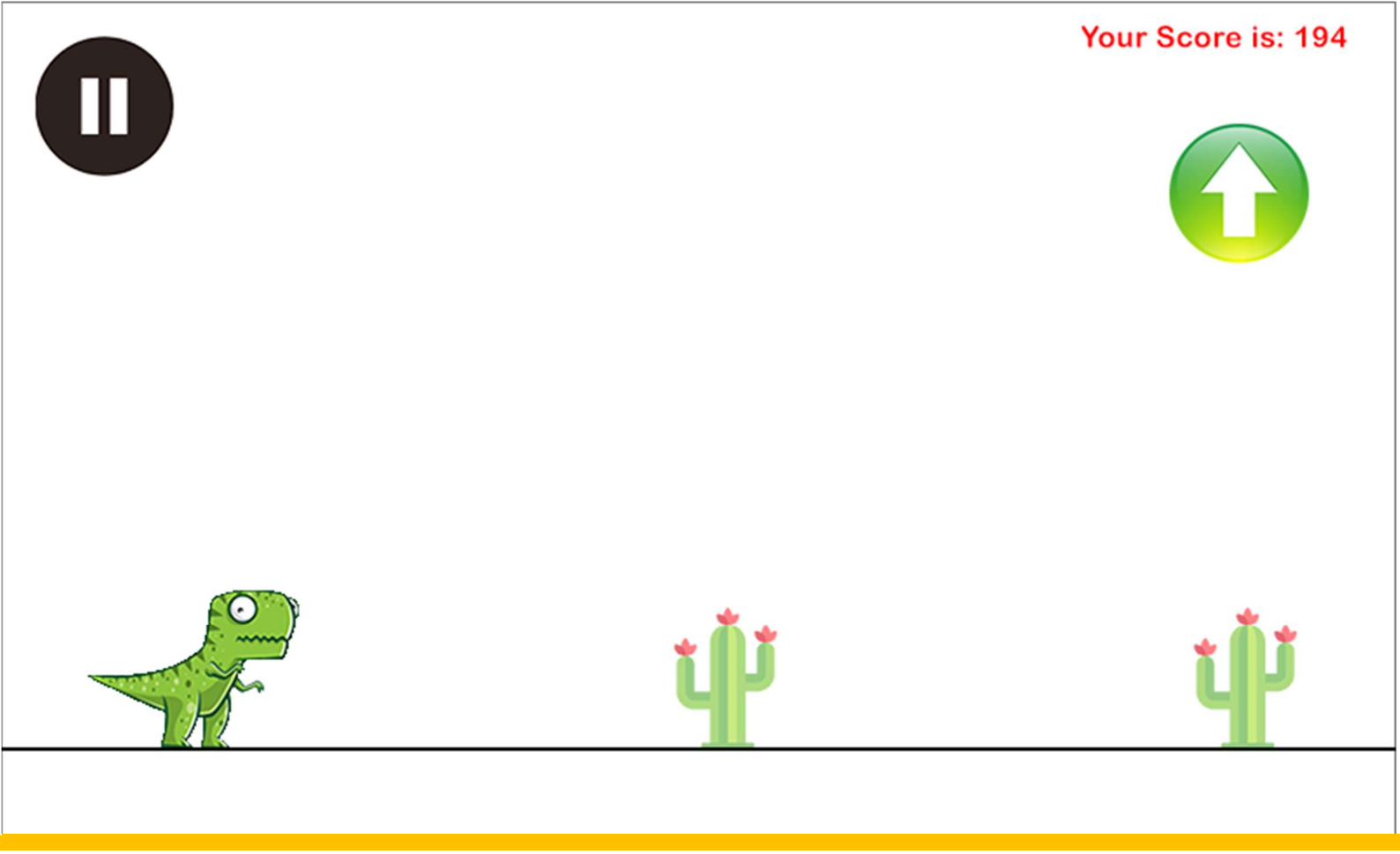

Google T-Rex Game

BTE5057 – Software Projekte

Report

Author/s:	David Hein Hoyer; Lukas Roth
Group:	-
Date:	2023-01-23
Tutor:	Elham Firouzi
Version:	1.0

Berner Fachhochschule

Technik und Informatik
Elektrotechnik und Informationstechnologie

Table of contents

List of figures

1 Introduction

This report presents, the design and implementation of a software project in C. This project was done for the module “BTE-5057 Software Projekte”. The idea was to design a Project on the LEGUAN Board of the BFH [1]. It has sensors like acceleration or temperature. More on it has an 850x450 pixel touch LCD Display. For this Project, only the touch LCD Display was used.
The T-Rex Google game, also known as the Chrome Dino game, is a popular and widely recognized mini-game that can be found on the Google Chrome browser. The game is an endless runner that is activated when the internet connection is lost, and the Chrome browser is unable to load a web page.

