Signage Navigation System

Software Design Document

Name : Bar Genish
313174583

Date: (03/15/2020)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

The intended audience for this document is the developer.
This sdd shows how the software system will be structured to satisfy the requirements. It contains all the information required by a programmer to write code. Performed in two stages. The first is a preliminary design in which the overall system architecture and data architecture is defined. In the second stage, i.e. the detailed design stage, more detailed data structures are defined and algorithms are developed for the defined architecture.

1.2 Scope

Signage navigation system,

The project will include a VR game(tour) where the player has a mission to get to a specific checkpoint.

The game will collect statistic information about the user's path.

The goal is to find how much does signboards help people operate in certain situations.

1.3 Overview

The rest of the sdd contains :
System overview and architecture in section 2 and 3
Data and component design in section 4 and 5
human interface design in section 6
requirements matrix in section 7.

2. SYSTEM OVERVIEW

The system will interact with the users in real time (vr). The system will show the user hotspots that represent choices to move around. The user will be able to travel between scenes by looking on the hotspots for a specific amount of time.
the connection between the user and the system should be easy , reliable (
To allow for a proper transition between scenes) and fast .
The system will be developed as a client side app, that will be installed on the vr headsets.

The system will include a module that will be responsible for the vr control and presentation, and a different module that will be responsible for collecting and analyzing meta data from that users actions.

3. System Architecture

3.1 Architectural Design

Vr presentation module

This module is responsible for the vr control and presentation , it is responsible for showing the current vr scene to the user, and the hotspots that connects to other scenes.

Navigation module

Responsible for building and managing the map for the app, keeps track of the users current location on the map. Provide an interface for the vr presentation module that provides the next picture and azimth given selected hotspot.

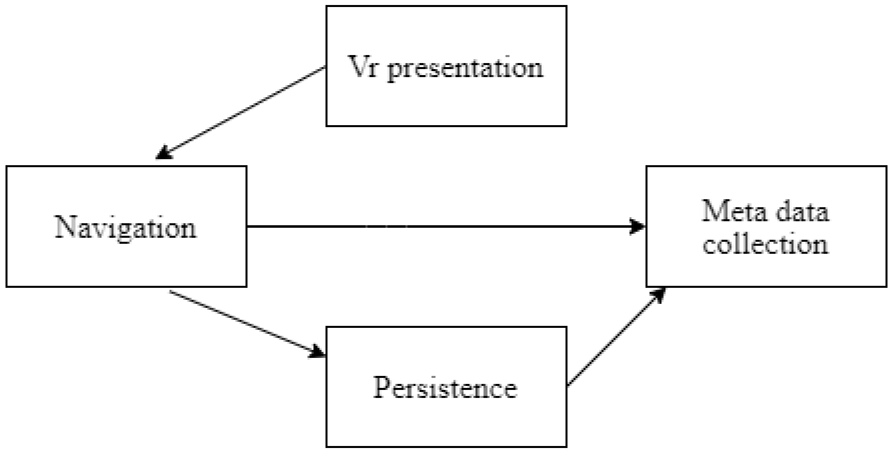
Meta data collection module

Collect and analyze meta data from the user including the full path and time at each location , interact with the navigation module to collects the meta data and provides an interface for the persistence module that provides the collected data to be stored for later.

Persistence module

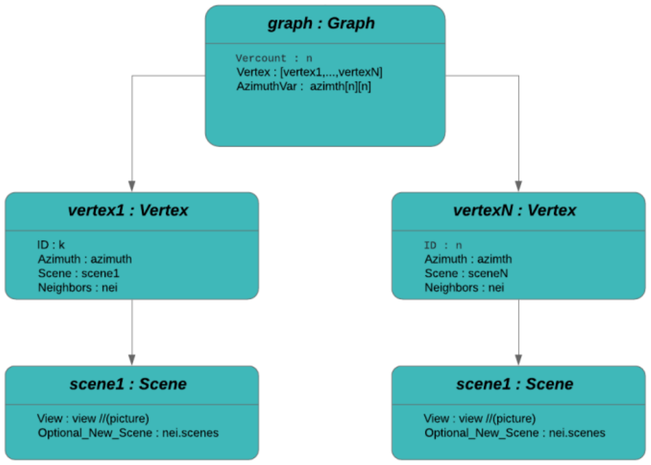
Two major responsibilities :

1. Read the input map data including the map layout, azimuth and hotspot locations. Read the data in the required format and transform it to system data objects.
2. Export the collected meta data into the required format and store it in a csv file.

