“The A-Maze-ing Game”

EE/CS 172L/130L: Digital Logic & Design

Syeda Haya Fatima, Dua e Sameen, Saira Junaid, Iqra Ahmed

20 Nov 2022

Abstract

“The A-Maze-ing Game” is a rendition of the classic single-player maze game in which the user navigates through the maze to escape it. The user interface consists of four, individual buttons. The user controls the buttons which each represent an aspect of the player’s movement: up, down, left, and right. The movement of the in-game player (portrayed by a single colored blip) is determined by the player through the manipulation of these buttons in any order. The original position of the player is known and the user can start moving the player from that point onward. The game continues until the player escapes the maze or the timer runs out. If the player manages to figure out the maze and escape it, a winning message is displayed. However, if the timer runs out before the player can do so, a losing message is displayed. Therefore, the game is essentially a race against time.

I. Introduction

Playing instructions

The player starts from a set point on one side of the maze and is supposed to navigate through the screen, however it is restricted to move within the walls surrounding it as collision with the walls results in no further movement (the player cannot move over or through the walls). The player is expected to exit the maze and reach a certain end point to win within the given time.

General controls

Up Button => Move up

Down Button => Move down

Left Button => Move Left

Right Button =>Move right

Start Button => Enters game state

Input Block

Input block manages user input. It relies on the input from 5 buttons and translates it to keep track of the movement button which the user pressed and cease movement if no button has been pressed.

