DIGITAL ELECTRONICS GAME

Object Oriented Programming Lab Project

BY:

Ali Asghar (21PWCSE2059)

Suleman Shah (21PWCSE1983)

Shahzad Bangash (21PWCSE1980)

INTRODUCTION:

"Digital Electronics Game" is an educational game that teaches players the basics of digital electronics through interactive and engaging gameplay. The game is designed to be accessible to players of all ages, with a user-friendly interface and clear explanations of digital concepts.

The game is designed to provide players with hands-on experience in creating and simulating digital logic circuits.

GAME CONCEPT AND GOAL:

- The game is an educational and fun game for everyone.
- It is designed to generate the circuit which the user will solve correctly.
- The goal of the game is to complete various logic circuit tasks within a given time frame.
- Digital Logic and Circuit building technique are used for building the game.

GAME WORKING:

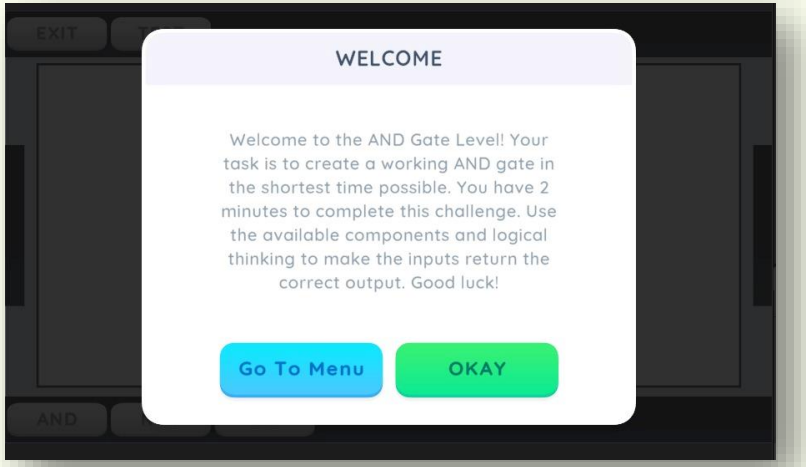
- The game consists of various levels, each with a different logic circuit task. In each level, the player is presented with a set of components and inputs.
- The user must use these components to create a working logic circuit that meets the specified requirements.
- **Simulation:**

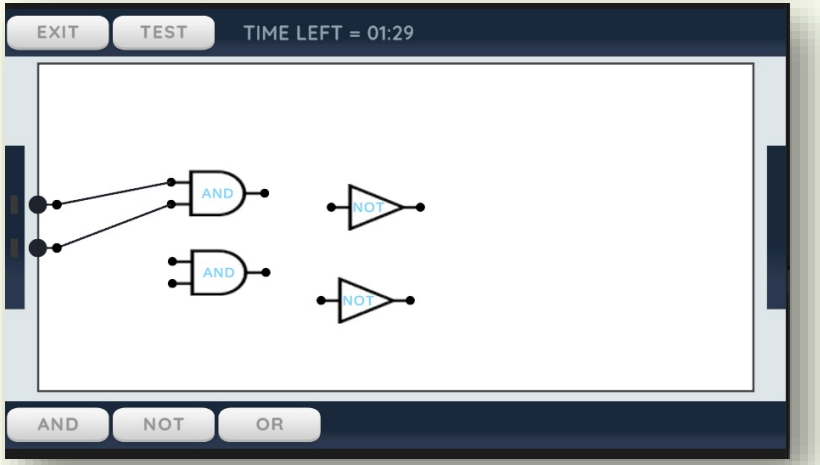
  The game features an advanced simulation engine that allows players to test their circuits and see the results in real-time.

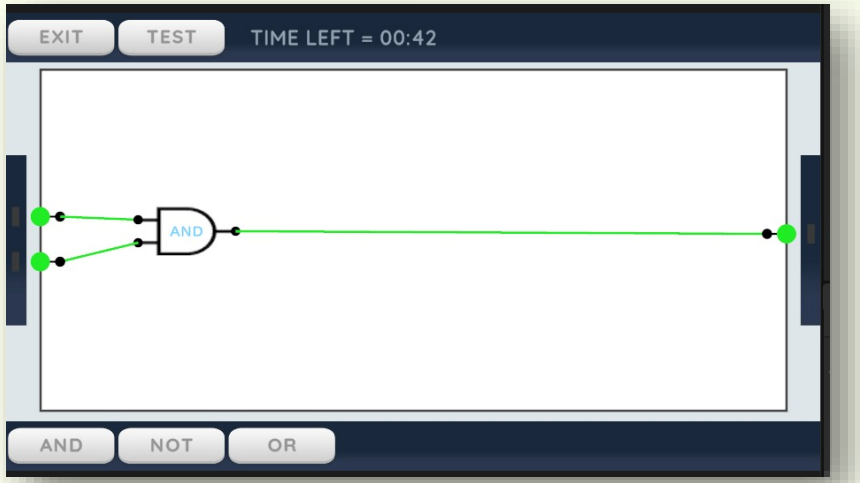
- **Time Limits:**

  Each level has a specific time limit, and the player must complete the task within this time frame to progress to the next level.