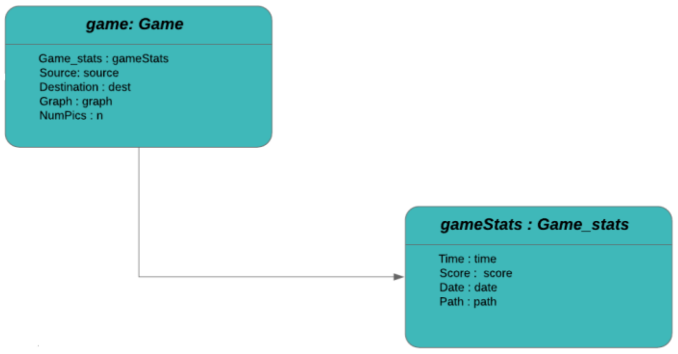


3.2 Decomposition Description

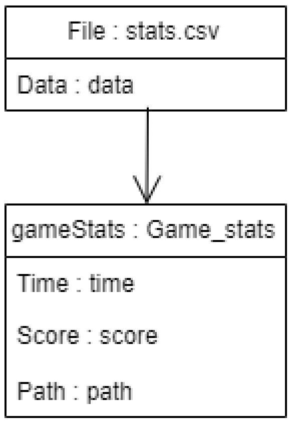
Navigation diagram

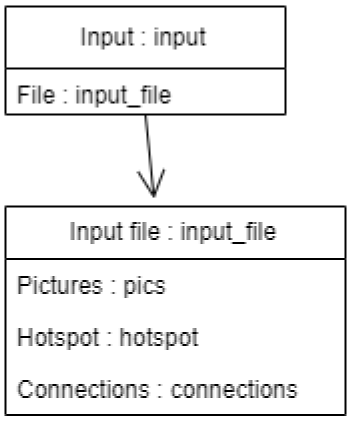


Meta data collection

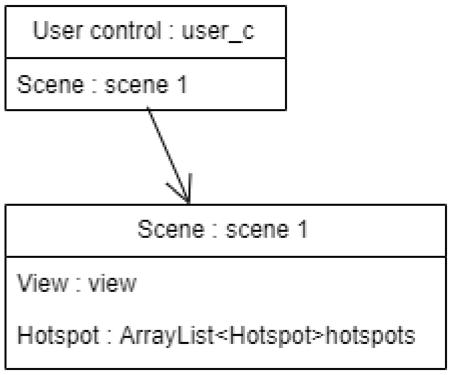


Persistence

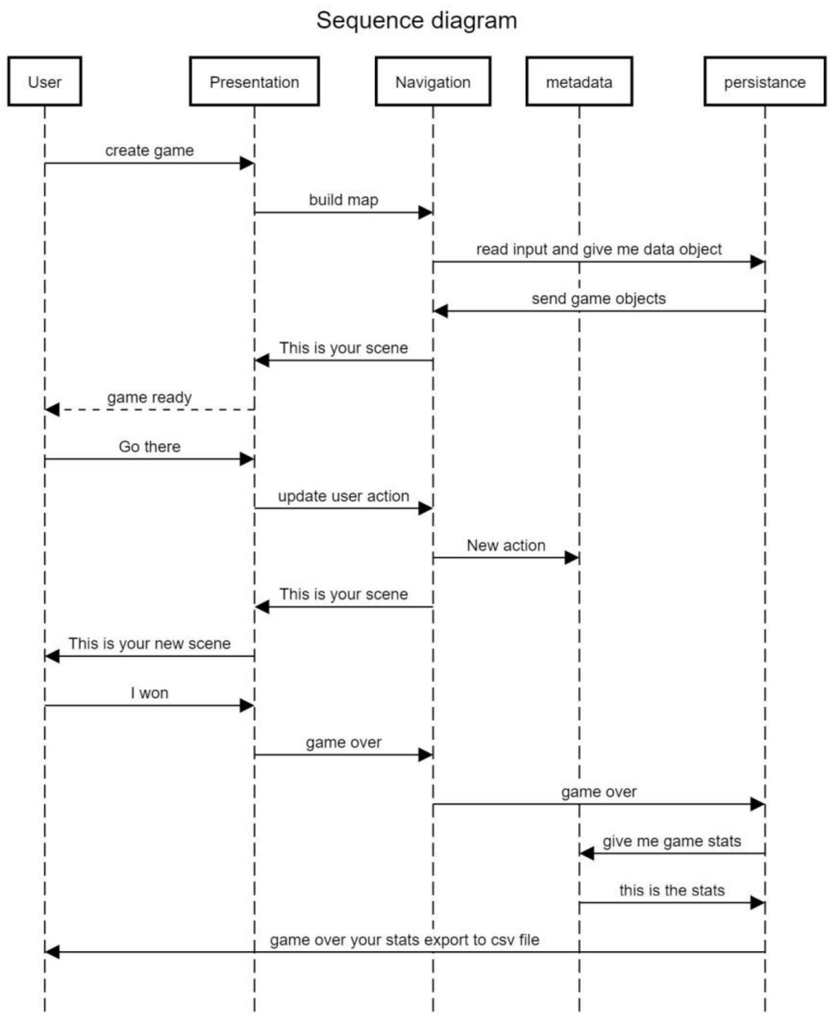




Presentation diagram



sequence diagram



3.3 Design Rationale

Vr headset is very unique and complex, and handling presentation on such devices is complicated and so I decided to create module responsible for the vr presentation and control that will fit the required vr headset and will allow low coupling between vr presentation logic to the rest of the system.

Navigation module is responsible for the main system logic , and will allow to gather the systems main logic in one module , and let this module provide interfaces to the rest of the system.

Meta data collection module responsible for another important system feature that is separated from the main logic, the meta data collection. This feature is important as well and by creating a module for it we can separate its functionality from the rest of the systems logic.

The system works with very complex data type including the vr pictures map layout and the collected meta data. We want a module that will be able to translate this complex data into system objects in a modular way , and provide an api for the rest of the system for storing and gathering data from the machine.

4. DATA DESIGN

4.1 Data Description

The game require a specific json format input, that contains points field , which every point represent a scene, and the possible neighbors scene with the azimuth off passing between scenes.
The game result exports to a csv file that contains the date , the user path and how many second the user spent in each scene .

4.2 Data Dictionary

My data type example :

The csv file contains : the current Timestamp , list of string that represent the user path and list of string that represent how many second the user spent in each point of his path.
The json file contain : Each view of the game that will represent scene , the view attribute are : the viewID , the view-PATH (the path of the picture) and possible crossover between views when each neighbor (crossover) contain the neighbor ID and the azimuth between the views.
The json file deserialize and parse to a graph object , the graph contains list of vertex that each vertex have the follow attributes: ID , PATH , list of neighbors that each neighbor represent as a tuple of the vertex and the azimuth.

