الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and scientific research
المدرسة الوطنية العليا للتكنولوجيا
National Higher School of Technology
Department of Electrical Engineering And Industrial Computing

---

Final Year Project to Obtain the Diploma of Engineering

-Field-
Electronics

-Speciality-
Embedded Systems

-Subject-

Design and Development of an Educational Gaming Console Based on Arduino with User-created Games

by

Nesrine BERRAHMOUNE

Examination Committee:

Mr. ZAOUI Abdelhalim	Chair
Mrs. DROUICHE Imane	Supervisor
Mrs. HABANI Lamia	Supervisor
Mrs. BOUCHAMA Samira	Examiner
Mr. KHOUMERI El Hadi	Examiner

Algiers, the ../../2023

Academic Year 2022 –2023

Acknowledgment

First and foremost, I would like to express my gratitude to Allah the Almighty for granting me the courage and patience to successfully complete this work.

I extend my heartfelt thanks to **Mrs. DROUICHE Imane**, my supervisor, for her unwavering belief in my project idea and for providing all the necessary facilities to bring it to fruition. Her patience and continuous support have been invaluable.

I am deeply grateful to **Mrs. HABANI Lamia** for her valuable feedback and assistance throughout the project. Her input has significantly enhanced the quality of my work.

I would like to express my sincere appreciation to the teachers who selflessly shared their knowledge and expertise with me over the years. Without their guidance, I would not have been able to transform the project into a reality.

A special word of thanks goes to the staff of the **Department of Electrical Engineering and Industrial Computing** for their steadfast support during the highs and lows of my academic journey.

I would like to acknowledge and thank **Mr. SLIMANI Oussama**, the Embedded Systems Doctoral Candidate, and **Mr. GOURINE Ahmed**, the Automation Engineer, for their assistance at the Electronics Laboratory of the University of Sciences and Technologies.

I am grateful to every individual who contributed, no matter how small their role, to the development of this project. Your collective efforts have made a significant impact.

In conclusion, I am hopeful that the knowledge I have acquired through this project will serve as a catalyst for the growth of Algeria, specifically within the Education System.

“

I dedicate this work to

My dearest Parents, whose heartfelt prayers have been a constant source of strength and inspiration throughout my journey. Thank you for being the pillar of my life.

My Siblings, ***Anouar****,* ***Rania*** *and* ***Mohamed*** *for being a source of encouragement and motivation.*

To my second family, ***Bahia*** *and* ***Nada****, who have been with me through thick and thin.*

To each of you, I offer my deepest gratitude. Your presence in my life has made it richer, and I am truly blessed to have you all by my side.

”

Abstract

This project report showcases the design and development of an educational gaming console tailored to the Algerian curriculum's block-based programming integration using Scratch. The console aims to empower students from second grade in middle school and beyond,with computational thinking fostering their creativity and active participation in learning.

A custom shield was designed for the Arduino Mega board, incorporating essential gaming console components. The software development section explores game implementation using Arduino IDE and mBlock, with a focus on compatibility with the educational programming language.

---

keywords: Educational Console, Novice Programmers, Block-Based Programming, Game Development.

Résumé

Ce rapport de projet met en valeur la conception et le développement d'une console de jeu éducative adaptée à l'intégration de la programmation par blocs dans le programme scolaire algérien en utilisant Scratch. La console vise à autonomiser les élèves à partir de la deuxième année du collège et au-delà en développant leur pensée informatique, en favorisant leur créativité et leur participation active à l'apprentissage.

Un shield personnalisé a été conçu pour la carte Arduino Mega, intégrant les composants essentiels d'une console de jeu. La section sur le développement logiciel explore l'implémentation de jeux en utilisant Arduino IDE et mBlock, en mettant l'accent sur la compatibilité avec le langage de programmation éducatif.

---

mots clés : Console éducative, Programmeurs débutants, Programmation par blocs, Développement de jeux.

ملخص

هذا المشروع يعرض تصميم وتطوير جهاز يسمح للمستخدم أن يبرمج فيه العاب بفاعلية تعلم البرمجة. حيث يتكامل مع المنهج الجزائري الجديد الذي يضمن سكراتش إبتداءً من الصف الثاني من الطور المتوسط .هدف المشروع هو تعزيز مهارات الطلاب في التفكير الخوارزمي من خلال تنمية قدراتهم الإبداعية وتشجيعهم على المشاركة الفاعلة في عملية التعلُّم.

تم تصميم درع مخصص لاردوينو ميغا يشتمل على جميع المكونات الأساسية التي تتواجد في أجهزة الألعاب. يستكشف قسم تطوير البرمجيات تنفيذ الألعاب باستخدام بيئة تطوير الأردوينو وبرنامج المبلوك ، مع التركيز على التوافق مع لغة البرمجة التعليمية

كلمات مفتاحية :الجهاز التعليمي، المبرمجون المبتدئون، البرمجة للاطفال ، تطوير الألعاب

Contents

Acknowledgment . . . . . **I**

Abstract . . . . . **III**

Résumé . . . . . **IV**

ملخص . . . . . **V**

Introduction **1**

1 Embedded Systems for Educational Game Development **2**
1.1 Embedded Systems 3
1.1.1 Principle of Operation 3
1.1.2 Constraints in Embedded Systems Development 3
1.1.3 Applications of Embedded System 4
1.2 Programming Embedded Systems 5
1.2.1 Programming Paradigms 5
1.2.2 Arduino Programming 10
1.2.3 Visual Programming 12
1.3 Integration of Block Based Programming in Algerian Schools 13
1.3.1 Scratch 13
1.3.2 BBC microbit 16
1.3.3 mBlock 17
1.4 Game Development For Interactive Learning 18
1.4.1 Basic Computational Concepts in Game Development 18
1.4.2 Core Elements of Game Development 19
1.4.3 Benefits of the Game Development Approach 19
1.5 Conclusion 20

2 Project Framework **21**
2.1 Introduction 22
2.2 Objectives 22
2.3 Scope 22
2.4 Requirements 23
2.4.1 Software Tools 23
2.4.2 Hardware Components 25
2.5 Synoptics Diagram 34
2.6 Component Pricing Table 35

2.7 Methodology . . . 35
2.7.1 Circuit prototyping . . . 36
2.7.2 PCB Fabrication . . . 36
2.7.3 Game Implementation . . . 37
2.7.4 Integration of Block Based Programming . . . 38
2.8 Conclusion . . . 38

3 Implementations . . . 40
3.1 Hardware Integration and Prototyping . . . 41
3.1.1 Arduino Mega 2560 . . . 41
3.1.2 LED Matrices with MAX7219 . . . 42
3.2 PCB Development . . . 42
3.2.1 Single Layer PCB Development and Testing . . . 42
3.2.2 Two-Layer PCB development . . . 45
3.3 Functionality Test . . . 48
3.4 Game Development . . . 50
3.4.1 Game Development Setup . . . 50
3.4.2 Game Development Constraints on the Console . . . 52
3.4.3 Potential Games . . . 54
3.4.4 Snake Game Implementation . . . 54
3.5 Compatibility with Block-Based Programming . . . 56

Conclusion and Perspectives . . . 58
3.6 Conclusion . . . 59

List of Figures

1.1 Input-Processing-Output Diagram . . . 3
1.2 Object Oriented Program Paradigm Example . . . 6
1.3 Functional Programming Paradigm Example. . . . 7
1.4 Procedural Programming Paradigm Example . . . 8
1.5 Imperative Programming Paradigm Example . . . 9
1.6 Arduino Language References [3]. . . . 10
1.7 Scratch Interface . . . 14
1.8 Types of Blocks in Scratch . . . 14
1.9 Colour Coded Scratch Blocks Palette . . . 15
1.10 mBlock Interface . . . 16
1.11 BBC micro:bit device . . . 16
1.12 mBlock Interface . . . 17
1.13 mBlock Boards . . . 17
1.14 mBlock Extentions . . . 18
1.15 Elements of Game Development . . . 19

2.1 Synoptics Diagram before the Hardware Selection . . . 26
2.2 Arduino mega Pinout Diagram . . . 27
2.3 Joystick Module [8]. . . . 29
2.4 Joystick Axis [8]. . . . 29
2.5 Input-Processing-Output Diagram [8] . . . 30
2.6 Input-Processing-Output Diagram [8] . . . 30
2.7 LED matrix multiplexing [8] . . . 32
2.8 MAX7219 module [8] . . . 32
2.9 Synoptics Diagram after Selection. . . . 34
2.10 Methodology Diagram. . . . 35

3.1 Breadboard Design using Fritzing. . . . 41
3.2 PCB Schematic using EasyEDA . . . 42
3.3 PCB Layout using EasyEDA . . . 43
3.4 Exposing the Photoresist to UV Light . . . 43
3.5 Developed Circuit Pattern After Treatment . . . 44
3.6 Etched PCB Surface After Etching Process . . . 44
3.7 Drilled Holes for Components . . . 44
3.8 Complete One Layer PCB After Soldering . . . 45
3.9 Two-Layer PCB Layout using Eagle . . . 45
3.10 Top View of Manufactured Two-Layer PCB . . . 46
3.11 Bottom View of Manufactured Two-Layer PCB . . . 47

3.12 Top View of Completed PCB after Soldering 47
3.13 Bottom View of Completed PCB after Soldering 48
3.14 Snake Game 55
3.15 Board Selection in mBlock 56
3.16 Serial Port Selection in mBlock 56
3.17 Successfully Connected 56
3.18 Upload the program in mBlock 57

List of Tables

2.1 Software Tools for Code Prototyping 23
2.2 Software Tools for Embedded Systems Design 24
2.3 Factors for Software Selection 25
2.4 Arduino Mega Technical Specifications 27
2.5 MPU6050 Technical Specifications 29

Abbreviations List

CNC	***C****omputer* ***N****umerical* ***C****ontrol*
CLK	***C****lock*
CS	***C****hip* ***S****elect*
DIN	***D****ata* ***I****nput*
IDE	***I****ntegrated* ***D****evelopment* ***E****nvironment*
I2C	***I****nter-****I****ntegrated* ***C****ircuit*
LED	***L****ight-****E****mitting* ***D****iode*
MPU6050	***M****otion* ***P****rocessing* ***U****nit 6050*
OOP	***O****bject-****O****riented* ***P****rogramming*
PCB	***P****rinted* ***C****ircuit* ***B****oard*
RGB	***R****ed* ***G****reen* ***B****lue*
SCL	***S****erial* ***C****lock*
SDA	***S****erial* ***D****ata*
SPI	***S****erial* ***P****eripheral* ***I****nterface*
UV	***U****ltraviolet*

Introduction

Did you know that the Algerian curriculum has taken a remarkable stride by integrating block-based programming using Scratch from second grade in middle school? This progressive approach equips students with vital programming fundamentals through an accessible and user-friendly programming language, setting the stage for their success in the digital age. However, to truly harness the power of programming, it is important to overcome some existing limitations and revolutionize the way we teach this essential skill.

In this report, we present the design and development of an educational gaming console that aligns seamlessly with the integration of block-based programming in the Algerian curriculum. Our aim is to enhance students' understanding of programming concepts, foster critical thinking and problem-solving skills, and ignite their passion for learning.

This report is structured into multiple chapters, each dedicated to exploring vital aspects of our project in detail.

Chapter 1 titled "Embedded Systems for Educational Game Development" provides an overview of embedded systems, It also explores the benefits of block-based programming and its integration within the Algerian curriculum. Additionally, it highlights the specific area of development within the curriculum and the importance of game development for interactive learning.

Chapter 2, titled "Project Framework" outlines the objectives and scope of the project. It provides an overview of the required materials, The chapter also details the methodology employed to ensure a systematic and efficient execution of the project.

In **Chapter 3**, titled "Implementation, PCB Development, and Testing," we delve into the development of a circuit prototype for the gaming console. We discuss the challenges encountered during the process and showcase how they were successfully overcome. Additionally, we explore the implementation of game development using Arduino IDE and mBlock, two widely used platforms.

Finally, **Chapter 4** provides a comprehensive summary of the project's key findings, reflecting on the achievements, challenges, and future possibilities for development. It also includes the reviews received from students, offering valuable insights into their experiences with the educational gaming console.

Our educational gaming console opens doors for creativity, collaboration, and exploration, allowing students to not only play games but also become creators of their own digital worlds.

Chapter 1

Embedded Systems for Educational Game Development

1.1 Embedded Systems

An embedded system is an autonomous computer and electronic system integrated into a device to perform a specific task. It operates in real-time and encompasses both hardware and software components. The software is executed on a micro-controller, microprocessor, or occasionally an FPGA, depending on the required computing power and necessary interfaces such as communication bus, screen, analog measurements, and more [1]

1.1.1 Principle of Operation

Embedded computer systems operate on a fundamental principle of information flow. They acquire data from the external environment through input devices or sensors, process it within the microprocessor, and then transmit the processed information back to the external world through output devices or actuators. This continuous cycle of information exchange enables the embedded system to interact, respond, and perform its designated functions effectively. See the diagram below for an overview of the information flow in an IPO system.

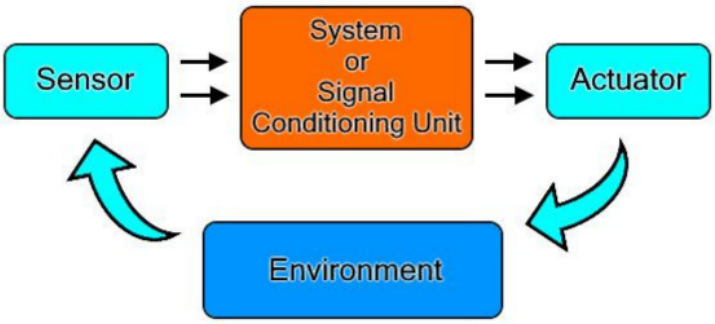


Figure 1.1: Input-Processing-Output Diagram

1.1.2 Constraints in Embedded Systems Development

Embedded systems must adhere to the following key constraints:

- **Autonomy** It should operate for extended periods without human intervention.
- **Reliability** Continuity of service and accurate results are crucial.
- **Reactivity** Timely response to the environment, often in real-time, is required.
- **Small Size** They must fit within limited physical space.
- **Security** Confidential data should be securely collected and stored.
- **Appropriate Computing Power** Energy consumption must be optimized by adjusting computational resources.

1.1.3 Applications of Embedded System

An embedded system can be defined as an autonomous electronic and computer system integrated into hardware equipment and dedicated to a specific task. These systems are used in more and more areas:

- **Industrial Automation**: Embedded systems are extensively used in industrial settings for control and monitoring of machinery, robotics, process automation, and quality control systems.
- **Smart Grids**: Embedded systems play a crucial role in modern power grids, enabling efficient energy management, smart metering, and integration of renewable energy sources.
- **Internet of Things (IoT)**: Embedded systems form the foundation of IoT devices, connecting everyday objects to the internet and enabling communication and data exchange between devices and users.
- **Aerospace and Defense**: Embedded systems are vital in aerospace and defense applications, including aircraft control systems, navigation systems, missile guidance systems, and surveillance equipment.
- **Consumer Electronics**: Embedded systems power a wide range of consumer electronics, including smartphones, tablets, smart TVs, digital cameras, wearable devices, and home entertainment systems.
- **Automated Home Systems**: Embedded systems are used in home automation systems to control lighting, temperature, security systems, and appliances, providing convenience, energy efficiency, and improved living experiences.
- **Financial Systems**: Embedded systems are utilized in financial institutions for secure transactions, electronic payment systems, ATMs, and point-of-sale devices.
- **Environmental Monitoring**: Embedded systems enable environmental monitoring and data collection for applications such as weather stations, air quality monitoring, and wildlife tracking.
- **Robotics**: Embedded systems form the backbone of robotics, enabling control, sensory feedback, and decision-making capabilities in autonomous robots and robotic systems.
- **Embedded Systems in Education**: Embedded systems are increasingly being incorporated into educational settings to teach programming, electronics, and computational thinking, allowing students to develop hands-on skills and creativity.

