Programming Project - 30010 Spring - DTU Space

Cosmic Broadside Battle - a game

Name	Student ID	Group	Student Signature
Alexander W. Aakersø	s223998	15	AAAlexander
Georg B. Dyvad	s224038	15	Georg
Frederik Ø. Larsen	s224032	15	FØL

Date: 23/06/2023

Left to right: Alexander, Georg, Frederik.

Contents

1 Abstract

This project focuses on the design and development of a game-application executed on the ARM Cortex M4 CPU of an STM32Nucleo microcontroller. The application is written in C and utilizes GitHub for code management. Structurally, the application is designed to be modular, separated into three different layers of abstraction; the Application layer, the API layer, and the Hardware abstraction layer. This has proved useful for a multitude of reasons, one of which being the easy delegation of tasks among group members. The game is designed with mainly a cooperative multiplayer game mode in mind, using both PuTTY and the STM32Nucleo peripherals for display & input, however, a simpler single-player mode is also available. This report provides an overview of our design process, the final product, and our takeaways from the project as a whole.

2 Resumé

Denne rapport er skrevet som dokumentation på det afsluttende projekt i et 3 ugers kursus på DTU Space; 30010 - Programmerings Projekt. I dette endelige projekt er der lavet en applikation, der skal eksekveres på en ARM Cortex M4 CPU lokaliseret på en STM32Nucleo mikrocontroller. Applikationen skal udgøre et spil, hvortil de periferere enheder på STM32Nucleoen benyttes. Denne rapport går i dybde med strukturen af programmet, vores opstillede krav, designvalg, projekt-planen og det endelige produkt. Ligeledes er der inkluderet en bruger manual, der fortæller, hvordan spillet skal opstilles. Til slut i rapporten reflekteres der på projektet.

3 Introduction

This report presents the development and implementation of an application designed to run on the ARM Cortex M4 CPU on the STM32Nucleo microcontroller. The application is written using the coding language C and developed using the code-development program STM32CubeIDE. To collaborate on-, store-, and save the code, GitHub was utilized.

The objective of this application is to create an enticing game. Our game can be played on both a computer-screen (using serial communication with PuTTY) and on the LCD display from the Nucleo at the same time. The game allows input from two sources: the computer keyboard and a joystick native to the course (30010). The game as a whole makes use of a multitude of the Nucleo's peripherals including timers, LCD, flash memory, UART, ADC, RGB/LED, and the Buzzer.

The game was designed with a cooperative multiplayer experience in mind, however single-player is also available. The application employs a modular design approach, having a three-layered structure separated into the application layer, the API-layer, and the Hardware Abstraction Layer (HAL).

Throughout this report we will explore these layers along with our choices of design, and the functionality of our application. Furthermore, we will evaluate the final product as well as the project in its entirety. It is our hope that by the end of this report readers will have gained an understanding of our development process and of the architecture of our application.

4 Project Specifications & Requirements

This section presents and discusses the specifications and requirements for our project, set in collaboration between our group and our professor/project Counselor, José M.G. Merayo.

4.1 Documentation Requirements

Unlike the Product Specifications, in this sub-section we will not discuss nor explain the presented requirements. It should be noticed that all of these requirements however have been met.

1. The length of the report must at the very least by 4500 words of which it is expected that each group member contributes 1500.
2. The report must contain the following chapters: Front page, Abstract(English), Resume(Danish), Introduction, Project specifications and requirements, Project Structure, Project Verification, Project Plan, User's manual, Conclusion, references and Appendix.
3. The Appendix must contain all source code written as well as a journal of the exercises completed during week one.
4. The application layer must be documented using flow-charts.
5. the API- and HAL-layers must be documented using block-diagrams.

4.2 Product Specifications

In this section we will firstly present the product specification requirements and then explain our choice of specifications.

Product Specification Requirements & Choice of specifications

Highlighted points in the following lists have been included in our product.

The product must (obligatory) include ALL of the following:

Asteroids/planets with gravity effect	**Game levels**
Number of lives on LCD/PuTTY	**Help screen**
Number of points on LCD/PuTTY	**Menu**
Power-ups	**Boss key**

The product must include MORE than TWO of the following:

Animated figure instead of bullet	**Multiple Bullets**
Game controlled through PuTTY	**Score/Highscore**
Two-players	Docking spaceships
Random delta-angle	

The product must finally include MORE than TWO the following:

Game controlled by 30010-Joystick	Full-game on the LCD
Game controlled by digital Joystick	**Show info on RGB/LED**
Lorentz force movement of bullet/ball	**Sound from buzzer**
Use of onboard accelerometer	

4.3 Implementation of Specifications

In this section we explain how and in what sense we have included the chosen specifications.

Asteroids/planets with gravity effect

Our game implements gravity between two types of entities; standard bullets and asteroids. Even though it would not be difficult for the gravity from the asteroids to affect more types of entities we chose to exclude this, fearing a more chaotic playing experience. The current gravity between entities are calculated as follows:

$$deltaX = x_{dir} \cdot \frac{G \cdot massObj}{dist^2}$$

where x_{dir} is either -1 or 1 dependent on the wished direction of the gravitational pull. 'massObj' is the mass of our asteroid. 'dist' is the manhattan distance between the two colliding entities. Gravity will only be applied if dist < 20. The same is true for the gravity in y-direction.

Game levels with increasing speed controlled by the timer/irq

Our game implements a dynamic difficulty setting controlled by the time spent in-game. This rewards players of higher skill as maximizing the amount of points collected over time becomes very important. Another option was to make the difficulty controlled by the players accumulated score. We however felt that this punishes good players who might accumulate points faster than others. It should also be noted, that a higher difficulty means increasing the spawn rate of enemies and the speed of the enemies. This includes both asteroids and spacecrafts.

Power-Ups

In our game the player can collect power-ups by shooting special asteroids and enemy spaceships that are marked by being special colours. The player can only

store one power-up at any time and if a power-up is collected while another is stored, the stored power-up will be discarded. This rewards continuously using your power-ups. There are a total of three different power-ups;

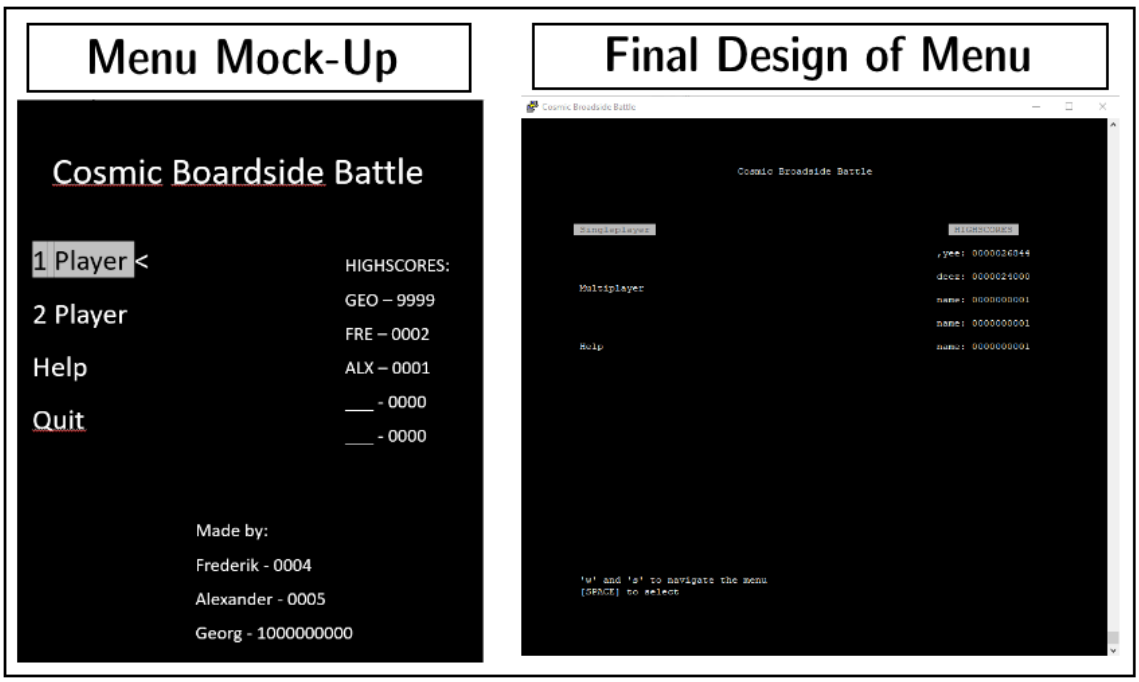
1. Green: Medpack; restore all of the players (3) health points
2. Blue : Shield; Grant the player an extra hitpoint up to a maximum of (4) total.
3. Red : Mega Bullet; Shoot a single very large bullet in a straight vertical upwards direction. This bullet penetrates enemy asteroids and space ships. This bullet can never damage the player.

Number of Lives on LCD/PuTTY & Number of points on LCD/PuTTY

Our game implements showcasing the Number of Lives as a part of our Game-UI, which is displayed on PuTTY. With a maximum of four Lives (when an extra life has been granted trough power-ups) we display the life-total using four squares, where three of these are filled red and the last is filled blue to signal it being an extra life.

Score is also displayed on PuTTY as part of our Game-UI and can be seen as a line of text, consisting of 10 chars. This makes us able to display scores up to nearly 10 billion, which is plenty, seeing as the score can never become greater than 0xFAA2B57F $\approx$ 4.2 billion. We already feel this score is impossible to reach (will take approximately 56 days of straight playing assuming lowest difficulty the whole time). If the score however reaches this amount it will saturate instead of overflow.

Menu



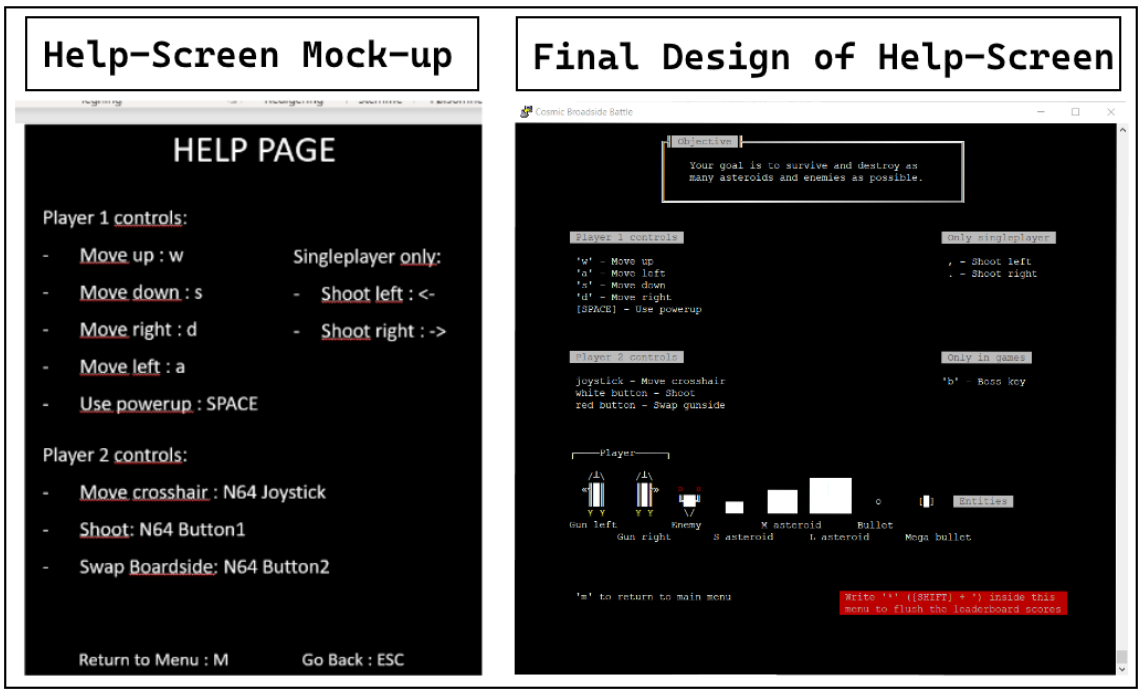
Comparison of our initial mock-up sketch of the menu and the final menu design

The Menu screen is implemented using PuTTY. This screen is the first game you see when the game is initialized and it is here you can chose to start a game of either

the singleplayer or multiplayer edition. The Help-screen can be accessed from this menu as well.

Help Screen

The Help-screen is implemented using PuTTY. This screen can be accessed from the menu and while playing the game. The help-screen contains a guide to the input-controls and also displays the different types of entities you might encounter while playing the game. We have chosen NOT to include a detailed guide in this menu, as the report contains an user manual with that exact information. While in the help menu you have the option to clear all highscores. To do this you need to input '*' which can only be done while holding down [SHIFT] while pressing another button on the keyboard, making the likelihood of deleting all highscores by accident much smaller.



Comparison between our initial mock-up sketch and the final design

It should also be included that when you visit the Help-screen while playing the game an option to return to the game by pressing [ESC] will be shown in the bottom left corner next to " 'm' to return to main menu".

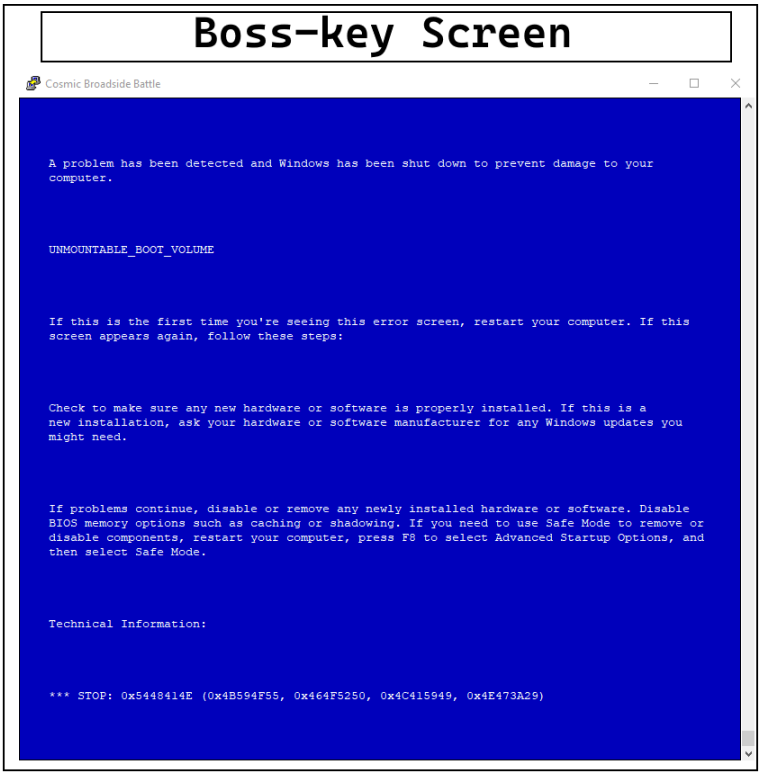
Game controlled through PuTTY

Using serial communication, our game is able to receive and transmit data from and to PuTTY. This means both that we're able to display our game using PuTTY and that we can take inputs from the PC-keyboard. Taking advantage of this fact, we're able to control our player using 'W' 'A' 'S' 'D' [SPACE] ',' and '.'. These inputs are also used for navigating between our menu, help-screen, game-screen and boss-screen and it is at last also used for storing a 4-digit name when registering a new highscore.

Multiple Bullets

Not only is the player allowed to shoot bullets, enemy spaceships can as well. In total our game can handle 16 bullets at one time. To make sure we wont go over that limit (and to balance the game) we've added a bullet cool-down to our player, limiting the spam-ability of the bullets. Enemy spaceships have no such cooldown but are gated by randomness. Enemy spaceships have a 1% chance each clockcycle to shoot a bullet. This roughly averages to every enemy spaceship shooting three bullets in a lifetime. Bullets are designated either friendly or not - the plan in the first place being that the player would not take damage from friendly bullets - this was later changed as having to dodge one's own gravity-affected bullets added another aspect to the game. The friendly designation is now only used for the buzzer.

Boss Key



Pressing 'b' while playing will hide the current game screen and instead bring up a screen meant to look like a classic Windows blue screen error. The player can return to the game at any point by pressing 'b' again. While the boss-screen is showing, the game will be paused. It's important to notice that the boss-screen can only be opened from the actual game screen - not from any of the menus. Our logic behind this choice is that a boss-key should hide the game quickly - this can also be achieved by simply closing down the game. Closing down the game while not currently in a game-run won't do the player any damage. Closing the game during a game-run will however - the boss-key aims to fix this issue which is why the game is only paused - not terminated - when the boss-key is entered during a game.

Score/Highscore

Our game implements a highscore system. If a game-run reaches a score greater than currently on the highscore board then the player will be prompted for a 4-digit

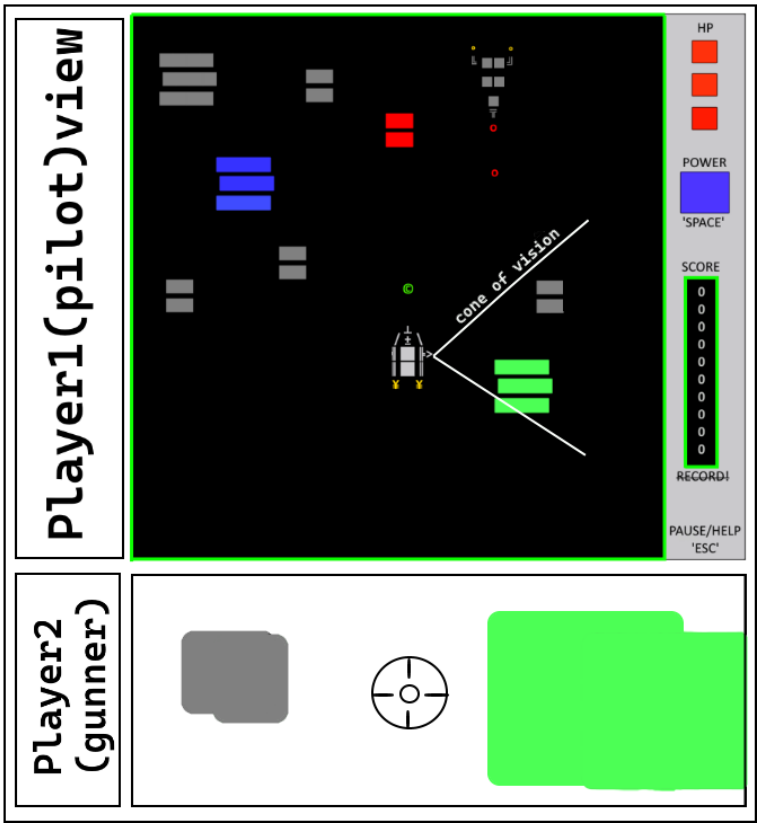
name and the score will afterwards be stored in flash memory. Score is accumulated throughout the game by time passing (100pts/tick), shooting asteroids(1111pts), and shooting spaceships(2222pts). We've discussed punishing the player for not shooting asteroids and spaceships by subtracting points, however we feel this isn't needed, seeing as the players are already rewarded for shooting. Shooting asteroids and spaceships can therefore be seen as optional. It is possible to achieve a high score simply by dodging.

Two players & Game Controlled by 30010-Joystick

Our game was designed with a two-player playstyle in mind. Unlike many other games, where two-player might mean one player facing off against the other, our two-player experience is cooperative.

One player controls the spaceship (pilot) - this is done using the keyboard. The other player controls the guns on the spaceships (gunner) - this is done using the 30010 Joystick and the accompanying buttons.

When playing the game with two players we recommend that the pilot only looks at PuTTY while the gunner only looks at the LCD display. This way of playing the game is harder than when the players share information, however it is also way more fun - forcing the players to constantly communicate!



Sketch of how the two-player system works. It should be noted that there is in fact not a visible cone displayed on screen nor are the objects on the LCD coloured. Figures displayed on the LCD are chosen on the basis of what object is being translated and how far away the object is.

PuTTY provides a top down view of the game, which lets the pilot see the big pic-

ture. It is the job of the pilot to dodge incoming objects and position/stabilize the spaceship for the gunner, who's job it is to shoot the passing asteroids and spacecraft. This would be hard if the gunner couldn't see his enemies. To fix this problem our game implements real-time translation of objects from PuTTY to the LCD. This means that if an asteroid would pass by the spaceship, this asteroid would be translated to the LCD and displayed. The LCD does however not have the ability to show the right and left of the spaceship at the same time and so the gunner can control what side he's facing with one of the buttons on the 30010 joystick.

Show Info on RGB/LED & Sound from Buzzer

Our game takes advantage of both the RGB/LED and the buzzer although the reason behind including these are more apparent when playing multiplayer. Here the LED and buzzer both provide valuable information to the gunner. The buttons on the 30010 Joystick are not the most reliable, so it can be confusing for the gunner to know whether he indeed did swap side and if he shot a bullet when he meant to.

The LED solves this issue, showing BLUE when facing RIGHT and RED when facing LEFT. The gunner does not need to keep track of what color means what - he simply needs to confirm that the color has changed to derive valuable information. When shooting, the LED will for the same reason flash GREEN.

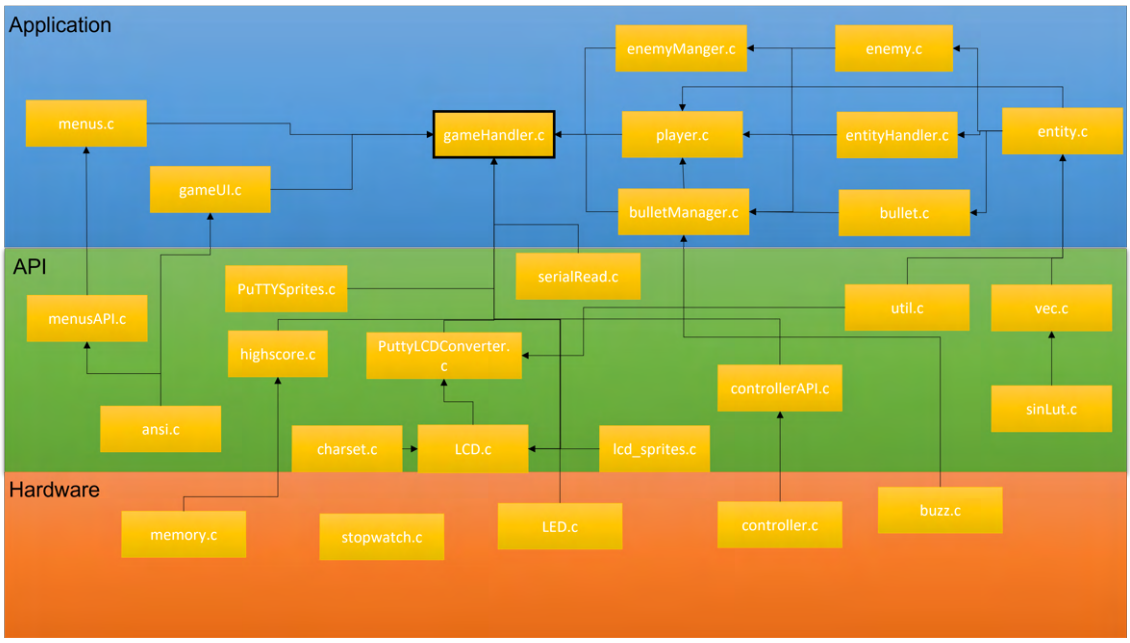
The buzzer aims to solve another issue: The gunner not knowing if he actually hit his target. If a friendly bullet collides with an asteroid or spacecraft, the buzzer will beep shortly.

Lastly it should be mentioned that both the buzzer and the LED work the exact same way in singleplayer. Here their utility is nearly non-existent, however we decided to include them as they (especially the buzzer) make the game more engaging.

5 Project Structure

5.1 Overall Structure

We will begin this section by presenting the overall structure of our program, visualised by a block diagram describing how each file of our project fits into either the abstraction layer, API or HAL.



Architecture abstraction model of our program

The arrows that link the fileblocks represent the use of functions or structures of the arrow-start file by the arrow-destination file. It's important to note that each fileblocks relative position in each layer does **not** reflect whether it leans more or less towards one of the other layers.

The gameHandler is the top level of our program and the file from which a few functions should be called in the main function in order to run the program. We use an entity class to keep the basemechanics uniform across the different gameobjects in our program. One could argue that the entity file should be in the API layer, but since we included specific functions to our application like calculateGravity we chose to declare entity as part of the application layer. If we simply moved calculateGravity somewhere else it would be entirely within reason to place entity in the API layer.

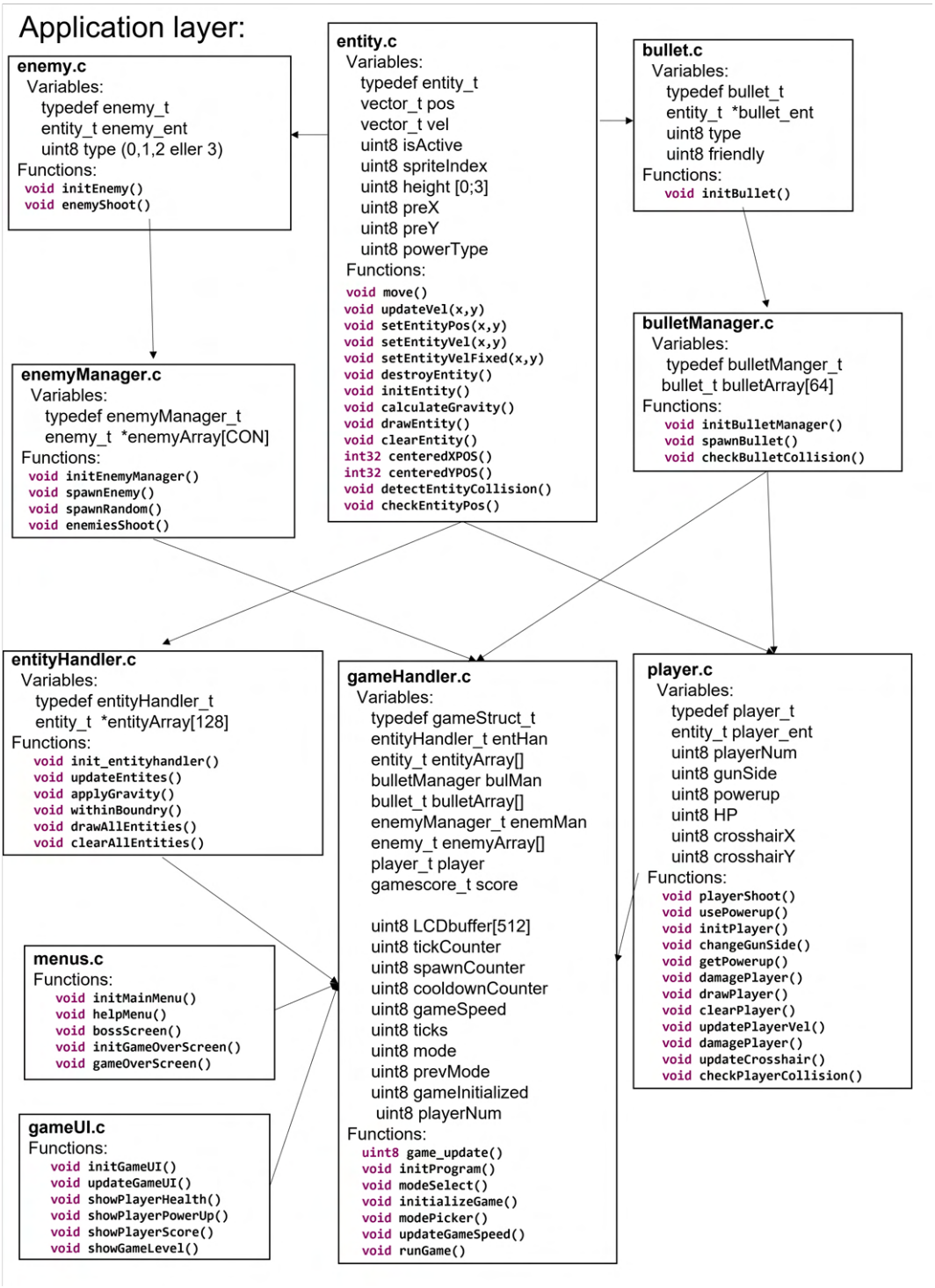
The stopwatch file is intentionally drawn without connections to any other file as it only initializes TIMER2 and uses its interrupt to control the global variable *timer_flag*. This flag value is then read in the gameHandler, but since the only communication between the files are done through the use of a global variable we didn't draw a direct line between them.

We didn't draw any architecture diagrams like this in the planning stages of our project development but we did create block diagrams for the application layer and a list of API functions / files we knew were needed for the program. The structure of our application layer did change throughout the development process, but these adjustments mostly expanded on the original idea of a lower abstraction entity file

with higher abstraction files build on top of it.

5.2 Application Layer

To gain a deeper understanding of the application layer we have created a blockdiagram to 'zoom in' on the fileblocks and reveal their variables and functions. The connections are the same as in the architecture diagram just shown vertically.



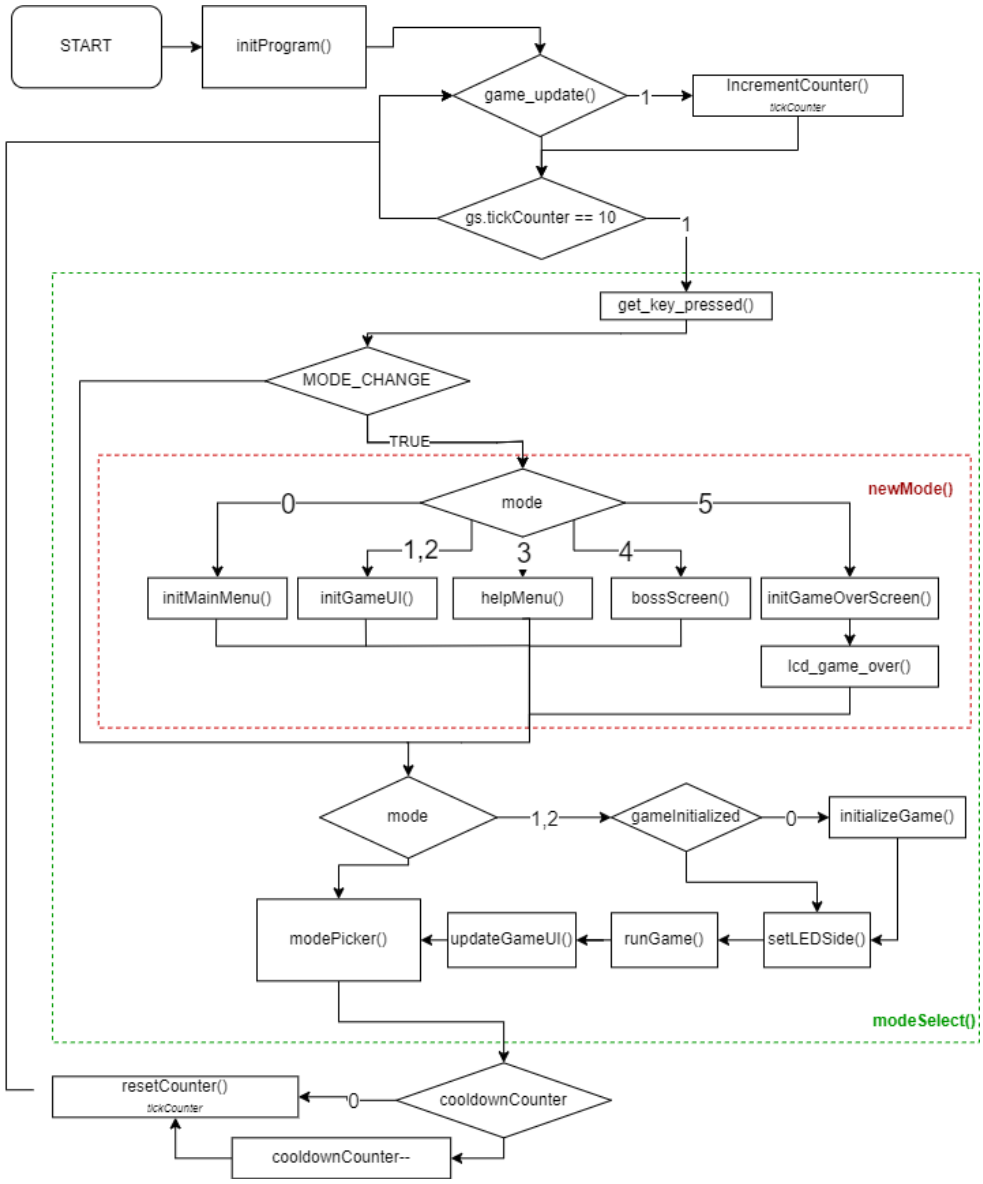
Block diagram of the application layer of our program.

This block diagram shows how our program architecture builds up the complexity and abstraction of its elements by going from a generic entity structure to the spe-

cific types of entities and their managers. Finally the top level file, gameHandler, holds all the information that is then passed through functions to the individual managers and API.

If we compare this final block diagram to the ones we made in preparation for the two status review meetings it's clear that we at first underestimated the scale of the application layer except the user interface part which we overcomplicated because such a modular userinterface system wasn't suited for a small project of this scale. In the second block diagram of the application layer the managers and handlers were introduced. The only difference to the final design was the top-level gameHandler file.

5.3 Application Layer - Flowchart



Flow chart of the top-level of our application. Note that the flowchart is not complete as this would make the figure too complex. We have however added more detail breaking up the functions modeSelect() and newMode().

This is the final flowchart for the logic in the main function that runs our pro-

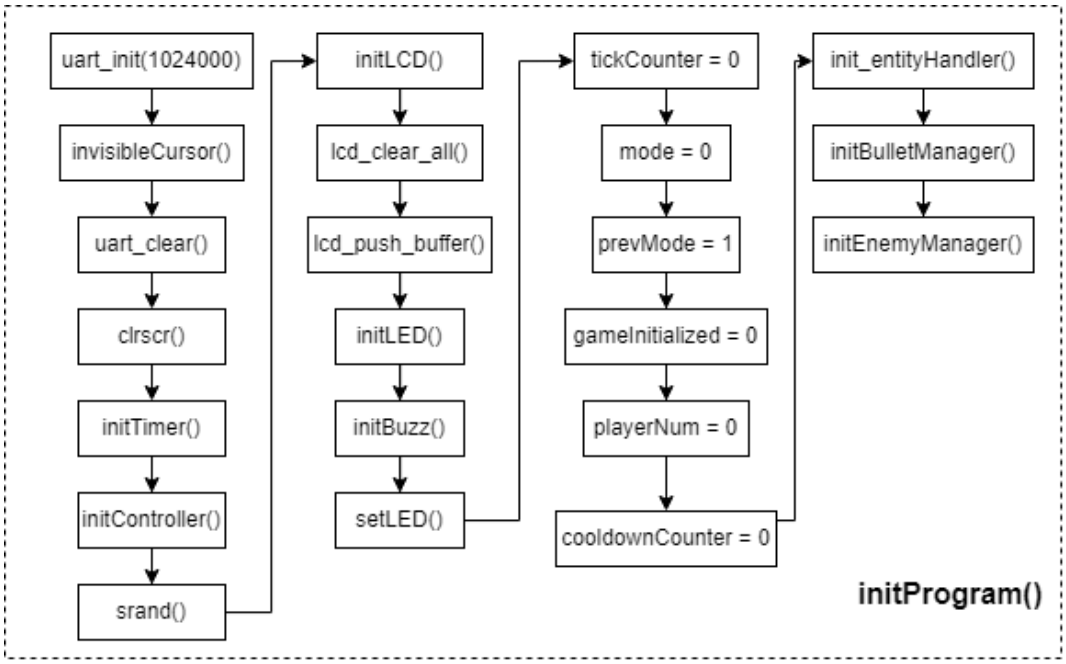
gram. We have expanded a few of the function calls to gameHandler as this transparency is required to understand the overall flow of the program. The first of these is the modeSelect() function. This function first takes input from the user using get_key_pressed(). If MODE_CHANGE is true (which it is if the mode has changed e.g. you've entered the help screen from the start menu) then newMode() will be called, which is a function that calls the function only needed to be called once when entering a specific mode. After newMode() a few additional functions are called, many of these only relevant if the mode is 1 or 2 (we're playing the game in singleplayer or multiplayer mode).

5.4 Important functions - Flow Charts

This section presents flow charts of other important functions that we could not fit onto our application-level flowchart. These include get_key_pressed(), runGame(), modePicker(), initProgram(), and initializeGame(). We will go through these one at a time, ordered by their appearance on our main flowchart. We have chosen to not include any of the arguments called with the functions as this would clutter the flowchart and be a factor of confusion rather than help.

initProgram()

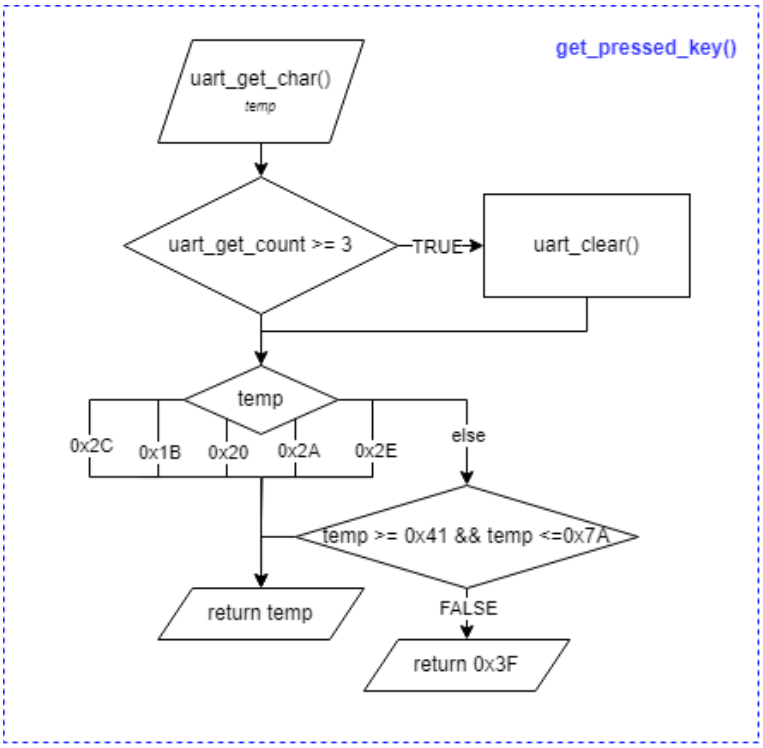
initProgram() is a function meant to initialize the very basics of our program. Here we start by initializing our hardware, then we initialize the portion of our gameStruct values needed to run our menus and lastly, we initialize our top-level structures such as the entityHandler, the bulletManager, and the enemyManager. It should of course be noted, that before initProgram() is called, a gameStruct has been created in which these values and lesser structures are kept.