Figure 1: T-Rex Google game

The Project idea is based on this game. The goal was to create a similar game on the LEGUAN Board, programmed in C. The LCD is used to display the game and to get the touch inputs for the game events. The report covers the game’s architecture and design, as well as the programming techniques and libraries used to achieve as smooth and as responsive gameplay as possible. Additionally, the report discusses the challenges faced during the development process and the solutions implemented to overcome them. The result is a fully functional game that showcases the capabilities of the LEGUAN Board and the programming skills of the development team.

2 Project Management

2.1 Versions Management & Backup

The whole project management is done using GitHub [5]. With help of GitHub the version-management of the Project is easily done because every change that was uploaded to GitHub[5] can be looked back in the GitHub history. More on GitHub serves as a perfect cloud backup of the whole Project.

2.2 Time tracking

To keep track of the Project, a excel [6] sheet "timestamps.xlsx" is created. After every work session the programmer filled in the excel sheet with what he has done, and the amount of time used for it. Like that the total time spent on the project can be tracked easily. More on the milestones of the Project can also be kept track of easily. The whole timestamps.xlsx file can be seen in the attachment.

2.3 Time management

In a big project like this, it is important, to have some sort of time management. In this project it was done by using an excel [6] sheet. First the biggest milestones were fixed. For example, to draw an image on the LCD or to move an object in a given speed. Next the date, this part of the program should work. And the last part is the actual date on which this part of the program worked.

Table 1: Time management table

MILESTONES	FINISH DATE (should)	FINISH DATE (is)
Pixel Array	20.10.2022	25.10.2022
BMP on LCD	30.10.2022	28.10.2022
Touch input	15.11.2022	05.11.2022
Shift BMP	30.11.2022	13.12.2022
Game Movement	15.12.2022	04.01.2023
collision detection	31.12.2022	24.01.2023
Final adjustments	20.01.2023	24.01.2023
Finish	25.01.2022	24.01.2023
Report writing	27.01.2022	25.01.2023

It can be seen, that in the beginning, the project was completely on track with the time plan. Every part before the collision detection went nicely but the collision detection itself was harder than imagined and took a little longer than anticipated. But because the team had two members, one could finish the collision detection and the other one could do the final adjustments, so in the end the project is finished on the given date.

2.4 Documentation

The team decided to use doxygen [4] to create a simple overview of the program. This program creates a website where the data Structure and all the functions of the program are displayed in a simple and compact form. To look at the data structure, go to "TRex/doxygen/html" and open the file "annotated". To see an overview of the file management and all the functions click on "Files". Now the file structure can be seen and if clicked on the name of a header file, all the functions, data structures and enumerations in the chosen file are displayed.

3 Requirements analysis

3.1 Requirement sheet

The first step in a software analysis is, to define a requirement sheet. In this case, it was easy and not a lot of work, because the google T-Rex game is basically the requirement sheet. The only difference is that in this project there is only one kind of obstacle. Apart from that the games should work the same way.

3.2 Bottom-up Analysis

To start of the Project, a bottom-up analysis was done. The first thing that must be done, was the conversion of a BMP file to a pixel array, so the BMP could be drawn on the LCD. This was solved by a simple C program, which read out the color values of the BMP and wrote them into a text file, so the data could be copied into the resources.h file of the TRex program. The next upper level is the drawing, positioning and moving of the game objects. This is all done in the object.c file. The last level are the game functions, which generate the time specific movement and the collision detection of two objects. These functions can all be found in the game.c file of the TRex program. These levels can clearly be seen in the paragraph “File-management” where a visualization of the file-structure of the program can be found.

