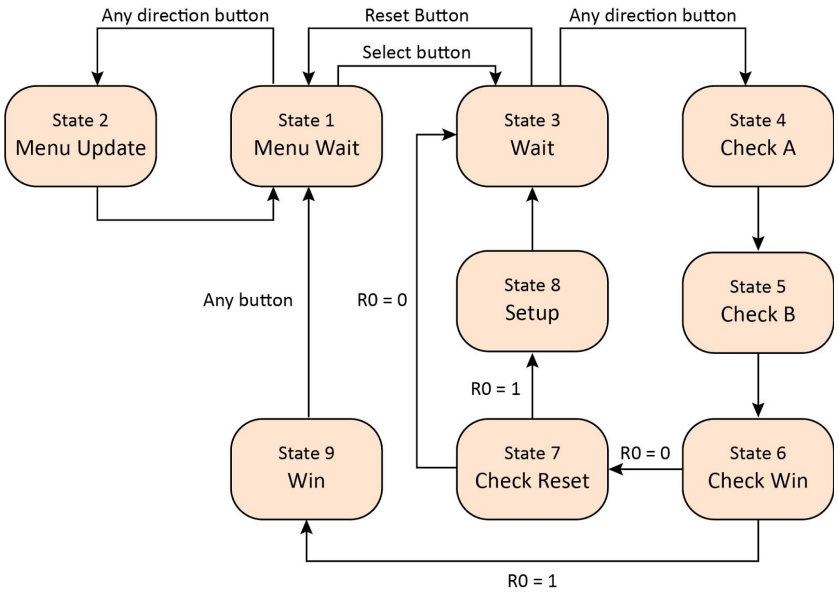


Figure 11: Finite State Machine Diagram

State 1 - Menu Wait, allows players to select the levels. Up and right buttons will increases the level while down and left buttons decreases the level.

When select is pressed, SP values are written into TP registers of the robots.

State 2 - Menu Update, will update the level register to the index of the new level.

State 3 - Wait, waits for a direction input from the player.

State 4 - Check A, checks if position of robot A is moving to has an obstacle. If there are no obstacles, update the TP of the robot. Next check if TP is equal to EP and store output in R1.

State 5 - Check A, checks if position of robot B is moving to has an obstacle. If there are no obstacles, update the TP of the robot. Next check if TP is equal to EP and store output in R2.

State 6 - Check Win, does a boolean AND check on R1 and R2.

If output is 1, it means both robots are at their exits, move to state 9 - Win.

If output is 0, move to state 7 - Check reset.

State 7 - Check Reset, does a boolean OR check on R1 and R2.

If output is 1, it means one of the robot is at their exit, hence move to state 8 - Setup, to restart the game.

If output is 0, it means none of the robots are at their end position, move to state 3 - wait, for the next direction input.

State 8 - Setup, resets the game. It will write SP of robots to TP.

There is an additional state called ‘Dead’, which performs function as the ‘Setup’ state, but displays the text ‘ded’ in the 7 segment display instead of ‘play’.

State 9 - Win, player has won the game.

6. Components budget

Components	Price($)
LED Matrix (2 were purchased)	$21.95 (2 * $9.98 + $1.99 shipping)
Buttons	$0 (From DSL)
Bostik Blue Tack (To hold components together before actual soldering work)	$2.60
40 Female to Female Jumper wire (20cm)	$5
3D printing	$0 (Fab Lab)
4-digit 7-segment display	$0 (From DSL)
Total Price($)	29.55

Table 1: Budget Breakdown

7. Summary

Building a game from scratch is not easy, especially when we were asked to build it on a software that we are not familiar with. The hardest part of the project was putting the hardware and software together. In addition, we needed to compact the hardware into our small gameboy-like casing. However it was a great learning process. The greatest satisfaction gain was during exhibition when we see students enjoying the game.