5. Component Design

Graph Code:

```
public class Graph
{
    public List<Vertex> _vertex = new List< Vertex > ();
    public Graph()
    {
        _vertex.Clear();
    }
    public Vertex GetVertexById(int Id)
    {
        foreach (Vertex v in this._vertex)
            if (v.ID == Id) return v;
        return null;
    }
}
public class Tuples
{
    private Vertex _verex;
    private double azimuth;

    public Vertex Verex { get => _verex; set => _verex = value; }
    public double Azimuth { get => azimuth; set => azimuth = value; }
    public Tuples(Vertex v,double azim)
    {
        _verex = new Vertex(v);
        azimuth = azim;
    }
}

public class Vertex
{
    private int id;
    private List<Tuples> nei = new List<Tuples>();

    public Vertex(int ID,string URL)
    {
        id = ID;
        Url = URL;
    }
    public Vertex(Vertex ot)
    {
        this.id = ot.id;
        this.nei.Clear();
        this.nei.AddRange(ot.nei);
    }
    public void addnei(Vertex v,double azimuth)
    {
        nei.Add(new Tuples(v,azimuth));
    }
    public List<Tuples> GetNei() { return nei; }
    public double Azimuth { get; set; }
    public string Url { get; set; }
    public int ID { get => id; set => id = value; }



   }
```

Parser code :

```
class Parser
{
    public class Points
    {
        public Point[] points { get; set; }
    }
    public class Point
    {
        public int id { get; set; }
        public string Picture { get; set; }
        public Neighbor[] Neighbors { get; set; }
    }
    public class Neighbor
    {
        public int PointID { get; set; }
        public int Azimut { get; set; }
        public Points DeserializeJson()
        {
            Points points =
JsonConvert.DeserializeObject<Points>(File.ReadAllText(@"config.json"));
            return points;
        }
       }
```

Export csv file code :

```
public static class Stats
{
    public static ArrayList Path = new ArrayList();
    public static ArrayList Times = new ArrayList();
    public static float timer = 0;
    public static void CreateCsvFile()
    {
        string data = System.DateTime.Now + ":,";
        for (int i = 0; i < Path.Count; i++)
        {
            data += Path[i] + " . Times is : " + Times[i] + ",";
        }
        string path = Application.persistentDataPath + "/stats.csv";
        Debug.Log(path);
        Debug.Log(data);
        try
        {
            //    saved2();
            if (!File.Exists(path))
            {
                File.WriteAllText(path, data);
                Debug.Log("new created");
            }
            else
            {
                File.AppendAllText(path, "\n" + data);
                Debug.Log("old modified");
            }
            // You can find the created folder in your phone's memory
        }
        catch (Exception e) { Debug.LogError(e); }
    }
   }
```

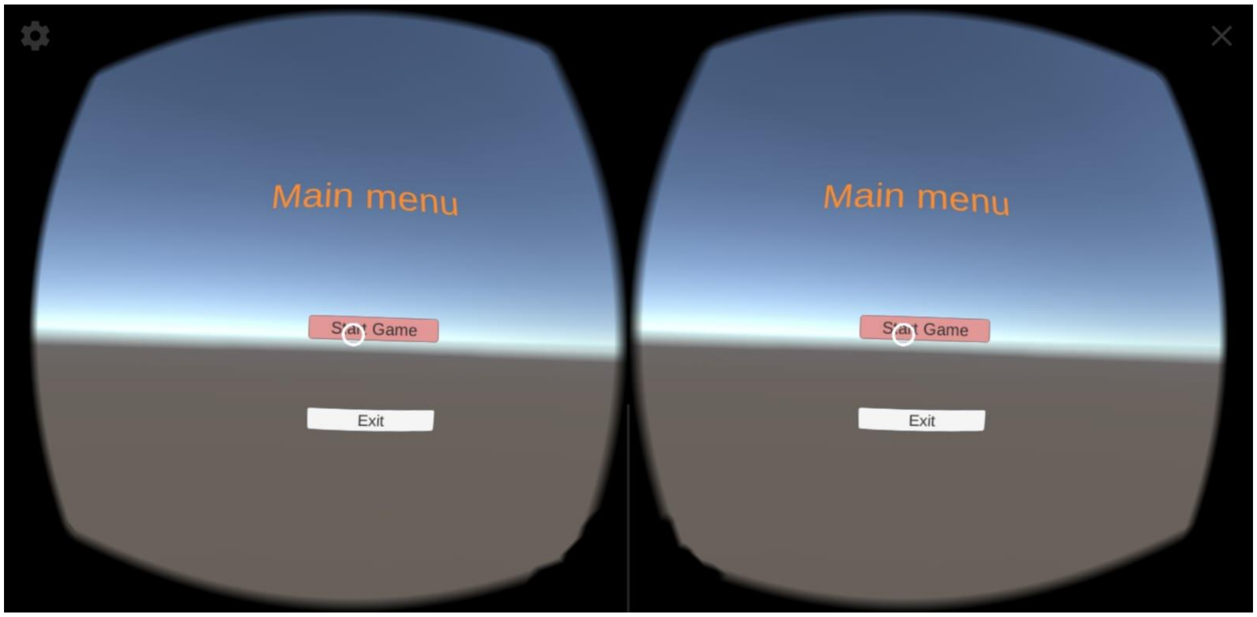
6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