GAME SCREENSHOTS:







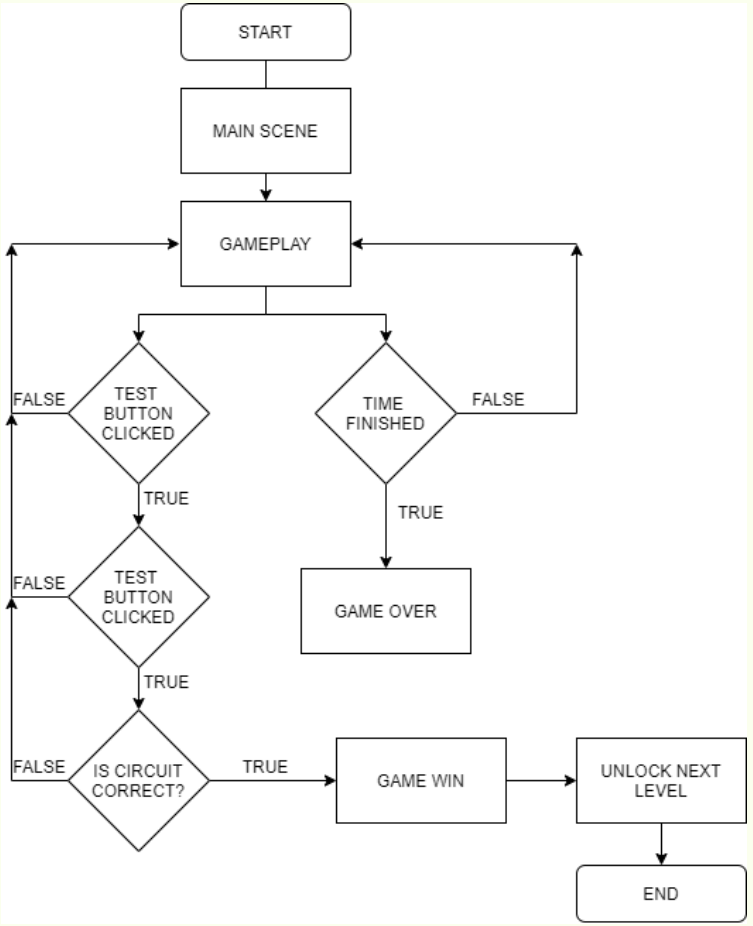
FRAMEWORK USED:

- The game is built using Unity, a powerful game engine that allows for the creation of interactive and immersive experiences.
- The game is built with C# language.
- The game features high-quality 3D graphics and animations that give players a realistic view of the virtual laboratory and the digital components they are working with.

OOP USED:

- Classes
- Objects
- Inheritance
- Static Classes
- Abstraction and Encapsulation

GAME FLOWCHART:



CODE SCREENSHOTS:

MANAGER CLASS

LevelsManager.cs | MainMenuManager.cs | Manager.cs | LoadLevel.cs

Assets > Scripts > Core > Manager.cs > Manager

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

9 references
public class Manager : MonoBehaviour
{

    1 reference
    public event System.Action<Chip> customChipCreated;
    5 references
    public Chip[] builtinChips;
    5 references
    ChipEditor activeChipEditor;
    0 references
    int currentChipCreationIndex;
    4 references
    public static Manager instance;
    1 reference
    public bool isGameOver = false;
    9 references
    public bool[] correctOutputs = { false, false, false, false };
    6 references
    public enum logicType
```

LevelsManager.cs | MainMenuManager.cs | Manager.cs | LoadLevel.c

Assets > Scripts > Core > Manager.cs > Manager > instance

```
    6 references
    public enum logicType
    {
        1 reference
        AND,
        1 reference
        NAND,
        1 reference
        OR,
        1 reference
        NOR,
        1 reference
        XOR
    }

