RoadWyze: A Game Development using Dijkstra's Algorithm

Monseph Johnrey M. Balenton
Lyceum of the Philippines University -Batangas
Capitol Site, Batangas City
09951471687
monsephjohnreybalenton@lpubatangas.edu.ph
ORCID: 0000-0002-6807-6787

John Frietz F. Dimaano
Lyceum of the Philippines University - Batangas
Capitol Site, Batangas City
09062209424
johnfrietzdimaano@lpubatangas.edu.ph
ORCID: 0000-0001-9053-7351

Dara Pamela L. Gutang
Lyceum of the Philippines University - Batangas
Capitol Site, Batangas City
09097959107
darapamelagutang@lpubatangas.edu.ph
ORCID: 0000-0002-7023-1854

Simon Luther E. Morales
Lyceum of the Philippines University - Batangas
Capitol Site, Batangas City
09479522189
simonluthermorales@lpubatangas.edu.ph
ORCID: 0000-0002-9414-2960

William Joseph C. Tan
Lyceum of the Philippines University - Batangas
Capitol Site, Batangas City
09491505827
williamjosephtan@lpubatangas.edu.ph
ORCID: 0000-0002-3491-2255

ABSTRACT

RoadWyze: A Game Development using Dijkstra's Algorithm, is a game where the player undergoes driver's education and learn the basics of operating a vehicle. RoadWyze was developed with the help of Unity3D, in which its primary pathfinding algorithm is Dijkstra's Algorithm, used in the in-game navigator that helps the player drive around the city's levels. The programming language for the game's code is C#, used for player movement, vehicle AI, and other aspects. The driving simulation game uses the standard WASD control scheme, and the mouse is used to look around within the driver's first-person line of sight. RoadWyze includes the quiz game mode, and all three levels. But the game's basic functions; vehicle movement, camera movement among few others are completed and are operational. With the base aspects finished, the player will be able to drive, see around and interact with their 3D environment.

Keywords: Dijkstra's Algorithm, Driving Game, Driving Simulator

1.0 INTRODUCTION

RoadWyze is a 3D first person view PC game, which is a pre-activity of driving a car. The game is similar to a real life driving school where you can follow the instructions of the instructor. By playing this game, the player is enrolled to a driving academy where the first event of the game is held. The player can either choose and take a short quiz or play the actual driving game.

Driving recklessly can cause road incidents, which often lead to endless traffic or unexpected deaths. Some main causes are intoxication, madness, road rage, and naïveté. For the actual driving lessons, one can just proceed to driving offices and take the courses, but coming unprepared can make it little bit tenser for aspiring drivers. To prepare them, the group introduced the

"*RoadWyze*", a driving test simulation. Historically, simulation in general is used in various fields which can be used as a virtual training ground through the use of technology which can be used for testing, training, education and even in games. It can also be used to show possible outcomes for instances to test if it may or may not be unsafe and unstable.

Unlike other formal driving simulators, "*RoadWyze"* not only provides education, but entertainment as well. It aims to simulate at real-time the thrill of operating a car in real life, as the player drives his car on environments closely resembling that of various locations. The game would be a first person view driving simulator, allowing the player to get a closer look inside the controls of their vehicle. The scoring system can give off a feeling of achievement to players whenever they accomplish goals set by the desktop game.

Enhancing players' knowledge that is all about driving is the main purpose of the game. Every step and every sign should be noted keenly for both driver's and pedestrian's safety. Traffic symbols and road signs are crucial key points when it comes to driving. Some road incidents are caused by inability to read and understand these warnings. This is why it is very important to keep this in someone's knowledge whenever they're on the road.

For the game to be well-known and widely used, the team targeted the teens who are willing to learn the mechanics of driving, specifically whose ages are 15 years and above. The reason is that this is the minimum age requirement according to the Land Transportation Office of the Philippines. To obtain a driver's license one must at least be at 17y/o, at the age of 15 they can already start learning the basics of traffic signs and road symbols, whilst having fun at the same time. At the span of two years they can already be introduced in familiarization of traffic signs and symbols.

Objectives of the Study

The thesis titled "RoadWyze" aimed to:

1. To develop a PC based game that will help drivers understand the meaning and use of traffic signs and symbols.

2. To apply Dijkstra's Algorithm in the development of the game.

3. To use C# programming language in the development of the game.

2.0 LITERATURE REVIEW

Game Development

Game development is the method of creating a game may it be for PC or for mobile devices. These games are usually created by a single person or a team of talented individuals. The evolution of games through the years expounds people's imagination and creativity that spans from countless genres. Meanwhile, this method had been existing in the Philippines as early as 1992, with the use of various algorithms. On the other hand, driving has been very beneficial for everyone. Unfortunately, this benefit has its' own scary downfall, human errors can cause numerous casualties. This is why safety is a big of a deal whenever people are introduced to anything that may be harmful. With the use of "Dijkstra's algorithm", the researchers have decided to combine game development and driving in a form of a simulator. Dijkstra's algorithm is applied to a navigator when the player is in the actual driving mode. The navigator will guide the player the drive around of the city, from a starting point to a destination.[2]

Simulation

Simulation is a method of imitating something relative to reality. This method existed as early as 1958 for games with what most people considered as the first game, "Tennis-For-Two" by a Physicist William Higginbotham. Since then, it has been used in various fields such as surgical operations, medical practices, sports, household chores, and driving in general. Simulation has been very helpful as a virtual training ground for most fields. Driving per se, has its' fair share for simulation. It's been helping users to prepare and know the rules and precautions whenever one steps foot on a car. As of today, many driving games are on the market.

Dijkstra's Algorithm

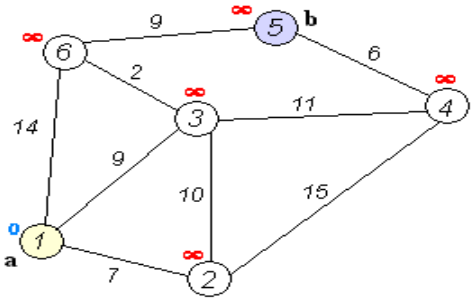

Figure 1. Dijkstra's Algorithm.
This figure shows what Dijkstra's Algorithm looks like. The starting node will eventually find the shortest route to the destination node.

The researchers have developed the project with the use of Dijkstra's algorithm. Dijkstra's algorithm is an algorithm to find the shortest path between the graphs. It is the best choice for the game, considering that it would be set on the Philippines, knowing that the roads are either under construction or on a heavy traffic and since that it is an Open Shortest-Path First Path Finding algorithm which finds the best and shortest path of a starting node (a) to its destination node (b), with this kind of algorithm it will help accomplish the objectives. [2]

[2] The step by step process of Dijkstra's algorithm uses the OSPF process:

Step 1: Start at the ending vertex by marking it with a distance of 0, because it is 0 units from the end. Call this vertex your current vertex, and put a circle around it indicating as such.

Step 2: Identify all of the vertices that are connected to the current vertex with an edge. Calculate their distance to the end by adding the weight of the edge to the mark on the current vertex. Mark each of the vertices with their corresponding distance, but only change a vertex's mark if it is less than a previous mark. Each time you mark the starting vertex with a mark, keep track of the path resulted in that mark.

Step 3: Label the current vertex as visited by putting an X over it. Once a vertex is visited, we won't look at it again.

Step 4: Of the vertices you just marked, find the one with the smallest mark, and make it your current vertex. Now, you can start again from step 2.

Step 5: Once you have labeled the beginning vertex as visited - stop. The distance of the shortest path is the mark of the starting vertex, and the shortest path is the path that resulted in that mark.

Traffic Signs & Symbols

Traffic signs and symbols are signage that can be found on the roads and highways. These are the signs and symbols that warn the driver of possible danger and provide information about road signs. Traffic signs tells the driver information about what the rules are, and the conditions of the road are like. Road Signs are classified into a number of categories:

Danger Warning Signs

Warning signs can indicate any potential hazard, obstacle or condition requiring special attention. [3]

Priority Signs

Priority traffic signs indicate the order in which vehicles should pass intersection points. [3]

Regulatory Signs

Regulatory traffic signs are used to prohibit certain types of maneuvers or some types of traffic. [3]

Mandatory Signs

Mandatory signs are road signs which are used to set the obligations of all traffic which use a specific area of road. Unlike prohibitory or restrictive signs, mandatory signs tell traffic what are 'must do', rather than 'must not do'. [3]

3.0 METHODS

Today's convenience of driving has been very helpful for almost everyone. It eases people's everyday routines, from travelling, to getting early to work, to just meeting up with someone, everything. But this advantage has its scary downfall, where those conveniences suddenly became danger. Some people causes road incidents and some causes heavy traffic that delays productivity for some. One major reason why is the inability to follow driving rules. The researchers of this study put their focus on that specific reason, that after the development of this study they may be able to help fix the problem.

Game Development Life Cycle GDLC)

This phase is where the game life cycle begins. The RoadWyze consists of five phases:

a. Idea
b. Game Design
c. Technical Requirement Analysis
d. Programming
e. Testing
f. Deployment

This study uses a descriptive research tool for justifying information. The researchers reviewed the approved proposal and the Introduction of the study that might help the flow of the research. *(See Figure 2).*

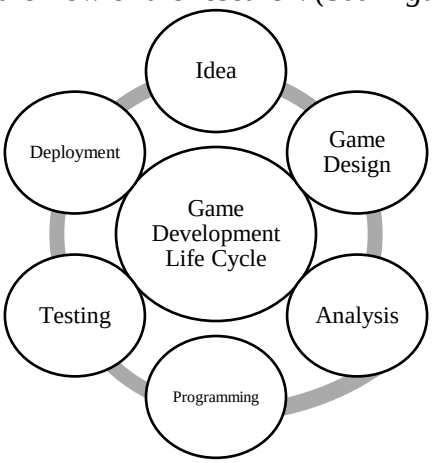

Figure 2. Game Development Life Cycle.
This figure represents the Game Development Life Cycle of RoadWyze.

3.1 Idea

In order to be successful with the study's goals, which are; 'To develop a PC based game that will help drivers understand the meaning and use of traffic signs and symbols.; 'To apply Dijkstra's Algorithm in the development of the game.'; and 'To use C# programming language in the development of the game.' the researchers selected a project development cycle over which they will follow for the rest of the study. The steps are listed below:

a. Creation of project plan.
b. Creation of resource plan.
c. Creation of financial plan,

d. Creation of risk management plan.
e. Creation of project maintenance.

