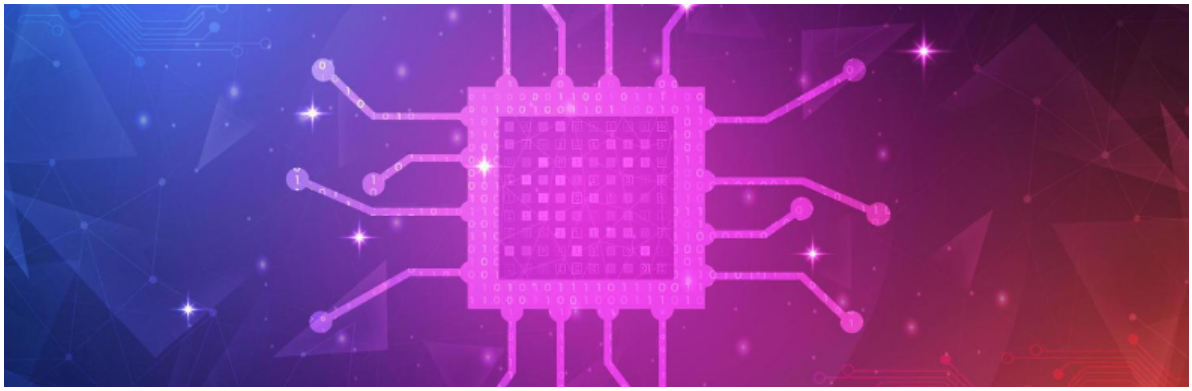

ECEN 260 - Final Project

Video Game

Marcel Pratikto

Instructor: Brother Watson
December 12, 2023

Contents

1 Lab Overview **3**
1.1 Goal . . . 3
1.2 Objectives . . . 3

2 Specifications **4**
2.1 Controls . . . 4
2.2 Parts List . . . 4

3 Schematics **6**

4 Test Plan and Test Results **7**
4.1 Test Plan Procedure . . . 7
4.2 Expected and Observed Results . . . 8

5 Code **9**
5.1 Code for main.c . . . 9

6 Conclusion **27**

List of Tables

List of Figures

1 Basics of the game 4
2 Final project schematic diagram 6

1 Lab Overview

This report is for my final project in ECEN 260 at Brigham Young University–Idaho. The goal of the final project is to assess my knowledge and experience with microprocessors. This will require that I use my expertise to successfully integrate three of the following concepts learned into my project: interrupts, ADC, PWM, timers, UART communication, display, or digital communication. I chose interrupts, display, and digital communication. The objective of my final project is to create a video game that will prove that I have learned how to integrate multiple concepts learned from this class, as per my goal.

1.1 Goal

- Show understanding of:
  - interrupts
  - display
  - digital communication

1.2 Objectives

- Create a video game

2 Specifications

My final project creates a system composed primarily of a micro-controller, a display, and a keypad. The three of them will sync in order to create a video game that you can play, much like the old school pixel based video games such as Pac-Man or space invaders.

Figure 1: Basics of the game

The display will render all the actors in the game, as seen in the image above. The player is the triangle actor, the only one that we can control using the keypad. Refer to 2.1 for the controls. The other actors are the O's and the X's. If the player collided/hit the X's, then game over, and the losing screen displays the words "YOU LOSE...". If the player collided with the O's, then they win. The victory screen displays the words "YOU WIN! ☺". The final version of the game contains multiple X's, one O's, and one player actor. The X's and O's will have random x-positions between 0 to 80. They will also have variable speeds, so that they fall at different rates to make the game more interesting.

2.1 Controls

- 2: UP
- 4: LEFT
- 8: DOWN
- B: RIGHT

2.2 Parts List

- 1x Nucleo32-L476RG development board

- 1x USB cable
- 1x breadboard
- 1x Nokia 5110 LCD display
- 1x 4x4 matrix keypad
- 1x 74c922 keypad encoder IC
- 1x 1 μF capacitor
- 1x 10 μF capacitor
- $\approx$ 30 jumper wires