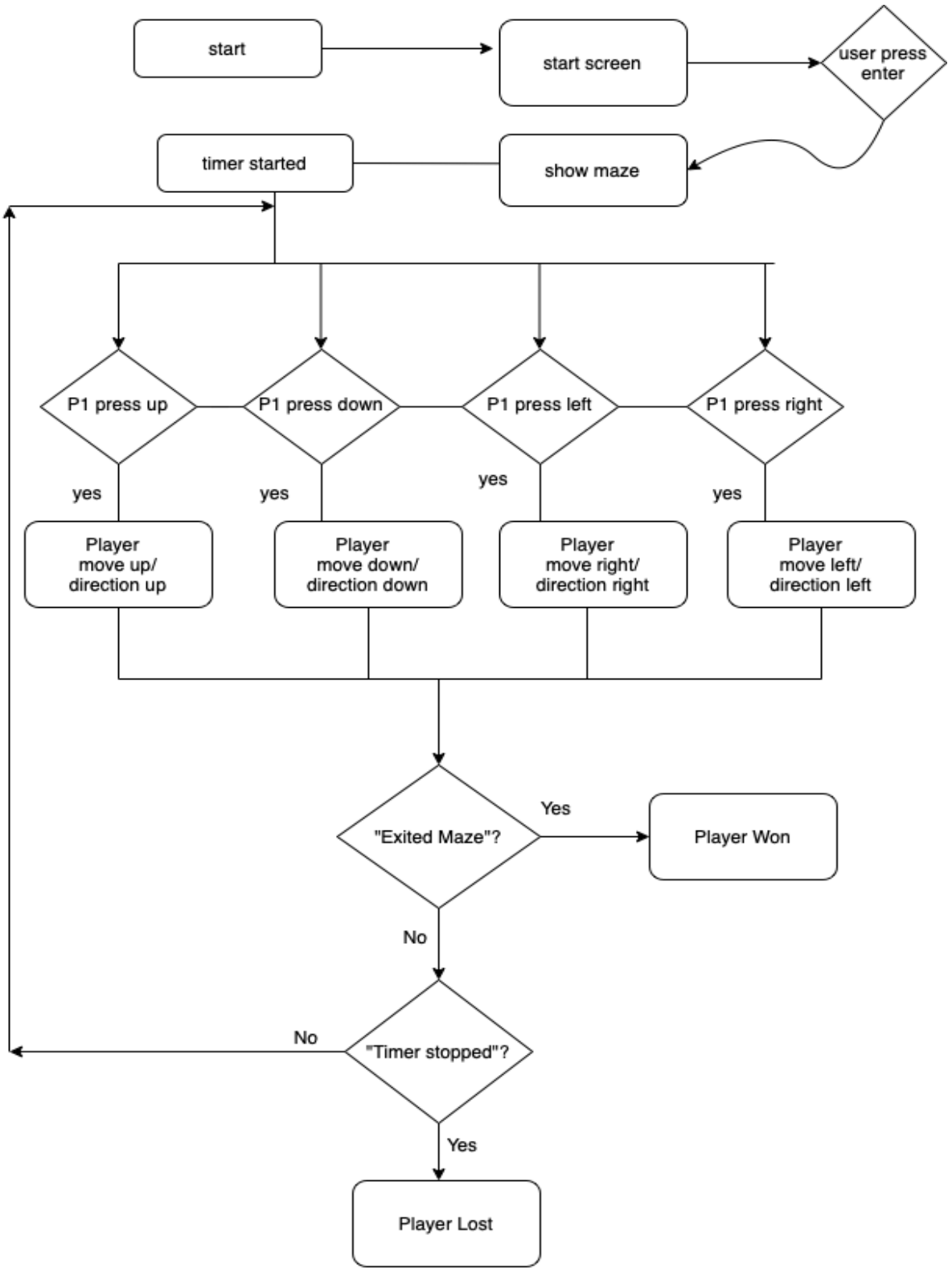
Output Block

The output block relies on a VGA display. Based on the data from the control block, it displays images of the maze and the current position of the player in the maze.

Control Block

The Control Unit of the system is mainly responsible for three tasks; processing user inputs, calculating the user's score, and controlling the mode of the game. This is accomplished using a major minor FSM setup. The control unit consists of five modules; the Menu FSM, Game FSM, Time Checker, Exit checker and Movement/Collision modules. Time checker and Exit checker constantly compute the state of the game while the control unit transitions between the other modules like an FSM.

User Flow Diagram



II. Input Block

Input block will manage inputs from external push buttons that are connected through the FPGA.

Input Block Implementation

The pushbuttons are connected to the FPGA via series resistors to prevent damage from inadvertent short circuits. The buttons are configured to generate a low output when they are at rest, and a high output only when they are pressed.

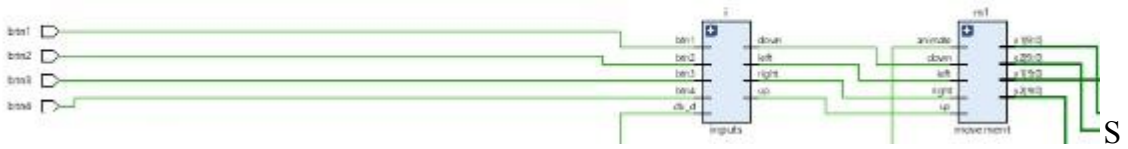
Pin Configuration

Input	Pin
-	PWR
-	GND
Btn1	P18
Btn2	N17
Btn3	M18
Btn4	K17
Btn5	R18

IO Details

Button	Movement
Btn1	↑
Btn2	←
Btn3	↓
Btn4	→
Btn5	Changes screen

Input Block Diagram



III. Output Block

Video Component

VGA (Video Graphics Array) is a video display standard introduced in the late 1980s in IBM PCs and allows for a display resolution of 640x480 pixels it is widely supported by PC graphics hardware and monitors. VGA is responsible for displaying pixels to the video screen. It uses a counter to calculate 640 pixels horizontally across the screen, followed by a blank period where the cathode ray tube (CRT) situates itself back to the beginning of the screen. This process continues until all 480 rows have been drawn, at which point the vertical blank occurs, during which the CRT resets back to the upper lefthand corner of the screen. The VGA module takes in the pixel clock signal and outputs the horizontal and vertical location of each pixel as it is displayed. The pixel_gen module assigns an RGB value to every pixel of the screen which is called by a top-level module vga_controller (the main display module) which generates and updates the output of the screen coming from the Control Block. These location signals are sent to the Top Level Module, which assigns the correct color values of each pixel.