3.2 Game Design

In the game design and development phase the researchers implemented computer application tools for this project to be assembled. The materials for organizing the game for software are as follows:

a. Unity3D
b. Adobe Photoshop CC 2017
c. Microsoft Visual Studio 2017
d. Audacity

The design of the three levels and 3D assets of buildings, roads, and cars are made in Unity3D.The assets are taken from the Asset Store of Unity which can be bought and downloaded and some of the assets that the developers used are free. The game developers and game editors of the team originally designed the concept of each level and it took time before finishing it. The goals are directed by the team's game analysts and game designers. Adobe Photoshop is used for the design of the GUI and 2D sprites. The design ideas are originally made by the team's 2D and 3D user interface designers. Sound effects are being edited using Audacity. Background music was downloaded from the internet for free and it is composed by video game sound producers.

3.3 Technical Requirement Analysis

RoadWyze will require an algorithm which is path-finding algorithm. Path-finding is an algorithm works step by step process of finding the shortest route between two points. Path-finding has some well know methods like A*, Dijkstra's Algorithm, and Depth and Breadth-First searches. With the use of a more probabilistic variant of algorithm that can simulate traffic in a smart city, where drivers may or may not drive down the shortest routes they know of. The researchers will use the Dijkstra's algorithm to find the shortest path while avoiding the obstacles in the driving school and the road. A modified version of the Dijkstra's algorithm can help car navigation systems find the shortest path from one place to another. The algorithm has been used to help find ways of improving the efficiency of mass transportation systems, examining Ankara's public transportation system and printing the stations as spatial nodes.

Dijkstra's algorithm as a pathfinding help unit movement in a real-time strategy game. It calculates the shortest path for a unit to move to a different location by computing the distance from the starting node to the destination node, and determining the path with the least cost *(see Figure 3).* [2]

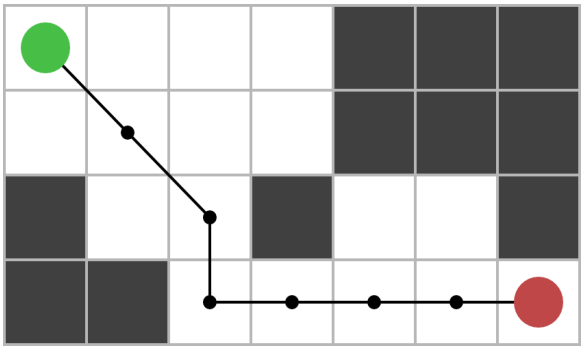

Figure 3. Dijkstra's algorithm 2D simple path. The green circle represents the starting node where it finds the shortest path going to the red circle which represents the destination node.

3.4 Programming

C# is the main programming language that is applied in this game project. The programmers used it for script writing. The scripts are applied on the car where the body of the car has a script for the movement

of the wheels, also for the first-person camera view. The script of each game mode and levels are written and analyzed by the team's programmers.

Unity has only two scripting languages that are available. The other language is JavaScript. The programmers choose C# because it is more familiar with the use of object-oriented programming.

3.5 Testing

For the hardware material, the team used a PC/Laptop as the main tool for the development of the game. Unity has a primary feature of testing a game project. The developers used this testing tool which is the play button, where you can test the game and debug errors.

3.6 Deployment

The computer game application has already surpassed the six phases of the game development life cycle, then it is ready to be released. The researchers implemented a complete version of the game and it is ready for official release. Also, for feedbacks the rate for the quality of the game, and FAQs. With the help of these techniques, the researchers will be informed for further enhancements and updates for the game.

Flowcharts

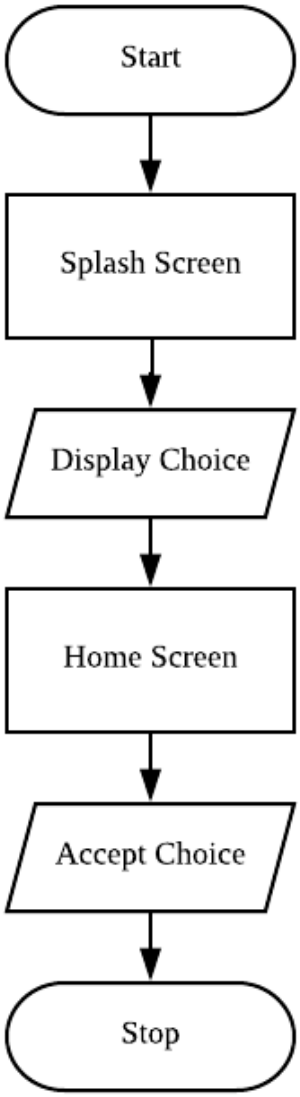

Figure 4. Start Screen
After clicking the icon the Start screen will be displayed and shown above is the process that will occur from that.

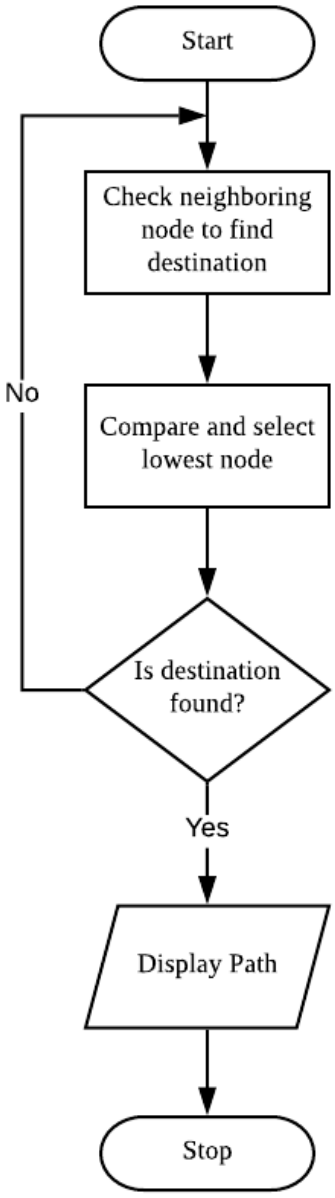

Figure 5. Algorithm
The following algorithm compares the nearby nodes and selects the one with the shortest distance. It repeats this process until it reaches the destination.

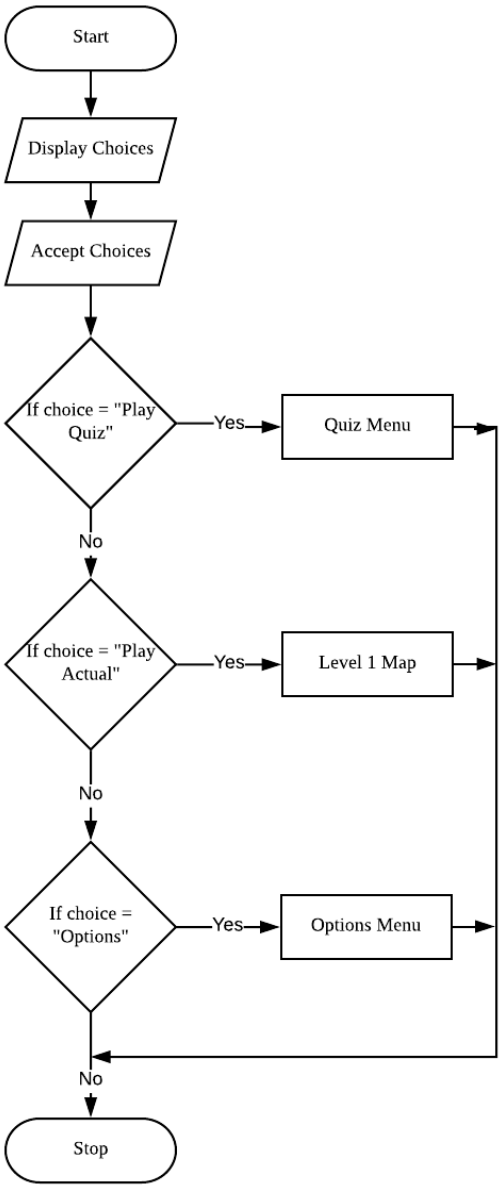

Figure 6. Main Menu
Flowchart representation of menu panel that have three choices to select to proceed to the game and its mechanics.

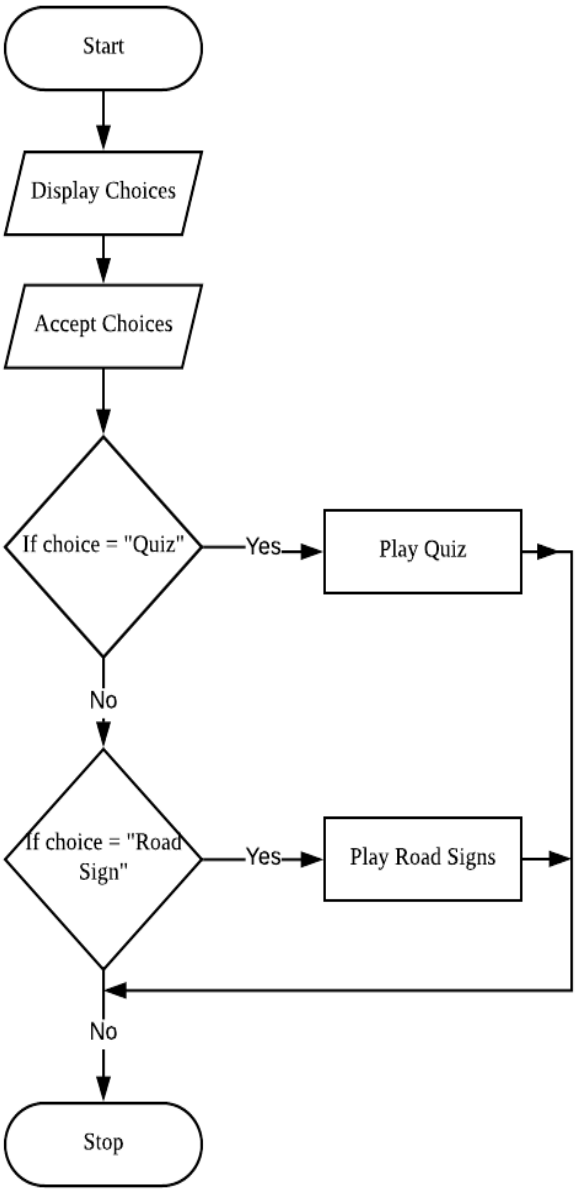

Figure 7. Quiz Game
This flowchart represents the Play Quiz button. There are two choices, Quiz and Road Signs. To be familiarize with rules and road signs players must play this game mode.