3 Schematics

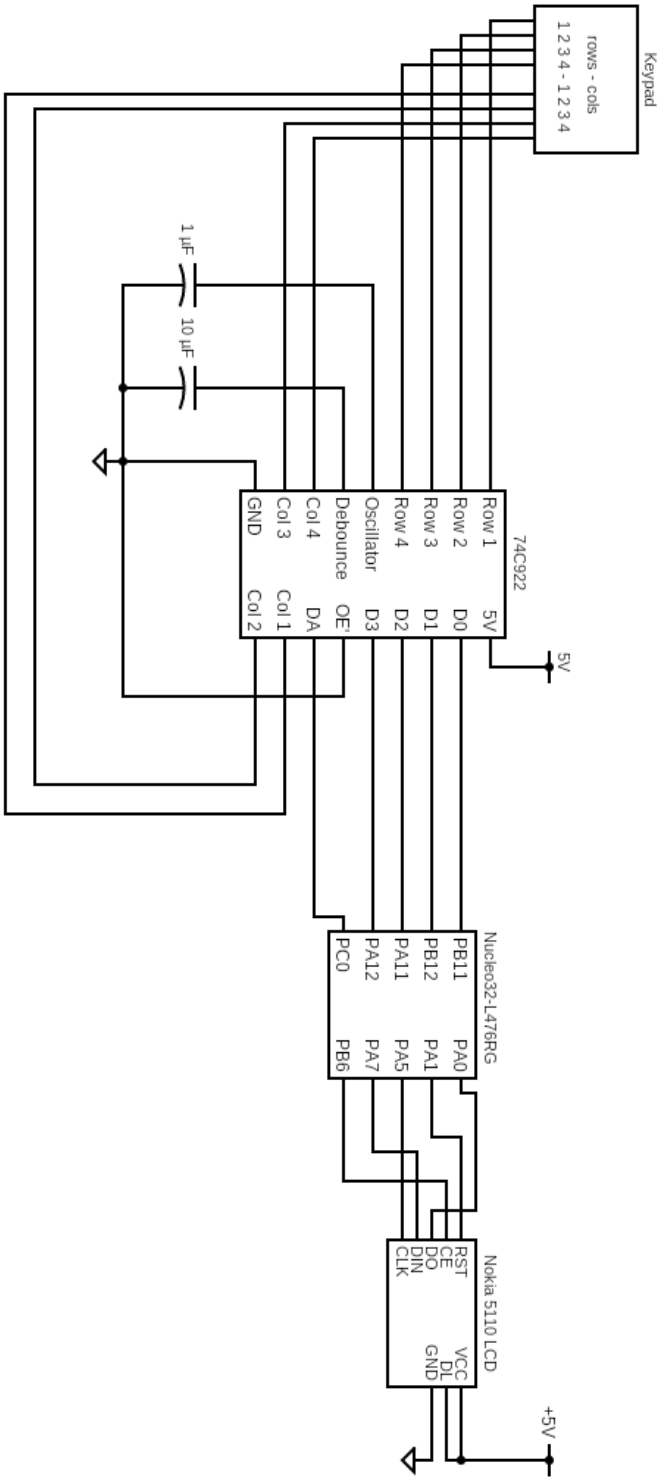

Figure 2: Final project schematic diagram

4 Test Plan and Test Results

I proceeded with the development of this project in parts, and my test plans reflected that. I started with wiring the display and keypad. Once I know that they are working as intended, I moved on to the software side. I had to figure out how to represent position, convert from position to image, add player, add player controls, add other actors, add collision, and add animation.

4.1 Test Plan Procedure

- Test Scenario #1: Wiring
  - Step 1: Wire display
  - Step 2: Display "Hello World!" on the screen
  - Step 3: Wire and program keypad
  - Step 5: Press the letter 'A' and see if it pops up on the screen
- Test Scenario #2: Position and Image
  - Step 1: Create a position coordinate system to represent each pixel on the Nokia display
  - Step 2: Convert pixels from the position system to an IMAGE array that the Nokia display can understand. (Convert from horizontal grouping to vertical grouping)
  - Step 3: Render image on the display
  - Step 4: Confirm that location and image is correct
- Test Scenario #3: Player Controls
  - Step 1: Program keypad to the controls specified in 2.1
  - Step 2: Press 2, player moves up on the display
  - Step 3: Press 4, player moves left on the display
  - Step 4: Press 8, player moves down on the display
  - Step 5: Press B, player moves right on the display
  - Step 6: Press other keys, nothing should happen
- Test Scenario #4: Render Actors and Collision
  - Step 1: Add an O
  - Step 2: Add an X
  - Step 3: Add collision between player and O and X
  - Step 4: Colliding with O sends the player to the victory screen

  - Step 5: Colliding with X sends the player to the losing screen
  - Step 6: Add more X, test if it can collide with player as well

- Test Scenario #5: Animation
  - Step 1: Create animation that moves position of all O's and X's from top of screen to gradually falling
  - Step 2: Leave player under trajectory of O
  - Step 3: Game should move to victory screen once the O moves down to the player's position
  - Step 4: Leave player under trajectory of X
  - Step 5: Game should move to victory screen once the X moves down to the player's position

4.2 Expected and Observed Results

This section should include the the expected and actual results of each test.