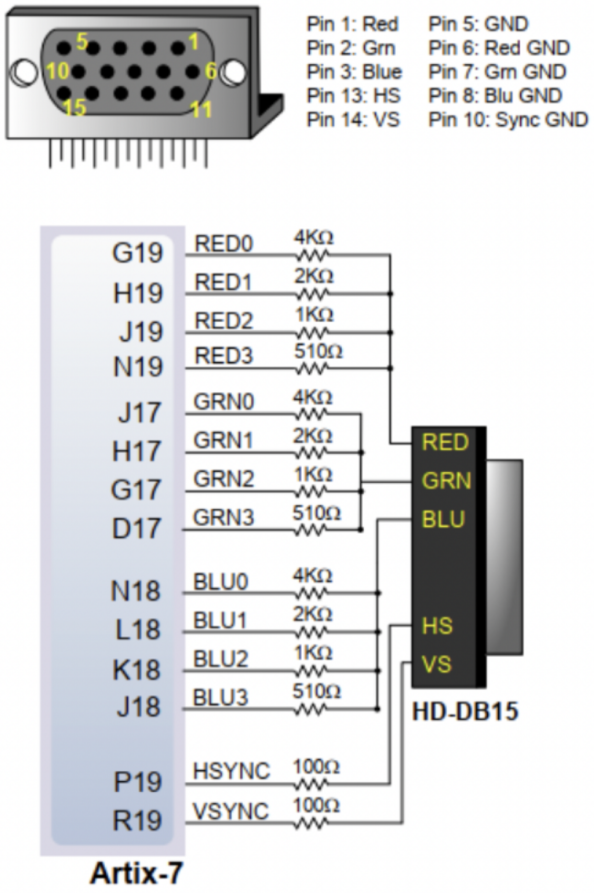
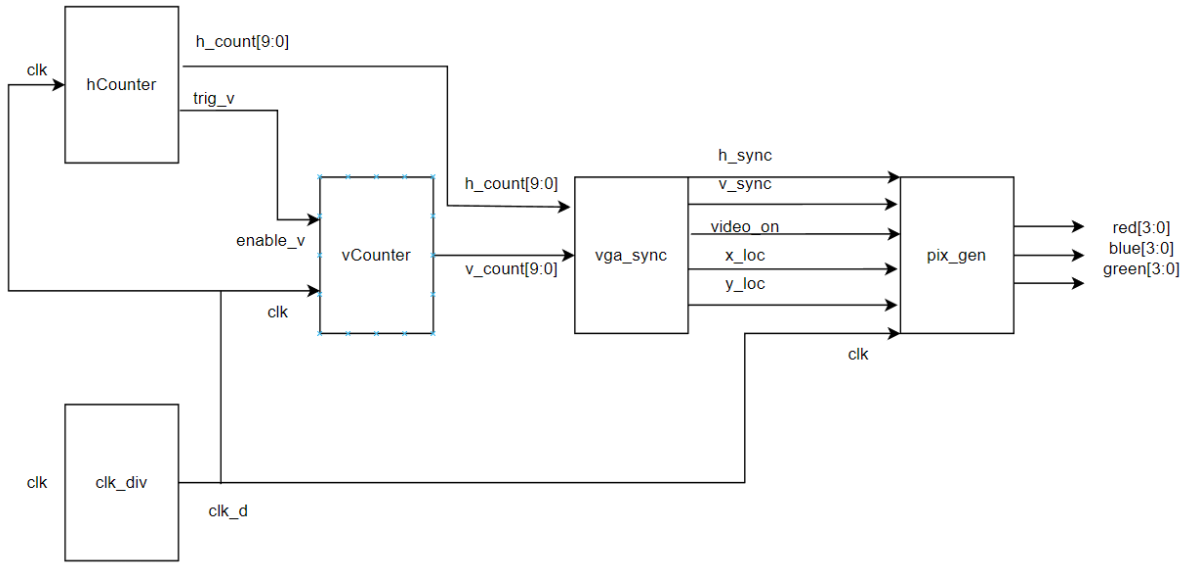