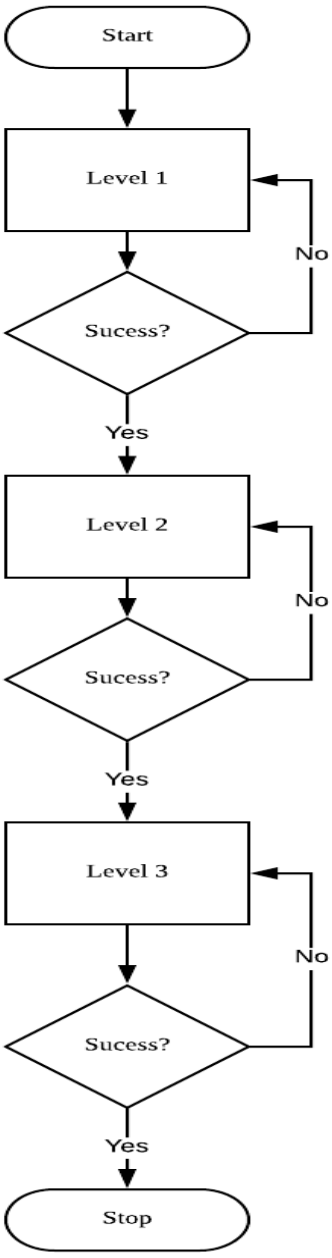

Figure 8. Main Game
This flowchart represents the first gameplay. The player has a choice of taking the tutorial or to not take it. The player will proceed to the preliminary test about road traffic signs and regulations. If player manages to have a score 30 and above the player will proceed to the actual driving, if not the player will take or retake the lessons.

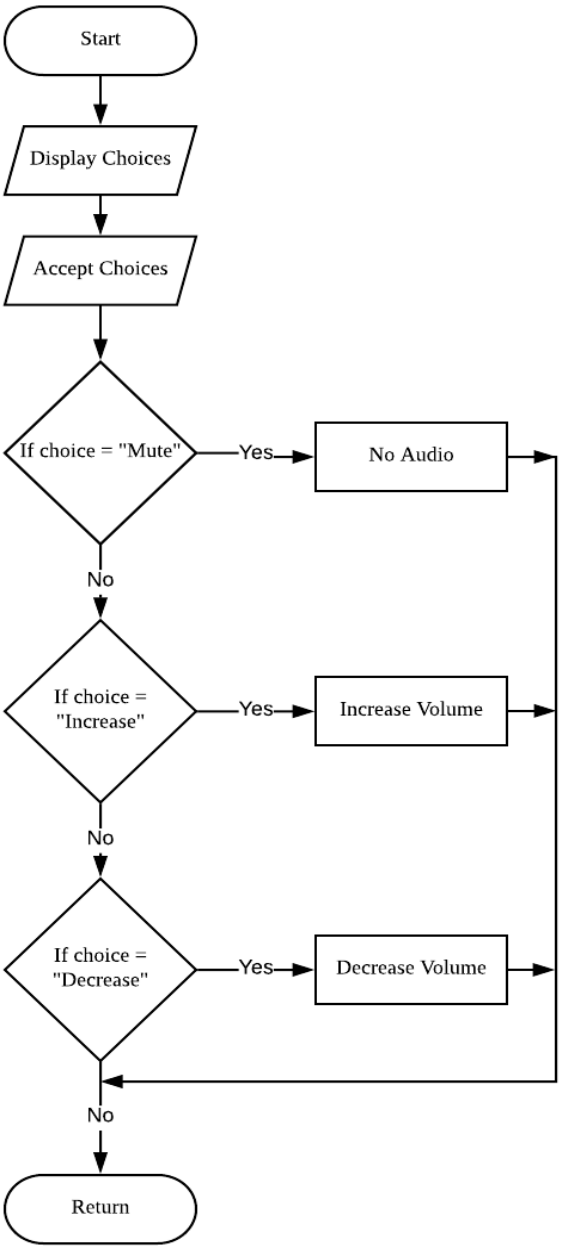

Figure 9. Options
After clicking the Option icon, sound settings will be displayed.

4.0 RESULTS

These are the results of the game, made by the game developers.

Figure 10. Game Icon
This is RoadWyze's game icon, which helps players identify the executable file quickly.

Figure 11. Splash Screen
This is the screen, which is first seen after the player launches the game.

Figure 12. Home Screen
This screen greets the players, where they can proceed to the main menu.

Figure 14. Quiz Main Menu
This screen presents the options for the quiz main menu.

Figure 13. Main Menu
In this screen the player can choose to proceed to the game, tinker around with the options, or quit the game.

Figure 15. Quiz Game
The interface represents the quiz game where the blue balloon pops questions, green box shows the score percentage and the violet texts are the choices.

Figure 16. Road Signs Matching
The interface of this scene represents the Road Signs quiz where you match signs together. There will be a timer and a count on the signs matched.

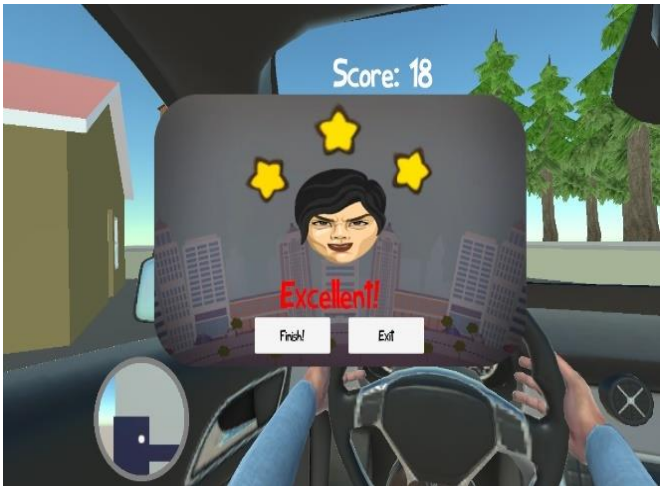

Figure 18. Level Success
This shows that you have completed a level. You may proceed to the next level or exit.

Figure 17. Level 1
The first level of the game. In the interface, you can see the player turning right, and the small circle as the radar or GPS of the game.

Figure 19. Level 2
This is the second level of the game. Dijkstra's algorithm is applied in the GPS.

Figure 20. Level 3
This is the third and the final level of the game. Dijkstra's algorithm was applied in the GPS.

Figure 21. Game Over
Game Over display shows if you have not succeeded the level. This interface shows on other different levels.

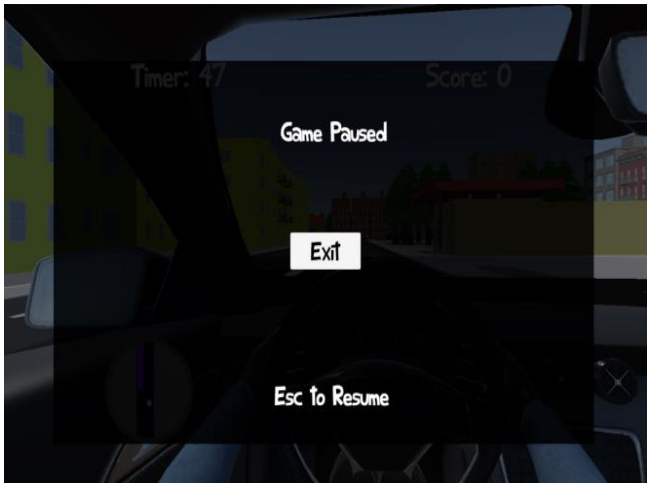

Figure 22. Pause
This is the interface shown when the game is paused. The player can resume anytime using Esc button.

5.0 SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

SUMMARY

RoadWyze is a driving game simulator for the players to understand traffic road signs and symbols and regulations on the road. The game is played with two game modes, which is the Quiz for the player to learn traffic sign and symbol, and Actual Driving for the player to apply his learnings and regulations. Simulation is a method of imitating something relative to reality. Dijkstra's algorithm is a method of finding the shortest and best path of road networks. The game was developed in Unity3D with the help of Microsoft Visual Studio to develop the codes of C# functional scripts.

Game development is the method of creating a game that is compatible on any devices. The development of the game requires a game development life cycle, which included six phases. The developers of

RoadWyze followed all phases for the game project to be accomplished. The testing and debugging parts of developing a game is important because it is easy to find errors and have them debugged right away.

CONCLUSIONS

For the game project has been running, the researchers and game developers would like to conclude the following:

1. RoadWyze is a beneficial tool in educating players on road signs and regulations. It helps in improving their reflexes, memory, and hand-eye coordination.
2. Dijkstra's algorithm is an effective method to plot the shortest path between two points in any given graph. In a driving game such as RoadWyze, it helps add a touch of realism, simulating drivers' tendencies to pick shortcuts.
3. The C# programming language is incredibly useful in game development. Its convenience allowed us to create scripts and functions for RoadWyze without hassle.

RECOMMENDATIONS

The improvement of the game would highly be a benefit for the developers of the game. Never doubt your skills in content making.

As game developers, if an aspiring developer wants to edit the contents of the game, the researchers would highly recommend that it would include, rewards for high score such as new cars, new maps and new avatars. Include definition of traffic symbols in the Road Signs options in the quiz game mode. A more realistic traffic and, if possible, include a pedestrian system. Stricter adherence to traffic rules and regulations (i.e. point deduction per illegal turn). In addition, if you add more gameplay the game would be more interesting and fun for the players.

References

[1] Dunn, F., & Parberry, I. (2015). 3D Math Primer for Graphics and Game Development. Retrieved from: https://tfetimes.com/wp-content/uploads/2015/04/F.Dunn-I.Parberry-3D-Math-Primer-for-Graphics-and-Game-Development.pdf

[2] Ibmua' (2008). Dijkstra's algorithm. Retrieved from: http://www.cse.unt.edu/~tarau/teaching/AnAlgo/Dijkstra's%20algorithm.pdf

[3] List of Traffic Signs and Symbols. (2017). under the Philippine Government. Retrieved from: https://www.affordablecebu.com/load/philippine_government/list_of_traffic_signs_in_the_philippines/5-1-0-30228

[4] LTO Exam Reviewer' (2018). Test Questions & Answers: English Retrieved from: https://www.ltoexamreviewer.com/test-questions-answers-english/

[5] Roads and Maritime Services. (2018). Safety Rules, and Road Rules, Signs. Retrieved from: https://www.rms.nsw.gov.au/roads/safety-rules/road-rules/signs.html

[6] Salce, K. (2018). List of Traffic Signs in the Philippines. Retrieved from: https://www.affordablecebu.com/load/philippine_government/list_of_traffic_signs_in_the_philippines/5-1-0-30228

