A

PROJECT REPORT

on

INDOOR NAVIGATION USING AUGMENTED REALITY

Submitted in partial fulfilment for the Award of Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

BY

Pavan Kalyan Inugurthi (160117737100)
&
Preethivardhan Anusri Ega (160117737102)

Under the guidance of

Smt. T. Satya Kiranmai,
Assistant Professor,
Dept. of IT, CBIT.

DEPARTMENT OF INFORMATION TECHNOLOGY
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)
(Affiliated to Osmania University; Accredited by NBA (AICTE) and NAAC (UGC), ISO Certified 9001:2015), Kokapet (V), GANDIPET(M), HYDERABAD – 500 075

Website: www.cbit.ac.in

2020-2021

CERTIFICATE

This is to certify that the project seminar report entitled "**INDOOR NAVIGATION USING AUGMENTED REALITY**" submitted by **PAVANKALYAN INUGURTHI (160117737100)** and **PREETHIVARDHAN ANUSRI EGA (160117737102)** in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF ENGINEERING** in **INFORMATION TECHNOLOGY** to **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY(A)**, affiliated to **OSMANIA UNIVERSITY,** Hyderabad, is a record of bonafide work carried out by them under my supervision and guidance. The results embodied in this report have not been submitted to any other University or Institute for the award of any other Degree or Diploma.

Head of the Department
Dr. K. Radhika.
Professor, IT Dept
CBIT, Hyderabad

Project Guide
Smt. T. Satya Kiranmai
Assistant Professor, IT Dept.
CBIT, Hyderabad.

DECLARATION

We declare that the project report entitled **"INDOOR NAVIGATION USING AUGMENTED REALITY"** is being submitted by us in the Department of Information Technology, Chaitanya Bharathi Institute of Technology (A), Osmania University.

This is a record of bonafide work carried out by us under the guidance and supervision of **Smt. T. Satya Kiranmai**, Assistant Professor, Dept. of IT, C.B.I.T

No part of the work is copied from books/journals/internet and wherever the portion is taken, the same has been duly referred to in the text. The reports are based on the project work done entirely by us and not copied from any other source.

PAVANKALYAN INUGURTHI (160117737100)

PREETHIVARDHAN ANUSRI EGA (160117737102)

ACKNOWLEDGEMENTS

It is our privilege to acknowledge with deep sense of gratitude and devotion for keen personal interest and invaluable guidance rendered by our Project Guide **Smt. T. Satya Kiranmai**, Assistant Professor, Department of Information Technology, Chaitanya Bharathi Institute of Technology.

We take the opportunity to express our thanks to **Dr K. Radhika**, Professor & Head of IT Department, CBIT for her valuable suggestions and moral support

We are grateful to our Principal **Prof. G. P. Saradhi Varma**, Chaitanya Bharathi Institute of Technology, for his cooperation and encouragement.

Finally, We also thank all the staff members, faculty of Dept. of IT, CBIT, and our friends, who with their valuable suggestions and support, directly or indirectly helped us in completing this project work

ABSTRACT

With a Smartphone in hand, it is pretty easy for us to find our way to the destination using outdoor GPS navigation mobile apps, even when we are in an unfamiliar city. However, it is possible to get lost indoors, with GPS satellite signals not being accurately traceable in case of navigation apps. Indoor navigation deals with navigation within buildings. Because GPS reception is normally non-existent inside buildings, Wi-Fi or Bluetooth Beacons can be used for indoor navigation. But these have an accuracy of 5 – 15 meters and require costly hardware installation. It's easier to navigate indoors when you can see your surroundings.

We intend to develop an Indoor Navigation Application using Augmented Reality which will show the directions to the destination in the user's camera screen. QR codes shall be installed at all possible destinations in the building assuming any destination can be the starting point of the user. User has to scan a QR code to select a destination. Google AR Core takes live feed from the user's camera and does simultaneous localization and mapping to update the user's location. Shortest path to the chosen destination is found using A * algorithm and the directions to the destination are shown in the user's camera screen using Augmented Reality.

The application is developed in Unity from scratch using some essential plugins like Google ARCore. We aim at developing the front end in the simplest way possible so that the users can easily reach their destination by just opening the camera where the directions are shown as animations in their surroundings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1. **AR**: Augmented Reality
2. **QR:** Quick Response
3. **RGB:** Red Green Blue
4. **SLAM:** Simultaneous Localization and Mapping

1. INTRODUCTION

With a Smartphone in hand, it is pretty easy for us to find our way to the destination using outdoor GPS navigation mobile apps, even when we are in an unfamiliar city. However, it is possible to get lost indoors, with GPS satellite signals not being accurately traceable in case of navigation apps.

Indoor navigation deals with navigation within buildings. Because GPS reception is normally non-existent inside buildings, Wi-Fi or Bluetooth Beacons can be used for indoor navigation. But these have an accuracy of 5 – 15 meters and require costly hardware installation. It's easier to navigate indoors when you can see your surroundings. So, we intend to develop an Indoor Navigation Application using Augmented Reality.