4 Design and Architecture

4.1 Program flow and Game States

The architecture of the design is shown in a flow chart in Figure 2. This basic program flow basically represents the code in the main file. The game's code is structured in a modular way, with different functions dedicated to different tasks.

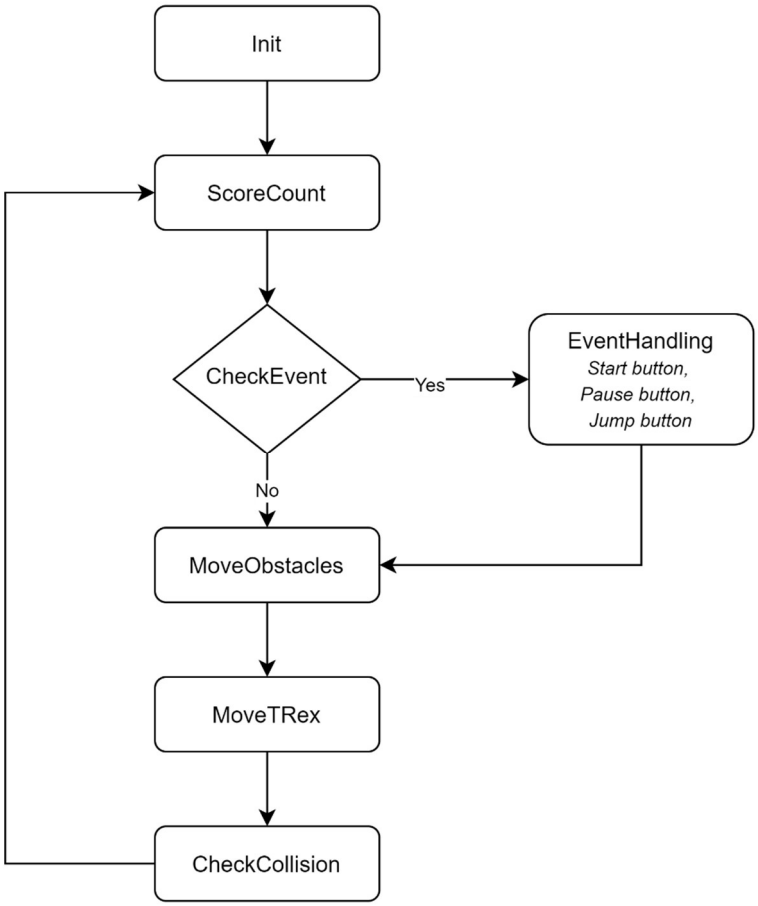

Figure 2: Basic program flow

The main functions of the game include:

- **Init:** This function initializes the Leguan board and sets up the game environment. It is responsible for initializing the LCD screen, touch input, and other necessary peripherals.
- **ScoreCount:** This function is responsible for counting and displaying the player's score on the LCD screen.
- **CheckEvent:** This function checks for touch input events on the LCD screen and returns a Boolean value indicating if an event has occurred.
- **EventHandling:** This function handles the touch input event and updates the game state accordingly.
- **MoveObstacles:** This function is responsible for moving the obstacles on the screen and updating their positions.
- **MoveTrex:** This function is responsible for moving the player's character (T-Rex) on the screen and updating its position based on the touch input events.
- **CheckCollision:** This function checks for collisions between the T-Rex and the obstacles on the screen.
- **ScoreCount:** This function is responsible for keeping track of the player's score and updating it on the screen.

The game's code follows a loop structure, where the functions are called in sequence, starting with the Init function, then the ScoreCount function, CheckEvent, EventHandling or MoveObstacles, MoveTrex and Check Collision and finally the ScoreCount function again. This loop continues indefinitely, providing an endless gaming experience for the player.

In terms of design patterns, the game makes use of the State pattern to handle the different states of the game such as the menu, the play state, pause state and the game over state.