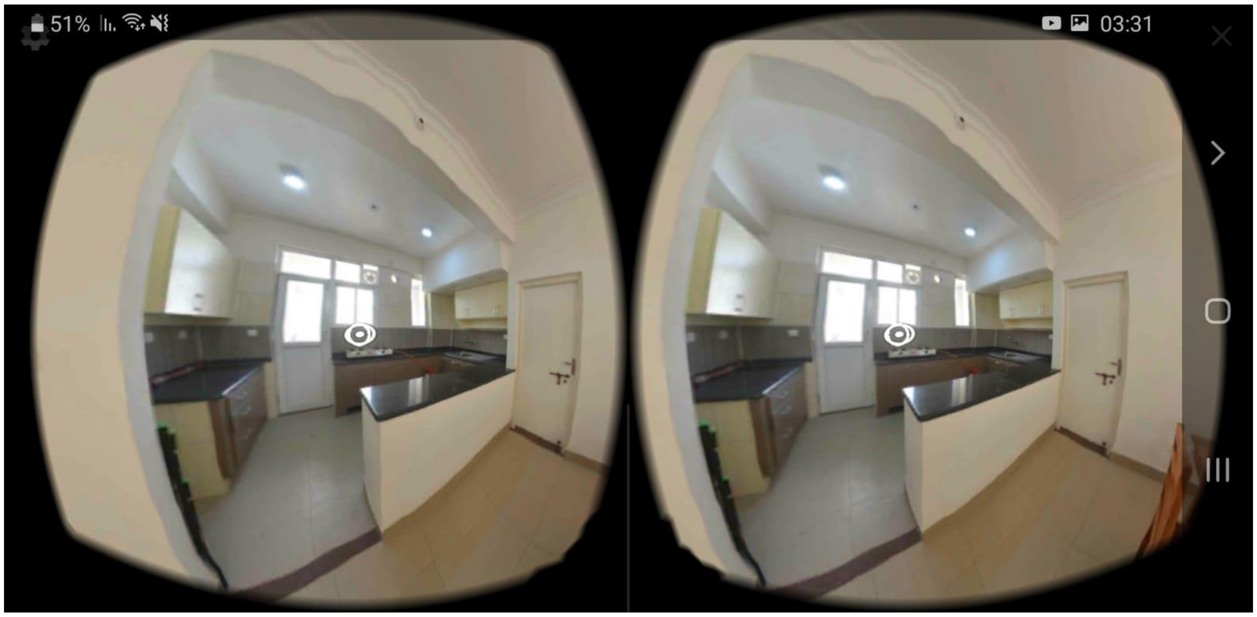
The user has to run the app, then click the menu to start a game and then he can travel between scenes until he can reach the final destination.
To switch between scenes the player has to look at a hotspot in which he wants to reach and click.