Flowchart of initProgram(). As expected functions are called to initialize hardware and structs and values are initialized to specific values.

get_key_pressed()

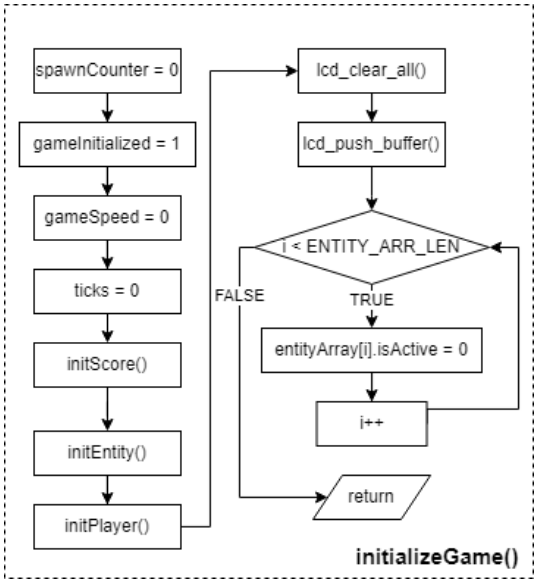
This function serves as the main layer of communication between the player and the game. All inputs (from the keyboard) are found using this function. The function utilizes a given function uart_get_char() which delivers the next character in the UART buffer. The function also makes sure that this buffer never gets too full. Our program never needs more than 2-3 key inputs at once, meaning of the buffer ever gets over that amount it should be cleared. This fixed issues of the player holding down the movement button instead of pressing it - instead of being locked out of the option to make new moves before the buffer is cleared our game will simply instantly clear the buffer. Lastly, get_key_pressed() serves to make sure our program only gets the inputs we would like. On "correct" inputs, this function returns the input itself, on "wrong" inputs the program returns '?'.



Flowchart of the get_key_pressed() function.

initializeGame()

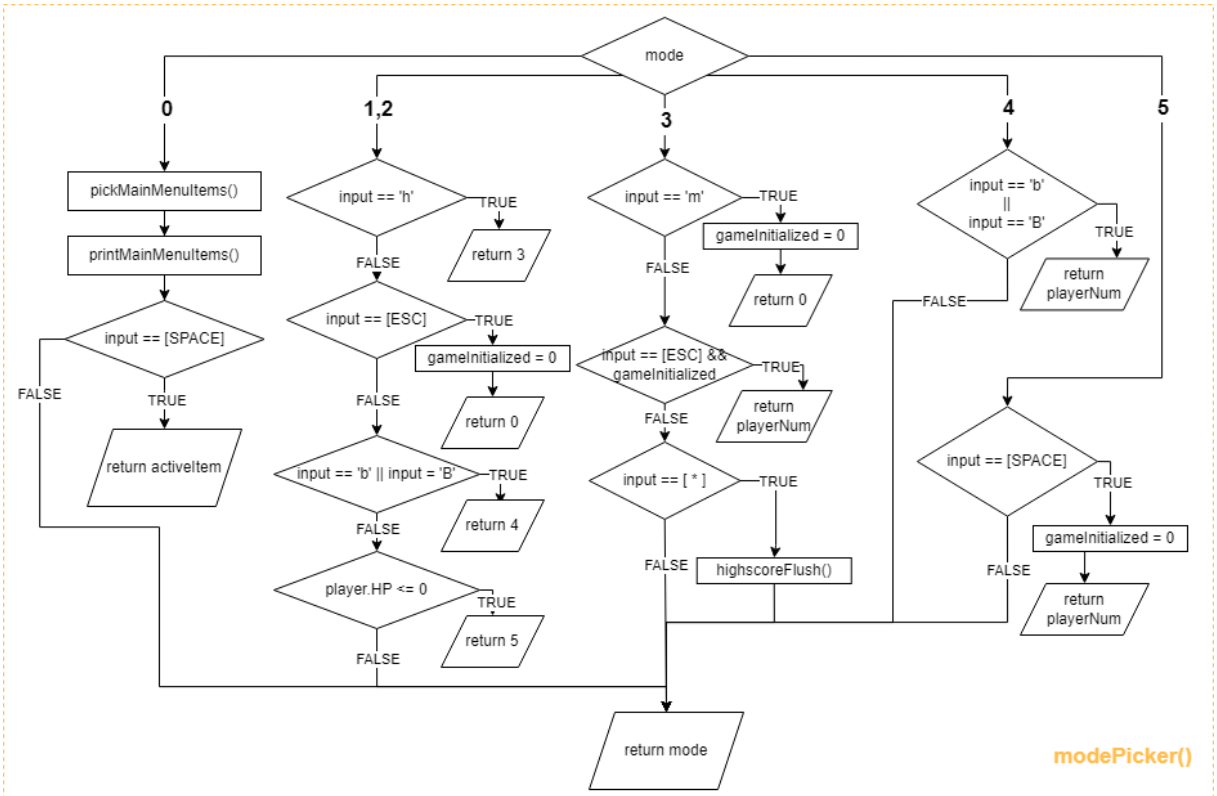
As can be seen from the primary flowchart, this function should only be called if gameInitialized is 0, meaning that there is not currently a game running. This makes sense, seeing that initializing a game when there's already a game running would mean overwriting the current game, which would not be ideal. To initialize a game we do much of the same as when we initialize the program itself. Values in the gameStruct are set, structures are initialized, and certain values in these structures are set as well. It should be noted, that the LCD is initialized both when playing single- and multiplayer. This is due to the LCD always displaying game-over if the game has been lost. Also, the for-loop controlled by 'i' should probably have been made into a function native to entityHandler, however, at the time of writing this slipped past our attention.



Flowchart of initializeGame()

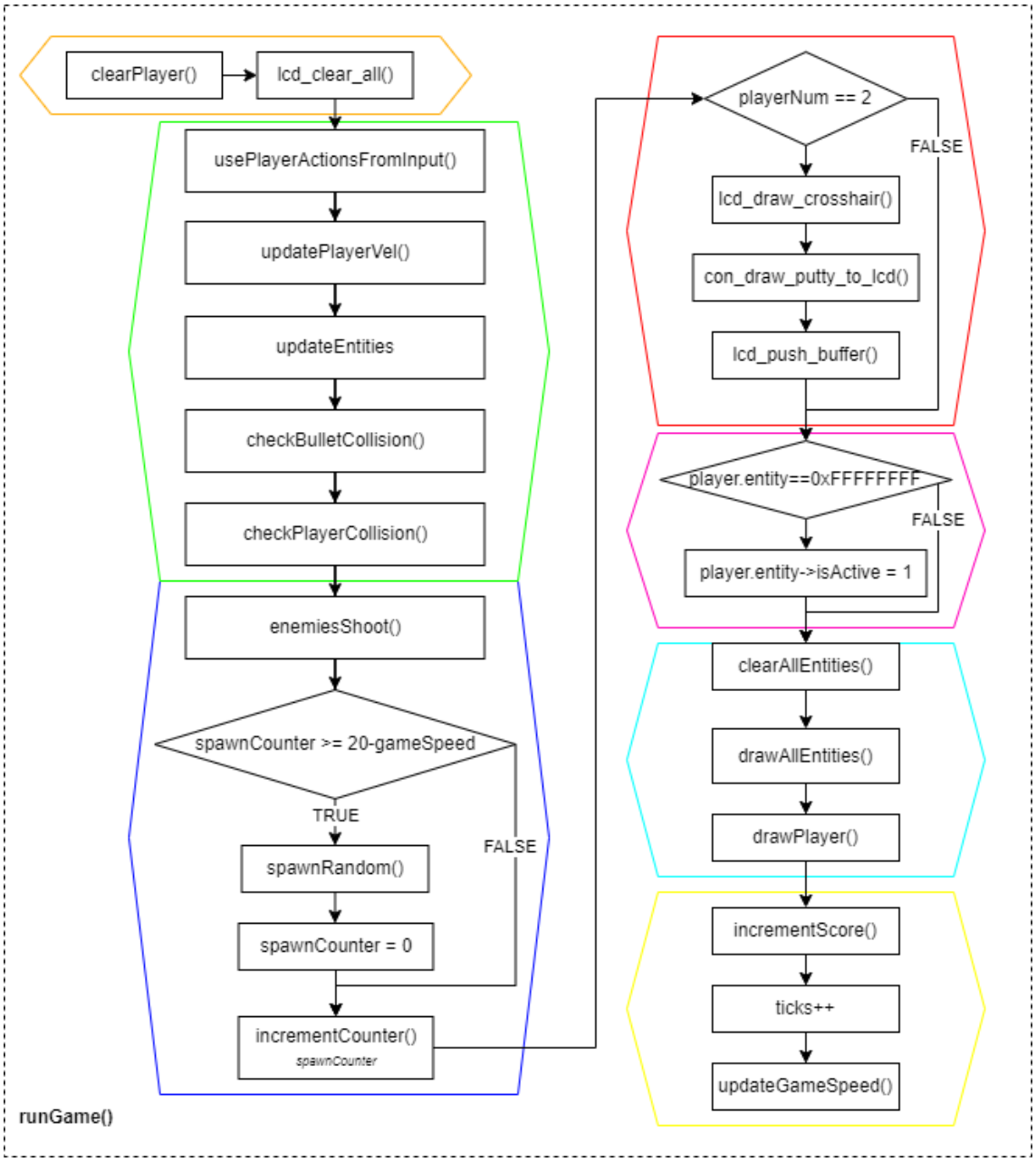
modePicker()

modePicker() is the function accountable for swapping between the different modes depending on user input. This is done using a switch statement using the current mode as an argument. By knowing what mode we're in and what inputs will make us exit that mode, we can use modePicker() to return a new 8-bit unsigned integer value corresponding to the new mode. This principle is also used to call functions in modePicker() such as displaying the gameOverScreen() while in mode 5 and deploying the interactive menu in mode 0.



Flowchart of modePicker()

runGame()



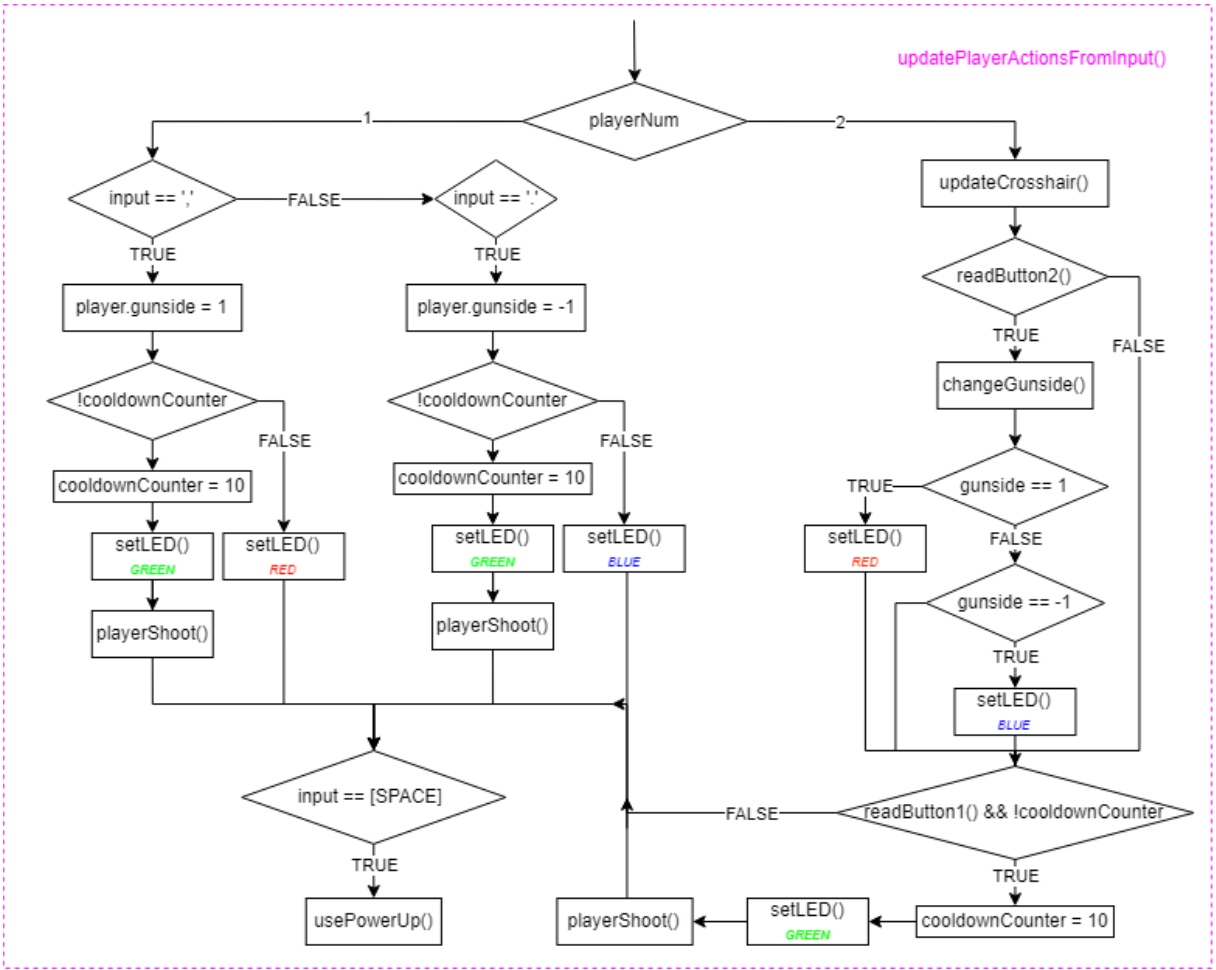
Flowchart of the runGame() function.

The runGame() function can be separated into seven different parts. Looking at the flowchart, these parts are grouped by color. The main goal of the runGame() function is to clear, update, create, and display all of the entities and game-values. Firstly, in the orange section, we clear. Then in the green area we update the player and check for collisions between entities. In the blue section, we create new entities, such as bullets when enemies shoot and spawn enemy entities, controlled by a spawnCounter. The red section is only relevant in multiplayer, where we need to draw on the LCD. In the next pink section, we do a bit of debugging. Running our code, we multiple times ran into an error, where the player would disappear. After debugging we found, that the pointer to the player now instead pointed to 0xFFFFFFFF and that the player would be designated as not active. In the cyan section, we clear all entities and then draw all entities. It is also here we draw

the player. Lastly, in the yellow section, we increment the score and update the gameSpeed, controlled by the ticks variable.

usePlayerActionsFromInput()

Lastly, the usePlayerActionsFromInput() function is a part of the runGame() function and very important. This function makes it possible for the player to shoot, change gunside, and use powerups. Some of these actions require different inputs dependent on single- or multiplayer mode.



Flowchart of the usePlayerActionsFromInput()

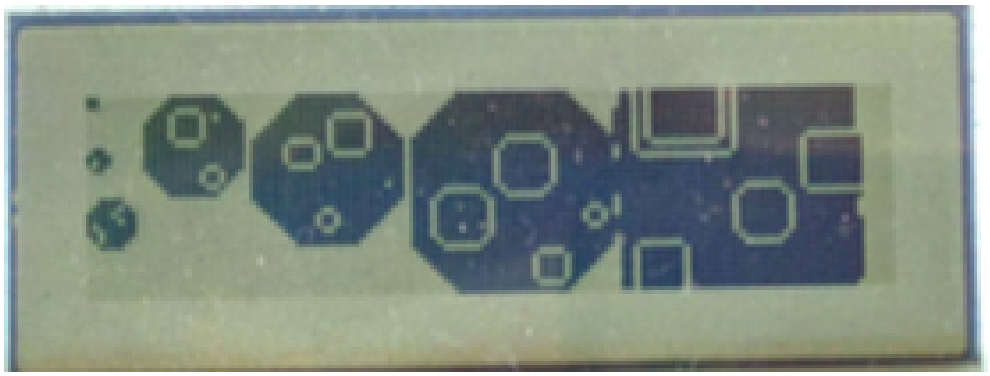
5.5 API layer

We will now shortly explain selected modules of the API layer and their purpose. These modules were selected because we feel that they are especially interesting.

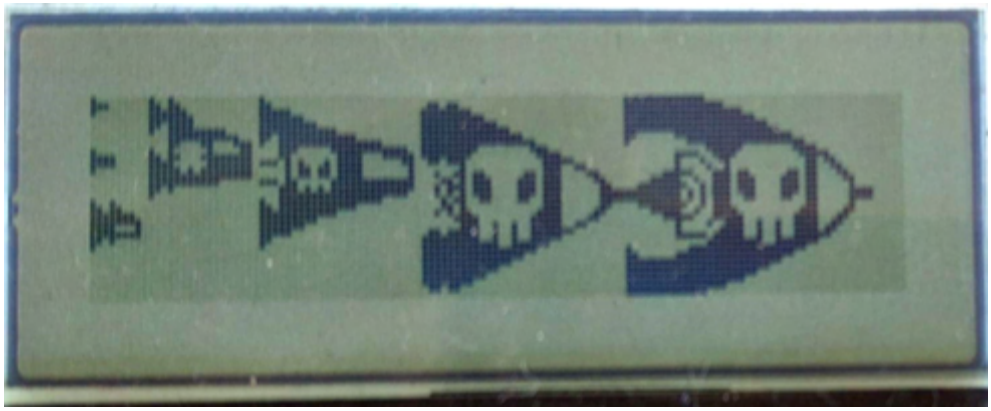
LCD.c

The LCD module is an expansion of the LCD file created in exercise six. It contains functions for drawing text and sprites on the LCD display peripheral. It does this by manipulating the 512 bytes of a buffer before its pushed to the display. Each byte controls a vertical strip of eight pixels on the display. The buffer array is onedimensional so we have included logic in our drawing functions to stop it from

drawing the bytes that couldn't fit on the next line.
To write text on the LCD it uses the given charset.c file given in the exercise as a lookup for the five bytes that make up each letter. Sprites are drawn with the same method using our own lcd_sprites.c file with a massive lookup for the 14 different sprites we can draw on the LCD. We might have spend a little too much time creating these sprite lookups.



All seven different sizes of asteroid sprites displayed on the LCD.



All seven different sizes of enemy spaceship sprites displayed on the LCD.

PuttyLCDConverter.c

This module contains the functions that establish the connection between what is seen on the computer screen in PuTTY and what objects can be seen flying past you on the LCD. It has functions that get the direction in PuTTY space for bullets fired by the player aiming the crosshair on the LCD. The remaining functions are dedicated to the positional translation between PuTTY- and LCD space. The distance between an entity and the player, whether the entity should be viewable in the cone view of the gunner player and finally what a position in PuTTY should translate to on the LCD (assuming its viewable). We will go more in-depth on these function in a later section.

PuTTYSprites.c

PuTTYSprites.c stores different sprites used to display the game in PuTTY. These are all structured in a three-dimensional array, so they are easier to access, but due to you not being able to create multidimensional arrays of different lengths in c,

some of them do waste space. To make sure the board is not wasting time writing empty characters to PuTTY, the functions to draw and clear sprites have helper functions that limit the kind of characters written. This way we can fill all of the indices of no interest with a single character which we do not wish to include. In our case '?' was chosen.

The worst offender is the regular bullet sprite which consists of a single character followed by 41 empty characters. Rather than checking all 42 characters to see which ones to draw or clear, the helper functions make sure that the board only writes or clears the single character where the bullet is and ignores the remaining part of the 2-dimensional sub-array.

util.c

The util.c file contains various generally useful functions for application development. Pseudo-random numbers, distance and mapping. Also a few small mathematical functions like absolute value and sign. We didn't want to import a large math library only to use a few functions that we wouldn't be sure were even optimized for our usecase.

highscore.c

The different functions found in highscore.c are designed to interface with the memory and make it easier for a programmer to read and save high scores in an efficient manner. It has basic functionalities such as reading names and scores from the leaderboard, but it also contains functions to save scores that automatically orders the high score list. Lastly, there is also a function to clear all high scores from the leaderboard which is mainly intended to run once whenever the code is uploaded to a new machine.

5.6 Hardware Abstraction Layer (HAL)

In this section we aim to explain selected parts of our hardware.

controller.c

The 30010 course-provided controller features two buttons and an analog joystick consisting of two potentiometers. The buttons are easily configured as they are digital and connected via a pull-down network. Reading analog values from the potentiometers requires a bit more work. These potentiometers have to be connected to specific pins configured to read analog inputs. The microcontroller has multiple of these and they are connected to an Analog to Digital Converter (ADC) through different channels. After calibrating the ADC, it is possible to start reading values from it. It has 12 bits of precision meaning the value it returns ranges from 0 to 4095 depending on how much voltage is measured on the output of the potentiometers. These values can be read using a function that returns a 16 bit unsigned integer. The potentiometers do not have full range, as they are slightly limited by the joystick itself, but they still have more than enough range to give reliable readings.

memory.c

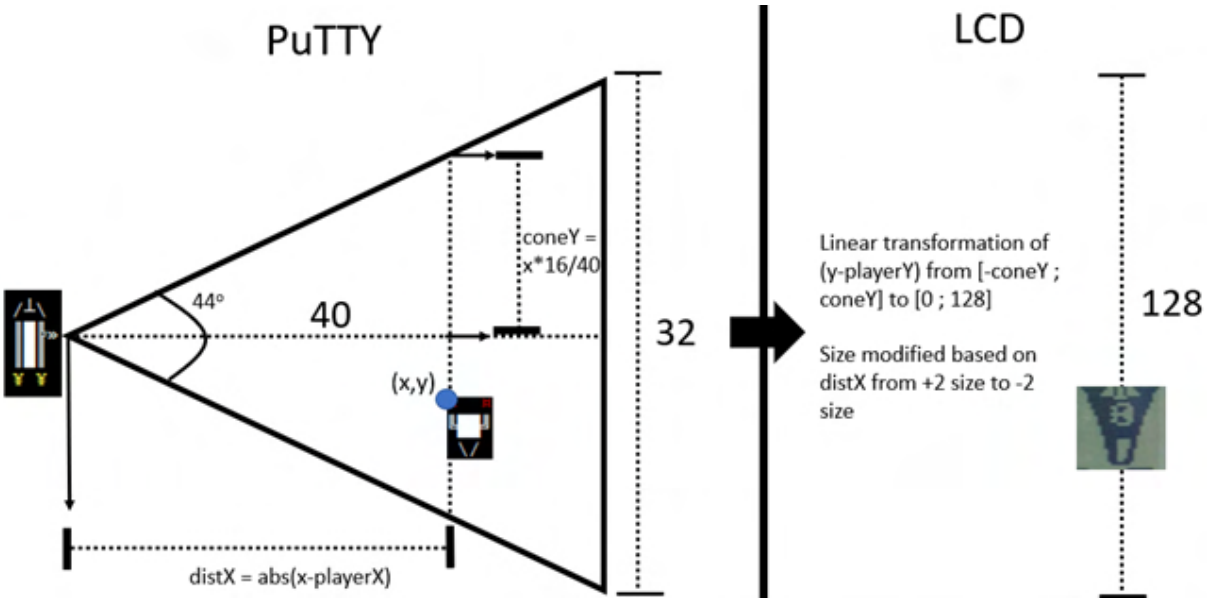
There is a more in depth explanation of how the memory is structured later in this report, so this part will mainly focus on how the memory functions have been designed to make interfacing with the memory easier. The STM32 Nucleo has 32 'pages' of memory and the memory 'spots' take up two bytes (16 bits). You can read from the memory simply by typecasting a memory address to a 16 bit unsigned integer pointer and read the value of this pointer. The function that handles this takes the base address offset that gives the last page in the memory. It then adds a further provided offset given as a parameter multiplied by two (since each memory spot is two bytes). This means you can read the information stored in a specific memory location simply by calling a function and giving it the additional small offset.
Writing to the memory is rather simple. First, you have to unlock it and then you can erase the entire page that you wish to write to. You can then write 16 bit unsigned integers to the memory in a similar way to how it was read. When all of that is done, you can lock the memory again.

6 Highlighted Features

In this section we want to more thoroughly explain some of the more complex and interesting extra features of our application.

6.1 Translation Between PuTTY Space and LCD Space

The view of the gunner is a cone with a base size of 40 PuTTY units height and 32 PuTTY units width. This size is set by defined constants. The translation between enemies in PuTTY and on the LCD is a linear transformation based on the slope of the cone. We have created a graphic to further explain the process:



Explanation of the linear transformation between PuTTY- and LCD space

This graphic is a little simplified as we also pad the values of the intervals in the mapping operation to allow enemies to be drawn around the edge of the LCD. As shown there are 7 different sizes for the LCD sprites for both the enemy spaceships and asteroids. The enemy spaceships always have a base size of 3. Meaning their base sprite with no size modifier on the LCD is 3x3 bytes = 24x24 pixels. This size can then be modified based on the distance between the entity and the player by up to +2 when the entity is close to the player and down to −2 when the entity is located in the most distant part of the cone. This means we actually dont use the 1/4 x 1/4 and 1/2 x 1/2 sprites for the enemy spaceships. For the asteroids the calculation is the same except they can have 3 different base sizes from 1 to 3.

When aiming and shooting with the crosshair on the LCD the translation to a PuTTY space direcion for the bullet is calculated in a similar but simpler way.

6.2 Flash Memory Usage

As mentioned earlier, the STM32 Nucleo has 32 memory 'pages' that are used to store the program on the board. We will be reserving the last of these pages to save high scores. These pages have way more space than needed to save a few high

scores, but it is necessary to dedicate an entire page to one thing as you have to erase the entire page whenever you want to write information to it.
The position a high score has in the memory is also its position on the leaderboard. This is shown in the table below:

Offset	32B	24B	16B	8B	0B
Data	`highscore4`	`highscore3`	`highscore2`	`highscore1`	`highscore0`
#Bits	64b	64b	64b	64b	64b
#Bytes	8B	8B	8B	8B	8B

Overall memory structure

Every individual high score is optimized to waste no amount of space. The score itself is a 32 bit unsigned integer which gets split up into two different 16 bit spots. A high score leaderboard is nothing without names next to the scores, and you can write most abbreviations with just four letters which is what there has been made space for. A single character takes one byte to store, so bit-shifting makes it possible to store two characters in a single spot. The structure of a single high score is shown in the table below:

	highscoreX					
Field	Name; 4 chars = 32 bit				Score; 32 bit unsigned	
Data	`name[0]`	`name[1]`	`name[2]`	`name[3]`	Score 16 bit MSB	Score 16 bit LSB
#Bits	8b	8b	8b	8b	16b	16b
#Bytes	1B	1B	1B	1B	2B	2B

Individual high score field structure

It goes without saying that this optimization was not strictly necessary in this project, and it was mainly done as an exercise in memory organization. It could, however, come in handy if this project was further developed in the future and became larger or if more than just the top five high scores were added.

6.3 Gravity

Our game's version of gravity is not very advanced. For gameplay reasons, gravity only affects bullets, and only asteroids have a gravitational pull. The game would simply be too confusing and very difficult if more entities had gravity. Another limiting factor is distance. Instead of having every single asteroid pull on every single bullet, only asteroids within a distance of 20 can pull a bullet. This distance is calculated using the positions of the two objects (the position of the asteroids are adjusted to be in the middle of their sprite) with the Manhattan distance API function getManDist(). This method uses the following formula:

$$\text{dist} = \text{abs}(x_2 - x_1) + \text{abs}(y_2 - y_1)$$

It should be noted, that we've already developed an API function absolute() which returns us the absolute value of an 32bit signed integer, which is the correct size for our fixed point 18.14 numbers. Not using this function in getManDist() is simply an oversight on our behalf. We've chosen not to change the code as we first became aware of the problem after having handed in our hardware, meaning we would not be able to test the changes to our code.

To calculate gravity, we use the calculateGravity() function. Gravity is calculated using a simplified version of Newtons law of universal gravitation, where the gravitational constant is simplified to 1 with the before-mentioned dist used as the distance. Seeing as we're operating in 2D, we need a gravitional pull in both X and Y direction. We also need to know what direction in X or Y the bullet is being pulled. This is achieved using the API function norm() which returns -1 if the argument is negative and 1 if the argument is positive. The used formula can be seen below:

$$\text{deltaX} = \text{norm}(x_2 - x_1) \cdot \frac{\text{G} \cdot \text{massObj}}{\text{dist} \cdot \text{dist}}$$
$$\text{deltaY} = \text{norm}(y_2 - y_1) \cdot \frac{\text{G} \cdot \text{massObj}}{\text{dist} \cdot \text{dist}}$$

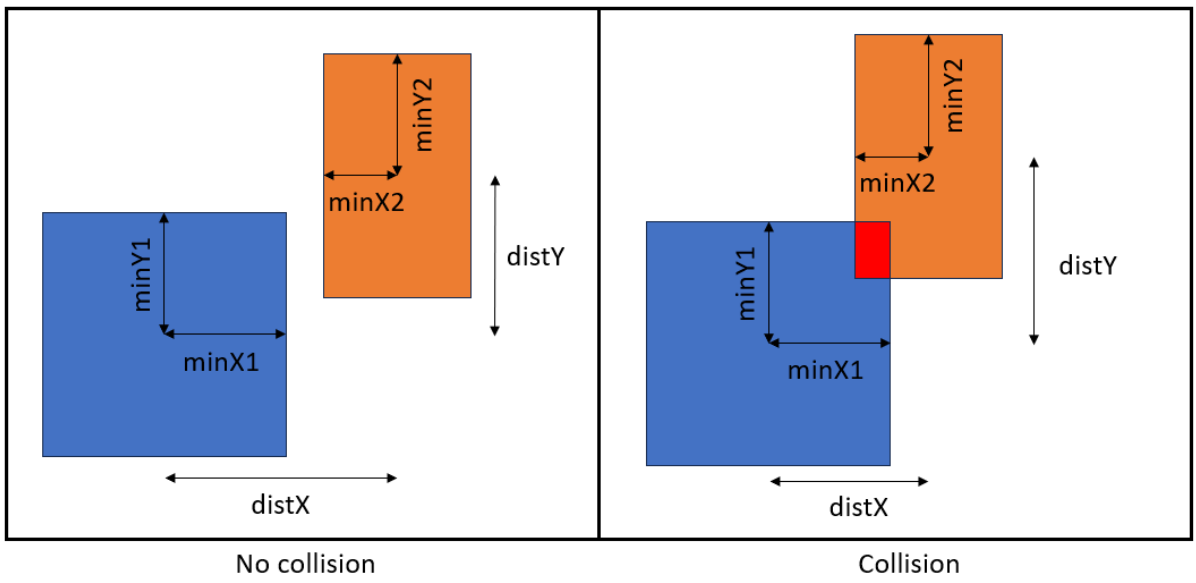
where massObj is found using a switch-statement controlled by the spriteIndex of the asteroid. Bigger asteroids will have a larger mass. We dont want the gravity to be too strong, if the distance is less than two, the distance will be set to two. The deltaX and deltY variables are then added to the X- and Y-velocity of the bullet.

Applying gravity is done using the function applyGravity() in entityHandler. Where we iterate through the entityHandler array checking every asteroid for every bullet (using their spriteIndex) while making sure that both entities are active using their isActive value.

6.4 Collisions

The game checks collisions in a few steps. It starts with checking if any active bullet has hit any other entity that is not the player and not another bullet. This detection system will be explained below. If there is a collision between a bullet and another valid entity, both objects will be destroyed. If the bullet came from the player, they will be awarded points for hitting asteroids and enemies and a short note will play from the buzzer.
After checking collisions between bullets and non-player entities, the game will check if the player has collided with any other entity - barring the Mega Bullet powerup. If this is the case, the player's health will decrease by one point for every entity they hit.

The collisions themselves are all calculated using the same function that takes two entities as its parameters. All have their own predefined minimum x- and y distances that make up their hitboxes.



Collision detection example

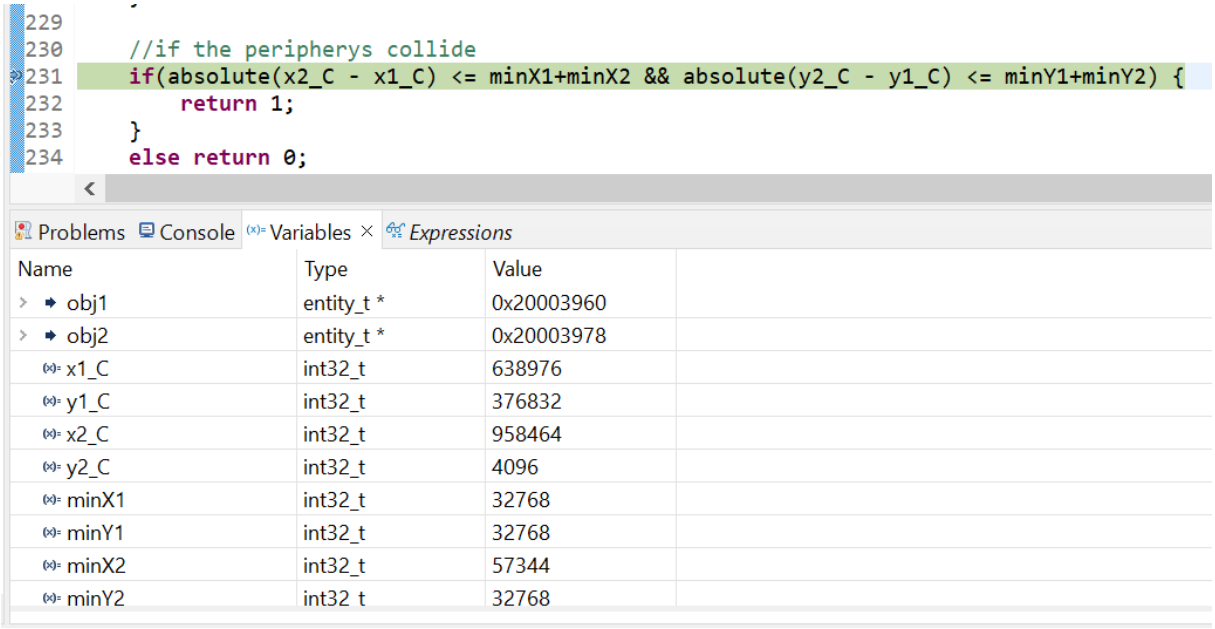
Two values are immediately calculated in this function: the difference between two entities' center-coordinates for x and y. If both of these are smaller than the entities predefined mininum x and y values put together, it detects a collision. In the left example, no collision is detected as only the y distance poses a problem. On the right, there is a clear overlap as the entities' minimum distances are both shorter than their respective difference in x- and y coordinates.
These calculations are all made using fixed point arithmetic to improve precision and increase speed, seeing as floating point calculations are very resource heavy and slow, which is not ideal for our microcontroller.

7 Project Verification

This section discusses different methods used in the verification and debugging process.

7.1 Debug Tool

The STM debug tool has been essential in solving the many bugs we encountered during our development process. The ability to set breakpoints and then examine the variables and expressions in the IDE at that exact moment was a very effective debugging strategy. The ability to track code execution backwards through the thread after a crash was also very useful.



Example of how the debugging tool can be used to examine the values of variables at a breakpoint.

7.2 Alternative Methods

While the built-in debugging tool is extremely powerful for certain tasks, it is not necessarily always the best thing for the job. After implementing certain time sensitive counters and delays, it was easier to monitor them in real time by printing them to PuTTY. This was done for the buzzer and the LED to ensure that everything was working as intended. These things would be harder to track in the debugging tool as breakpoints stop the execution of the program and make timers difficult to keep track of. Another downside to using the debugging tool came from the buzzer. Using the buzzer while the debugging tool was active was extremely annoying as it would not turn off since execution of the program was halted.

7.3 Module Tests

After writing a module we tested its functions on its own in a separate function in main to verify the functionality and decrease debugging once the entire project was

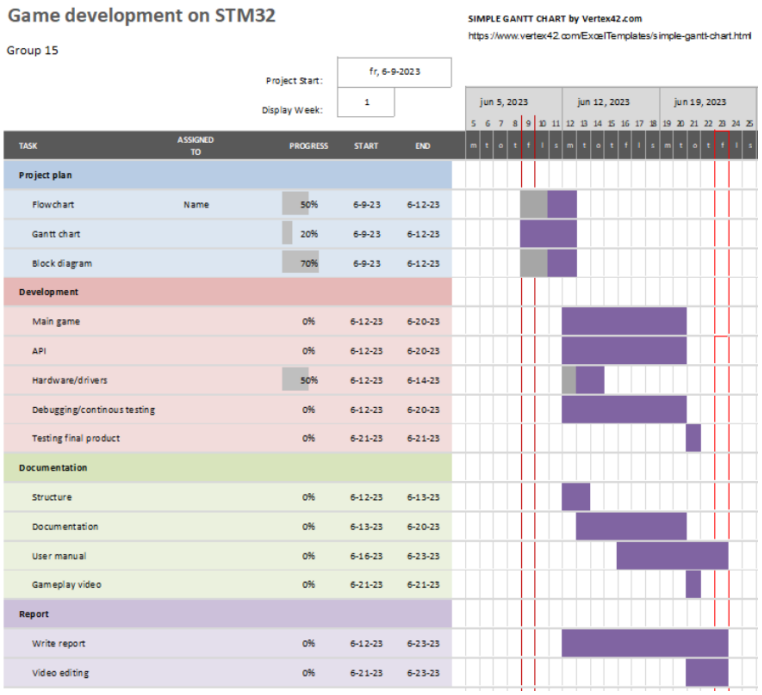
assembled from the smaller modules. These small top level test functions in main.c makes it easy to test modules but also come back to them for expansions if needed.

8 Project Plan

This section describes the planning and timeline of the project as it changed along with the project itself.

8.1 Original Plan

The timeline of this project has been structured around a Gantt Chart as it is easy to read while allowing you to keep track of subsections and deadlines.
We decided to use a free online Gannt Chart template provided by Vertex42[1].