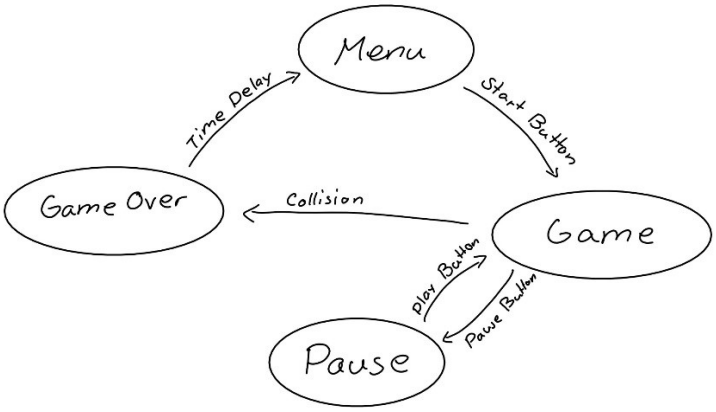

Figure 3: Game state

The game's architecture is designed to be simple and easy to understand, making it easy to add new features or make changes to the existing code. The modular structure of the code also makes it easy to test and debug individual functions.

4.2 File-management

The File structure is designed as simple as possible; a visualization can be seen in Figure 4. In the file recources.h are all the arrays with the pixel data of the BMP files. Next the object.c file generates a struct for each bmp. Within this structure all important information of an object can be found. More on the object.c file contains all functions, that are used to draw a BMP or shift a BMP for a given number of pixels in each direction. This nis done by using the LibLeguan library of the BFH. If a function like this is called, it also overwrites the object data, like the position or the visibility Flag. More information about the object struct can be found in paragraph 5.2.
The game.c file contains all time independent function as well as the incoming touch event functions and the object collision functions. In the end, in the main program only functions of the game.c file can be used to control all the objects. The data management is all done internally by the game.c and object.c files.BMP implementation

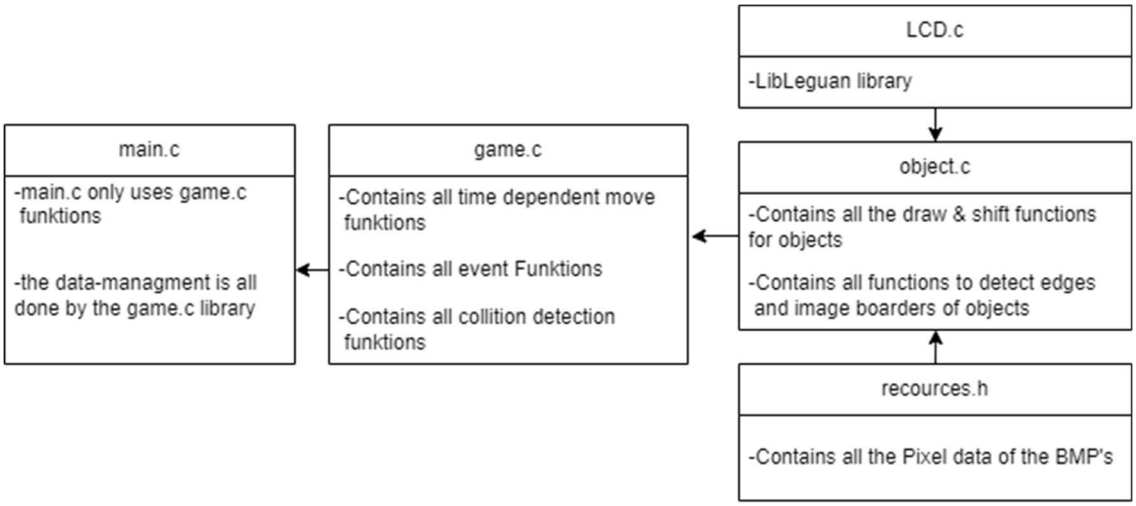

Figure 4: File-management

5 implementation and testing

In this chapter smaller functions/problems as well as their code pieces are discussed
In addition, the testing of the individual functions is explained and demonstrated.

5.1 Event Handling

For this game the event handling only considers the touch input. So, there is only 1 event type that can happen. The event is stored in a struct which is shown in Figure 5.

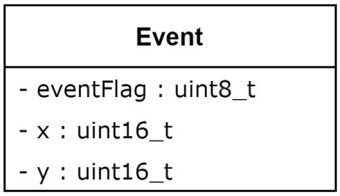

Figure 5: Event Struct

As can be seen in Figure 2 at the beginning of every loop the CheckEvent function is called. this function checks if the display has been touched and sets the flag accordingly.
Id an event happened the function stores the x and y position of the point where the LCD was touched and returns the event struct.