- Test Scenario #1: Wiring
  - Expected Result: Displays "A" on the screen when the key "A" is pressed
  - Actual Result: Displays "A" on the screen when the key "A" is pressed
- Test Scenario #2 Position and Image
  - Expected Result: Renders an image on the display that corresponds with the IMAGE array
  - Actual Result: The rendered image needs to be flipped 180 degrees.
- Test Scenario #3: Player Controls
  - Expected Result: Player can move left, right, up, down
  - Actual Result: Player can move left, right, up, down
- Test Scenario #4: Render Actors and Collision
  - Expected Result: Colliding with O wins the game, X loses the game
  - Actual Result: Colliding with O wins the game, X loses the game
- Test Scenario #5: Animation
  - Expected Result: All non-player actors falls down from top of display to bottom at different rates
  - Actual Result: Actors move accordingly, but their collision points didn't follow alongside the image

5 Code

5.1 Code for main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define WIDTH 84
```

```
#define HEIGHT 48
#define NUM_BANKS 6
/* USER CODE END PD */

/* Private macro -------------------------------------------------------
   */
/* USER CODE BEGIN PM */
#define CE_PORT GPIOB // PB6 chip enable (aka slave select)
#define CE_PIN GPIO_PIN_6
#define DC_PORT GPIOA // PA0 data/control
#define DC_PIN GPIO_PIN_0
#define RESET_PORT GPIOA // PA1 reset
#define RESET_PIN GPIO_PIN_1
#define GLCD_WIDTH 84
#define GLCD_HEIGHT 48
#define NUM_BANKS 6
/* USER CODE END PM */

/* Private variables ---------------------------------------------------
   */
SPI_HandleTypeDef hspi1;

/* USER CODE BEGIN PV */
const char font_table[][6] = {
  {0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // space            0
  {0x3E, 0x41, 0x41, 0x7F, 0x40, 0x20}, // 'A'              1
  {0x7F, 0x49, 0x49, 0x76, 0x40, 0x20}, // 'B'              2
  {0x3E, 0x41, 0x41, 0x41, 0x40, 0x20}, // 'C'              3
  {0x7F, 0x41, 0x41, 0x7E, 0x40, 0x20}, // 'D'              4
  {0x3E, 0x49, 0x49, 0x49, 0x40, 0x20}, // 'E'              5
  {0x7F, 0x09, 0x09, 0x09, 0x10, 0x20}, // 'F'              6
  {0x3E, 0x41, 0x49, 0x39, 0x10, 0x20}, // 'G'              7
  {0x7F, 0x08, 0x08, 0x7F, 0x40, 0x20}, // 'H'              8
  {0x41, 0x7F, 0x41, 0x41, 0x40, 0x20}, // 'I'              9
  {0x21, 0x51, 0x7F, 0x21, 0x20, 0x20}, // 'J'              10
  {0x7F, 0x0C, 0x12, 0x21, 0x40, 0x20}, // 'K'              11
  {0x04, 0x7E, 0x45, 0x42, 0x40, 0x20}, // 'L'              12
  {0x7F, 0x02, 0x04, 0x02, 0x7F, 0x20}, // 'M'              13
  {0x7F, 0x04, 0x08, 0x10, 0x7F, 0x20}, // 'N'              14
  {0x3E, 0x41, 0x41, 0x3E, 0x20, 0x20}, // 'O'              15
  {0x7E, 0x11, 0x11, 0x1E, 0x10, 0x20}, // 'P'              16
  {0x3E, 0x49, 0x51, 0x3E, 0x20, 0x20}, // 'Q'              17
  {0x7F, 0x19, 0x29, 0x46, 0x40, 0x20}, // 'R'              18
  {0x46, 0x49, 0x49, 0x71, 0x40, 0x20}, // 'S'              19
  {0x01, 0x01, 0x7F, 0x41, 0x41, 0x20}, // 'T'              20
  {0x3F, 0x40, 0x40, 0x3F, 0x20, 0x20}, // 'U'              21
  {0x7F, 0x20, 0x10, 0x1F, 0x10, 0x20}, // 'V'              22
  {0x7F, 0x20, 0x10, 0x20, 0x7F, 0x20}, // 'W'              23
  {0x41, 0x22, 0x1C, 0x24, 0x42, 0x20}, // 'X'              24
  {0x4F, 0x50, 0x50, 0x7F, 0x40, 0x20}, // 'Y'              25
  {0x71, 0x49, 0x45, 0x47, 0x40, 0x20}, // 'Z'              26
  {0x00, 0x00, 0x40, 0x00, 0x00, 0x00}, // period           27
  {0x00, 0x00, 0x5F, 0x00, 0x00, 0x00}, // !                28
  {0x00, 0x00, 0x7E, 0x81, 0xB5, 0xA1}, // left smiley face     29
```

```
  {0xA1, 0xB5, 0x81, 0x7E, 0x00, 0x00}, // right smiley face      30
};

// IMAGE is like the texture
char IMAGE[HEIGHT][WIDTH];

/* SCREEN is the converted image to an array type that we can display
 * {TOP, 2, 3, 4, 5, BOTTOM} LEFT
 *  ...
 * {TOP, 2, 3, 4, 5, BOTTOM} RIGHT
 *
 * {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
 * {0x01, 0x00, 0x00, 0x00, 0x00, 0x80},
 * {0x01, 0x00, 0x00, 0x00, 0x00, 0x80},
 * {0x01, 0x00, 0x00, 0x00, 0x00, 0x80}
 */
// 84x48 pixels, 48 separated to six 8-bit pixel lines
char SCREEN[WIDTH][NUM_BANKS];