Figure 11. 9: VGA port connections

Output Block Diagram

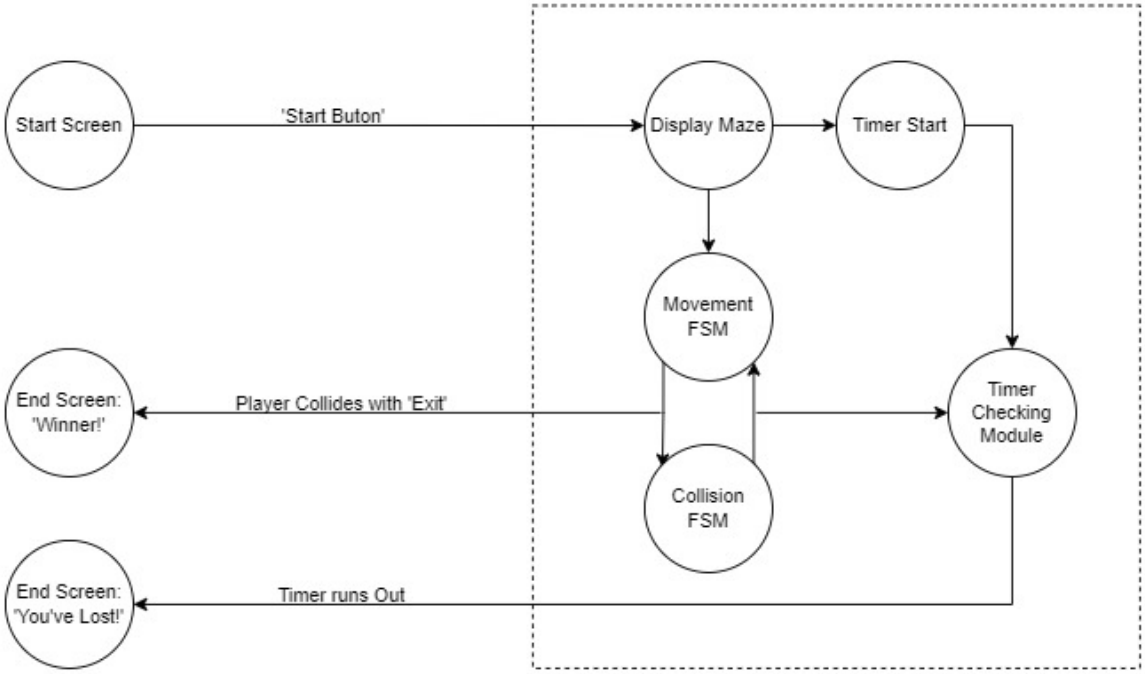


IV. Control Block

Control block will manage the finite state machine and it consists of four main states:

1. Start screen
2. Game screen
3. End screen: Loss/Timer ran out
4. End screen: Win/Player exited maze

Top Level Control Block



This top level control block explains the main mechanics of the game. The game begins on a start screen. Upon pressing a “start” key, it switches to the game state where the actual game runs.

As soon as the maze initializes and displays, a timer is started. The movement FSM controls the player's movement while in sync with the collision FSM which takes care of the player bumping into the walls.

If the player collides with ‘exit’ (a specific colored door which acts as the exit of the maze), the game switches to the win game state where a winning message is displayed.

Throughout the game running process, the timer is continuously checked to initiate the lose game sequence (if it has run out).

If the player runs out of time before he can do so, the game switches to the lose game state where a losing message is displayed.

FSM Implementation:

FSM

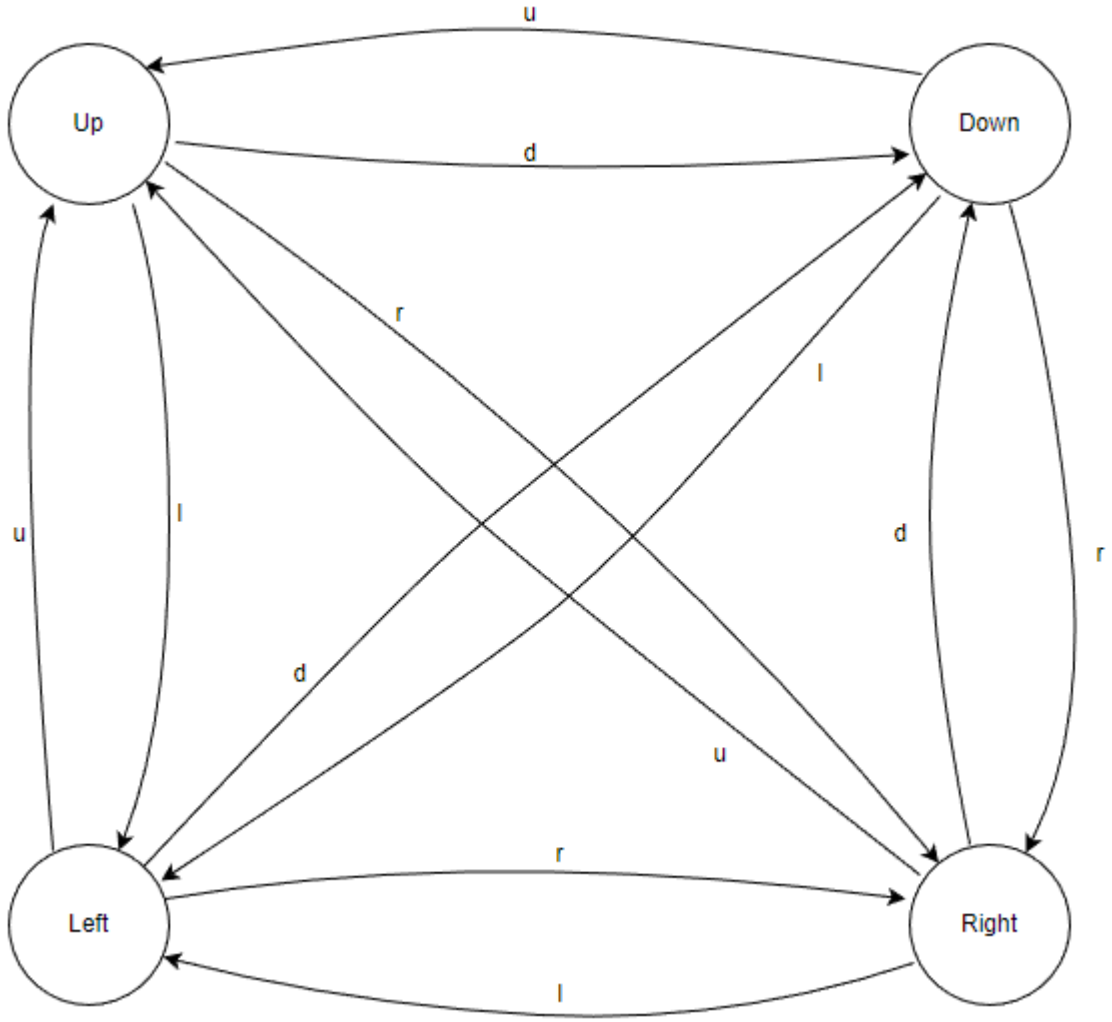
```
module maze(clk, SGP, state, next);
  input clk;
  input [2:0] SGP;
  output [2:0] state, next;
  reg [2:0] state = 2'b0;
  reg [2:0] next = 2'b0;

  parameter start = 3'b0;
  parameter game = 3'b01;
  parameter win = 3'b10;
  parameter lose = 3'b11;
  parameter reset = 3'b100;

  always @(posedge clk)
    begin
    state <= next;
  end
  always @(SGP or state)
      begin
        case (SGP[2:0])
          1'b0: if (state == reset) next = start;
          1'b1: if (state == start) next = game;
          else if (state == win || state == lose) next = start;
        endcase
        case (SGP[1:0])
          2'b10: if (state == game) next = win;
          2'b11: if (state == game) next = lose;
        endcase
      end
endmodule
```

Game Screen:

The “Game Screen” basically integrates multiple modules that displays: the maze screen, a cube(user representation) and their movement with direction and colliSsion.