1.2 Programming Embedded Systems

The following section introduces the key principles and techniques of programming embedded systems. It explores the interaction between software and physical components, emphasizing the crucial role of programming in achieving seamless integration of software and hardware.

1.2.1 Programming Paradigms

Paradigms in embedded systems refer to types or styles of programming that provide fundamental approaches to guide the design and development of embedded systems. These paradigms encompass principles, methodologies, and techniques for organizing and implementing embedded software and hardware. The next section introduces various programming paradigms in embedded systems.

Popular programming paradigms include imperative programming, procedural programming, functional programming, declarative programming, object-oriented programming.[2]

Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming paradigm that revolves around the concept of classes, objects, and methods. It provides a way to organize and structure software by creating reusable code blueprints known as classes, from which individual objects can be instantiated. OOP is widely implemented in various programming languages such as JavaScript, C++, Java, and Python. It is considered the standard approach for coding in many educational programs.

- **E**xample

  The following snipped code demonstrates the use of OOP paradigm to blink a LED, it is organized into two classes: LEDControllerand SerialCommunication. The LEDController class represents the LED and provides methods to turn it on and off. It takes the LED pin as a parameter in the constructor and sets the pin mode in the initialization. The turnOn and turnOff methods control the state of the LED. The SerialCommunication class provides a static method readData to read the data from the Serial port. It checks if there is data available and returns it, or returns a null character if no data is available. In the main function, an instance of LEDController is created, and the pin number is passed as an argument. The while loop continuously reads data from the Serial port using the SerialCommunication class. If the received data is '1', the LED is turned on, and if it is '0', the LED is turned off.

```cpp
class LEDController {
private:
    const int pin;
public:
    LEDController(int ledPin) : pin(ledPin) {
        pinMode(pin, OUTPUT);
    }
    void turnOn() {
        digitalWrite(pin, HIGH);
    }
    void turnOff() {
        digitalWrite(pin, LOW);
    }
};
class SerialCommunication {
public:
    static char readData() {
        if (Serial.available()) {
            return Serial.read();
        }
        return '\0';
    }
};
int main() {
    const int ledPin = 13;
    LEDController led(ledPin);
    while (true) {
        char received = SerialCommunication::readData();
        if (received == '1') {
            led.turnOn();
        } else if (received == '0') {
            led.turnOff();
        }
    }
    return 0;
}

```

Figure 1.2: Object Oriented Program Paradigm Example .

Functional Programming

Functional programming is a coding style that emphasizes using functions as building blocks. It promotes immutability and avoids mutable state, focusing on declarative and pure functions. This improves code maintainability and readability by prioritizing modularity, reusability, and composability. Functional programming supports parallelism, enables better testability, and enhances code clarity. Languages like Haskell and Lisp fully embrace these principles, while functional concepts can be applied in non-purely functional languages like JavaScript and Python.

- **E**xample

  The following snipped code shows how to control LED blinking using functional programming paradigm, functions are defined to control the LED. The turn.On() and turn.Off() functions handle turning the LED on and off, while the blink.led() function combines these actions to create a blinking effect.

  The main program enters an infinite loop where the LED blinks repeatedly by calling the blink.led() function.

```python
from time import sleep

def turn_on(pin):
    print(f"Turning on LED at pin {pin}")
    # Code to turn on the LED

def turn_off(pin):
    print(f"Turning off LED at pin {pin}")
    # Code to turn off the LED

def blink_led(pin):
    turn_on(pin)
    sleep(1)  # Delay for 1 second
    turn_off(pin)
    sleep(1)  # Delay for 1 second

def main():
    led_pin = 13

    while True:
        blink_led(led_pin)

if __name__ == "__main__":
    main()
```

Figure 1.3: Functional Programming Paradigm Example.

Procedural Programming

Procedural programming is a fundamental programming paradigm where instructions are written in a step-by-step manner to guide the computer in completing a task. It follows a linear top-down approach, treating data and procedures as separate entities. The key features of procedural programming include: predefined functions, local variables, global variables, modularity, parameter passing.

- **Example**

  The following snipped code shows an example of controlling a LED using a procedural programming paradigm in Python where "board' and 'digitalio' libraries was used to control a LED connected to a specfic pin on the microcontroller board.

  The blink.led() procedure is responsible for blinking the LED. It turns on the LED by setting led.pin.value to True, waits for 1 second using time.sleep(), turns off the LED by setting led.pin.value to False, and waits for another 1 second.

  In the main program, we enter an infinite loop where the LED blinks repeatedly by calling the blink.led() procedure.

```
2  import board
3  import digitalio
4  import time
5
6  # Define LED pin
7  led_pin = digitalio.DigitalInOut(board.LED)
8
9  # Set LED pin as output
10 led_pin.direction = digitalio.Direction.OUTPUT
11
12 # Blink the LED Procedure
13 def blink_led():
14     led_pin.value = True  # Turn on the LED
15     time.sleep(1)         # Delay for 1 second
16     led_pin.value = False # Turn off the LED
17     time.sleep(1)         # Delay for 1 second
18
19 # Main program
20 while True:
21     blink_led()
```

Figure 1.4: Procedural Programming Paradigm Example

Imperative Programming

Imperative programming is a paradigm where code explicitly defines each step to solve a problem, without relying on pre-coded models. The focus of imperative programming is how the problem should be solved, which requires a detailed step-by-step guide. Because the written code performs the functions instead of models, the programmer must code each step. Imperative programming languages include: C, C++, Java, Python, Ruby, JavaScript, Swift, Go, Rust, PHP.

- **Example** In the following scenario, a program of Blink LED control is created using Arduino IDE which is an open-source electronics platform that interacts with Arduino boards and many others micro controller and microprocessor boards.

  – Connect LED to pin13 of the Arduino Board

  – Upload the following code to the Arduino Board

```
const int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);   // Turn on the LED
  delay(1000);                  // Delay for 1 second
  digitalWrite(ledPin, LOW);    // Turn off the LED
  delay(1000);                  // Delay for 1 second
}
```

Figure 1.5: Imperative Programming Paradigm Example .

Analysis

By examining the examples of controlling an LED to blink in different programming paradigms, we can draw some conclusions. In object-oriented programming (OOP), the use of classes and objects provides a structured and modular approach to control the LED. Functional programming promotes modularity and composability through the use of functions. Procedural programming follows a series of step-by-step procedures to achieve the blinking effect. Imperative programming, with its explicit coding of each step, allows for precise control of the LED.

Among these paradigms, imperative programming stands out as the most suitable choice for novice programmers and educators. Its clear and straightforward approach, focusing on explicit steps, helps beginners grasp the logic of embedded systems programming more easily. Additionally, the Arduino IDE offers a user-friendly and simplified programming environment that greatly appeals to beginners and educators alike.

1.2.2 Arduino Programming

The Arduino programming language is based on C and C++, providing a familiar syntax and structure for programmers. It encompasses a set of core functions and libraries that simplify hardware interactions and enable users to control various components such as sensors, actuators, and displays. These libraries offer pre-written code for common tasks, allowing users to concentrate on the specific functionality of their projects without getting entangled in low-level details. The Arduino IDE provides a solid foundation for learners to experiment, create, and gain confidence in their embedded systems development journey.

Language References

Arduino programming language can be divided into three main parts: functions, values (variables and constants), and structure.

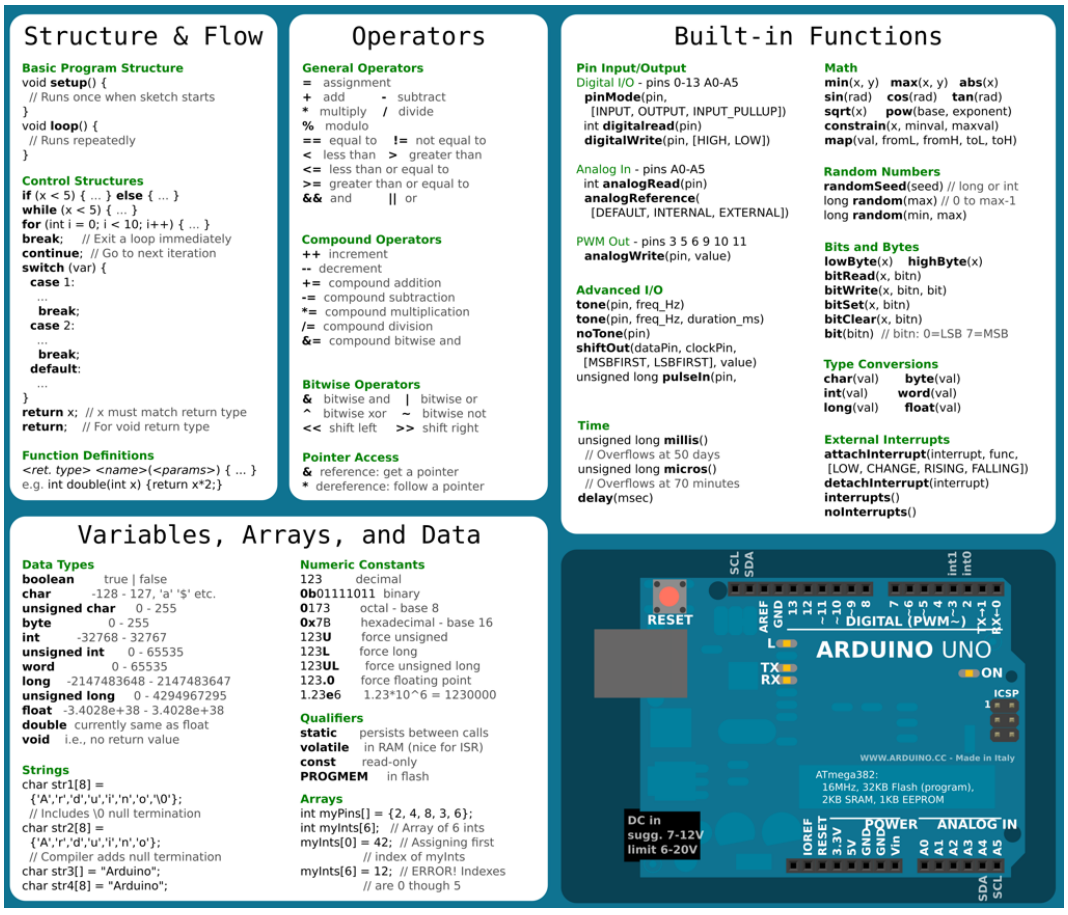


Figure 1.6: Arduino Language References [3].

- **Functions**

  There are variety of functions for controlling the Arduino board and performing computations. These functions cover digital and analog I/O, advanced I/O operations, timing, math calculations, trigonometry, character operations, random number generation, bits and bytes manipulation, interrupts, and communication protocols. They enable efficient interaction with hardware components and facilitate the development of diverse embedded systems applications.

- **Variables**

  Variables, arrays, and data are fundamental elements in Arduino programming. Variables store and manipulate data, while arrays allow for storing multiple values of the same data type in a sequential manner. Arduino supports various data types such as integers, floating-point numbers, booleans, characters, and strings to handle different kinds of data.

- **Structure**

  Arduino (C++) code comprises various elements that enable the development of functional and interactive sketches.

Libraries

The Arduino environment offers a wide range of libraries that extend its functionality for various purposes. These libraries provide additional capabilities for tasks such as working with hardware, manipulating data, communication, data processing, device control, display, sensors, signal input/output, timing, and more. Some libraries are included by default in the Arduino IDE, while others can be downloaded or created by users. To utilize a library in a sketch, you can import it through the Sketch menu. The available libraries cover different categories, including communication, data processing, data storage, device control, display, sensors, signal input/output, timing, and others. These libraries greatly enhance the versatility and flexibility of the Arduino platform, allowing developers to leverage pre-existing code and streamline their embedded systems projects.

In conclusion, while the Arduino IDE provides a convenient way to avoid low-level complexities and is suitable for learners passionate about programming, it may not be the optimal choice for getting kids interested in the field. For young novice programmers, visual programming languages offer a more engaging and interactive experience.

1.2.3 Visual Programming

A visual programming language (VPL) or block coding is a programming language that allows users to create programs by manipulating program elements graphically instead of using textual specifications. VPLs use visual expressions and spatial arrangements of text and graphic symbols, often represented as "boxes and arrows," to represent program logic and relationships. This visual approach simplifies programming and makes it more accessible, particularly for beginners and non-technical users.

The primary objective of visual programming languages (VPLs) is to enhance accessibility for novice programmers and provide support at three different levels.

- **Syntax**

  VPLs utilize visual elements like icons/blocks, forms, and diagrams to prevent syntactic errors and enable the creation of well-formed programs.

- **Semantics**

  VPLs often incorporate mechanisms to provide information about the meaning of programming primitives. This can be achieved through built-in documentation functions and helpful features integrated within the programming language.

- **Pragmatics**

  VPLs facilitate the exploration of program behavior in specific situations, enabling users to manipulate artifacts created with the VPL to observe the program's response. For example, in platforms like AgentSheets or AgentCubes, users can set games or simulations to a particular state to observe the program's behavior. Similarly, in the Thymio programming language, users can manipulate a robot into a specific state to observe the activation of sensors and the corresponding reactions.

List of Visual Programming Languages Categories

Visual Programming Languages (VPLs) cover various categories, including:

- **Educational**

  These VPLs are named Block Based Programming Langauges specifically designed to introduce programming concepts and computational thinking to learners, particularly children. Examples include Scratch, Alice, Blockly, and mBlock.

- **Multimedia**

  VPLs in this category focus on creating audio, video, and multimedia content. They often include features for sound design, image processing, and 3D modeling. Examples include Max/MSP, Pure Data, TouchDesigner, and Isadora.

- **Video Games**

  VPLs in the video game category are used for developing and designing games. They provide tools for creating game mechanics, interactions, and graphics. Examples

include Unity (with its visual scripting system), GameMaker Studio, Construct, and Unreal Engine (with Blueprints).

- **Systems/Simulations**

  These VPLs are used for modeling and simulating complex systems, such as control architectures, electronic circuits, and data analytics. Examples include LabVIEW, Simulink, OpenModelica, and Node-RED.

- **Automation**

  VPLs in the automation category are used for creating workflows, orchestrating processes, and automating tasks. They often involve integrating different software systems and data processing. Examples include Apache Nifi, IBM InfoSphere DataStage, and Microsoft SQL Server Integration Services (SSIS).

In conclusion, Visual Programming Languages (VPLs) offer a user-friendly and accessible approach to programming embedded systems. VPLs simplify the programming process and make it engaging, especially for learners and beginners. Scratch, as one of the most popular VPLs for education, provides a seamless introduction to programming concepts and empowers users to create interactive projects. The integration of Scratch in the Algerian education system will be further explored in the next section.

1.3 Integration of Block Based Programming in Algerian Schools

Algerian schools have embarked on a remarkable path to equip the next generation with 21st-century skills, including programming and computational thinking. This progressive approach recognizes the importance of introducing students to coding at an early stage, fostering creativity, problem-solving, and digital literacy. As part of this effort, various block-based programming platforms have been integrated into Algerian schools, enabling students to engage in interactive and hands-on learning experiences. [4]

1.3.1 Scratch

Public schools in Algeria have integrated Scratch, a widely-used block-based programming language, into their curriculum. This powerful tool enables students to unleash their creativity by designing animations, games, and interactive stories through an intuitive drag-and-drop interface. As part of the Algerian curriculum, students learn fundamental programming concepts, including events, loops, conditions, and variables, which empower them to create dynamic and engaging projects. By combining hands-on coding experiences with educational content, Scratch equips Algerian students with essential digital skills and cultivates their problem-solving abilities.