// Player: stores position, model
struct Actor{
  int x;
  int y;
  int model_width;
  int model_height;
  int speed;
  char* name;
  char* model;
};
char player_model[] = {
  "   x   "
  "  xxx  "
  " xxxxx "
  "xxxxxxx"
};
char O_model[] = {
  "  xxx  "
  " xxxxx "
  "xxxxxxx"
  " xxxxx "
  "  xxx  "
};
char X_model[] = {
  "xx   xx"
  " xx xx "
  "  xxx  "
  " xx xx "
  "xx   xx"
};

// Keeps track of the key that was pressed
char KEY = 0;

// Keeps track if the user wins or lose
```

```
145 char WIN = 0;
146 char LOSE = 0;
147
148 /* USER CODE END PV */
149
150 /* Private function prototypes -----------------------------------------------
    */
151 void SystemClock_Config(void);
152 static void MX_GPIO_Init(void);
153 static void MX_SPI1_Init(void);
154 /* USER CODE BEGIN PFP */
155 void SPI_write(unsigned char data);
156 void GLCD_data_write(unsigned char data);
157 void GLCD_command_write(unsigned char data);
158 void GLCD_init(void);
159 void GLCD_setCursor(unsigned char x, unsigned char y);
160 void GLCD_clear(void);
161 void GLCD_putchar(int font_table_row);
162 unsigned char keypad_decode();
163 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
164 /* USER CODE END PFP */
165
166 /* Private user code ---------------------------------------------------------
    */
167 /* USER CODE BEGIN 0 */
168 /*
169  * clears the image so that it is filled with empty spaces
170  */
171 void clearImage(){
172   for (int y = 0; y < HEIGHT; ++y)
173   {
174     for (int x = 0; x < WIDTH; ++x)
175     {
176       IMAGE[y][x] = ' ';
177     }
178   }
179 }
180
181 /*
182  * Takes data of all actors, such as their position and texture,
183  * then converts it to an IMAGE array
184  * that represents each pixel on the display/screen.
185  * PARAMETER 1: array of address of actors
186  * PARAMETER 2: number of actors
187  */
188 void convertToImage(struct Actor* actors[], int num_actors)
189 {
190   for (int i = 0; i < num_actors; ++i)
191   {
192     // create IMAGE
193     int model_pos = 0;
194     for (int y = 0; y < HEIGHT; ++y)
195     {
196       for (int x = 0; x < WIDTH; ++x)
```

```
197       {
198         if (x >= actors[i]->x && x < actors[i]->x+actors[i]->model_width && y
    >= actors[i]->y && y < actors[i]->y+actors[i]->model_height)
199         {
200           if (model_pos < actors[i]->model_height * actors[i]->model_width)
201           {
202             IMAGE[y][x] = actors[i]->model[model_pos];
203             ++model_pos;
204           }
205         }
206       }
207     }
208   }
209 }
210
211 /*
212  * Converts image to format that the screen can accept.
213  */
214 void convertToScreen()
215 {
216   // convert IMAGE to SCREEN
217   for (int x = 0; x < WIDTH; ++x)
218   {
219         int bank = 0;
220         int pixel = 0;
221         char binary = 0b00000000;
222         for (int y = 0; y < HEIGHT; ++y)
223         {
224             if (IMAGE[y][x] == 'x')
225             {
226                 binary |= (1 << pixel);
227             }
228             ++pixel;
229             if (pixel == 8)
230             {
231                 SCREEN[x][bank] = binary;
232                 ++bank;
233                 pixel = 0;
234                 binary = 0b00000000;
235             }
236         }
237   }
238 }
239
240 /*
241  * draw image on screen by column from left to right
242  * works by filling six vertical banks (1 bank = 8 pixels vertically)
243  */
244 void drawScreen()
245 {
246   for (int y=0; y<6; ++y)
247   {
248     for (int x=0; x<84; ++x)
249     {
```

```
250         GLCD_data_write(SCREEN[x][y]);
251       }
252     }
253     //HAL_Delay(5000);
254     //GLCD_clear();
255 }
256 
257 /*
258  * Player controls.
259  * PARAMETER 1: pointer to player
260  */
261 void controls(struct Actor* player)
262 {
263   // Keypad
264     switch(KEY)
265     {
266     case 2:    // UP
267       player->y -= 2;
268       KEY = 0;
269       break;
270     case 8:    // DOWN
271       player->y += 2;
272       KEY = 0;
273       break;
274     case 4:    // LEFT
275       player->x -= 5;
276       KEY = 0;
277       break;
278     case 11:     // RIGHT
279       player->x += 5;
280       KEY = 0;
281       break;
282     default:
283       KEY = 0;
284       break;
285     }
286 }
287 
288 /*
289  * Check if a collision happened between player and other Actors
290  * PARAMETER 1: pointer to player
291  * PARAMETER 2: array of Actors
292  * RETURN:     int collision_type
293  *           0: nothing
294  *           1: win
295  *           2: lose
296  */
297 int checkCollisions(struct Actor* player, struct Actor* actors[], int
      sizeOfActors)
298 {
299   int player_left = player->x;
300   //int player_right = player->x + player->model_width;
301   int player_top = player->y;
302   //int player_bottom = player->y + player->model_height;
```