    5 references
    public logicType levelType;
    0 references
    void Awake()
    {
        instance = this;
        activeChipEditor = FindObjectOfType<ChipEditor>();
    }
```

CODE SCREENSHOTS:

MANAGER CLASS

LevelsManager.cs | MainMenuManager.cs | Manager.cs | LoadLevel.cs

Assets > Scripts > Core > Manager.cs > Manager > instance

```
32
        3 references
33      public static ChipEditor ActiveChipEditor
34      {
35          get
36          {
37              return instance.activeChipEditor;
38          }
39      }
        4 references
40      public void SpawnChip(Chip chip)
41      {
42          activeChipEditor.chipInteraction.SpawnChip(chip);
43      }
44
        0 references
45      public void LoadMainMenu()
46      {
47          SceneManager.LoadScene(0);
48      }
        0 references
49      public void RestartSCene()
50      {
51          SceneManager.LoadScene(SceneManager.GetActiveScene().name);
```

LevelsManager.cs | MainMenuManager.cs | Manager.cs | LoadLevel.cs

Assets > Scripts > Core > Manager.cs > Manager > instance

```
51          SceneManager.LoadScene(SceneManager.GetActiveScene().name);
52      }
        0 references
53      public void StartGame()
54      {
55          CountdownTimer.instance.StartTimer();
56          GameUI.instance.hideWelcome();
57      }
58
        0 references
59      public void testResult()
60      {
61          //foreach(ChipSignal s in activeChipEditor.inputsEditor.signals)
62          //  Debug.Log(s.currentState);
63
64          //foreach(ChipSignal s in activeChipEditor.outputsEditor.signals)
65          //  Debug.Log(s.currentState);
66
67          List<ChipSignal> inputSignals = activeChipEditor.inputsEditor.signals;
68          List<ChipSignal> outputSignals = activeChipEditor.outputsEditor.signals;
69
70          if (inputSignals.Count == 2 && outputSignals.Count == 1)
```

CODE SCREENSHOTS:

MANAGER CLASS

LevelsManager.cs | MainMenuManager.cs | Manager.cs | LoadLevel.cs

Assets > Scripts > Core > Manager.cs > Manager > instance

```
        if (inputSignals.Count == 2 && outputSignals.Count == 1)
        {
            StartCoroutine(checkIO(inputSignals, outputSignals));
            Debug.Log("AND COROUTINE STARTED");
        }

        else
        {
            ModalWindow.instance.ShowModal("ERROR", "CHECK INPUTS AND OUTPUTS FIRST");
        }
    }

    1 reference
    public void checkFinalResult()
    {
        if (correctOutputs[0] && correctOutputs[1] &&
            correctOutputs[2] && correctOutputs[3])
        {
            GameUI.instance.ShowWin();
            CountdownTimer.instance.StopTimer();
            if (LevelsManager.instance)
                LevelsManager.instance.UnclockNextLevel();
```


LevelsManager.cs | MainMenuManager.cs | Manager.cs | LoadLevel.cs

Assets > Scripts > Core > Manager.cs > Manager > instance

```
            LevelsManager.instance.UnclockNextLevel();

        Debug.Log("YOU WON");
    }

    else
    {
        ModalWindow.instance.ShowModal("ERROR","ALL OUTPUTS ARE NOT CORRECT.\nPLEASE MAK
        CountdownTimer.instance.StartTimer();
        Debug.Log("ALL OUTPUTS NOT CORRECT");
    }


    1 reference
    Enumerator checkIO(List<ChipSignal> iSignalsList, List<ChipSignal> oSignalsList)

    CountdownTimer.instance.StopTimer();
    int s_no;
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
```

CODE SCREENSHOTS:

MANAGER CLASS

LevelsManager.cs | MainMenuManager.cs | Manager.cs | LoadLevel.cs

Assets > Scripts > Core > Manager.cs > Manager > instance

```
1 reference
Enumerator checkIO(List<ChipSignal> iSignalsList, List<ChipSignal> oSignalsList

    CountdownTimer.instance.StopTimer();
    int s_no;
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
            s_no = System.Convert.ToInt32(i + "" + j, 2);
            ((InputSignal)iSignalsList[0]).SendSignal(i);
            ((InputSignal)iSignalsList[1]).SendSignal(j);

            yield return new WaitForSeconds(0.08f);

            if (levelType == logicType.AND)
            {
                Debug.Log("AND");

                if (((OutputSignal)oSignalsList[0]).currentState == (i & j))
                {
                    correctOutputs[s_no] = true;
                    Debug.Log("Success");
```

Manager.cs

Assets > Scripts > Core > Manager.cs > Manager > checkIO(List<ChipSignal> iSignalsList, List<ChipSignal> oSignals

