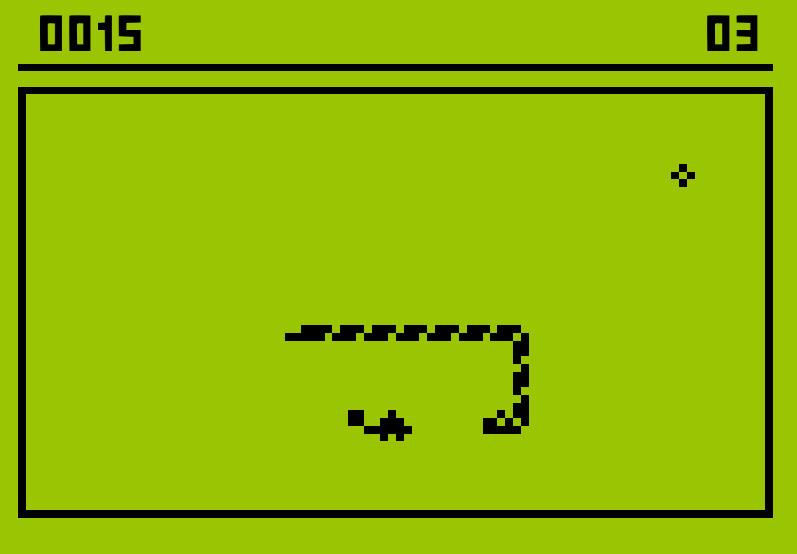
**2D LED GRID : SNAKE GAME**

CPROGP2 Assignment



Student: Jan Sevrin | 657716 Teacher: Ruud Elsinghorst

Datum: 02-02-2021

**Version log**

|  |  |  |
| --- | --- | --- |
| 02-02-2021 | V1 | Full |

**Summary**

For the C programming assignment of the second semester a student must choose his own assignment and further choose his own sub-assignment. In this project the student has chosen to build a 2D LED Grid Simulator alongside a fully functional Snake Game as an example output for that grid.

This project is programmed fully in the language C with the help of Windows libraries. The IDE used is Eclipse and further programs used include PlantUML to visualize the FSM and diagrams.net to create the layered architecture and the flow chart graphs. In this project the grid may only output 3 types of different visual output to the user: both LEDs off, red or green. To stimulate this a text document is created by the C program with either a dot to indicate LEDs off, an ‘R’ to indicated red led on and an ‘G’ to indicate a green led on.

Further there is fully functional sound output, the game is fully modulized and there are various settings to scale the game into bigger grids, which means that the game is not bound to only work on one set of dimensions for a 2D LED Grid. One more thing to note is that the game uses multithreading to achieve the most smooth flow of gameplay. The game also is able to save a top score locally and display it on the start menu.

Various tests have been carried out to ensure the game’s functionality and finally a user manual was made to provide instructions to users of the game.

# Introduction

This project is an assignment for C Programming Practicum of the Electrical Engineering bachelor study at the university of HAN, Arnhem. The purpose of this assignment is to let the student become more familiar with state machines.

This report explains how the assignment was built from scratch and which exact steps were used to get from nothing to a fully functional multi-feature multi-threaded game. The structure of the report is as follows:

* Chapter 2 describes the definition phase. The definition phase contains the functional and technical specifications that show how the program works and which products have been used.
* Chapter 3 describes the design phase. Here you can read how a technical solution for the snake game was created and diagrams are showing how the FSM is constructed.
* Chapter 4 contains information about the realization phase and testing. The realization phase shows how the program is structured and shows what the user environment is like. It is briefly described how the tests were carried out and where the tests can be found.
* Chapter 5 contains the final result and recommendations. The project is reviewed and a conclusion is described.
* At the end of the report is the appendix, the appendix contains the user manual of the snake game.

# Definition phase

The snake game consists of various functions and performs all kinds of different outputs depending on the input. Exactly what the snake game is defined to do is described here below:

## Functional specification

The 2D LED Grid must accord to all kinds of requirements, which were given to the student by the lecturer. These requirements are:

## 

## *Figure 0 Requirements*

## Technical specification

To create the 2D LED Grid Snake Game, various software is used. The full list of the software used is specified below:

1. Programming language: C
2. Development environment: Eclipse
3. UML diagram editor: PlantUML
4. Flow chart and arhictecture diagram software: diagrams.net and paint.net
5. Operating System: Windows 10
6. Runtime software: Windows CMD
7. Compiler: MinGW Compiler Settings: -O0 -g3 -Wall -c -fmessage-length=0