```
303   for (int i = 0; i < sizeOfActors; ++i)
304   {
305     if (strcmp(player->name, actors[i]->name) != 0)
306     {
307       int actor_left = actors[i]->x;
308       //int actor_right = actors[i]->x + actors[i]->model_width;
309       int actor_top = actors[i]->y;
310       //int actor_bottom = actors[i]->y + actors[i]->model_height;
311 
312       if (
313           (abs(actor_left - player_left) <= player->model_width) &&
314           (abs(actor_top - player_top) <= player->model_height)
315         )
316       {
317         if (strcmp(actors[i]->name, "O") == 0)
318           return 1;
319         if (strcmp(actors[i]->name, "X") == 0)
320           return 2;
321       }
322     }
323   }
324   return 0;
325 }
326 
327 /*
328  * Random number generation.
329  * PARAMETER 1: lowest number you want generated
330  * PARAMETER 2: highest number you want generated
331  * RETURN: int in range [lower, upper]
332  */
333 int randomINT(int lower, int upper)
334 {
335   return (rand() % (upper + 1)) + lower;
336 }
337 
338 /*
339  * Animate actors.
340  * PARAMETER 1: pointer to player
341  * PARAMETER 2: array of addresses of actors
342  * PARAMETER 3: number of actors
343  */
344 void animateActors(struct Actor* player, struct Actor* actors[], int
       sizeOfActors)
345 {
346   int defaultX = -10;
347   int defaultY = -10;
348   int minSpeed = 1;
349   int maxSpeed = 3;
350   int minX = 0;
351   int maxX = 80;
352   int minY = -5;
353   int maxY = 55;
354   for (int i = 0; i < sizeOfActors; ++i)
355   {
```

```
356     if (strcmp(player->name, actors[i]->name) != 0)
357     {
358       // if location is at the starting default of (-10,-10)
359       // draw it at a random location at the top of the screen
360       // with random x position in range [0,80]
361       // with random speed in range [1,5]
362       if ((actors[i]->x == defaultX) && (actors[i]->y == defaultY))
363       {
364         actors[i]->y = minY;
365         actors[i]->x = randomINT(minX,maxX);
366         actors[i]->speed = randomINT(minSpeed,maxSpeed);
367       }
368       // if actor goes over the bottom of the screen,
369       // redraw on top of screen with a different x position
370       // with a different speed
371       if (actors[i]->y > maxY)
372       {
373         actors[i]->y = minY;
374         actors[i]->x = randomINT(minX,maxX);
375         actors[i]->speed = randomINT(minSpeed,maxSpeed);
376       }
377       else
378       {
379         // make sure that an object's speed is never zero
380         // so that it keeps falling
381         if (actors[i]->speed == 0)
382           actors[i]->speed = randomINT(minSpeed,maxSpeed);
383         actors[i]->y += actors[i]->speed;
384       }
385     }
386   }
387 }
388 /* USER CODE END 0 */
389
390 /**
391   * @brief  The application entry point.
392   * @retval int
393   */
394 int main(void)
395 {
396   /* USER CODE BEGIN 1 */
397
398   /* USER CODE END 1 */
399
400   /* MCU Configuration--------------------------------------------------------
      */
401
402   /* Reset of all peripherals, Initializes the Flash interface and the Systick
      . */
403   HAL_Init();
404
405   /* USER CODE BEGIN Init */
406
407   /* USER CODE END Init */
```

```
408
409  /* Configure the system clock */
410  SystemClock_Config();
411
412  /* USER CODE BEGIN SysInit */
413
414  /* USER CODE END SysInit */
415
416  /* Initialize all configured peripherals */
417  MX_GPIO_Init();
418  MX_SPI1_Init();
419  /* USER CODE BEGIN 2 */
420  GLCD_init(); // initialize the screen
421  GLCD_clear(); // clear the screen
422
423  // WIN screen
424  int win_screen[] = {
425      25,15,21,0,23,9,14,28,29,30
426  };
427
428  // LOSE screen
429  int lose_screen[] = {
430      25,15,21,0,12,15,19,5,27,27,27
431  };
432
433  // player initialization
434  struct Actor player;
435  player.model = player_model;
436  player.model_width = 7;
437  player.model_height = 4;
438  player.x = 84/2 - player.model_width/2;
439  player.y = 48-player.model_height;
440  player.name = "player";
441  player.speed = 0; // we move the player, so its speed is 0
442
443  // 3 X, 1 O, 1 player
444  int num_X = 2;
445  int num_O = 1;
446  int num_player = 1;
447  int num_Actors = num_X + num_O + num_player;
448  // array of the address of each actor
449  struct Actor* actors[num_Actors];
450  for (int i = 0; i < num_Actors; ++i)
451  {
452    if (i < num_X)
453    {
454    actors[i] = (struct Actor*)malloc(sizeof(struct Actor));
455
456    actors[i]->model = X_model;
457    actors[i]->model_width = 7;
458    actors[i]->model_height = 5;
459    actors[i]->x = -10;
460    actors[i]->y = -10;
461    actors[i]->name = "X";
```