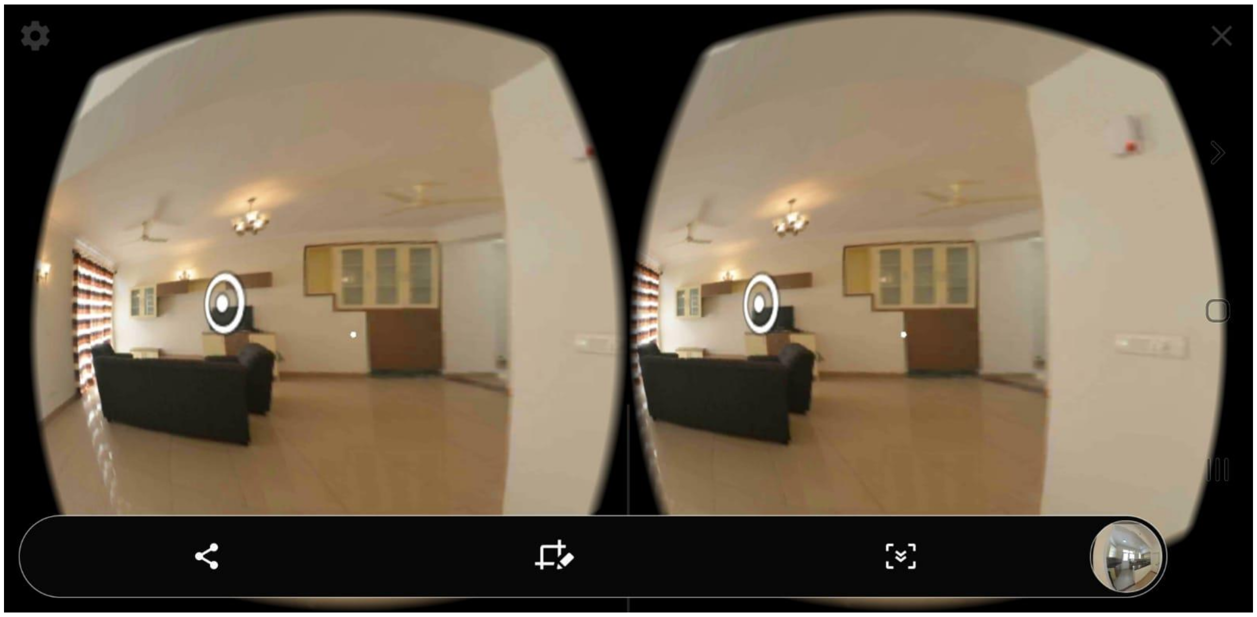
6.2 Screen Images

Screenshot of the menu :