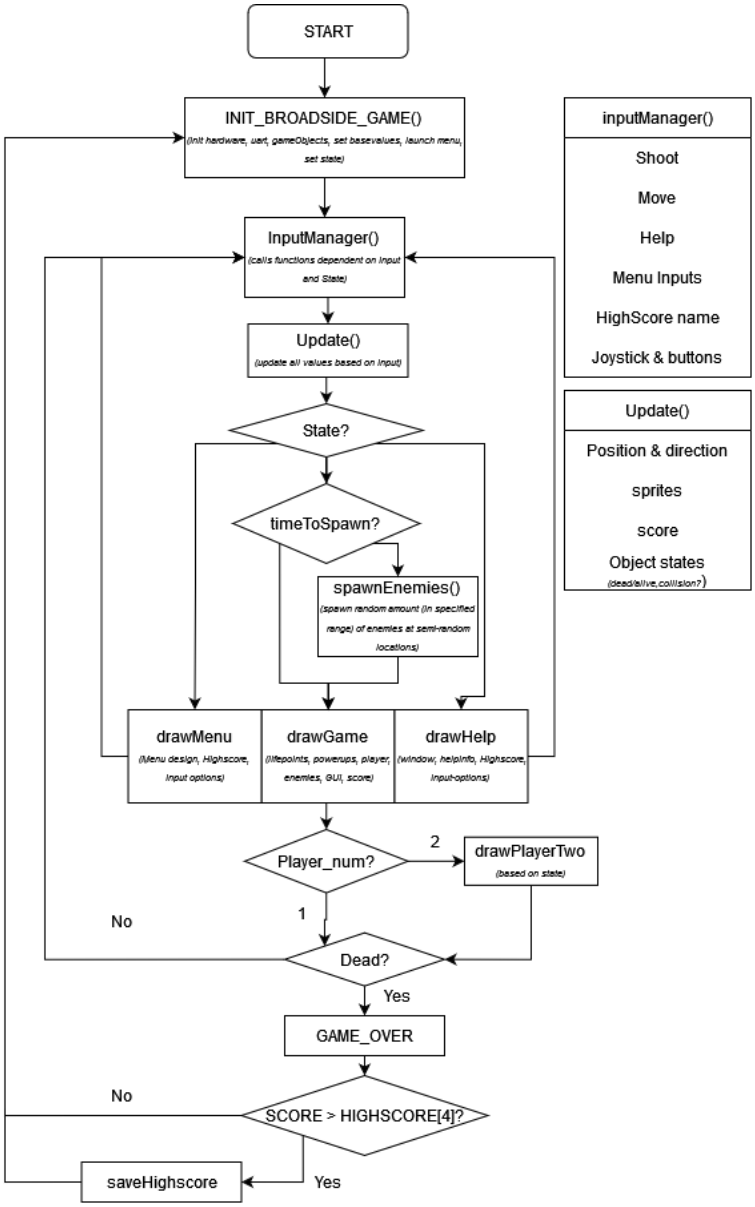


Original project plan

The original timeline is shown above where the major parts of the project were inserted with their respective deadlines. The bars on the right show how much time is assigned to a certain task, and the gray part of that part displays how large a percentage of it is already done.
We did not assign people to specific tasks since we kept it agile and let multiple people work on the same thing and switch around as necessary.

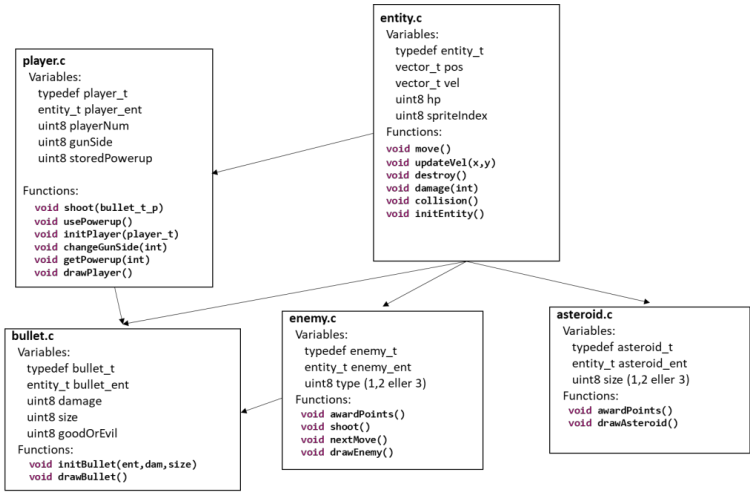
The original flowchart can be seen below, and it would initialize all hardware and software related features right away. Then it would move on and take user inputs before updating whichever mode the program was currently in (Main menu, Help menu, Game etc.). Everything, including hardware initialization, would run again whenever the player would die and it would even check if a high score was beaten while sitting in the main menu. Needless to say, there were many things that could be optimized, but parts of the general structure largely stayed the same throughout the development process.

[1]Template link: https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html



Original flowchart

The block original block diagram for the project is shown below, and it was very simple and only contained the absolutely necessary functions such as initializations and updates. This was not meant to be a complete list of functions but rather an initial outline of things needed for to get the game up and running.

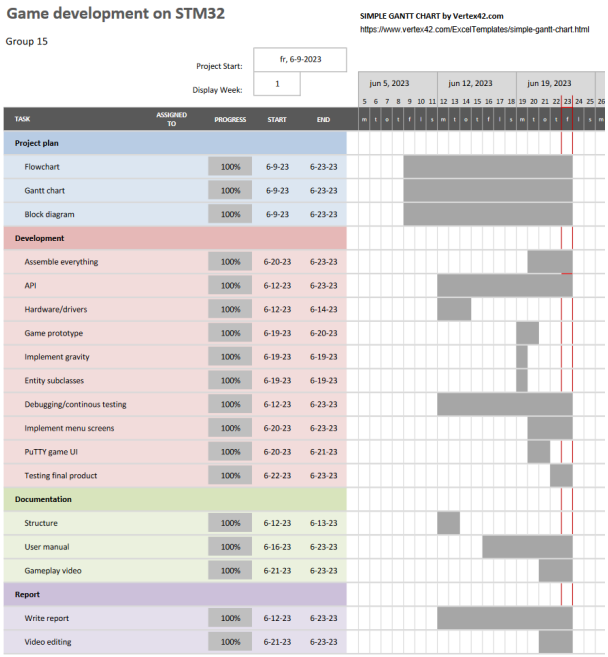


Original block diagram of the application layer

We also made an entire block diagram for UI elements, but it was later scrapped since it was overly complex compared to the project at hand.

8.2 Final Plan

The Gannt Chart was used throughout the project to keep track of progress and deadlines. This was changed in the final stages of the project to include more smaller parts of the game to work as a checklist of missing features.
We were slightly behind schedule as we approached the deadline for returning hardware, but this was extended to the 23rd rather than the 21st of June. This allowed us to finalize the game and fix multiple bugs that would be harmful to the user experience.



Final project plan

We did spend day more than expected assembling the program and fixing the subsequent bugs. This is reflected in the Gannt Chart as it was continuously updated,

and it was made possible by the agility of the project structure as well as the group members being able to switch and focus on different things as needed.
Another thing to note is that the light blue 'Project Plan' section has been extended in the final version. We did finish our original planning within the first week, but the plans evolved along with the scope of the project. All major changes to the application structure had to be documented in the flowchart as well as the block diagram, and the deadlines of these changes and additions were managed in the Gantt Chart.

There have been rather large changes to the flowchart, but certain core parts remained the same. Most of the changes come from the system that changes between different game states and menus. The original idea was simply to draw everything on screen every single cycle, but that proved challenging as the display would flash way too much and be uncomfortable for the user. This means we had to rethink how the output was generated and optimized it accordingly. All the static parts of the screen get updated once whenever a mode change is detected and only the dynamic parts get updated every cycle.
The parts that did not undergo large changes were the two fundamental ideas of sampling inputs before changing modes and splitting the program up into subsections depending on the mode. All of this is reflected in the updated version of the flowchart shown earlier in the report.

The final block diagram shown earlier in the report has grown to include way more parts of the program. More blocks have been added, but it still retains clear separation between all of them.

8.3 Who Did What

This table shows which files (both .c and .h) each member has contributed to. An X indicates that a group member has made a contribution to the file shown on the left.

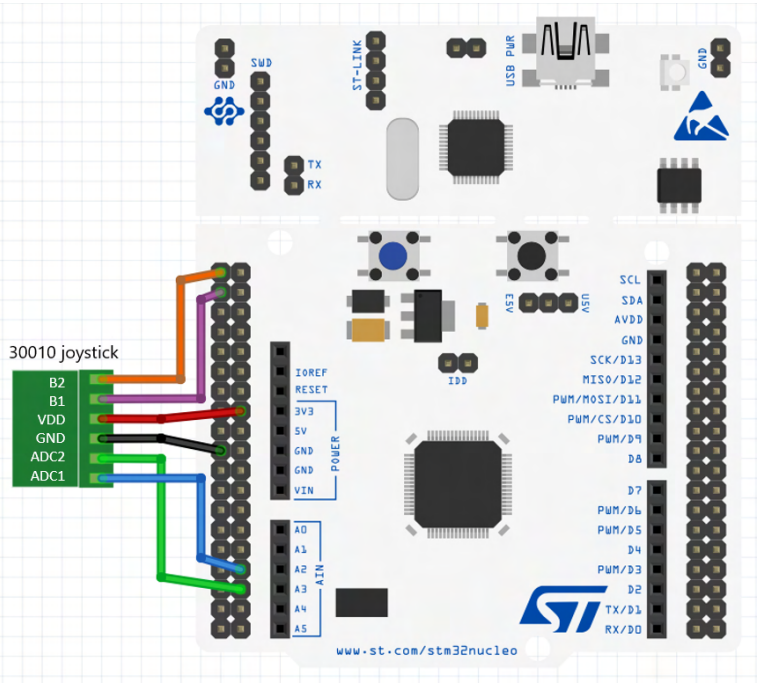
File name	s223998 Alexander	s224038 Georg	s224032 Frederik
main	X		
ansi	X	X	X
bullet		X	
bulletManager	X	X	
buzz			X
controller			X
controllerAPI			X
enemy		X	
enemyManager		X	
entity	X	X	X
entityHandler	X	X	
gameHandler	X		X
gameUI		X	X
highscore			X
lcd_sprites		X	
LCD		X	
LED	X		X
memory			X
menus			X
menusAPI			X
player	X	X	X
PuttyLCDConverter		X	
PuTTYSprites	X		
scoreCalc	X		
serialRead	X		
sinLut	X	X	X
stopwatch		X	
util		X	
vec	X	X	X

9 User Manual

This section serves as a guide to show you how to set up PuTTY and some things to do when running the program for the first time. This guide assumes you have already downloaded the project as a zip file and extracted it to the correct destination before opening it in STM32CubeIDE.

9.1 Hardware Setup

Our application partially runs on the provided joystick and buttons and it needs to be connected correctly to work.
It is important that this is connected before the code runs since the random seed is generated based on values from the potentiometers in the joystick.

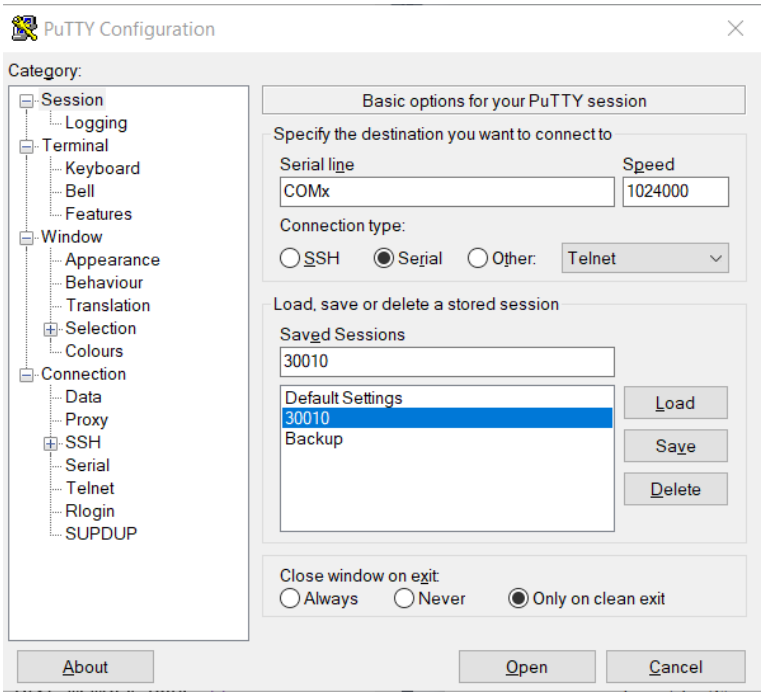


Hardware wiring diagram

NOTE: ADC1, ADC2, and VDD are all connected to the inner row of pins while the rest are connected to the outer row.

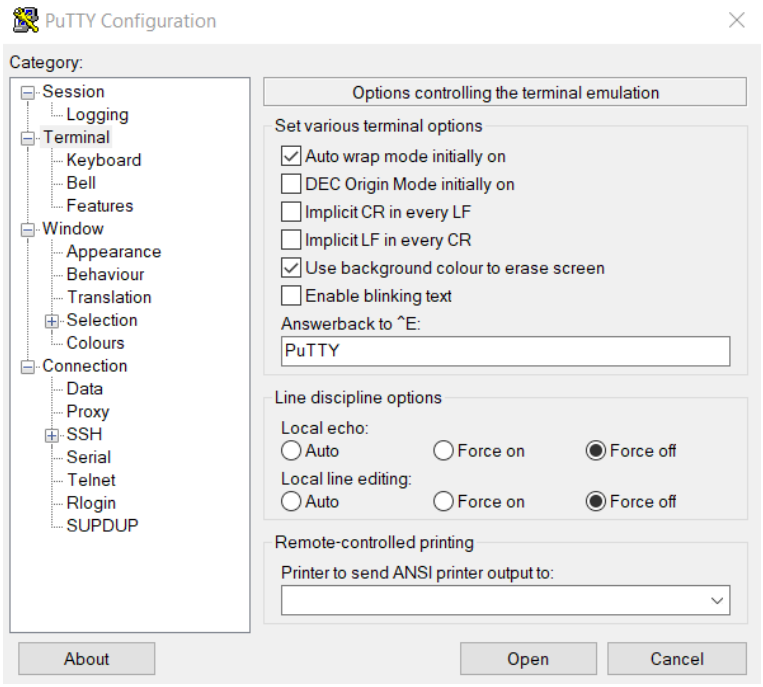
9.2 PuTTY Settings

Open up PuTTY and change the 'Connection type' to 'Serial'. Now set 'Speed' to '1024000' and set 'Serial line' to the COM-port your STM32 Nucleo is connected to. Your window should look similar to what is shown below:



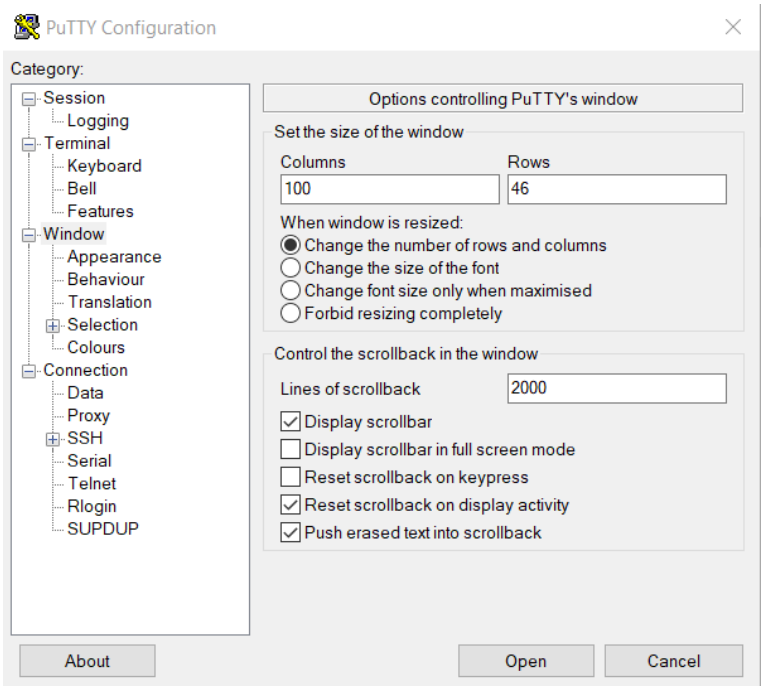
PuTTY 'Session' settings

Now, head to 'Terminal' on the left and change both options under 'Line discipline options' to 'Force off'. It should look like this:



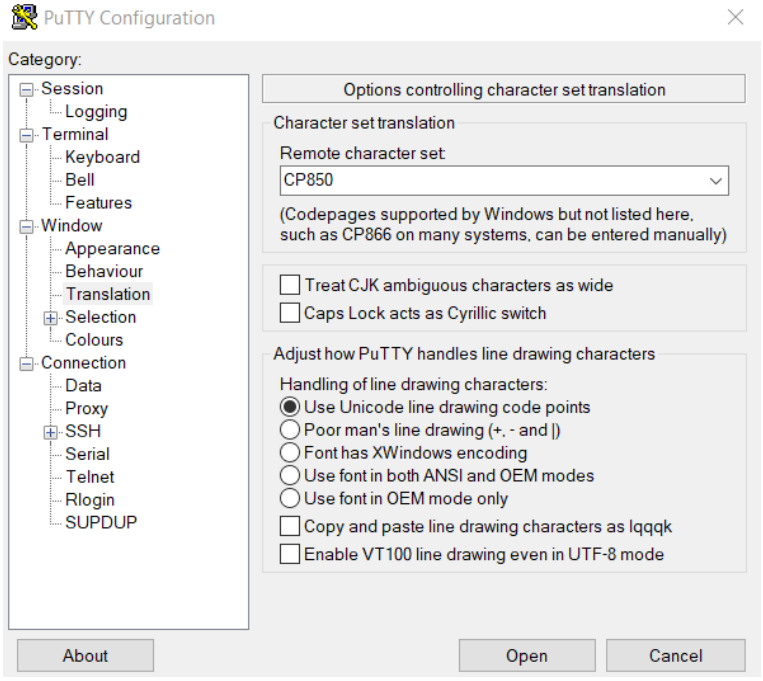
PuTTY 'Terminal' settings

The next step is to go into 'Window' on the left and set 'Columns' to '100' and 'Rows' to '46'. It should look like what is shown below:



PuTTY 'Window' settings

Finally, you have to go into 'Translation' and write 'CP850' under 'Remote character set'. You should now see this:



PuTTY 'Translation' settings

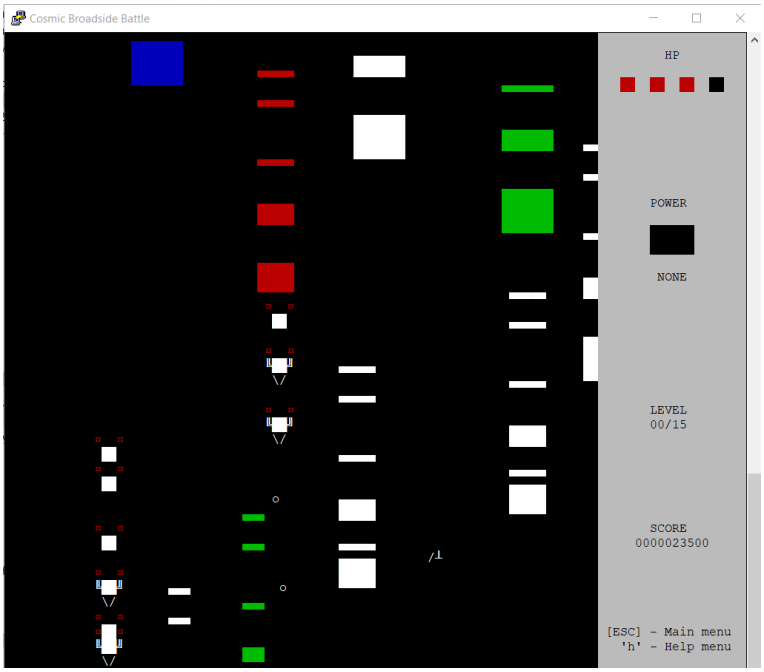
When everything is set up, you can press 'Open' to start the game window.
If there are any issues, we recommend you first set up PuTTY with the base configuration recommended in the course and overwrite any conflicting changes with our settings.

9.3 First Run

Everything should be set up and ready now. Go ahead and upload the project files to the board if you have not already done it.
We recommend you press 'RESET' on the STM32 Nucleo board after uploading code to it. You should be able to see the main menu if all was done correctly, but it might have some strange high scores displayed on the right. These are read directly from the flash memory, so any saved values from other programs will be read as high scores. To reset these, go into the help menu and follow the instructions in the bottom right corner. WARNING: This will clear everything saved on the last memory page.
You can return to the main menu after doing this where the high scores should be reset to default values.

9.4 Disclaimer

The game has a few known issues. The first is that pressing the arrow keys on the keyboard while in a game brings you back to the main menu as it is recognized as the escape key, which brings you straight back to the main menu.
The second issue slightly harder to track down as it only appeared sometimes on one of our computers. It also happens to be the lowest spec computer in the group, so PuTTY might struggle to run at the high baud rate on lower-end hardware. The issue is the game becomes unresponsive and does not clear sprites correctly which looks like this:



Game interface issues on lower-end hardware

10 Conclusion

We set out to create an enticing game that utilizes many different peripherals of the STM32 Nucleo microcontroller. This was successful in many ways as the game runs smoothly and uses many peripherals that enhance the playing experience. Furthermore, we did spent some extra time to add small quality of life features that make the game more enjoyable to play.

Not everything went swimmingly, as we kept changing certain parts of our program structure very late into the development cycle. While everything was designed to be as agile as possible, we would still have liked to narrow down one specific design path earlier in the process and stuck with it. This is reflected in the changing flowcharts, block diagrams, and the Gannt Charts. The idea for the game was largely the same from a very early point, but the execution changed many times during development which made it harder to assemble everything and stay on top of all the moving parts.

There are many ways this project could be expanded in future revisions. For one, it would be nice to have seperate high score leaderboards for the singleplayer- and multiplayer modes.
Another addition could be different types of enemy spaceships that all bring their own personal characteristics and challenge the player in different ways.
Lastly, there could also be a mode where two players have to compete directly against each other instead of working together to see who can achieve the highest score in the game.

The main takeaway from this project is definitely how to structure larger projects, and we all leave this project with more experience in organizing, developing, and documenting software development cycles.

References

[1] ST (2017).*RM0365 Reference Manual*.

[2] Love, T. (1996), *ANSI C for Programmers on UNIX Systems* Cambridge University Engeneering Department.

[3] Yates, R (2007) *Fixed-Point Arithmetic: An Introduction* Digital Signal Labs.

11 Appendix

Source files

```
#include "stdio.h"
#include "string.h"
#include "ansi.h"
#define ESC 0x1B

void fgcolor(uint8_t foreground) {
/*  Value      foreground    Value    foreground
    -------------------------------------------------
      0        Black          8       Dark Gray
      1        Red            9       Light Red
      2        Green         10       Light Green
      3        Brown         11       Yellow
      4        Blue          12       Light Blue
      5        Purple        13       Light Purple
      6        Cyan          14       Light Cyan
      7        Light Gray    15       White
*/
  uint8_t type = 22;          // normal text
   if (foreground > 7) {
     type = 1;              // bold text
      foreground -= 8;
   }
  printf("%c[%d;%dm", ESC, type, foreground+30);
}

void bgcolor(uint8_t background) {
/*
    Value      Color
    ------------------
      0        Black
      1        Red
      2        Green
      3        Brown
      4        Blue
      5        Purple
      6        Cyan
      7        Gray
*/
  printf("%c[%dm", ESC, background+40);
}

void color(uint8_t foreground, uint8_t background) {
// combination of fgcolor() and bgcolor() - uses less bandwidth
  uint8_t type = 22;          // normal text
   if (foreground > 7) {
     type = 1;              // bold text
      foreground -= 8;
   }
  printf("%c[%d;%d;%dm", ESC, type, foreground+30, background+40);
```

```
}

void resetbgcolor() {
// gray on black text, no underline, no blink, no reverse
  printf("%c[m", ESC);
}

void clrscr(){
   //Clear screen and home cursor. (Doesnt home for putty 0.7)
   printf("%c[2J",ESC);
}

void clreol(){
   printf("%c[2K",ESC);
}

void gotoxy(uint8_t x, uint8_t y){
   printf("%c[%d;%dH",ESC,y,x);
}

void underline(uint8_t on){
   if(on != 0){
      printf("%c[4m",ESC);
   }else{
      printf("%c[24m",ESC);
   }
}

void blink(uint8_t on){
   if(on != 0){
      printf("%c[5m",ESC);
   }else{
      printf("%c[25m",ESC);
   }
}

void inverse(uint8_t on){
   if(on != 0){
      printf("%c[7m",ESC);
   }else{
      printf("%c[27m",ESC);
   }
}

void window(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, char title[]){
   //Top left corner
   gotoxy(x1,y1);
   printf("%c",201);
   //Top right corner
   gotoxy(x2,y1);
   printf("%c",187);
   //Bottom right corner
   gotoxy(x2,y2);
```

```
   printf("%c",188);
   //Bottom left corner
   gotoxy(x1,y2);
   printf("%c",200);

   // Borders for title:
   gotoxy(x1+1,y1);
   printf("%c",185);


   char windowTekst[] = " ";
   strcat(windowTekst, title);
   strcat(windowTekst, " ");

   inverse(1);
   for(int i = 0; i < x2-x1-2; i++)
   {
      gotoxy(x1+2+i,y1);
      if(i < strlen(windowTekst)){
         printf("%c",windowTekst[i]);
      }
      else if(i == strlen(windowTekst)){
         inverse(0);
         printf("%c",204);
      }
      else{
         printf("%c",205);
      }
   }
   inverse(0);
   for(int i = x1+1; i < x2; i++){
      gotoxy(i,y2);
      printf("%c",205);
   }
   for(int i = y1+1; i < y2; i++){
      gotoxy(x1,i);
      printf("%c",186);
      gotoxy(x2,i);
      printf("%c",186);
   }
}

void moveCursorX(uint8_t X, uint8_t UP){ //GOING RIGHT/LEFT
    printf("%c[%d%c", ESC, X, UP ? 'C' : 'D');
}

void moveCursorY(uint8_t Y, uint8_t UP){ //GOING UP/DOWN
    printf("%c[%d%c", ESC, Y, UP ? 'A' : 'B');
}

void invisibleCursor(){
   printf("%c[?25l", ESC);
}
```

Listing 1: ansi.c

```
#include "entity.h"
#include "bullet.h"

void initBullet(bullet_t *bullet, entity_t *entity, uint8_t type, uint8_t
    friendly)
{
   bullet->entity = entity;
   bullet->type = type;
   bullet->friendly = friendly;
}
```

Listing 2: bullet.c

```
#include "enemyManager.h"
#include "entity.h"
#include "bullet.h"
#include "entityHandler.h"
#include <stdlib.h>
#include "scoreCalc.h"
#include "buzz.h"
#include "bulletManager.h"

/*
 * use malloc to allocate space for initial entity references.
 * This only happens once on startup of program so it shouldnt be a
     problem that we dont free it
 */
void initBulletManager(bulletManager_t *bulletManager, bullet_t *bulArr)
{
   for(int i = 0; i < BULLET_ARR_LENGTH; i++)
   {
      bulletManager->bulletArray[i] = &(bulArr[i]);
      bulletManager->bulletArray[i]->entity =
          (entity_t*)malloc(sizeof(entity_t));
      bulletManager->bulletArray[i]->entity->isActive = 0;
   }

}

/*
 * instantiates a bullet by finding a nonactive entity in the bullet
     indexes and
 * using that entity with a nonactive bullet that it finds to make the
     new bullet
 */
void spawnBullet(bulletManager_t *bulletManager,entityHandler_t
    *entHan,int16_t xPos, int16_t yPos,int32_t xVel,int32_t yVel,uint8_t
    fixedVel,uint8_t bulletType, uint8_t height, uint8_t friendly)
{
   for(uint8_t i = BULLET_ARR_LENGTH; i < ENTITY_ARR_LEN; i++)
```

```
   {
      if(!(entHan->entityArray[i]->isActive))
      {
         initEntity(entHan->entityArray[i],6+bulletType,xPos,yPos,xVel,
             yVel,fixedVel,height,0);

         for(uint8_t j = 0; j < BULLET_ARR_LENGTH; j++)
         {
            if(!(bulletManager->bulletArray[j]->entity->isActive))
            {
               initBullet(bulletManager->bulletArray[j],
                   entHan->entityArray[i],bulletType, friendly);
               break;
            }
         }
         entHan->entityArray[i]->isActive = 1;
         break;
      }
   }

   //if no empty spot is found the new bullet is simply not spawned
}

//bullets should collide with asteroids and enemy ships. We check if
    player collide with bullets in player.
//bullets should only collide if same height as the other object.
void checkBulletCollision(bulletManager_t *bulletManager, entityHandler_t
    *entityHandler, gamescore_t *score,uint8_t *playerPowerUp)
{
   uint8_t v;
   uint8_t w;
   for(w = 0; w < BULLET_ARR_LENGTH; w++)
   {
      if(bulletManager->bulletArray[w]->entity->isActive)
      {
         for(v = 1; v <= ENEMY_ARR_LENGTH; v++) //only check enemies not
             player
         {
            if(entityHandler->entityArray[v]->isActive)
            {
               if(detectEntityCollision(bulletManager->bulletArray[w]
                   ->entity, entityHandler->entityArray[v]))
               {
                  if(bulletManager->bulletArray[w]->type == 1 ||
                      (bulletManager->bulletArray[w]->entity->height >=
                      entityHandler->entityArray[v]->height &&
                      bulletManager->bulletArray[w]->entity->height <=
                      entityHandler->entityArray[v]->height + 2))
                  {
                     if(bulletManager->bulletArray[w]->friendly)
                     {
                        if(entityHandler->entityArray[v]->spriteIndex == 2)
                            //enemy spaceship hit
```

```
                    {
                        incrementScore(score, 2222);
                    }
                    else //asteroid hit
                    {
                        incrementScore(score, 1111);
                    }
                    setFreq(100); //buzzer noise

                    //give powerup
                    if(entityHandler->entityArray[v]->powerType)
                        *playerPowerUp =
                        entityHandler->entityArray[v]->powerType;
                }

                //mega bullets pierce (they dont destroy themselves)
                if(bulletManager->bulletArray[w]->type == 0)
                    destroyEntity(bulletManager->bulletArray[w]->entity);
                destroyEntity(entityHandler->entityArray[v]);
            }
        }
    }
  }
}
```

Listing 3: bulletManager.c

```
#include "buzz.h"

void initBuzz(){
  RCC->APB1ENR |= RCC_APB1Periph_TIM2; // Enable clock line to timer 2
  RCC->AHBENR |= RCC_AHBPeriph_GPIOB; // Enable clock for GPIO Port B

  TIM2->CR1 = 0x0000; // Configure timer 2 and disable counter
  TIM2->PSC = 1023; // Set prescale value
  //assuming a systemclock of 64mhz this reload value and prescale will
     result in a 1/100 seconds upcounting-mode timeout period
  TIM2->CR1 = 0x0001; // Configure timer 15 and enable counter

  TIM2->CCER &= ~TIM_CCER_CC3P; // Clear CCER register
  TIM2->CCER |= 0x00000001 << 8; // Enable OC3 output
  TIM2->CCMR2 &= ~TIM_CCMR2_OC3M; // Clear CCMR2 register
  TIM2->CCMR2 &= ~TIM_CCMR2_CC3S;
  TIM2->CCMR2 |= TIM_OCMode_PWM1; // Set output mode to PWM1
  TIM2->CCMR2 &= ~TIM_CCMR2_OC3PE;
  TIM2->CCMR2 |= TIM_OCPreload_Enable;

  //Set pin PB10 to alternate
  GPIOB->MODER &= ~(0x00000003 << (10 * 2)); // Clear mode register
  GPIOB->MODER |= (0x00000002 << (10 * 2)); // Set mode register (0x00
     Input, 0x01 - Output, 0x02 - Alternate Function, 0x03 - Analog
```

```
      in/out)
   GPIOB->PUPDR &= ~(0x00000003 << (10 * 2)); // Clear push/pull register
   GPIOB->PUPDR |= (0x00000002 << (10 * 2)); // Set push/pull register
      (0x00 - No pull, 0x01 - Pull-up, 0x02 - Pull-down)

   GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_1); //Map PWM to PB10
}

void setFreq(uint16_t freq) {
   uint32_t reload;

   if(!freq){
      reload = 64e6; //64*10^6
   } else {
      reload = ((64e6 / freq) / (2000 + 1)) - 1;
   }

   TIM2->ARR =  reload;  // Set auto reload value
   TIM2->CCR3 = reload/2; // Set compare register
   TIM2->EGR |= 0x01;
}

void turnOffBuzz(){
   static uint16_t counter = 0;

   if(TIM2->ARR != 64e6){ //
      counter++;
   }

   if (counter == 1){ //Wait for 100 ms
      counter = 0;
      setFreq(0);
   }
}
```

Listing 4: buzz.c

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include <string.h>
#include "controller.h"

void initController(){
   RCC->AHBENR |= RCC_AHBPeriph_GPIOA; // Enable clock for GPIO Port A
   RCC->AHBENR |= RCC_AHBPeriph_GPIOB; // Enable clock for GPIO Port B
   RCC->AHBENR |= RCC_AHBPeriph_GPIOC; // Enable clock for GPIO Port C


   // Set pin PA4 to input
   GPIOA->MODER |= (0x00000000 << (4 * 2)); // Set mode register (0x00
      Input, 0x01 - Output, 0x02 - Alternate Function, 0x03 - Analog
      in/out)
   GPIOA->PUPDR &= ~(0x00000003 << (4 * 2)); // Clear push/pull register
```

```
    GPIOA->PUPDR |= (0x00000002 << (4 * 2)); // Set push/pull register
        (0x00 - No pull, 0x01 - Pull-up, 0x02 - Pull-down)

    // Set pin PB0 to input
    GPIOB->MODER |= (0x00000000 << (0 * 2)); // Set mode register (0x00
        Input, 0x01 - Output, 0x02 - Alternate Function, 0x03 - Analog
        in/out)
    GPIOB->PUPDR &= ~(0x00000003 << (0 * 2)); // Clear push/pull register
    GPIOB->PUPDR |= (0x00000002 << (0 * 2)); // Set push/pull register
        (0x00 - No pull, 0x01 - Pull-up, 0x02 - Pull-down)

    // Set pin PC10
    GPIOC->MODER &= ~(0x00000003 << (10 * 2)); // Clear mode register
    GPIOC->MODER |= (0x00000000 << (10 * 2)); // Set mode register (0x00
        Input, 0x01 - Output, 0x02 - Alternate Function, 0x03 - Analog
        in/out)
    GPIOC->PUPDR &= ~(0x00000003 << (10 * 2)); // Clear push/pull register
    GPIOC->PUPDR |= (0x00000002 << (10 * 2)); // Set push/pull register
        (0x00 - No pull, 0x01 - Pull-up, 0x02 - Pull-down)

    // Set pin PC12
    GPIOC->MODER &= ~(0x00000003 << (12 * 2)); // Clear mode register
    GPIOC->MODER |= (0x00000000 << (12 * 2)); // Set mode register (0x00
        Input, 0x01 - Output, 0x02 - Alternate Function, 0x03 - Analog
        in/out)
    GPIOC->PUPDR &= ~(0x00000003 << (12 * 2)); // Clear push/pull register
    GPIOC->PUPDR |= (0x00000002 << (12 * 2)); // Set push/pull register
        (0x00 - No pull, 0x01 - Pull-up, 0x02 - Pull-down)

    //Pots
    RCC->CFGR2 &= ~RCC_CFGR2_ADCPRE12; // Clear ADC12 prescaler bits
    RCC->CFGR2 |= RCC_CFGR2_ADCPRE12_DIV6; // Set ADC12 prescaler to 6
    RCC->AHBENR |= RCC_AHBPeriph_ADC12; // Enable clock for ADC12

    ADC1->CR = 0x00000000; // Clear CR register
    ADC1->CFGR &= 0xFDFFC007; // Clear ADC1 config register
    ADC1->SQR1 &= ~ADC_SQR1_L; // Clear regular sequence register 1

    ADC1->CR |= 0x10000000; // Enable internal ADC voltage regulator
    for (int i = 0 ; i < 1000 ; i++) {} // Wait for about 16 microseconds

    ADC1->CR |= 0x80000000; // Start ADC1 calibration
    while (!(ADC1->CR & 0x80000000)); // Wait for calibration to finish
    for (int i = 0 ; i < 100 ; i++) {} // Wait for a little while

    ADC1->CR |= 0x00000001; // Enable ADC1 (0x01 - Enable, 0x02 - Disable)
    while (!(ADC1->ISR & 0x00000001)); // Wait until ready
}

uint16_t readPot1(){
    ADC_RegularChannelConfig(ADC1, ADC_Channel_5, 1,
        ADC_SampleTime_1Cycles5);
    ADC_StartConversion(ADC1); // Start ADC read
```

```
   while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == 0); // Wait for ADC
      read

   return ADC_GetConversionValue(ADC1); // Read the ADC value
}

uint16_t readPot2(){
   ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1,
      ADC_SampleTime_1Cycles5);
   ADC_StartConversion(ADC1); // Start ADC read
   while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == 0); // Wait for ADC
      read

   return ADC_GetConversionValue(ADC1); // Read the ADC value
}

//used for debugging
void potsToString(char array[]){
   sprintf(array, "Pot1: %04hd | Pot2: %04hd\n", readPot1(), readPot2());
}

uint8_t readButtons(){
   uint8_t temp = 0;

   //The two buttons get returned as 000000XY
   temp |= (GPIOC->IDR & (0x0001 << 10)) >> 10; //bit 0
   temp |= (GPIOC->IDR & (0x0001 << 12)) >> 11; //bit 1

   return temp;
}
```

Listing 5: controller.c

```
#include "controller.h"
#include "controllerAPI.h"

#define DEADZONE 300
#define CENTER_X 1230
#define CENTER_Y 1310

uint8_t readJoystick(){
   uint16_t pot1 = readPot1();
   uint16_t pot2 = readPot2();