Appendices

A. Codes

CarController

```
using System;
using UnityEngine;

namespace UnityStandardAssets.Vehicles.Car
{
    internal enum CarDriveType
    {
        FrontWheelDrive,
        RearWheelDrive,
        FourWheelDrive
    }
    internal enum SpeedType
    {
        MPH,
        KPH
    }
    public class CarController : MonoBehaviour
    {
        [SerializeField] private CarDriveType
m_CarDriveType = CarDriveType.FourWheelDrive;
        [SerializeField] private WheelCollider[]
m_WheelColliders = new WheelCollider[4];
        [SerializeField] private GameObject[]
m_WheelMeshes = new GameObject[4];
        [SerializeField] private WheelEffects[]
m_WheelEffects = new WheelEffects[4];
        [SerializeField] private Vector3
m_CentreOfMassOffset;
        [SerializeField] private float
m_MaximumSteerAngle;
        [Range(0, 1)] [SerializeField] private
float m_SteerHelper; // 0 is raw physics , 1 the
car will grip in the direction it is facing
        [Range(0, 1)] [SerializeField] private
float m_TractionControl; // 0 is no traction
control, 1 is full interference
        [SerializeField] private float
m_FullTorqueOverAllWheels;
        [SerializeField] private float
m_ReverseTorque;
        [SerializeField] private float
m_MaxHandbrakeTorque;
        [SerializeField] private float
m_Downforce = 100f;
        [SerializeField] private SpeedType
m_SpeedType;
        [SerializeField] private float m_Topspeed
= 200;
        [SerializeField] private static int
NoOfGears = 5;
        [SerializeField] private float
m_RevRangeBoundary = 1f;
        [SerializeField] private float
m_SlipLimit;
        [SerializeField] private float
m_BrakeTorque;

        private Quaternion[]
m_WheelMeshLocalRotations;
        private Vector3 m_Prevpos, m_Pos;
        private float m_SteerAngle;
        private int m_GearNum;
        private float m_GearFactor;
        private float m_OldRotation;
        private float m_CurrentTorque;
        private Rigidbody m_Rigidbody;
        private const float k_ReversingThreshold
= 0.01f;
        public bool Skidding { get; private set;
}
        public float BrakeInput { get; private
set; }
        public float CurrentSteerAngle{ get {
return m_SteerAngle; }}
        public float CurrentSpeed{ get { return
m_Rigidbody.velocity.magnitude*2.23693629f; }}
        public float MaxSpeed{get { return
m_Topspeed; }}
        public float Revs { get; private set; }
        public float AccelInput { get; private
set; }
        // Use this for initialization
        private void Start()
        {
            m_WheelMeshLocalRotations = new
Quaternion[4];
            for (int i = 0; i < 4; i++)
            {
                m_WheelMeshLocalRotations[i] =
m_WheelMeshes[i].transform.localRotation;
            }

m_WheelColliders[0].attachedRigidbody.centerOfMas
s = m_CentreOfMassOffset;

            m_MaxHandbrakeTorque =
float.MaxValue;

            m_Rigidbody =
GetComponent<Rigidbody>();
            m_CurrentTorque =
m_FullTorqueOverAllWheels -
(m_TractionControl*m_FullTorqueOverAllWheels);
        }

        private void GearChanging()
        {
            float f =
Mathf.Abs(CurrentSpeed/MaxSpeed);
            float upgearlimit = (1/(float)
NoOfGears)*(m_GearNum + 1);
            float downgearlimit = (1/(float)
NoOfGears)*m_GearNum;

            if (m_GearNum > 0 && f <
downgearlimit)
            {
                m_GearNum--;
            }

            if (f > upgearlimit && (m_GearNum <
(NoOfGears - 1)))
            {
```

```
                m_GearNum++;
            }
        }


        // simple function to add a curved bias
towards 1 for a value in the 0-1 range
        private static float CurveFactor(float
factor)
        {
            return 1 - (1 - factor)*(1 - factor);
        }

        // unclamped version of Lerp, to allow
value to exceed the from-to range
        private static float ULerp(float from,
float to, float value)
        {
            return (1.0f - value)*from +
value*to;
        }
           private void CalculateGearFactor()
        {
            float f = (1/(float) NoOfGears);
            // gear factor is a normalised
representation of the current speed within the
current gear's range of speeds.
            // We smooth towards the 'target'
gear factor, so that revs don't instantly snap up
or down when changing gear.
            var targetGearFactor =
Mathf.InverseLerp(f*m_GearNum, f*(m_GearNum + 1),
Mathf.Abs(CurrentSpeed/MaxSpeed));
            m_GearFactor =
Mathf.Lerp(m_GearFactor, targetGearFactor,
Time.deltaTime*5f);
        }

        private void CalculateRevs()
        {
            // calculate engine revs (for display
/ sound)
            // (this is done in retrospect - revs
are not used in force/power calculations)
            CalculateGearFactor();
            var gearNumFactor = m_GearNum/(float)
NoOfGears;
            var revsRangeMin = ULerp(0f,
m_RevRangeBoundary, CurveFactor(gearNumFactor));
            var revsRangeMax =
ULerp(m_RevRangeBoundary, 1f, gearNumFactor);
            Revs = ULerp(revsRangeMin,
revsRangeMax, m_GearFactor);
        }
           public void Move(float steering, float
accel, float footbrake, float handbrake)
        {
            for (int i = 0; i < 4; i++)
            {
                Quaternion quat;
                Vector3 position;

m_WheelColliders[i].GetWorldPose(out position,
out quat);

m_WheelMeshes[i].transform.position = position;

m_WheelMeshes[i].transform.rotation = quat;
            }
            //clamp input values
            steering = Mathf.Clamp(steering, -1,
1);
            AccelInput = accel =
Mathf.Clamp(accel, 0, 1);
            BrakeInput = footbrake = -
1*Mathf.Clamp(footbrake, -1, 0);
            handbrake = Mathf.Clamp(handbrake, 0,
1);
            //Set the steer on the front wheels.
            //Assuming that wheels 0 and 1 are
the front wheels.
            m_SteerAngle =
steering*m_MaximumSteerAngle;
            m_WheelColliders[0].steerAngle =
m_SteerAngle;
            m_WheelColliders[1].steerAngle =
m_SteerAngle;

            SteerHelper();
            ApplyDrive(accel, footbrake);
            CapSpeed();

            //Set the handbrake.
            //Assuming that wheels 2 and 3 are
the rear wheels.
            if (handbrake > 0f)
            {
                var hbTorque =
handbrake*m_MaxHandbrakeTorque;
                m_WheelColliders[2].brakeTorque =
hbTorque;
                m_WheelColliders[3].brakeTorque =
hbTorque;
            }

            CalculateRevs();
            GearChanging();

            AddDownForce();
            CheckForWheelSpin();
            TractionControl();
        }
        private void CapSpeed()
        {
            float speed =
m_Rigidbody.velocity.magnitude;
            switch (m_SpeedType)
            {
                case SpeedType.MPH:

                    speed *= 2.23693629f;
                    if (speed > m_Topspeed)
                        m_Rigidbody.velocity =
(m_Topspeed/2.23693629f) *
m_Rigidbody.velocity.normalized;
                    break;

                case SpeedType.KPH:
                    speed *= 3.6f;
                    if (speed > m_Topspeed)
                        m_Rigidbody.velocity =
(m_Topspeed/3.6f) *
m_Rigidbody.velocity.normalized;
                    break;
            }
        }
        private void ApplyDrive(float accel,
float footbrake)
```

```
        {

            float thrustTorque;
            switch (m_CarDriveType)
            {
                case CarDriveType.FourWheelDrive:
                    thrustTorque = accel *
(m_CurrentTorque / 4f);
                    for (int i = 0; i < 4; i++)
                    {

m_WheelColliders[i].motorTorque = thrustTorque;
                    }
                    break;
                case
CarDriveType.FrontWheelDrive:
                    thrustTorque = accel *
(m_CurrentTorque / 2f);

m_WheelColliders[0].motorTorque =
m_WheelColliders[1].motorTorque = thrustTorque;
                    break;
                case CarDriveType.RearWheelDrive:
                    thrustTorque = accel *
(m_CurrentTorque / 2f);

m_WheelColliders[2].motorTorque =
m_WheelColliders[3].motorTorque = thrustTorque;
                    break;
            }

            for (int i = 0; i < 4; i++)
            {
                if (CurrentSpeed > 5 &&
Vector3.Angle(transform.forward,
m_Rigidbody.velocity) < 50f)
                {

m_WheelColliders[i].brakeTorque =
m_BrakeTorque*footbrake;
                }
                else if (footbrake > 0)
                {

m_WheelColliders[i].brakeTorque = 0f;

m_WheelColliders[i].motorTorque = -
m_ReverseTorque*footbrake;
                }
            }
        }

        private void SteerHelper()
        {
            for (int i = 0; i < 4; i++)
            {
                WheelHit wheelhit;

m_WheelColliders[i].GetGroundHit(out wheelhit);
                if (wheelhit.normal ==
Vector3.zero)
                    return; // wheels arent on
the ground so dont realign the rigidbody velocity
            }
            // this if is needed to avoid gimbal
lock problems that will make the car suddenly
shift direction
            if (Mathf.Abs(m_OldRotation -
transform.eulerAngles.y) < 10f)
            {
                var turnadjust =
(transform.eulerAngles.y - m_OldRotation) *
m_SteerHelper;
                Quaternion velRotation =
Quaternion.AngleAxis(turnadjust, Vector3.up);
                m_Rigidbody.velocity =
velRotation * m_Rigidbody.velocity;
            }
            m_OldRotation =
transform.eulerAngles.y;
        }
        // this is used to add more grip in
relation to speed
        private void AddDownForce()
        {

m_WheelColliders[0].attachedRigidbody.AddForce(-
transform.up*m_Downforce*

m_WheelColliders[0].attachedRigidbody.velocity.ma
gnitude);
        }
        // checks if the wheels are spinning and
is so does three things
        // 1) emits particles
        // 2) plays tiure skidding sounds
        // 3) leaves skidmarks on the ground
        // these effects are controlled through
the WheelEffects class
        private void CheckForWheelSpin()
        {
            // loop through all wheels
            for (int i = 0; i < 4; i++)
            {
                WheelHit wheelHit;

m_WheelColliders[i].GetGroundHit(out wheelHit);

                // is the tire slipping above the
given threshhold
                if
(Mathf.Abs(wheelHit.forwardSlip) >= m_SlipLimit
|| Mathf.Abs(wheelHit.sidewaysSlip) >=
m_SlipLimit)
                {

m_WheelEffects[i].EmitTyreSmoke();

                    // avoiding all four tires
screeching at the same time
                    // if they do it can lead to
some strange audio artefacts
                    if (!AnySkidSoundPlaying())
                    {

m_WheelEffects[i].PlayAudio();
                    }
                    continue;
                }
                // if it wasnt slipping stop all
the audio
                if
(m_WheelEffects[i].PlayingAudio)
                {

m_WheelEffects[i].StopAudio();
                }
                // end the trail generation
```

```
                m_WheelEffects[i].EndSkidTrail();
            }
        }

        // crude traction control that reduces
the power to wheel if the car is wheel spinning
too much
        private void TractionControl()
        {
            WheelHit wheelHit;
            switch (m_CarDriveType)
            {
                case CarDriveType.FourWheelDrive:
                    // loop through all wheels
                    for (int i = 0; i < 4; i++)
                    {

m_WheelColliders[i].GetGroundHit(out wheelHit);


AdjustTorque(wheelHit.forwardSlip);
                    }
                    break;

                case CarDriveType.RearWheelDrive:

m_WheelColliders[2].GetGroundHit(out wheelHit);

AdjustTorque(wheelHit.forwardSlip);


m_WheelColliders[3].GetGroundHit(out wheelHit);

AdjustTorque(wheelHit.forwardSlip);
                    break;

                case
CarDriveType.FrontWheelDrive:

m_WheelColliders[0].GetGroundHit(out wheelHit);

AdjustTorque(wheelHit.forwardSlip);


m_WheelColliders[1].GetGroundHit(out wheelHit);

AdjustTorque(wheelHit.forwardSlip);
                    break;
            }
        }
        private void AdjustTorque(float
forwardSlip)
        {
            if (forwardSlip >= m_SlipLimit &&
m_CurrentTorque >= 0)
            {
                m_CurrentTorque -= 10 *
m_TractionControl;
            }
            else
            {
                m_CurrentTorque += 10 *
m_TractionControl;
                if (m_CurrentTorque >
m_FullTorqueOverAllWheels)
                {
                    m_CurrentTorque =
m_FullTorqueOverAllWheels;
                }
            }
        }


        private bool AnySkidSoundPlaying()
        {
            for (int i = 0; i < 4; i++)
            {
                if
(m_WheelEffects[i].PlayingAudio)
                {
                    return true;
                }
            }
            return false;
        }
    }
}
```

