PROJECT REPORT

Fall 2022

CSE208L Object Oriented Programming Lab

Suleman Shah (**21PWCSE1983)**

Shahzad Bangash (**21PWCSE1980)**

Ali Asghar (**21PWCSE2059)**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Engr. Sumayyea Salahuddin

February 6, 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

DIGITAL ELECTRONICS GAME

Introduction:

The digital electronics game is a unity-based educational game that teaches players about digital logic circuits. The game is designed to provide players with hands-on experience in creating and simulating digital logic circuits. The goal of the game is to complete various logic circuit tasks within a given time frame. The game's focus is on teaching players the basic principles of digital electronics and how to apply these principles to solve real-world problems.

Gameplay:

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, and must use these components to create a working logic circuit that meets the specified requirements. The player has a limited amount of time to complete each task, which adds an element of challenge and excitement to the game.

Components:

The game includes components, such as AND gates, OR gates, NOT gates. Players are able to connect these components together to create more complex circuits. Each component is animated to show its state, which helps players understand how the circuit works and what is happening at each stage.

Simulation:

The game features an advanced simulation engine that allows players to test their circuits and see the results in real-time. This allows players to experiment with different circuit configurations and see the effect on the outputs. The simulation engine is highly accurate and provides a realistic representation of how real-world digital circuits behave.

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. This adds an element of challenge and encourages players to think quickly and work efficiently. The time limits can be adjusted to suit the player's skill level and provide a greater or lesser degree of difficulty.

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 are able to build their own circuits and see the results of their work, which reinforces their understanding of the subject and helps them retain the information.

Framework Used:

Unity Engine and C#.

Oop Used in Project:

- Objects and Classes(Object Oriented Programming)
- Inheritance
- Abstraction & Encapsulation
- Static Classes

Game Flow Chart:

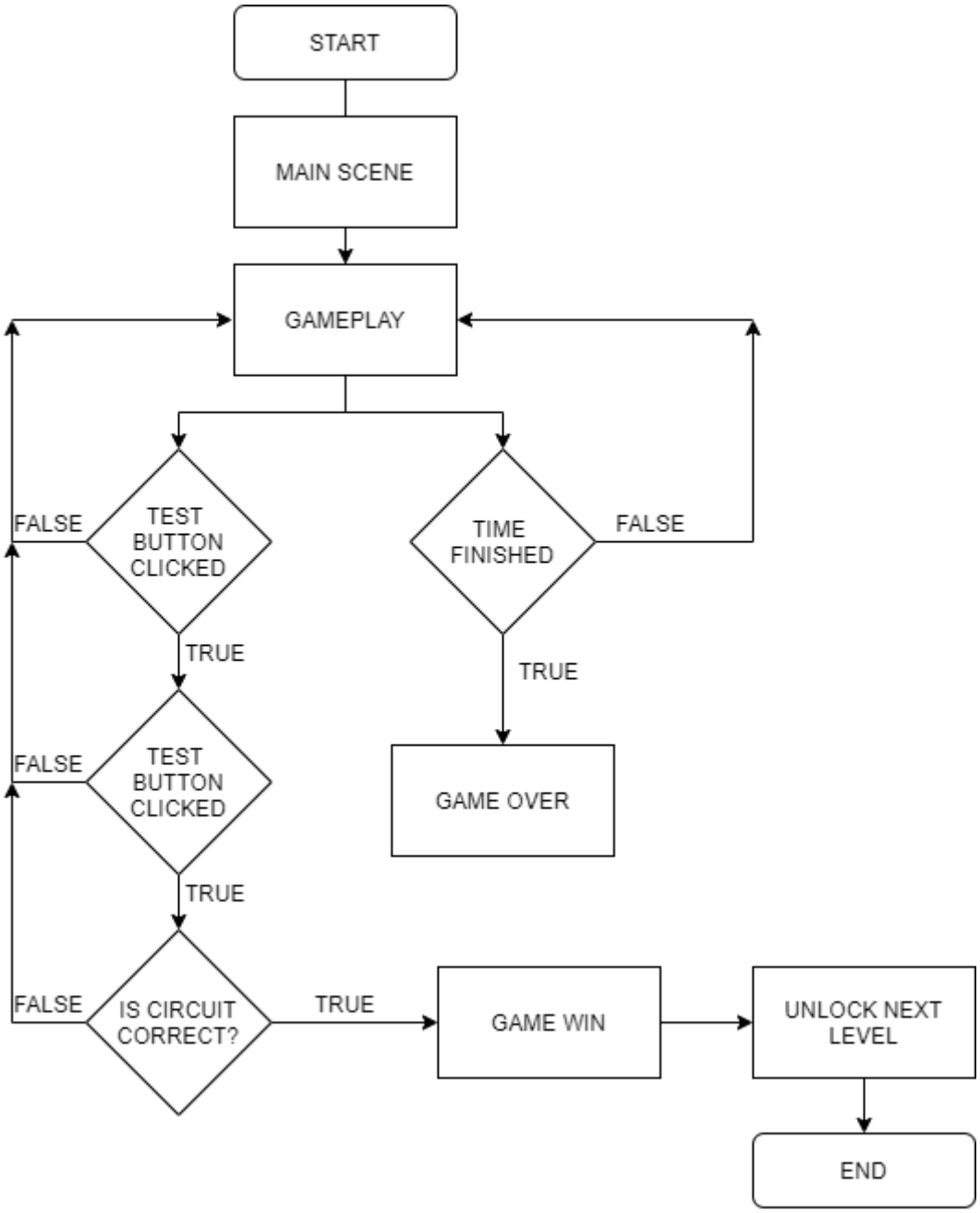

Figure 1

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
```

Figure 2

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

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

```
        6 references
16      public enum logicType
17      {
            1 reference
18          AND,
            1 reference
19          NAND,
            1 reference
20          OR,
            1 reference
21          NOR,
            1 reference
22          XOR
23      }
24
        5 references
25      public logicType levelType;
        0 references
26      void Awake()
27      {
28          instance = this;
29          activeChipEditor = FindObjectOfType<ChipEditor>();
30      }
31
32
```

Figure 3

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);
```

Figure 4

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

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

```
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
    0 references
    public void StartGame()
    {
        CountdownTimer.instance.StartTimer();
        GameUI.instance.hideWelcome();
    }

    0 references
    public void testResult()
    {
        //foreach(ChipSignal s in activeChipEditor.inputsEditor.signals)
        //  Debug.Log(s.currentState);

        //foreach(ChipSignal s in activeChipEditor.outputsEditor.signals)
        //  Debug.Log(s.currentState);

        List<ChipSignal> inputSignals = activeChipEditor.inputsEditor.signals;
        List<ChipSignal> outputSignals = activeChipEditor.outputsEditor.signals;

        if (inputSignals.Count == 2 && outputSignals.Count == 1)
```

Figure 5

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();
```


Figure 6

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++)
        {
```

Activate Windows

Figure 7

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");
```

Figure 8

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))
```


Figure 9

Manager.cs

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

```
144                 if (((OutputSignal)oSignalsList[0]).currentState == (i | j))
145                 {
146                     correctOutputs[s_no] = true;
147                     Debug.Log("Success");
148                 }
149             }
150
151             if (levelType == logicType.NOR)
152             {
153                 Debug.Log("NOR");
154
155                 if (((OutputSignal)oSignalsList[0]).currentState != (i | j))
156                 {
157                     correctOutputs[s_no] = true;
158                     Debug.Log("Success");
159                 }
160             }
161
162             if (levelType == logicType.XOR)
163             {
164                 Debug.Log("XOR");
165
```

Activate Windows

Figure 10

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");
```

Figure 11

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))
                {
```

Figure 12

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();
    }
}
```

Figure 13

AND GATE CLASS

Manager.cs | Chip.cs | AndGate.cs

Assets > Scripts > Chip > AndGate.cs > ...

```
using UnityEngine;

0 references
public class AndGate : BuiltinChip {

    3 references
    protected override void Awake () {
        base.Awake ();
    }

    1 reference
    protected override void ProcessOutput () {
        int outputSignal = inputPins[0].State & inputPins[1].State;
        outputPins[0].ReceiveSignal (outputSignal);
    }

}
```

Figure 14

NOT GATE CLASS

Manager.cs | Chip.cs | AndGate.cs | NotGate.cs

ssets > Scripts > Chip > NotGate.cs > NotGate

```
0 references
public class NotGate : BuiltinChip {

    3 references
    protected override void Awake () {
        base.Awake ();
    }

    1 reference
    protected override void ProcessOutput () {
        int outputSignal = 1 - inputPins[0].State;
        outputPins[0].ReceiveSignal (outputSignal);
    }
}
```

Figure 15

OR GATE CLASS

ssets > Scripts > Chip > OrGate.cs > OrGate