```
462     actors[i]->speed = 0;
463     }
464     else if (i < num_X + num_O)
465     {
466     // O initialization
467     struct Actor O;
468     O.model = O_model;
469     O.model_width = 7;
470     O.model_height = 5;
471     O.x = -10;
472     O.y = -10;
473     O.name = "O";
474     O.speed = 0;
475
476     actors[i] = &O;
477     }
478     else
479     {
480       actors[i] = &player;
481     }
482   }
483
484   // use current time as seed for random number generator
485   time_t t;
486   srand((unsigned)time(&t));
487   /* USER CODE END 2 */
488
489   /* Infinite loop */
490   /* USER CODE BEGIN WHILE */
491   while (1)
492   {
493     /* USER CODE END WHILE */
494
495     /* USER CODE BEGIN 3 */
496     if (WIN)
497     {
498       GLCD_clear();
499       int size = sizeof(win_screen)/sizeof(win_screen[0]);
500       for (int i=0; i < size; ++i)
501       {
502         GLCD_putchar(win_screen[i]);
503         if (i < size - 2)
504           HAL_Delay(500);
505       }
506       HAL_Delay(5000);
507     }
508     else if (LOSE)
509     {
510       GLCD_clear();
511       int size = sizeof(lose_screen)/sizeof(lose_screen[0]);
512       for (int i=0; i < size; ++i)
513       {
514         GLCD_putchar(lose_screen[i]);
515         HAL_Delay(500);
```

```
516       }
517       HAL_Delay(5000);
518     }
519     else
520     {
521       // create image with actors combined
522       // then renders the screen
523       clearImage();
524       convertToImage(actors, num_Actors);
525       convertToScreen();
526       drawScreen();
527 
528       // Handles controls
529       controls(&player);
530 
531       // Handles collisions
532       int collision_type = checkCollisions(&player, actors, num_Actors);
533       // 0 : nothing
534       // 1 : win
535       // 2 : lose
536       if (collision_type == 1)
537         WIN = 1;
538       else if (collision_type == 2)
539         LOSE = 1;
540 
541       // moves position of actors
542       animateActors(&player, actors, num_Actors);
543       HAL_Delay(300);
544     }
545   }
546   /* USER CODE END 3 */
547 }
548 
549 /**
550   * @brief System Clock Configuration
551   * @retval None
552   */
553 void SystemClock_Config(void)
554 {
555   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
556   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
557 
558   /** Configure the main internal regulator output voltage
559   */
560   if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
561   {
562     Error_Handler();
563   }
564 
565   /** Initializes the RCC Oscillators according to the specified parameters
566   * in the RCC_OscInitTypeDef structure.
567   */
568   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
569   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
```

```
570   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
571   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
572   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
573   RCC_OscInitStruct.PLL.PLLM = 1;
574   RCC_OscInitStruct.PLL.PLLN = 10;
575   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
576   RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
577   RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
578   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
579   {
580     Error_Handler();
581   }
582
583   /** Initializes the CPU, AHB and APB buses clocks
584   */
585   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
586                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
587   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
588   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
589   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
590   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
591
592   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
593   {
594     Error_Handler();
595   }
596 }
597
598 /**
599   * @brief SPI1 Initialization Function
600   * @param None
601   * @retval None
602   */
603 static void MX_SPI1_Init(void)
604 {
605
606   /* USER CODE BEGIN SPI1_Init 0 */
607
608   /* USER CODE END SPI1_Init 0 */
609
610   /* USER CODE BEGIN SPI1_Init 1 */
611
612   /* USER CODE END SPI1_Init 1 */
613   /* SPI1 parameter configuration*/
614   hspi1.Instance = SPI1;
615   hspi1.Init.Mode = SPI_MODE_MASTER;
616   hspi1.Init.Direction = SPI_DIRECTION_2LINES;
617   hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
618   hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
619   hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
620   hspi1.Init.NSS = SPI_NSS_SOFT;
621   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
622   hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
623   hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
```