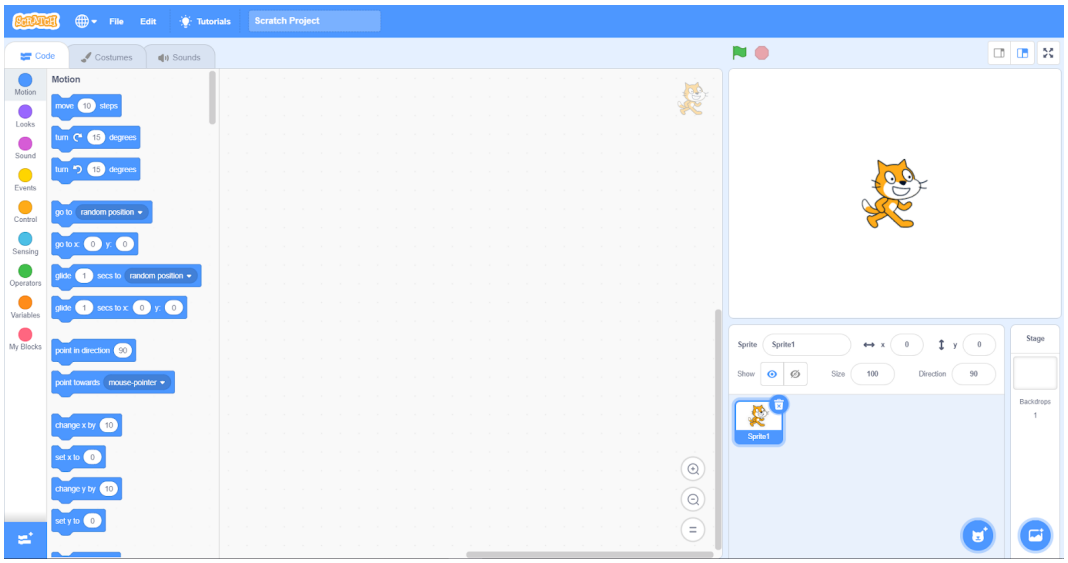


Figure 1.7: Scratch Interface

Types of Blocks

With over 100 different Scratch blocks available that can be put into 1 of 4 types

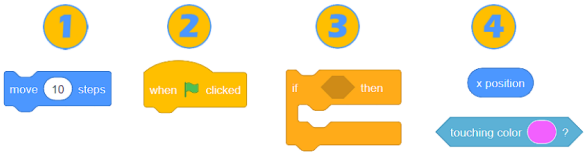


Figure 1.8: Types of Blocks in Scratch .

- **Stack Blocks**

  The majority of Scratch blocks fall into this category. Stack blocks have a notch at the top and a bump at the bottom, allowing them to be connected like Lego pieces to create a script.

- **Hat Blocks**

  Hat blocks have a rounded or curved top and a bump at the bottom, indicating that they can be attached to stack blocks. Hat blocks enable the creation of event-driven scripts that execute in response to specific events, such as clicking the green flag.

- **Control Blocks**

  Similar to command blocks, have notches at the top and bumps on the bottom. They can be placed above or below other blocks, allowing for stacking. Additionally, control blocks have the capability to contain other blocks within them.

- **Reporter Blocks**

  Reporter blocks provide information for other blocks to process. They have rounded or angled sides and fit inside other blocks without notches and bumps. For instance, reporter blocks can retrieve the sprite's location along the x-y axis.

Categories of Blocks

Scratch offers a wide range of code blocks, totaling over 100, categorized into eight main groups on the blocks palette. These categories include:

- **Motion** Control blocks for sprite placement, movement, rotation, and direction.
- **Looks** Blocks that modify sprite and background appearance and enable text display.
- **Sound** Blocks to control musical notes and audio file playback, adjusting volume.
- **Events** Blocks that trigger script execution
- **Control** Blocks that handle loops and implement conditional logic
- **Sensing** Blocks to determine mouse-pointer location, distance between sprites, and sprite interactions.
- **Variables** Blocks to store and retrieve data during program execution.



Figure 1.9: Colour Coded Scratch Blocks Palette .

However, it's important to acknowledge that Scratch, while being a versatile platform for learning programming concepts, is primarily focused on virtual simulations and lacks direct visibility of programs on physical devices. Students using Scratch may not have the opportunity to see their programs running on real-world consoles or hardware. This limitation highlights the need for complementary tools that bridge the gap between virtual and physical computing experiences.

1.3.2 BBC microbit

Private institutions have recognized the value of BBC micro:bit as a learning tool to acquire programming skills. The micro:bit software application offers block-based programming similar to Scratch, organized into various categories. These categories include Input, Output, Basic, Logic, Loops, Variables, Functions, and Extensions. These blocks empower students to create diverse projects related to electronics and explore different programming concepts.

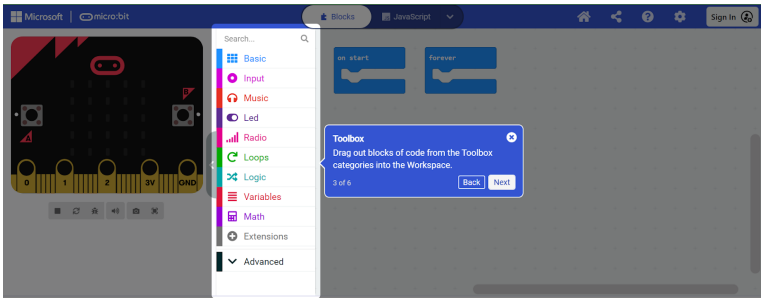


Figure 1.10: mBlock Interface .

BBC microbit Features

The micro:bit device itself is a small, pocket-sized microcontroller packed with features. It includes buttons, 5*5 LED matrix, an accelerometer, a compass, and a Bluetooth module, among others. This compact yet powerful device enables students to bring their code to life by uploading and running it on a physical device. The immediate feedback and tangible interaction foster a deeper understanding of programming principles. One

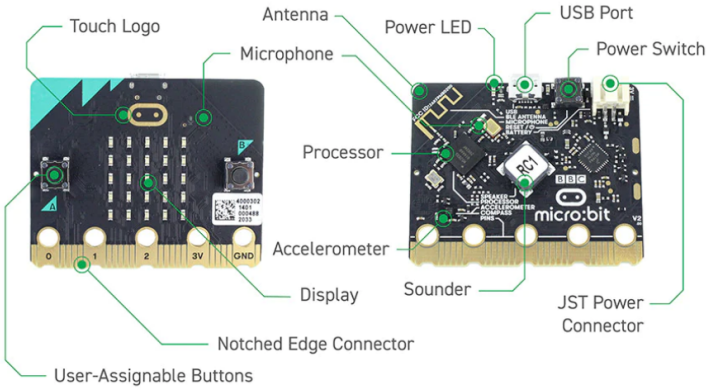


Figure 1.11: BBC micro:bit device

limitation of the micro:bit platform is its compatibility with the specific hardware device. While students can experience the joy of running their code on a real device, they are restricted to using devices with the micro:bit brand. This closed ecosystem means that users cannot utilize other hardware options or extend their programming skills to different devices.

1.3.3 mBlock

mBlock, an open-source programming language based on Scratch, is a valuable tool to equip children with essential robotics and computer science skills. Beyond Tekhmamek, an organization dedicated to modernizing education, recognizes the potential of mBlock and has chosen it as a tool to empower young learners.

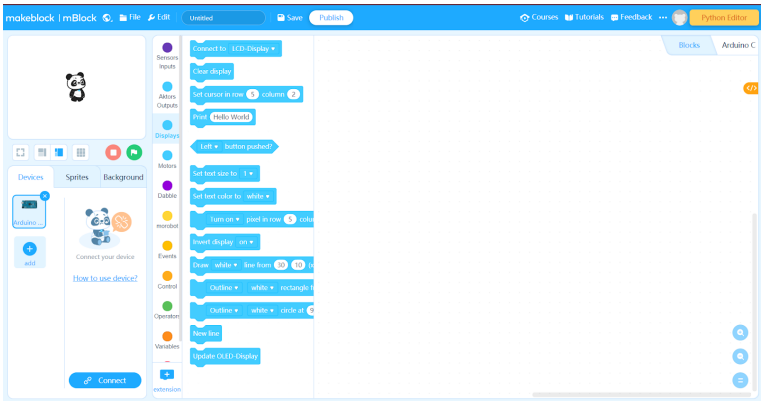


Figure 1.12: mBlock Interface .

mBlock Features

One of the standout features of mBlock is its compatibility with a wide range of microcontrollers, including Makeblock, Arduino, and ESP32 boards.

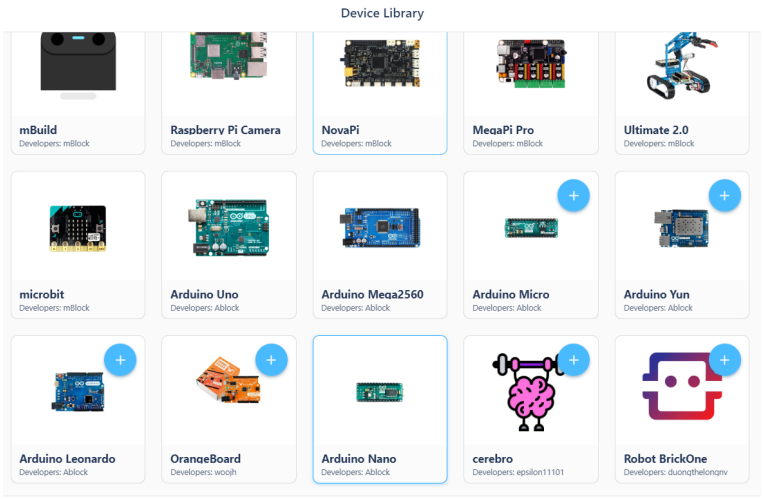


Figure 1.13: mBlock Boards .

mBlock allows users to easily incorporate extensions to interact with a variety of additional hardware components.

The integration of block-based programming platforms, such as Scratch, micro:bit, and mBlock, in Algerian schools and institutions has had a remarkable impact on cultivating students' enthusiasm for technology and programming from an early age. While each platform has its unique features and limitations, none of them have been fully utilized to

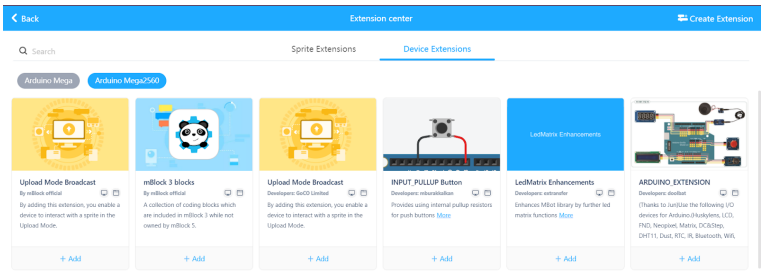


Figure 1.14: mBlock Extentions .

their potential. However, through further research, it has been discovered that combining the features of these tools with a game development approach can provide a powerful and engaging learning experience for students. By leveraging the strengths of these platforms and incorporating game development principles, Algerian educators can unlock the full potential of block-based programming in teaching computer science, fostering creativity, problem-solving, and a passion for technology among students.

1.4 Game Development For Interactive Learning

Game development provides a unique and engaging approach to teaching programming concepts. As game designers program interactive experiences, they often rely on fundamental computational concepts, such as variables, loops, conditionals, and event handling. This section aims to explore the game development approach as a solution to promote interactive learning.

1.4.1 Basic Computational Concepts in Game Development

Game development inherently relies on various computational concepts that designers utilize when programming. These concepts include:

- **Variables** Games often require the use of variables to store and manipulate data. For instance, variables can represent a player's score, health, or the position of game objects.
- **Loops** Loops are essential for creating repetitive behaviors in games, such as updating the game state or iterating through lists of game objects.
- **Conditionals** Conditionals enable game designers to control the flow of the game based on specific conditions. They allow for decisions to be made, such as determining whether a player has achieved a goal or detecting collisions between game objects.
- **Event Handling** Games heavily rely on events triggered by user interactions, such as button clicks or keyboard inputs. Designers use event handling to respond to these interactions and create dynamic game experiences.

1.4.2 Core Elements of Game Development

In addition to programming concepts, game development incorporates several core elements that contribute to the overall game experience. These elements include :

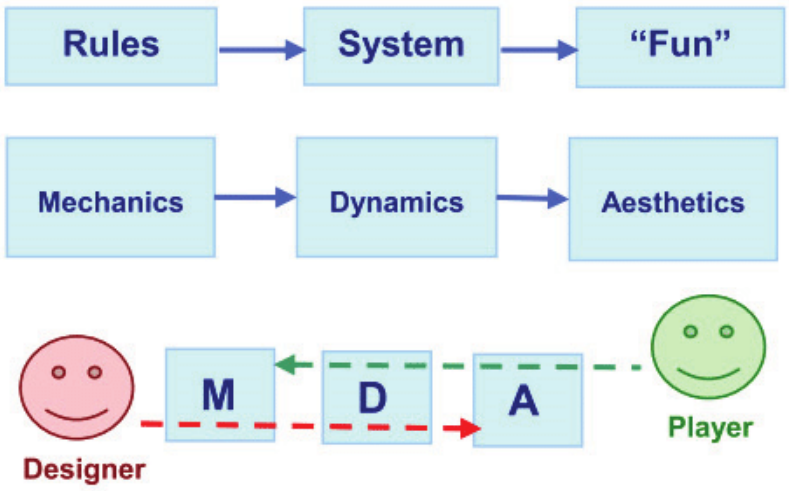


Figure 1.15: Elements of Game Development .
[5]

- **Mechanics** The rules and interactions that govern the gameplay experience, including goals, challenges, and progression.
- **Dynamics** The behaviors and responses of the game system and its components, creating a sense of flow and engagement.
- **Aesthetics** The visual and auditory elements of the game, including art, sound, and music, which contribute to the overall atmosphere and player immersion.
- **User Interface and User Experience** The design of interfaces and interactions that facilitate player engagement and ease of use. [5]

1.4.3 Benefits of the Game Development Approach

The game development approach presents numerous benefits in the realm of programming education. By incorporating game development into the curriculum, students can experience a range of advantages that enhance their learning journey.

Concrete Examples

Game development offers tangible and visual examples of abstract programming concepts. Students can see how variables affect game behavior, how loops create repetitive actions, how conditionals impact game decisions, and how event handling responds to user interactions. These concrete examples facilitate a deeper understanding of programming concepts.

Active Learning

Engaging in game development allows students to actively participate in the learning process. They have the opportunity to experiment, make mistakes, and iterate on their designs, fostering a deeper understanding of programming concepts through trial and error.

Practical Application

Game development bridges the gap between theoretical knowledge and practical application. Students can see firsthand how programming concepts translate into interactive game mechanics, making the learning process more relevant and meaningful.

Collaboration and Creativity

Game development encourages collaboration among students, promoting teamwork and problem-solving skills. Students can work together to design and implement games, sharing ideas and learning from each other's experiences. Additionally, game development fosters creativity as students have the freedom to express their ideas through interactive and engaging games.[6]

1.5 Conclusion

In conclusion, the integration of embedded systems into game development provides an innovative and engaging way for students to learn computational thinking. By incorporating platforms like Arduino IDE and block-based programming, students can explore the practical applications of programming in the context of game development. This approach not only enhances their problem-solving abilities but also fosters creativity, collaboration, and critical thinking. Ultimately, the aim of this chapter is to empower students to become proficient in computational thinking through the exciting medium of game development with embedded systems.

Chapter 2

Project Framework

2.1 Introduction

Game development has emerged as a powerful approach in teaching computer science fundamentals, offering enhanced student engagement and fostering creativity, problem-solving skills, and critical thinking. It has been recognized as a valuable tool for creating interactive and immersive learning experiences. In this chapter we will Introduce the project framework, including the objectives, and the scope of the project, and then delve into the selection process of the software tools and hardware components that are required.