## User Interface(Start Menu)

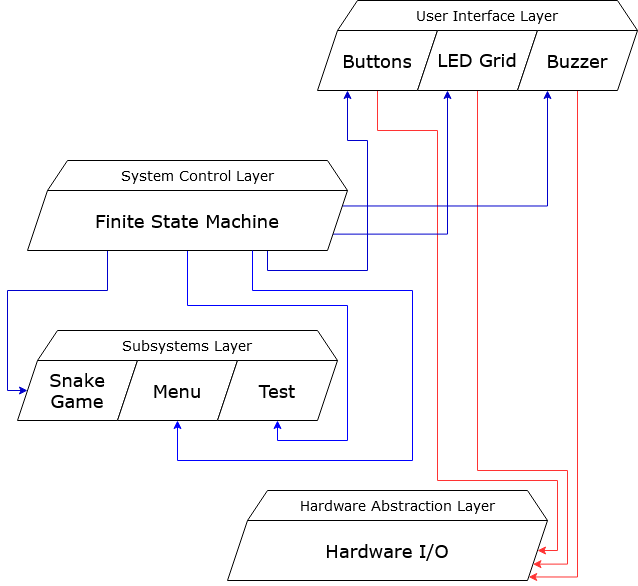
*Figure 1 User Interface(Start Menu). This figure shows the start menu of the snake game and the 2D LED Grid. The amount of the G’s displayed at the top indicate the top score ever achieved in the game.*

# Design phase

This chapter describes the technical design part of the whole project step by step.

## Layered Architecture

First, the layered architecture is created of the 2D LED Grid Snake Game. This architecture shows the header structure of the Finite State Machine of the project.



*Figure 2 Layered architecture of the 2D LED Grid Snake Game*

### User Interface Layer

In this layer, the various functions are defined that take input from the user and provide output back to him via the HAL.

### System Control Layer

The FSM (finite state machine) controls the whole program/machine, the states are executed here. The machine can be in one state at a time and moves to another state by means of an event. Main functions are also executed in the FMS. Some functions are executed concurrently(multithreading) to ensure smooth game flow.

### Subsystems Layer

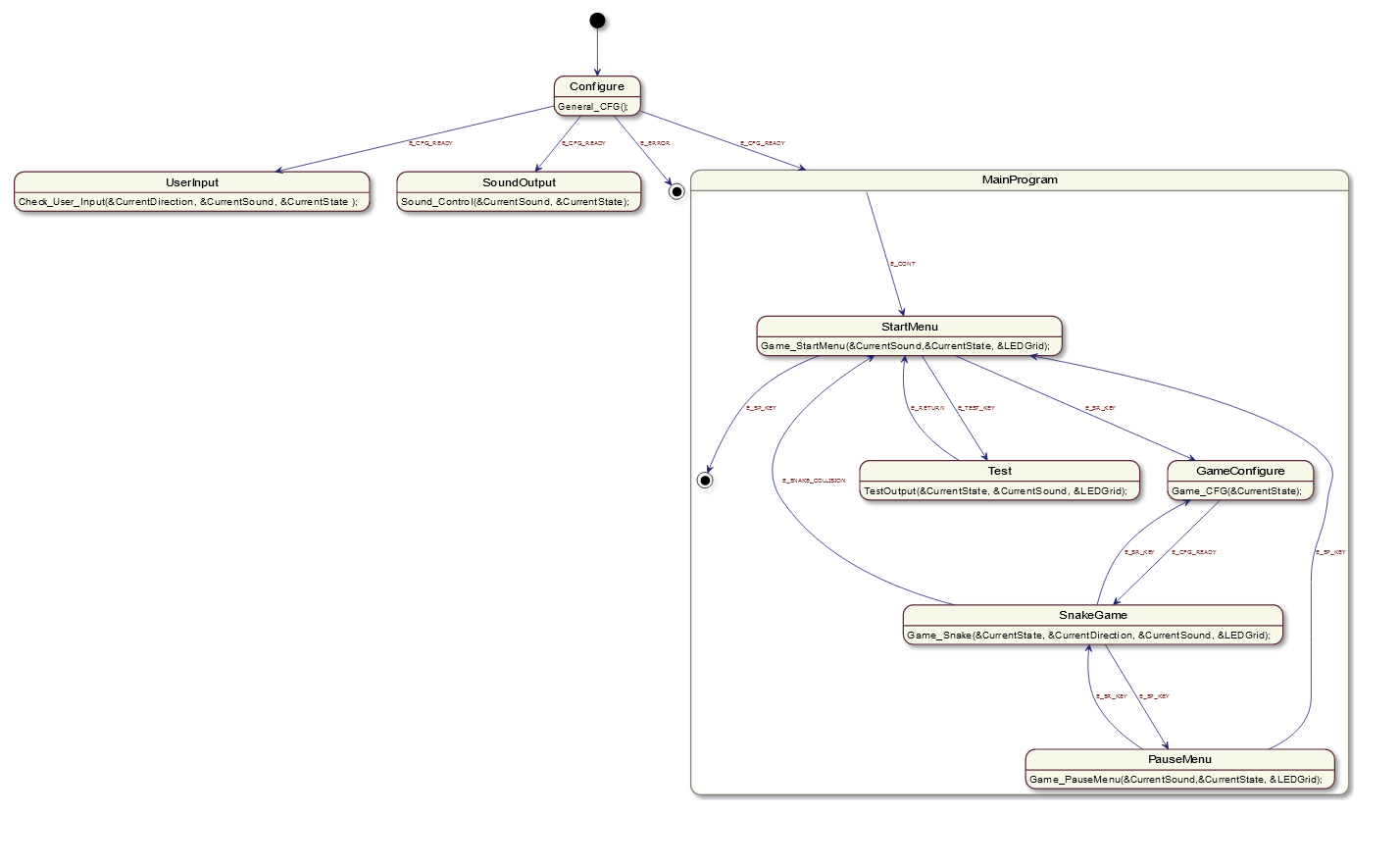
This layer contains the systems that are controlled from the FSM. This includes the menu, the testing part and the game itself.