```
624   hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
625   hspi1.Init.CRCPolynomial = 7;
626   hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
627   hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
628   if (HAL_SPI_Init(&hspi1) != HAL_OK)
629   {
630     Error_Handler();
631   }
632   /* USER CODE BEGIN SPI1_Init 2 */
633
634   /* USER CODE END SPI1_Init 2 */
635
636 }
637
638 /**
639   * @brief GPIO Initialization Function
640   * @param None
641   * @retval None
642   */
643 static void MX_GPIO_Init(void)
644 {
645   GPIO_InitTypeDef GPIO_InitStruct = {0};
646 /* USER CODE BEGIN MX_GPIO_Init_1 */
647 /* USER CODE END MX_GPIO_Init_1 */
648
649   /* GPIO Ports Clock Enable */
650   __HAL_RCC_GPIOC_CLK_ENABLE();
651   __HAL_RCC_GPIOH_CLK_ENABLE();
652   __HAL_RCC_GPIOA_CLK_ENABLE();
653   __HAL_RCC_GPIOB_CLK_ENABLE();
654
655   /*Configure GPIO pin Output Level */
656   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0|GPIO_PIN_1, GPIO_PIN_RESET);
657
658   /*Configure GPIO pin Output Level */
659   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
660
661   /*Configure GPIO pin : DA_Pin */
662   GPIO_InitStruct.Pin = DA_Pin;
663   GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
664   GPIO_InitStruct.Pull = GPIO_NOPULL;
665   HAL_GPIO_Init(DA_GPIO_Port, &GPIO_InitStruct);
666
667   /*Configure GPIO pins : PA0 PA1 */
668   GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
669   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
670   GPIO_InitStruct.Pull = GPIO_NOPULL;
671   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
672   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
673
674   /*Configure GPIO pin : USART_RX_Pin */
675   GPIO_InitStruct.Pin = USART_RX_Pin;
676   GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
677   GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
678   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
679   GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
680   HAL_GPIO_Init(USART_RX_GPIO_Port, &GPIO_InitStruct);
681
682   /*Configure GPIO pins : D0_Pin D1_Pin */
683   GPIO_InitStruct.Pin = D0_Pin|D1_Pin;
684   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
685   GPIO_InitStruct.Pull = GPIO_NOPULL;
686   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
687
688   /*Configure GPIO pins : D2_Pin D3_Pin */
689   GPIO_InitStruct.Pin = D2_Pin|D3_Pin;
690   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
691   GPIO_InitStruct.Pull = GPIO_NOPULL;
692   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
693
694   /*Configure GPIO pin : PB6 */
695   GPIO_InitStruct.Pin = GPIO_PIN_6;
696   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
697   GPIO_InitStruct.Pull = GPIO_NOPULL;
698   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
699   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
700
701   /* EXTI interrupt init*/
702   HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
703   HAL_NVIC_EnableIRQ(EXTI0_IRQn);
704
705 /* USER CODE BEGIN MX_GPIO_Init_2 */
706 /* USER CODE END MX_GPIO_Init_2 */
707 }
708
709 /* USER CODE BEGIN 4 */
710 void SPI_write(unsigned char data)
711 {
712   // Chip Enable (low is asserted)
713   HAL_GPIO_WritePin(CE_PORT, CE_PIN, GPIO_PIN_RESET);
714   // Send data over SPI1
715   HAL_SPI_Transmit(&hspi1, (uint8_t*) &data, 1, HAL_MAX_DELAY);
716   // Chip Disable
717   HAL_GPIO_WritePin(CE_PORT, CE_PIN, GPIO_PIN_SET);
718 }
719
720 void GLCD_data_write(unsigned char data)
721 {
722   // Switch to "data" mode (D/C pin high)
723   HAL_GPIO_WritePin(DC_PORT, DC_PIN, GPIO_PIN_SET);
724   // Send data over SPI
725   SPI_write(data);
726 }
727
728 void GLCD_command_write(unsigned char data)
729 {
730   // Switch to "command" mode (D/C pin low)
731   HAL_GPIO_WritePin(DC_PORT, DC_PIN, GPIO_PIN_RESET);
```

```
732   // Send data over SPI
733   SPI_write(data);
734 }
735
736 void GLCD_init(void)
737 {
738   // Keep CE high when not transmitting
739   HAL_GPIO_WritePin(CE_PORT, CE_PIN, GPIO_PIN_SET);
740   // Reset the screen (low pulse - down & up)
741   HAL_GPIO_WritePin(RESET_PORT, RESET_PIN, GPIO_PIN_RESET);
742   HAL_GPIO_WritePin(RESET_PORT, RESET_PIN, GPIO_PIN_SET);
743   // Configure the screen (according to the datasheet)
744   GLCD_command_write(0x21); // enter extended command mode
745   GLCD_command_write(0xB0); // set LCD Vop for contrast (this may be adjusted)
746   GLCD_command_write(0x04); // set temp coefficient
747   GLCD_command_write(0x15); // set LCD bias mode (this may be adjusted)
748   GLCD_command_write(0x20); // return to normal command mode
749   GLCD_command_write(0x0C); // set display mode normal
750 }
751
752 void GLCD_setCursor(unsigned char x, unsigned char y)
753 {
754   GLCD_command_write(0x80 | x); // column
755   GLCD_command_write(0x40 | y); // bank
756 }
757
758 void GLCD_clear(void)
759 {
760   int i;
761   for(i = 0; i < (GLCD_WIDTH * NUM_BANKS); i++)
762   {
763     GLCD_data_write(0x00); // write zeros
764   }
765   GLCD_setCursor(0,0); // return cursor to top left
766 }
767
768 void GLCD_putchar(int font_table_row)
769 {
770   // go through each value in order to print the character
771   int i;
772   for (i=0; i<6; i++){
773     GLCD_data_write(font_table[font_table_row][i]);
774   }
775 }
776
777 // decode the keypad according to what we need each keys to be
778 unsigned char keypad_decode(){
779   unsigned char key = 0x0;
780   unsigned char data = 0b0000;
781
782   // read the data pins and combine into the 4-bit value: D3_D2_D1_D0
783   if (HAL_GPIO_ReadPin(D0_GPIO_Port, D0_Pin))
784     data |= bit(0);
785   if (HAL_GPIO_ReadPin(D1_GPIO_Port, D1_Pin))
```