2.2 Objectives

The primary objectives of this project are as follows:

- Develop an educational game console using embedded systems to make the learning process more interactive and enjoyable for students.
- Enable students to design, develop, and play their own games, promoting hands-on learning and active involvement in the educational experience.
- Support block-based programming to remove barriers and allow a wider range of students to engage in game development.
- Empower students by fostering creativity, problem-solving skills, and critical thinking through game creation

2.3 Scope

The scope of this project encompasses the following aspects:

- **Target audience** : Youngsters aged between 12 and 18 years old.
- **Educational subjects** : Primarily focused on computer science, the console aims to foster computational thinking and programming skills among students through game development.
- **Hardware features** : The educational game console will be designed as a handheld device, resembling a real console, to enhance the gaming experience. It will be cost-effective, making it accessible to a wide range of kids, schools and institutions. The console will incorporate essential hardware components, including, motherboard, screen, push buttons, joystick, and other sensors. It will also utilize open-source programming software
- **Integration** : Collaboration with schools and educational institutions is a crucial aspect of this project. By establishing partnerships, the console can be incorporated into the existing curriculum, empowering teachers and students alike.

2.4 Requirements

Designing an educational game console based on embedded systems requires expertise in selecting hardware and software components. Careful consideration of compatibility, affordability, and functionality is crucial. The chosen hardware components should seamlessly work together and meet the console's requirements. Similarly, software frameworks and tools need to be compatible with the hardware and provide necessary features for game development and educational content delivery. Prioritizing ease of use and intuitive interfaces can facilitate the creation and customization of educational games. By carefully evaluating these factors, an engaging and educational gaming platform can be successfully implemented.

2.4.1 Software Tools

- **For Code Prototyping** A software tool for code prototyping is essential to serve as a platform for developing and testing code specifically designed for micro-controllers based on the ATmega series. Among the available options, Arduino IDE stands out as the preferred tool for this purpose.

Table 2.1: Software Tools for Code Prototyping

Software	Description
Arduino IDE	The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

- **For Embedded System Design** The design of the embedded system board, which forms the core of the console, will be created using software tools such as Fritzing, EasyEDA and Eagle. These tools provide a platform for schematic capture, PCB layout design, and component placement.

Table 2.2: Software Tools for Embedded Systems Design

Software	Description
Fritzing	Fritzing is an open-source initiative to develop amateur or hobby CAD software for the design of electronics hardware, intended to allow designers and artists to build more permanent circuits from prototypes. It was developed at the University of Applied Sciences Potsdam.
Easy Eda	EasyEDA is a web-based EDA tool suite that enables hardware engineers to design, simulate, share - publicly and privately - and discuss schematics, simulations and printed circuit boards. Other features include the creation of a bill of materials, Gerber files and pick and place files and documentary outputs in PDF, PNG and SVG formats
Eagle	EAGLE is electronic design automation (EDA) software that lets printed circuit board (PCB) designers seamlessly connect schematic diagrams, component placement, PCB routing, and comprehensive library content.

- **For Game Development**

  In the process of selecting software for game development and programming the console, a meticulous evaluation will be conducted to identify the most suitable option.

  Three potential software options have been identified: PictoBlox, MakeCode Arcade, and mBlock.

  - **PictoBlox** PictoBlox is a graphical programming software based on Scratch, where interactive games, animations, or program robots and projects can be made ref .

  - **MakeCode Arcade** The MakeCode environment is an open source online web environment that allows programming in blocks of code, using the scripting language JavaScript and Python. [7]

  - **mBlock** mBlock 3 is a user-friendly block-based programming software inspired by Scratch 2.0. It offers a seamless interface with Makeblock controller boards and compatible Arduino-based hardware, enabling users to design interactive applications with ease. The software allows users to visually program using blocks, which can be converted to Arduino C code. mBlock 3 is compatible with multiple operating systems, including macOS, Windows, Linux, and Chromebook.

Table 2.3: Factors for Software Selection

Software	**Block Based Programming**	**Boards Compatible with Available Boards Compatibility**	**Hardware Extension Availability**
PictoBlox	**Yes**	STM32 (No) ESP32 (Yes) Arduino (Yes)	- No No
MakeCode Arcade	**Yes**	STM32 (Yes) ESP32 (No) Arduino (No)	Yes - -
mBlock	**Yes**	STM32 (No) ESP32 (Yes) Arduino (Yes)	- No Yes

Several factors will be taken into consideration, including support for block-based programming, compatibility with available boards, and the availability of extensions to facilitate hardware integration.

After evaluating the options presented in the comparison table, the final decision for the software selection came down to mBlock and MakeCode Arcade. Both options showed promising features for game development and programming the console. However, further investigation revealed that certain aspects of making the STM32 microcontroller compatible with the software are still under development.Given the project's timeline and the need for feasibility, the decision was made to choose mBlock.

mBlock was selected as the software solution for the educational game console due to its support for block-based programming, compatibility with popular boards like ESP32, Arduino Uno, and Arduino Mega, and its existing capabilities in extended hardware integration.

Additionally, there is a keen interest in exploring the integration of the STM32 microcontroller with MakeCode Arcade in future research and development endeavors.

2.4.2 Hardware Components

In the upcoming sections, the hardware selection process for the console will focus on identifying and choosing the most essential components required for a game console.

These components include the motherboard, input devices such as four push buttons, a joystick, and a motion sensor; additional pins will be added for users to connect their own sensors.

Furthermore, the selection of output devices, such as the screen, the speaker and other indicators, will be discussed. These components play a crucial role in providing visual

feedback and enhancing the user experience on the console.

Additionally, the next synoptics diagram will serve as a visual aid to better understand the connections and communication pathways within the gaming console.

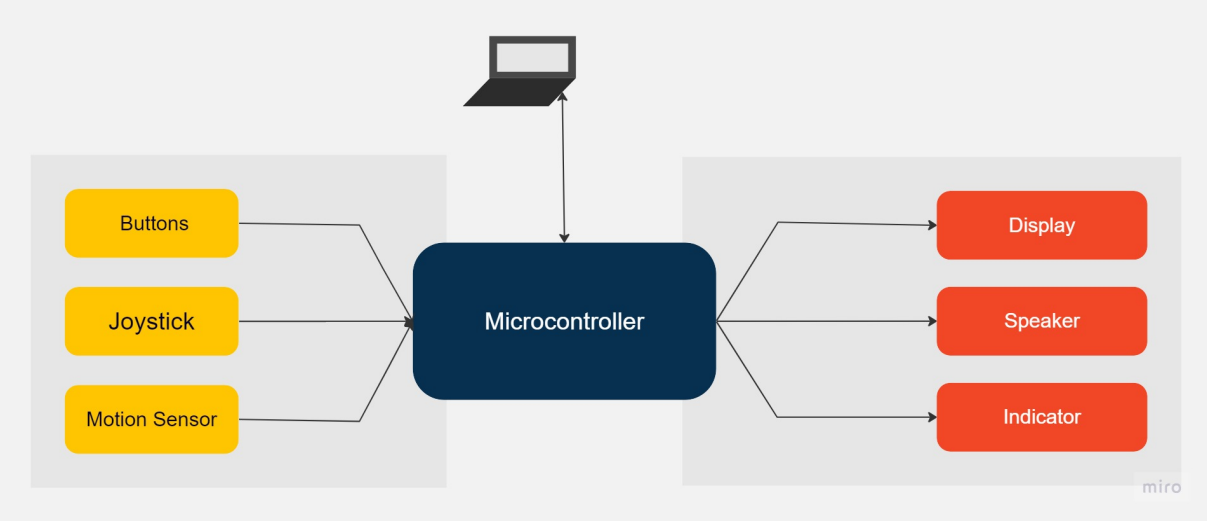


Figure 2.1: Synoptics Diagram before the Hardware Selection

During the hardware selection process, careful considerations will be given to factors like affordability, availability, and compatibility with the chosen software. These considerations will guide the decision-making process and ensure that the chosen hardware components align with the project's objectives and seamlessly integrate with the selected software.

Motherboard Selection

In the process of selecting the motherboard for the console, the compatibility with the chosen software, mBlock, was a crucial factor. While the ESP32 board is known for its high performance, it has limitations in terms of available extensions for extended hardware integration, specifically for the required inputs and outputs of the console.

Considering the need for multiple inputs and outputs, the Arduino Mega board emerged as the optimal choice. Its extensive range of digital and analog pins provides the necessary flexibility to accommodate the various components and peripherals of the console. The Arduino Mega's robust capabilities make it well-suited to handle the computational requirements and support the desired functionalities of the educational game console.

- **Arduino Mega**
  - **Description** The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega 2560 board

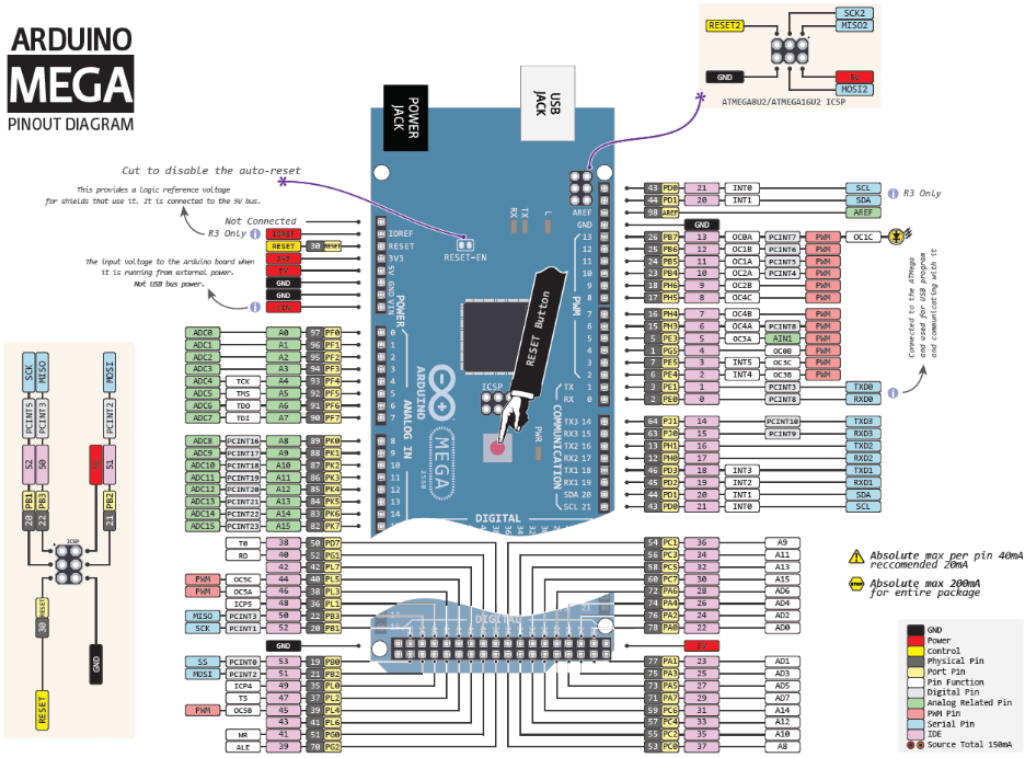


Figure 2.2: Arduino mega Pinout Diagram

– **Technical Specifications**

Table 2.4: Arduino Mega Technical Specifications

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	96-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
Led builtin	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

- **Operating Mode**

  Code Execution: Arduino Mega executes uploaded code written in Arduino programming language.

  Event-Driven Architecture: Arduino Mega responds to events or triggers.

  Input and Output Handling: Interacts with environment through input/output pins.

  Power Management: Can be powered through USB or external sources; code optimizes energy consumption.

  Libraries and API s: Supports pre-written functions for various functionalities.

  Communication: Uses different protocols to communicate with other devices.

  Error Handling: Implements mechanisms to handle errors or unexpected situations.

  Memory Management: Efficiently manages limited memory resources.

  Timing and Scheduling: Ensures precise timing and scheduling of tasks.

  Debugging and Testing: Debugs and tests code for identifying and fixing errors.

Inputs Selection

- **Joystick**

  - **Description** This joystick is strikingly similar to the one used in the PS2 (PlayStation 2). It is a self-centering spring-loaded joystick, which means that when you release it, it will center itself. It also has a comfortable cup-type knob/cap that feels like a thumb-stick.

  - **Operating Mode** The basic idea behind a joystick is to convert the stick's position on two axes — the X-axis (left to right) and the Y-axis (up and down) — into an electrical signal that a micro-controller can process. This is accomplished by incorporating two 5K potentiometers (one for each axis) connected with a gimbal mechanism that separates "horizontal" and "vertical" movements. This joystick also has a momentary push-button switch that is actuated when you press the joystick knob. It is designed in such a way that the lever operates regardless of the position of the joystick. **Reading analog values from Joystick** The joystick outputs an analog signal whose voltage varies between 0 and 5V. When you move the joystick along the X axis from one extreme to the other, the X output changes from 0 to 5V, and the same thing happens when you move it along the Y axis. And, when the joystick is centered (rest position), the output voltage is approximately half of VCC,

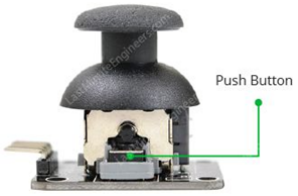


Figure 2.3: Joystick Module [8].

or 2.5V. This output voltage can be fed to an ADC on a microcontroller to determine the physical position of the joystick.

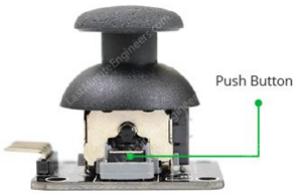


Figure 2.4: Joystick Axis [8].

Because the Arduino board has a 10-bit ADC resolution, the values on each analog channel (axis) can range from 0 to 1023. Therefore, when the joystick is moved from one extreme to the other, it will read a value between 0 and 1023 for the corresponding channel. When the joystick is centered, the vertical and horizontal channels will both read 512.

- **MPU6050 Accelerometer and Gyroscope Sensor**

  - **Description**
    At the core of the module is a low-power, low-cost 6-axis MotionTracking chip – MPU6050 – that integrates a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor (DMP) into a tiny 4mm x 4mm package. It can measure angular momentum or rotation along all three axes, static acceleration caused by gravity, and dynamic acceleration caused by motion, shock, or vibration. The module includes an on-board LD3985 3.3V regulator, so we can safely use it with a 5V logic microcontroller like Arduino.

  - **Technical Specifications**
    Here are the specifications:

Table 2.5: MPU6050 Technical Specifications

Operating Voltage	5V (typical)
Accelerometer Range	±2g, ±4g, ±8g, ±16g
Gyroscope Range	±250°/s, ±500°/s, ±1000°/s, ±2000°/s
Temperature Range	-40 to +85°C
Absolute Maximum Acceleration	Up to 10,000g

The MPU6050 consumes less than 3.6mA during measurements and only 5 A when idle. Because of its low power consumption, it can be used in battery-powered devices. Additionally, the module has a Power LED that illuminates when the module is powered on.

– **Operating mode**

* **Measuring Acceleration** The MPU6050 has an on-chip accelerometer that can measure acceleration over four programmable full scale ranges of ±2g, ±4g, ±8g, and ±16g.

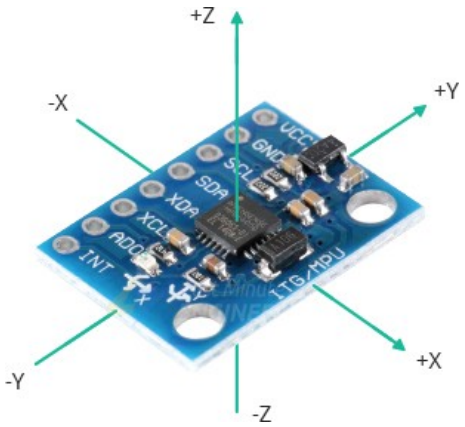


Figure 2.5: Input-Processing-Output Diagram [8] .

The MPU6050 is equipped with three 16-bit analog-to-digital converters that simultaneously sample the three axes of movement (along the X, Y, and Z axes).

* **Measuring Rotation** The MPU6050 has an on-chip gyroscope that can measure angular rotation over four programmable full scale ranges of ±250°/s, ±500°/s, ±1000°/s, and ±2000°/s.