CollisionStopLevel1 (Level 1)

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

public class CollisionStopLevel1 : MonoBehaviour
{
    [SerializeField] private Image threeStars,
twoStars, oneStar, noStar;
    public GameObject arrow1, arrow2, arrow3,
arrow4, arrow5, arrow6, roundabout, arrow7,
arrow8, arrow9, arrow10, arrow11, arrow12,
arrow13, arrow14, arrow15, arrow16, arrow17,
arrow18, arrow19, arrow20, waypointparking;
    public static int count;
    private Rigidbody rb;
    public Text countText;
    public Text result;
    public Text subtitles;
    public Text guide;
    public GameObject player;
    public GameObject cam;
    public Button m_button1, m_button2,
m_button3;
    public Text button01;
    public Text button02;
    public Text button03;
    float timer = 2.0f;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        count = 0;
        SetCountText();

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;

        subtitles.text = "Hi, this is Drvina,
your AI driving instructor for Roadwyze Drivng
Simulator. Through out this day's session you'll
be learning acceleration, decceleration, and
basic parking. Please bare with me as I teach you
```

```
the step by step process. For the sake of
inquiry, I am very strict, so don't try to graze
this car just do as I say and you'll be fine.";
        guide.text = "Press space to continue.";
    }

    private void Update()
    {
        timer -= Time.deltaTime;
    }
      void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Pick
Up"))
        {
            other.gameObject.SetActive(false);
            count = count + 20;
            subtitles.text = "Keep following the
Arrows.";
            SetCountText();
            arrow7.SetActive(true);
        }

        if
(other.gameObject.CompareTag("Destination"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            subtitles.text = "Follow the
Arrows.";
            arrow1.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination2"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow2.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination3"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow3.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination4"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow4.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination5"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow5.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination6"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            subtitles.text = "Park on the marked
spot.";
            arrow6.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination7"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            subtitles.text = "Collect the Round-
A-Bout Sign.";
            roundabout.SetActive(true);

        }
        if
(other.gameObject.CompareTag("Destination8"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow8.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination9"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow9.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination10"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow10.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination11"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow11.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination12"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow12.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination13"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow13.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination14"))
        {
            other.gameObject.SetActive(false);
```

```
            count = count + 10;
            SetCountText();
            arrow14.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination15"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow15.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination16"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow16.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination17"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow17.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination18"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow18.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination19"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow19.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination20"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            arrow20.SetActive(true);
        }
        if
(other.gameObject.CompareTag("Destination21"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            subtitles.text = string.Empty;
            SetCountText();
            waypointparking.SetActive(true);
        }
        if
(other.gameObject.CompareTag("WaypointPark"))
        {
            other.gameObject.SetActive(false);
            threeStars.enabled = true;
            result.text = "Excellent!";

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;

cam.GetComponent<CameraFollow>().enabled = false;

            m_button3.gameObject.SetActive(true);
            button03.text = "Proceed";
            m_button2.gameObject.SetActive(true);
            button02.text = "Exit";
        }
    }

    private void OnCollisionEnter(Collision
other)
    {
        if
(other.gameObject.CompareTag("TrafficCar") ||
other.gameObject.CompareTag("Collide1"))
        {
            noStar.enabled = true;
            result.text = "Game Over!";

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;

cam.GetComponent<CameraFollow>().enabled = false;

            m_button1.gameObject.SetActive(true);
            button01.text = "Restart";
            m_button2.gameObject.SetActive(true);
            button02.text = "Exit";
        }
    }

    void SetCountText()
    {
        countText.text = "Score: " +
count.ToString();
    }
    void addCount()
    {
        count = count + 1;
    }
}
```

CollisionStopLevel2 (Level 2)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class CollisionStopLevel2 : MonoBehaviour
{
    [SerializeField] private Image threeStars,
twoStars, oneStar, noStar;
    public static int count;
    private Rigidbody rb;
    public Text countText;
    public Text result;
    public Text subtitles;
    public Text guide;
```

```
    public GameObject player;
    public GameObject cam;
    public Button m_button1, m_button2,
m_button3;
    public Text button01;
    public Text button02;
    public Text button03;
    float Timer = 60.0f;
    public Text timer;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        count = 0;
        SetCountText();

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;

player.GetComponent<CollisionStopLevel2>().enable
d = false;

       subtitles.text = "Time Attack! Now, for
the second stage of your course, you'll have 1
minute to get to your destination, at best.
You'll still be scored even when you arrived in
more than the time.Note the traffic signs along
the way and be aware of your surroundings.This
session will teach you how to arrive at your
destination fast and safe.Good luck and let's be
on our way.";
       guide.text = "Press space to continue.";
    }
    private void Update()
    {
        Timer -= Time.deltaTime;
        timer.text = "Timer: " +
Timer.ToString("F0");

    }
    void OnTriggerEnter(Collider other)
    {
        if
(other.gameObject.CompareTag("Destination"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();

            if (Timer >= 30)
            {
                threeStars.enabled = true;

                result.text = "Excellent!";

m_button3.gameObject.SetActive(true);
                button03.text = "Proceed";

m_button2.gameObject.SetActive(true);
                button02.text = "Exit";
                timer.text = "Timer: 0";
            }

            if (Timer <= 30 && Timer >= 10)
            {
                twoStars.enabled = true;

                result.text = "Great";

m_button3.gameObject.SetActive(true);
                button03.text = "Proceed";

m_button2.gameObject.SetActive(true);
                button02.text = "Exit";
                timer.text = "Timer: 0";
            }
            if (Timer < 10)
            {
                threeStars.enabled = true;

                result.text = "Good";

m_button1.gameObject.SetActive(true);
                button01.text = "Restart";

m_button2.gameObject.SetActive(true);
                button02.text = "Exit";
                timer.text = "Timer: 0";
            }
            if (Timer == 0)
            {
                noStar.enabled = true;
                result.text = "Game Over!";

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;

cam.GetComponent<CameraFollow>().enabled = false;


m_button1.gameObject.SetActive(true);
                button01.text = "Restart";

m_button2.gameObject.SetActive(true);
                button02.text = "Exit";
                timer.text = "Timer: 0";
            }
        }
    }

    private void OnCollisionEnter(Collision
other)
    {
        if
(other.gameObject.CompareTag("TrafficCar") ||
other.gameObject.CompareTag("Collide1"))
        {
            noStar.enabled = true;
            result.text = "Game Over!";

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;

cam.GetComponent<CameraFollow>().enabled = false;

            m_button1.gameObject.SetActive(true);
            button01.text = "Restart";
            m_button2.gameObject.SetActive(true);
            button02.text = "Exit";
```

```
            timer.text = "Timer: 0";
        }
    }
    void SetCountText()
    {
        countText.text = "Score: " +
count.ToString();
    }
    void addCount()
    {
        count = count + 1;
    }
}
```

CollisionStop (Level 3)

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

public class CollisionStop : MonoBehaviour {

    [SerializeField] private Image threeStars,
twoStars, oneStar, noStar;
    public static int count;
    private Rigidbody rb;
    public Text countText;
    public Text result;
    public GameObject player;
    public GameObject cam;
    public Button m_button1, m_button2,
m_button3;
    public Text button01;
    public Text button02;
    public Text button03;
    public Text subtitles;
    public Text guide;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        count = 0;
        SetCountText();


player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;

        subtitles.text = "For the final stage of
the course, I am requiring you to gather all
traffic signs and symbols along the way. This
session will teach you each symbol's conveying
information and instruction, which is a driver
must be aware of on the road.";
        guide.text = "Press space to continue.";
    }



    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Pick
Up"))
        {
            other.gameObject.SetActive(false);
            addCount();
            SetCountText();
        }

        if
(other.gameObject.CompareTag("Destination"))
        {
            other.gameObject.SetActive(false);
            count = count + 10;
            SetCountText();
            if (count == 18)
            {
                threeStars.enabled = true;
                result.text = "Excellent!";

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;


m_button3.gameObject.SetActive(true);
                button03.text = "Finish!";

m_button2.gameObject.SetActive(true);
                button02.text = "Exit";
            }
            else if (count < 18 && count >= 11)
            {
                twoStars.enabled = true;
                result.text = "Great!";

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;


m_button1.gameObject.SetActive(true);
                button01.text = "Restart";

m_button2.gameObject.SetActive(true);
                button02.text = "Exit";

            }
            else if (count <= 10)
            {
                oneStar.enabled = true;
                result.text = "Good!";

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;


m_button1.gameObject.SetActive(true);
                button01.text = "Restart";

m_button2.gameObject.SetActive(true);
                button02.text = "Exit";
            }
        }
    }
```

```
    private void OnCollisionEnter(Collision
other)
    {
        if
(other.gameObject.CompareTag("TrafficCar") ||
other.gameObject.CompareTag("Collide1"))
        {
            noStar.enabled = true;
            result.text = "Game Over!";