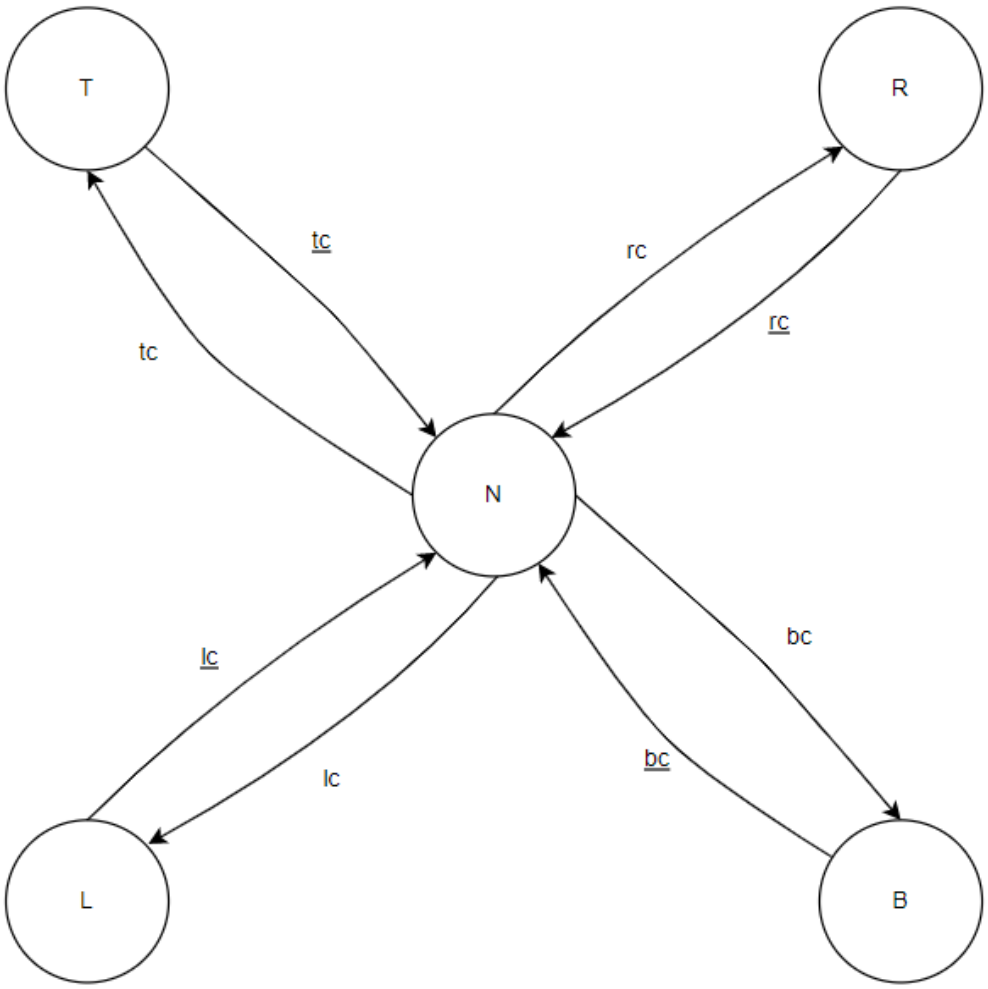
Movement FSM

The movement of a player is controlled through a simple FSM, as follows



Collision FSM

N represents a ‘No Collision State’ in this case. If a Right Collision occurs (rc), the collision FSM moves to a Right Collision State (represented by R). And if the No Right Collision occurs (rc), it moves back to it’s ‘No Collision State’.