```
786     data |= bit(1);
787   if (HAL_GPIO_ReadPin(D2_GPIO_Port, D2_Pin))
788     data |= bit(2);
789   if (HAL_GPIO_ReadPin(D3_GPIO_Port, D3_Pin))
790     data |= bit(3);
791 
792   // The key encoder gives the following "data" values:
793   // 0 1 2 3
794   // 4 5 6 7
795   // 8 9 A B
796   // C D E F
797 
798   // The following switch statement re-maps it to these "key" names:
799   // 1 2 3 A
800   // 4 5 6 B
801   // 7 8 9 C
802   // E 0 F D, where E is * and F is #
803 
804   switch(data){
805     case 0x0: key = 0x1; break; // fill out the missing key values (?) in this
      switch statement
806     case 0x1: key = 0x2; break;
807     case 0x2: key = 0x3; break;
808     case 0x3: key = 0xA; break;
809     case 0x4: key = 0x4; break;
810     case 0x5: key = 0x5; break;
811     case 0x6: key = 0x6; break;
812     case 0x7: key = 0xB; break;
813     case 0x8: key = 0x7; break;
814     case 0x9: key = 0x8; break;
815     case 0xA: key = 0x9; break;
816     case 0xB: key = 0xC; break;
817     case 0xC: key = 0xE; break;
818     case 0xD: key = 0x0; break;
819     case 0xE: key = 0xF; break;
820     case 0xF: key = 0xD; break;
821   }
822 
823   return key;
824 }
825 
826 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
827 {
828   if(GPIO_Pin == DA_Pin)
829   {
830     // Read data
831     unsigned char key = keypad_decode();  // determine which key was pressed
832     // RETURNS KEY
833     switch (key){
834     case 0x1:
835       KEY = 1;
836       break;
837     case 0x2:
838       KEY = 2;
```

```
839         break;
840       case 0x3:
841         KEY = 3;
842         break;
843       case 0x4:
844         KEY = 4;
845         break;
846       case 0x5:
847         KEY = 5;
848         break;
849       case 0x6:
850         KEY = 6;
851         break;
852       case 0x7:
853         KEY = 7;
854         break;
855       case 0x8:
856         KEY = 8;
857         break;
858       case 0x9:
859         KEY = 9;
860         break;
861       case 0xA:
862         KEY = 10;
863         break;
864       case 0xB:
865         KEY = 11;
866         break;
867       case 0xC:
868         KEY = 12;
869         break;
870       case 0xD:
871         KEY = 13;
872         break;
873       case 0xE:
874         KEY = 14;
875         break;
876       case 0xF:
877         KEY = 15;
878         break;
879       }
880   }
881 }
882 /* USER CODE END 4 */
883
884 /**
885   * @brief  This function is executed in case of error occurrence.
886   * @retval None
887   */
888 void Error_Handler(void)
889 {
890   /* USER CODE BEGIN Error_Handler_Debug */
891   /* User can add his own implementation to report the HAL error return state
      */
```

```
892   __disable_irq();
893   while (1)
894   {
895   }
896   /* USER CODE END Error_Handler_Debug */
897 }
898
899 #ifdef  USE_FULL_ASSERT
900 /**
901   * @brief  Reports the name of the source file and the source line number
902   *         where the assert_param error has occurred.
903   * @param  file: pointer to the source file name
904   * @param  line: assert_param error line source number
905   * @retval None
906   */
907 void assert_failed(uint8_t *file, uint32_t line)
908 {
909   /* USER CODE BEGIN 6 */
910   /* User can add his own implementation to report the file name and line
      number,
911     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
      */
912   /* USER CODE END 6 */
913 }
914 #endif /* USE_FULL_ASSERT */
```

6 Conclusion

This project has deepened my knowledge on display using SPI communication, controls, interrupts. Using SPI communication, I was able to send data to a Nokia 5110 LCD display from the micro-controller. The same micro-controller also receives data from a 4x4 matrix keypad. In order to make game play smoother, my code does not pause and wait for user input. Instead, it responds only when a user have pressed a key. It is this balance between display, compute, and input that is the basis of modern devices today. I have also learned a great deal of programming a game engine, albeit a simple one. My game engine is able to render pixel-based textures on the screen. It can also calculate collision between the player and other actors in the game.

I had the biggest problem trying to figure out image rendering, physics, and gameplay. Although this does not relate to this class, it solidified my programming abilities, especially my c programming, graphics programming, physics programming, and gameplay programming. As a computer science major, these skills might be helpful in the long run should I choose to get a career in game development.

References