With the **CheckEventBmp(pauseButtonBmp, event)** function we can check if the event occurred on the object or not. But because the bmp structs can't be accessed from the main file every object needs a separate function which will call the CheckEventBmp function for the corresponding object.

5.2 BMP implementation

The BMP struct is the heart of every game object. Inside this struct every information needed in the program can be found. The first four variables are the **X** and **Y** position as well as **with** and **height** of the object. The X and Y values get updated every time the object is moved to a different position. The with and height variables are used by the draw function, to know how big the draw area must be on the LCD Display.

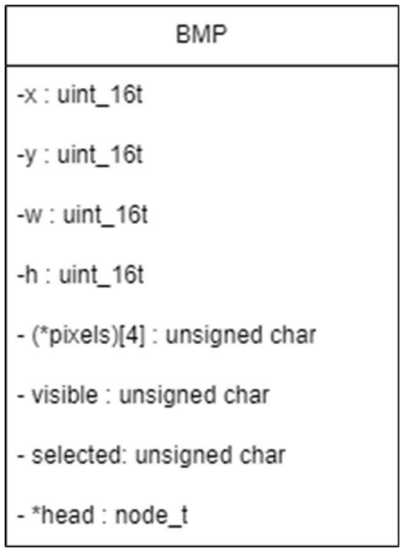

Figure 6: BMP Struct

The fifth variable in the BMP struct is a pointer to the **pixel array** in which the image data is saved. Next there are two flags. The first is a **visibility flag** for the obstacles because not always all obstacles are displayed. The second flag is a **selection flag** for the T-Rex. This is needed to select the BMP for the T-Rex because the user can select between two different T-Rex images.
The last variable is again a pointer variable. This pointer points to the **head** of a linked list. This linked list contains all boarder pixels of the BMP.

5.2.1 PNG to pixelData array

To be able to draw an image on the LCD display, there are a couple of steps needed before. The first problem is the PNG file format. It is hard to just read out the color values of all pixels. To solve this problem, the PNG file is first converted into a BMP file format. The BMP file format allows simple reading because all color values are just listed behind each other. To get the image data into an array, a small C program is written in Qt creator [2] which reads out the color values of the BMP file and writes them into a text file. The format in the text file can be seen in Figure 7 below.

```
Bitmap Info BMC:\Users\lukir\OneDrive\Documents
Width in pixel:       59
Height in pixel:      80
Total Pixel:          4720
Bits per pixel:       32

unsigned char PictureData [4720][4] = {
{252, 2, 1, 6}, {252, 2, 1, 0}, {252, 2, 1, 0},
```

Figure 7: Data format in text file

The image color data is already in the syntax of an C array, so it can just be copied into the recources.c file of the program. The array is a two-dimensional array. In the first dimension, all the image pixels are listed. In the second dimension the color values of each pixel are listed. It is important to note, that not only the tree color values are read out but also the transparent value of each pixel. This transparent value is later used to detect the edges of the image.

5.2.2 Draw BMP

At the beginning of the project the Bmp files were written using the LCD_Pixel function provided by the lcd.h library. The problem here is that the LCD is written via SPI. The LCD_Pixel function has 3 SPI commands. One to set the draw area, one to enable the draw mode and the last one to send the pixel color. If only a few pixels are written this is not so bad, but since we are writing bmp files with more than 10'000 pixels, the time used for sending the additional commands becomes noticeable.

Therefore, we created the function DrawBmp, which defines the draw area at the beginning, the draw mode enabled and then all pixels are described. Thus, we save 2/3 of the commands.

A disadvantage is that the background of the bmp files, which is transparent, is always overwritten with the background color. Because we can only write whole rectangles to the LCD display and not specific pixels. This means that if two bmp files overlap with transparent and non-transparent, the bmp file that will be drawn later will be overwritten with the background of the new bmp file. There are ways around this, for example by making a buffer for the whole LCD display. This could then be sent to the LCD display with every screen update. Unfortunately, this is almost not possible because on the one hand the RAM is not enough to create such a large array and on the other hand the whole display would have to be written which would take too much time.