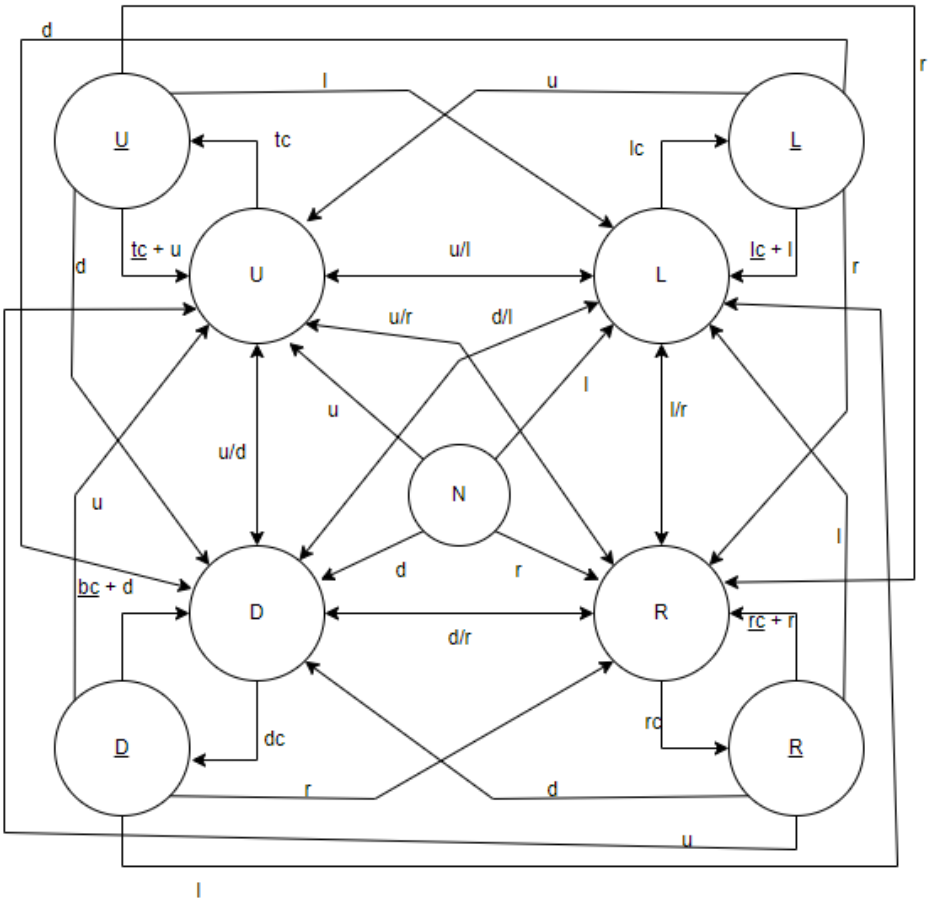


States		Inputs	
N	No Collision		
T	Top Collision	tc	Top Collision
R	Right Collision	rc	Right Collision
B	Bottom Collision	bc	Bottom Collision
L	Left Collision	lc	Left Collision

Movement-Collision FSM

By combining the above to FSM, we reach a complex solution to our problem. The player has 9 different states, explained in the table below. Upon receiving any input (or none at all), the

next state can be seen through the FSM, as well as the truth table. For example, from U (Up Movement), if we take input of Top Collision (tc) and up key (u), we reach the U state (Up Movement Stopped). Therefore, if there is a top collision, it cannot return to U state, unless given the input of tc + u (No Top Collision and Up button).



States		Inputs	
U	Up Movement	u	Up
U	Up Movement Stopped	d	Down
D	Down Movement	r	Right
D	Down Movement Stopped	l	Left
R	Right Movement	tc	Top Collision
R	Right Movement	bc	Bottom Collision

	Stopped		
L	Left Movement	rc	Right Collision
$\underline{L}$	Left Movement Stopped	lc	Left Collision

*underline represents the opposite of that state/input

State Table:

Input	Description	Representation
-	No Input	-
u	Up	00
d	Down	01
l	Left	10
r	Right	11

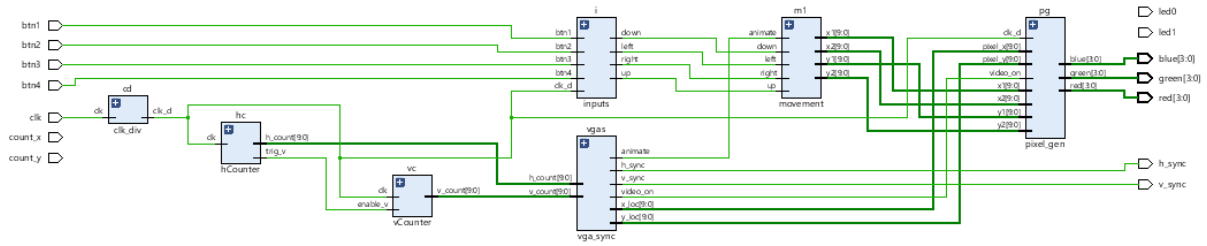
State	Description	Representation
U	Up Movement	000
$\underline{U}$	No Up Movement	001
D	Down Movement	010
$\underline{D}$	No Down Movement	011
L	Left Movement	100
$\underline{L}$	No Left Movement	101
R	Right Movement	110
$\underline{R}$	No Right Movement	111
N	Rest State	-

Current State	Present State			Input		Next State		
	A	**B**	**C**	**x**	**y**	**W**	**X**	**Y**
U	0	0	0	-	-	-	-	-
	0	0	0	0	0	0	0	0