### Hardware Abstraction Layer

Here the actual hardware is controlled. This is controlled from the subsystems, but can also be controlled directly via the User Interface Layer, for example for testing purposes.

## State machine

The design of the statemachine is engineered from a UML diagram. The UML state diagram of the 2D LED Grid Snake Game is shown in figure 3. In the UML state diagram you can read which states the FSM has and which events (for example a key on the keyboard or a collision of the snake, etc…) are required to enter the state.



*Figure 3 Finite State Machine of the 2D LED Grid Snake Game*

The first state the machine enters after start-up is the configure state. The configuration of the whole program happens here. If the configuration fails, then the program exits with the event E\_ERROR, otherwise it continues with E\_CFG\_READY.

After initialization, the machine creates three different states. The first one is called UserInput, which will be used to continously monitor for UserInput. The second one will be the SoundOutput, which will be used to provide smooth output of the sound to the user. Finally the last one is the 2D LED Grid itself, it either shows to the user the menu, the test menu or the game itself. Here most of the time the code is creating and updating output on the screen, which also gets saved into a textfile in real-time in the .exe’s folder.

UserInput state:

Here the function Check\_User\_Input receives three pointers which it is able to modify and read. This function contnously reads for user input and further performs different logic depending on which state the user is found in. So it does not only check and update user input, but it also able to change affect different parts of the program indirectly via the pointers provided to it.

SoundOutput state:

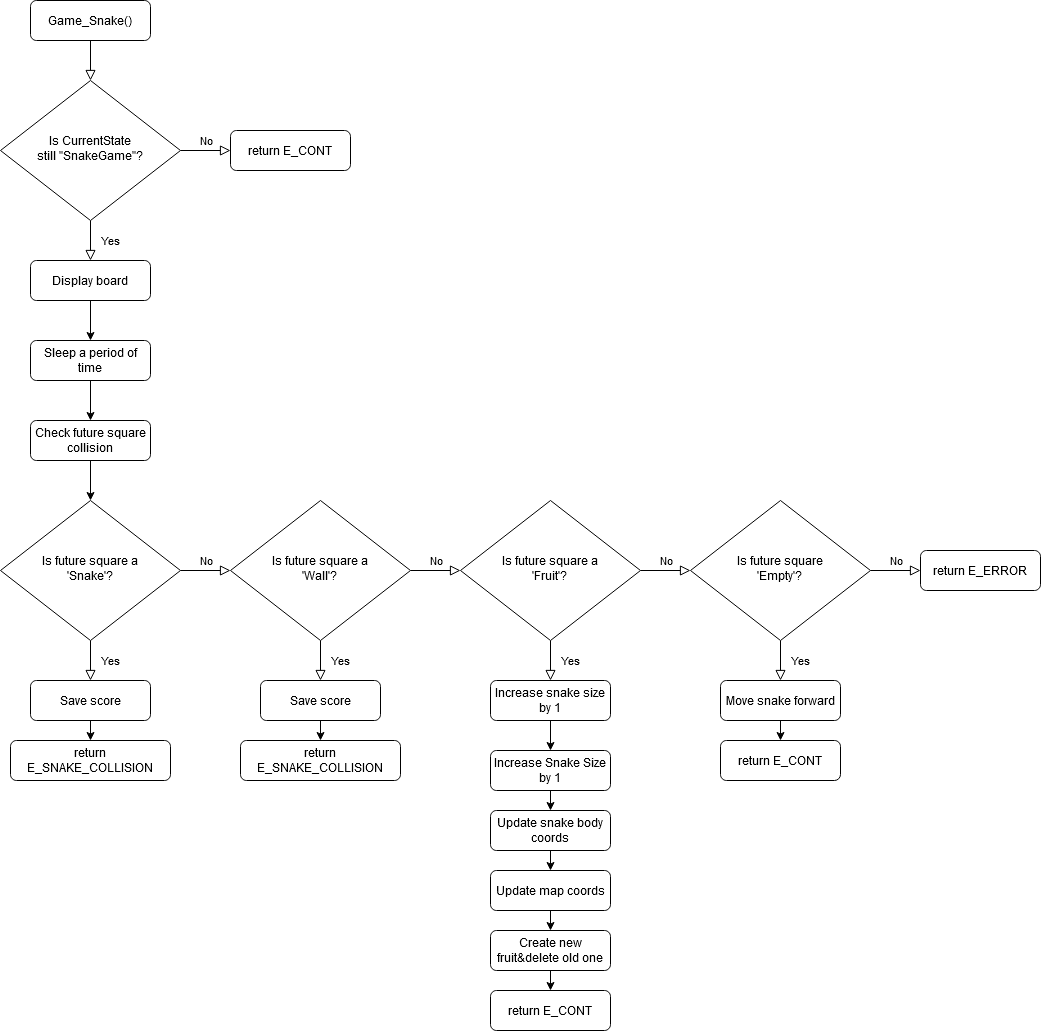
Here the function Sound\_Control receives three pointers which it is able to modify and read. This function continouslly check to see if the variable CurrentSound is updated or not to provide sound(or lack of). It also continously checks for the user state to make sure that sound does not flow from one state to another.

MainProgram FSM:

The main program is a whole finite state machine. As is visible, it has different states: StartMenu, Test, GameConfigure, SnakeGame and PauseMenu. It starts off with the StartMenu and you can see on the graph which events are required to go into which state. Using this graph, you can easily see the paths a user may take between different states. For example the test key allows a user to go from the StartMenu into the Test state and if the test succeeds it goes back through the event “E\_RETURN”. Further assuming the user starts the game, eventually his snake will collide with an object and he will be returned to the start menu through the event “E\_SNAKE\_COLLISION”. **In the C code all events are enumerated using enum in a global define file header.**

Snake Game:

Most of the project time was spent on developing this state, here is where the actual game logic happens. To start even developing this state, a flow chart was made to have a rough idea of what to program:



*Figure 4 Snake Game Flow Chart*

Roughly as is seen on the flow chart, the game continously prints out the game board to the user imiating a 2D LED Grid. Then it sleeps a certain modifiable period of time, after which it checks various logic against the future square of where the snake is headed towards. The future square is modifed by the UserInput state through the &CurrentDirection pointer. If the future square is a snake or a wall, then it counts as a collision and the user loses. His score is then saved and the state returns back to the start menu through the E\_SNAKE\_COLLISION event.

Otherwise if the future square is a fruit, then the snake is grown by 1 into the future square, the fruit is deleted, the coordinates are updated and the function further returns to the FSM with the event E\_CONT. If the square is only an empty square, then the snake is simply moved one square forward and the function returns to the FSM with E\_CONT.