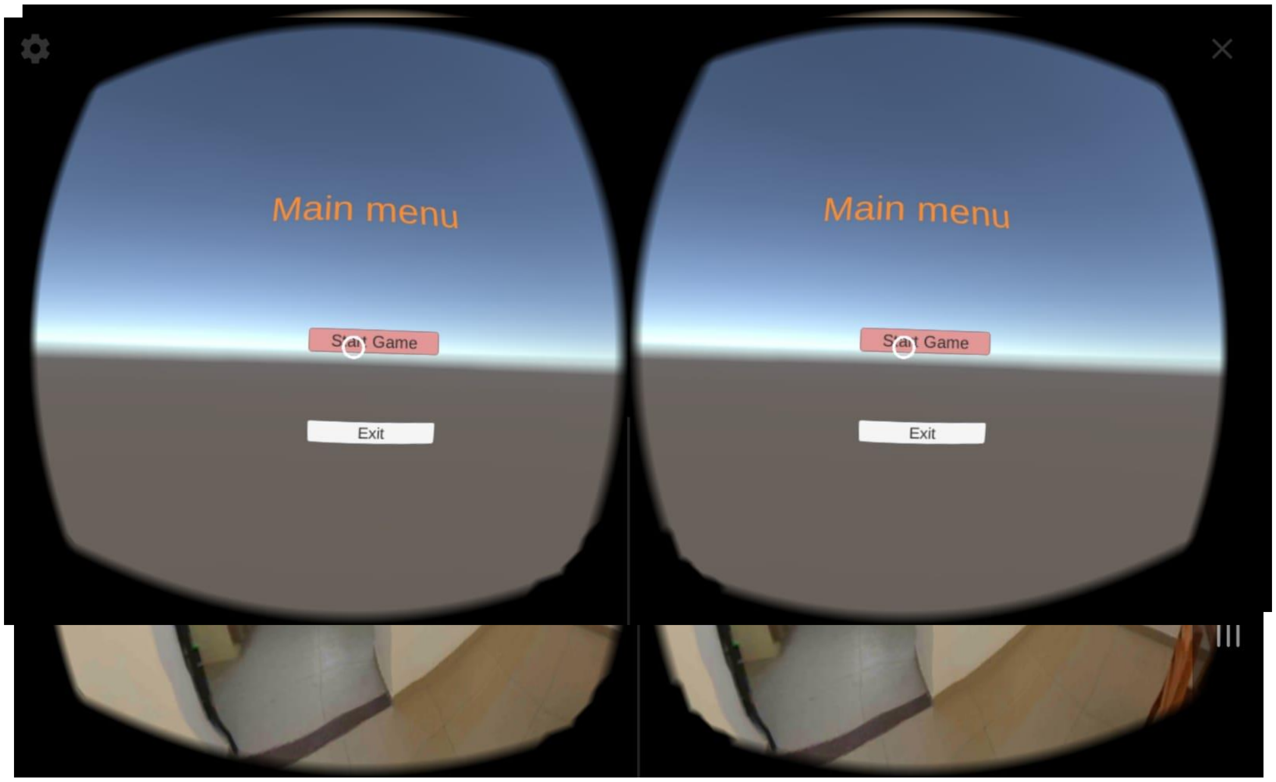
Screenshot of option to crossover between scenes :