	0	0	0	0	1	0	1	0
	0	0	0	1	0	1	0	0
	0	0	0	1	1	1	1	1
U	0	0	1	-	-	-	-	-
	0	0	1	0	0	0	0	1
	0	0	1	0	1	0	1	0
	0	0	1	1	0	1	0	0
	0	0	1	1	1	1	1	1
D	0	1	0	-	-	-	-	-
	0	1	0	0	0	0	0	0
	0	1	0	0	1	0	1	0
	0	1	0	1	0	1	0	0
	0	1	0	1	1	1	1	1
D	0	1	1	-	-	-	-	-
	0	1	1	0	0	0	0	0
	0	1	1	0	1	0	1	1
	0	1	1	1	0	1	0	0
	0	1	1	1	1	1	1	1
L	1	0	0	-	-	-	-	-
	1	0	0	0	0	0	0	0
	1	0	0	0	1	0	1	0
	1	0	0	1	0	1	0	0
	1	0	0	1	1	1	1	1
L	1	0	1	-	-	-	-	-
	1	0	1	0	0	0	0	0
	1	0	1	0	1	0	1	0
	1	0	1	1	0	1	0	1

	1	0	1	1	1	1	1	1
R	1	1	0	-	-	-	-	-
	1	1	0	0	0	0	0	0
	1	1	0	0	1	0	1	0
	1	1	0	1	0	1	0	0
	1	1	0	1	1	1	1	1
R	1	1	1	-	-	-	-	-
	1	1	1	0	0	0	0	0
	1	1	1	0	1	0	1	0
	1	1	1	1	0	1	0	0
	1	1	1	1	1	1	1	1

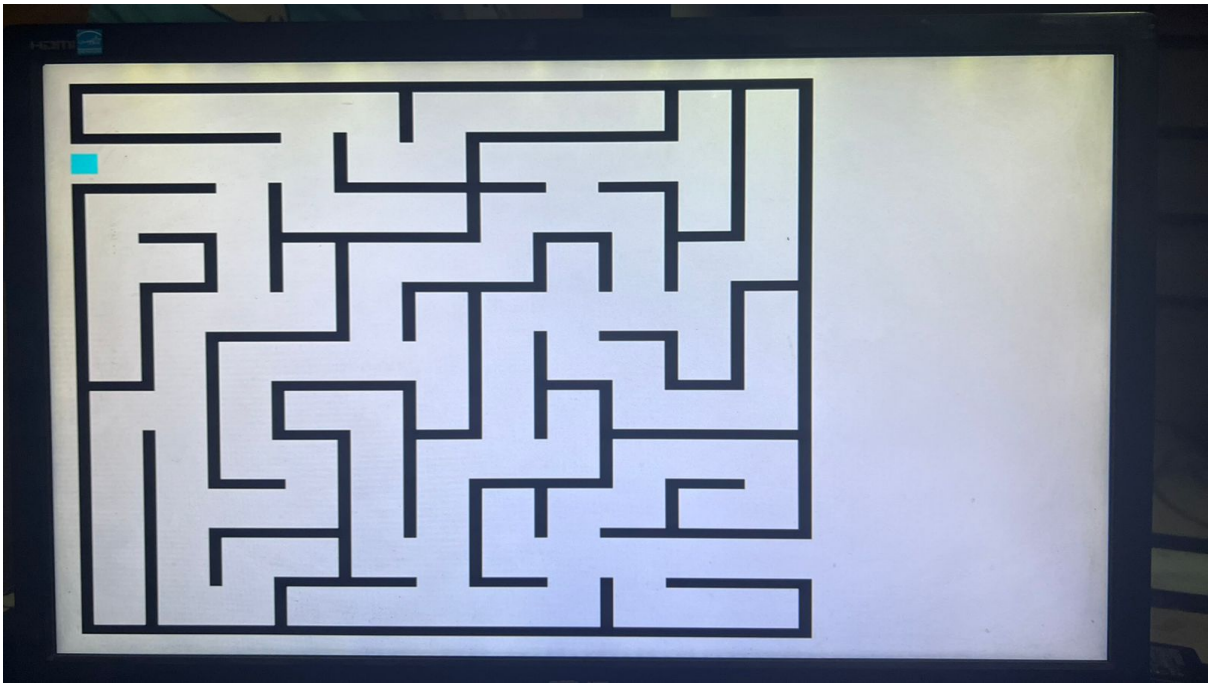
RTL Elaborated Design



Screen Results

Start Screen:

Game Screen:



End Screen

References:

https://github.com/DuaeSameen/DLD-Project