1.1 PROBLEM STATEMENT

A mobile application which can scan QR codes for live positioning of the user and can navigate the user from current location to any destination chosen within the building using Augmented Reality by showing directions in the camera.

1.2 APPLICATIONS

Indoor Navigation has its applications in various parts of everybody's daily life. Deploying the Indoor Navigation System in public places like Shopping malls, Movie Theaters, Office Complexes, etc. will be very helpful for the users to navigate to their destination inside the building quickly without any hassle.

1.3 ORGANIZATION OF THE REPORT

Chapter 1 deals with the introduction of the project and explains the purpose of the project.

Chapter 2 deals with the literature survey

Chapter 3 deals with the requirements that are needed in order to execute the project

Chapter 4 deals with the methodology, system design and features of the project

Chapter 5 deals with the implementation of the project

Chapter 6 involves the results of the project

Chapter 7 involves conclusion and future scope of the project.

1.4 AIM OF THE PROJECT

The main aim of the project is to develop a mobile application which can navigate the user to any destination from any starting point in real time indoors using Augmented Reality. Also, to build this application in such a way that even if the user goes in the wrong direction this should be able to correct it while maintaining its accuracy.

2. LITERATURE SURVEY

Based on the problem statement and the project requirements the following research papers have been referred:

2.1 PAPER1:

Title: Bluetooth-based Indoor Navigation Mobile System
Authors: Adam Satan
Source: IEEE
Description: A bluetooth positioning based indoor navigation application was presented in this paper. For high availability, Indoor Navigation system should use sensors such as Wi-fi, Bluetooth, gyro sensors, accelerometer and compass. In this project they focused on the usage of the Bluetooth, especially Bluetooth Low Energy and compass sensors, because they can be found in most smartphones.

The application uses Bluetooth Low Energy beacons to calculate the user's position and a modified Dijkstra's Shortest Path Algorithm to determine the path. Information about the building is stored in a local database on the smartphone so the app can work off-line. The application was implemented in the Android platform.