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarController>().enabled = false;

player.GetComponent<UnityStandardAssets.Vehicles.
Car.CarUserControl>().enabled = false;

cam.GetComponent<CameraFollow>().enabled = false;

            m_button1.gameObject.SetActive(true);
            button01.text = "Restart";
            m_button2.gameObject.SetActive(true);
            button02.text = "Exit";
        }
    }

    void SetCountText()
    {
        countText.text = "Score: " +
count.ToString();
    }

    void addCount()
    {
        count = count + 1;
    }
}
```

Textcontrol

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

public class textcontrol : MonoBehaviour {

    //Quetions
    public static List<string> questions = new
List<string>() { "Signs that are triangular and
\nwith red-colored border are \ncalled?", "Signs
that are rounded, inverted \ntriangle, or
octagonal and with \nred-colored borders are
called?", "Flashing yellow light means?", "The
minimum distance away from \nthe vehicle you are
following is?", "On a two lane road,
overtaking\nis only allowed only at the?", "When
driving at night, you should?", "A single solid
yellow or white\nline means?", "Signs that's
rounded, rectangular \nwith white and blue
background \nare called?", "The penalty for
overcharging or\nundercharging of authorized
\nrates for first offense is?", "On wet road you
must?", "To obtain one’s driver’s license,\none
must be at least?", "Should a pre-trip
inspection\nbe completed?" };
    //Answers
    List<string> correctAnswer = new
List<string>() { "3", "1", "2", "3", "4", "2",
"1", "3", "4", "2", "1", "4" };
    //To avoid repeat
    List<int> prevQuestions = new List<int>() { -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1};

    //Declaration

    public int questionNumber = 0;

    public Transform resultObj;

    public Transform scoreObj;

    public Transform procObj;

    public Transform retrObj;

    public Transform fiOBj;

    public Transform seOBj;

    public Transform thOBj;

    public Transform foOBj;

    public Transform proc;

    public Transform smile;

    public Transform retr;

    public TextMesh questObj;

    public double totalCorrect = 0;

    public double totalQuestions = 12;

    public double scorePer = 0;

    public static string selectedAnswer;

    public static string choiceSelected = "n";

    public static int randQuestion = -1 ;

    static double total1;

   // public GameObject CameraFolObj;


    // Use this for initialization
    void Start()
    {
        //GetComponent<TextMesh> ().text =
questions [0];
    }

        // Update is called once per frame
        void Update () {

        if (totalQuestions > 0)
        {
            scorePer = (totalCorrect /
totalQuestions) * 100;
            total1 = scorePer;
        }

        string s = scorePer.ToString("f2");
```

```
        scoreObj.GetComponent<TextMesh>().text =
"Score: " + s + "%";

        //Proceed button
        if (totalQuestions == 12)
        {
            fiOBj.gameObject.SetActive(false);
            seOBj.gameObject.SetActive(false);
            thOBj.gameObject.SetActive(false);
            foOBj.gameObject.SetActive(false);

            if (scorePer >= 75)
            {

procObj.gameObject.SetActive(true);
                proc.gameObject.SetActive(true);
                questObj.text = "Congratulations,
you passed!!";
                smile.gameObject.SetActive(true);
            }
            else
            {

procObj.gameObject.SetActive(false);
                proc.gameObject.SetActive(false);
                questObj.text = "You failed!!
Retake this quiz.";

retrObj.gameObject.SetActive(true);
                retr.gameObject.SetActive(true);
            }

        }
        else
        {
            procObj.gameObject.SetActive(false);
        }

        //Randomize question
        if(randQuestion == -1)
        {
            randQuestion = Random.Range(0, 12);
            for(int i = 0; i<12; i++)
            {
                if(randQuestion !=
prevQuestions[i])
                {

                }
                else
                {
                    randQuestion = -1;
                }
            }
        }
        if (randQuestion > -1)
        {
            GetComponent<TextMesh>().text =
questions[randQuestion];
            prevQuestions[questionNumber] =
randQuestion;
        }

        //To check if correct or incorrect
        if (choiceSelected == "y")
        {
            choiceSelected = "n";
            totalQuestions += 1 ;
            questionNumber += 1 ;

            if(correctAnswer[randQuestion] ==
selectedAnswer)
            {
                totalCorrect += 1;
                resultObj.GetComponent<TextMesh>
().text = "Correct.";
                textcontrol.randQuestion = -1;

            }

            else

            {

resultObj.GetComponent<TextMesh>().text =
"Incorrect!!";
                textcontrol.randQuestion = -1;
            }
        }
        }
}
```

GameController

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

public class GameController : MonoBehaviour {

    [SerializeField]
    private Sprite bgImage;

    public Sprite[] puzzles;

    public List<Sprite> gamePuzzles = new
List<Sprite>();

    public List<Button> btns = new
List<Button>();

    private bool firstGuess, secondGuess;

    private int countGuesses, correctGuesses,
gameGuesses;
    private int firstGIndex, secondGIndex;

    private string firstGPuzz, secondGPuzz;

    public static int match;

    public GameObject next;
    public GameObject medRetry;

    public Text subtitles;
    public Text guide;
    public Text bot; //message when the game is
finished
    public Text matchText;
    public GameObject tim;
    float Timer = 60.0f;
    public Text timer;
    public GameObject gc;
    static int total2;

    private void Update()
```

```
    {
        Timer -= Time.deltaTime;
        timer.text = "Timer: " +
Timer.ToString("F0");

        if (Timer < 0)
        {
            Timer = 0;

            if (Timer == 0)
            {
                tim.gameObject.SetActive(false);
                next.gameObject.SetActive(false);

medRetry.gameObject.SetActive(true);

gc.GetComponent<GameController>().enabled =
false;
            }
        }
    }

    private void Awake()
    {
        puzzles =
Resources.LoadAll<Sprite>("Sprites");
    }

    void Start()
    {
        match = 0;
        GetButtons(); //calling the buttons
        AddListeners(); //this should be below
because you can't call the buttons if there isn't
one
        AddGamePuzzles(); //calling addgamepuzzle
        Shuffle(gamePuzzles); //calling shuffle

        gameGuesses = gamePuzzles.Count / 2;

        subtitles.text = "Match all signage
before the time runs out!";
        guide.text = "Press space to continue.";

        bot.gameObject.SetActive(false);
        next.gameObject.SetActive(false);
        medRetry.gameObject.SetActive(false);

    }

    void GetButtons()
    {
        GameObject[] objects =
GameObject.FindGameObjectsWithTag("PuzzleButton")
;

        for(int i = 0; i < objects.Length; i++)
        {

btns.Add(objects[i].GetComponent<Button>() );
            btns[i].image.sprite = bgImage;
        }
    }

    void AddGamePuzzles()
    {
        int looper = btns.Count;
        int index = 0;

        for (int i = 0; i < looper; i++)
        {
            if(index == looper / 2)
            {
                index = 0;
            }
            gamePuzzles.Add(puzzles[index]);
            index++;
        }
    }

    void AddListeners()
    {
        //one sample to process the list
        /*for (int i = 0; i < btns.Count; i++) */

        //Another sample and the best way to
process the list
        foreach (Button btn in btns)
        {
            btn.onClick.AddListener( () =>
PickAPuzzle() );
        }
    }

    public void PickAPuzzle()
    {
        string name =
UnityEngine.EventSystems.EventSystem.current.curr
entSelectedGameObject.name;

        //        Debug.Log("You're clicking a
button named" + name);

        if(!firstGuess)
        {
            firstGuess = true;

            firstGIndex =
int.Parse(UnityEngine.EventSystems.EventSystem.cu
rrent.currentSelectedGameObject.name); //to
convert string to int

            firstGPuzz =
gamePuzzles[firstGIndex].name; //to match with
second guess

            btns[firstGIndex].image.sprite =
gamePuzzles[firstGIndex];
        }
        else if (!secondGuess)
        {
            secondGuess = true;

            secondGIndex =
int.Parse(UnityEngine.EventSystems.EventSystem.cu
rrent.currentSelectedGameObject.name);

            secondGPuzz =
gamePuzzles[secondGIndex].name; //to match with
first guess

            btns[secondGIndex].image.sprite =
gamePuzzles[secondGIndex];


StartCoroutine(CheckIfThePuzzlesMatch());

            countGuesses++;
```

```
        }
    }

    IEnumerator CheckIfThePuzzlesMatch()
    {
        yield return new WaitForSeconds(.2f);
//waiting time

        if (firstGPuzz == secondGPuzz)
        {
            yield return new WaitForSeconds(.2f);

            btns[firstGIndex].interactable =
false;
            btns[secondGIndex].interactable =
false;

            btns[firstGIndex].image.color = new
Color(0,0,0,0);
            btns[secondGIndex].image.color = new
Color(0, 0, 0, 0);

            CheckIfTheGameIsFinished();
        }
        else
        {
            yield return new WaitForSeconds(.2f);

            btns[firstGIndex].image.sprite =
bgImage;
            btns[secondGIndex].image.sprite =
bgImage;

            Timer -= 3;
        }

        yield return new WaitForSeconds(.2f);

        firstGuess = secondGuess = false;
    }

    void CheckIfTheGameIsFinished() //in console,
tells if the game is finished and it's time
    {
        correctGuesses++;
        match += 1;
        matchCount();

        if(correctGuesses == gameGuesses)
        {
            if (Timer >= 30 && match >= 4)
            {
                bot.gameObject.SetActive(true);
                bot.text = "Great! Click Next";
                tim.gameObject.SetActive(false);
                next.gameObject.SetActive(true);
                total2 = 75;
            }

            else if (Timer < 30 || Timer >= 20 &&
match >= 5)
            {
                bot.gameObject.SetActive(true);
                bot.text = "Good! Click Next";
                tim.gameObject.SetActive(false);
                next.gameObject.SetActive(true);
                total2 = 50;
            }

            else if (Timer < 20 || Timer >= 10 &&
match >= 6)
            {
                bot.gameObject.SetActive(true);
                bot.text = "Nice! Click Next";
                tim.gameObject.SetActive(false);
                next.gameObject.SetActive(true);
                total2 = 25;
            }
        }
    }

    void Shuffle(List<Sprite> list)
    {
        for(int i = 0; i < list.Count; i++)
        {
            Sprite temp = list[i]; //getting
reference using index i
            int randomIndex =
Random.Range(i,list.Count); // getting a random
index between i and list.count

            list[i] = list[randomIndex];
            list[randomIndex] = temp;
        }
    }

    void matchCount()
    {
        matchText.text = "Matches: " +
match.ToString();
    }
}
```

FirstScene

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

public class FirstScene : MonoBehaviour {

    public void PlayGame()
    {

SceneManager.LoadScene(SceneManager.GetActiveScen
e().buildIndex + 1);
    }

    public void PlayActual()
    {
        SceneManager.LoadScene(6);
    }

        public void QuitGame()
        {
                Debug.Log ("Application Quit");
                Application.Quit ();
        }
}
```