   /*
    *  UP LEFT   = 0x80
    *  UP RIGHT  = 0x02
    *  UP        = 0x01
    *
    *  DOWN LEFT = 0x20
    *  DOWN RIGHT = 0x08
    *  DOWN      = 0x10
    *
```

```
    *  LEFT      = 0x40
    *  RIGHT     = 0x04
    *  CENTER    = 0x00
    */

   if(pot2 >= CENTER_Y + DEADZONE){
      if(pot1 <= CENTER_X - DEADZONE){
         return 0x80; //UP LEFT
      } else if(pot1 >= CENTER_X + DEADZONE){
         return 0x02; //UP RIGHT
      } else {
         return 0x01; //UP
      }
   } else if(pot2 <= CENTER_Y - DEADZONE) {
      if(pot1 <= CENTER_X - DEADZONE){
         return 0x20; //DOWN LEFT
      } else if(pot1 >= CENTER_X + DEADZONE){
         return 0x08; //DOWN RIGHT
      } else {
         return 0x10; //DOWN
      }
   } else {
      if(pot1 <= CENTER_X - DEADZONE){
         return 0x40; //LEFT
      } else if(pot1 >= CENTER_X + DEADZONE){
         return 0x04; //RIGHT
      } else {
         return 0x00; //CENTER
      }
   }

   return 0x00;
}

uint8_t readButton1(){
   return readButtons() & 0x01;
}

uint8_t readButton2(){
   return readButtons() >> 1;
}
```

Listing 6: controllerAPI.c

```
#include "bullet.h"
#include "util.h"
#include "entity.h"
#include "bulletManager.h"
#include "entityHandler.h"
#include "PuTTYSprites.h"
#include "vec.h"
#include "enemy.h"
```

```
void initEnemy(entity_t * entity, enemy_t * enemy, uint8_t type)
{
   enemy->entity = entity;
   enemy->type = type; // = 0 for spaceship, = 1 for 1x1 asteroid, = 2
       for 2x2 asteroid and = 3 for 3x3 asteroid
}

void enemyShoot(bulletManager_t *bulletManager,entityHandler_t
    *entHan,enemy_t * enemy,int32_t bulletSpeed)
{
   if(getRandomInterval(0,100) >= 98) //2% chance at shooting each tick
   {
      spawnBullet(bulletManager, entHan,
          getXint(&(enemy->entity->pos))+1,
          getYint(&(enemy->entity->pos))+4,0,bulletSpeed,1,0,
          enemy->entity->height, 0);
   }
}
```

Listing 7: enemy.c

```
#include "bullet.h"
#include "util.h"
#include "entity.h"
#include "bulletManager.h"
#include "entityHandler.h"
#include "PuTTYSprites.h"
#include "vec.h"
#include "enemy.h"

void initEnemy(entity_t * entity, enemy_t * enemy, uint8_t type)
{
   enemy->entity = entity;
   enemy->type = type; // = 0 for spaceship, = 1 for 1x1 asteroid, = 2
       for 2x2 asteroid and = 3 for 3x3 asteroid
}

void enemyShoot(bulletManager_t *bulletManager,entityHandler_t
    *entHan,enemy_t * enemy,int32_t bulletSpeed)
{
   if(getRandomInterval(0,100) >= 98) //2% chance at shooting each tick
   {
      spawnBullet(bulletManager, entHan,
          getXint(&(enemy->entity->pos))+1,
          getYint(&(enemy->entity->pos))+4,0,bulletSpeed,1,0,
          enemy->entity->height, 0);
   }
}
```

Listing 8: enemyManager.c

```
#include "PuTTYSprites.h"
#include "util.h"
#include "entity.h"
```

```
//moves a given entity by its velocity. (both in fixed point so easy to
    add).
void move(entity_t * ptr){
   ptr->pos.x += ptr->vel.x;
   ptr->pos.y += ptr->vel.y;
}

//updates the velocity of a given struct.
void updateVel(entity_t * ptr, int8_t x, int8_t y){
   updateVectorInt(&(ptr->vel),x,y);
}

//function to initialize the values of an entity struct
void initEntity(entity_t * ptr, uint8_t spriteIndex, uint8_t xPos,
    int16_t yPos,int32_t xVel,int32_t yVel, uint8_t fixedVel, uint8_t
    height,uint8_t powerType){
   setEntityPos(ptr,xPos,yPos);
   if(fixedVel)
   {
      setEntityVelFixed(ptr,xVel,yVel);
   }
   else
   {
      setEntityVel(ptr,xVel,yVel);
   }
   ptr->height = capInterval(height,0,3);
   ptr->spriteIndex = capInterval(spriteIndex,0,7);
   ptr->isActive = 0;
   ptr->preX = getXint(&(ptr->pos));
   ptr->preY = getYint(&(ptr->pos));
   ptr->powerType = powerType;
}

void setEntityVel(entity_t * ptr, int8_t x, int8_t y){
   setVectorInt(&(ptr->vel),x,y);
}

void setEntityVelFixed(entity_t *ptr, int32_t x, int32_t y)
{
   ptr->vel.x = x;
   ptr->vel.y = y;
}

void setEntityPos(entity_t * ptr, int16_t x, int16_t y)
{
   setVectorInt(&(ptr->pos),x,y);
}

//function to "destroy" an entity by setting it inactive.
void destroyEntity(entity_t * ptr)
{
   clearEntity(ptr);
```

```
   ptr->isActive = 0;
}

//draws the entity in PuTTY
void drawEntity(entity_t * ptr, uint8_t powerType)
{
   uint8_t color = 15;
   switch(powerType)
   {
      case 1:
         color = 2; //green medkit
         break;
      case 2:
         color = 4; //blue shield
         break;
      case 3:
         color = 1; //red megabullet
         break;
   }

   ui_draw_sprite(ptr->spriteIndex, color, 0, getXint(&(ptr->pos)),
       getYint(&(ptr->pos)));
}

//Clears the exact PuTTY chars that make up the entity
void clearEntity(entity_t * ptr)
{
   ui_clear_sprite(ptr->spriteIndex, 15, 0, ptr->preX, ptr->preY);
}

//calculate gravity between two entities (bullet and an asteroid)
//using fixed point 18x14
void calculateGravity(entity_t * bullet, entity_t * solidObj){
   int32_t x1 = bullet->pos.x;
   int32_t y1 = bullet->pos.y;
   int32_t x2 = solidObj->pos.x;
   int32_t y2 = solidObj->pos.y;
   int32_t massObj;
   int32_t deltaX;
   int32_t deltaY;
   int32_t dist;

   switch(solidObj->spriteIndex){
   case 3:
      massObj = 1;
      x2 += 3 << 13;
      y2 += 1 << 14;
      break;
   case 4:
      massObj = 3;
      x2 += 5 << 13;
      y2 += 3 << 13;
      break;
```

```
   case 5:
      massObj = 5;
      x2 += 7 << 13; //add 1.5 to center the 3x3 asteroid
      y2 += 2 << 14; //add 1.5 to center the 3x3 asteroid
      break;
   default:
      massObj = 0;
   }

   if(x1 == x2 && y1 == y2) return;

   dist = getManDistance(x1>>14, y1>>14, x2>>14, y2>>14); //int dist
   if(dist > 20) return; //max range = 20
   dist = (dist < 2 ? 2 : dist); //shouldnt be able to get too close

   deltaX = ((norm(x2-x1)<<14) * ( ((G*massObj) << 28) / (dist*dist <<
       14) )) >> 14;
   deltaY = ((norm(y2-y1)<<14) * ( ((G*massObj) << 28) / (dist*dist <<
       14) )) >> 14;

   //Finally add the gravity to our velocity vector for our bullet
   bullet->vel.x += deltaX; //both fixed point
   bullet->vel.y += deltaY;
}

//Returns the x coordinate for the center of the entities sprite
int32_t centeredXPOS(entity_t * ptr)
{
   switch(ptr->spriteIndex)
   {
      case(0): //add 2
      case(1): //add 2
      case(2): //add 2
         return ptr->pos.x + (2 << 14);
      case(5): //add 3.5
         return ptr->pos.x + (7 << 13);
      case(3): //add 1.5
         return ptr->pos.x + (3 << 13);
      case(6): //add 0.5
         return ptr->pos.x + (1 << 13);
      case(4): //add 2.5
         return ptr->pos.x + (5 << 13);
      case(7): //add 1
         return ptr->pos.x + (1 << 14);
      default:
         return ptr->pos.x;
   }
}

//Returns the y coordinate for the center of the entities sprite
int32_t centeredYPOS(entity_t * ptr)
{
   switch(ptr->spriteIndex)
```

```
   {
      case(0): //add 2
      case(1): //add 2
      case(5): //add 2
         return ptr->pos.y + (2 << 14);
      case(2): //add 1.5
      case(4): //add 1.5
         return ptr->pos.y + (3 << 13);
      case(3): //add 1
      case(7): //add 1
         return ptr->pos.y + (1 << 14);
      case(6): //add 0.5
         return ptr->pos.y + (1 << 13);
      default:
         return ptr->pos.y;
   }
}

/*
 *
 */
uint8_t detectEntityCollision(entity_t * obj1, entity_t * obj2)
{
   int32_t x1_C = centeredXPOS(obj1);
   int32_t y1_C = centeredYPOS(obj1);
   int32_t x2_C = centeredXPOS(obj2);
   int32_t y2_C = centeredYPOS(obj2);
   int32_t minX1, minY1, minX2, minY2; //radius

   //switch statement to set minX1 and minY1 values
   switch(obj1->spriteIndex){
      case(0): // 4x4
      case(1): // 4x4
         minX1 = 2 << 14; //2
         minY1 = 2 << 14; //2
         break;
      case(6): //1x1
         minX1 = 1 << 13; //0.5
         minY1 = 1 << 13; //0.5
         break;
      case(7): //3x1
         minX1 = 3 << 13; //1.5
         minY1 = 1 << 13; //0.5
         break;
      default:
         return 0;
   }
   //switch statement to set minX2 and minY2 values
   switch(obj2->spriteIndex){
      case(2): //4x3
         minX2 = 2 << 14; //2;
         minY2 = 1 << 14; //1;
         break;
```

```
      case(5): //7x4
         minX2 = 7 << 13; //3.5;
         minY2 = 2 << 14; //2;
         break;
      case(3): //3x2
         minX2 = 3 << 13; //1.5
         minY2 = 1 << 14; //1
         break;
      case(4): //5x3
         minX2 = 5 << 13; //2.5
         minY2 = 3 << 13; //1.5
         break;
      case(6): //1x1
         minX2 = 1 << 13; //0.5
         minY2 = 1 << 13; //0.5
         break;
      default:
         return 0;
   }

   //if the peripherys collide
   if(absolute(x2_C - x1_C) <= minX1+minX2 && absolute(y2_C - y1_C) <=
       minY1+minY2) {
      return 1;
   }
   else return 0;
}

//Destroys entities out of bounds
void checkEntityPos(entity_t * ptr){
   int32_t x = ptr->pos.x >> 14;
   int32_t y = ptr->pos.y >> 14;

   //should create a bit of a bufferzone around our actual screen, making
       sure that the objects can pass seamlessly.
   if(x < 0 || x > 81 || y > 47 || y < -3) destroyEntity(ptr);
}
```

Listing 9: entity.c

```
#include "entity.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include "entity.h"
#include "vec.h"
#include "enemyManager.h"
#include "entityHandler.h"
#include "bulletManager.h"

// to initialize our entityhandler make it point to the actual array of
    entities to be held in the gameHandler
void init_entityHandler(entityHandler_t * ptr, entity_t * entArr){
```

```
   uint8_t i;

   for(i = 0; i < ENTITY_ARR_LEN; i++)
   {
      ptr->entityArray[i] = &(entArr[i]);
      ptr->entityArray[i]->isActive = 0;
   }
}

// for each entity in our entityHandler array - move the entity,
    applygravity and check of out of bounds
void updateEntities(entityHandler_t * ptr)
{
   uint8_t i;

   applyGravity(ptr);

   for(i = 0; i < ENTITY_ARR_LEN; i++)
   {
      if(ptr->entityArray[i]->isActive)
      {
         ptr->entityArray[i]->preX = getXint(&(ptr->entityArray[i]->pos));
         ptr->entityArray[i]->preY = getYint(&(ptr->entityArray[i]->pos));

         if(i == 0) //make sure player cant move outside the edges
         {
            if(withinBoundry(ptr->entityArray[0]))
               move(ptr->entityArray[0]);
            continue;
         }
         move(ptr->entityArray[i]);
         checkEntityPos(ptr->entityArray[i]);
      }
   }
}

//Draws all active entities whose position changed
void drawAllEntities(entityHandler_t * ptr){
   uint8_t i;
   for(i = 1; i < ENTITY_ARR_LEN; i++) //all except player
   {
      if(ptr->entityArray[i]->isActive && (ptr->entityArray[i]->preX !=
          getXint(&(ptr->entityArray[i]->pos)) ||
          ptr->entityArray[i]->preY !=
          getYint(&(ptr->entityArray[i]->pos))))
      {
         drawEntity(ptr->entityArray[i],ptr->entityArray[i]->powerType);
      }
   }
}

//Clears all active entities whose position changed
void clearAllEntities(entityHandler_t * ptr){
```

```
   uint8_t i;
   for(i = 1; i < ENTITY_ARR_LEN; i++) //all except player
   {
      if(ptr->entityArray[i]->isActive && (ptr->entityArray[i]->preX !=
          getXint(&(ptr->entityArray[i]->pos)) ||
          ptr->entityArray[i]->preY !=
          getYint(&(ptr->entityArray[i]->pos))))
      {
         clearEntity(ptr->entityArray[i]);
      }
   }
}

//Gravity between each bullet and each asteroid in range
void applyGravity(entityHandler_t * ptr)
{
   uint8_t i;
   uint8_t j;
   for(i = BULLET_ARR_LENGTH; i < ENTITY_ARR_LEN; i++) //checks all
       bullet entities
   {
      if(ptr->entityArray[i]->isActive &&
          ptr->entityArray[i]->spriteIndex == 6)
      {
         for(j = 1; j <= ENEMY_ARR_LENGTH; j++) //access the enemy part of
             entityarray
         {
            if(ptr->entityArray[j]->isActive &&
                (ptr->entityArray[j]->spriteIndex >= 3 &&
                ptr->entityArray[j]->spriteIndex <= 5)) //only active
                asteroids
            {
               calculateGravity(ptr->entityArray[i], ptr->entityArray[j]);
            }
         }
      }
   }
}

//used to keep the player within the screen
uint8_t withinBoundry(entity_t *ptr)
{
   if(getXint(&(ptr->pos))+getXint(&(ptr->vel)) > 76 ||
       getXint(&(ptr->pos))+getXint(&(ptr->vel)) < 1 ||
       getYint(&(ptr->pos))+getYint(&(ptr->vel)) < 1 ||
       getYint(&(ptr->pos))+getYint(&(ptr->vel)) > 42) return 0;
   else return 1;
}
```

Listing 10: entityHandler.c

```
#include <LED.h>
#include "PuttyLCDConverter.h"
```

```
#include "stdint.h"
#include "ansi.h"
#include "controller.h"
#include "controllerAPI.h"
#include "entityHandler.h"
#include "bulletManager.h"
#include "enemyManager.h"
#include "player.h"
#include "LCD.h"
#include "stm32f30x.h"
#include "stopwatch.h"
#include "scoreCalc.h"
#include "menus.h"
#include "menusAPI.h"
#include "serialRead.h"
#include "highscore.h"
#include "gameUI.h"
#include <stdio.h>
#include <stdlib.h>
#include "sinLut.h"
#include "buzz.h"
#include "gameHandler.h"

uint8_t game_update()
{
   if(timer_Flag == 1)
   {
      timer_Flag = 0;
      return 1;
   }

   return 0;
}

void initProgram(gameStruct_t * gs_p){
   //Initialize UART
   uart_init(1024000);
   invisibleCursor();
   uart_clear();
   color(15, 0);
   clrscr();

   //Initialize HARDWARE
   initTimer();
   initController();
   srand( (unsigned)(lutSin(readPot1()) + lutCos(readPot2())) );
      //comment to debug same seed
   initLCD();
   lcd_clear_all(gs_p->LCDbuffer,0x00);
   lcd_push_buffer(gs_p->LCDbuffer);
   initLED();
   initBuzz();
   setLED(0, 0, 0);
```

```
   //Initialize gamestruct values
   gs_p->tickCounter = 0;
   gs_p->mode = 0;
   gs_p->prevMode = 1;
   gs_p->gameInitialized = 0;
   gs_p->playerNum = 0;
   gs_p->cooldownCounter = 0;


   //INIT TOP LEVEL STRUCTS
   init_entityHandler(&(gs_p->entHan),gs_p->entityArray);
   initBulletManager(&(gs_p->bulMan),gs_p->bulletArray);
   initEnemyManager(&(gs_p->enemMan),gs_p->enemyArray);
}

void modeSelect(gameStruct_t * gs_p)
{
   char input = get_key_pressed();
   turnOffBuzz();
   if(MODE_CHANGE){
      newMode(gs_p);
   }
   if(gs_p->mode == 1 || gs_p->mode == 2){
      if(gs_p->gameInitialized == 0) initializeGame(gs_p);
      setLEDSide(gs_p);
      runGame(gs_p, input);
      updateGameUI(&(gs_p->player), &(gs_p->score),gs_p->gameSpeed);
   }
   gs_p->mode = modePicker(gs_p->mode, input, gs_p);
}

void newMode(gameStruct_t * gs_p){
   setLED(0, 0, 0);
   color(15, 0);
   uart_clear();
   clrscr();
   gs_p->prevMode = gs_p->mode;
   switch(gs_p->mode){
   case(0):
      initMainMenu();
      break;
   case(1):
   case(2):
      initGameUI(&(gs_p->player),gs_p->gameSpeed);
      gs_p->playerNum = gs_p->mode;
      break;
   case(3):
      helpMenu(gs_p->gameInitialized);
      break;
   case(4): // BOSS SCREEN
      bossScreen();
      break;
```

```
   case(5): // GAME OVER
      initGameOverScreen(gs_p);
      lcd_game_over(gs_p->LCDbuffer);
      break;
   default:
      break;
   }
}

void initializeGame(gameStruct_t * gs_p){
   //INIT CONTROL VARIABLES
   gs_p->spawnCounter = 0;
   gs_p->gameInitialized = 1;
   gs_p->gameSpeed = 0;
   gs_p->ticks = 0;
   initScore(&(gs_p->score));

   //ADD PLAYER - WITH SET NUMBER OF PLAYERS.
   initEntity(&(gs_p->entityArray[0]),0,40,23,0,0,0,0,0);
   gs_p->entityArray[0].isActive = 1;
   initPlayer(&(gs_p->entityArray[0]), &(gs_p->player), gs_p->playerNum);

   //INIT LCD
   lcd_clear_all(gs_p->LCDbuffer,0x00);
   lcd_push_buffer(gs_p->LCDbuffer);

   for(uint8_t i = 1; i < ENTITY_ARR_LEN; i++)
   {
      gs_p->entityArray[i].isActive = 0;
   }
   return;
}

uint8_t modePicker(uint8_t mode, char input, gameStruct_t * gs_p){
   static uint8_t activeItem = 1;

   switch(mode){
      case 0: //MAIN MENU
         activeItem += pickMainMenuItems(input, activeItem);
         printMainMenuItems(activeItem);

         if(input == ' '){
            return activeItem;
         }
         break;
      case 1: //SINGLEPLAYER
      case 2: //MULTIPLAYER
         if(input == 'h') return 3;
         else if(input == 0x1B) //ESC
         {
            gs_p->gameInitialized = 0;
            return 0;
         } else if(input == 'b' || input == 'B') return 4;
```

```
        if(gs_p->player.HP <= 0) return 5;
        break;
      case 3: //HELP MENU
        if(input == 'm')
        {
          gs_p->gameInitialized = 0;
          return 0;
        } else if(input == 0x1B && gs_p->gameInitialized){ //ESC
          return gs_p->playerNum;
        } else if(input == 0x2A){ //'*'
          highscoreFlush();
        }
        break;
      case 4: //BOSS KEY
        if(input == 'b' || input == 'B'){
          return gs_p->playerNum;
        }
        break;
      case 5: //GAME OVER
        gameOverScreen(gs_p, input);
        if(input == ' ')
        {
          gs_p->gameInitialized = 0;
          return 0;
        }
        break;
      default :
        return mode;
    }
    return mode;
}

void updateGameSpeed(gameStruct_t * gs_p)
{
    //tick up every minute:
    gs_p->gameSpeed = capInterval(gs_p->ticks / 600,0,15);
}

void runGame(gameStruct_t * gs_p, char input)
{
    //Clear
    clearPlayer(&(gs_p->player));
    lcd_clear_all(gs_p->LCDbuffer,0x00);

    //update playeractions based on inputs:
    usePlayerActionsFromInput(gs_p, input);

    //update entities:
    updatePlayerVel(&(gs_p->player), input);
    updateEntities(&(gs_p->entHan));
    checkBulletCollision(&(gs_p->bulMan),&(gs_p->entHan),
        &(gs_p->score),&(gs_p->player.powerUp));
    checkPlayerCollision(&(gs_p->player),&(gs_p->entHan));
```

```
   //Create new entities
   enemiesShoot(&(gs_p->bulMan),&(gs_p->entHan),&(gs_p->enemMan),(1 <<
       14) + ((gs_p->gameSpeed << 14)*(9 << 7) >> 14)); //bulletSpeed is 1
       + 0.07*gameSpeed
   if(gs_p->spawnCounter >= 20-gs_p->gameSpeed)
   {
      spawnRandom(&(gs_p->enemMan),&(gs_p->entHan),gs_p->playerNum == 2 ?
          2 : 0, (1 << 12) + ((gs_p->gameSpeed << 14)*(13 << 6) >> 14));
          //enemySpeed is 1/4 + 0.051*gameSpeed
      gs_p->spawnCounter = 0;
   }
   incrementCounter(&(gs_p->spawnCounter), 1);

   //Draw LCD
   if(gs_p->playerNum == 2)
   {
      lcd_draw_crosshair(gs_p->LCDbuffer,gs_p->player.crosshairX,
          gs_p->player.crosshairY);
      con_draw_putty_to_lcd(&(gs_p->enemMan),
          &(gs_p->player),gs_p->LCDbuffer);
      lcd_push_buffer(gs_p->LCDbuffer);
   }

   //DEBUG:
   if(gs_p->player.entity == 0xFFFFFFFF)
   {
      gs_p->player.entity->isActive = 1;
   }

   //Draw PuTTY
   clearAllEntities(&(gs_p->entHan));

   drawAllEntities(&(gs_p->entHan));
   drawPlayer(&(gs_p->player));

   //update score and difficulty
   incrementScore(&(gs_p->score), 100);
   (gs_p->ticks)++;
   updateGameSpeed(gs_p);
}

void usePlayerActionsFromInput(gameStruct_t * gs_p, char input){
   switch(gs_p->playerNum){
      case 1: //SINGLEPLAYER
         //SHOOTING AND CHANGING GUNSIDE
         if(input == ',')
         {
            gs_p->player.gunSide = 1;
            if(!(gs_p->cooldownCounter)){
               gs_p->cooldownCounter = 10;
               setLED(0, 1, 0);
               playerShoot(&(gs_p->player), &(gs_p->bulMan),
```

```
                &(gs_p->entHan),0,0);
          }
          else setLED(1, 0, 0);
       }
       else if(input == '.'){
          gs_p->player.gunSide = -1;
          if(!(gs_p->cooldownCounter)){
             gs_p->cooldownCounter = 10;
             setLED(0, 1, 0);
             playerShoot(&(gs_p->player),&(gs_p->bulMan),
                 &(gs_p->entHan),0,0);
          }
          else setLED(0, 0, 1);
       }
       break;
    case 2: //MULTIPLAYER
       //SHOOTING AND CHANGING GUNSIDE
       updateCrosshair(&(gs_p->player),readJoystick());
       if(readButton2())
       {
          changeGunside(&(gs_p->player));
          if(gs_p->player.gunSide == 1)
          {
             setLED(1, 0, 0);
          } else if(gs_p->player.gunSide == -1){
             setLED(0, 0, 1);
          }
       }
       if(readButton1() && !(gs_p->cooldownCounter))
       {
          gs_p->cooldownCounter = 10;
          setLED(0, 1, 0);
          playerShoot(&(gs_p->player),&(gs_p->bulMan),&(gs_p->entHan),
              0,gs_p->player.crosshairY);
       }
       break;
    default:
       break;
  }
  //USING POWERUP
  if(input == ' ') usePowerUp(&(gs_p->player),&(gs_p->bulMan),
      &(gs_p->entHan));
}
```

Listing 11: gameHandler.c

```
#include <stdint.h>
#include <stdio.h>
#include "ansi.h"
#include "player.h"
#include "scoreCalc.h"
#include "gameUI.h"
```

```
//called when the screen changes to the game
void initGameUI(player_t *player, uint8_t gameLevel)
{
   color(15, 7);

   for(uint8_t i = 1; i <= 46; i++)
   {
      gotoxy(81, i);
      printf("                  ");
   }

   //HP
   color(0, 7);
   gotoxy(90, 2);
   printf("HP");
   showPlayerHealth(player);

   //POWERUP
   color(0, 7);
   gotoxy(88, 12);
   printf("POWER");
   gotoxy(88, 17);
   printf(" NONE ");
   color(0, 0);
   gotoxy(88, 14);
   printf("     ");
   gotoxy(88, 15);
   printf("     ");
   color(0, 7);

   //Game level
   color(0, 7);
   gotoxy(88, 26);
   printf("LEVEL");
   showGameLevel(gameLevel);
   printf("/15");

   //SCORE
   gotoxy(88, 34);
   printf("SCORE");

   //MENU NAVIGATION
   gotoxy(82, 41);
   printf("[ESC] - Main menu");
   gotoxy(84, 42);
   printf("'h' - Help menu");
}

//updates the part of the UI that changed
void updateGameUI(player_t *player, gamescore_t *score,uint8_t gameLevel)
{
   static uint8_t prevHealth = 0;
   static uint8_t prevPowerUp = 0;
```

```
   static uint8_t prevLevel = 0;

   if(player->HP != prevHealth) showPlayerHealth(player);
   if(player->powerUp != prevPowerUp) showPlayerPowerUp(player);
   if(gameLevel != prevLevel) showGameLevel(gameLevel);
   showPlayerScore(score);

   prevHealth = player->HP;
   prevPowerUp = player->powerUp;
   prevLevel = gameLevel;
}

void showPlayerHealth(player_t *player)
{
   color(7, 1);
   switch(player->HP){
      case 4:
         gotoxy(84, 4);
         printf(" ");
         gotoxy(88, 4);
         printf(" ");
         gotoxy(92, 4);
         printf(" ");
         gotoxy(96, 4);
         color(7, 4);
         printf(" ");
         break;
      case 3:
         gotoxy(84, 4);
         printf(" ");
         gotoxy(88, 4);
         printf(" ");
         gotoxy(92, 4);
         printf(" ");
         gotoxy(96, 4);
         color(7, 0);
         printf(" ");
         break;
      case 2:
         gotoxy(84, 4);
         printf(" ");
         gotoxy(88, 4);
         printf(" ");
         gotoxy(92, 4);
         color(7, 0);
         printf(" ");
         gotoxy(96, 4);
         printf(" ");
         break;
      case 1:
         gotoxy(84, 4);
         printf(" ");
         gotoxy(88, 4);
```

```
        color(7, 0);
        printf(" ");
        gotoxy(92, 4);
        printf(" ");
        gotoxy(96, 4);
        printf(" ");
        break;
     default:
        gotoxy(84, 4);
        color(7, 0);
        printf(" ");
        gotoxy(88, 4);
        printf(" ");
        gotoxy(92, 4);
        printf(" ");
        gotoxy(96, 4);
        printf(" ");
        break;
   }
}

void showPlayerPowerUp(player_t *player)
{
   color(0, 7);

   switch(player->powerUp){
      case 1: //MEDKIT
         gotoxy(88, 17);
         printf(" HEAL ");
         color(0, 2);
         break;
      case 2: //SHIELD
         gotoxy(88, 17);
         printf("SHIELD");
         color(0, 4);
         break;
      case 3: //MEGA BULLET
         gotoxy(88, 17);
         printf(" MEGA ");
         color(0, 1);
         break;
      default: //NONE
         gotoxy(88, 17);
         printf(" NONE ");
         color(0, 0);
         break;
   }

   gotoxy(88, 14);
   printf("     ");
   gotoxy(88, 15);
   printf("     ");
}
```

```
void showPlayerScore(gamescore_t *score){
   gotoxy(86, 35);
   color(0, 7);
   printf("%010lu", score->score);
}

void showGameLevel(uint8_t gameLevel)
{
   gotoxy(88,27);
   color(0, 7);
   printf("%02d",gameLevel);
}
```

Listing 12: gameUI.c

```
/*
 * highscore.c
 *
 *  Created on: 12. jun. 2023
 *      Author: frede
 */

#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include <stdlib.h>
#include <string.h>
#include "memory.h"
#include "ansi.h"
#include "highscore.h"

void saveHighscore(char name[], uint32_t score){
   uint16_t oldScores[20];
   uint8_t pos = 0;

   for(uint8_t i = 0; i < 20; i++){ //Copy all the old data into an array
      oldScores[i] = readMemory(i);
   }

   for(uint8_t i = 0; i < 5; i++){ //Check where a new score would fir
       into the array
      uint32_t oldScore = readMemory(i*4) | (readMemory(i*4 + 1) << 16);
      if(oldScore < score){
         break;
      }
      pos++;
   }

   if(pos == 5){ //Do not save if the score does not make the list
      return;
   }

   for(uint8_t i = 4; i > pos; i--){ //Shift all positions below the new
```

```
      score
      //Shift the score
      oldScores[i*4+0] = oldScores[(i-1)*4+0];
      oldScores[i*4+1] = oldScores[(i-1)*4+1];

      //Shift the name
      oldScores[i*4+2] = oldScores[(i-1)*4+2];
      oldScores[i*4+3] = oldScores[(i-1)*4+3];
   }

   //Save the new score
   oldScores[pos*4] = score & 0x0000FFFF;
   oldScores[pos*4 + 1] = score >> 16;

   oldScores[pos*4 + 3] = (name[0] << 8) | name[1];
   oldScores[pos*4 + 2] = (name[2] << 8) | name[3];

   //Write it to memory
   saveToMemory(oldScores, 20);
}

char* readHighscoreName(uint8_t place){
   if(place < 0 || place > 4){
      return NULL;
   }

   //Allocate space for name array
   char *name = malloc(5*sizeof(char));

   //Read name from memory and save to char array
   uint16_t temp = readMemory(4*place + 3);

   name[0] = temp >> 8;
   name[1] = temp & 0x00FF;

   temp = readMemory(4*place + 2);

   name[2] = temp >> 8;
   name[3] = temp & 0x00FF;
   name[4] = '\0';

   return name;
}

uint32_t readHighscore(uint8_t place){
   if(place < 0 || place > 4){
      return 0;
   }

   return readMemory(place*4) | readMemory(place*4 + 1) << 16;
}

void highscoreFlush(){
```

```
   uint16_t oldScores[20] = {0};

   saveToMemory(oldScores, 20);

   char *name = "name";

   uint32_t sco = 1;

   for(uint8_t i = 0; i < 5; i++){
      saveHighscore(name, sco);
   }
}
```

Listing 13: highscore.c

```
/*
 * lcd_sprites.c
 *
 *  Created on: 12. jun. 2023
 *      Author: georg
 */


#include "lcdSprites.h"

/*
 * This is sprites for the LCD. The bytes represent vertical strips in
    the LCD
*/
const uint8_t lcd_sprites[14][4][40] = {
     { // 1/4 x 1/4 Asteroid
        {0x03,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00, 0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00}
     },
     { // 1/2 x 1/2 Asteroid
        {0x06,0x0B,0x0F,0x06,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

```
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00}
  },
  { // 1x1 Asteroid
    {0x3C,0x6E,0x9F,0xFF,0xFB,0xF5,0x7E,0x3C,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00}
  },
  { // 2x2 Asteroid
    {0xF0,0xF8,
        0xFC,0xFE,0x87,0x7B,0x7B,0x7B,0x7B,0x87,0xFF,0xF7,0xFE,
        0xFC,0xF8,0xF0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00},
    {0x0F,0x1F,0x3F,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xCF,0xB7,
        0xB7,0x4F,0x3F,0x1F,0x0F,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00}
  },
  { // 3x3 Asteroid
    {0x80,0xC0,0xE0,0xF0,0xF8,0xFC,0x7E,0x7F,0x7F,0x7F,0xFF,
```

```
        0xFF,0x0F,0xF7,0xF7,0xF7,0xF7,0xF6,0x0C,0xF8,0xF0,0xE0,
        0xC0,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0xFF,0xFF,0xFF,0xFF,0xFF,0xF8,0xF7,0xF7,0xF7,0xF7,0xF8,
        0xFF,0xFE,0xFD,0xFD,0xFD,0xFD,0xFD,0xFE,0xFF,0xFF,0xBF,
        0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x01,0x03,0x07,0x0F,0x1F,0x3F,0x7F,0xFF,0xFF,0xFF,0xE7,
        0xDB,0xDB,0xE7,0xFF,0xFF,0xFF,0x7F,0x3F,0x1F,0x0F,0x07,
        0x03,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00}
},
{ // 4x4 Asteroid
    {0x80,0xC0,0xE0,0xF0,0xF8,0xFC,0xFE,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0xFF,
        0xFF,0xFF,0xFF,0xFE,0xFC,0xF8,0xF0,0xE0,0xC0,0x80,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0xFF,0xFF,0xFF,0xFF,0xFF,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,
        0xFF,0xFF,0x81,0x7E,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x7E,
        0x81,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0xFF,0xFF,0xFF,0x81,0x7E,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0x7E,0x81,0xFF,0xFF,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xE7,0xDB,0xDB,0xE7,0xFF,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x01,0x03,0x07,0x0F,0x1F,0x3E,0x7E,0xFE,0xFE,0xFE,0xFE,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC3,0xBD,0xBD,
        0xBD,0xBD,0xC3,0x7F,0x3F,0x1F,0x0F,0x07,0x03,0x01,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00}
},
{ // 5x5 Asteroid
    {0xF8,0xFF,0xFF,0x00,0xFF,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0xFF,0x00,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x7F,0x7F,
        0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x00},
    {0xF9,0xFF,0xFF,0xFC,0xFB,0xFB,0xFA,0xFA,0xFA,0xFA,0xFA,
        0xFA,0xFA,0xFA,0xFA,0xFA,0xFA,0xFB,0xFB,0xFC,0xFF,0x7F,
        0x7F,0x7F,0x7F,0x7F,0x7F,0xFF,0xFF,0xFF,0x00,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF},
    {0x71,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x81,0x7E,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0x7E,0x81,0xFF,0xFF,0xFE,0xFE,
        0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xE2},
    {0x00,0xFF,0xFF,0x01,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,
        0x01,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFE,
        0xFE,0xFE,0xFE,0xFE,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x7F}
},
{ // 1/4 x 1/4 Enemy spaceship
```

```
        {0x07,0x02,0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    },
    { // 1/2 x 1/2 Enemy spaceship
        {0x0F,0x06,0x06,0x06,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    },
    { // 1x1 Enemy spaceship
        {0xFF,0x7E,0x3C,0x7F,0x2E,0x2C,0x2C,0x38,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    },
```

```
{ // 2x2 Enemy spaceship
   {0x3F,0x1E,0x0C,0xFF,0xFE,0x7C,0x78,0x60,0xE0,0xE0,0x60,
      0x60,0x40,0xC0,0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00},
   {0xFF,0x7E,0x3C,0x7F,0x3A,0x30,0x38,0x30,0x3A,0x3F,0x3E,
      0x1E,0x1E,0x1E,0x1F,0x1F,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00}
},
{ // 3x3 Enemy spaceship
   {0xFF,0x7E,0x3C,0xFC,0xFC,0xF8,0xF8,0xF0,0xF0,0xE0,0xE0,
      0xE0,0xC0,0xC0,0xC0,0x80,0x80,0x80,0x80,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00},
   {0x24,0x24,0x24,0xC3,0xFF,0xE1,0x9C,0xD8,0x80,0xD8,0x9C,
      0xE1,0xFF,0xFF,0xFF,0xFF,0xE1,0xE1,0xE1,0xE3,0xE3,0xE3,
      0xE7,0x7E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00},
   {0xFF,0x7E,0x3C,0x3F,0x3F,0x1F,0x1F,0x0F,0x0F,0x07,0x07,
      0x07,0x03,0x03,0x03,0x01,0x01,0x01,0x01,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00}
},
{ //4x4 Enemy spaceship
   {0xFC,0xFE,0xFE,0xFC,0xFE,0xFE,0xFC,0xFC,0xF8,0xF8,0xF0,
      0xF0,0xE0,0xE0,0xC0,0xC0,0x80,0x80,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00},
   {0x03,0x07,0xD7,0x23,0xE7,0x57,0xFF,0x03,0xF1,0xF0,0xE0,
      0x00,0x00,0x00,0x00,0xE0,0xF0,0xF1,0x03,0xFF,0xFE,0xFE,
      0x04,0x04,0x08,0x08,0x10,0x30,0xE0,0xC0,0x80,0x80,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00},
   {0xC0,0xE0,0xE4,0xC3,0xE6,0xE9,0xFF,0xFC,0xF8,0x81,0x81,
      0xF0,0x00,0x00,0xF0,0x81,0x81,0xF8,0xFC,0xFF,0x7F,0x7F,
      0x20,0x20,0x10,0x10,0x08,0x0C,0x07,0x03,0x01,0x01,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00},
   {0x3F,0x7F,0x7F,0x3F,0x7F,0x7F,0x3F,0x3F,0x1F,0x1F,0x0F,
      0x0F,0x07,0x07,0x03,0x03,0x01,0x01,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