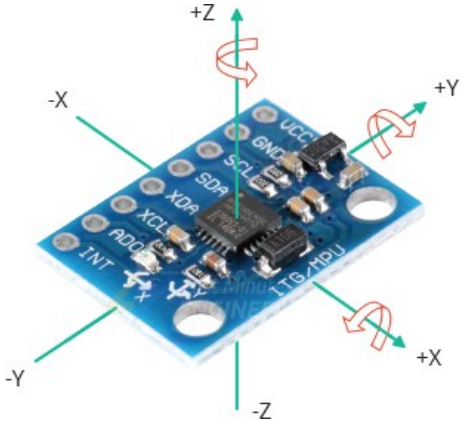


Figure 2.6: Input-Processing-Output Diagram [8] .

The MPU6050 is equipped with three more 16-bit analog-to-digital converters that simultaneously sample the three axes of rotation (along the X, Y, and Z axes). The sampling rate can be adjusted from 3.9 to 8000 samples per second.

* **Measuring Temperature** The MPU6050 includes an embedded temperature sensor that can measure temperatures from -40 to 85°C with a ±1°C accuracy. Note that this temperature measurement is of the silicon die itself, not the ambient temperature. These measurements are typically used to compensate for accelerometer and gyroscope calibration or to detect temperature changes rather than measuring absolute temperatures.

Push Buttons

- **Description**
  A push button switch is a mechanical device used to control an electrical circuit in which the operator manually presses a button to actuate an internal switching mechanism. They come in a variety of shapes, sizes, and configurations, depending on the design requirements.

- **Operating mode**
  Push button switches rely on a simple in-out actuation mechanism. They can be employed to break (off) or initiate (on) a circuit. Alternatively, they can provide an input for the user interface of a piece of equipment or start/stop a particular function. Push button switches may be categorized as being either momentary (where the switch function only continues for as long as the operator is pushing the button) or maintained (where the switch function stays latched in that status after it has been actuated).

Output Devices Selection

- **Screen**
  The screen selection process involved evaluating different options such as OLED screens, LCD screens, and LED matrix screens. After careful consideration, The LED matrix screen was selected for its grid-like structure, which allows for efficient manipulation of individual LEDs and facilitates the implementation of dynamic game elements. Two LED matrix screens of 8*8 were utilized to enhance display capabilities and provide flexibility in creating interactive and visually engaging game elements.

  – **Description**
    An 8×8 dot matrix display typically has 16 pins, 8 for each row and 8 for each column. All rows and columns are wired together in order to reduce the number of pins. If this were not the case, an 8×8 dot matrix display would require 65 pins, one for each LED and one for a common anode or common cathode connector. By connecting rows and columns, only 16 pins are needed to control the entire matrix. This technique of controlling a large number of LEDs with fewer pins is referred to as Multiplexing.

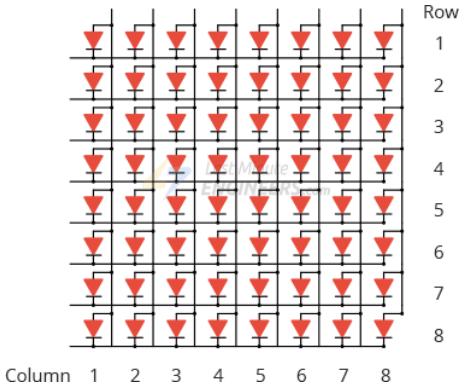


Figure 2.7: LED matrix multiplexing [8] .

In this technique, each column is activated for a very short time, and at the same time, the LEDs on that column are lit by addressing the corresponding row. As a result, no more than eight LEDs are lit at the same time. The columns are switched so fast (hundreds or thousands of times per second) that the human eye perceives the display as fully lit.

– **Operating mode**
When it comes to controlling dot-matrix displays, there is hardly a better option than the MAX7219. The only problem with multiplexing is that you have to refresh the display all the time to keep the image stable.

Then there's the MAX7219 Chip, which handles all of the control and refresh work for you. All you have to do is send it serial commands through the 4-pin SPI interface, and it will take care of the rest.



Figure 2.8: MAX7219 module [8] .

It can fully control 64 individual LEDs while keeping their brightness constant. Once the microcontroller has updated the display, the MAX7219 handles the work of refreshing the display at 800 Hz. This frees up the microcontroller to do other important things. The MAX7219 has a power saving mode in which the display can be turned off to save power. It also turns off the LEDs during startup, preventing wacky displays for the first few seconds of operation. The MAX7219 communicates via the SPI interface, so it only needs 3 data pins to connect to a microcontroller. In addition, we can daisy-chain multiple modules together for a larger display using the same 3 wires.

- **Buzzer**

  - **Description** An audio signaling device like a beeper or buzzer may be electromechanical or piezoelectric or mechanical type. The main function of this is to convert the signal from audio to sound. Generally, it is powered through DC voltage. Based on the various designs, it can generate different sounds like alarm, music, bell siren.

  - **Specifications**
    The specifications of the buzzer include the following:

    * Color is black
    * The frequency range is 3,300Hz
    * Operating Temperature ranges from – 20° C to +60°C
    * Operating voltage ranges from 3V to 24V DC
    * The sound pressure level is 85dBA or 10cm
    * The supply current is below 15mA

  - **Working Principle**
    The working principle of a buzzer depends on the theory that, once the voltage is given across a piezoelectric material, then a pressure difference is produced. A piezo type includes piezo crystals among two conductors. Once a potential disparity is given across these crystals, then they thrust one conductor drag the additional conductor through their internal property. So this continuous action will produce a sharp sound signal.

- **RGB LED**

  - **Description**
    LEDs are like tiny lightbulbs. Low energy consumption, small size, rapid switching and long lifespan makes them ideal for mobile devices and other low-power applications. LED stands for Light Emitting Diode. They are a special type of diode that convert electrical energy into light. They have very similar electrical characteristics to a normal PN junction diode. That's why the symbol of LED is similar to the normal PN junction diode except that it contains arrows pointing away from the diode indicating that light is being emitted by the diode.

  - **Working Principle**

    Like an ordinary diode, the LED operates only in forward bias condition. When the LED is forward biased, the free electrons cross the PN junction and recombine with holes. Since these electrons fall from a higher to a lower energy level, they radiate energy in the form of photons (light). In ordinary diodes, this energy is radiated as heat while in an LED, energy is radiated as light. This effect is called Electroluminescence.

2.5 Synoptics Diagram

Below is the synoptics diagram illustrating the hardware architecture of our gaming console. The diagram showcases the selected components integrated into a custom PCB (Printed Circuit Board), the software platform used for interaction with the hardware, and the communication protocols employed between these components and the Arduino Mega.

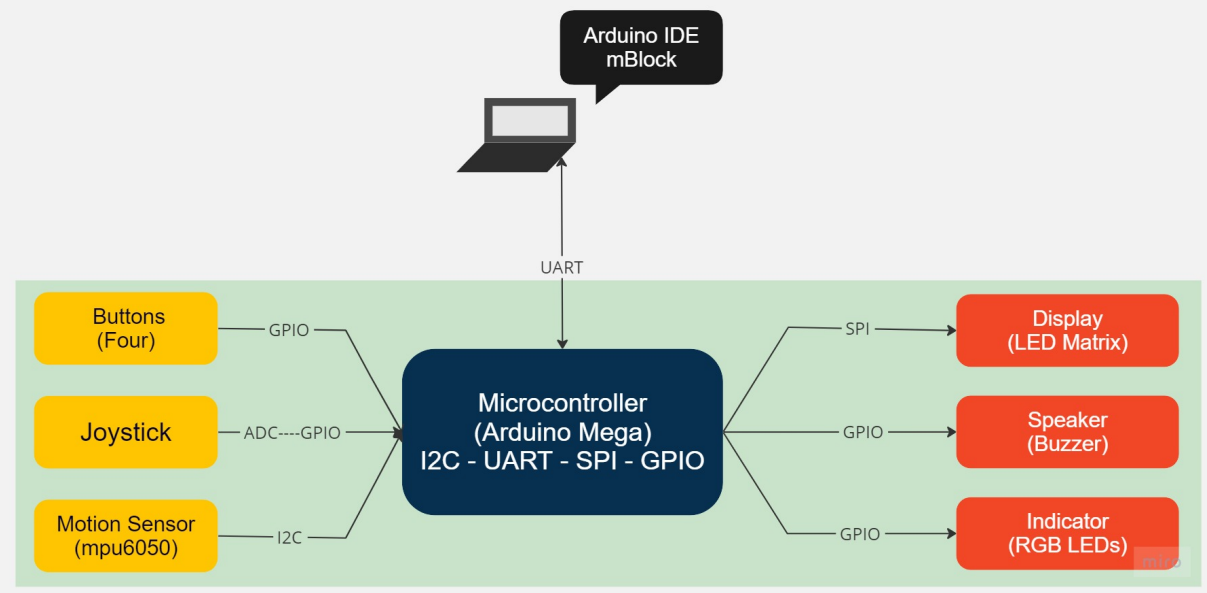


Figure 2.9: Synoptics Diagram after Selection.

Interactions and Communication Protocols:

- **Joystick:** Analog input communication with the Arduino Mega via GPIO (General Purpose Input/Output) pins.
- **Four Push Buttons:** Digital input communication with the Arduino Mega via GPIO pins.
- **MPU6050 Motion Sensor:** I2C (Inter-Integrated Circuit) communication with the Arduino Mega.
- **LED Matrix (MAX7219):** SPI (Serial Peripheral Interface) communication with the Arduino Mega.
- **Buzzer:** Digital output communication with the Arduino Mega via GPIO pins.
- **RGB LED:** Digital output communication with the Arduino Mega via GPIO pins.

2.6 Component Pricing Table

Below is a table outlining the estimated prices for the components required for the gaming console project. The prices are provided in Algerian Dinar (DZD).

Component	**Estimated Price (DZD)**
Joystick	400 DZD
Four Push Buttons	160 DZD
MPU6050 Motion Sensor	800 DZD
LED Matrix (MAX7219)	850 DZD
Buzzer	95 DZD
RGB LED	30 DZD
Custom PCB	1600 DZD
Total Estimated Cost	**3895 DZD**

2.7 Methodology

The methodology outlines a systematic approach for the development and implementation of the gaming console project. It comprises a series of well-defined steps that guide the entire process from conception to realization.

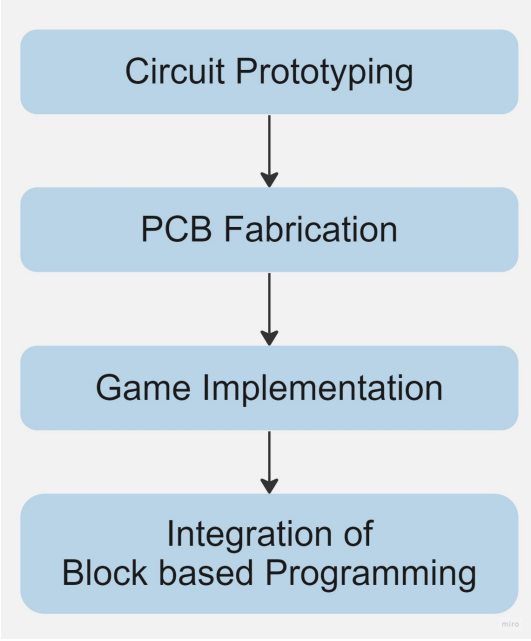


Figure 2.10: Methodology Diagram.

2.7.1 Circuit prototyping

The purpose of circuit prototyping is to validate the design concept and functionality of the electronic circuit before its implementation on a permanent PCB (Printed Circuit Board). It serves as a preliminary stage in the development process to test and verify the connections, component compatibility, and overall performance of the circuit. By prototyping the circuit on a breadboard or similar platform like Fritzing, developers can assess its feasibility, identify potential issues or errors, and make necessary adjustments before moving on to the fabrication stage. Circuit prototyping significantly reduces the risk of costly mistakes and allows for iterative improvements, ensuring a more robust and optimized final design.

- **Component Identification**

  Identify and gather the necessary electronic components required for your circuit design.

- **Breadboard Design** Make the initial wiring connection for the gaming console components

- **Functional and Compatibility Testing**

  Power up the circuit and perform functional tests for each component and ensure accurate outputs.

2.7.2 PCB Fabrication

PCB fabrication is a key aspect of embedded system engineering, showcasing the engineer's skills in transforming circuit designs into functional devices. In this section, the steps involved in PCB fabrication are outlined, from circuit schematic design to component soldering. the following section explores the process in two methods :

- **Chemical Etching Method**

  - Design the circuit schematic and layout using Easy EDA or Eagle
  - Transfer the circuit design onto a copper-clad board using a printed transparency and a UV light source.
  - Develop the board by removing the unexposed photoresist, leaving behind the circuit pattern.
  - Etch the board in a chemical solution (such as ferric chloride or ammonium persulfate) to remove the exposed copper, leaving the circuit traces intact.
  - Clean and neutralize the board using a solution of water and sodium bicarbonate to remove any remaining photoresist and etchant.

  - Drill holes in the board at the designated component connection points using a drill bit or a rotary tool.
  - Test the continuity of the circuit traces using an ohmmeter and mark any potential micro-cuts for repair.
  - Prepare the components by bending their leads and sorting them in the order of soldering.
  - Use a soldering iron and solder (composed of tin and lead or lead-free alternatives) to solder the components onto the board, starting with the smaller ones and progressing to larger ones.
  - Ensure proper alignment and polarization of components and solder the connection wires.[9]

- **CNC Milling Method**

  - Design the circuit schematic and layout, then transfer it to a CNC software.
  - Set up the CNC milling machine and connect it to a computer.
  - Secure the copper-clad board on the machine's work surface.
  - Use the CNC milling machine with a milling bit to carve out the circuit traces and drill holes according to the design.
  - Once the milling is complete, clean the board to remove any debris.
  - Tin the exposed copper traces by applying solder to them.
  - Mount and solder the components onto the board, following the designated layout.
  - Finish the board by applying an additional tin coating to prevent oxidation[9]

2.7.3 Game Implementation

Develop the user-created game using Arduino IDE. This includes:

- Design the game mechanics, graphics, and user interface
- Implement the game logic and interactions by programming the Arduino Mega using the Arduino IDE
- Testing and debugging the game for functionality and performance.

2.7.4 Integration of Block Based Programming

To integrate block-based programming in embedded systems using extensions, these necessary steps need to be followed:

- Select a block-based programming platform compatible with your embedded system requirements.
- Install and configure the extensions in the programming environment.
- Utilize extension blocks to control hardware peripherals, access sensors, or implement communication protocols.
- Design and develop the desired functionality by dragging and dropping extension blocks.
- Compile the block-based program and generate the necessary code or firmware.
- Upload the compiled program to the embedded system for execution and testing.
- Debug and refine the program as needed, making adjustments to the extension blocks or configurations.[10]

By following these steps, block-based programming can be seamlessly integrated along with extensions into embedded systems. This integration enables enhanced functionality and simplifies the development process.

2.8 Conclusion

In this chapter, we introduced the project framework for developing an educational game console using embedded systems. The primary objectives were to create an interactive and enjoyable learning experience, empower students through game creation, and foster computational thinking and programming skills.

We discussed the synoptics of the project, including the target audience, educational subjects, and hardware features of the console. Collaboration with schools and educational institutions was highlighted as an important aspect of the project.

The software tools for embedded system design were presented, including Fritzing, EasyEDA, and Eagle for circuit design. The selection of the IDE for game development and programming the console involved evaluating options such as PictoBlox, MakeCode Arcade, and mBlock. mBlock was chosen for its support for block-based programming and compatibility with popular boards.

The hardware selection process focused on choosing components like the motherboard, input devices (joystick, push buttons, mpu6050), and output devices (LED matrix screen, buzzer, RGB LED). The Arduino Mega board was selected as the motherboard due to its compatibility and flexibility.

In the next chapter, we will move into the implementation phase, where we will explore the step-by-step process of developing the educational game console and creating user-designed games using both Arduino IDE and block-based programming.

Chapter 3

Implementations

3.1 Hardware Integration and Prototyping

During circuit prototyping, the identified components, including Arduino Mega 2560, cascaded LED matrices with MAX7219 encoders, joystick, MPU6050, four push buttons, buzzer, and RGB LED, were connected on a breadboard. Fritzing, a popular circuit design software, was used to create a visual representation of the circuit connections, here are the connections used to ensure accuracy and functionality.