The DrawBmp function loops trough every pixel of the bmp file. If the pixel is transparent de background color is sent, else the color data of the bmp pixel is sent.

The LCD works with BGR_565 But the color data of the bmp files are BGRA_8888. So, we need to convert them first before sending them. In an earlier stage of the project this was done in the DrawBmp function each time for every pixel. Late this was moved to the initialization of the program. So, the conversion only needs to be done one time. This was done to speed up the process of writing the bmp to the LCD.

While working with the lcd.c library a mistake was noticed. In the LCD_Rect function, the draw area had to be adjusted because it was initialized incorrectly. Thus in the width always one bit too much was written which were then missing at the lower end of the rectangle. Here is the corrected version:

```
result_t LCD_Rect(uint16_t x, uint16_t y, uint16_t width, uint16_t height) {
    /* Set working region */
    R_TRY(LCD_SetDrawArea(x, y, x + width - 1, y + height - 1));
    LCD_EnableDrawMode();

    /* Fill working region with color */
    for (uint32_t i = 0; i < width * height; i++)
        LCD_Set(&m_foreground_color);

    return RESULT_SUCCESS;
}
```

5.2.3 Move and Shift BMP

To move an object to a different position on the LCD is not a problem, it can just be drawn new at the given location. But the Problem is that the LCD does not clear a pixel itself. So, if an object is moved, the old pixels must be deleted. This is done by drawing the pixel again in background color. Visualization of BMP move:

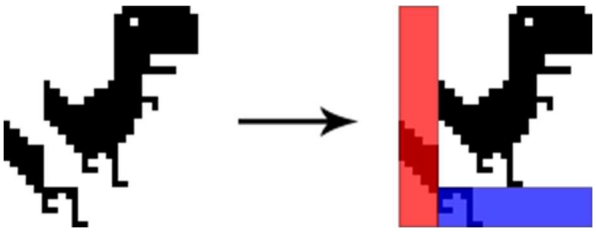

Figure 8: Move BMP visualization

When a BMP is moved to a different location, there are two options. Option one is, that the move is bigger than the dimensions of the BMP, so the old BMP is completely overwritten with background colour. The second option is shown in the Image above. First the new BMP is drawn, then the remaining image blocks are overwritten in background colour. By not always overwriting the whole bmp file with the background colour in the old position and redrawing it in the new one, flickering is prevented when the images are constantly moving. In addition, this method saves a lot of processing time.

5.3 Time specific object Movement

To get the game running, all the objects must move at a given velocity, which is time dependent. Therefore, the Hal library is used to get the counter value. Now every time a move function is called, the first thing done, is the calling of the counter value. This value is saved and compared to the last saved value. If the difference is bigger than a given value (10ms in Figure 9), the object is moved. This means, an object can only move every 10ms. If the difference is less than 10ms, the move is not done. Below a Nassi Schneidermann diagram can be seen.