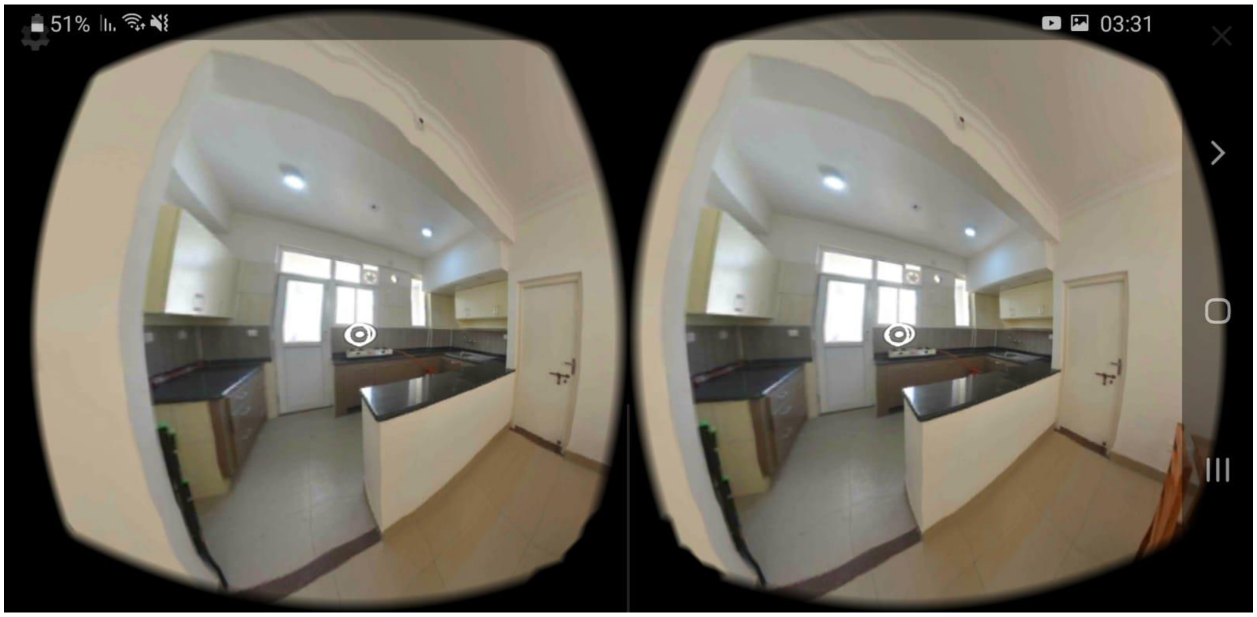


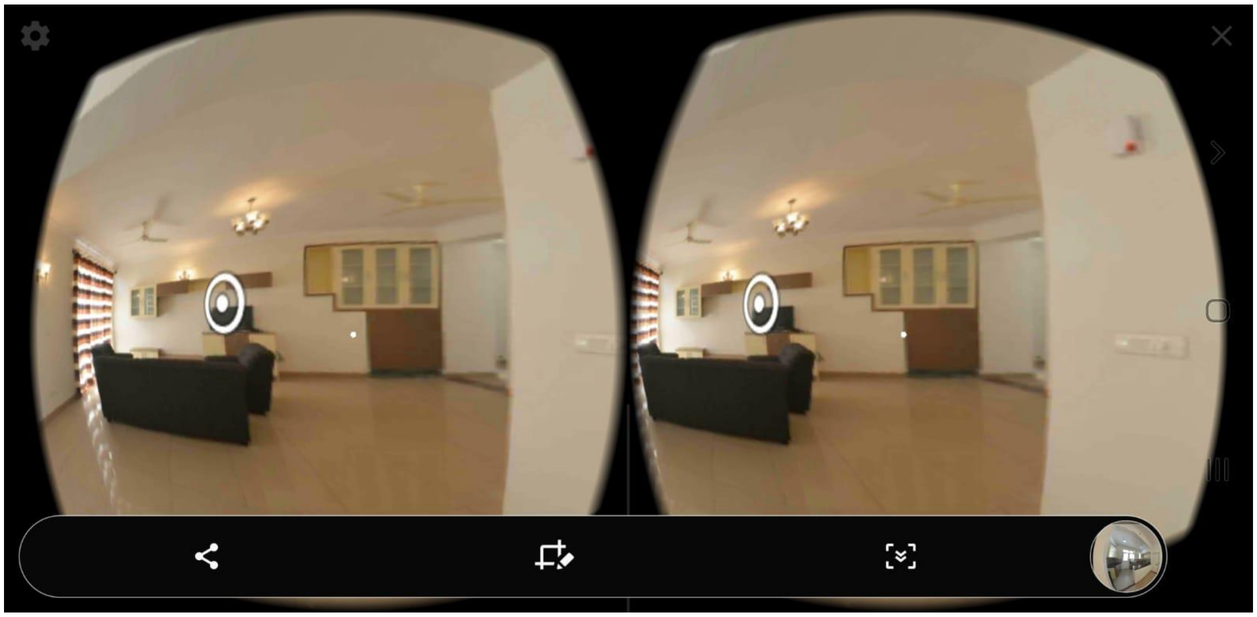
Screenshot of the menu :

Here the user have the choice to start game or quit the app by moving until the marker is on the wanted option and click .



In this two photos the player is in the middle, and he has to choose where he wants to go, he has 2 options that only one can lead him to the destination.





7. Requirements Matrix

	VR presentation	Navigation	Persistence	Meta data collection
System input			✓	
Export stats				✓
Set auto hotspots	✓	✓		
The system build generically based on input		✓	✓	
Move between possible scenes	✓			
Inform the user when he gets to the destination	✓	✓		
Measure times				✓
Transfer 360 images to vr			✓	

Asked for safety	✓			
Save the user path				✓
Inform the user on the start and destination point	✓	✓		
Build graph		✓	✓	
Move to the wanted azimuth	✓	✓		
Quit game if the input invalid			✓	
Transform each image to scene		✓	✓	
Putting pressure on user	✓			