```
    },
    { //5x5 Enemy spaceship
       {0xFF,0x7F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x7F,0x7F,0xFF,
          0xFF,0xFF,0xFE,0xFE,0xFE,0xFC,0xFC,0xFC,0xF8,0xF8,0xF0,
          0xF0,0xE0,0xE0,0xC0,0xC0,0x80,0x80,0x00,0x00,0x00,0x00,
          0x00,0x00,0x00,0x00,0x00,0x00,0x00},
       {0x80,0x80,0xC0,0xC0,0xE0,0xE0,0xF0,0xF0,0xFE,0x52,0x92,
          0x22,0xC4,0x09,0xF3,0xF7,0xFF,0x03,0xF1,0xF0,0xE0,0x00,
          0x00,0x00,0x00,0xE0,0xF0,0xF1,0x03,0xFF,0xFF,0x02,0x06,
          0x0C,0x18,0x30,0xE0,0x80,0x80,0x80},
       {0x01,0x01,0x03,0x03,0x07,0x07,0x0F,0x0F,0x7F,0x4A,0x49,
          0x44,0x43,0x90,0xCF,0xEF,0xFF,0xFC,0xF8,0x81,0x81,0xF0,
          0x00,0x00,0xF0,0x81,0x81,0xF8,0xFC,0xFF,0xFF,0x40,0x60,
          0x30,0x18,0x0C,0x07,0x01,0x01,0x01 },
       {0xFF,0xFE,0xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0xFE,0xFE,0xFF,
          0xFF,0xFF,0x7F,0x7F,0x7F,0x3F,0x3F,0x3F,0x1F,0x1F,0x0F,
          0x0F,0x07,0x07,0x03,0x03,0x01,0x01,0x00,0x00,0x00,0x00,
          0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    }
};
```

Listing 14: lcd$_sprites.c$

```
/*
 * LCD.c
 *
 *  Created on: 7. jun. 2023
 *      Author: georg
 */
#include "stdio.h"
#include "string.h"
#include "stm32f30x_conf.h"
#include "30010_io.h"
#include "charset.h"
#include "stopwatch.h"
#include "lcdSprites.h"
#include "LCD.h"


void initLCD()
{
   lcd_init();
}
/*
 * X = slice is [0;127] and Y = Line is [0;3]
 */
void lcd_write_string(char s[], uint8_t slice, uint8_t line,uint8_t
    *LCDbuffer_p)
{
   //Draws the string at the start position (only the part that can fit
       the rest of the line)
   for(int i = 0; i < strlen(s); i++)
   {
```

```
      for(int j = 0; j < 5; j++)
      {
         if(slice+i*5+j <= 127)
         {
            *(LCDbuffer_p+slice+i*5+j+line*128) =
                character_data[s[i]-0x20][j];
         }
      }
   }
}

/*
 * See lcd_sprites for the sprites that the type parameter pick between
 */
void lcd_draw_sprite(uint8_t * LCDbuffer,int16_t slice, int16_t line,
    uint8_t type, uint8_t mirror)
{
   for(uint8_t i = 0; i < 4; i++)
   {
      for(int16_t j = (mirror ? 39 : 0); (mirror ? j >= 0 : j < 40); j +=
          (mirror ? -1 : 1))
      {
         if(lcd_sprites[type][i][j] != 0x00 && i+line < 4 && slice+(mirror
             ? 39-j : j) < 128 && i+line >= 0 && slice+(mirror ? 39-j : j)
             >= 0)
         {
            *(LCDbuffer + slice + (line+i)*128 + (mirror ? 39-j : j)) |=
                lcd_sprites[type][i][j];
         }
      }
   }
}

void lcd_draw_crosshair(uint8_t * LCDbuffer, uint8_t slice, uint8_t line)
{
   uint8_t sprite[8] = {0x3C, 0x5A, 0x99, 0xFF, 0xFF, 0x99, 0x5A, 0x3C};

   for(int i = 0; i < 8; i++)
   {
      if(slice + i < 128)
      {
         *(LCDbuffer + slice + line*128 + i) ^= sprite[i];
      }
   }

   //free sprite? or is it automatically freed?
}


void lcd_clear_all(uint8_t * LCDbuffer,uint8_t byte)
{
   memset(LCDbuffer,byte,512);
}
```

```
void lcd_game_over(uint8_t * LCDbuffer)
{
   lcd_clear_all(LCDbuffer,0x00);
   lcd_write_string("Game Over!",40,1,LCDbuffer);
   lcd_push_buffer(LCDbuffer);
}
```

Listing 15: LCD.c

```
#include "LCD.h"
#include "LED.h"

void initLED(){
   RCC->AHBENR |= RCC_AHBPeriph_GPIOA; // Enable clock for GPIO Port A
   RCC->AHBENR |= RCC_AHBPeriph_GPIOB; // Enable clock for GPIO Port B
   RCC->AHBENR |= RCC_AHBPeriph_GPIOC; // Enable clock for GPIO Port C

   //PIN PA9 -> BLUE
   GPIOA->OSPEEDR &= ~(0x00000003 << (9 * 2)); // Clear speed register
   GPIOA->OSPEEDR |= (0x00000002 << (9 * 2)); // set speed register(0x02
      - 2 MHz)
   GPIOA->OTYPER &= ~(0x0001 << (9)); // Clear output type register
   GPIOA->OTYPER |= (0x0000 << (9)); // Set output type register(0x00 -
      Push pull)
   GPIOA->MODER &= ~(0x00000003 << (9 * 2)); // Clear mode register
   GPIOA->MODER |= (0x00000001 << (9 * 2)); // Set mode register(0x01 -
      Out)

   //PIN PB4 -> RED
   GPIOB->OSPEEDR &= ~(0x00000003 << (4 * 2)); // Clear speed register
   GPIOB->OSPEEDR |= (0x00000002 << (4 * 2)); // set speed register(0x02
      - 2 MHz)
   GPIOB->OTYPER &= ~(0x0001 << (4)); // Clear output type register
   GPIOB->OTYPER |= (0x0000 << (4)); // Set output type register(0x00 -
      Push pull)
   GPIOB->MODER &= ~(0x00000003 << (4 * 2)); // Clear mode register
   GPIOB->MODER |= (0x00000001 << (4 * 2)); // Set mode register(0x01 -
      Out)


   //PIN PC7 -> GREEN
   GPIOC->OSPEEDR &= ~(0x00000003 << (7 * 2)); // Clear speed register
   GPIOC->OSPEEDR |= (0x00000002 << (7 * 2)); // set speed register(0x02
      - 2 MHz)
   GPIOC->OTYPER &= ~(0x0001 << (7)); // Clear output type register
   GPIOC->OTYPER |= (0x0000 << (7)); // Set output type register(0x00 -
      Push pull)
   GPIOC->MODER &= ~(0x00000003 << (7 * 2)); // Clear mode register
   GPIOC->MODER |= (0x00000001 << (7 * 2)); // Set mode register(0x01 -
      Out)
}
```

```
void clearLED(){
   GPIOB->ODR &= ~(0x0001 << 4); // Clear register
   GPIOC->ODR &= ~(0x0001 << 7); // Clear register
   GPIOA->ODR &= ~(0x0001 << 9); // Clear register
}

void setLED(uint8_t x1, uint8_t x2, uint8_t x3){
   //clear registers
   clearLED();

   //set registers
   if(!x1){GPIOB->ODR |= 0x0001 << 4;} //pin PB4 (RED)
   if(!x2){GPIOC->ODR |= 0x0001 << 7;} //pin PC7 (GREEN)
   if(!x3){GPIOA->ODR |= 0x0001 << 9;} //pin PA9 (BLUE)
}

void setLEDSide(gameStruct_t * gs_p){
   static uint16_t counter = 0;

   if(!(GPIOC->ODR & (0x0001 << 7))){ //If the green LED is on
      counter++;
   }

   if (counter == 5){ //Wait for 500 ms
      counter = 0;
      if(gs_p->player.gunSide == 1){
         setLED(1, 0, 0);
      } else if(gs_p->player.gunSide == -1){
         setLED(0, 0, 1);
      }
   }
}
```

Listing 16: LED.c

```
#include <stm32f30x_conf.h> // STM32 config
#include <30010_io.h> // Input/output library for this course
#include "ansi.h"
#include "sinLut.h"
#include "vec.h"
#include "stopwatch.h"
#include "LED.h"
#include "LCD.h"
#include <stdlib.h>
#include <stdio.h>
#include "util.h"
#include "controller.h"
#include "PuttyLCDConverter.h"
#include "controllerAPI.h"
#include "entity.h"
#include "player.h"
#include "serialRead.h"
#include "entityHandler.h"
```

```
#include "bulletManager.h"
#include "enemyManager.h"
#include "gameHandler.h";

int main(void)
{
    gameStruct_t gs;

    initProgram(&gs);

    while(1)
    {
        if(game_update())
        {
            incrementCounter(&(gs.tickCounter), 1);
        }

        if(gs.tickCounter == 10)
        {
            modeSelect(&gs);
            if(gs.cooldownCounter) gs.cooldownCounter--;

            resetCounter(&(gs.tickCounter));
        }
    }
}
```

Listing 17: main.c

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include <stdio.h>
#include <stdlib.h>
#include "memory.h"

//Example code to read high score
//printf("%s: %010lu\n", readHighscoreName(0), readHighscore(0));

uint16_t readMemory(uint16_t offset){
   //Typecast to uint16_t pointer and read the value at this location
   //Add 0x0800F800 to get to the correct memory page
   return *(uint16_t *)(0x0800F800 + offset * 2);
}

void saveToMemory(uint16_t array[], uint8_t len){
   //Return if you are attempting to save more data than we have space for
   if(len > 128){ return; }

   //Unlock memory
   FLASH_Unlock();
   FLASH_ClearFlag( FLASH_FLAG_EOP | FLASH_FLAG_PGERR | FLASH_FLAG_WRPERR
      );
```

```
   //Erase page we want to write to
   FLASH_ErasePage(0x0800F800);

   //Write data to memory
   for(uint8_t i = 0; i < len; i++){
      FLASH_ProgramHalfWord(0x0800F800 + i*2, array[i]);
   }

   //Lock memory
   FLASH_Lock();
}

void clearMemory(){
   //Unlock memory
   FLASH_Unlock();
   FLASH_ClearFlag( FLASH_FLAG_EOP | FLASH_FLAG_PGERR | FLASH_FLAG_WRPERR
       );

   //Erase page
   FLASH_ErasePage(0x0800F800);

   for(uint8_t i = 0; i < 20; i++){
      FLASH_ProgramHalfWord(0x0800F800 + i*2, 0);
   }

   //Lock memory
   FLASH_Lock();
}
```

Listing 18: memory.c

```
/*
 * menus.c
 *
 *  Created on: 13. jun. 2023
 *      Author: frede
 */

#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#include "scoreCalc.h"
#include "gameHandler.h"
#include "ansi.h"
#include "highscore.h"
#include "serialRead.h"
#include "menusAPI.h"
#include "menus.h"

void initMainMenu(){
```

```
   color(15, 0);
   gotoxy(38, 5);
   printf("Cosmic Broadside Battle");

   gotoxy(10, 40);
   printf(" 'w' and 's' to navigate the menu ");
   gotoxy(10, 41);
   printf(" [SPACE] to select ");

   printScores();
}

void helpMenu(uint8_t bool){
   char *objective = "Objective";

   color(15, 0);

   window(25, 2, 75, 7, objective);
   gotoxy(30, 4);
   printf("Your goal is to survive and destroy as");
   gotoxy(30, 5);
   printf("many asteroids and enemies as possible.");


   gotoxy(10, 10);
   inverse(1);
   printf(" Player 1 controls ");
   inverse(0);

   gotoxy(10, 12);
   printf(" 'w' - Move up ");
   gotoxy(10, 13);
   printf(" 'a' - Move left ");
   gotoxy(10, 14);
   printf(" 's' - Move down ");
   gotoxy(10, 15);
   printf(" 'd' - Move right ");
   gotoxy(10, 16);
   printf(" [SPACE] - Use powerup ");

   inverse(1);
   gotoxy(10, 20);
   printf(" Player 2 controls ");
   inverse(0);

   gotoxy(10, 22);
   printf(" joystick - Move crosshair ");
   gotoxy(10, 23);
   printf(" white button - Shoot ");
   gotoxy(10, 24);
   printf(" red button - Swap gunside ");

   gotoxy(72, 10);
```

```
   inverse(1);
   printf(" Only singleplayer ");
   inverse(0);

   gotoxy(72, 12);
   printf(" , - Shoot left ");
   gotoxy(72, 13);
   printf(" . - Shoot right ");

   gotoxy(72, 20);
   inverse(1);
   printf(" Only in games ");
   inverse(0);

   gotoxy(72, 22);
   printf("'b' - Boss key");

   gotoxy(10, 40);
   printf(" 'm' to return to main menu ");

   if(bool){
      gotoxy(10, 41);
      printf(" [ESC] to return to game ");
   }

   drawMenuSprites();

   color(1, 7);
   inverse(1);
   gotoxy(55, 40);
   printf(" Write '*' ([SHIFT] + ') inside this ");
   gotoxy(55, 41);
   printf(" menu to flush the leaderboard scores ");
   inverse(0);
}

void bossScreen(){
   color(0, 4);
   clrscr();

   gotoxy(5, 5);
   color(15, 4);
   printf("A problem has been detected and Windows has been shut down to
      prevent damage to your");
   gotoxy(5, 6);
   printf("computer.");

   gotoxy(5, 11);
   printf("UNMOUNTABLE_BOOT_VOLUME");

   gotoxy(5, 16);
   printf("If this is the first time you're seeing this error screen,
      restart your computer. If this");
```

```
   gotoxy(5, 17);
   printf("screen appears again, follow these steps:");

   gotoxy(5, 22);
   printf("Check to make sure any new hardware or software is properly
       installed. If this is a");
   gotoxy(5, 23);
   printf("new installation, ask your hardware or software manufacturer
       for any Windows updates you");
   gotoxy(5, 24);
   printf("might need.");

   gotoxy(5, 29);
   printf("If problems continue, disable or remove any newly installed
       hardware or software. Disable");
   gotoxy(5, 30);
   printf("BIOS memory options such as caching or shadowing. If you need
       to use Safe Mode to remove or");
   gotoxy(5, 31);
   printf("disable components, restart your computer, press F8 to select
       Advanced Startup Options, and");
   gotoxy(5, 32);
   printf("then select Safe Mode.");

   gotoxy(5, 37);
   printf("Technical Information:");

   gotoxy(5, 42);
   printf("*** STOP: 0x5448414E (0x4B594F55, 0x464F5250, 0x4C415949,
       0x4E473A29)"); //Easter egg
}

void initGameOverScreen(gameStruct_t * gs_p){
   gotoxy(46, 5);
   printf("GAME OVER");

   gotoxy(10, 10);
   inverse(1);
   printf(" Your score: %010lu ", gs_p->score.score);
   inverse(0);

   gotoxy(10, 40);
   if(gs_p->score.score > readHighscore(4)){
      printf("Write your name using the keyboard and press [SPACE] to
          return to main menu ");
      gotoxy(16, 13);
      printf("Your name:");
   } else {
      printf("Press [SPACE] to return to main menu");
   }

   printScores();
}
```

```
void gameOverScreen(gameStruct_t * gs_p, char input){
   static char name[] = {'_', '_', '_', '_'};

   color(15, 0);
   if(gs_p->score.score > readHighscore(4)){
      gotoxy(17, 15);
      printf("%c %c %c %c", name[0], name[1], name[2], name[3]);

      for(uint8_t i = 0; i < 4; i++){ //Write your name
         if(input != 0x3F && name[i] == 0x5F){
            name[i] = input;
            break;
         }
      }

      if(input == 0x20){ //Save highscore when you press space
         saveHighscore(name, gs_p->score.score);
         for(uint8_t i = 0; i < 4; i++){ //And reset the name for next
             round
            name[i] = '_';
         }
      }
   }


}
```

Listing 19: menus.c

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include <stdio.h>
#include <stdlib.h>

#include "PuTTYSprites.h"
#include "ansi.h"
#include "highscore.h"
#include "serialRead.h"
#include "menusAPI.h"

int8_t pickMainMenuItems(char key, uint8_t activeItem){
   if(key == 'w' && activeItem > 1){
      return -1;
   } else if (key == 's' && activeItem < 3){
      return 1;
   }

   return 0;
}

void printMainMenuItems(uint8_t activeItem){
   color(15, 0);
```

```
   //MENU ITEMS
   gotoxy(10, 10);
   if(activeItem == 1){ inverse(1); } else { inverse(0); }
   printf(" Singleplayer ");

   gotoxy(10, 15);
   if(activeItem == 2){ inverse(1); } else { inverse(0); }
   printf(" Multiplayer ");

   gotoxy(10, 20);
   if(activeItem == 3){ inverse(1); } else { inverse(0); }
   printf(" Help ");

   inverse(0);
}

void printScores(){
   color(15, 0);

   //Read highscores
   char *highName0 = readHighscoreName(0);
   char *highName1 = readHighscoreName(1);
   char *highName2 = readHighscoreName(2);
   char *highName3 = readHighscoreName(3);
   char *highName4 = readHighscoreName(4);

   //Print highscores
   gotoxy(74, 10);
   inverse(1);
   printf(" HIGHSCORES ");
   inverse(0);
   gotoxy(72, 12);
   printf("%s: %010lu", highName0, readHighscore(0));
   gotoxy(72, 14);
   printf("%s: %010lu", highName1, readHighscore(1));
   gotoxy(72, 16);
   printf("%s: %010lu", highName2, readHighscore(2));
   gotoxy(72, 18);
   printf("%s: %010lu", highName3, readHighscore(3));
   gotoxy(72, 20);
   printf("%s: %010lu", highName4, readHighscore(4));

   //Free memory
   free(highName0);
   free(highName1);
   free(highName2);
   free(highName3);
   free(highName4);
}

//for the help screen
void drawMenuSprites(){
```

```
    ui_draw_sprite(0, 15, 0, 12, 30);
    ui_draw_sprite(1, 15, 0, 20, 30);
    ui_draw_sprite(2, 15, 0, 28, 31);
    ui_draw_sprite(3, 15, 0, 36, 32);
    ui_draw_sprite(4, 15, 0, 43, 31);
    ui_draw_sprite(5, 15, 0, 50, 30);
    ui_draw_sprite(6, 15, 0, 61, 32);
    ui_draw_sprite(7, 15, 0, 68, 32);

    color(15, 0);
    gotoxy(74, 32);
    inverse(1);
    printf(" Entities ");
    inverse(0);

    gotoxy(10, 28);
    printf("%c%c%c%c%cPlayer%c%c%c%c%c%c", 0xDA, 0xC4, 0xC4, 0xC4, 0xC4,
        0xC4, 0xC4, 0xC4, 0xC4, 0xC4, 0xBF);

    gotoxy(10, 34);
    printf("Gun left");

    gotoxy(18, 35);
    printf("Gun right");

    gotoxy(26, 34);
    printf(" Enemy");

    gotoxy(34, 35);
    printf("S asteroid");

    gotoxy(42, 34);
    printf("M asteroid");

    gotoxy(50, 35);
    printf("L asteroid");

    gotoxy(58, 34);
    printf("Bullet");

    gotoxy(66, 35);
    printf("Mega bullet");
}
```

Listing 20: menusAPI.c

```
#include <stdint.h>
#include "ansi.h"
#include "PuTTYSprites.h"
#include "vec.h"
#include "entity.h"
#include "PuttyLCDConverter.h"
#include "util.h"
```

```
#include "player.h"

void initPlayer(entity_t *entity, player_t *player, uint8_t num){
   player->entity = entity;
   player->playerNum = num;
   player->gunSide = 1; //1 = LEFT | -1 = RIGHT
   player->powerUp = 0;
   player->crosshairX = 63; //middle of LCD
   player->crosshairY = 1;
   player->entity->isActive = 1;
   player->HP = 3;
}

void updateCrosshair(player_t *ptr,uint8_t joystickVal)
{
   switch(joystickVal)
   {
      case 0x01:
         ptr->crosshairY = capInterval(ptr->crosshairY-1,0,3);
         break;
      case 0x02:
         ptr->crosshairY = capInterval(ptr->crosshairY-1,0,3);
         ptr->crosshairX = capInterval(ptr->crosshairX+8,0,127);
         break;
      case 0x04:
         ptr->crosshairX = capInterval(ptr->crosshairX+8,0,127);
         break;
      case 0x08:
         ptr->crosshairY = capInterval(ptr->crosshairY+1,0,3);
         ptr->crosshairX = capInterval(ptr->crosshairX+8,0,127);
         break;
      case 0x10:
         ptr->crosshairY = capInterval(ptr->crosshairY+1,0,3);
         break;
      case 0x20:
         ptr->crosshairY = capInterval(ptr->crosshairY+1,0,3);
         ptr->crosshairX = capInterval(ptr->crosshairX-8,0,127);
         break;
      case 0x40:
         ptr->crosshairX = capInterval(ptr->crosshairX-8,0,127);
         break;
      case 0x80:
         ptr->crosshairY = capInterval(ptr->crosshairY-1,0,3);
         ptr->crosshairX = capInterval(ptr->crosshairX-8,0,127);
         break;
      default:
         break;
   }
}

void changeGunside(player_t *player){
   if(player->gunSide == 1){
      player->gunSide = -1;
```

```
   } else {
      player->gunSide = 1;
   }
}

void drawPlayer(player_t *player)
{
   if(player->gunSide == 1)
   {
      ui_draw_sprite(0, 15, 0, getXint(&(player->entity->pos)),
         getYint(&(player->entity->pos)));
   } else {
      ui_draw_sprite(1, 15, 0, getXint(&(player->entity->pos)),
         getYint(&(player->entity->pos)));
   }
}

void clearPlayer(player_t *player)
{
   if(player->gunSide == 1)
   {
      ui_clear_sprite(0, 15, 0, getXint(&(player->entity->pos)),
         getYint(&(player->entity->pos)));
   } else {
      ui_clear_sprite(1, 15, 0, getXint(&(player->entity->pos)),
         getYint(&(player->entity->pos)));
   }
}

void isAlive(player_t * player){
   if(player->HP < 1) player->entity->isActive = 0;
}

void damagePlayer(player_t *ptr, int8_t x){
   ptr->HP -= x;
   ptr->HP = ptr->HP > 0 ? ptr->HP : 0;
   ptr->HP = ptr->HP > 3 ? 3 : ptr->HP;
}

void updatePlayerVel(player_t *player, char input){
   switch(input){
      case('w'):
         if(player->entity->vel.y > -1){
            updateVel(player->entity, 0, -1);
         }
         break;
      case('a'):
         if(player->entity->vel.x > -1){
            updateVel(player->entity, -1, 0);
         }
         break;
      case('s'):
         if(player->entity->vel.y < 1){
```

```
            updateVel(player->entity, 0, 1);
         }
         break;
      case('d'):
         if(player->entity->vel.x < 1){
            updateVel(player->entity, 1, 0);
         }
         break;
   }
}

/*
 * uses PuttyLCDConverter functions to get the direction of the crosshair
     on the LCD converted to putty then spawns bullet
 */
void playerShoot(player_t *ptr, bulletManager_t
    *bulletManager,entityHandler_t *entHan, uint8_t bulletType, uint8_t
    height)
{
   spawnBullet(bulletManager, entHan,offsetBulletCoordX(ptr),
       offsetBulletCoordY(ptr), con_getVecX(ptr->gunSide),
       con_getVecY(ptr->crosshairX,ptr->gunSide),1,0, ptr->crosshairY, 1);
}

//check if player collides with an asteroid, enemy ship or bullet.
//damage player and set the other entity inactive if collision.
void checkPlayerCollision(player_t * ptr, entityHandler_t * array)
{
   uint8_t v;
   for(v = 1; v < ENTITY_ARR_LEN; v++) //avoids checking player itself
   {
      if(array->entityArray[v]->isActive &&
          array->entityArray[v]->spriteIndex != 7) //cant collide with
          megabullet
      {
         if(detectEntityCollision(ptr->entity, array->entityArray[v]))
         {
            damagePlayer(ptr,1);
            destroyEntity(array->entityArray[v]);
         }
      }
   }
}

void usePowerUp(player_t * ptr, bulletManager_t *
    bulletManager,entityHandler_t * entHan){
   switch(ptr->powerUp){
      case(1): //restock HP
         ptr->HP = ptr->HP < 3 ? 3 : ptr->HP;
         ptr->powerUp = 0;
         break;
      case(2): //add shield
         ptr->HP += (ptr->HP < 4 ? 1 : 0);
```

```
        ptr->powerUp = 0;
        break;
     case(3): //spawn megabullet in front of player with velocity -2.
        spawnBullet(bulletManager,entHan,getXint(&(ptr->entity->pos))+1,
           getYint(&(ptr->entity->pos))-1, 0, -2, 0, 1, 0, 1);
        ptr->powerUp = 0;
        break;
     default:
        return;
  }
}
```

Listing 21: player.c

```
#include "stdio.h"
#include "stm32f30x_conf.h"
#include "30010_io.h"
#include "util.h"
#include "LCD.h"
#include "vec.h"
#include "PuTTYSprites.h"
#include "entity.h"
#include "PuttyLCDConverter.h"

#define CONE_VIEW_WIDTH 32
#define CONE_VIEW_LENGTH 40

/*
 * Returns the fixed point 18.14 y coordinate for the unitvector in the
    direction given by the LCD slice
 *
 * for left gunside = 1;
 * for right gunside = -1;
 */
int32_t con_getVecY(uint8_t slice, int8_t gunside)
{
  return (-gunside)*(((((slice+1) >> 2) - 16) << 14) /
      CONE_VIEW_LENGTH); //rightshift by 2 to divide by 4
}

/*
 * Returns the fixed point 18.14 x coordinate for the unitvector in the
    given gunner direction
 *
 * for left gunside = 1;
 * for right gunside = -1;
 */
int32_t con_getVecX(int8_t gunside)
{
  return (-gunside + 0x80000000) << 14; //plus with 0x80000000 to
      convert to a 32bit int for fixed point. Probably not needed
}
```

```
/*
 * Returns an int in the interval [-3;2] based on distanceX from ship:
 * dist = ]0;8] returns 2
 * dist = ]8;16] returns 1
 * dist = ]16;24] returns 0
 * dist = ]24;32] returns -1
 * dist = ]32;40] returns -2
 * dist > 40 returns -3
 *
 * Add this with entity size to determine sprite-size to be drawn on the
    LCD
 */
int8_t con_getDistanceX(uint8_t playerX, uint8_t entX)
{
   switch((absolute(playerX - entX)-1) >> 3) //divide by 8
   {
      case 0:
         return 2;
      case 1:
         return 1;
      case 2:
         return 0;
      case 3:
         return -1;
      case 4:
         return -2;
      default:
         return -3;
   }
}

/*
 * Returns 1 if the entity is viewable inside the coneview
 * (also returns 1 if the entities position is outside the cone but part
    of its sprite could be inside)
 *
 * Returns 0 if the entity is NOT viewable inside the coneview
 *
 * Parameters:
 *       gunside = 1 for left and = -1 for right
 */
uint8_t con_inCone(uint8_t playerX, uint8_t playerY, uint8_t entX,
   uint8_t entY, int8_t gunside)
{
   if((gunside > 0 ? playerX > entX : entX > playerX) && absolute(playerX
      - entX) <= CONE_VIEW_LENGTH && absolute(playerY - entY) <=
      ((CONE_VIEW_WIDTH * absolute(playerX - entX)) /
      (2*CONE_VIEW_LENGTH)) + 2) //+2 to allow entities just outside to
      be drawn
   {
      return 1;
   }
   else{return 0;}
```

```
}


/*
 * returns the slice for the entity to be drawn on
 *
 * can return slices outside of [0:128] so that entities on their way
    into the cone view can be drawn smoothly moving in
 *
 * Parameters:
 *      gunside = 1 for left and = -1 for right
 */
int16_t con_posToSlice(uint8_t playerX, uint8_t playerY, uint8_t entX,
   uint8_t entY, int8_t gunside)
{
   int16_t temp = ((CONE_VIEW_WIDTH * (absolute(playerX - entX))) /
      (2*CONE_VIEW_LENGTH));
   int16_t coneY = (temp == 0 ? 1 : temp); //to avoid div by zero

   //padding on the coneY value allows enemies to be drawn on the edge of
      the LCD
   return mapInterval(-coneY-2,coneY+2,(gunside < 0 ? -(256/coneY) :
      (128)+(256/coneY)),(gunside < 0 ? (128)+(256/coneY) :
      -(256/coneY)),(entY-playerY));
   /*-2 on minOld to allow ships slightly outside to be drawn smoothly in
    * same reason for -256/coneY and the same for the max
   */
}


/*
 * Translates an enemy from putty space to LCD space. The enemy is drawn
    on the LCD if its inside the defined cone
 */
void con_draw_putty_to_lcd(enemyManager_t *enemMan, player_t
   *player,uint8_t * LCDbuffer)
{
   for(int i = 0; i < ENEMY_ARR_LENGTH; i++)
   {
      if(enemMan->enemyArray[i]->entity->isActive)
      {
         if(con_inCone(offsetBulletCoordXCone(player)
             ,offsetBulletCoordY(player),
             centeredXPOS(enemMan->enemyArray[i]->entity) >> 14,
             getYint(&(enemMan->enemyArray[i]->entity->pos))+2,
             player->gunSide))
         {
            //Perhaps this should be split into more lines
            lcd_draw_sprite(LCDbuffer,
                con_posToSlice(offsetBulletCoordXCone(player),
                offsetBulletCoordY(player),
                centeredXPOS(enemMan->enemyArray[i]->entity) >>
                14,getYint(&(enemMan->enemyArray[i]->entity->pos))+2
```

```
                ,player->gunSide),enemMan->enemyArray[i]->entity->height,
                (enemMan->enemyArray[i]->type ?
                enemMan->enemyArray[i]->type :
                10)+1+con_getDistanceX(offsetBulletCoordXCone(player),
                centeredXPOS(enemMan->enemyArray[i]->entity) >> 14),
                player->gunSide == 1 ? 1 : 0);
        }
      }
   }
}
```

Listing 22: PuttyLCDConverter.c

```
#include "ansi.h"
#include <stdint.h>
#include <stdio.h>
#include <PuTTYSprites.h>

/*  sprites:
 *    spriteArray[0] = player (GUN LEFT)
 *    spriteArray[1] = player (GUN RIGHT)
 *  spriteArray[2] = enemy spaceship
 *  spriteArray[3] = small asteroid
 *  spriteArray[4] = medium asteroid
 *  spriteArray[5] = big asteroid
 *  spriteArray[6] = bullet
 * spriteArray[7] = mega Bullet
 *
*/

//we use 0x3F as 'blank'
const char spriteArray[8][6][7] = {

      //0 PLAYER WITH LEFT GUN (4x4)
      {{0x3F,0x2F,0xC1,0x5C,0x3F,0x3F,0x3F},
      {0xAE,0xB9,0xDB,0xBA,0x3F,0x3F,0x3F},
      {0x3F,0xBA,0xDB,0xBA,0x3F,0x3F,0x3F},
      {0x3F,0xBE,0x3F,0xBE,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F}},
      //1 PLAYER WITH RIGHT GUN (4x4)
      {{0x3F,0x2F,0xC1,0x5C,0x3F,0x3F,0x3F},
      {0x3F,0xBA,0xDB,0xCC,0xAF,0x3F,0x3F},
      {0x3F,0xBA,0xDB,0xBA,0x3F,0x3F,0x3F},
      {0x3F,0xBE,0x3F,0xBE,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F}},
      //2 Enemy spaceShip (4x3)
      {{0xCF,0x3F,0x3F,0xCF,0x3F,0x3F,0x3F},
      {0xC8,0xDB,0xDB,0xBC,0x3F,0x3F,0x3F},
      {0x3F,0x5C,0x2F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
```

```
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F}},
      //3 small Asteroid (3x2)
      {{0xDC,0xDC,0xDC,0x3F,0x3F,0x3F,0x3F},
      {0xDF,0xDF,0xDF,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F}},
      //4 medium Asteroid (5x3)
      {{0xDC,0xDC,0xDC,0xDC,0xDC,0x3F,0x3F},
      {0xDB,0xDB,0xDB,0xDB,0xDB,0x3F,0x3F},
      {0xDF,0xDF,0xDF,0xDF,0xDF,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F}},
      //5 big Asteroid (7x4)
      {{0xDC,0xDC,0xDC,0xDC,0xDC,0xDC,0xDC},
      {0xDB,0xDB,0xDB,0xDB,0xDB,0xDB,0xDB},
      {0xDB,0xDB,0xDB,0xDB,0xDB,0xDB,0xDB},
      {0xDF,0xDF,0xDF,0xDF,0xDF,0xDF,0xDF},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F}},
      //6 bullet - 1x1
      {{0x6F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F}},
      //7 mega bullet - 3x1
      {{0x5B,0xDB,0x5D,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},
      {0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F}}
};


void ui_draw_sprite(uint8_t index, uint8_t FGC, uint8_t BGC, uint8_t x,
    uint8_t y){
   uint8_t i,j,subX,subY;

   //optimizes by ignoring the blank space of smaller sprites
   switch(index){
   case(0):
      subX = 3;
      subY = 2;
      break;
   case(1):
      subX = 2;
      subY = 2;
      break;
```

```
   case(2):
      subX = 3;
      subY = 3;
      break;
   case(3):
      subX = 4;
      subY = 4;
      break;
   case(4):
      subX = 2;
      subY = 3;
      break;
   case(5):
      subX = 0;
      subY = 2;
      break;
   case(6):
      subX = 6;
      subY = 5;
      break;
   case(7):
      subX = 4;
      subY = 5;
   }

   gotoxy(x, y);

   for(i = 0; i <= 5-subY; i++){
      for(j = 0; j <= 6-subX; j++){
         color(FGC, BGC);
         if(spriteArray[index][i][j] == 0xBE){color(11,0);} //thruster
             colors
         if(spriteArray[index][i][j] == 0xCF){color(1,0);}
         if(x+j > 0 && x+j < 81 && y+i > 0 && y+i < 47)
         {
            //doesn't draw blank spots
            spriteArray[index][i][j] == 0x3F ? moveCursorX(1,1) :
                printf("%c", spriteArray[index][i][j]);
         } else {
            moveCursorX(1,1);
         }
      }
      moveCursorY(1,0);
      moveCursorX(7-subX,0);
   }
}

//almost same as draw sprite
void ui_clear_sprite(uint8_t index, uint8_t FGC, uint8_t BGC, uint8_t x,
    uint8_t y){
   uint8_t i, j, subX, subY;
   switch(index){
      case(0):
```

```
        subX = 3;
        subY = 2;
        break;
     case(1):
        subX = 2;
        subY = 2;
        break;
     case(2):
        subX = 3;
        subY = 3;
        break;
     case(3):
        subX = 4;
        subY = 4;
        break;
     case(4):
        subX = 2;
        subY = 3;
        break;
     case(5):
        subX = 0;
        subY = 2;
        break;
     case(6):
        subX = 6;
        subY = 5;
        break;
     case(7):
        subX = 4;
        subY = 5;
     }

   gotoxy(x, y);
   color(FGC, BGC);

   for(i = 0; i <= 5-subY; i++){
      for(j = 0; j <= 6-subX; j++){
         if(x+j > 0 && x+j < 81 && y+i > 0 && y+i < 47){
            spriteArray[index][i][j] == 0x3F ? moveCursorX(1,1) : printf("
                ");
         } else {
            moveCursorX(1,1);
         }
      }
      moveCursorY(1,0);
      moveCursorX(7-subX,0);
   }
}

//x coordinate for the char infront of the player gun
int16_t offsetBulletCoordX(player_t *player)
{
   if(player->gunSide == 1){
```

```
      return getXint(&(player->entity->pos)) - 3;
   } else {
      return getXint(&(player->entity->pos)) + 7;
   }
}

//x coordinate to start the cone
int16_t offsetBulletCoordXCone(player_t *player)
{
   if(player->gunSide == 1){
      return getXint(&(player->entity->pos));
   } else {
      return getXint(&(player->entity->pos)) + 4;
   }
}

//y coordinate for the char infront of the player gun
int16_t offsetBulletCoordY(player_t *player)
{
   return getYint(&(player->entity->pos)) + 1;
}
```

Listing 23: PuTTYSprites.c

```
#include "scoreCalc.h"

void initScore(gamescore_t * Score){
   Score->score = 0;
}

void incrementScore(gamescore_t * Score, uint32_t points){
   if(Score->score >= 0xFAA2B57F){ //Make sure the score cannot overflow
       and hit 0 (optimistic to hit this score)
      Score->score = 0xFAA2B57F;
   } else {
      Score->score += points;
   }
}
```

Listing 24: scoreCalc.c

```
#include "serialRead.h"

uint8_t get_key_pressed(){

   uint8_t temp = uart_get_char();
   if(uart_get_count() >= 3) uart_clear();

   switch(temp){
   case(0x1B): //esc
   case(0x2A): //'*'
   case(0x20): //Space
   case(0x2E): //Full Stop
   case(0x2c): //COMMA
```

```
      return(temp);
   default:
      return(temp >= 0x41 && temp <= 0x7A ? temp : 0x3F);
   }
}
```

Listing 25: serialRead.c

```
// ====================================================================
//   Look-Up Tables
//    SIN: sin(x*pi/256)
//
// Exported by Cearn's excellut v1.0
// (comments, kudos, flames to daytshen@hotmail.com)
//
// ====================================================================

#include "stdio.h"
#include "sinLut.h"
// --------------------------------------------------------------------
// SIN: a 512 long LUT of 16bit values in 2.14 format
// sin(x*pi/256)
const signed short SIN[512]=
{
   0x0000,0x00C9,0x0192,0x025B,0x0324,0x03ED,0x04B5,0x057E,
   0x0646,0x070E,0x07D6,0x089D,0x0964,0x0A2B,0x0AF1,0x0BB7,
   0x0C7C,0x0D41,0x0E06,0x0ECA,0x0F8D,0x1050,0x1112,0x11D3,
   0x1294,0x1354,0x1413,0x14D2,0x1590,0x164C,0x1709,0x17C4,
   0x187E,0x1937,0x19EF,0x1AA7,0x1B5D,0x1C12,0x1CC6,0x1D79,
   0x1E2B,0x1EDC,0x1F8C,0x203A,0x20E7,0x2193,0x223D,0x22E7,
   0x238E,0x2435,0x24DA,0x257E,0x2620,0x26C1,0x2760,0x27FE,
   0x289A,0x2935,0x29CE,0x2A65,0x2AFB,0x2B8F,0x2C21,0x2CB2,