8. References

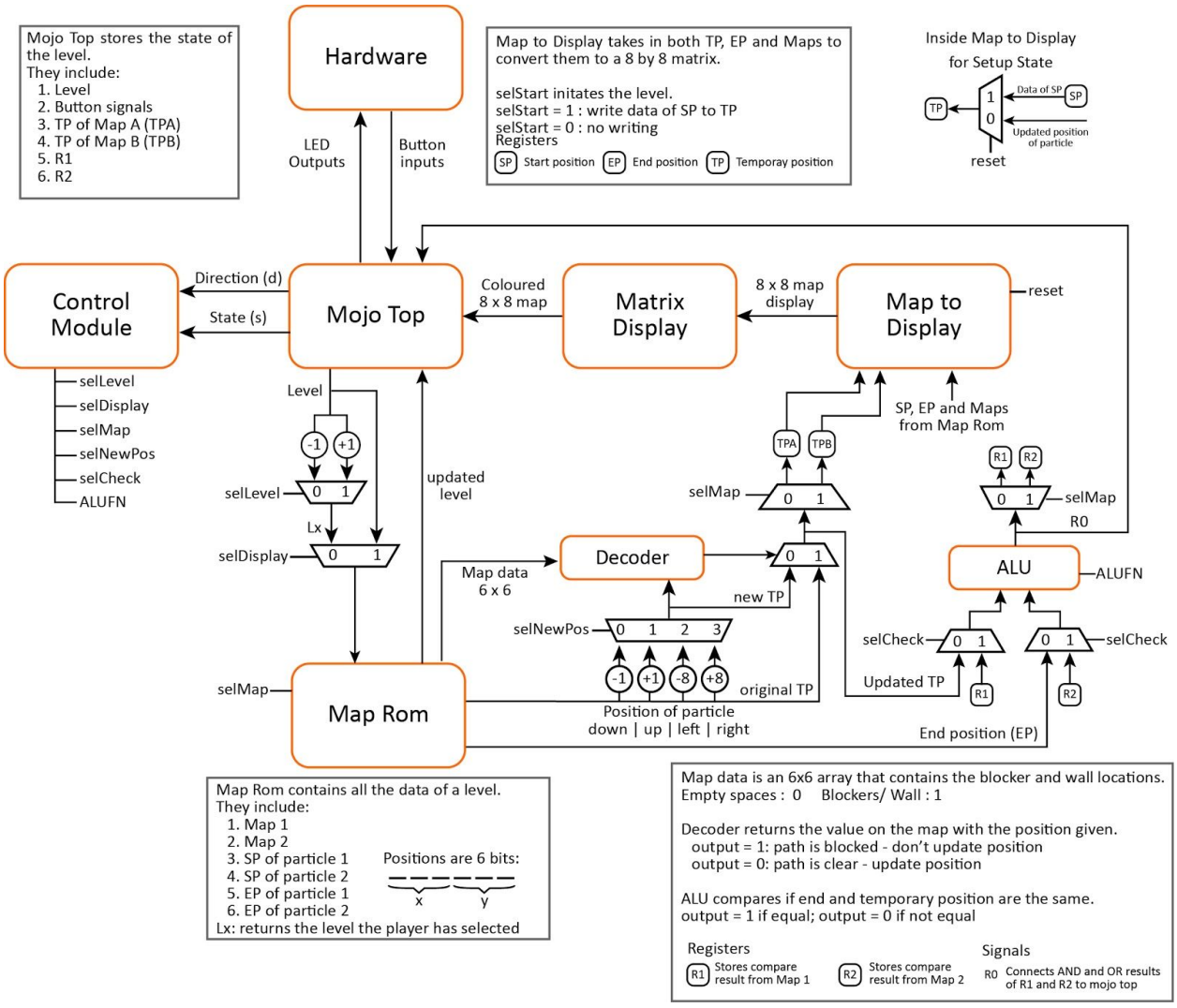
Evan-Amos [Public domain], from Wikimedia Commons

https://upload.wikimedia.org/wikipedia/commons/a/a9/Game-Boy-Pocket-FL.jpg

Past years 1D projects

9. Appendix

Appendix A: Beta Diagram



Appendix B: Control Signals

s	1	2	3	4	5	6	7	8	9
selLevel	d[0]	--	--	--	--	--	--	--	--
selDisplay	0	1	1	1	1	1	1	1	1
selMap	--	--	--	0	1	--	--	--	--
selCheck	--	--	--	0	0	1	1	--	--
ALUFN	--	--	--	==	==	AND	OR	--	--
selNewPos	--	--	--	d	d	--	--	--	--
reset	--	0	r?1:0	--	--	--	--	1	--

s 1: Select level
s 2: Update level
s 3: Wait for movement
s 4: Check Map A
s 5: Check Map B
s 6: Check Win
s 7: Check Reset
s 8: Setup
S 9: Win

Buttons	Signals set
Down:	d = 00
Up:	d = 01
Left:	d = 10
Right:	d = 11
Select:	s
Reset:	r

Appendix C: Project management log

Task	**Person in Charge**
Idea generation	Everyone
Generating game logic	Everyone
Hardware design	Zhi Yao
Writing code	Everyone
Building prototype	Ryan and Zhi Yao
Generating difficulty levels for game	Ivan and Zhao Tong
Designing poster	Ivan
Writing report	Everyone