```
using UnityEngine;

0 references
public class OrGate : BuiltinChip {

    3 references
    protected override void Awake () {
        base.Awake ();
    }

    1 reference
    protected override void ProcessOutput () {
        int outputSignal = inputPins[0].State | inputPins[1].State;
        outputPins[0].ReceiveSignal (outputSignal);
    }

}
```

Figure 16

COUNT DOWN TIMER CLASS

ssets > Scripts > UI > MyScripts > CountdownTimer.cs > CountdownTimer > t

```
using UnityEngine;
using UnityEngine.UI;

5 references
public class CountdownTimer : MonoBehaviour
{
    5 references
    public static CountdownTimer instance;
    1 reference
    public float timeLimit;
    1 reference
    public TMPro.TMP_Text timerText;
    5 references
    private float timeRemaining;
    4 references
    private bool timerIsRunning = false;

    0 references
    private void Awake() {
        instance = this;
    }
    0 references
    private void Start()
    {
        timeRemaining = timeLimit;
```

Figure 17

ssets > Scripts > UI > MyScripts > CountdownTimer.cs > CountdownTimer > timerIsRunning

```
17  timeRemaining = timeLimit;
18
19
    0 references
20  ate void Update()
21
22  if (timerIsRunning)
23  {
24      if (timeRemaining > 0)
25      {
26          timeRemaining -= Time.deltaTime;
27          //int hours = (int)(timeRemaining / 3600);
28          int minutes = (int)((timeRemaining % 3600) / 60);
29          int seconds = (int)(timeRemaining % 60);
30          //timerText.text = string.Format("{0:00}:{1:00}:{2:00}", hours, minutes, seconds
31          timerText.text = "TIME LEFT = "+string.Format("{0:00}:{1:00}", minutes, seconds)
32
33      }
34      else
35      {
36          timerIsRunning = false;
37          TimeUp();
```

Figure 18

```
        }
    }

    2 references
    public void StartTimer()
    {
        timerIsRunning = true;
    }

    2 references
    public void StopTimer()
    {
        timerIsRunning = false;
    }

    1 reference
    public void TimeUp()
    {
        Debug.Log("Time's up!");
        Manager.instance.isGameOver = true;
        GameUI.instance.ShowLose();
    }
}
```

Figure 19

END GAME SCREEN CLASS

Assets > Scripts > UI > MyScripts > EndGameScreen.cs > ...

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
2 references
public class EndGameScreen : MonoBehaviour
{
    1 reference
    public static EndGameScreen instance;
    3 references
    public GameObject endGameWindow;
    1 reference
    public Button nextLevelButton;
    1 reference
    public Button restartLevelButton;
    1 reference
    public TMP_Text messageTitle;
    1 reference
    public TMP_Text messageText;

    0 references
    private void Awake() {
        instance = this;
    }
    0 references
    private void Start(){
```

Figure 20

Assets > Scripts > UI > MyScripts > EndGameScreen.cs > EndGameScreen

```
15      }
        0 references
16      private void Start(){
17          endGameWindow.SetActive(false);
18      }
19
        2 references
20      public void ShowEndGameScreen(string title, string message){
21          messageTitle.text = title;
22          messageText.text = message;
23          endGameWindow.SetActive(true);
24      }
25
        0 references
26      public void CloseEndGameScreen(){
27          endGameWindow.SetActive(false);
28      }
29
        0 references
30      public void OnNextButtonClicked(){
31          if (LevelsManager.instance)
32              LevelsManager.instance.PlayLevel();
33      }
34  }
```

Figure 21

MODAL WINDOW CLASS

Assets > Scripts > UI > MyScripts > ModalWindow.cs > ModalWindow > ShowModal(string t

```
using TMPro;
4 references
public class ModalWindow : MonoBehaviour
{
    3 references
    public static ModalWindow instance;
    3 references
    public GameObject modalWindow;
    1 reference
    public TMP_Text messageTitle;
    1 reference
    public TMP_Text messageText;

    0 references
    private void Awake() {
        instance = this;
    }
    0 references
    private void Start(){
        modalWindow.SetActive(false);
    }
    2 references
    public void ShowModal(string title, string message){
        messageTitle.text = title;
        messageText.text = message;
        modalWindow.SetActive(true);
    }
    0 references
    public void CloseModal(){
        modalWindow.SetActive(false);
    }
}
```

Figure 22

GAME SCREENSHOTS:

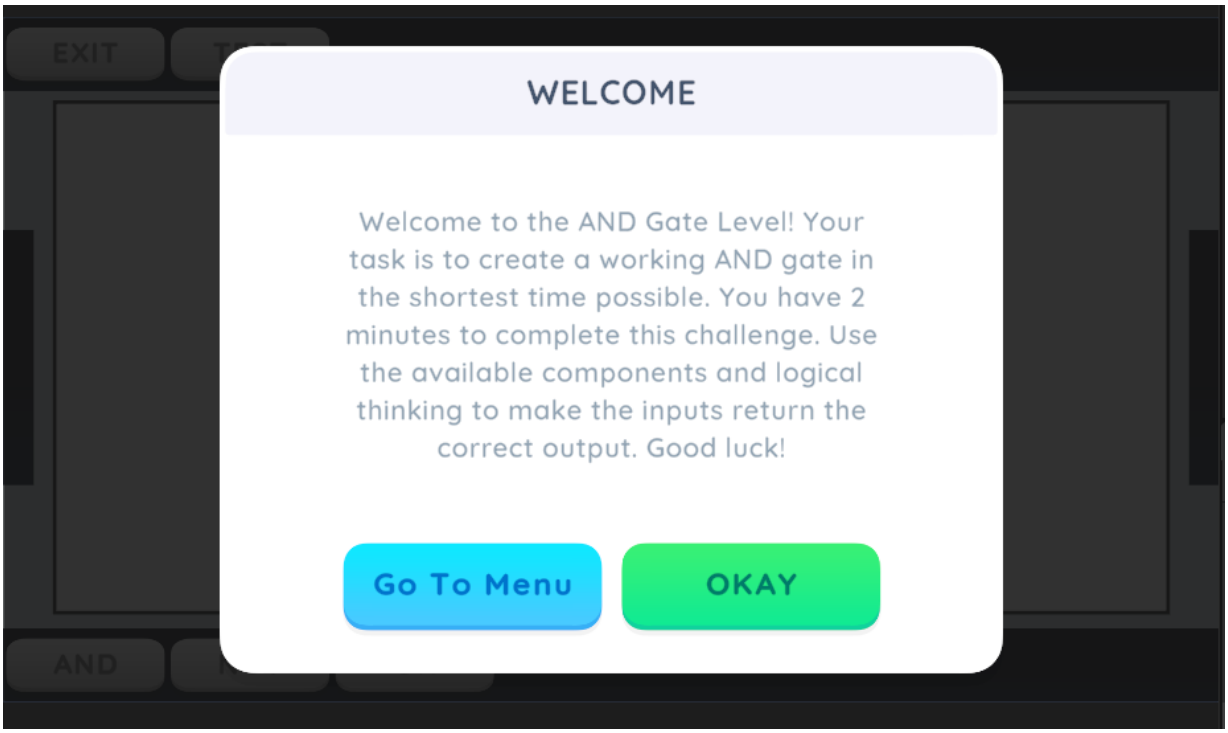

Figure 23

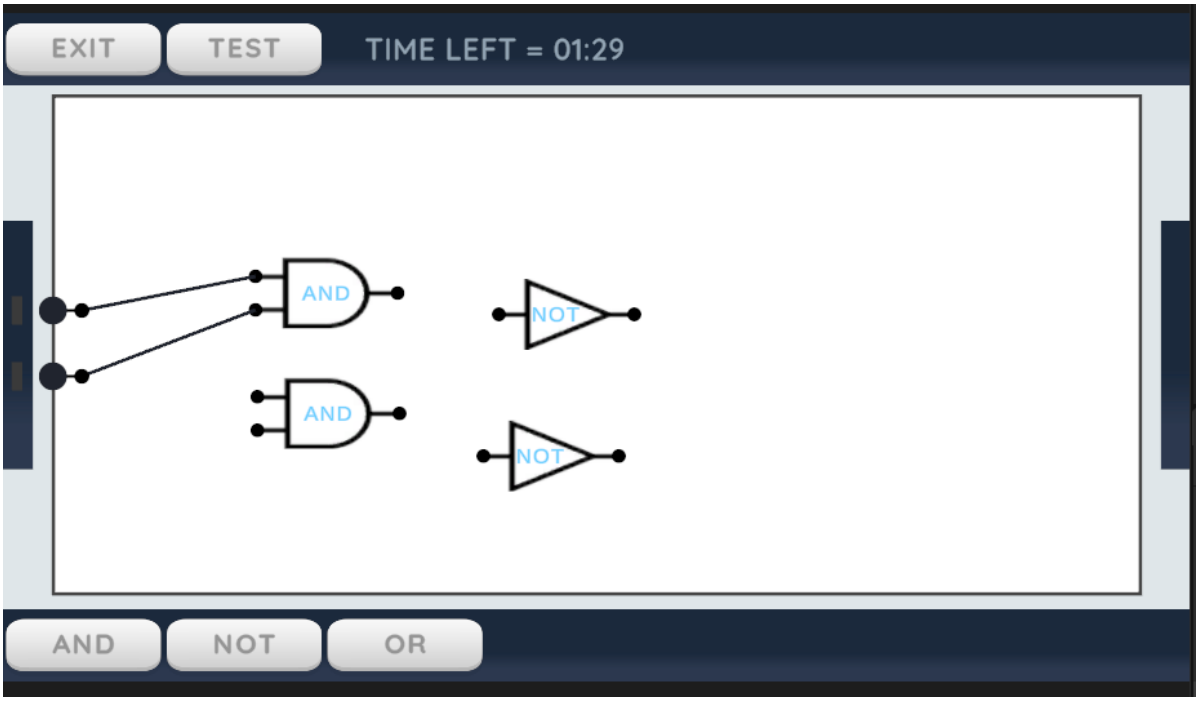

Figure 24

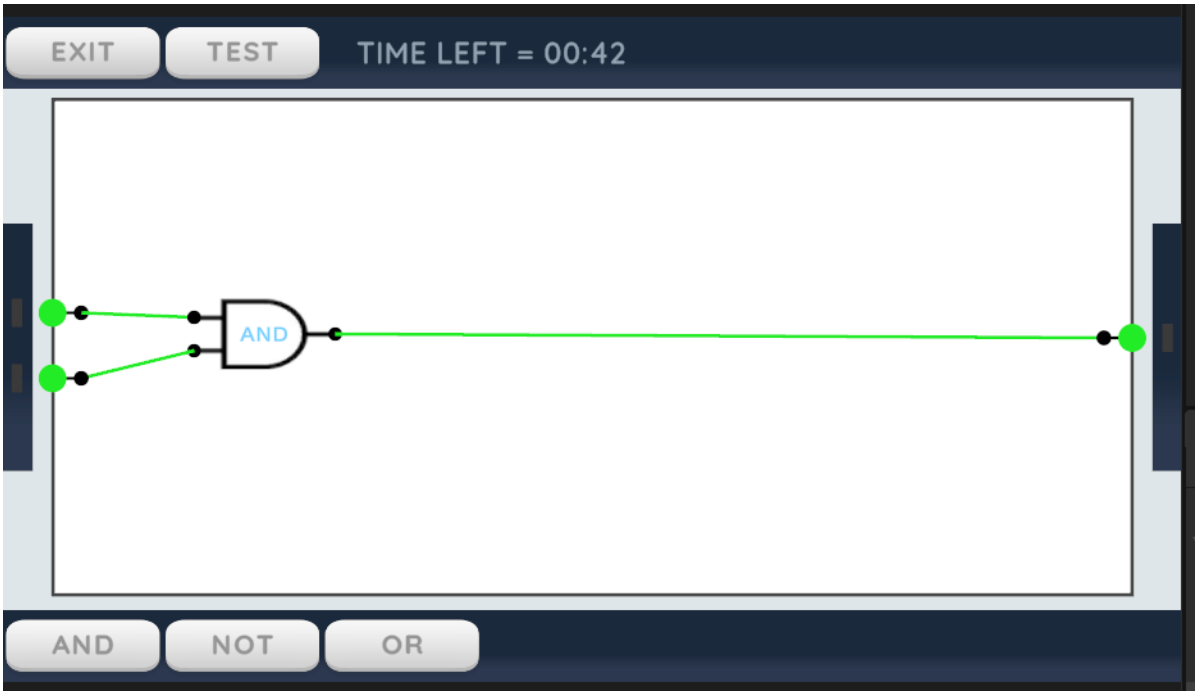

Figure 25

Conclusion:

The digital electronics game is a fun and educational tool that can help players learn about digital electronics in a fun and interactive way. The game is built using the unity platform, which provides a rich and immersive experience for players. The combination of hands-on learning, simulation, and time limits makes the game an ideal tool for teaching digital electronics and providing a fun and engaging way for players to learn.