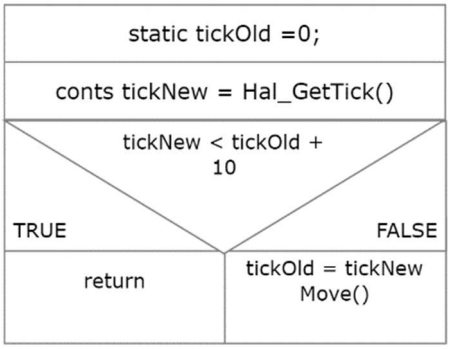

Figure 9: Nassi-Schneiderman for time specific movement

5.3.1 Obstacle movement

For the obstacle movement, the above explained method is used. All the visible objects are moved for 5 pixels every 15ms. The important thing about the obstacle move function is, that it creates an object in a random distance from the object before and deletes an object of it gets to the end of the screen. This is solved by using a for loop to go through every obstacle. If an obstacle is visible, first the position of the objects gets tested and if it is at the left edge of the screen the obstacle gets deleted. If the obstacle is not at the left edge, it gets moved to the left.
If an obstacle is not visible, the distance from the last obstacle to the right edge of the screen gets checked as it can be seen in the code below. It is important to note that with this if statement the distance between the obstacles is random. The obstacles have a base distance of 300 pixels (LCD_WIDTH - 400 - 2*75) and a random offset between 0 and 150. This means that the obstacles can have a distance of 300, 375 or 450 pixels.
Every time before the random distance is generated, the random function is given a see by calling **srand()**. The given seed is calculated out of the current object number and the System Tick of the Hal library. This seed is really important to get good random numbers out of the **rand()** function.

```
//--- give random funktion a seed
srand(j*20000/(HAL_GetTick()%1000 +1));

//--- if last obstacle moved 300, 375 or 450 pixels (random distance)
if(obstacleBmp[j].x < LCD_WIDTH - (400 + ((rand()%3-1) *75))){
   //--- create new obstacle
   DrawBmp(&obstacleBmp[i],
           LCD_WIDTH - obstacleBmp[i].w,
           LCD_HEIGHT - GROUND_HEIGHT - obstacleBmp[i].h);
}
```

5.3.2 T-Rex jump

For the jumping motion of the T-Rex, the same method is used for time dependency. For this function a global variable is used for the jumping direction of the T-Rex. If it has the value of zero, the T-Rex is not jumping. If the jump button gets pressed, the **InitTrexJump()** function is called, which sets the jumping direction to 1 which means it is jumping up. Now the T-Rex gets moved the same way the obstacles get moved, the only difference is that the higher the Y position of the T-Rex is, the slower it gets moved. This is done by a couple of if statements that control the Y position of the T-Rex. If it reached the maximum Y value, the jumping direction gets inverted and the T-Rex is moved down, the same way as it was moved up.

5.4 Border

To determine a collision of objects more accurately than if only the images overlap, the border of the bmp objects must be detected. The border is there where the pixels are changing from transparent to non-transparent. At the beginning the border was detected by looping through the whole bmp file and checking if a pixel has a neighbor which is transparent. If this was the case this pixel was added to a linked list which can be accessed through the bmp struct.

Figure 10: T-Rex Google Border 1

The result of the first border function can be seen in **Fehler! Verweisquelle konnte nicht gefunden werden.**. Because we wanted to improve the collision detection (Which will be explained in chapter

5.5) only the edges of the border should remain. So, a function was created which would take in the head of the linked border pixel list and remove all pixel nodes which were not an edge which led to the following border.

Figure 11: T-Rex Google Border 2

Now the next important problem was to sort the linked list so one could create a line from each edge pixel to the next. A SortBorderEdges function was created, but it did not work as well as was expected. And the number of edge pixels was not pleasing either. So, a different approach was taken.

The edges of the bmp files are defined directly in photoshop.
They are given the value: Red = 1, Green = 2, Blue = 252, Transparent = 0
This value was taken as it is unlikely that a file will contain other pixels with these color values. Now in the code these marked border edge pixels can easily be detected.

Figure 12: TRex Google Border 3

But a big challenge is that the pixels are saved in the correct order in the linked list, because it is important for the collision detection that the order is correct, otherwise no lines can be formed along the edge of the objects. Just sorting the pixel list so that the next pixel always has the smallest distance is not going to work as can be seen in Figure 12 the nearest pixel is not always the next pixel.

To achieve the final result the function `node_t *GetBoarder (bmp_t bmp)` is created. First the function loops trough the image till a non-transparent pixel is found. From there on it uses the **Moore-Neighborhood-Algorithm**. The basic idea of this algorithm is to scan the neighbour pixels of a pixel an (in this case) check if they are transparent or not. As soon a non-transparent pixel is detected go to this pixel and start again.

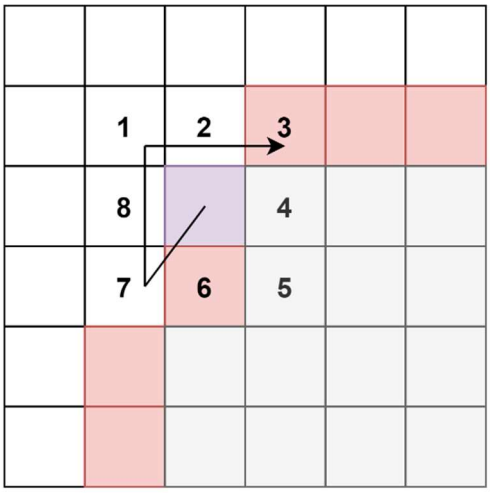

Figure 13: Moore Neighborhood Algorithm

It is important that the direction of the scan is always the same, in this case clockwise.