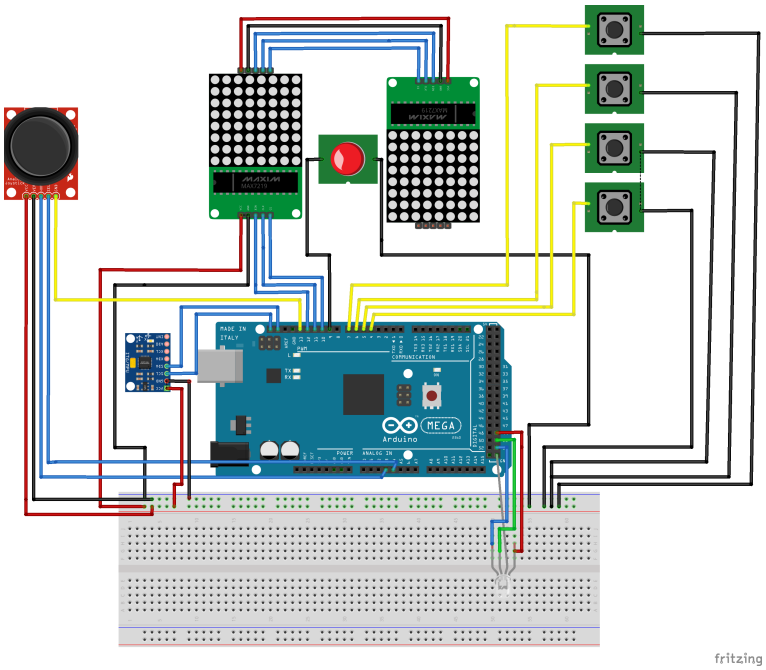


Figure 3.1: Breadboard Design using Fritzing.

3.1.1 Arduino Mega 2560

- The LED matrices were connected to the Arduino using SPI communication protocol. The MAX7219 encoders were connected to the digital pins of the Arduino (CLK to Pin 13, DIN to Pin 11, and CS to Pin 10).
- The Joystick was connected to the analog pins of the Arduino (X-axis to A0, Y-axis to A1, and the button to Pin 2).
- The MPU6050 sensor was connected to the Arduino using the I2C interface (SDA to SDA and SCL to SCL pins).
- The four push buttons were connected to digital pins (left Button 1 to Pin 4, Right Button to Pin 5, Top Button to Pin 6, and Down Button to Pin 7).
- The buzzer was connected to a digital pin (Pin 9) to generate sound effects.
- The RGB LED was connected to the digital pins for controlling the individual color channels (Red to Pin 49, Green to Pin 51, Blue to Pin 53).

3.1.2 LED Matrices with MAX7219

The cascading of the LED matrices involved connecting the CLK, DIN, and CS pins of the first matrix to the corresponding pins of the second matrix. Note: The above connections were made during the circuit prototyping phase and were later adapted and modified in the PCB design for improved performance and layout. A functionality test was conducted by implementing a snake game code controlled by buttons, a joystick, or the MPU6050 sensor using the Arduino IDE. This test ensured the proper integration of components and their interaction. One challenge faced was effectively utilizing the MPU6050 sensor's output data for console direction. To address this challenge, a Kalman filter was implemented to improve the accuracy and usefulness of the MPU6050's output in determining the console's direction. Despite this progress, another challenge arose in translating the outputted directions into inputs for the snake game, which remained unresolved during this phase. Further exploration and refinement are required to overcome this challenge and fully integrate the MPU6050 sensor into the snake game.

3.2 PCB Development

3.2.1 Single Layer PCB Development and Testing

The circuit prototyping phase was followed by the development and testing of a single-layer PCB for the Arduino Mega Shield at the Electronics Lab of the University of Sciences and Technologies Houari Boumediene. The objective was to create a compact and reliable circuit layout. The following steps were undertaken:

Chemical Etching Method

- Design the circuit schematic and layout using software tools such as EasyEDA or Eagle.

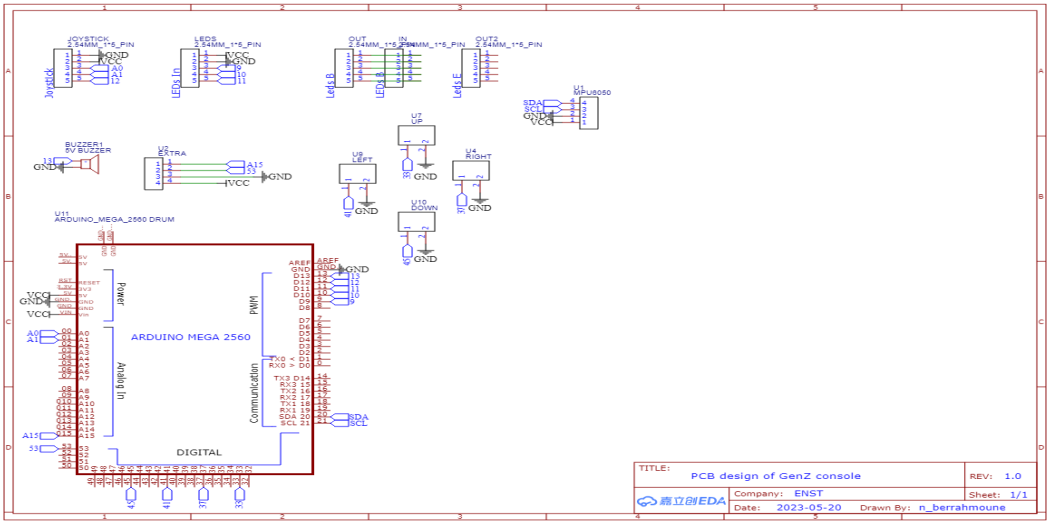


Figure 3.2: PCB Schematic using EasyEDA

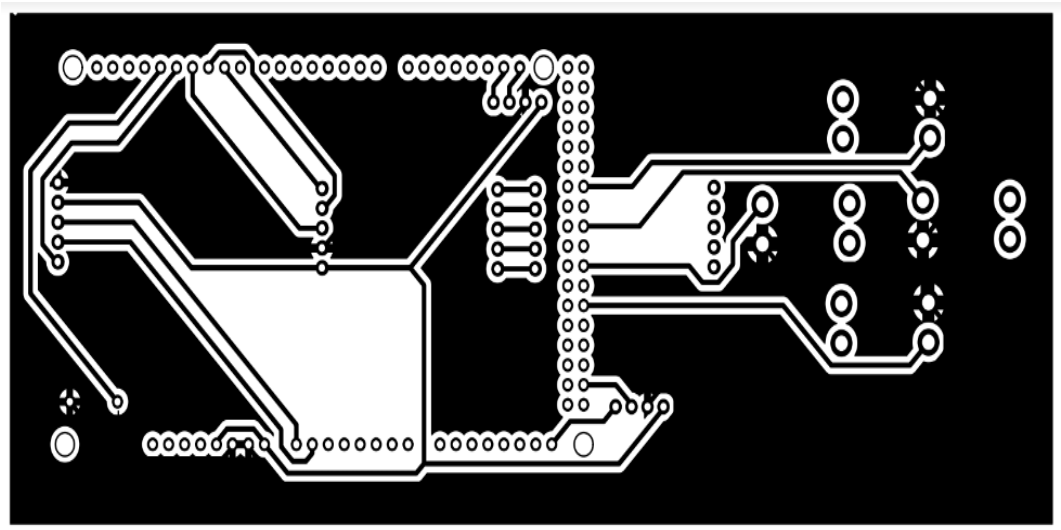


Figure 3.3: PCB Layout using EasyEDA

- Transfer the circuit design onto a copper-clad board using a printed transparency and a UV light source.

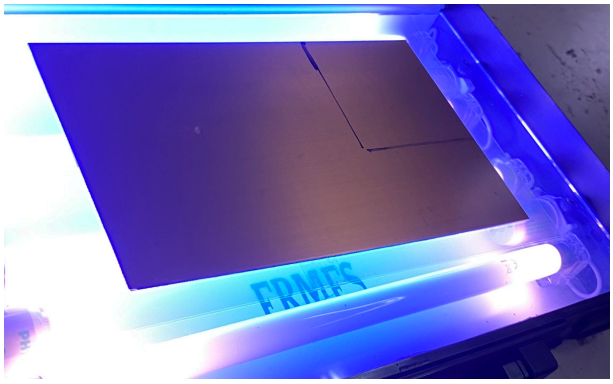


Figure 3.4: Exposing the Photoresist to UV Light

- Develop the board by removing the unexposed photoresist, leaving behind the circuit pattern.
- Etch the board in ferric chloride to remove the exposed copper, leaving the circuit traces intact.
- Clean and neutralize the board using a solution of water and sodium bicarbonate to remove any remaining photoresist and etchant.
- Drill holes in the board at the designated component connection points using a drill bit
- After testing the continuity of the circuit traces using an ohmmeter and marking any potential micro-cuts for repair, the next step in the PCB development process is soldering. Soldering involves the careful and precise placement of electronic components onto the PCB and securing them in place using solder. This ensures proper

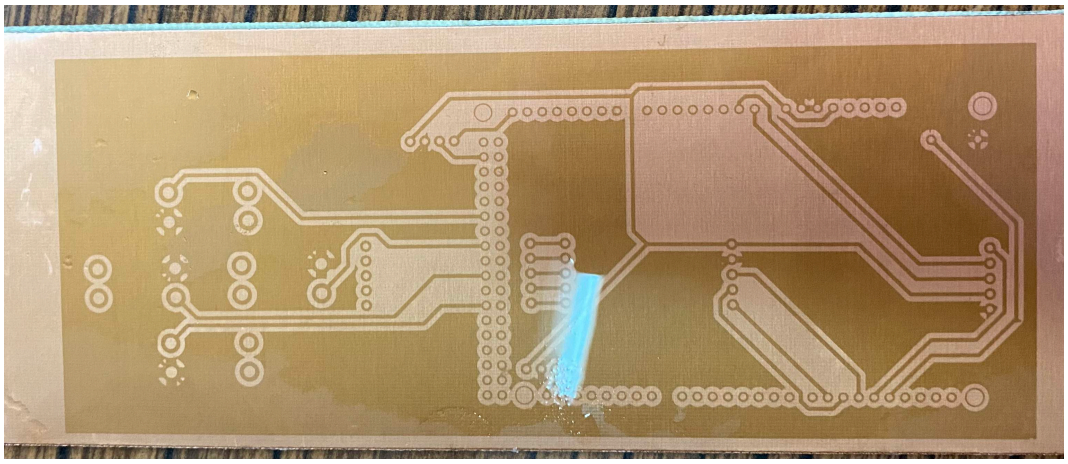


Figure 3.5: Developed Circuit Pattern After Treatment

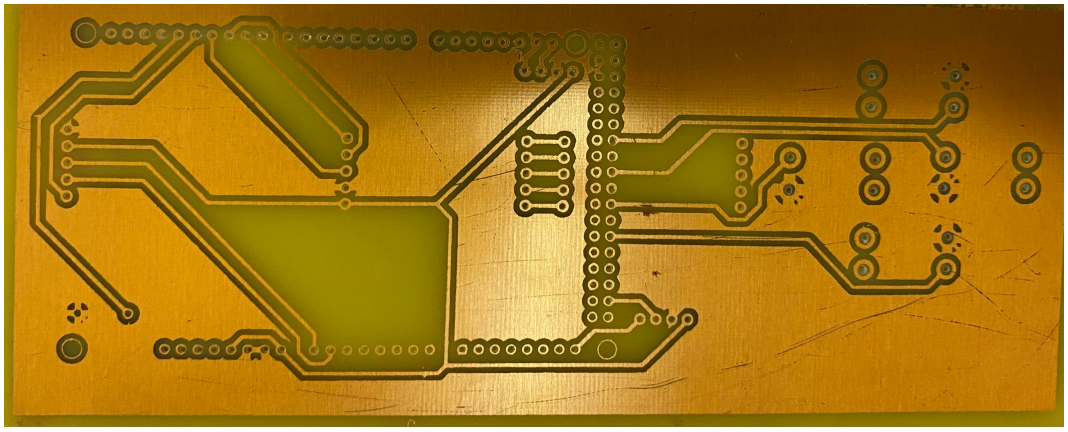


Figure 3.6: Etched PCB Surface After Etching Process

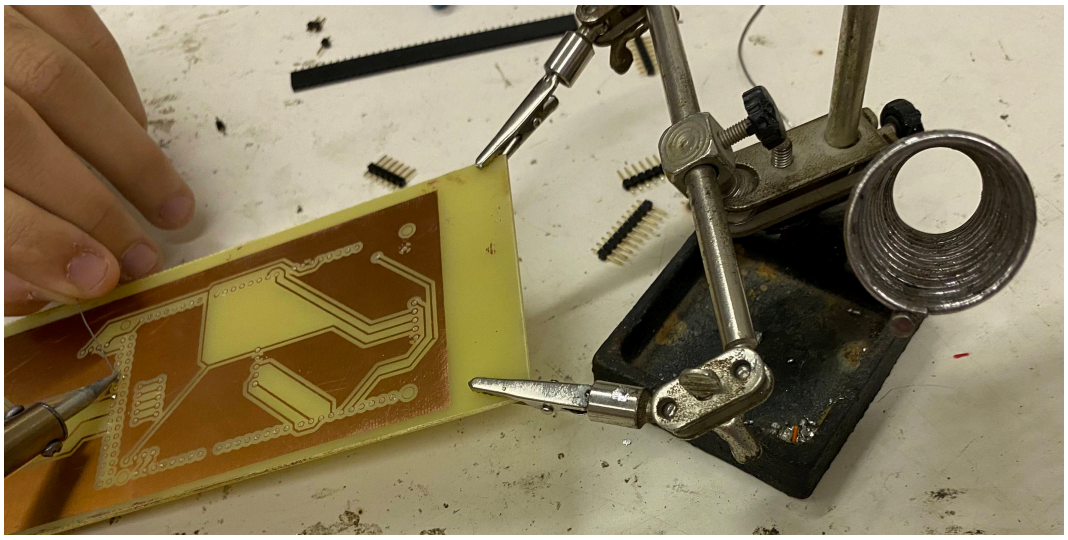


Figure 3.7: Drilled Holes for Components

electrical connections and mechanical stability. The soldering process requires skill and attention to detail to avoid any solder bridges or cold joints that can lead to faulty connections.

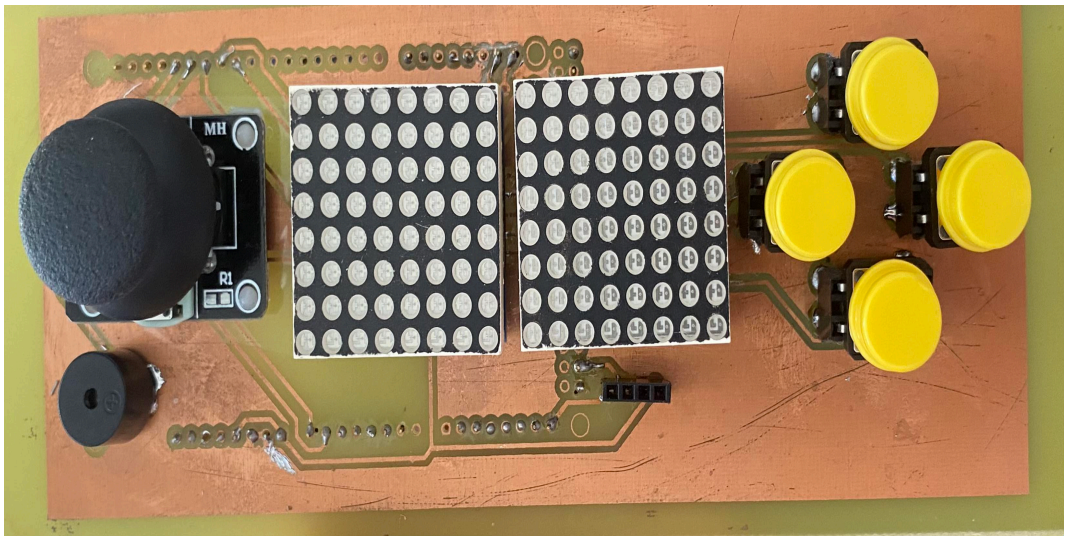


Figure 3.8: Complete One Layer PCB After Soldering

Unfortunately, after soldering all the components onto the PCB, it was realized that there was a problem with mirroring the components. This issue arose due to the limitation of using a single-layer PCB design. The mirroring problem resulted in incorrect orientations of the first LED matrix, affecting the overall functionality of the circuit. This discovery emphasized the need to adapt and transition to a 2-layer PCB design in order to overcome such limitations and ensure proper component placement and connectivity.