   0x2D41,0x2DCF,0x2E5A,0x2EE4,0x2F6C,0x2FF2,0x3076,0x30F9,
   0x3179,0x31F8,0x3274,0x32EF,0x3368,0x33DF,0x3453,0x34C6,
   0x3537,0x35A5,0x3612,0x367D,0x36E5,0x374B,0x37B0,0x3812,
   0x3871,0x38CF,0x392B,0x3984,0x39DB,0x3A30,0x3A82,0x3AD3,
   0x3B21,0x3B6D,0x3BB6,0x3BFD,0x3C42,0x3C85,0x3CC5,0x3D03,
   0x3D3F,0x3D78,0x3DAF,0x3DE3,0x3E15,0x3E45,0x3E72,0x3E9D,
   0x3EC5,0x3EEB,0x3F0F,0x3F30,0x3F4F,0x3F6B,0x3F85,0x3F9C,
   0x3FB1,0x3FC4,0x3FD4,0x3FE1,0x3FEC,0x3FF5,0x3FFB,0x3FFF,

   0x4000,0x3FFF,0x3FFB,0x3FF5,0x3FEC,0x3FE1,0x3FD4,0x3FC4,
   0x3FB1,0x3F9C,0x3F85,0x3F6B,0x3F4F,0x3F30,0x3F0F,0x3EEB,
   0x3EC5,0x3E9D,0x3E72,0x3E45,0x3E15,0x3DE3,0x3DAF,0x3D78,
   0x3D3F,0x3D03,0x3CC5,0x3C85,0x3C42,0x3BFD,0x3BB6,0x3B6D,
   0x3B21,0x3AD3,0x3A82,0x3A30,0x39DB,0x3984,0x392B,0x38CF,
   0x3871,0x3812,0x37B0,0x374B,0x36E5,0x367D,0x3612,0x35A5,
   0x3537,0x34C6,0x3453,0x33DF,0x3368,0x32EF,0x3274,0x31F8,
   0x3179,0x30F9,0x3076,0x2FF2,0x2F6C,0x2EE4,0x2E5A,0x2DCF,

   0x2D41,0x2CB2,0x2C21,0x2B8F,0x2AFB,0x2A65,0x29CE,0x2935,
   0x289A,0x27FE,0x2760,0x26C1,0x2620,0x257E,0x24DA,0x2435,
```

```
   0x238E,0x22E7,0x223D,0x2193,0x20E7,0x203A,0x1F8C,0x1EDC,
   0x1E2B,0x1D79,0x1CC6,0x1C12,0x1B5D,0x1AA7,0x19EF,0x1937,
   0x187E,0x17C4,0x1709,0x164C,0x1590,0x14D2,0x1413,0x1354,
   0x1294,0x11D3,0x1112,0x1050,0x0F8D,0x0ECA,0x0E06,0x0D41,
   0x0C7C,0x0BB7,0x0AF1,0x0A2B,0x0964,0x089D,0x07D6,0x070E,
   0x0646,0x057E,0x04B5,0x03ED,0x0324,0x025B,0x0192,0x00C9,

   0x0000,0xFF37,0xFE6E,0xFDA5,0xFCDC,0xFC13,0xFB4B,0xFA82,
   0xF9BA,0xF8F2,0xF82A,0xF763,0xF69C,0xF5D5,0xF50F,0xF449,
   0xF384,0xF2BF,0xF1FA,0xF136,0xF073,0xEFB0,0xEEEE,0xEE2D,
   0xED6C,0xECAC,0xEBED,0xEB2E,0xEA70,0xE9B4,0xE8F7,0xE83C,
   0xE782,0xE6C9,0xE611,0xE559,0xE4A3,0xE3EE,0xE33A,0xE287,
   0xE1D5,0xE124,0xE074,0xDFC6,0xDF19,0xDE6D,0xDDC3,0xDD19,
   0xDC72,0xDBCB,0xDB26,0xDA82,0xD9E0,0xD93F,0xD8A0,0xD802,
   0xD766,0xD6CB,0xD632,0xD59B,0xD505,0xD471,0xD3DF,0xD34E,

   0xD2BF,0xD231,0xD1A6,0xD11C,0xD094,0xD00E,0xCF8A,0xCF07,
   0xCE87,0xCE08,0xCD8C,0xCD11,0xCC98,0xCC21,0xCBAD,0xCB3A,
   0xCAC9,0xCA5B,0xC9EE,0xC983,0xC91B,0xC8B5,0xC850,0xC7EE,
   0xC78F,0xC731,0xC6D5,0xC67C,0xC625,0xC5D0,0xC57E,0xC52D,
   0xC4DF,0xC493,0xC44A,0xC403,0xC3BE,0xC37B,0xC33B,0xC2FD,
   0xC2C1,0xC288,0xC251,0xC21D,0xC1EB,0xC1BB,0xC18E,0xC163,
   0xC13B,0xC115,0xC0F1,0xC0D0,0xC0B1,0xC095,0xC07B,0xC064,
   0xC04F,0xC03C,0xC02C,0xC01F,0xC014,0xC00B,0xC005,0xC001,

   0xC000,0xC001,0xC005,0xC00B,0xC014,0xC01F,0xC02C,0xC03C,
   0xC04F,0xC064,0xC07B,0xC095,0xC0B1,0xC0D0,0xC0F1,0xC115,
   0xC13B,0xC163,0xC18E,0xC1BB,0xC1EB,0xC21D,0xC251,0xC288,
   0xC2C1,0xC2FD,0xC33B,0xC37B,0xC3BE,0xC403,0xC44A,0xC493,
   0xC4DF,0xC52D,0xC57E,0xC5D0,0xC625,0xC67C,0xC6D5,0xC731,
   0xC78F,0xC7EE,0xC850,0xC8B5,0xC91B,0xC983,0xC9EE,0xCA5B,
   0xCAC9,0xCB3A,0xCBAD,0xCC21,0xCC98,0xCD11,0xCD8C,0xCE08,
   0xCE87,0xCF07,0xCF8A,0xD00E,0xD094,0xD11C,0xD1A6,0xD231,

   0xD2BF,0xD34E,0xD3DF,0xD471,0xD505,0xD59B,0xD632,0xD6CB,
   0xD766,0xD802,0xD8A0,0xD93F,0xD9E0,0xDA82,0xDB26,0xDBCB,
   0xDC72,0xDD19,0xDDC3,0xDE6D,0xDF19,0xDFC6,0xE074,0xE124,
   0xE1D5,0xE287,0xE33A,0xE3EE,0xE4A3,0xE559,0xE611,0xE6C9,
   0xE782,0xE83C,0xE8F7,0xE9B4,0xEA70,0xEB2E,0xEBED,0xECAC,
   0xED6C,0xEE2D,0xEEEE,0xEFB0,0xF073,0xF136,0xF1FA,0xF2BF,
   0xF384,0xF449,0xF50F,0xF5D5,0xF69C,0xF763,0xF82A,0xF8F2,
   0xF9BA,0xFA82,0xFB4B,0xFC13,0xFCDC,0xFDA5,0xFE6E,0xFF37,
};

int32_t expand(int32_t i)
{
   // Converts an 18.14 fixed point number to 16.16
   return i << 2;
}

void printFix(int32_t i)
{
   // Prints a signed 16.16 fixed point number
```

```
   if ((i & 0x80000000) != 0) // Handle negative numbers
   {
      printf("-");
      i = ~i + 1;
   }
   printf("%ld.%04ld", i >> 16, 10000 * (uint32_t)(i & 0xFFFF) >> 16);
   // Print a maximum of 4 decimal digits to avoid overflow
}

/*
 * degrees not in 360 but in 512
 * 45 real degrees = 45/360*512 = 64 for this method
 */
int16_t lutSin(int16_t degrees)
{
   return SIN[degrees & 0x1FF];
}

int16_t lutCos(int16_t degrees)
{
   return lutSin(degrees+128);
}
```

Listing 26: sinLut.c

```
/*
 * stopwatch.c
 *
 *  Created on: 6. jun. 2023
 *      Author: georg
 */
#include "stdio.h"
#include "stm32f30x_conf.h"
#include "30010_io.h"
#include "stopwatch.h"

volatile uint8_t timer_Flag;

void initTimer()
{
   RCC->APB2ENR |= RCC_APB2Periph_TIM15; // Enable clock line to timer 15;
   TIM15->CR1 = 0x0000; // Configure timer 15 and disable counter
   TIM15->ARR = 624; // Set reload value
   TIM15->PSC = 1023; // Set prescale value
   //assuming a systemclock of 64mhz this reload value and prescale will
       result in a 1/100 seconds upcounting-mode timeout period
   TIM15->CR1 = 0x0001; // Configure timer 15 and enable counter

   //Interupt part:
   TIM15->DIER |= 0x0001; // Enable timer 15 interrupts
   NVIC_SetPriority(TIM1_BRK_TIM15_IRQn, 1); // Set interrupt priority
   NVIC_EnableIRQ(TIM1_BRK_TIM15_IRQn); // Enable interrupt
}
```

```
void TIM1_BRK_TIM15_IRQHandler(void)
{
   timer_Flag = 1; //global flag

   TIM15->SR &= ~(0x0001); //Clear the interupt bit
}
```

Listing 27: stopwatch.c

```
/*
 * util.c
 *
 *  Created on: 14. jun. 2023
 *      Author: georg
 */

#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include "stdlib.h"
#include "stdio.h"
#include "entity.h"
#include "util.h"

/*
 * Remember to initialize the PRNG with srand(int) before using this
 *
 * Returns a number from min to max. min is inclusive. max is exclusive.
 */
int32_t getRandomInterval(int32_t min, int32_t max)
{
   return mapInterval(0,RAND_MAX >> 16,min,max,rand() >> 16);
}


int32_t absolute(int32_t x)
{
   return x >= 0 ? x : -x;
}

/*
 * returns the manhattan distance of the integers (unreliable distance)
 */
int32_t getManDistance(int32_t x1, int32_t y1, int32_t x2, int32_t y2)
{
   return abs(x1-x2) + abs(y1-y2);
}

int32_t norm(int32_t x)
{
   return (x > 0 ? 1 : -1);
}
```

```
/*
 * maxOld cannot be equal to minOld !!!!!!!!!!
 */
int32_t mapInterval(int32_t minOld, int32_t maxOld,int32_t minNew,
    int32_t maxNew,int32_t value)
{
   return (value-minOld) * (maxNew-minNew) / (maxOld-minOld) + minNew;
}

int32_t capInterval(int32_t value, int32_t min, int32_t max)
{
   return value > max ? max : (value < min ? min : value);
}

void incrementCounter(uint8_t * ptr, uint8_t count){
   (*ptr) += count;
}

void resetCounter(uint8_t * ptr){
   *ptr = 0;
}
```

Listing 28: util.c

```
#include "stdio.h"
#include "sinLut.h"
#include "vec.h"

/*
 * Vector is given in fixed point 18.14 number representation
 */
void rotateVector(vector_t *v, int degrees)
{
   int32_t temp = v->x;
   v->x = ((temp*lutCos(degrees)) >> 14) - ((v->y*lutSin(degrees)) >> 14);
   v->y = ((temp*lutSin(degrees)) >> 14) + ((v->y*lutCos(degrees)) >> 14);
}

/*
 * Takes x and y coordinates as integers an initializes the vectors values
 * as the fixed point 18.14 representation of those ints
 */
void setVectorInt(vector_t *v, int32_t x, int32_t y)
{
   //Testing point values:
   v->x = x << 14;
   v->y = y << 14;
}

/*
 * Returns the integer part of the fixed point x coordinate
 */
int16_t getXint(vector_t *v)
```

```
{
   return v->x >> 14;
}

/*
 * Returns the integer part of the fixed point y coordinate
 */
int16_t getYint(vector_t *v)
{
   return v->y >> 14;
}

void updateVectorInt(vector_t *v, int32_t x, int32_t y)
{
   v->x += (x << 14);
   v->y += (y << 14);
}
```

Listing 29: vec.c

Header files

```
#ifndef ANSI_H_
#define ANSI_H_

#include "stdint.h"

void printTime(uint8_t hours, uint8_t minutes, uint8_t seconds, uint8_t
    hundredsthsOfSeconds,uint8_t x, uint8_t y, char title[]);
void fgcolor(uint8_t foreground);
void bgcolor(uint8_t background);
void color(uint8_t foreground, uint8_t background);
void resetbgcolor();
void clrscr();
void clreol();
void gotoxy(uint8_t x, uint8_t y);
void underline(uint8_t on);
void blink(uint8_t on);
void inverse(uint8_t on);
void window(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, char title[]);
void moveCursorX(uint8_t X, uint8_t UP);
void moveCursorY(uint8_t Y, uint8_t UP);
void invisibleCursor();

#endif /* ANSI_H_ */
```

Listing 30: ansi.h

```
#ifndef BULLET_H_
#define BULLET_H_

#include <stdint.h>
```

```
#include "entity.h"

typedef struct {
   entity_t *entity;
   uint8_t type;
   uint8_t friendly;
} bullet_t;

void initBullet(bullet_t *bullet, entity_t *entity, uint8_t type, uint8_t
    friendly);

#endif /* BULLET_H_ */
```

Listing 31: bullet.h

```
#ifndef BULLETMANAGER_H_
#define BULLETMANAGER_H_

#define BULLET_ARR_LENGTH 16

#include "entityHandler.h"
#include "entity.h"
#include "bullet.h"
#include "scoreCalc.h"
#include <stdint.h>

typedef struct{
   bullet_t * bulletArray[BULLET_ARR_LENGTH];
} bulletManager_t;

void spawnBullet(bulletManager_t *bulletManager,entityHandler_t
    *entHan,int16_t xPos, int16_t yPos,int32_t xVel,int32_t yVel,uint8_t
    fixedVel,uint8_t bulletType, uint8_t height, uint8_t friendly);
void initBulletManager(bulletManager_t *bulletManager, bullet_t *bulArr);
void checkBulletCollision(bulletManager_t *bulletManager, entityHandler_t
    *entityHandler, gamescore_t *score,uint8_t *playerPowerUp);

#endif /* BULLETMANAGER_H_ */
```

Listing 32: bulletManage.h

```
#ifndef BUZZ_H_
#define BUZZ_H_

#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include "stdint.h"

void initBuzz();
void setFreq(uint16_t freq);
void turnOffBuzz();

#endif /* BUZZ_H_ */
```

Listing 33: buzz.h

```
#ifndef CONTROLLER_H_
#define CONTROLLER_H_

#include <stdint.h>

void initController();
uint16_t readPot1();
uint16_t readPot2();
void potsToString(char array[]);
uint8_t readButtons();


#endif /* CONTROLLER_H_ */
```

Listing 34: controller.h

```
#ifndef CONTROLLERAPI_H_
#define CONTROLLERAPI_H_

#include <stdint.h>

uint8_t readJoystick();
uint8_t readButton1();
uint8_t readButton2();

#endif /* CONTROLLERAPI_H_ */
```

Listing 35: controllerAPI.h

```
#ifndef ENEMY_H_
#define ENEMY_H_

#include <stdint.h>
#include "bulletManager.h"
#include "vec.h"

typedef struct {
   entity_t *entity;
   uint8_t type;
} enemy_t;

void initEnemy(entity_t * entity, enemy_t * enemy, uint8_t type);
void enemyShoot(bulletManager_t *bulletManager,entityHandler_t
    *entHan,enemy_t * enemy,int32_t bulletSpeed);

#endif /* ENEMY_H_ */
```

Listing 36: enemy.h

```
#ifndef ENEMYMANAGER_H_
```

```
#define ENEMYMANAGER_H_

#define ENEMY_ARR_LENGTH 15

#include "entity.h"
#include "enemy.h"
#include <stdint.h>
#include "entityHandler.h"

typedef struct{
   enemy_t *enemyArray[ENEMY_ARR_LENGTH];
} enemyManager_t;

void initEnemyManager(enemyManager_t *enemyManager,enemy_t *enemArr);
void spawnEnemy(enemyManager_t *enemyManager,entityHandler_t *entHan,
    uint8_t xPos, int16_t yPos,int32_t xVel, int32_t yVel,uint8_t
    enemyType, uint8_t height, uint8_t fixedVel, uint8_t powerType);
void spawnRandom(enemyManager_t * enemMan, entityHandler_t * entHan,
    uint8_t maxHeight, int32_t fixedSpeed);
void enemiesShoot(bulletManager_t *bulMan, entityHandler_t *entHan,
    enemyManager_t *enemMan, int32_t bulletSpeed);


#endif /* ENEMYMANAGER_H_ */
```

Listing 37: enemyManager.h

```
#ifndef ENTITY_H
#define ENTITY_H

#include <stdint.h>
#include <stdio.h>
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include "vec.h"
#include "util.h"

// struct of type entity_t
// contains standard fields that all entities will have
// such as position, velocity, hp, and spriteIndex
// also has a setActive field, which is very useful for object pooling in
    our entity handler

typedef struct{
   vector_t pos;
   vector_t vel;
   uint8_t height;
   uint8_t spriteIndex;
   uint8_t isActive;
   uint8_t preX;
   uint8_t preY;
   uint8_t powerType; //0 = nothing, 1 = medkit [Green], 2 = shield
       [Blue], 3 = megabullet [Red]
```

```
} entity_t;

#define G 1 //int

void move(entity_t * ptr);
void updateVel(entity_t * ptr, int8_t x, int8_t y);
void setEntityVel(entity_t * ptr, int8_t x, int8_t y);
void setEntityVelFixed(entity_t *ptr, int32_t x, int32_t y);
void setEntityPos(entity_t * ptr, int16_t x, int16_t y);
void initEntity(entity_t * ptr, uint8_t spriteIndex, uint8_t xPos,
    int16_t yPos,int32_t xVel,int32_t yVel, uint8_t fixedVel, uint8_t
    height,uint8_t powerType);
void destroyEntity(entity_t * ptr);
void clearEntity(entity_t * ptr);
void drawEntity(entity_t * ptr, uint8_t powerType);
void calculateGravity(entity_t * bullet, entity_t * solidObj);
uint8_t detectEntityCollision(entity_t * player, entity_t * obj2);
void checkEntityPos(entity_t * ptr);

#endif
```

Listing 38: entity.h

```
#ifndef ENTITYHANDLER_H_
#define ENTITYHANDLER_H_

// INCLUDES
#include "entity.h"

// DEFINES

#define ENTITY_ARR_LEN 32

typedef struct{
   entity_t *entityArray[ENTITY_ARR_LEN];
} entityHandler_t;

void init_entityHandler(entityHandler_t * ptr, entity_t * entArr);
void updateEntities(entityHandler_t * ptr);
void applyGravity(entityHandler_t * array);
void drawAllEntities(entityHandler_t * ptr);
void clearAllEntities(entityHandler_t * ptr);
uint8_t withinBoundry(entity_t *ptr);

#endif /* ENTITYHANDLER_H_ */
```

Listing 39: entityHandler.h

```
#ifndef INITGAME_H
#define INITGAME_H

#define MODE_CHANGE gs_p->prevMode != gs_p->mode

#define ENTITY_ARR_LEN 32
```

```
#define BULLET_ARR_LENGTH 16
#define ENEMY_ARR_LENGTH 15

#include "entityHandler.h"
#include "bulletManager.h"
#include "enemyManager.h"
#include "player.h"
#include "scoreCalc.h"

typedef struct{
   entityHandler_t entHan;
   entity_t entityArray[ENTITY_ARR_LEN];

   bulletManager_t bulMan;
   bullet_t bulletArray[BULLET_ARR_LENGTH];

   enemyManager_t enemMan;
   enemy_t enemyArray[ENEMY_ARR_LENGTH];

   player_t player;

   gamescore_t score;

   uint8_t LCDbuffer[512];
   uint8_t tickCounter;
   uint8_t spawnCounter;
   uint8_t cooldownCounter;
   uint8_t gameSpeed; //goes from 0 to 15 over a long lasting game
   uint32_t ticks; //counts number of ticks since game launch

   uint8_t mode;
   uint8_t prevMode;
   uint8_t gameInitialized;
   uint8_t playerNum;

} gameStruct_t;

uint8_t game_update();
void initProgram(gameStruct_t * gs_p);
void modeSelect(gameStruct_t * gs_p);
void newMode(gameStruct_t * gs_p);
void initializeGame(gameStruct_t * gs_p);
uint8_t modePicker(uint8_t mode, char input, gameStruct_t * gs_p);
void updateGameSpeed(gameStruct_t * gs_p);
void runGame(gameStruct_t * gs_p, char input);
void usePlayerActionsFromInput(gameStruct_t * gs_p, char input);

#endif  // _INITGAME_
```

Listing 40: gameHandler.h

```
#ifndef GAMEUI_H_
#define GAMEUI_H_
```

```
#include "scoreCalc.h"

void initGameUI(player_t *player, uint8_t gameLevel);
void updateGameUI(player_t *player, gamescore_t *score,uint8_t gameLevel);
void showPlayerHealth(player_t *player);
void showPlayerPowerUp(player_t *player);
void showPlayerScore(gamescore_t *score);
void showGameLevel(uint8_t gameLevel);

#endif /* GAMEUI_H_ */
```

Listing 41: gameUI.h

```
#ifndef HIGHSCORE_H_
#define HIGHSCORE_H_

#include "scoreCalc.h"
#include <stdint.h>

void saveHighscore(char name[], uint32_t score);
char* readHighscoreName(uint8_t place);
uint32_t readHighscore(uint8_t place);
void highscoreFlush();

#endif /* HIGHSCORE_H_ */
```

Listing 42: highscore.h

```
#ifndef LCD_H_
#define LCD_H_

#include "stdint.h"

//lcd_reset() will reboot and reconfigure the LCD
//lcd_transmit_bye() is used to send data and commands to the display,
    but shouldn't be used unless you understand it
//lcd_push_buffer() transmits a byte array of size 512 to the LCD and
    shows the data. This is what I will use to update the display

void initLCD();
void lcd_write_string(char s[], uint8_t slice, uint8_t line,uint8_t
    *LCDbuffer_p);
void lcd_scrolling_text(uint8_t *LCDbuffer_p, uint8_t line, uint8_t
    *remainBytes_p,uint8_t right);
void lcd_draw_sprite(uint8_t * LCDbuffer,int16_t slice, int16_t line,
    uint8_t type, uint8_t mirror);
void lcd_draw_crosshair(uint8_t * LCDbuffer, uint8_t slice, uint8_t line);
void lcd_clear_all(uint8_t * LCDbuffer,uint8_t byte);
void lcd_game_over(uint8_t * LCDbuffer);


#endif /* LCD_H_ */
```

Listing 43: LCD.h

```
#ifndef LCDSPRITES_H_
#define LCDSPRITES_H_

#include "stdint.h"
#include "stdio.h"
#include "stm32f30x_conf.h"
#include "30010_io.h"

extern const uint8_t lcd_sprites[14][4][40];

#endif /* LCDSPRITES_H_ */
```

Listing 44: lcd$_sprites.h$

```
#ifndef JOYSTICK_H
#define JOYSTICK_H

#include <stdint.h>
#include <stdio.h>
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include "gameHandler.h"
#include "LCD.h"

void initLED();
void clearLED();
void setLED(uint8_t x1, uint8_t x2, uint8_t x3);
void setLEDSide(gameStruct_t * gs_p);

#endif  // JOYSTICK_H
```

Listing 45: LED.h

```
#ifndef MEMORY_H_
#define MEMORY_H_

#include <stdint.h>

uint16_t readMemory(uint16_t offset);
void saveToMemory(uint16_t array[], uint8_t len);
void clearMemory();

#endif /* MEMORY_H_ */
```

Listing 46: memory.h

```
#ifndef MENUS_H_
#define MENUS_H_

#include "gameHandler.h"
```

```
#include <stdint.h>

void initMainMenu();
void helpMenu(uint8_t bool);
void bossScreen();
void initGameOverScreen(gameStruct_t * gs_p);
void gameOverScreen(gameStruct_t * gs_p, char input);

#endif /* MENUS_H_ */
```

Listing 47: menus.h

```
#ifndef MENUSAPI_H_
#define MENUSAPI_H_

#include <stdint.h>

int8_t pickMainMenuItems(char key, uint8_t activeItem);
void printMainMenuItems(uint8_t activeItem);
void printScores();
void drawMenuSprites();

#endif /* MENUSAPI_H_ */
```

Listing 48: menusAPI.h

```
#ifndef PLAYER_H_
#define PLAYER_H_

#include <stdint.h>
#include "vec.h"
#include "entity.h"
#include "bulletManager.h"
#include "entityHandler.h"

typedef struct {
   entity_t * entity;
   uint8_t playerNum;
   int8_t gunSide;
   uint8_t powerUp;
   int8_t HP;
   uint8_t crosshairX;
   uint8_t crosshairY;
} player_t;

void initPlayer(entity_t *entity, player_t *player, uint8_t num);
void changeGunside(player_t *player);
void getPowerUp(player_t *player, uint8_t num);
void drawPlayer(player_t *player);
void clearPlayer(player_t *player);
void damagePlayer(player_t *ptr, int8_t x);
void updatePlayerVel(player_t *player, char input);
void damagePlayer(player_t *ptr, int8_t x);
void playerShoot(player_t *ptr, bulletManager_t
```

```
    *bulletManager,entityHandler_t *entHan, uint8_t bulletType, uint8_t
    height);
void updateCrosshair(player_t *ptr,uint8_t joystickVal);
void usePowerUp(player_t * ptr, bulletManager_t *
    bulletManager,entityHandler_t * entHan);
void checkPlayerCollision(player_t * ptr, entityHandler_t * array);

#endif /* PLAYER_H_ */
```

Listing 49: player.h

```
#ifndef PUTTYLCDCONVERTER_H_
#define PUTTYLCDCONVERTER_H_

#include "enemyManager.h"
#include "player.h"
#include "stdint.h"

int32_t con_getVecY(uint8_t slice, int8_t gunside);
int32_t con_getVecX(int8_t gunside);
int8_t con_getDistanceX(uint8_t playerX, uint8_t entX);
uint8_t con_inCone(uint8_t playerX, uint8_t playerY, uint8_t entX,
    uint8_t entY, int8_t gunside);
int16_t con_posToSlice(uint8_t playerX, uint8_t playerY, uint8_t entX,
    uint8_t entY, int8_t gunside);
void con_draw_putty_to_lcd(enemyManager_t *enemMan, player_t
    *player,uint8_t * LCDbuffer);

#endif /* PUTTYLCDCONVERTER_H_ */
```

Listing 50: PuttyLCDConverter.h

```
#ifndef PUTTYSPRITES_H_
#define PUTTYSPRITES_H_

#include <stdint.h>
#include "player.h"

// DEFINES:

void ui_draw_sprite(uint8_t index, uint8_t FGC, uint8_t BGC, uint8_t x,
    uint8_t y);
void ui_clear_sprite(uint8_t index, uint8_t FGC, uint8_t BGC, uint8_t x,
    uint8_t y);
int16_t offsetBulletCoordX(player_t *player);
int16_t offsetBulletCoordXCone(player_t *player);
int16_t offsetBulletCoordY(player_t *player);

#endif /* PUTTYSPRITES_H_ */
```

Listing 51: PuTTYSprites.h

```
#ifndef SCORECALC_H_
#define SCORECALC_H_
```

```
#include <stdint.h>
#include <stdio.h>

typedef struct{
   uint32_t score;
}gamescore_t;

void initScore(gamescore_t * Score);
void incrementScore(gamescore_t * Score, uint32_t points);


#endif /* SCORECALC_H_ */
```

Listing 52: scoreCalc.h

```
#ifndef SERIALREAD_H_
#define SERIALREAD_H_

#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include "stdio.h"
#include "stdint.h"
#include "string.h"

uint8_t get_key_pressed();

#endif /* SERIALREAD_H_ */
```

Listing 53: serialRead.h

```
// ========================================================================
//
// Exported by Cearn's excellut v1.0
// (comments, kudos, flames to daytshen@hotmail.com)
//
// ========================================================================

#ifndef SINLUT_H
#define SINLUT_H

// === LUT SIZES ===
#define SIN_SIZE 512

#include "stdint.h"

int16_t lutSin(int16_t degrees);
int16_t lutCos(int16_t degrees);
int32_t expand(int32_t i);
void printFix(int32_t i);

// === LUT DECLARATIONS ===
extern const signed short SIN[512];
```

```
#endif  // SINLUT_H
```

Listing 54: sinLut.h

```
/*
 * stopwatch.h
 *
 *  Created on: 6. jun. 2023
 *      Author: georg
 */

#ifndef STOPWATCH_H_
#define STOPWATCH_H_

#include "stdint.h"

extern volatile uint8_t timer_Flag;

void initTimer();
void TIM1_BRK_TIM15_IRQHandler(void);


#endif /* STOPWATCH_H_ */
```

Listing 55: stopwatch.h

```
#ifndef UTIL_H_
#define UTIL_H_

#include "stdint.h"

int32_t getRandomInterval(int32_t min, int32_t max);
int32_t absolute(int32_t x);
int32_t getManDistance(int32_t x1, int32_t y1, int32_t x2, int32_t y2);
int32_t mapInterval(int32_t minOld, int32_t maxOld,int32_t minNew,
    int32_t maxNew,int32_t value);
int32_t capInterval(int32_t value, int32_t min, int32_t max);
void incrementCounter(uint8_t * ptr, uint8_t count);
void resetCounter(uint8_t * ptr);
int32_t norm(int32_t x);


#endif /* UTIL_H_ */
```

Listing 56: util.h

```
#ifndef VEC_H
#define VEC_H

#include "stdint.h"

typedef struct{
   int32_t x,y;
} vector_t;
```

```
void rotateVector(vector_t *v, int degrees);
void setVectorInt(vector_t *v, int32_t x, int32_t y);
int16_t getXint(vector_t *v);
int16_t getYint(vector_t *v);
void updateVectorInt(vector_t *v, int32_t x, int32_t y);

#endif  // VEC_H
```

Listing 57: vec.h

Journal of exercises

The journal was completed and documented using Google Docs, and as it is already extremely long and structured in its own way, we will simply merge the two PDF-files to keep the structure of both documents intact without causing major problems to readability.

Journal - Gruppe 15

Indhold

Opgave 0

- Calculate the negative of the numbers 56, 178, 1002, and 7586 in binary form (8 or 16 bits) using the two's-complement representations. I.e., invert every single bit and add binary one:

	56	178	1002	7586
Positive number	00111000 (8)			
One's Complement (neg.)	11000111 (8)			
+1	+1			
Two's Complement (neg.)	11001000 (8)			

	56	178	1002	7586
Positive number	00111000	10110010	00000011 11101010	00011101 10100010
One's complement	11000111	01001101	11111100 00010101	11100010 01011101
+1	+1	+1	+1	+1
Two's complement	11001000	01001110	11111100 00010110	11100010 01011110

- Express the decimal numbers 56, 178, 1002, and 7586 in binary form using the Unsigned, Signed-magnitude, One's-complement, Two's-complement, and Biased representations. Also note the number of bits (8 or 16) needed for the representations.

		Unsigned	Signed-Magnitude	Ones Complement	Twos Complement	Biased
56		00111000 (8)	00111000 (8)	00111000 (8)	00111000 (8)	10110111 (8)
178						
1002						
7586						

		Unsigned	Signed-mag	Ones-comp	Twos-Comp	Biased
56		00111000 (8)	00111000 (8)	00111000 (8)	00111000 (8)	10110111 (8)
178		10110010 (8)	00000000 10110010 (16)	10110010 (8)	10110010 (8)	10000000 10110001 (16)
1002		00000011 11101010 (16)	00000011 11101010(16)	00000011 11101010 (16)	00000011 11101010 (16)	10000011 11101001 (16)
7586		00011101 10100010 (16)	00011101 10100010(16)	00011101 10100010 (16)	11100010 01011110(16)	10011101 10100001 (16)

Subtract two numbers by using the following "trick": A-B is the same as A+(-B). This means that if we invert B (by using two's complement), we can just add the two numbers A and (-B).

	(	11110000)			(	)	
56	56		00111000	178	178		
-56	+(-56)	+	11001000	-177	+(-177)	+	
0	0		100000000	1	1		
	(	)			(	)	
7576	7576			1001	1001		
-7586	+(-7586)	+		-1002	+(-1002)	+	
-10	-10			-1	-1		

Carry		1	1	1	1	1	1		
178		1	0	1	1	0	0	1	0
+(-177)		0	1	0	0	1	1	1	1
=1	1	0	0	0	0	0	0	0	1

Carry									
1001		00	00	00	11	11	10	10	01
+(-1002)		11	11	11	00	00	01	01	10
-1		11	11	11	11	11	11	11	11

Carry							11		
7576		00	01	11	01	10	01	10	00
+(-7586)		11	10	00	10	01	01	11	10
=-10		11	11	11	11	11	11	01	10

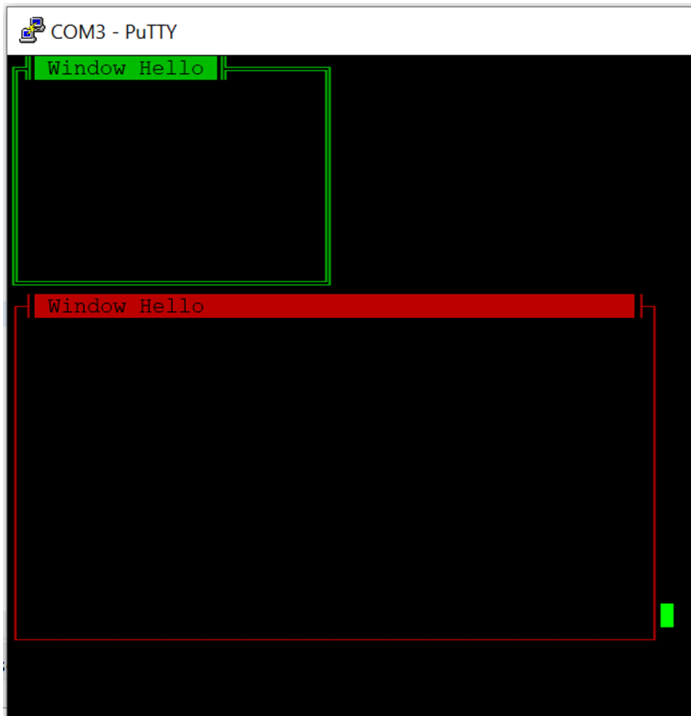
Opgave 1

1.1

```
void clrscrn(){
    printf("%c[2J", ESC);
}

void clrRLine(){
    printf("%c[K", ESC);
}

void goToXY(uint8_t X, uint8_t Y){
    printf("%c[%d;%dH", ESC, Y, X);
}

void toggleUnderline(uint8_t on){
    printf("%c[%dH", ESC, on ? 04 : 24);
}

void toggleInverse(uint8_t on){
    printf("%c[%dH", ESC, on ? 07 : 27);
}

void toggleBlink(uint8_t on){
```

```
    printf("%c[%dH", ESC, on ? 05 : 25);
}

void moveCursorX(uint8_t X, uint8_t UP){ //GOING RIGHT/LEFT
    printf("%c[%d%c", ESC, X,  UP ? 'A' : 'B');
}

void moveCursorY(uint8_t Y, uint8_t UP){ //GOING UP/DOWN
    printf("%c[%d%c", ESC, Y,  UP ? 'A' : 'B');
}
```

1.2



```
11
12 int main(void)
13 {
14     // Setup communication with the PC
15     uart_init(9600);
16     clrscr();
17     fgcolor(2);
18     window(1,1,25,10,"Hello",1);
19
20     fgcolor(1);
21     window(1,11,50,25,"Hello",2);
22     fgcolor(0);
23     while(1){}
24 }
25
```

```
98 void window(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, char title[], uint8_t style){
99     if(style == 1)
100    {
101        //Top left corner
102        gotoxy(x1,y1);
103        printf("%c",201);
104        //Top right corner
105        gotoxy(x2,y1);
106        printf("%c",187);
107        //Bottom right corner
108        gotoxy(x2,y2);
109        printf("%c",188);
110        //Bottom left corner
111        gotoxy(x1,y2);
112        printf("%c",200);
113
114        // Borders for title:
115        gotoxy(x1+1,y1);
116        printf("%c",185);
117
118
119        char windowTekst[] = " Window ";
120        strcat(windowTekst, title);
121        strcat(windowTekst, " ");
122
123        inverse(1);
124        for(int i = 0; i < x2-x1-2; i++)
125        {
126            gotoxy(x1+2+i,y1);
127            if(i < strlen(windowTekst)){
128                printf("%c",windowTekst[i]);
```

```
            }
            else if(i == strlen(windowTekst)){
                inverse(0);
                printf("%c",204);
            }
            else{
                printf("%c",205);
            }
        }
        inverse(0);
        for(int i = x1+1; i < x2; i++){
            gotoxy(i,y2);
            printf("%c",205);
        }
        for(int i = y1+1; i < y2; i++){
            gotoxy(x1,i);
            printf("%c",186);
            gotoxy(x2,i);
            printf("%c",186);
        }
    }
    else if(style == 2)
    {
        //Top left corner
        gotoxy(x1,y1);
        printf("%c",218);
        //Top right corner
        gotoxy(x2,y1);
        printf("%c",191);
        //Bottom right corner
        gotoxy(x2,y2);
        printf("%c",217);
```

```
        //Bottom left corner
        gotoxy(x1,y2);
        printf("%c",192);

        //Borders for title:
        gotoxy(x1+1,y1);
        printf("%c",180);
        gotoxy(x2-1,y1);
        printf("%c",195 );

        char windowTekst[] = " Window ";
        strcat(windowTekst, title);
        strcat(windowTekst, " ");

        inverse(1);
        for(int i = 0; i < x2-x1-3; i++)
        {
            gotoxy(x1+2+i,y1);
            if(i < strlen(windowTekst)){
                printf("%c",windowTekst[i]);
            }
            else{
                printf(" ");
            }
        }
        inverse(0);
        for(int i = x1+1; i < x2; i++){
            gotoxy(i,y2);
            printf("%c",196);
        }
```