If the border pixel is found the starting position for the next search is the cell on the opposite side plus one. In the example in Figure 13 a border was found in cell 3. So, the next search will start in the cell number 8. Each pixel which is detected by this algorithm if checked if it's a border marked pixel. If it is, then it's added to the border linked list. The head of the linked list is returned by the function and added to the pointer in the bmp struct.

5.5 Collision Detection

The rough border detection is to check if only the border of the image of the T-Rex and the obstacles collide. But with this border detection there is often the case that the two objects are not really colliding and only some transparent pixels collided. But it is a good first check if a better collision detection even is necessary. So, the first step of the collision detection is to check if the border images collide. If not, the function already returns false, and no further analysis must be done.
First the specific collision detection was done by comparing each pixel of both objects and check if they have the same position. But this resulted in way too many queries. This was done by using the border which is displayed in figure 11.
As a result of this problem the new borders were created. Now the specific collision detection takes the border of the object as liens and compares if any of the lines intersect.

The **CheckLineIntersection** function first determines the orientation of the 4 points (2 points = 1 line) for 4 different combinations. It then checks for general and special cases to determine if the two lines intersect. The general case is when the orientation of first two combinations is different and the orientation of the second two combinations is different. If this is true, the function returns true, indicating that the lines intersect. The special cases are when the orientation of one of the combinations is collinear, and one of the points lies on the other line segment. To check if a point lies on the segment, it uses the **onSegment** function. If none of these cases are true, the function returns false, indicating that the lines do not intersect.

This function was pulled from geeksforgeeks [3].

6 Discussion

In the beginning of this project the T-Rex google game was analysed and the goals for this game were set and the first tasks were assigned. First we needed to gather information about Bitmap files and how an image can be drawn on a LCD display. The overall theory was easy to understand but still we ran into some problems with the colour coding of the LCD display. But after a team session of analysing this problem, it got solved and the first image was on the LCD.
After that point, the actual game programming started. We started to move objects over the display and make the T-Rex jump. But then we ran into the biggest problem in the Project. It was the detection of the collision of two objects. There were some ideas first, but they turned out to be to slow and making the game unplayable. So, some research was done on how a collision detection was done in general. We learned that it normally is done by checking Line intersection as described in paragraph 5.5.
This method was implemented in the program and the last big problem was solved. After that, only small adjustments were made to make the game more enjoyable and visually appealing. And in the end to project was finished on time and worked as we wanted it to.

7 Conclusions

In conclusion, we learned a lot with this project. We learned about image file formats and how images are drawn on LCD displays. More on we learned about what influence hardware has on a program. For example, that our game was not running as fast as it should because of the SPI interface and the LCD_Pixel function (described in paragraph 5.2.2).
Lastly we learned an interesting method to detect boarders in an image with the moor-edge finding algorithm and also the method of line intersection to check for a collision of two objects.
Over all we are happy about how the project turned out and about how much we learned through it.

8 Literaturverzeichnis

[1] T. Kluter, M. Balmer und M. Baour, „LEGUAN, English, Wiki,“ BFH-EIT, Biel/Bienne, Switzerland, 9 11 2021. [Online]. Available: https://leguan.ti.bfh.ch/. [Zugriff am 10 12 2021].

[2] Qt Creator, „QT,“ Qt Group, [Online]. Available: https://www.qt.io/?hsLang=en. [Zugriff am 27 10 2022].

[3] GeeksforGeeks, "check-if-two-given-line-segments-intersect," GeeksforGeeks, 13 07 2022. [Online]. Available: https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/. [Accessed 23 01 2023].

[4] Dimitri van Heesch, „doxygen release 1.9.6,“ 01 2023. [Online]. Available: https://github.com/doxygen/doxygen/releases/tag/Release_1_9_6. [Zugriff am 25 01 2023].

[5] T. Preston-Werner, C. Wanstrath und P. Hyett, „GitHub,“ 02 2008. [Online]. Available: https://github.com/.

[6] C. Simonyi, „Excel 2019,“ Mikrosoft, 24 09 2018. [Online].