3.2.2 Two-Layer PCB development

In the process of developing the PCB for the Arduino Mega Shield, the next step involved adapting to a two-layer PCB design using the software Eagle. This transition was prompted by the limitations of the single-layer PCB design, which affected component mirroring and overall functionality. To overcome these challenges and ensure proper component placement and connectivity, a two-layer PCB design was deemed necessary. Eagle's offline capabilities facilitated efficient layout modifications and circuit optimization.

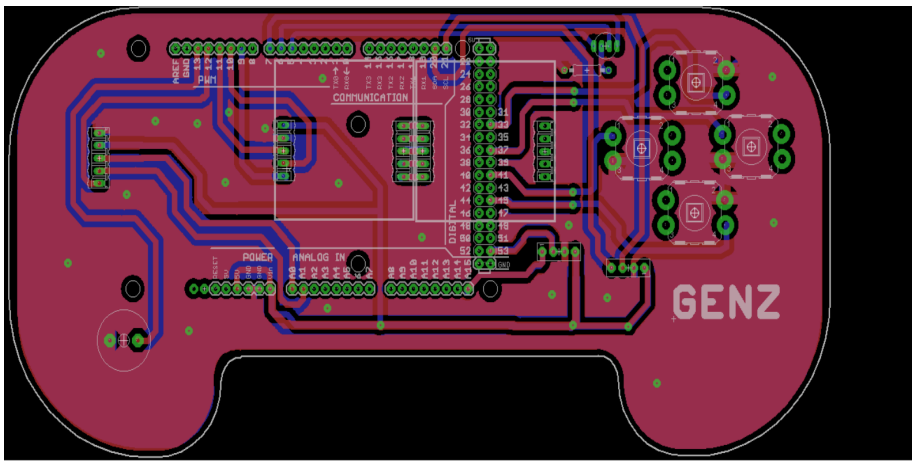


Figure 3.9: Two-Layer PCB Layout using Eagle

Once the design was finalized, it was sent to a local enterprise specialized in PCB fabrication that follows the CNC milling method . The local enterprise employed a series of well-defined steps to produce the PCB, ensuring its quality and functionality. These steps were as follows:

- Board cutting: Using a cutting tool to shape the PCB. CNC drilling and routing: Utilizing computer-controlled machinery for precise hole drilling and routing of traces. Board brushing and cleaning: Removing any debris or contaminants from the PCB surface.
- Chemical metallization of the holes: Applying a chemical process to deposit a conductive layer inside the drilled holes.
- Lamination with a photoresist film: Applying a photosensitive film to protect certain areas of the PCB during the etching process.
- UV exposure in a UV light box: Exposing the PCB to ultraviolet light to cure the photoresist film and create a pattern.
- Development and chemical etching: Immersing the PCB in a chemical solution to remove the exposed copper and create circuit traces.
- Board brushing and cleaning: Performing additional brushing and cleaning to remove any remaining chemicals and residues.
- Tinning to protect the copper with chemical tin: Applying a layer of chemical tin to protect the exposed copper surfaces.
- Application of solder mask or ink film for protection: Coating the PCB with a solder mask or ink film to protect the circuitry and provide insulation.

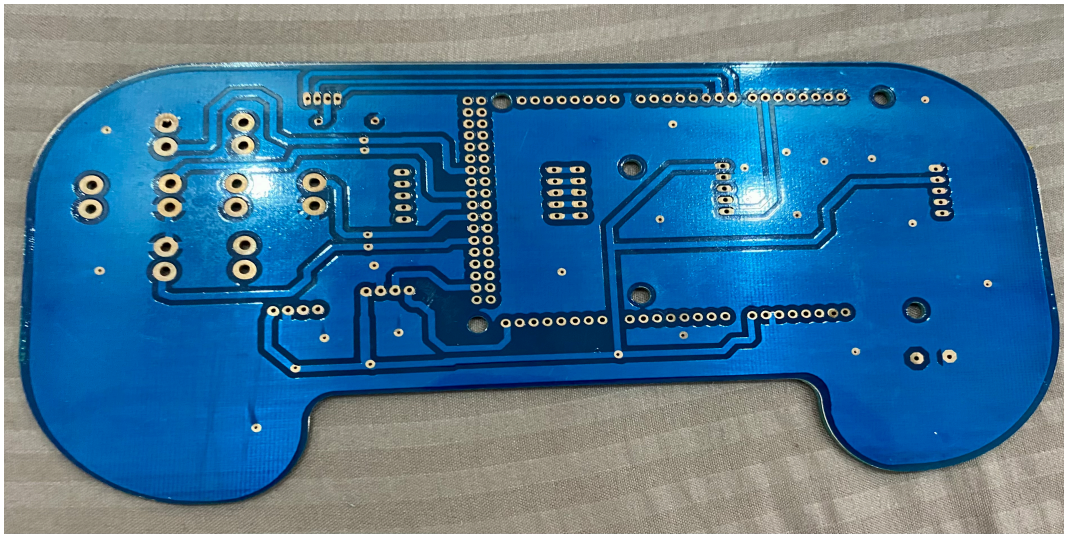


Figure 3.10: Top View of Manufactured Two-Layer PCB

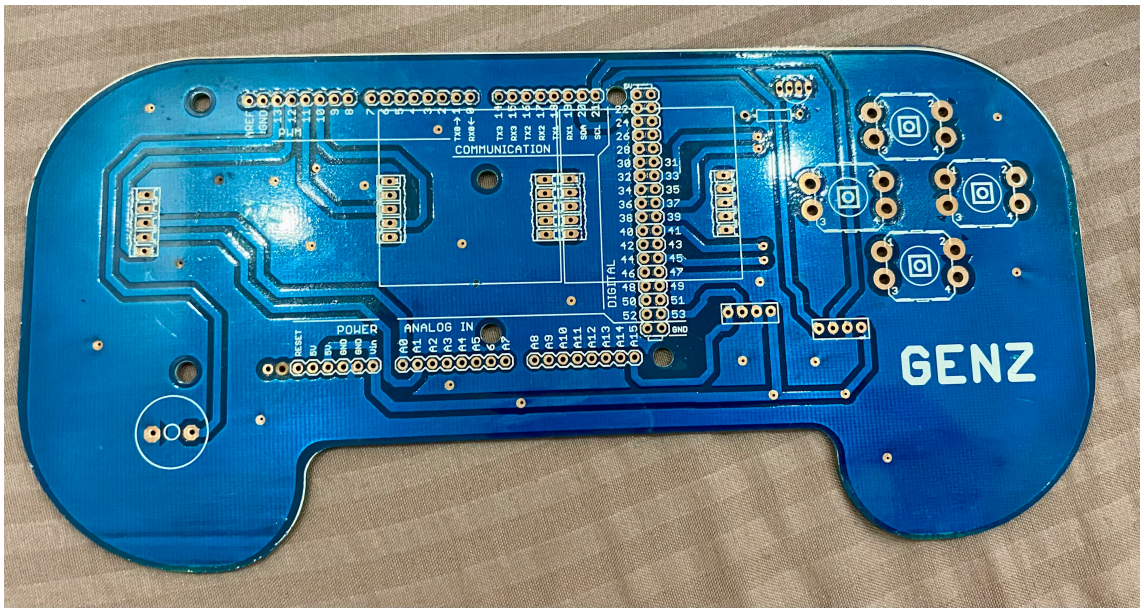


Figure 3.11: Bottom View of Manufactured Two-Layer PCB

Following the completion of the PCB production process, the fabricated board was inspected to verify its integrity and quality. The next step in the implementation phase involved the assembly and soldering of electronic components onto the PCB. This meticulous process required careful placement and alignment of each component according to the PCB layout and schematic diagram. Soldering techniques were employed to establish secure electrical connections between the components and the PCB traces. It was crucial to adhere to industry standards and best practices to avoid solder bridges, cold joints, or any other soldering defects that could compromise the functionality of the circuit. Once all the components were successfully soldered, a comprehensive inspection was conducted to ensure proper solder joints and electrical connections.

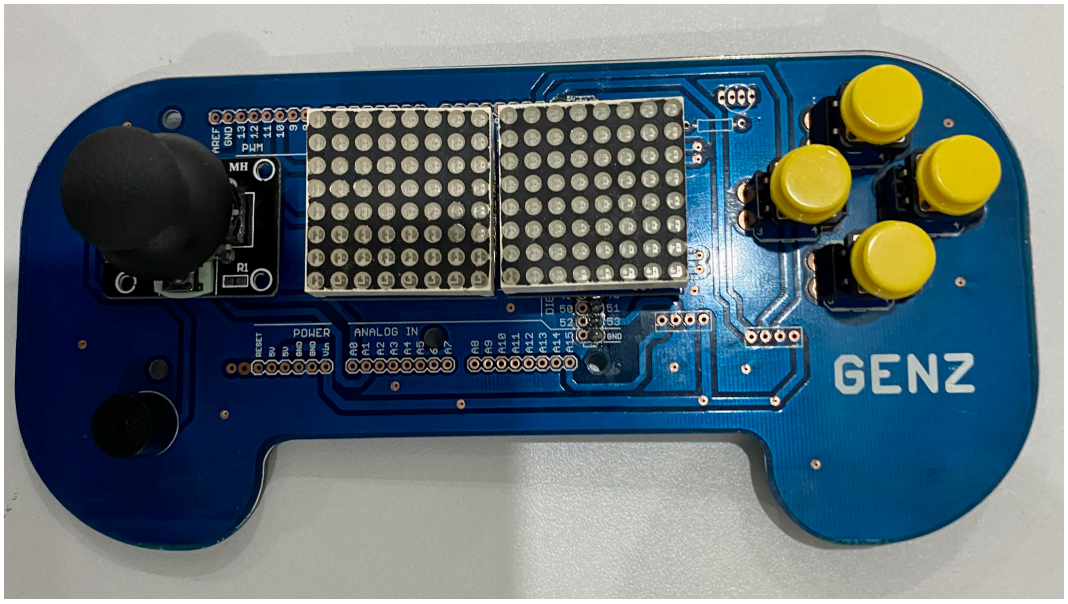


Figure 3.12: Top View of Completed PCB after Soldering

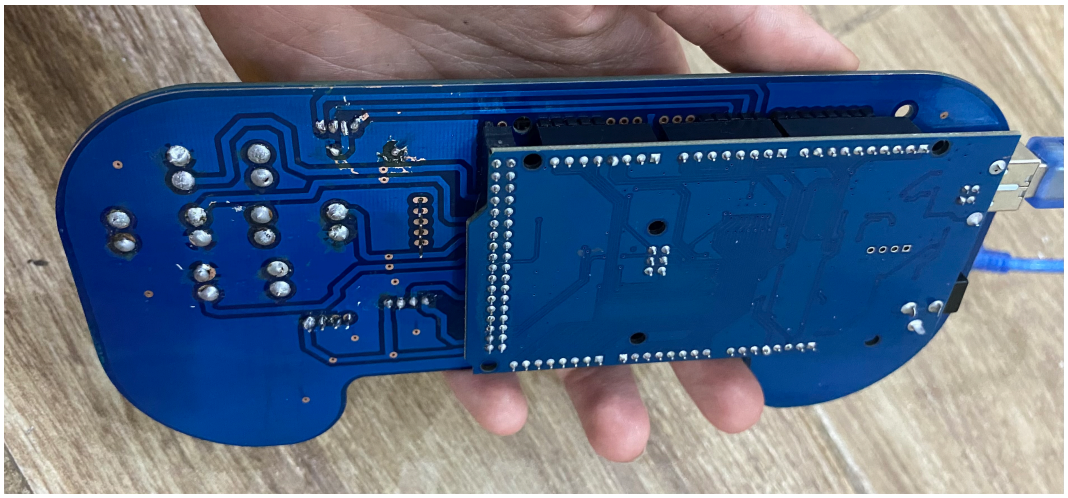


Figure 3.13: Bottom View of Completed PCB after Soldering

With the successful completion of the assembly and soldering process, the PCB was ready for further testing and integration with Arduino IDE and Block Based Programming. This marked a significant milestone in the implementation phase, bringing the project closer to its final stages.

3.3 Functionality Test

During the Functionality Test, each component of the gaming console was individually examined to ensure seamless operation and seamless integration. Before proceeding with the individual component tests, a preliminary step was taken to ensure proper conductivity and connections. This was achieved by using a multi-meter to verify the connections based on the schematics used in the coding phase. The testing phase was performed by following these steps

LED Matrix Test

- Display various patterns and characters on the LED matrix.
- Verify that all LED s are functioning correctly and properly connected to the Arduino Mega board.

Joystick Test

- Test directional input by reading and interpreting analog values from the joystick.
- Test button press functionality of the joystick.

Buttons Test

- Test the responsiveness and accuracy of the console buttons.

- Monitor each button press to ensure that the corresponding actions are correctly registered.

MPU6050 Test

- Validate the gyroscope and accelerometer data readings from the MPU6050 sensor.
- Test the console's ability to detect and respond to tilting and shaking movements.

Buzzer Test

- Play various tones and melodies using the buzzer.
- Verify that the sound output is functioning correctly and can provide auditory feedback during gameplay.

RGB Test

- Test the RGB LED's ability to display different colors.
- Cycle through various color combinations to ensure correct wiring and visual effects in the games.

Each component's functionality is tested individually and thoroughly. The corresponding images showcase the testing process for visual reference.

Challenges Encountered During Testing

During the testing phase, several challenges were encountered:

- The joystick module was affected by the heating during soldering, which necessitated its replacement after troubleshooting.
- Proper calibration of the MPU6050 sensor was required to ensure accurate readings.
- An error in placing the RGB LED affected the display of the green light, which was resolved by re-soldering it in its correct position.
- The buzzer emitted a beep during code transfer, which was later understood to be a useful indication and not an issue.

These challenges were addressed promptly to ensure the proper functioning of the educational gaming console. For a comprehensive collection of all the individual component test codes for the gaming console, please refer to the following hyperlink: Gaming Console Component Test Codes. With the hardware components validated and functioning correctly, the next step involved preparing the consolidated code for hardware interfacing. The following section, "Setting up the Console," will detail the necessary code and configurations to facilitate the integration of components and serve as a template for implementing various games.

3.4 Game Development

3.4.1 Game Development Setup

Before implementing any game, it is crucial to set up the hardware interfacing and libraries to ensure a smooth and efficient development process. This code snippet serves as a comprehensive template that showcases the interfacing of all components in the console, including two cascaded LED matrices, a joystick module, four push buttons, an MPU6050 accelerometer, an RGB LED, and a buzzer.