```
          for(int i = y1+1; i < y2; i++){
              gotoxy(x1,i);
              printf("%c",179);
              gotoxy(x2,i);
              printf("%c",179);
          }
      }
}
```

1.3

HEX	DEC	BINARY
0x14	20	00010100
0xE6	224	11100000
0x58	88	01011000
0xB7	183	10110111
0x9F	159	10011111
0x3D	61	00111101

Turn on BITS 4 and 7 in a byte -> IRQ |= 0x48 (bitwise | med (01001000)

Turn off BITS 2 and 5 in a byte -> IRQ &= 0xED (bitwise & med 11101101)

Writing -1 in binary, assuming size of a byte: (twos complement) 11111111

Writing -2 in binary, assuming size of a byte: (twos complement) 11111110

DEC (IF UNSIGNED)	DEC (IF SIGNED) (twos-comp)	BINARY
8	8	00001000
254	-2	11111110

	HIGHEST		LOWEST	
	BINARY	DECIMAL	BINARY	DECIMAL
UNSIGNED	11111111	255	00000000	0
SIGNED	01111111	127	10000000	-128

How do you tell if a signed byte is positive or negative?

Look at the MSB, if it's 1, the byte is negative.

How do you change the sign of a binary number?

Invert and add one.

How do you multiply a binary number by two?

Bit shift one to the left.

How do you divide a binary number by two?

Bit shift one to the right.

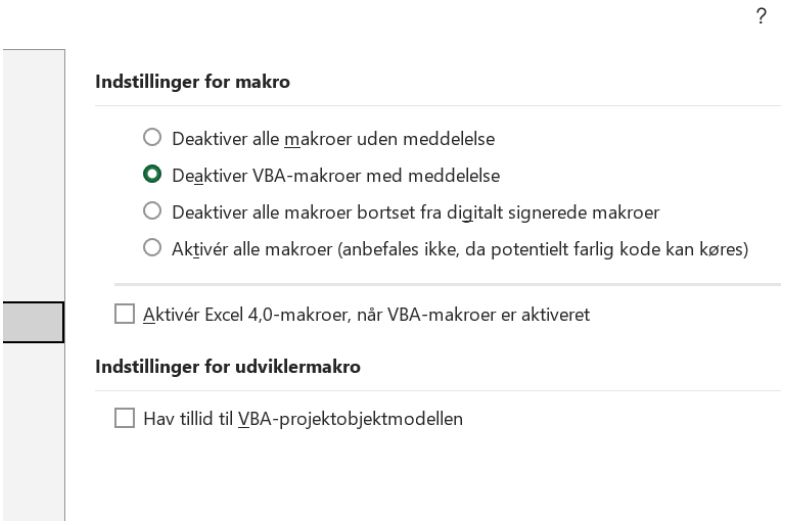
When dividing by two, how is rounding performed?

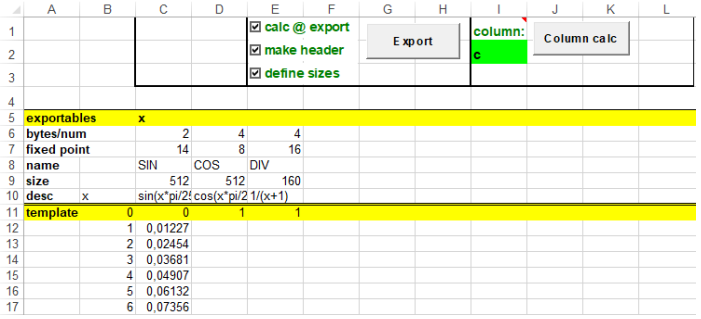
Rounded down – look for example at dividing 11 by two: 1011 -> 0101 (11 -> 5)

	Binary	HEX
0x12 | 0x01	00010011	0x13
0x13 & 0xFE	00010010	0x12
~0x3F	11000000	0xC0
~0x01	11111110	0xFE
0x0C >> 2	00000011	0x03
0x0C >> 4	00000000	0x00
0x0C << 4	11000000	0xC0

Opgave 2

2.1






```
#include "realLUT.h"

//
-----------------------------------------------------------------------
// SIN: a 512 long LUT of 16bit values in 2.14 format
// sin(x*pi/256)
const signed short SIN[512]=
{
    0x0000,0x00C9,0x0192,0x025B,0x0324,0x03ED,0x04B5,0x057E,
    0x0646,0x070E,0x07D6,0x089D,0x0964,0x0A2B,0x0AF1,0x0BB7,
    0x0C7C,0x0D41,0x0E06,0x0ECA,0x0F8D,0x1050,0x1112,0x11D3,
    0x1294,0x1354,0x1413,0x14D2,0x1590,0x164C,0x1709,0x17C4,
    0x187E,0x1937,0x19EF,0x1AA7,0x1B5D,0x1C12,0x1CC6,0x1D79,
    0x1E2B,0x1EDC,0x1F8C,0x203A,0x20E7,0x2193,0x223D,0x22E7,
    0x238E,0x2435,0x24DA,0x257E,0x2620,0x26C1,0x2760,0x27FE,
    0x289A,0x2935,0x29CE,0x2A65,0x2AFB,0x2B8F,0x2C21,0x2CB2,
```

```
0x2D41,0x2DCF,0x2E5A,0x2EE4,0x2F6C,0x2FF2,0x3076,0x30F9,
0x3179,0x31F8,0x3274,0x32EF,0x3368,0x33DF,0x3453,0x34C6,
0x3537,0x35A5,0x3612,0x367D,0x36E5,0x374B,0x37B0,0x3812,
0x3871,0x38CF,0x392B,0x3984,0x39DB,0x3A30,0x3A82,0x3AD3,
0x3B21,0x3B6D,0x3BB6,0x3BFD,0x3C42,0x3C85,0x3CC5,0x3D03,
0x3D3F,0x3D78,0x3DAF,0x3DE3,0x3E15,0x3E45,0x3E72,0x3E9D,
0x3EC5,0x3EEB,0x3F0F,0x3F30,0x3F4F,0x3F6B,0x3F85,0x3F9C,
0x3FB1,0x3FC4,0x3FD4,0x3FE1,0x3FEC,0x3FF5,0x3FFB,0x3FFF,

0x4000,0x3FFF,0x3FFB,0x3FF5,0x3FEC,0x3FE1,0x3FD4,0x3FC4,
0x3FB1,0x3F9C,0x3F85,0x3F6B,0x3F4F,0x3F30,0x3F0F,0x3EEB,
0x3EC5,0x3E9D,0x3E72,0x3E45,0x3E15,0x3DE3,0x3DAF,0x3D78,
0x3D3F,0x3D03,0x3CC5,0x3C85,0x3C42,0x3BFD,0x3BB6,0x3B6D,
0x3B21,0x3AD3,0x3A82,0x3A30,0x39DB,0x3984,0x392B,0x38CF,
0x3871,0x3812,0x37B0,0x374B,0x36E5,0x367D,0x3612,0x35A5,
0x3537,0x34C6,0x3453,0x33DF,0x3368,0x32EF,0x3274,0x31F8,
0x3179,0x30F9,0x3076,0x2FF2,0x2F6C,0x2EE4,0x2E5A,0x2DCF,

0x2D41,0x2CB2,0x2C21,0x2B8F,0x2AFB,0x2A65,0x29CE,0x2935,
0x289A,0x27FE,0x2760,0x26C1,0x2620,0x257E,0x24DA,0x2435,
0x238E,0x22E7,0x223D,0x2193,0x20E7,0x203A,0x1F8C,0x1EDC,
0x1E2B,0x1D79,0x1CC6,0x1C12,0x1B5D,0x1AA7,0x19EF,0x1937,
0x187E,0x17C4,0x1709,0x164C,0x1590,0x14D2,0x1413,0x1354,
0x1294,0x11D3,0x1112,0x1050,0x0F8D,0x0ECA,0x0E06,0x0D41,
0x0C7C,0x0BB7,0x0AF1,0x0A2B,0x0964,0x089D,0x07D6,0x070E,
0x0646,0x057E,0x04B5,0x03ED,0x0324,0x025B,0x0192,0x00C9,

0x0000,0xFF37,0xFE6E,0xFDA5,0xFCDC,0xFC13,0xFB4B,0xFA82,
0xF9BA,0xF8F2,0xF82A,0xF763,0xF69C,0xF5D5,0xF50F,0xF449,
0xF384,0xF2BF,0xF1FA,0xF136,0xF073,0xEFB0,0xEEEE,0xEE2D,
0xED6C,0xECAC,0xEBED,0xEB2E,0xEA70,0xE9B4,0xE8F7,0xE83C,
0xE782,0xE6C9,0xE611,0xE559,0xE4A3,0xE3EE,0xE33A,0xE287,
0xE1D5,0xE124,0xE074,0xDFC6,0xDF19,0xDE6D,0xDDC3,0xDD19,
0xDC72,0xDBCB,0xDB26,0xDA82,0xD9E0,0xD93F,0xD8A0,0xD802,
0xD766,0xD6CB,0xD632,0xD59B,0xD505,0xD471,0xD3DF,0xD34E,

0xD2BF,0xD231,0xD1A6,0xD11C,0xD094,0xD00E,0xCF8A,0xCF07,
0xCE87,0xCE08,0xCD8C,0xCD11,0xCC98,0xCC21,0xCBAD,0xCB3A,
0xCAC9,0xCA5B,0xC9EE,0xC983,0xC91B,0xC8B5,0xC850,0xC7EE,
0xC78F,0xC731,0xC6D5,0xC67C,0xC625,0xC5D0,0xC57E,0xC52D,
0xC4DF,0xC493,0xC44A,0xC403,0xC3BE,0xC37B,0xC33B,0xC2FD,
0xC2C1,0xC288,0xC251,0xC21D,0xC1EB,0xC1BB,0xC18E,0xC163,
0xC13B,0xC115,0xC0F1,0xC0D0,0xC0B1,0xC095,0xC07B,0xC064,
0xC04F,0xC03C,0xC02C,0xC01F,0xC014,0xC00B,0xC005,0xC001,
```

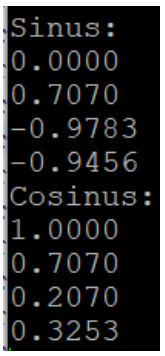
```
    0xC000,0xC001,0xC005,0xC00B,0xC014,0xC01F,0xC02C,0xC03C,
    0xC04F,0xC064,0xC07B,0xC095,0xC0B1,0xC0D0,0xC0F1,0xC115,
    0xC13B,0xC163,0xC18E,0xC1BB,0xC1EB,0xC21D,0xC251,0xC288,
    0xC2C1,0xC2FD,0xC33B,0xC37B,0xC3BE,0xC403,0xC44A,0xC493,
    0xC4DF,0xC52D,0xC57E,0xC5D0,0xC625,0xC67C,0xC6D5,0xC731,
    0xC78F,0xC7EE,0xC850,0xC8B5,0xC91B,0xC983,0xC9EE,0xCA5B,
    0xCAC9,0xCB3A,0xCBAD,0xCC21,0xCC98,0xCD11,0xCD8C,0xCE08,
    0xCE87,0xCF07,0xCF8A,0xD00E,0xD094,0xD11C,0xD1A6,0xD231,

    0xD2BF,0xD34E,0xD3DF,0xD471,0xD505,0xD59B,0xD632,0xD6CB,
    0xD766,0xD802,0xD8A0,0xD93F,0xD9E0,0xDA82,0xDB26,0xDBCB,
    0xDC72,0xDD19,0xDDC3,0xDE6D,0xDF19,0xDFC6,0xE074,0xE124,
    0xE1D5,0xE287,0xE33A,0xE3EE,0xE4A3,0xE559,0xE611,0xE6C9,
    0xE782,0xE83C,0xE8F7,0xE9B4,0xEA70,0xEB2E,0xEBED,0xECAC,
    0xED6C,0xEE2D,0xEEEE,0xEFB0,0xF073,0xF136,0xF1FA,0xF2BF,
    0xF384,0xF449,0xF50F,0xF5D5,0xF69C,0xF763,0xF82A,0xF8F2,
    0xF9BA,0xFA82,0xFB4B,0xFC13,0xFCDC,0xFDA5,0xFE6E,0xFF37,
};
```

2.2

```
int32_t expand(int32_t i) {
    // Converts an 18.14 fixed point number to 16.16
    return i << 2;
}

int32_t Sinosoid(int16_t x){ //takes inputs based on 512 step circle (64
= 45*)
    x &= 511;
    return SIN[x];
}

int32_t Cosinoid(int16_t numba){
    return Sinosoid(numba + 128) ;
}

void printFix(int32_t i) {
    // Prints a signed 16.16 fixed point number
    if ((i & 0x80000000) != 0) { // Handle negative numbers
       printf("-");
       i = ~i + 1;
    }
    printf("%ld.%04ld", i >> 16, 10000 * (uint32_t)(i & 0xFFFF) >> 16);
```

```
    // Print a maximum of 4 decimal digits to avoid overflow
}
```

```
void sineTest();

int main(void)
{
    sineTest();
}


void sineTest(){
    uart_init(9600);
    clrscrn();
    printf("Sinus:\n");
    printFix(expand(Sinosoid(0)));
    printf("\n");
    printFix(expand(Sinosoid(64)));
    printf("\n");
    printFix(expand(Sinosoid(-111)));
    printf("\n");
    printFix(expand(Sinosoid(923)));
    printf("\n");
    printf("Cosinus:\n");
    printFix(expand(Cosinoid(0)));
    printf("\n");
    printFix(expand(Cosinoid(64)));
    printf("\n");
    printFix(expand(Cosinoid(-111)));
    printf("\n");
    printFix(expand(Cosinoid(923)));
    printf("\n");

    while(1){}
}
```



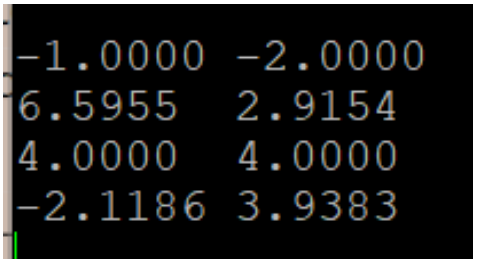
2.3

```c
#include "vectored.h"

void initVector(vector_t *v, int16_t x1, int16_t y1){
    v->x = x1;
    v->y = y1;
}

void rotateVectorByReference(vector_t *v, int16_t angle){
    int32_t temp = v->x;
    v->x = temp * Cosinoid(angle) - v->y * Sinosoid(angle);
    v->y = temp * Sinosoid(angle) + v->y * Cosinoid(angle);
}
```

```c
#ifndef VECTORED_H_
#define VECTORED_H_

#include <stdint.h>
#include <stdio.h>
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h"

typedef struct {
    int32_t x,y;
} vector_t;

void initVector(vector_t *v, int16_t x1, int16_t y1);
```

```
void rotateVectorByReference(vector_t *v, int16_t angle);

#endif /* VECTORED_H_ */
```

```
void rotateTest();

int main(void)
{
    rotateTest();
}

void rotateTest(){
    uart_init(9600);
    clrscrn();
    vector_t vec;
    initVector(&vec,1,2);
    rotateVectorByReference(&vec,256);
    printFix(expand(vec.x));
    printf("\t");
    printFix(expand(vec.y));
    printf("\n");
    initVector(&vec,6,4);
    rotateVectorByReference(&vec,-14);
    printFix(expand(vec.x));
    printf("\t");
    printFix(expand(vec.y));
    printf("\n")
    initVector(&vec,-4,-4);
    rotateVectorByReference(&vec,1280);
    printFix(expand(vec.x));
    printf("\t");
    printFix(expand(vec.y));
    printf("\n")
    initVector(&vec,-4,2);
    rotateVectorByReference(&vec,-50);
    printFix(expand(vec.x));
    printf("\t");
    printFix(expand(vec.y));
    printf("\n")

    while(1){}
}
```


-1.0000 -2.0000
6.5955 2.9154
4.0000 4.0000
-2.1186 3.9383

2.4

I denne opgave skal man antage 8.8 notation

30. Man sætter charens 8 bits ind på de 8 mest signifikante bits i 8.8 notationen.

31. Man tager de otte mest signifikante bits og bruger dem som char? Dvs man runder bare ned og bruge det som char?

32. På 8.8 form ville det være 2^-8

33. De adderes på samme måde som normale binære tal

34. Man multiplikerer de to fixed point tal som om de bar normale binære tal og så right shifter man med 8 bits.

35. Samme metode som med to fixed point tal. Man skal være forsigtig idet der kan ske overflow.

36. Det bliver rundet ned

37. Hvis den første bit efter punktummet er 1 så skal der rundes op. Det gøres ved at plusse med 0000 0000 . 1000 0000 og så konvertere til char

38. 0.125 i hexadecimal er 0x00.20

39. 0.25 i hexadecimal er 0x00.40

40. 2.125 i hexadecimal er 0x02.20

41. 2.25 i hexadecimal er 0x02.40

42.

	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
*	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
=	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
>> 8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Det endelige resultat er altså $(0000\ 0000\ .0000\ 1000)_2 = (0.03125)_{10}$

43. $(2.125 \cdot 2.25)_{10} = (0000\ 0010.0010\ 0000 \cdot 0000\ 0010.0100\ 0000)_2$

				0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
*				0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
				1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
+	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
=	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
>> 8				0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0

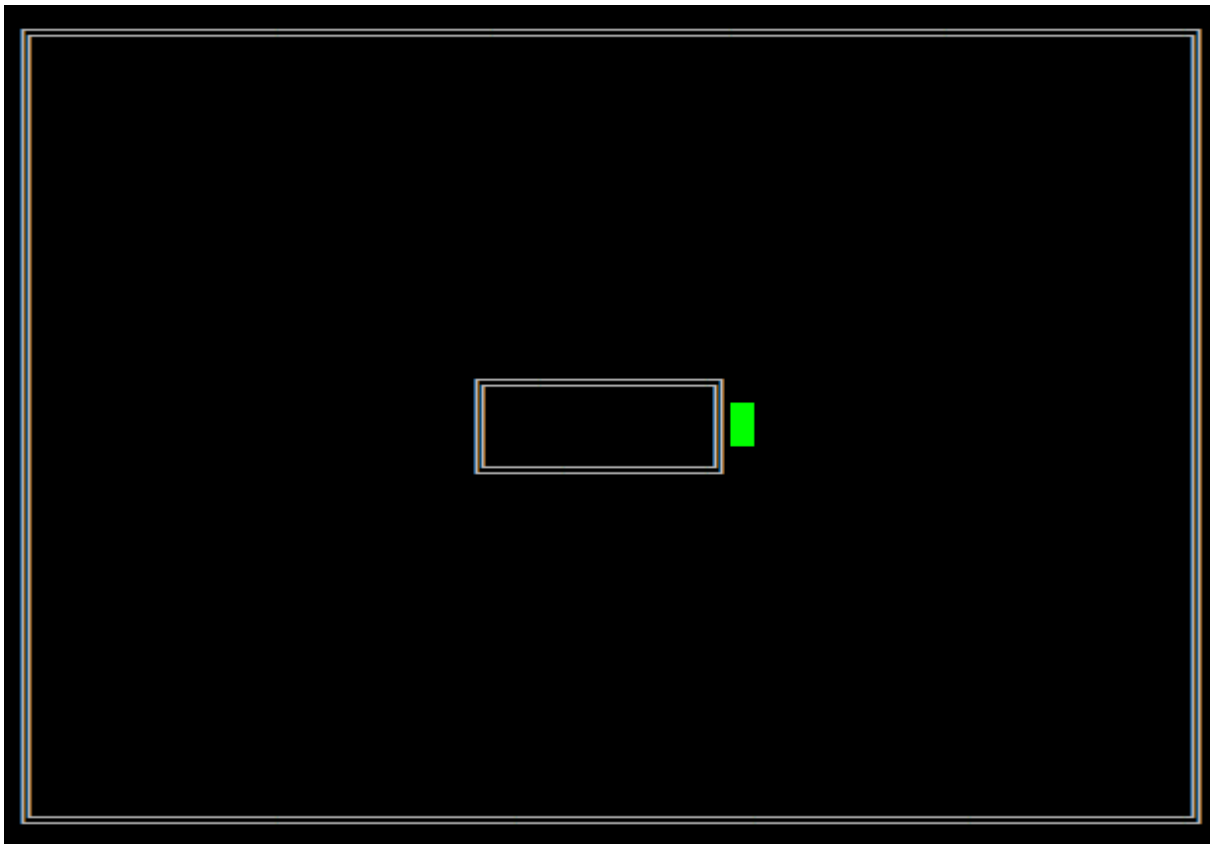
Det endelige resultat er $(4.78125)_{10} = (0000\ 0100\ .1100\ 1000)_2$

44. Overflow - vores data gemmer ikke de tre røde bits, der i programmet vil blive skåret af. Skader ikke, hvis det kun er 0'ere som i 42, men i 43 spiller det en rolle.

45. Mulig løsning #1 : Introducer temporary memory, der kan holde vores overflow. Maksimum skal vi allokere 16 ekstra bits som temporary værdi.
Mulig løsning #2 : da vi alligevel skal bitshifte 8 bits mod højre er vi ligeglade med de mindste 8 bits. Det er muligt at vores gangestykke giver os et 32 bit resultat, så de sidste 8 bits overflow skal gemmes i en temporary memory.

Opgave 3

3.1



```
20      clrscr();
21      window(1,1,50,19,"",3);
22      window(20,9,30,11,"",3);
```

Inde i window metoden kan man nu vælge style == 3:

```
198     else if(style == 3)
199     {
200         //Top left corner
201         gotoxy(x1,y1);
202         printf("%c",201);
203         //Top right corner
204         gotoxy(x2,y1);
205         printf("%c",187);
206         //Bottom right corner
207         gotoxy(x2,y2);
208         printf("%c",188);
209         //Bottom left corner
210         gotoxy(x1,y2);
211         printf("%c",200);
212 
213         for(int i = x1+1; i < x2; i++)
214         {
215             gotoxy(i,y2);
216             printf("%c",205);
217             gotoxy(i,y1);
218             printf("%c",205);
219         }
220 
221         for(int i = y1+1; i < y2; i++)
222         {
223             gotoxy(x1,i);
224             printf("%c",186);
225             gotoxy(x2,i);
226             printf("%c",186);
227         }
228     }
229 }
```

3.2

```
1  #ifndef BALL_H
2  #define BALL_H
3  
4  #include "stdint.h"
5  #include "stdio.h"
6  #include "vec.h"
7  #include "ansi.h"
8  
9  typedef struct{
10     vector_t pos,vel;
11 } ball_t;
12 
13 //Insert function declarations:
14 void setPos(ball_t * ball,int32_t x, int32_t y);
15 void setVel(ball_t * ball,int32_t x, int32_t y);
16 void updatePos(ball_t * ball, int32_t k);
17 void drawBall(ball_t * ball);
18 void initBall(ball_t * ball);
19 int overlap(ball_t * ball,uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2);
20 
21 #endif  // BALL_H
```

```c
#include "ball.h"

void setPos(ball_t * ball,int32_t x, int32_t y)
{
    initVector(&(ball->pos),x,y);
}

void setVel(ball_t * ball,int32_t x, int32_t y)
{
    initVector(&(ball->vel),x,y);
}

void initBall(ball_t * ball)
{
    vector_t pos;
    vector_t vel;
    ball->pos = pos;
    ball->vel = vel;
}


void updatePos(ball_t * ball, int32_t k)
{
    ball->pos.x = ball->pos.x + (((k << 14)*ball->vel.x) >> 14);
    ball->pos.y = ball->pos.y + (((k << 14)*ball->vel.y) >> 14);
}
```

```c
void drawBall(ball_t * ball)
{
    gotoxy(ball->pos.x >> 14,ball->pos.y >> 14);
    printf("o");
}

int overlap(ball_t * ball,uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2)
{
    //temps
    int32_t x = ball->pos.x >> 14;
    int32_t y = ball->pos.y >> 14;

    //Corner cases:
    if(x  == x1+1 && y == y1+1){return 3;}
    if(x == x2-1 && y == y1+1){return 3;}
    if(x == x1+1 && y == y2-1){return 3;}
    if(x == x2-1 && y == y2-1){return 3;}

    //Wall cases:
    if(x <= x1+1 || x >= x2-1){return 1;}
    if(y <= y1+1 || y >= y2-1){return 2;}

    //No collision:
    return 0;
}

```

```c
int main(void)
{
    // Setup communication with the PC
    uart_init(115200);

    //Init ball:
    ball_t ball;
    initBall(&ball);
    setPos(&ball,5,5);
    setVel(&ball,1,1);

    int32_t tickCounter = 0;
    int32_t collisionCounter = 0;

    while(1)
    {
        if(tickCounter == 1000000)//Based on a counter to slow it down after we rose the baud rate
        {
            clrscr();

            //Draw hits window:
            window(20,9,30,11,"",3);
            gotoxy(21,10);
            printf("Hits: %d", collisionCounter);

            //Draw ball:
            updatePos(&ball,1);
            drawBall(&ball);


            //Draw window
            window(1,1,50,19,"",3);

            uint8_t normal = overlap(&ball,1,1,50,19);

            switch(normal)
            {
                case 0: //Nothing hit
                    break;
                case 1: //Side-wall hit
                    ball.vel.x *= -1;
                    collisionCounter++;
                    break;
                case 2: //Top or bottom -wall hit
                    ball.vel.y *= -1;
                    collisionCounter++;
                    break;
                case 3: //corner hit
                    rotateVector(&(ball.vel), 256); //rotate by 180 degrees
                    collisionCounter++;
                    break;
                default:
                    break;
            }

            tickCounter = 0;
        }
        tickCounter++;
    }
}
```

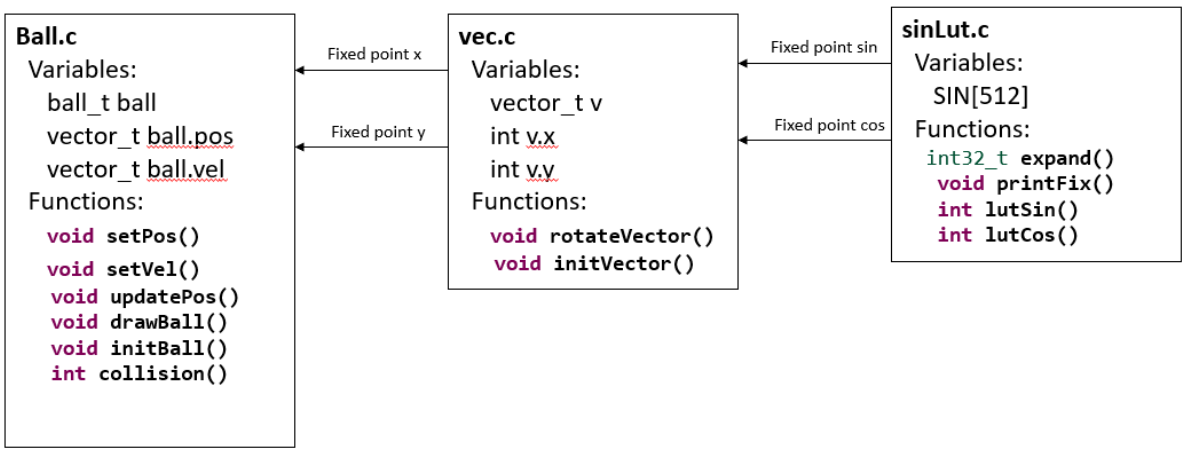

Hits: 80


Hits: 1001


Hits: 3860

3.3

Bouncing Ball:



setPos Sets the position vector af a ball to the given values

---

Syntax: void setPos(ball_t * ball,int32_t x, int32_t y);

Parameters: *ball*: pointer to the struct for the ball

x: new desired value for the x coordinate in the pos vector in ball

y: new desired value for the y coordinate in the pos vector in ball

The function sets the position vector af a ball to the given values in 18.14 fixed point representation.

setVel Sets the velocity vector af a ball to the given values

---

Syntax: void setVel(ball_t * ball,int32_t x, int32_t y);

Parameters: *ball*: pointer to the struct for the ball

x: new desired value for the x coordinate in the vel vector in ball

y: new desired value for the y coordinate in the vel vector in ball

The function sets the velocity vector af a ball to the given values in 18.14 fixed point representation.

updatePos Updates position of a ball

_____

Syntax: void updatePos(ball_t * ball, int32_t k);

Parameters: *ball*: pointer to the struct for the ball

k: scaling factor for velocity

The function updates the position of a ball by adding its velocity vector multiplied by factor *k* to its position. Everything is calculated in 18.14 fixed point representation.

drawBall Draws the ball at its position

_____

Syntax: void void drawBall(ball_t * ball)

Parameters: *ball*: pointer to the struct for the ball

The function draws the ball as an 'o' by using *printf()* at its position by using *gotoxy()* and reading only the whole numbers of the 18.14 fixed point representation..

initBall Initializes the struct members of the ball

_____

Syntax: void initBall(ball_t * ball);

Parameters: *ball*: pointer to the struct for the ball

The function initializes the vector struct members of the ball without setting the members of the vectors.

collision Checks for collision between ball and wall

_____

Syntax: int collision(ball_t * ball,uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2);

Parameters: *ball*: pointer to the struct for the ball

x1: x-coordinate of upper left corner of the window edges

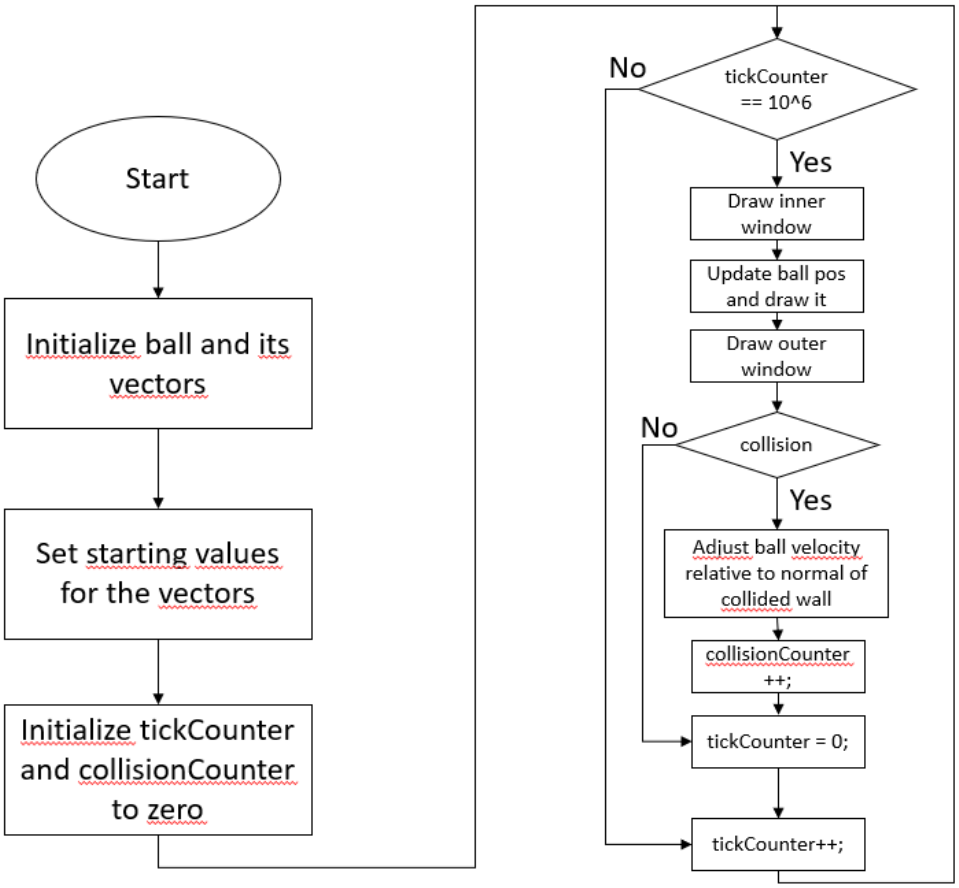
y1: y-coordinate of upper left corner of the window edges

x2: x-coordinate of bottom right corner of the window edges

y2: y-coordinate of bottom right corner of the window edges

The function checks if the ball is about to collide with a wall and returns a number from 0 to 3 based on what wall it is:

- 0: No collision
- 1: Collision with side walls
- 2: Collision with top or bottom wall
- 3: Collision in exactly one of the four corners of the window


Start
Initialize ball and its vectors
Set starting values for the vectors
Initialize tickCounter and collisionCounter to zero
No
tickCounter == 10^6
Yes
Draw inner window
Update ball pos and draw it
Draw outer window
No
collision
Yes
Adjust ball velocity relative to normal of collided wall
collisionCounter ++;
tickCounter = 0;
tickCounter++;

Opgave 4

4.1

Initialisering af joystick pins:

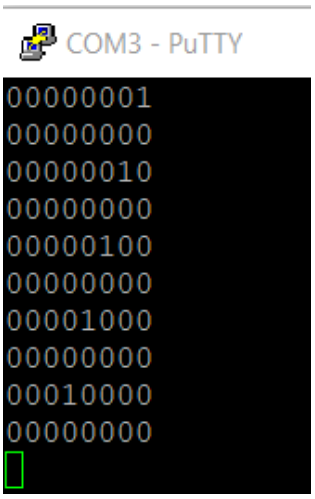
```
void initJoystick(){

    RCC->AHBENR |= RCC_AHBPeriph_GPIOA; // Enable clock for GPIO Port A
    RCC->AHBENR |= RCC_AHBPeriph_GPIOB; // Enable clock for GPIO Port B
    RCC->AHBENR |= RCC_AHBPeriph_GPIOC; // Enable clock for GPIO Port C

    //RIGHT
    // Set pin PC0 to input
    GPIOC->MODER &= ~(0x00000003 << (0 * 2)); // Clear mode register
    GPIOC->MODER |= (0x00000000 << (0 * 2)); // Set mode register (0x00
- Input)
    GPIOC-> PUPDR &= ~(0x00000003 << (0 * 2)); // Clear push/pull
register
    GPIOC->PUPDR |= (0x00000002 << (0 * 2)); // Set push/pull register
(0x02 - Pull-down)

    //LEFT
    // Set pin PC1 to input
    GPIOC->MODER &= ~(0x00000003 << (1 * 2)); // Clear mode register
    GPIOC->MODER |= (0x00000000 << (1 * 2)); // Set mode register (0x00
- Input)
    GPIOC-> PUPDR &= ~(0x00000003 << (1 * 2)); // Clear push/pull
register
    GPIOC->PUPDR |= (0x00000002 << (1 * 2)); // Set push/pull register
(0x02 - Pull-down)

    //UP
    // Set pin PA4 to input
    GPIOA->MODER &= ~(0x00000003 << (4 * 2)); // Clear mode register
    GPIOA->MODER |= (0x00000000 << (4 * 2)); // Set mode register (0x00
- Input)
    GPIOA-> PUPDR &= ~(0x00000003 << (4 * 2)); // Clear push/pull
register
    GPIOA->PUPDR |= (0x00000002 << (4 * 2)); // Set push/pull register
(0x02 - Pull-down)

    //DOWN
    // Set pin PB0 to input
```

```
    GPIOB->MODER &= ~(0x00000003 << (0 * 2)); // Clear mode register
    GPIOB->MODER |= (0x00000000 << (0 * 2)); // Set mode register (0x00
- Input)
    GPIOB-> PUPDR &= ~(0x00000003 << (0 * 2)); // Clear push/pull
register
    GPIOB->PUPDR |= (0x00000002 << (0 * 2)); // Set push/pull register
(0x02 - Pull-down)

    //CENTER
    // Set pin PB5 to input
    GPIOB->MODER &= ~(0x00000003 << (5 * 2)); // Clear mode register
    GPIOB->MODER |= (0x00000000 << (5 * 2)); // Set mode register (0x00
- Input,)
    GPIOB-> PUPDR &= ~(0x00000003 << (5 * 2)); // Clear push/pull
register
    GPIOB->PUPDR |= (0x00000002 << (5 * 2)); // Set push/pull register
(0x02 - Pull-down)
}
```

Kode for at læse værdier på joystick-pins:

```
uint8_t readJoystick(){

    uint8_t values = 0;
    values |= (GPIOA->IDR & (0x0001 << 4)) >> 4; //Read from pin PA4,
shift 0(UP)
    values |= (GPIOB->IDR & (0x0001 << 0)) << 1; //Read from pin PB0,
shift 1(DOWN)
    values |= (GPIOC->IDR & (0x0001 << 1)) << 1; //Read from pin PC1,
shift 2(LEFT)
    values |= (GPIOC->IDR & (0x0001 << 0)) << 3; //Read from pin PC0,
shift 3(RIGHT)
    values |= (GPIOB->IDR & (0x0001 << 5)) >> 1; //Read from pin PB5,
shift 4(CENTER)

    return values;
}
```

Det giver følgende output i PuTTY når joysticket bevæges i de forskellige retninger:



4.2

LED'ens pins initialiseres:

```
void initLED(){
    //PIN PA9 -> BLUE
    GPIOA->OSPEEDR &= ~(0x00000003 << (9 * 2)); // Clear speed register
    GPIOA->OSPEEDR |= (0x00000002 << (9 * 2)); // set speed
register(0x02 - 2 MHz)
    GPIOA->OTYPER &= ~(0x0001 << (9)); // Clear output type register
    GPIOA->OTYPER |= (0x0000 << (9)); // Set output type register(0x00 -
Push pull)
    GPIOA->MODER &= ~(0x00000003 << (9 * 2)); // Clear mode register
    GPIOA->MODER |= (0x00000001 << (9 * 2)); // Set mode register(0x01 -
Out)

    //PIN PB4 -> RED
    GPIOB->OSPEEDR &= ~(0x00000003 << (4 * 2)); // Clear speed register
    GPIOB->OSPEEDR |= (0x00000002 << (4 * 2)); // set speed
register(0x02 - 2 MHz)
    GPIOB->OTYPER &= ~(0x0001 << (4)); // Clear output type register
    GPIOB->OTYPER |= (0x0000 << (4)); // Set output type register(0x00 -
Push pull)
    GPIOB->MODER &= ~(0x00000003 << (4 * 2)); // Clear mode register
    GPIOB->MODER |= (0x00000001 << (4 * 2)); // Set mode register(0x01 -
Out)
```

```
    //PIN PC7 -> GREEN
    GPIOC->OSPEEDR &= ~(0x00000003 << (7 * 2)); // Clear speed register
    GPIOC->OSPEEDR |= (0x00000002 << (7 * 2)); // set speed
register(0x02 - 2 MHz)
    GPIOC->OTYPER &= ~(0x0001 << (7)); // Clear output type register
    GPIOC->OTYPER |= (0x0000 << (7)); // Set output type register(0x00 -
Push pull)
    GPIOC->MODER &= ~(0x00000003 << (7 * 2)); // Clear mode register
    GPIOC->MODER |= (0x00000001 << (7 * 2)); // Set mode register(0x01 -
Out)
}
```

Farven kan sættes med følgende funktioner (clear først og derefter set):