ButtonScript

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

```
public class ButtonsScript : MonoBehaviour {

        public void RestartScene1()
    {
        SceneManager.LoadScene(6);
    }

    public void RestartScene2()
    {
        SceneManager.LoadScene(7);
    }

    public void RestartScene3()
    {
        SceneManager.LoadScene(8);
    }

    public void FinishScene()
    {
        SceneManager.LoadScene(1);
    }

    public void QuitGame()
    {
        SceneManager.LoadScene(1);
    }

    public void ProceedScene()
    {

SceneManager.LoadScene(SceneManager.GetActiveScen
e().buildIndex + 1);
    }
}
```

CameraFollow

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

public class CameraFollow : MonoBehaviour {

    public float CameraMoveSpeed = 120.0f;
    public GameObject CameraFollowObj;
    Vector3 FollowPOS;
    public float clampAngle = 80.0f;
    public float inputSensitivity = 150.0f;
    public GameObject CameraObj;
    public GameObject PlayerObj;
    public float camDistanceXToPlayer;
    public float camDistanceYToPlayer;
    public float camDistanceZToPlayer;
    public float mouseX;
    public float mouseY;
    public float finalInputX;
    public float finalInputZ;
    public float smoothX;
    public float smoothY;
    private float rotY = 0.0f;
    private float rotX = 0.0f;
    public float senX = 15.0f;
    public float minx = -90.0f;
    public float maxx = 90.0f;
    float rotationX = 0.0f;

        void Update () {

        float inputX =
Input.GetAxis("RightStickHorizontal");
        float inputZ =
Input.GetAxis("RightStickVertical");
        mouseX = Input.GetAxis("Mouse X");
       // mouseY = Input.GetAxis("Mouse Y");
        finalInputX = inputX + mouseX;
        finalInputZ = inputZ + mouseY;

        rotY += finalInputX * inputSensitivity *
Time.deltaTime;
        rotX += finalInputZ * inputSensitivity *
Time.deltaTime;

        rotX = Mathf.Clamp(rotX, -clampAngle,
clampAngle);

        rotationX += Input.GetAxis("Mouse X") *
senX;
        rotationX = Mathf.Clamp(rotationX, minx,
maxx);

        transform.localEulerAngles = new
Vector3(0, rotationX, 0);

    }

    void LateUpdate() {
        CameraUpdater();

    }

    void CameraUpdater() {
        Transform target =
CameraFollowObj.transform;
        float step = CameraMoveSpeed *
Time.deltaTime;
        transform.position =
Vector3.MoveTowards(transform.position,
target.position, step);
    }

}
```

SAP2DManager (Algorithm)

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using SAP2D;

namespace SAP2D {

        [AddComponentMenu("Pathfinding 2D/SAP2D
Manager")]
        public class SAP2DManager :
MonoBehaviour {

                private static SAP2DManager
Singleton;

                public static SAP2DManager
singleton{
                        get{
```

```
                                if(Singleton ==
null){
        Singleton =
FindObjectOfType<SAP2DManager>();

        if(Singleton == null){
        Singleton = new
GameObject("SAP2D").AddComponent<SAP2DManager>();
                                        }
                                }
                                return
Singleton;
                        }
                }

                public GridGraph grid = new
GridGraph ();

                public bool UsePhysics2D = true;
//use 2D physics (calculate 2D colliders)
                public List<string>
IgnoreCollisionTags = new List<string>(); //array
of collider tags that will be ignored in the
system colculations

                public UserObstaclesData
ObsUserData = new UserObstaclesData();

                //find path from point A to
point B
                public Vector2[]
FindPath(Vector2 from, Vector2 to,
PathfindingConfig2D config){

                        Clear ();

                        List<Tile> OpenList =
new List<Tile> ();   //list of tiles to check
                        List<Tile> ClosedList =
new List<Tile> (); //list of tiles to ignore

                        Tile CurrentTile =
grid.GetTileFromWorldPosition (from); //current
check tile, that to beginning of path searching
equal start tile
                        Tile toTile =
grid.GetTileFromWorldPosition (to);        //end
tile

                        if (!toTile.isWalkable)
                                return null;


        OpenList.Add(CurrentTile);
                        CurrentTile.State =
Tile.listState.Open;

                        while (toTile.State !=
Tile.listState.Close) {


        if(OpenList.Count == 0){

        Debug.Log("Path not found!");
                                        return
null;
                                }

                                OpenList.Remove
(CurrentTile); //delete current from the open
list

                                ClosedList.Add
(CurrentTile);  //add current tile to closed list

        CurrentTile.State =
Tile.listState.Close;


        CalculateTilesAround (CurrentTile,
toTile, OpenList, config); //calculate tile
parameters around current tile

                                int minF =
int.MaxValue; //minimum F value

                                //searching tile
with minimum F value in open list
                                foreach (Tile t
in OpenList) {

                                        if(t.F <
minF){

        minF = t.F;

        //current tile equal tile with minimum F
value

        CurrentTile = t;
                                        }
                                }
                        }
                        return PathRecovery
(from, to);
                }

                //calculate tile parameters
around ceterTile
                void CalculateTilesAround(Tile
centerTile, Tile toTile, List<Tile> OpenList,
PathfindingConfig2D config){

                        //loop that ckecks tiles
around the central tile
                        for (int y =
centerTile.y - 1; y <= centerTile.y + 1; y++) {
                                for(int x =
centerTile.x - 1; x <= centerTile.x + 1; x++){

                                        //loop
values should not be greater than grid size
                                        if(x >=
0 && x < grid.GridWidth && y >= 0 && y <
grid.GridHeight){


        Tile current = grid.tile[x,y];


        if(!config.DiagonalMovement)

        if(x != centerTile.x && y !=
centerTile.y)
```

```
        continue;


        if(!current.isWalkable ||
!ConnerManager(centerTile, current, config))

        continue;


        //current checked tile should not to be
in the closed list

        if(current.State !=
Tile.listState.Close){



        //if current tile is not in any list

        if(current.State ==
Tile.listState.Empty){



                current.ParentTile = centerTile;
//set central tile as parent tile for current
tile



                CalculateTileValues(current,
centerTile, toTile);



                //add current tile to open list

                OpenList.Add(current);

                current.State =
Tile.listState.Open;

        }



        //if tile is already in open list, we
should check the shortest path across this tile

        //compare the already calculated G value
and new G value

        if(current.State ==
Tile.listState.Open){



                //save the already calculated
values of G, H and ParentTile

                int oldG = current.G;

                int oldH = current.H;

                Tile oldParentTile =
current.ParentTile;

                //calculate new values of G and
H

                current.ParentTile = centerTile;

                CalculateTileValues(current,
centerTile, toTile);



                //compare old values and new
values

                //if the new value of G is
greater than the old G value, then the path is
not shorter

                if(current.G >= oldG){



                        //return old values

                        current.ParentTile =
oldParentTile;

                        current.G = oldG;

                        current.H = oldH;

                        current.F = oldH + oldG;

                }
                                        }

        }
                                        }
                                }
                        }
                }

                //calculate tile parameters
                void CalculateTileValues(Tile
tile, Tile centerTile, Tile toTile){  //14 10 14
if current tile coordinate equal the central tile
coordinates
                        int G = 0;
//10 ** 10  equal the central tile coordinates
(**),
                        if (tile.x ==
centerTile.x || tile.y == centerTile.y)
//14 10 14  then G of current tile = 10
                                G = 10;
                        else
//if both coordinates of the current tile are not
equal to central tile cordinates,
                                G = 14;
//then G of current tile = 14
                        G += tile.ParentTile.G;
//summarize parent tile G value and setted G
                        //calculate distance
between current and end tiles, ignoring vertical
movement (calculations)
                        int H =
(Mathf.Abs(tile.x - toTile.x) + Mathf.Abs(tile.y
- toTile.y))*10;  // (|x1 - x2| + |y1 - y2|)*10
(x1, y1) - current tile cordinates
```

```
                        int F = G + H;
//                            (x2, y2) - end
tile cordinates

                        tile.G = G;
                        tile.H = H;
                        tile.F = F;
                }

                //path recovering
                //the path is recovered
beginning end tile, move from the parent tile to
parent tile, to the starting tile
                Vector2[] PathRecovery(Vector2
from, Vector2 to){

                        Tile fromTile =
grid.GetTileFromWorldPosition (from);
                        Tile current =
grid.GetTileFromWorldPosition (to);

                        List<Vector2> path = new
List<Vector2> ();

                        while (current !=
fromTile) {


        path.Add(current.WorldPosition);

                                current =
current.ParentTile;
                        }

                        path.Reverse ();

                        return path.ToArray ();
                }

                //returns false if angular
movement or angle cut is impossible
                bool ConnerManager(Tile
centerTile, Tile current, PathfindingConfig2D
config){

                        bool canWalk = true;

                        //if rule the catting
conners is off, then forbid cutting corner
C - centerTile * - current X - unwalkable tile
                        if
(!config.IgnoreCorners) {

                                if (centerTile.x
+ 1 == current.x && centerTile.y + 1 ==
current.y) { //0 X *



                    //0 C 0
                                        if
(!grid.tile [current.x-1,current.y].isWalkable) {
//O O O

        canWalk = false;
                                        }
                                }

                                if (centerTile.x
- 1 == current.x && centerTile.y - 1 ==
current.y) { //0 X C

//0 * 0
                                        if
(!grid.tile [current.x,current.y+1].isWalkable) {
//O O O

        canWalk = false;
                                        }
                                }

                                if (centerTile.x
- 1 == current.x && centerTile.y + 1 ==
current.y) { //* X 0



                            //0 C 0
                                        if
(!grid.tile [current.x+1,current.y].isWalkable) {
//O O O

        canWalk = false;
                                        }
                                }

                                if (centerTile.x
+ 1 == current.x && centerTile.y - 1 ==
current.y) { //C X 0

//0 * 0
                                        if
(!grid.tile [current.x,current.y+1].isWalkable) {
//O O O

        canWalk = false;
                                        }
                                }

                                if (centerTile.x
+ 1 == current.x && centerTile.y - 1 ==
current.y) { //0 0 0

//0 C 0
                                        if
(!grid.tile [current.x-1,current.y].isWalkable) {
//O X *

        canWalk = false;
                                        }
                                }

                                if (centerTile.x
- 1 == current.x && centerTile.y + 1 ==
current.y) { //0 0 0

//0 * 0
                                        if
(!grid.tile [current.x,current.y-1].isWalkable) {
//0 X C

        canWalk = false;
                                        }
                                }
```

```
                                if (centerTile.x
- 1 == current.x && centerTile.y - 1 ==
current.y) { //0 0 0

//0 C 0
                                        if
(!grid.tile [current.x+1,current.y].isWalkable) {
//* X 0

        canWalk = false;
                                        }
                                }

                                if (centerTile.x
+ 1 == current.x && centerTile.y + 1 ==
current.y) { //0 0 0

//0 * 0
                                        if
(!grid.tile [current.x,current.y-1].isWalkable) {
//C X 0

        canWalk = false;
                                        }
                                }

                        }
                        //if rule the catting
conners is on, then check possibility of cutting
conner
                        else {

                                if (centerTile.x
+ 1 == current.x && centerTile.y + 1 ==
current.y) { //0 X *  - in this case, the angle
cut is impossible

//0 C X
                                        if
(!grid.tile [current.x-1,current.y].isWalkable) {
//0 0 0

        if (!grid.tile [current.x,current.y-
1].isWalkable)

        canWalk = false;
                                        }
                                }

                                if (centerTile.x
- 1 == current.x && centerTile.y + 1 ==
current.y) { //* X 0  - in this case, the angle
cut is impossible

//X C 0
                                        if
(!grid.tile [current.x+1,current.y].isWalkable) {
//0 0 0

        if (!grid.tile [current.x,current.y-
1].isWalkable)

        canWalk = false;
                                        }
                                }

                                if (centerTile.x
- 1 == current.x && centerTile.y - 1 ==
current.y) { //0 0 0  - in this case, the angle
cut is impossible

//X C 0
                                        if
(!grid.tile [current.x+1,current.y].isWalkable) {
//* X 0

        if (!grid.tile
[current.x,current.y+1].isWalkable)

        canWalk = false;
                                        }
                                }

                                if (centerTile.x
+ 1 == current.x && centerTile.y - 1 ==
current.y) { //0 0 0  - in this case, the angle
cut is impossible

//0 C X
                                        if
(!grid.tile [current.x-1,current.y].isWalkable) {
//0 X *

        if (!grid.tile
[current.x,current.y+1].isWalkable)

        canWalk = false;
                                        }
                                }
                        }

                        return canWalk;
                }

                //calculate 2D colliders
                public void
CalculateColliders(){

                        for (int y = 0; y <
grid.GridHeight; y++) {
                                for(int x = 0; x
< grid.GridWidth; x++){


        Collider2D[] colls =
Physics2D.OverlapCircleAll
(grid.tile[x,y].WorldPosition, grid.TileDiameter
/ 4);


        if(grid.tile[x,y].Lock)

        continue;


        if(UsePhysics2D){


        if (colls.Length > 0) {



        grid.tile[x,y].isWalkable =
CanWalkable(colls);


                                } else {
```

```
        grid.tile[x,y].isWalkable = true;

        }
                                        }else{


        grid.tile[x,y].isWalkable = true;
                                        }
                                }
                        }
                }

                //checking current tile for its
walk parameter
                bool CanWalkable(Collider2D[]
colls){
                        //check all found
colliders that collide the current tile
                        //if one of the found
colliders has the parameter IsTrigger = false,
then this tile is unwalkable
                        bool walkable = true;

                        foreach (Collider2D coll
in colls) {
                                //if the current
collider has a tag that is in list of ignored
tags, then
                                //checking
current collider is not necessary

        if(IgnoreCollisionTags.Contains(coll.tag
))

        continue;

                        if(coll.isTrigger ==
false){
                                walkable =
false;
                                }
                        }
                        return walkable;
                }

                void Clear(){
                        for (int y = 0; y <
grid.GridHeight; y++) {
                                for (int x = 0;
x < grid.GridWidth; x++) {


        grid.tile[x,y].State =
Tile.listState.Empty;

        grid.tile[x,y].F = 0;

        grid.tile[x,y].G = 0;

        grid.tile[x,y].H = 0;

        grid.tile[x,y].ParentTile = null;
                                }
                        }
                }

                public void GetTilesData(){
                        foreach (Tile t in
ObsUserData.t) {

        grid.tile[t.x,t.y].isWalkable = false;

        grid.tile[t.x,t.y].Lock = true;
                        }
                }

                public void WriteTileData(Tile
tile){
                        if
(ObsUserData.t.Contains (tile))
                                return;
                        ObsUserData.t.Add
(tile);
                }

                public void RemoveTileData(){
                        ObsUserData.t.Clear ();
                }

                public void DeleteTileData(int
x, int y){
                        for (int i=0;
i<ObsUserData.t.Count; i++) {

        if(ObsUserData.t[i].x == x &&
ObsUserData.t[i].y == y){

        ObsUserData.t.Remove(ObsUserData.t[i]);
                                        break;
                                }
                        }
                }
        }

        [System.Serializable]
        public class UserObstaclesData{
                public List<Tile> t = new
List<Tile>();
        }
}
```