```
                    correctOutputs[s_no] = true;
                    Debug.Log("Success");
                }
            }

            if (levelType == logicType.NAND)
            {
                Debug.Log("NAND");

                if (((OutputSignal)oSignalsList[0]).currentState != (i & j))
                {
                    Debug.Log( ~(i & j));
                    correctOutputs[s_no] = true;
                    Debug.Log("Success");
                }
            }

            if (levelType == logicType.OR)
            {
                Debug.Log("OR");

                if (((OutputSignal)oSignalsList[0]).currentState == (i | j))
```

CODE SCREENSHOTS:

MANAGER CLASS

LevelsManager.cs | MainMenuManager.cs | Manager.cs | LoadLevel.cs

Assets > Scripts > Core > Manager.cs > Manager > instance

```
                Debug.Log("Success");
            }
        }

        if (levelType == logicType.NAND)
        {
            Debug.Log("NAND");

            if (((OutputSignal)oSignalsList[0]).currentState != (i & j))
            {
                Debug.Log( ~(i & j));
                correctOutputs[s_no] = true;
                Debug.Log("Success");
            }
        }

        if (levelType == logicType.OR)
        {
            Debug.Log("OR");

            if (((OutputSignal)oSignalsList[0]).currentState == (i | j))
            {
                correctOutputs[s_no] = true;
                Debug.Log("Success");
```

Manager.cs

Assets > Scripts > Core > Manager.cs > Manager > checkIO(List<ChipSignal> iSignalsList, List<ChipSignal> oSignal

```
            if (((OutputSignal)oSignalsList[0]).currentState == (i | j))
            {
                correctOutputs[s_no] = true;
                Debug.Log("Success");
            }
        }

        if (levelType == logicType.NOR)
        {
            Debug.Log("NOR");

            if (((OutputSignal)oSignalsList[0]).currentState != (i | j))
            {
                correctOutputs[s_no] = true;
                Debug.Log("Success");
            }
        }

        if (levelType == logicType.XOR)
        {
            Debug.Log("XOR");
```

CODE SCREENSHOTS:

MANAGER CLASS

LevelsManager.cs | MainMenuManager.cs | Manager.cs | LoadLevel.cs

Assets > Scripts > Core > Manager.cs > Manager > instance

```
                    correctOutputs[s_no] = true;
                    Debug.Log("Success");
                }
            }

            if (levelType == logicType.NOR)
            {
                Debug.Log("NOR");

                if (((OutputSignal)oSignalsList[0]).currentState != (i | j))
                {
                    correctOutputs[s_no] = true;
                    Debug.Log("Success");
                }
            }

            if (levelType == logicType.XOR)
            {
                Debug.Log("XOR");

                if (((OutputSignal)oSignalsList[0]).currentState == (i ^ j))
                {
```

Manager.cs

Assets > Scripts > Core > Manager.cs > Manager > checkIO(List<ChipSignal> iSignalsList, List<ChipSignal> o

```
                    if (((OutputSignal)oSignalsList[0]).currentState != (i | j))
                    {
                        correctOutputs[s_no] = true;
                        Debug.Log("Success");
                    }
                }

                if (levelType == logicType.XOR)
                {
                    Debug.Log("XOR");

                    if (((OutputSignal)oSignalsList[0]).currentState == (i ^ j))
                    {
                        correctOutputs[s_no] = true;
                        Debug.Log("Success");
                    }
                }
            }
        }
        checkFinalResult();
    }
}
```

CODE DESCRIPTION:

- The Code is of manager class that we have implemented in our project. This class handles the level win and lose logic.
- It contains functions to load main menu or restarting a scene.
- It also contains a function which starts the countdown timer in the scene.
- The count down timer class count the remaining time in the level of the game, when the time runs out in the level the player loses.
- Just like manager class, we have implemented many classes for And, Or and Not logic etc. l
- All the classes are mentioned in the lab report.

LEARNING OUTCOME:

- The digital electronics game provides players with an interactive and engaging way to learn about digital electronics.
- The game is designed to teach players the fundamental principles of digital circuits, such as Boolean algebra, gate logic, and circuit simulation.
- By playing the game, players can build their own circuits and see the results of their work, which reinforces their understanding of the subject and helps them retain the information.

CONCLUSION:

Overall, **"Digital Electronics Game"** is an engaging and educational game that is designed to teach players the basics of digital electronics through interactive gameplay and challenges. The game is perfect for anyone interested in learning about digital electronics, whether they are students, teachers, or professionals in the field.

THANK YOU FOR YOUR TIME