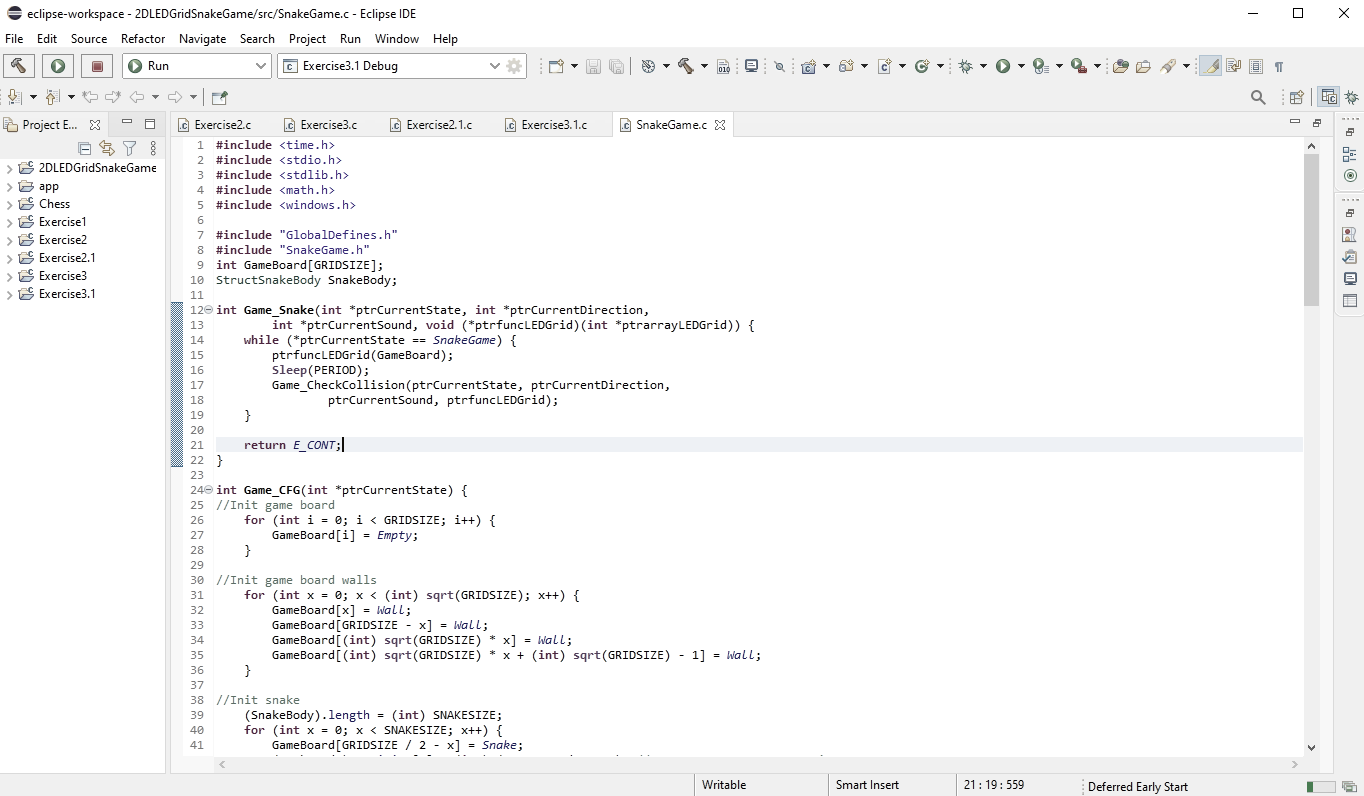
After the function returns to the FSM, assuming the user did not lose through E\_SNAKE\_COLLISION event, then the SnakeGame function is looped over again, seeing that the state is still set to SnakeGame.

# Realization and testing

The way in which the project is realized can be read in the chapters below. A test was carried out to meet all the correct customer requirements and to ensure no logic flaws in the game itself.

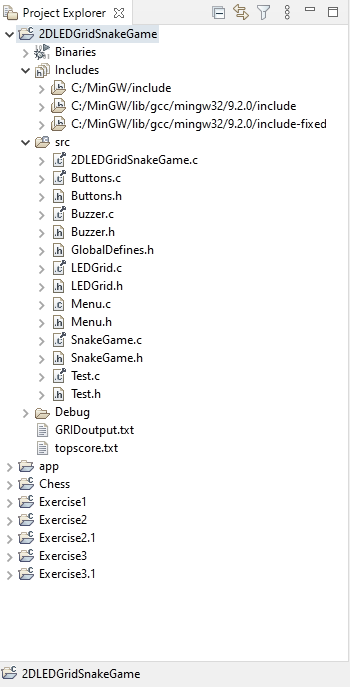
## Development environment

The project "2D LED Grid Snake Game" was developed with the program Eclipse, an IDE capable of helping a user program in various languages. Here is a figure showing what the IDE looks like:

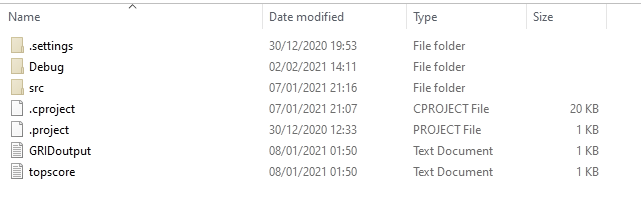


*Figure 5 Development environment Qt-Creator*

In the left part is the project folder with the header files and sources below it, and also the Eclipse-specific .project and .cproject files. Here is a figure showing all the files used in the project from the view of Eclipse and from the view of Windows Explorer:



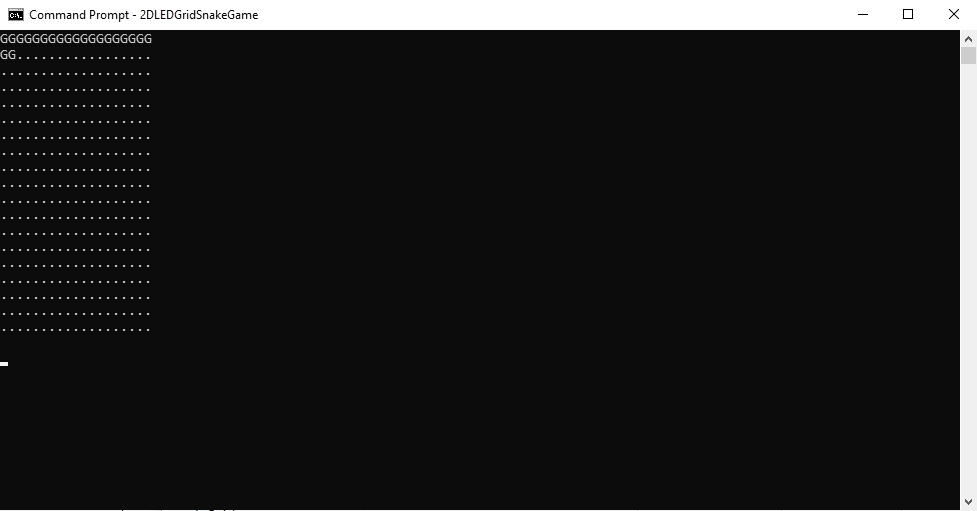
*Figure 6 Eclipse Folder View*

**

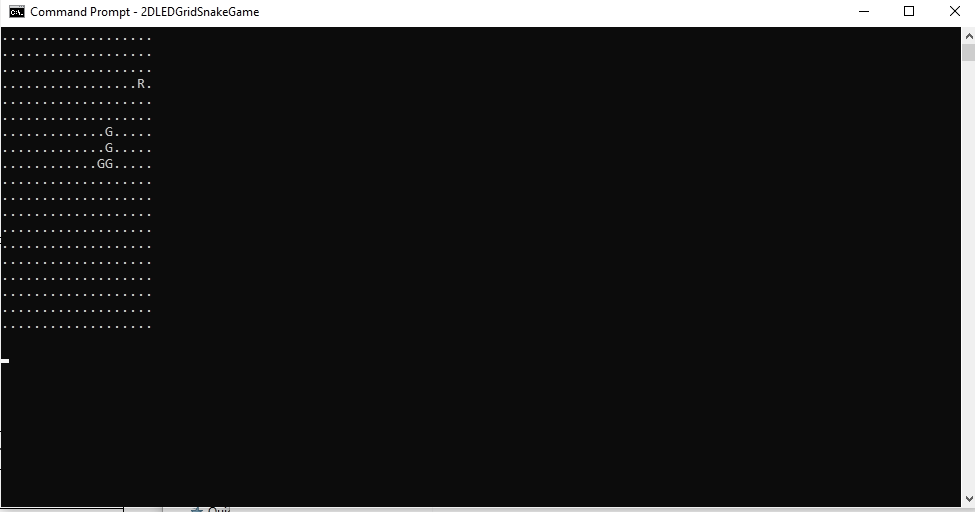
*Figure 7 Windows Explorer View*

## Detailed User Interface

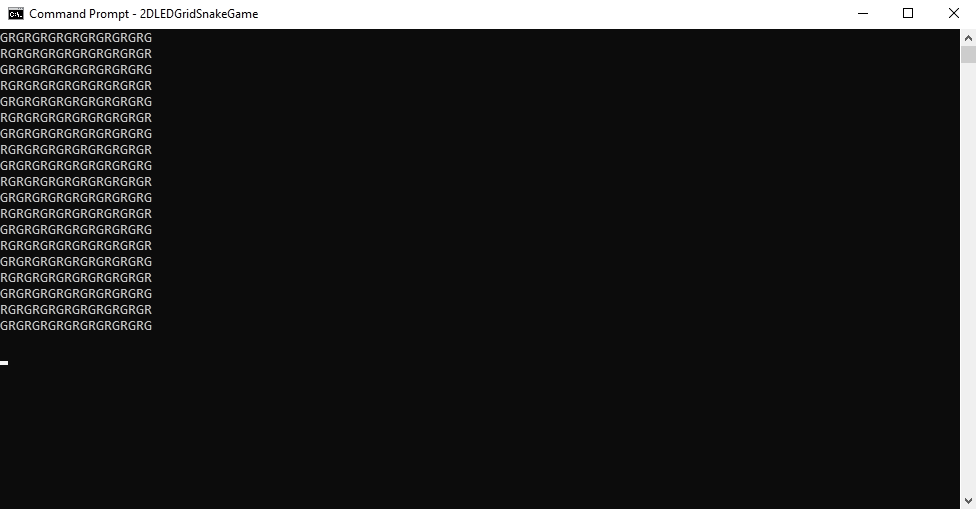
The following figures display various states of the user interface:



*Figure 8 Start Menu*



*Figure 9 Game + Game Pause Interface*



*Figure 10 Testing Interface*

## Acceptance test

In order to deliver the 2D LED Grid Snake Game to the customer properly, it is necessary to thoroughly test the FSM. Figure 11 shows a test table of the tests performed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Project name: 2D LED Grid Snake Game** | | | | **Date:** 11.01.2021 | | |
| **Test case ID: 001** | | **Name: Jan Sevrin** | |  | | |
| **Function/module/program under test:**  **2D Led Grid Snake Game** | | | |  | | |
| **Test** | **Action/input** | | **Expected result** |  | **Pass**  **Fail** | **Actual result if test has failed** |
| 1. | Application start properly | | Displayed main manu |  | Pass |  |
| 2. | Application accept user input (all variants) | | Application accepted all defined imputs |  | Pass |  |
| 3. | Game works properly part 1 | | Snake continously moves in response to movement keys |  | Pass |  |
| 4. | Game works properly part 2 | | Snake grows when eating a fruit, game over when collision happens. Also fruit properly respanws. |  |  |  |
| 4. | Snake cannot eat it’s last part of it’s body if that part shall move in the future | | Game does not finish when you try to eat yourself with a fixed length of 4 squares |  | Pass |  |
| 5. | Tests phase works properly | | Fully lit-up LED grid is shown with sound playing |  | Pass |  |
| 6. | Savescore and text filegeneration | | Game properly copied grid to text-file and a savescore is generated when a collision is detected |  | Pass |  |

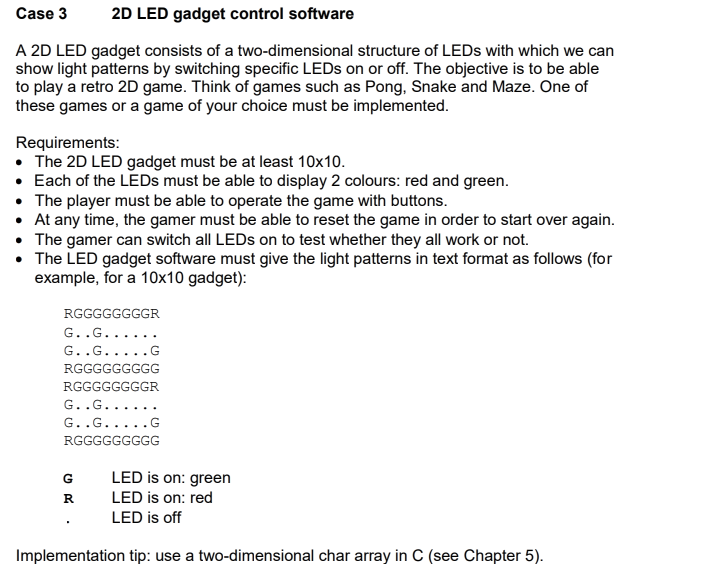
*Figure 11 Test table*

The table shows that the tests meet the requirements of the machine.

# End result and recommendations

The purpose of the application is to provide entertainment to the user through the game ‘Snake’ on a scalable 2D LED grid. All requirements have been met and no bugs have been found.

The following **must** requirements have been fully realized:



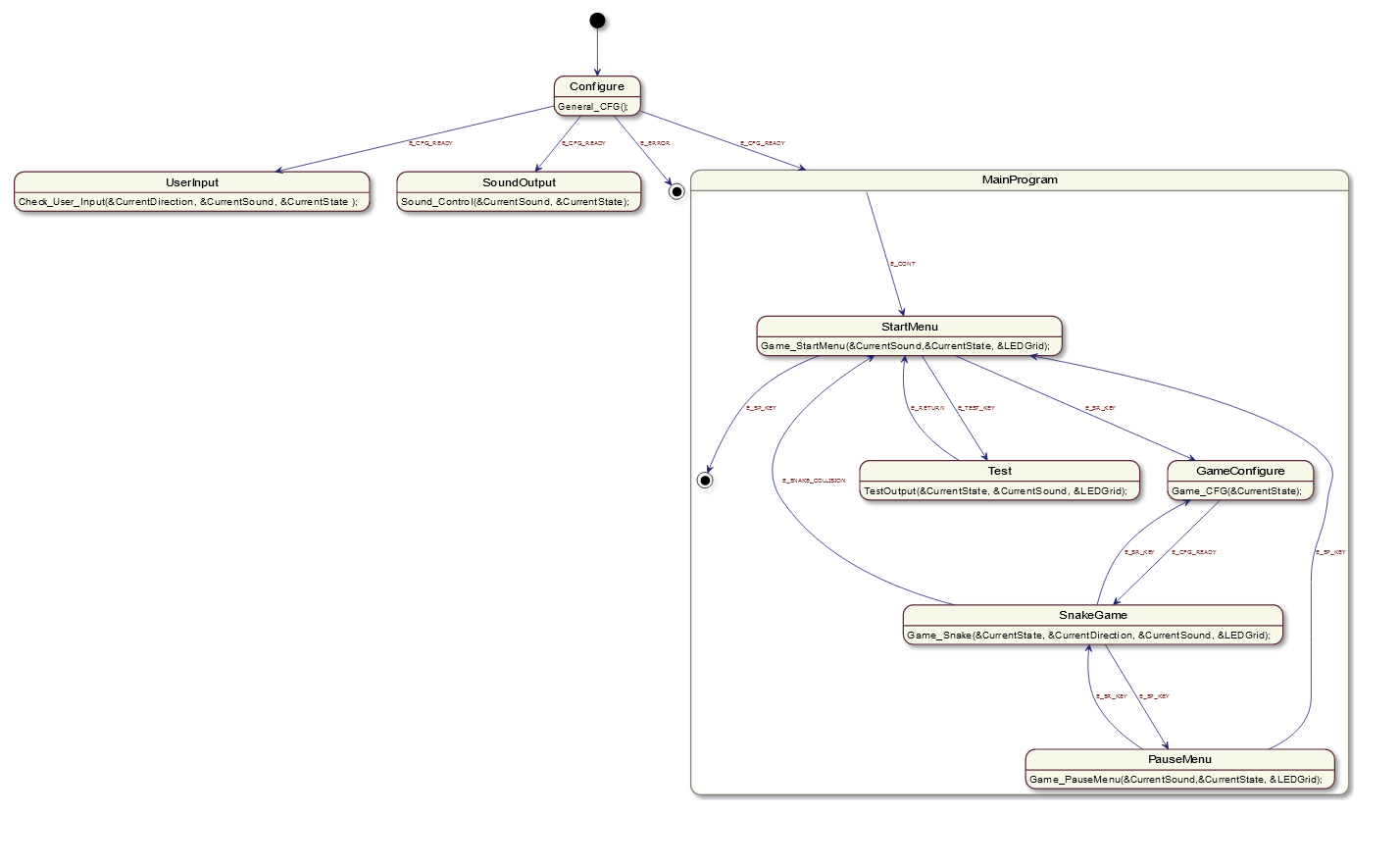
## *Figure 0 Requirements*

All tests have been passed and the project is fully functional.

For future development, it is recommended to work on further expanding the range of colors and the total resolution of the output GRID. Further, the sounds may be upgraded to a higher quality noise. Since all functions are modulized and use pointers as input, the functions may be freely modified without breaking the whole program unexpectedly. Perhaps a better library can be used for the sound output and instead of using an extra thread, the user input and sound output can be ran in one thread. Also it is highly recommended to not use the language C for programming a game, thus for the best results, translating this C game into a C++ game will provide the most benefits in the future.

# User Manual

To make the user manual easier, the following figure is used again:



*Figure 3 Finite State Machine of the 2D LED Grid Snake Game*

The following keybinds are used in the general FSM:

E\_TEST\_KEY : ‘T’

E\_SR\_KEY : ‘ENTER’

E\_SP\_KEY : ‘BACKSPACE’

To play the game, simply hit ‘ENTER’ in the start menu. The output will change to the game, where a snake and a fruit will be spawned. The snake will be in the middle and the fruit will be spawned somewhere randomly. The objective of the game is to grow your snake as far as possible without colliding with yourself or the walls.

The following keybinds are used for the snake game:

UP: ‘UP ARROW’

LEFT: ‘LEFT ARROW’

RIGHT: ‘RIGHT ARROW’

DOWN: ‘DOWN ARROW’

PAUSE : ‘BACKSPACE’

EXIT : ‘BACKSPACE’ in PAUSE MODE.

RESTART : ‘ENTER’

The top score is located in the start menu in the form of the length of the ‘G’ letters. The more ‘G’ displayed, the higher the score. No in-game control exists for sound volume modification, the user must adjust the sound accordingly on his own operating system. No specific extra software is required to run the game such as DirectX or OpenGL, etc…