B. Road Signs

Warning Sign

Two-Way Traffic Sign

Traffic Signal Ahead Sign

Stop Sign

No Entry Sign

No Left Turn Ahead

Speed Limit Sign

No Parking Sign

No Overtaking Sign

No Right Turn Ahead

Roundabout Sign

C. Written Exam Questions

Following the flow of the process of the game, before the player takes the driving course, the written exam must be taken. Using www.lto.gov.ph as the reference, questions were formulated to suffice the written exam. The set of questions were as follows:

For the **Quiz** round, the questions are:

1. Q: Flashing yellow light means?
   A: *Proceed through the intersection with caution.*

2. Q: Signs that are rounded, rectangular, with white and blue background are called?
   A: *Informative Signs*

3. Q: Should a pre-trip inspection be completed?
   A: *Before operating the motor vehicle.*

4. Q: When driving at night, you should?
   A: *Always turn your headlights on.*

5. Q: The penalty for overcharging or undercharging of authorized rates for first offense?
   A: *Suspension of license for one month.*

6. Q: On a two-lane road, overtaking is only allowed at the?
   A: *Left Lane*

7. Q: To obtain one's driver's license, one must be at least?
   A: *17 years old*

8. Q: Signs that are rounded, inverted, triangle, or octagonal and with red-colored borders are called?
   A: *Regulatory Signs*

9. Q: On wet road you must?
   A: *Slow down*

10. Q: The minimum distance away from the vehicle you are following is?
    A: *One car length*

11. Q: A single solid yellow or white line means?
    A: *Passive/Overtaking is not allowed*

12. Q: Signs that are triangular and with red-colored border are called?
    A: *Caution or Warning Sign*

CURRICULUM VITAE

PERSONAL INFORMATION

Name	:	Monseph Johnrey M. Balenton
Age	:	20
Gender	:	Male
Date of Birth	:	September 7, 1998
Address	:	Victoria, Oriental Mindoro
Email Address	:	monsephjohnreybalenton@lpubatangas.edu.ph
Contact Number	:	09951471687
Nationality	:	Filipino

EDUCATIONAL BACKGROUND

Tertiary : **Lyceum of the Philippines University - Batangas**
Capitol Site, Batangas City
Bachelor of Science in Computer Science
2015- 2019

Secondary : **Good Sheperd Academy**
Poblacion II, Victoria, Oriental Mindoro
2011- 2015

Primary : **Ma. Conception Elementary School**
Ma. Concepcion, Socorro, Oriental Mindoro
2005- 2011

CURRICULUM VITAE

PERSONAL INFORMATION

Name	:	John Frietz F. Dimaano
Age	:	20
Gender	:	Male
Date of Birth	:	November 10, 1998
Address	:	Currency Drive St., Hilltop,Kumintang Ibaba, Batangas City
Email Address	:	johnfrietzdimaano@lpubatangas.edu.ph
Contact Number	:	09062209424
Nationality	:	Filipino

EDUCATIONAL BACKGROUND

Tertiary : **Lyceum of the Philippines University - Batangas**
Capitol Site, Batangas City
Bachelor of Science in Computer Science
2015- 2019

Secondary : **The Pleasant Mount School**
12 Pleasant St, Summerville Subdivision Mayamot, Antipolo City, Rizal
2011- 2015

Primary : **The Pleasant Mount School**
12 Pleasant St, Summerville Subdivision Mayamot, Antipolo City, Rizal
2005- 2011

CURRICULUM VITAE

PERSONAL INFORMATION

Name	:	Dara Pamela L. Gutang
Age	:	20
Gender	:	Female
Date of Birth	:	January 24, 1999
Address	:	Camp Winston Ebersole, San Jose, Occidental Mindoro
Email Address	:	darapamelagutang@lpubatangas.edu.ph
Contact Number	:	09097959107
Nationality	:	Filipino

EDUCATIONAL BACKGROUND

Tertiary : **Lyceum of the Philippines University - Batangas**
Capitol Site, Batangas City
Bachelor of Science in Computer Science
2015- 2019

Secondary : **Occidental Mindoro State College**
Quirino St, San Jose, Occidental Mindoro
2011- 2015

Primary : **Occidental Mindoro State College**
Quirino St, San Jose, Occidental Mindoro
2005- 2011

CURRICULUM VITAE

PERSONAL INFORMATION

Name	:	Simon Luther E. Morales
Age	:	20
Gender	:	Male
Date of Birth	:	April 18, 1999
Address	:	755 Pine Tree St. Cuta, Batangas City
Email Address	:	simonluthermorales@lpubatangas.edu.ph
Contact Number	:	09479522189
Nationality	:	Filipino

EDUCATIONAL BACKGROUND

Tertiary : **Lyceum of the Philippines University - Batangas**
Capitol Site, Batangas City
Bachelor of Science in Computer Science
2015- 2019

Secondary : **King's Kids Christian Academy**
M.H Del Pilar St, Cuta Central, Batangas City, Batangas
2011- 2015

Primary : **King's Kids Christian Academy**
M.H Del Pilar St, Cuta Central, Batangas City, Batangas
2005- 2011

CURRICULUM VITAE

PERSONAL INFORMATION

Name	:	William Joseph C. Tan
Age	:	20
Gender	:	Male
Date of Birth	:	March 08, 1999
Address	:	Sta Clara Subd. Brgy. Sampaloc II, Sariaya, Quezon
Email Address	:	williamjosephtan@lpubatangas.edu.ph
Contact Number	:	09491505827
Nationality	:	Filipino

EDUCATIONAL BACKGROUND

Tertiary :
Lyceum of the Philippines University - Batangas
Capitol Site, Batangas City
Bachelor of Science in Computer Science
2015- 2019

Secondary :
Sariaya Institute Inc.
Sariaya, Quezon
2011- 2015

Primary :
Sariaya Conservative Baptist Christian School Inc.
Sariaya, Quezon
2009- 2011

St. Joseph's Academy
Sariaya, Quezon
2006-2009

San Roque Elementary School
Davao City
2005-2006