```
void clearLED(){
    GPIOB->ODR &= ~(0x0001 << 4); // Clear register
    GPIOC->ODR &= ~(0x0001 << 7); // Clear register
    GPIOA->ODR &= ~(0x0001 << 9); // Clear register
}


void setLED(uint8_t x1, uint8_t x2, uint8_t x3){
    clearLED();
    //set registers
    if(!x1){GPIOB->ODR |= 0x0001 << 4;} //pin PB4 (RED)
    if(!x2){GPIOC->ODR |= 0x0001 << 7;} //pin PC7 (GREEN)
    if(!x3){GPIOA->ODR |= 0x0001 << 9;} //pin PA9 (BLUE)
}
```

'0' betyder tændt og '1' betyder slukket, siden LED'erne er common anode.

main:

```
int main(void)
{
    initLED();
    initJoystick();

    rgbJoystickTest();
}
```

```
void rgbJoystickTest(){
    uint8_t jsValold;
    uint8_t jsValnew;

    uart_init(9600);
    clrscrn();

    jsValold = readJoystick();

    while(1){
       jsValnew = readJoystick();
       if(jsValold != jsValnew){
             printf("%d%d%d%d%d%d%d%d\n", (jsValnew & 64) >> 7,
(jsValnew     & 32) >> 6, (jsValnew & 32) >> 5, (jsValnew & 16) >> 4,
(jsValnew & 8) >> 3, (jsValnew & 4) >> 2, (jsValnew & 2) >> 1, (jsValnew
& 1) >> 0);
             RGBJoystick(jsValnew);
             jsValold = jsValnew;
       }
    }
}
```

Inde i main initialiseres alle pins først og derefter printes joystickværdien i binær og sætter LED'ens farver ved brug af nedenstående funktion:

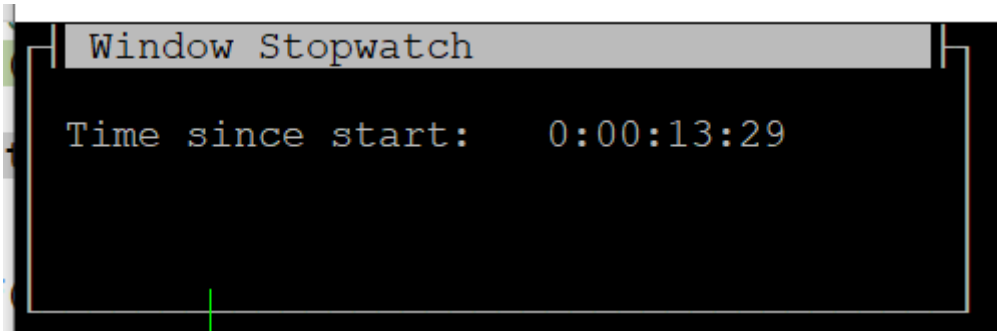
```
void RGBJoystick(uint8_t val){
    switch(val){
       case(1):
             setLED(0,1,0); //UP = GREEN
             break;

       case(2):
             setLED(1,0,0); //DOWN = RED
             break;

       case(4):
             setLED(0,1,1); //LEFT = CYAN
             break;

       case(8):
             setLED(1,0,1); //RIGHT = PURPLE
```

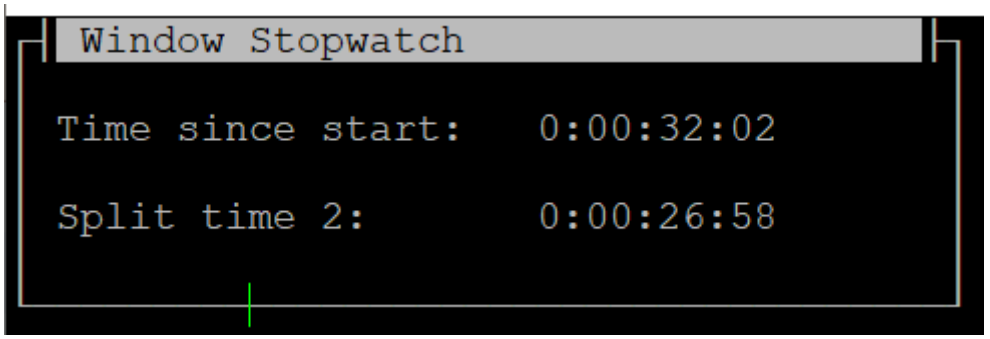
```
                break;

        case(16):
                setLED(1,1,0); //CENTER = YELLOW
                break;

        default:
                setLED(0,0,0);
    }
}
```

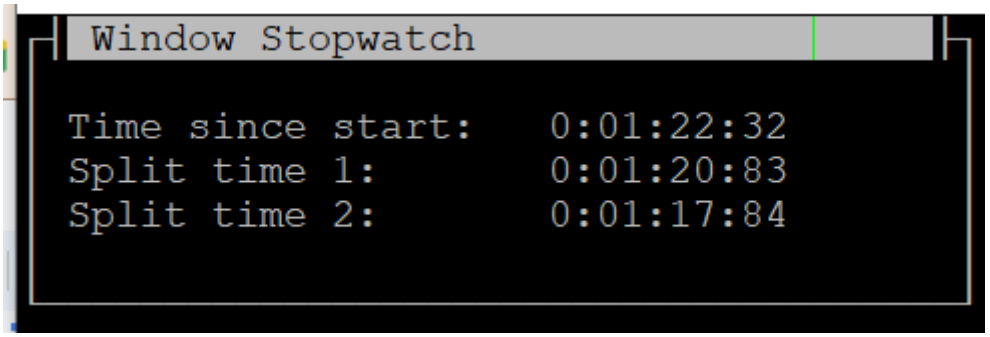
Opgave 5

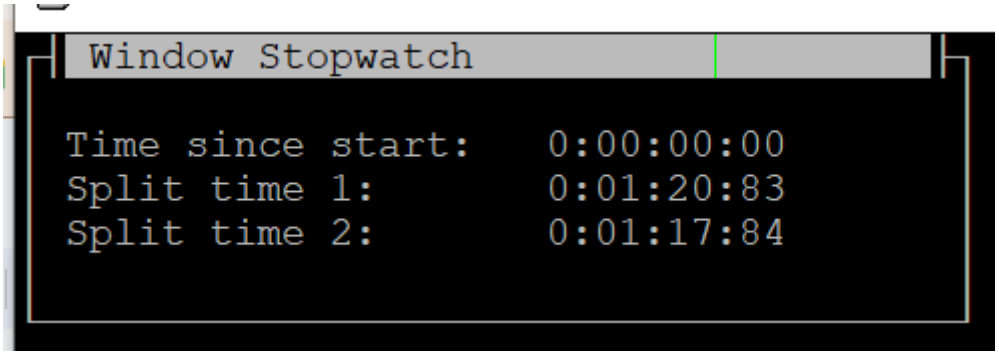
5.1

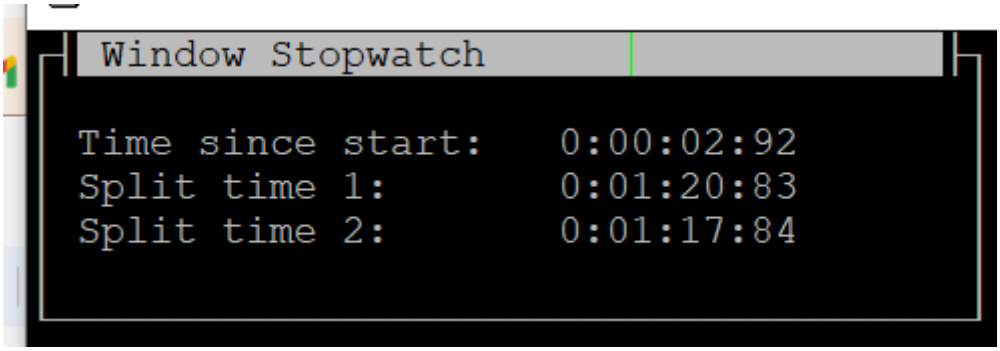
We use a high baud rate (115200) so we can display the hundredths of seconds:











stopwatch.h:

```
#ifndef STOPWATCH_H_
#define STOPWATCH_H_

#include "stdint.h"
#include "stdio.h"
#include "stm32f30x_conf.h"
#include "30010_io.h"

typedef struct{
    uint8_t hours;
    uint8_t minutes;
    uint8_t seconds;
    uint8_t hundredsthsOfSeconds;
```

```
} stopwatch_t;

extern volatile stopwatch_t stopwatch;

void initTimerStuff();
void updateTimer();
void toggleTimer();
void stopTimer();
void resetStopwatch();
void TIM1_BRK_TIM15_IRQHandler(void);


#endif /* STOPWATCH_H_ */
```

stopwatch.c:

```
#include "stopwatch.h"

volatile stopwatch_t stopwatch;

void initTimerStuff()
{
    RCC->APB2ENR |= RCC_APB2Periph_TIM15; // Enable clock line to timer
15;
    TIM15->CR1 = 0x0000; // Configure timer 15 and disable counter
    TIM15->ARR = 624; // Set reload value
    TIM15->PSC = 1023; // Set prescale value
    //assuming a systemclock of 64mhz this reload value and prescale
will result in a 1/100 seconds upcounting-mode timeout period
    TIM15->CR1 = 0x0001; // Configure timer 15 and enable counter

    //Interrupt part:
    TIM15->DIER |= 0x0001; // Enable timer 15 interrupts
    NVIC_SetPriority(TIM1_BRK_TIM15_IRQn, 1); // Set interrupt priority
    NVIC_EnableIRQ(TIM1_BRK_TIM15_IRQn); // Enable interrupt
}

void updateTimer()
{
    if(stopwatch.hundredsthsOfSeconds == 99)
    {
       stopwatch.hundredsthsOfSeconds = 0;
```

```
        if(stopwatch.seconds == 59)
        {
              stopwatch.seconds = 0;
              if(stopwatch.minutes == 59)
              {
                    stopwatch.minutes = 0;
                    stopwatch.hours++;
              }
              else
              {
                    stopwatch.minutes++;
              }
        }
        else
        {
              stopwatch.seconds++;
        }
    }
    else
    {
        stopwatch.hundredsthsOfSeconds++;
    }
}

void toggleTimer()
{
    TIM15->CR1     ^= 1;
}

void stopTimer()
{
    TIM15->CR1     &= ~1;
}

void resetStopwatch()
{
    stopwatch.hours = 0;
    stopwatch.minutes = 0;
    stopwatch.seconds = 0;
    stopwatch.hundredsthsOfSeconds = 0;
}

void TIM1_BRK_TIM15_IRQHandler(void)
{
    //do something here
```

```
    updateTimer();

    TIM15->SR &= ~(0x0001); //Clear the interupt bit}
```

main.c til stopwatch (i vores kode står det med ordentlig indentation, men vores formatter til docs laver forkert indentation nogle gange):

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course
#include "ansi.h"
#include "sinLut.h"
#include "vec.h"
#include "ball.h"
#include "stopwatch.h"
#include "joystick.h"
void runStopwatch();

int main(void)
{
    // Setup communication with the PC
    uart_init(115200);

    runStopwatch();
}
void runStopwatch()
{
    initJoystick();
    uint8_t preJoystick = readJoystick();

    resetStopwatch();

    clrscr();
    window(1,1,40,7,"Stopwatch",2);

printTime(stopwatch.hours,stopwatch.minutes,stopwatch.seconds,stopwatch.
hundredsthsOfSeconds,3,3,"Time since start");

    initTimerStuff();

    while(1)
    {
       //printf("%d%d%d%d%d%d%d%d\n", (preJoystick & 64) >> 7,
(preJoystick & 32) >> 6, (preJoystick & 32) >> 5, (preJoystick & 16) >>
4, (preJoystick & 8) >> 3, (preJoystick & 4) >> 2, (preJoystick & 2) >>
```

```
1, (preJoystick & 1) >> 0);
        window(1,1,40,7,"Stopwatch",2);

printTime(stopwatch.hours,stopwatch.minutes,stopwatch.seconds,stopwatch.
hundredsthsOfSeconds,3,3,"Time since start");

        if(readJoystick() != preJoystick)
        {
              preJoystick = readJoystick();
              switch(readJoystick())
              {
                    case 2: //Down
                          stopTimer();
                          resetStopwatch();

printTime(stopwatch.hours,stopwatch.minutes,stopwatch.seconds,stopwatch.
hundredsthsOfSeconds,3,3,"Time since start");
                          break;
                    case 4://Left

printTime(stopwatch.hours,stopwatch.minutes,stopwatch.seconds,stopwatch.
hundredsthsOfSeconds,3,4,"Split time 1");
                          break;
                    case 8://Right

printTime(stopwatch.hours,stopwatch.minutes,stopwatch.seconds,stopwatch.
hundredsthsOfSeconds,3,5,"Split time 2");
                          break;
                    case 16: //Center
                          toggleTimer();
                          break;
                    default:
                          break;
              }
        }
    }
}
```

5.2

```
int main(void) {
    char buff[256] = {'\0'};
```

```
    uart_init(9600);
    clrscr();
    initStopwatch();
    goToXY(1, 14);
    userGuide();
    uart_clear();

    while (1) {
       runStopwatch();
       takeInputs(&buff[0]);
    }
}
```

Da vi havde problemer med de indbyggede uart funktioner, skrev vi vores egne baseret på uart_get_char().

```
 void takeInputs(char *ptr){
    char temp = uart_get_char();

    inputHandler(ptr, temp);
    if(temp == 0x0D){
       switch(commandCheck(ptr)){
             case(1):
                   stopTimer();
                   toggleTimer();
                   break;
             case(2):
                   stopTimer();

printTime(stopwatch.hours,stopwatch.minutes,stopwatch.seconds,stopwatch.
hundredsthsOfSeconds,3,3,"Time since start");
                   break;
             case(3):

printTime(stopwatch.hours,stopwatch.minutes,stopwatch.seconds,stopwatch.
hundredsthsOfSeconds,3,4,"Split time 1");
                   break;
             case(4):

printTime(stopwatch.hours,stopwatch.minutes,stopwatch.seconds,stopwatch.
hundredsthsOfSeconds,3,5,"Split time 2");
                   break;
```

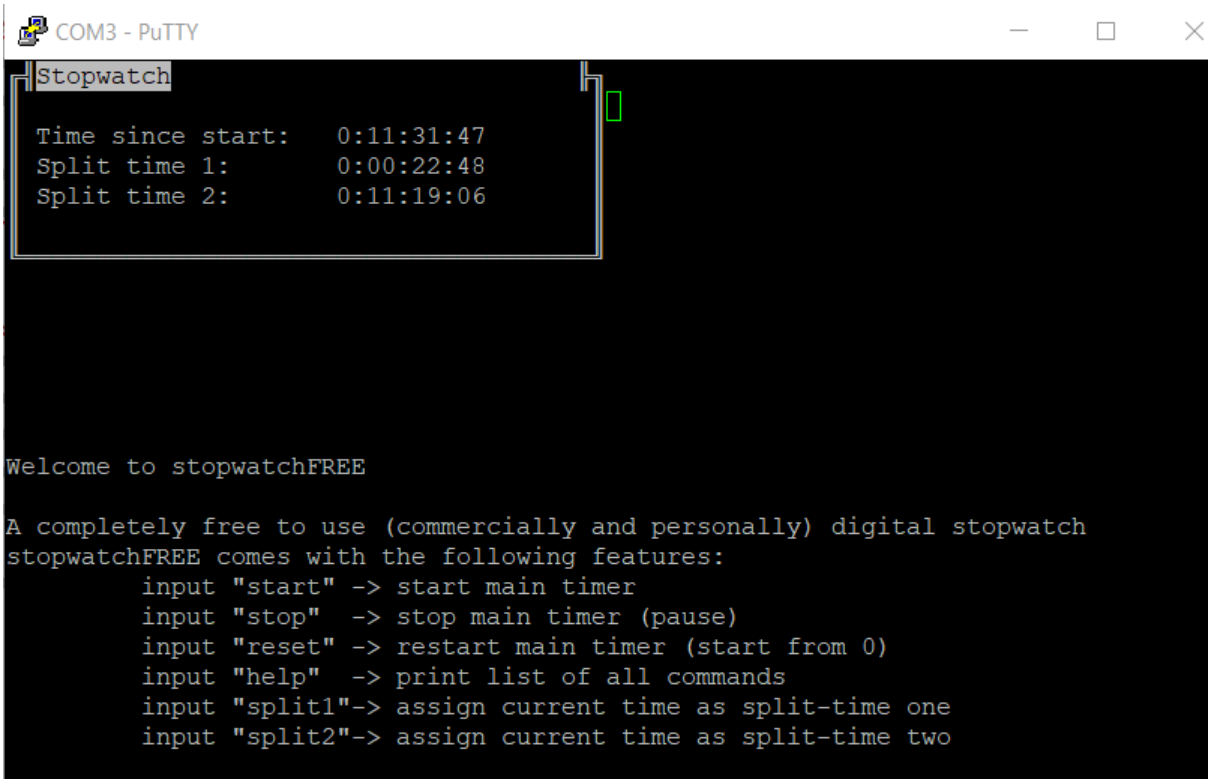
```
            case(5):
                  resetStopwatch();
                  break;
            case(6):
                  goToXY(1, 14);
                  userGuide();
                  break;
            default:
                  goToXY(1, 18);
                  printFunc();
      }
   }
}
```

Inden under en ny serialRead.c fil findes følgende funktioner, der sammenligner inputs med prædefinerede funktioner.

```
void inputHandler(char *ptr, char temp){
    static uint8_t realUartCount;

    if(realUartCount == 256){
       realUartCount = 0;
    }

    if(temp == 0x0D){
       *(ptr+realUartCount) = '\0';
       realUartCount = 0;

    } else if (temp != 0x00){
       *(ptr+realUartCount) = temp;
       realUartCount++;
    }
}

uint8_t commandCheck(char *ptr){
    char com0[6] = {'s', 't', 'a', 'r', 't', '\0'};
    char com1[5] = {'s', 't', 'o', 'p', '\0'};
    char com2[7] = {'s', 'p', 'l', 'i', 't', '1', '\0'};
    char com3[7] = {'s', 'p', 'l', 'i', 't', '2', '\0'};
    char com4[6] = {'r', 'e', 's', 'e', 't', '\0'};
    char com5[5] = {'h', 'e', 'l', 'p', '\0'};
    char check[256] = {'\0'};
```

```
    uint8_t i = 0;

    while(*(ptr+i) != '\0'){
       check[i] = *(ptr+i);
       i++;
    }

    if(!strcmp(com0, check)){ return 1; }
    if(!strcmp(com1, check)){ return 2; }
    if(!strcmp(com2, check)){ return 3; }
    if(!strcmp(com3, check)){ return 4; }
    if(!strcmp(com4, check)){ return 5; }
    if(!strcmp(com5, check)){ return 6; }

    return 0;
}

void userGuide(){
    printf("Welcome to stopwatchFREE\n\n");
    printf("A completely free to use (commercially and personally)
digital stopwatch\n");
    printf("stopwatchFREE comes with the following features:\n");
    printFunc();
}

void printFunc(){
    printf("\t input \"start\" -> start main timer\n");
    printf("\t input \"stop\"  -> stop main timer (pause)\n");
    printf("\t input \"reset\" -> restart main timer (start from 0)\n");
    printf("\t input \"help\"  -> print list of all commands\n");
    printf("\t input \"split1\"-> assign current time as split-time
one\n");
    printf("\t input \"split2\"-> assign current time as split-time
two\n");
}
```

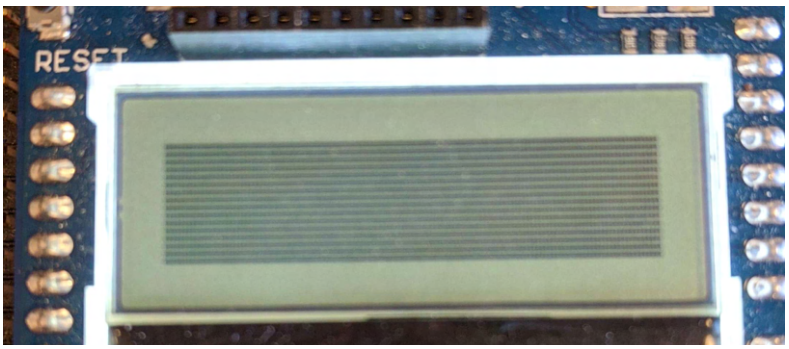
Det giver outputtet i PuTTY:



Opgave 6

6.1

```
int main(void)
{
    // Setup communication with the PC
    uart_init(115200);
    LCDStuff();
}

void LCDStuff()
{
    initLCD();
    uint8_t LCDbuffer[512];
    memset(LCDbuffer,0xAA,512);
    lcd_push_buffer(LCDbuffer);

    while(1){}
}
```



6.2

lcd_update funktion specifikt for 6.2med de concatenatede nuller:

```
void lcd_update(char *s)
{
    if(LCD_Flag == 1)
    {
       strcat(s,"0");
       LCD_Flag = 0;
    }
}
```

main.c kode specifikt til 6.2 med de concatenatede nuller:

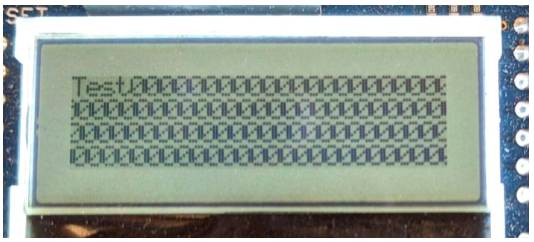
```
int main(void)
{
    // Setup communication with the PC
    uart_init(115200);

    initTimerStuff();
    LCDStuff();
}

void LCDStuff()
{
    initLCD();
    uint8_t LCDbuffer[512];
    memset(LCDbuffer,0xAA,512);
    lcd_push_buffer(LCDbuffer);
    char testString[200] = "Test";


    while(1)
```

```
    {
        lcd_write_string(testString,0,0,LCDbuffer);
        lcd_push_buffer(LCDbuffer);
        lcd_update(testString);
    }
}
```

Billeder af LCD display til 6.2:

(Taget imens nullerne kommer frem)

(Taget efter alle nullerne er blevet skrevet)



6.3

Vi opdaterede først lcd_write_string til at gemme resten af strengen i bytes hvis der ikke var plads til den (igen gør vores doc kode formatter at de lange kodelinjer går ned på næste linje):

```
void lcd_write_string(char s[], uint8_t slice, uint8_t line,uint8_t
*LCDbuffer_p, uint8_t *remainBytes_p)
{
    //Draws the string at the start position (only the part that can fit
the rest of the line)
    for(int i = 0; i < strlen(s); i++)
```

```
    {
       for(int j = 0; j < 5; j++)
       {
             if(slice+i*5+j <= 127)
             {
                   *(LCDbuffer_p+slice+i*5+j+line*128) =
character_data[s[i]-0x20][j];
             }
             else
             {
                   //Saves the rest of the string that couldnt fit the
lcd in remainBytes
                   for(int k = i; k < strlen(s); k++)
                   {
                         for(int l = 0; l < 5;l++)
                         {
                               if(k == i)
                               {
                                     *(remainBytes_p + l + line*128) =
character_data[s[k]-0x20][l+j];
                                     if(l+j == 4){break;}
                               }
                               else
                               {
                                     *(remainBytes_p + (k-1 - i)*5 + l +
j + line*128) = character_data[s[k]-0x20][l];
                               }
                         }
                   }
                   return;
             }

       }
    }
}
```

Så skrev vi lcd_leftscrolling_text som egentlig også kan bruges til at shifte ikke kun tekst. Vi har også tænkt os at udvide så den kan shifte til højre til vores spil.

```
void lcd_leftscrolling_text(uint8_t *LCDbuffer_p, uint8_t line, uint8_t
*remainBytes_p)
{
```

```
    /*
    Shifts LCDbuffer to the left by 1
    * New bytes shifted in from the right should be the remaining part
of the string (if any).
    * If no left of remaining part use the parts of the string that were
pushed out the left side.
    * If nothing has been pushed out yet print blank spaces.
    */

    //saves the byte ,in lcdbuffer that is about to be shifted out, in
the first 'empty' (=! 0xAA) spot in remainbytes
    for(int i = line*128; i < line*128+128; i++)
    {
       if(*(remainBytes_p+i) == 0xAA)
       {
             *(remainBytes_p+i) = *(LCDbuffer_p+line*128);
             break;
       }
    }

    //shift everything except the last element one to the left
    for(int i = 0; i < 127; i++)
    {
       *(LCDbuffer_p+i+line*128) = *(LCDbuffer_p+i+line*128+1);
    }

    //The last element shifted in is the first element in remainbytes
(on that line)
    *(LCDbuffer_p+127+line*128) = *(remainBytes_p+line*128);

    //Shifts everything on the specific line in remainbytes one to the
left except the last where 0xAA (empty) is inserted
    for(int i = line*128; i < line*128+127; i++)
    {
       *(remainBytes_p+i) = *(remainBytes_p+i+1);
    }
    *(remainBytes_p+127+line*128) = 0xAA;

}
```

Vi opdaterede også lcd_update til bare at returnere 1 når lcd flaget er højt:

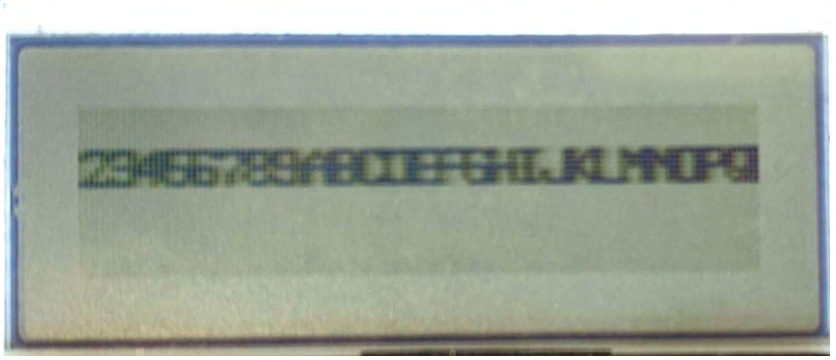
```
int lcd_update()
{
    if(LCD_Flag == 1)
    {
       LCD_Flag = 0;
       return 1;
    }

    return 0;
}
```

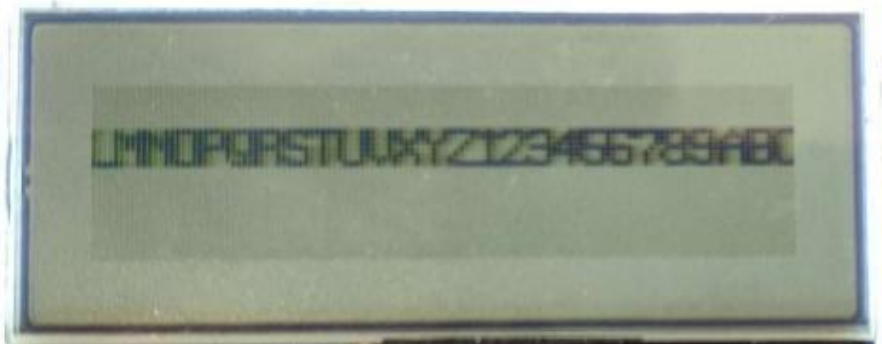
I main kørte vi det hele på denne måde:

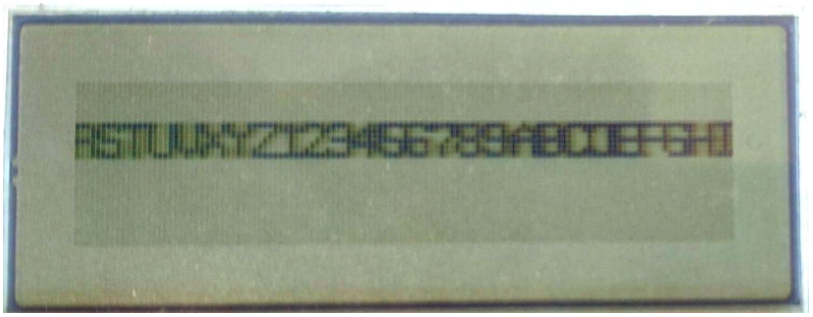
```
int main(void)
{
    // Setup communication with the PC
    uart_init(115200);

    initLCDMain();
}

void initLCDMain()
{
    uint8_t LCDbuffer[512];
    uint8_t remainBytes[512];
    char s[] = "ABCDEFGHIJKLMNOPQRSTUVXYZ123456789";
    uint8_t tickCounter = 0;

    initTimerStuff(); //Comment to debug
    initLCD();

    memset(LCDbuffer,0x00,512);
    memset(remainBytes,0xAA,512);
    lcd_write_string(s,1,1,LCDbuffer,remainBytes);
    lcd_push_buffer(LCDbuffer);

    while(1)
    {
       if(lcd_update())
       {
             tickCounter++;
       }

       if(tickCounter == 7)
```

```
        {
                lcd_leftscrolling_text(LCDbuffer, 1, remainBytes);
                lcd_push_buffer(LCDbuffer);
                tickCounter = 0;
        }
    }
}
```

Billeder af LCD der scroller



: 



Opgave 7

7.1

Kode til at initialisere pins og ADC:

```
void initPots(){
    RCC->AHBENR |= RCC_AHBPeriph_GPIOA; // Enable clock for GPIO Port A

    // Set pin PA0 to input
    GPIOA->MODER |=  (0x00000000 << (0 * 2)); // Set mode register (0x00
- Input, 0x01 - Output, 0x02 - Alternate Function, 0x03 - Analog in/out)
    GPIOA->PUPDR &= ~(0x00000003 << (0 * 2)); // Clear push/pull
register
    GPIOA->PUPDR |=  (0x00000002 << (0 * 2)); // Set push/pull register
(0x00 - No pull, 0x01 - Pull-up, 0x02 - Pull-down)
    //uint16_t val = GPIOA->IDR & (0x0001 << pin); //Read from pin PA0

    // Set pin PA1 to input
```

```
    GPIOA->MODER |=  (0x00000000 << (1 * 2)); // Set mode register (0x00
- Input, 0x01 - Output, 0x02 - Alternate Function, 0x03 - Analog in/out)
    GPIOA->PUPDR &= ~(0x00000003 << (1 * 2)); // Clear push/pull
register
    GPIOA->PUPDR |=  (0x00000002 << (1 * 2)); // Set push/pull register
(0x00 - No pull, 0x01 - Pull-up, 0x02 - Pull-down)

    //Pots
    RCC->CFGR2 &= ~RCC_CFGR2_ADCPRE12; // Clear ADC12 prescaler bits
    RCC->CFGR2 |= RCC_CFGR2_ADCPRE12_DIV6; // Set ADC12 prescaler to 6
    RCC->AHBENR |= RCC_AHBPeriph_ADC12; // Enable clock for ADC12

    ADC1->CR = 0x00000000; // Clear CR register
    ADC1->CFGR &= 0xFDFFC007; // Clear ADC1 config register
    ADC1->SQR1 &= ~ADC_SQR1_L; // Clear regular sequence register 1

    ADC1->CR |= 0x10000000; // Enable internal ADC voltage regulator
    for (int i = 0 ; i < 1000 ; i++) {} // Wait for about 16
microseconds

    ADC1->CR |= 0x80000000; // Start ADC1 calibration
    while (!(ADC1->CR & 0x80000000)); // Wait for calibration to finish
    for (int i = 0 ; i < 100 ; i++) {} // Wait for a little while

    ADC1->CR |= 0x00000001; // Enable ADC1 (0x01 - Enable, 0x02 -
Disable)
    while (!(ADC1->ISR & 0x00000001)); // Wait until ready
}
```

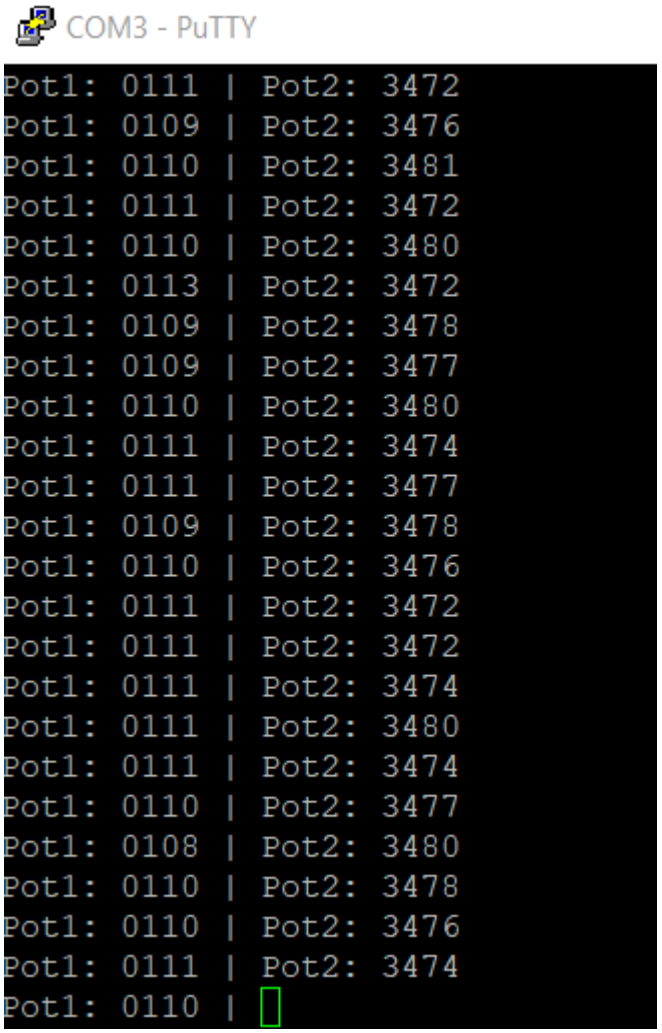
Kode til at læse værdier fra potmetre:

```
uint16_t readPot1(){
    ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1,
ADC_SampleTime_1Cycles5);
    ADC_StartConversion(ADC1); // Start ADC read
    while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == 0); // Wait for ADC
read
```

```
    return ADC_GetConversionValue(ADC1); // Read the ADC value
}

uint16_t readPot2(){
    ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 1,
ADC_SampleTime_1Cycles5);
    ADC_StartConversion(ADC1); // Start ADC read
    while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == 0); // Wait for ADC
read

    return ADC_GetConversionValue(ADC1); // Read the ADC value
}
```

7.2

Her printes potmetrenes værdier til PuTTY via printf():



Funktion der skriver en string med målinger til et givent array:

```
void potsToString(char array[]){
    sprintf(array, "Pot1: %04hd | Pot2: %04hd\n", readPot1(),
readPot2());
}
```

Opgave 8

8.1

Initialisering af timer 2 samt pin til PWM signal:

```
void initBuzz(){
    RCC->APB1ENR |= RCC_APB1Periph_TIM2; // Enable clock line to timer 2
```

```
    RCC->AHBENR  |= RCC_AHBPeriph_GPIOB; // Enable clock for GPIO Port B

    TIM2->CR1 = 0x0000; // Configure timer 2 and disable counter
    TIM2->PSC = 1023; // Set prescale value
    //assuming a systemclock of 64mhz this reload value and prescale
will result in a 1/100 seconds upcounting-mode timeout period
    TIM2->CR1 = 0x0001; // Configure timer 2 and enable counter

    TIM2->CCER  &= ~TIM_CCER_CC3P;   // Clear CCER register
    TIM2->CCER  |=  0x00000001 << 8; // Enable OC3 output
    TIM2->CCMR2 &= ~TIM_CCMR2_OC3M;  // Clear CCMR2 register
    TIM2->CCMR2 &= ~TIM_CCMR2_CC3S;
    TIM2->CCMR2 |=  TIM_OCMode_PWM1; // Set output mode to PWM1
    TIM2->CCMR2 &= ~TIM_CCMR2_OC3PE;
    TIM2->CCMR2 |=  TIM_OCPreload_Enable;

    //Set pin PB10 to alternate
    GPIOB->MODER &= ~(0x00000003 << (10 * 2)); // Clear mode register
    GPIOB->MODER |= (0x00000002 << (10 * 2));  // Set mode register
(0x00 - Input, 0x01 - Output, 0x02 - Alternate Function, 0x03 - Analog
in/out)
    GPIOB->PUPDR &= ~(0x00000003 << (10 * 2)); // Clear push/pull
register
    GPIOB->PUPDR |= (0x00000002 << (10 * 2));  // Set push/pull register
(0x00 - No pull, 0x01 - Pull-up, 0x02 - Pull-down)

    GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_1); //Map PWM to
PB10
}
```

Given kode til at skifte frekvens:

```
void setFreq(uint16_t freq) {
    uint32_t reload = 64e6 / freq / (2000 + 1) - 1;
    TIM2->ARR  =  reload;   // Set auto reload value
    TIM2->CCR3 =  reload/2; // Set compare register
    TIM2->EGR  |= 0x01;
}
```

Brugt sammen med potmetre til at ændre frekvensen alt efter potmetrets position:

```
int main(void) {
    char pots[25] = {'\0'};

    initBuzz();
```

```
    initPots();
    uart_init(9600);

    setFreq(100);
    clrscr();
    uart_clear();

    while (1) {
       setFreq(readPot1());
       potsToString(pots);
       printf("%s", pots);
    }
}
```

8.2

Memory bliver læst fra den sidste page (0x0800F800) plus offsettet på siden:

```
uint16_t readMemory(uint16_t offset){
    //Typecast to uint16_t pointer and read the value at this location
    //Add 0x0800F800 to get to the correct memory page
    return *(uint16_t *)(0x0800F800 + offset * 2);
}
```

Her skrives der til boardets memory:

```
void saveToMemory(uint16_t array[], uint8_t len){
    //Return if you are attempting to save more data than we have space
for
    if(len > 128){ return; }

    //Unlock memory
    FLASH_Unlock();
    FLASH_ClearFlag( FLASH_FLAG_EOP | FLASH_FLAG_PGERR |
FLASH_FLAG_WRPERR );

    //Erase page we want to write to
    FLASH_ErasePage(0x0800F800);

    //Write data to memory
    for(uint8_t i = 0; i < len; i++){
```

```
        FLASH_ProgramHalfWord(0x0800F800 + i*2, array[i]);
    }

    //Lock memory
    FLASH_Lock();
}
```