The required libraries for each component are included:

```
// Include the required libraries
#include <LedControl.h>  // Library for driving LED matrices
#include <Wire.h>        // Library for I2C communication
#include <MPU6050.h>     // Library for MPU6050 accelerometer and gyroscope
#include <Tone.h>        // Library for using the buzzer
```

Next, constants are defined for each pin used in the hardware setup:

```
// Constants for LED Matrices
const int dataPin = 11;     // Data pin for LED Matrix
const int clkPin = 9;       // Clock pin for LED Matrix
const int csPin = 10;       // Chip Select pin for LED Matrix
const int numMatrices = 2;  // Number of cascaded LED Matrices
// Constants for Joystick Module
const int joyXPin = A0;     // Joystick X-axis analog input pin
const int joyYPin = A1;     // Joystick Y-axis analog input pin
const int joyButtonPin = 12;  // Joystick push button pin
// Constants for Push Buttons
const int buttonUpPin = 33;
const int buttonDownPin = 45;
const int buttonRightPin = 37;
const int buttonLeftPin = 41;
// Constants for MPU6050
const int mpu6050Address = 0x68;  // I2C address of MPU6050
// Constants for RGB LED
const int redPin = 5;
const int greenPin = 6;
const int bluePin = 7;
// Constants for Buzzer
const int buzzerPin = 13;
```

Variables are defined to store the joystick values, its button state. and the states of the four push buttons

```
// Define variables to store joystick values
int joyXValue, joyYValue;
bool joyButtonState;
// Define variables to store button states
bool button1State, button2State, button3State, button4State;
```

LedControl library is initialized with the specified data ,clock ,chip select pins and the number of cascaded LED matrices.

```
// Initialize the LedControl library with the defined pins
LedControl lc = LedControl(dataPin, clockPin, csPin, numMatrices);
```

Function prototypes for the setup of each component are defined:

```
// Function to read joystick values
void readJoystick() {
  joyXValue = analogRead(joyXPin);
  joyYValue = analogRead(joyYPin);
  joyButtonState = digitalRead(joyButtonPin);}
// Function to read button states
void readButtons() {
  button1State = digitalRead(button1Pin);
  button2State = digitalRead(button2Pin);
  button3State = digitalRead(button3Pin);
  button4State = digitalRead(button4Pin);}
// Function to read MPU6050 data
void readMPU6050() {
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);}
// Function to control the RGB LED
void setRGBColor(int red, int green, int blue) {
  analogWrite(redPin, red);
  analogWrite(greenPin, green);
  analogWrite(bluePin, blue);}
// Function to play a tone on the buzzer
void playTone(int frequency, int duration) {
  tone(buzzerPin, frequency, duration);}
```

In the **setup()** function, we initialize the Serial communication. Then, we call the **initializeComponents()** function to set up all the hardware components.

```
void setup() {
  // Initialize Serial communication (if needed)
  Serial.begin(9600);

  // Call the function to initialize all components
  initializeComponents();

  // Add any additional setup code here...
}
```

Finally, in the **loop()** function, examples demonstrate reading input from the hardware components and controlling them. However, users should implement their specific game logic within this function.

```
void loop() {
  // Read values from components
  readJoystick();
  readButtons();
  readMPU6050();

  // Add your game logic and functionality code here...

  // Example: Display something on the LED matrices
  lc.setLed(0, 0, 0, true); // Turn on the LED at position (0, 0) on the first matrix
  lc.setLed(1, 7, 7, true); // Turn on the LED at position (7, 7) on the second matrix

  // Example: Control the RGB LED
  setRGBColor(255, 0, 0); // Set the color to red

  // Example: Play a tone on the buzzer
  playTone(1000, 100); // Play a 1000Hz tone for 100 milliseconds

  // Add any additional code for your game...

  // End of loop
}
```

With this code template, learners have a solid foundation for developing various games for the console, taking full advantage of its hardware capabilities and features. Now, let's dive into the actual implementation of exciting games using this versatile setup.

3.4.2 Game Development Constraints on the Console

To achieve successful game development on the console, it is essential to consider the constraints imposed by the hardware. Understanding these limitations will help users to make informed decisions during the game creation process, ensuring optimal performance and a seamless learning experience. The primary constraints to be aware of are as follows:

- **Display Compatibility**:
  - The game should be compatible with two cascaded LED matrices for visuals.
  - Adjust graphics to fit the LED matrix format.
- **Input Controls**
  - Utilize the joystick module with analog X and Y axes for character movement.
  - The joystick's push button should act as an action button.
  - Support at least four additional push buttons for control.
- **Motion Sensing**
  - Optionally use the MPU6050 accelerometer and gyroscope for motion sensing.
  - Integrate motion features to enhance game-play.

- **RGB LED Feedback**
  - Provide visual feedback and cues using the RGB LED.
  - Use indicators for in-game events, health, points, etc.
- **Buzzer Integration**
  - Use the buzzer for audio feedback, sound effects, or background music.
  - Explore different tones and sounds with the buzzer.
- **Hardware Constraints**
  - Optimize the game to perform smoothly within hardware limitations.
  - Balance graphics, audio, and game-play with hardware capabilities.
- **Input Flexibility**
  - Design the game to accommodate various play-styles (e.g., joystick, buttons, motion).
  - Ensure all input mechanisms are well-calibrated and responsive.
- **Game Content and Genre**
  - Choose games that suit available hardware components and capabilities.
  - Avoid games that require complex controls or high-definition graphics.
- **User Experience**
  - Prioritize an enjoyable and intuitive user experience.
  - Gather user feedback and perform play-testing for improvements.
- **Legal and Ethical Considerations**:
  - Ensure the right to adapt and use the game on your console.
  - Respect original developer's rights and avoid any unethical content.

3.4.3 Potential Games

With the robust code template and the considered constraints for hardware and development, the possibilities for game development on the console are boundless. By providing customized lesson plans in collaboration with professional institutions, learners can unlock their creativity and acquire the necessary skills to create a plethora of captivating games and even explore beyond. Some of the games they will be able to develop include:

- **Pong Game:** A classic arcade game where two players control paddles with the joystick to hit the ball back and forth. The first player to miss the ball loses a point.
- **Dinosaur Game (Infinite Runner):** An endless runner game where players use the joystick to control a dinosaur, making it jump over obstacles and collect points as it runs.
- **Snake Game:** Control a snake using the joystick to eat food and grow longer while avoiding collisions with walls and the snake's own tail.
- **Flappy Bird Clone:** A side-scrolling game where players tap the push buttons to navigate a bird through narrow gaps between pipes, earning points for each successful pass.
- **Space Invaders:** Defend the earth from descending waves of alien invaders. Players can move the spaceship horizontally with the joystick and shoot lasers to destroy the invaders.

3.4.4 Snake Game Implementation

This section covers the implementation of the classic Snake Game using two cascaded LED matrices for visuals and a joystick module or four push buttons for controlling the snake's movement. The objective is to guide the snake to eat apples, grow longer, and avoid collisions. The following steps outline the process:

- **Data Representation**: Represent the Snake as a series of (row, column) coordinates and the Apple as a single (row, column) coordinate.
- **Game Board**: Use an LED matrix or another suitable platform to represent the game board.
- **Initial State**: Set up the initial state of the game, including the starting position of the Snake's head, its initial length, and a randomly positioned Apple.
- **Game Loop**: Create a loop that continuously updates the game state and renders it on the LED matrix or chosen platform. Handle user input to change the Snake's direction.
- **Collision Detection**: Implement collision detection to check if the Snake hits the borders or collides with itself. Take appropriate actions when collisions occur, such as ending the game or resetting the Snake.

- **Apple Consumption**: Design the logic for the Snake to grow when it consumes an Apple. Increase the Snake's length and generate a new random position for the Apple.
- **Timing and Speed**: Manage the timing and speed of the game. Decide how often the game should update and move the Snake to control the speed and difficulty.
- **Sound Effects (Optional)**: Optionally, add sound effects for events like Snake consuming an Apple or the game ending.
- **Testing and Debugging**: Thoroughly test the implementation and fix any issues or bugs encountered. Test edge cases to ensure proper functionality.
- **Optimization (Optional)**: Optionally, optimize the code for better performance without altering functionality.

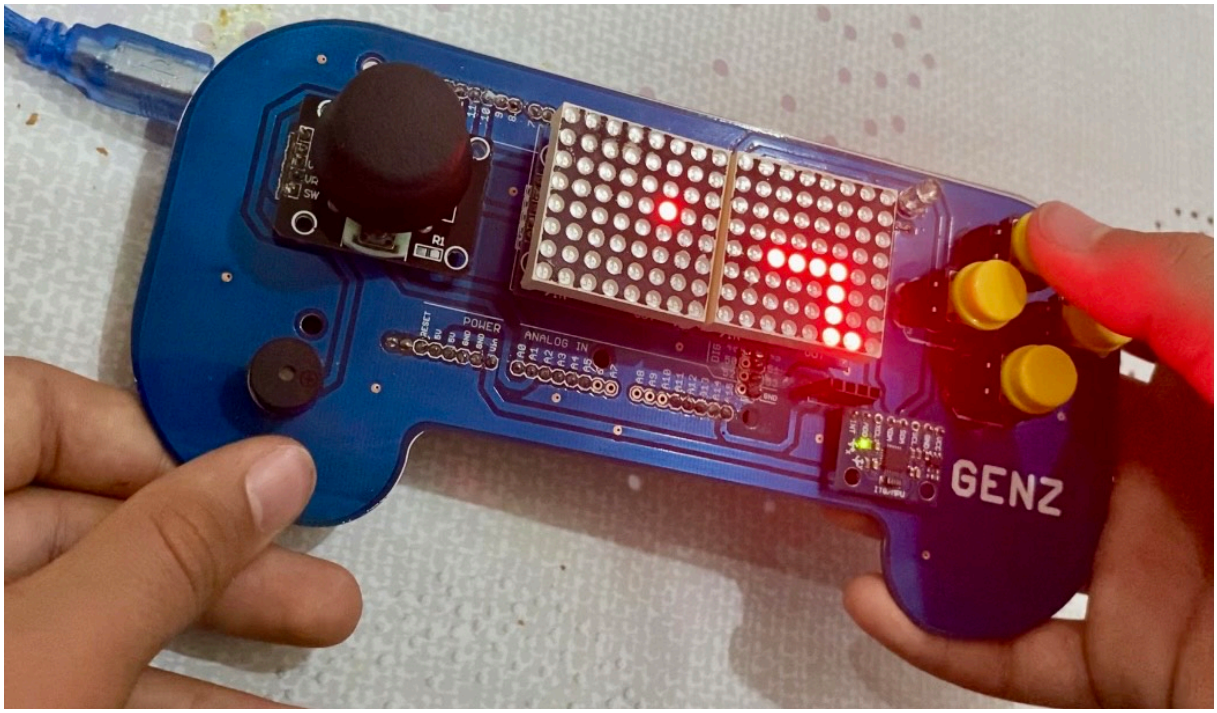


Figure 3.14: Snake Game

For the complete code of the Snake Game implementation, click here

3.5 Compatibility with Block-Based Programming

The developed gaming console is successfully integrated with block-based programming languages such as mBlock, providing an accessible and user-friendly platform for students in the target age group. The console's hardware and software work harmoniously, enabling students to program and interact with the console using a visual programming interface.

To program the gaming console using mBlock, follow these steps:

1. Connect the gaming console board to your computer using a USB cable.
2. Open the mBlock software on your computer.
3. Select the appropriate board from the "Boards" menu in mBlock (e.g., Arduino Mega).



Figure 3.15: Board Selection in mBlock

4. Choose the correct serial port that corresponds to the connected board in the "Connect" menu.



Figure 3.16: Serial Port Selection in mBlock

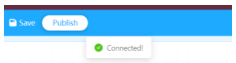


Figure 3.17: Successfully Connected

5. Write your code using the block-based programming interface in mBlock.
6. Click on the ”Upload” button to transfer the code to the gaming console board.

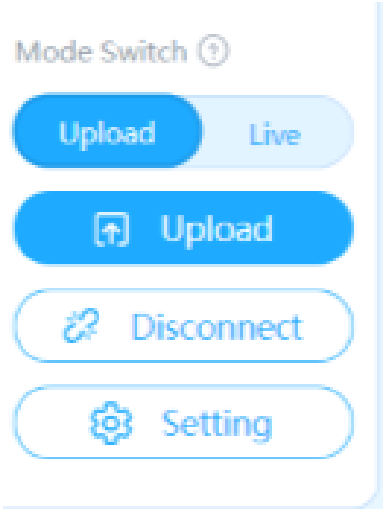


Figure 3.18: Upload the program in mBlock

7. Interact with the gaming console and enjoy the programmed games and applications!

Additionally, the console's open-source design allows developers to create personalized extensions that align perfectly with the specific hardware components used in the gaming console. This extensibility opens up endless opportunities for customization and further enhancement, catering to the diverse needs of educators and students alike. By collaborating with developers, personalized extensions can be added to the console, continuously enriching its capabilities and empowering educators to create engaging and tailored learning experiences that align seamlessly with the Algerian curriculum.

Conclusion and Perspectives

3.6 Conclusion

In conclusion, this project report showcases the successful design and development of an educational gaming console based on Arduino, specifically tailored to the Algerian curriculum and aimed at promoting programming integration using Scratch. Despite facing numerous challenges and having limited resources, the project team achieved a significant milestone by successfully creating a functional PCB for the console. This accomplishment highlights the team's perseverance and problem-solving abilities in overcoming obstacles encountered during the implementation phase.

The educational gaming console serves as a powerful tool to empower students from second grade in middle school and beyond, fostering computational thinking and active participation in the learning process. By incorporating user-created games and providing a platform for creativity, the console encourages students to develop their programming skills, problem-solving abilities, and critical thinking capabilities.

Looking ahead, there are several exciting perspectives for further development and enhancement of the educational gaming console. One key aspect involves the continuous improvement of the block-based programming software, ensuring perfect compatibility with the console and alignment with the Algerian curriculum. By developing additional extensions and features, including advanced display choices such as LCD or OLED screens, the console can offer an even more immersive and engaging learning experience for students. These advanced displays would enable the console to showcase complex graphics, text, and user interface elements, thereby enhancing the overall user experience and expanding possibilities for game design and educational content.

Furthermore, integrating advanced features such as Wi-Fi connectivity and more powerful microcontrollers can significantly expand the console's capabilities. These additions would enable students to explore and create games that leverage online resources, connect with other learners, and interact with a broader community. The integration of such advanced features will enhance the educational value of the console and provide students with opportunities to apply their programming skills in real-world applications.

While significant progress has been made, it is important to recognize the ongoing need for resources and support to further develop and scale the educational gaming console. Securing additional funding and partnerships can help overcome limitations and ensure the console's accessibility to a wider audience of students and educators across Algeria.

In conclusion, the successful completion of the PCB and the team's unwavering determination throughout the project underscore the immense potential of the educational gaming console. By focusing on the continuous development of the block-based programming software, integrating advanced features such as an advanced display choice, Wi-Fi connectivity, and more powerful microcontrollers, and seeking additional resources, the console has the potential to revolutionize the way computational thinking is taught and learned in Algerian schools.

References

[1] I. Perenc, T. Jaworski, and P. Duch. "Teaching programming using dedicated Arduino Educational Board." In: *Comput Appl Eng Educ* 27 (2019), pp. 943–954.

[2] GeeksforGeeks. *Introduction of Programming Paradigms*. https://www.geeksforgeeks.org/introduction-of-programming-paradigms/.

[3] Arduino. *Arduino Reference*. Online. URL: https://www.arduino.cc/reference/en/.

[4] *Algerian Curriculum*. ar. 2022. URL: https://www.ency-education.com/plans_annuels_moyenne.html (visited on 11/23/2023).

[5] Simon Rose. "Developing children's computational thinking using programming games." PhD thesis. United Kingdom: Sheffield Hallam University, 2019.

[6] Stamatios Papadakis. "Evaluating a game-development approach to teach introductory programming concepts in secondary education." In: *International Journal of Technology Enhanced Learning* 12.2 (2020), pp. 127–145.

[7] Patrik Voštinár. "MakeCode Arcade: Interesting environment for programming 2D games." In: *2021 IEEE World Conference on Engineering Education (EDUNINE)*. IEEE. 2021, pp. 1–6.

[8] *Learn Electronics*. https://lastminuteengineers.com/.

[9] Zellat Khadidja. *Etude & Réalisation*. Higher National School Of Technology, 2022.

[10] Andouglas Gonçalves Da Silva Junior, Luiz Marcos Garcia Gonçalves, Glauco A. De Paula Caurin, et al. "Bipes: Block based integrated platform for embedded systems." In: *IEEE Access* 8 (2020), pp. 197955–197968.