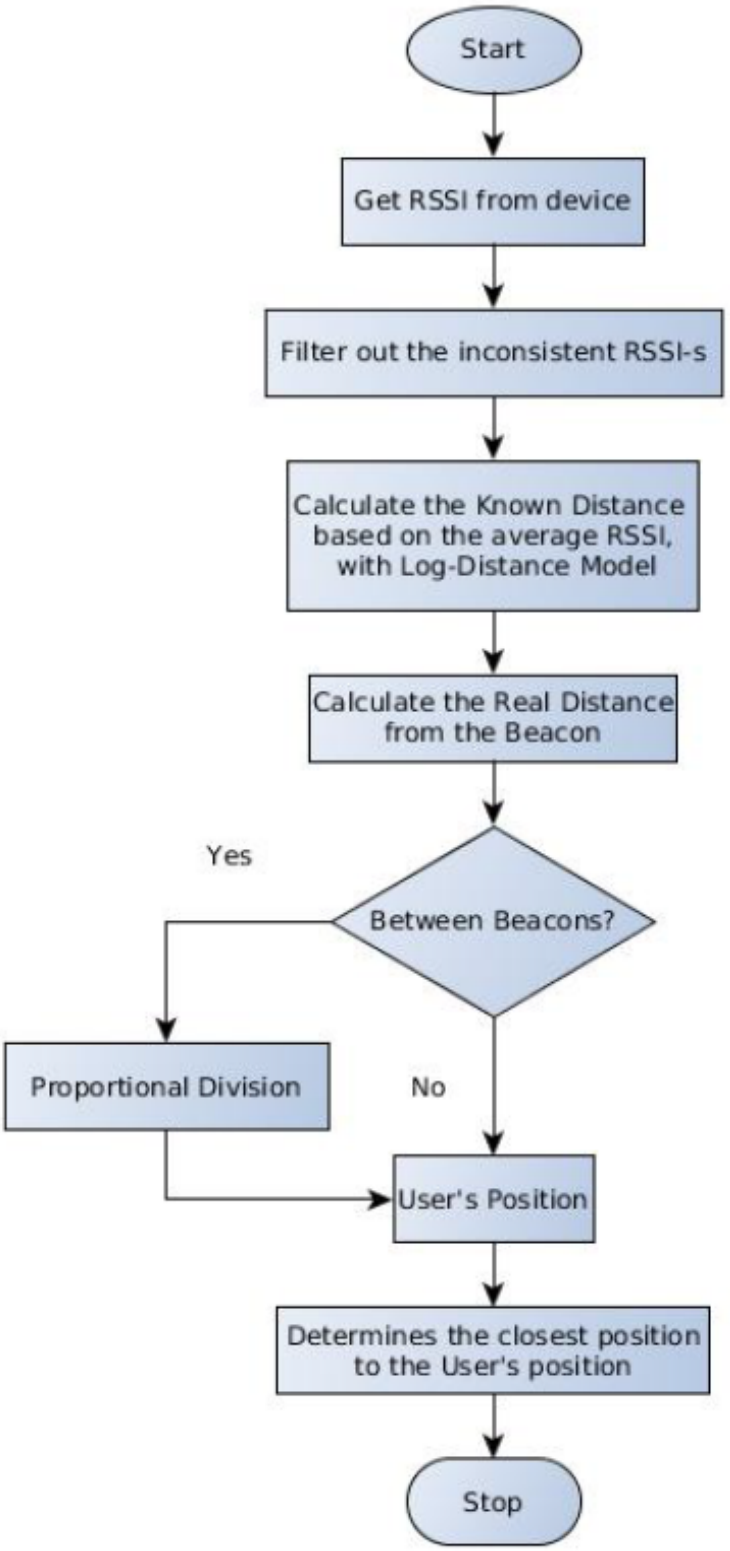

Figure 2.1.1: Flowchart of the Bluetooth based Indoor Navigation System

The major drawback we found in this project is the Accuracy of this system is very less and sometimes the algorithm provided a wrong path. Moreover deploying this system is very costly as we have to install Bluetooth Beacons across the building.

2.2 PAPER 2:

Title: Indoor Navigation System Using Visual Positioning System with Augmented Reality

Authors: Ravinder Yadav, Himanika Chugh, Vandit Jain, Prasenjit Baneriee

Source: IEEE

Description: This project intends to develop an Indoor Navigation System using Augmented Reality with ARToolkit. It proposes a development methodology for an interactive indoor navigation system which uses both image processing and augmented reality as satellite positioning cannot perform well in an indoor environment as the satellite signals cannot penetrate the building.

In this paper they used ARToolkit to display directions for navigation. ARToolkit is an open source tracking library which can superimpose virtual 3D imagery on real world environment. AR toolkit does two important works where first is viewpoint tracking and second is virtual object interaction. AR toolkit is able to calculate real time camera orientation and position. Once ARtoolkit has calculated the real camera position then it aligns the virtual camera with the real camera and then it draws 3D computer graphics exactly over the real marker.

This project uses Dijkstra's Shortest Path Algorithm to determine the path from users location to chosen destination inside the building.

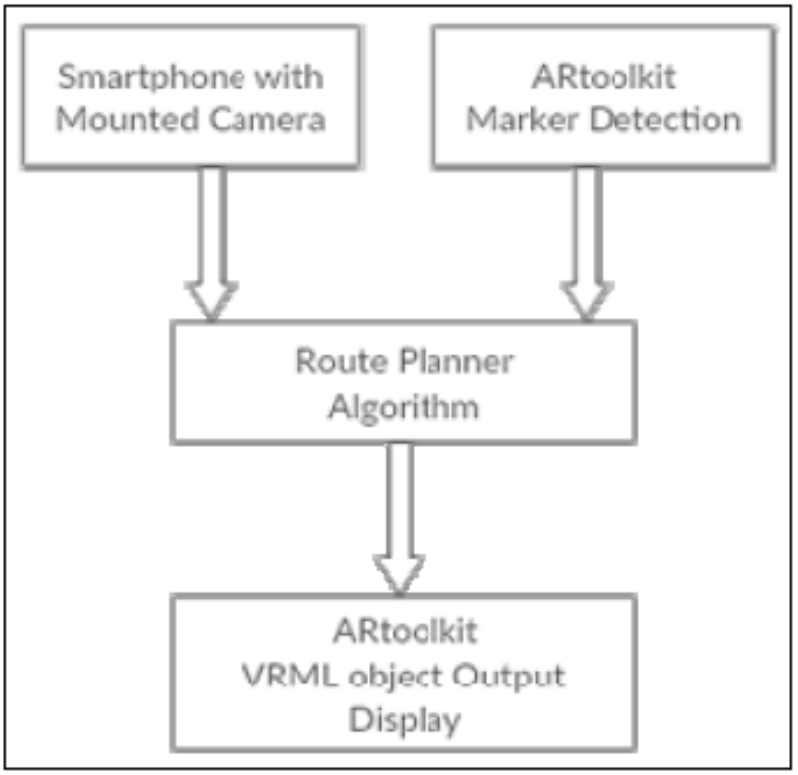

Figure 2.2.1: Flowchart of the ARToolkit based Indoor Navigation System

We found 2 major drawbacks in this project. Firstly, it uses ARToolkit which is an old tool and is not much reliable for developing AR applications. Second, it uses Dijkstra's Shortest Path Algorithm to determine the path which has its own limitations.

2.3 PAPER3:

Title: A multi-functional method of QR code used during the process of indoor navigation

Authors: Daria Mamaeva; Mikhail Afanasev; Vitaliy Bakshaev; Mark Kliachin

Source: IEEE

Description: In this paper, they have proposed a method for indoor navigation using QR code combined with AR technology. QR codes must be placed at all hub locations inside the building, such as intersections, corridors, staircases, and classroom/office/service space doors. An optimal route from point A to point B is based on A* search algorithm.

Using QR codes, the navigation map identifies the user's location and places a 3D object on the smartphone's screen. 3D objects are presented by arrows which set the direction to the next point. Depending on the chosen path, AR arrow points towards the next node along the path. Thus, the user knows exactly where it is needed to make right turns or left turns.

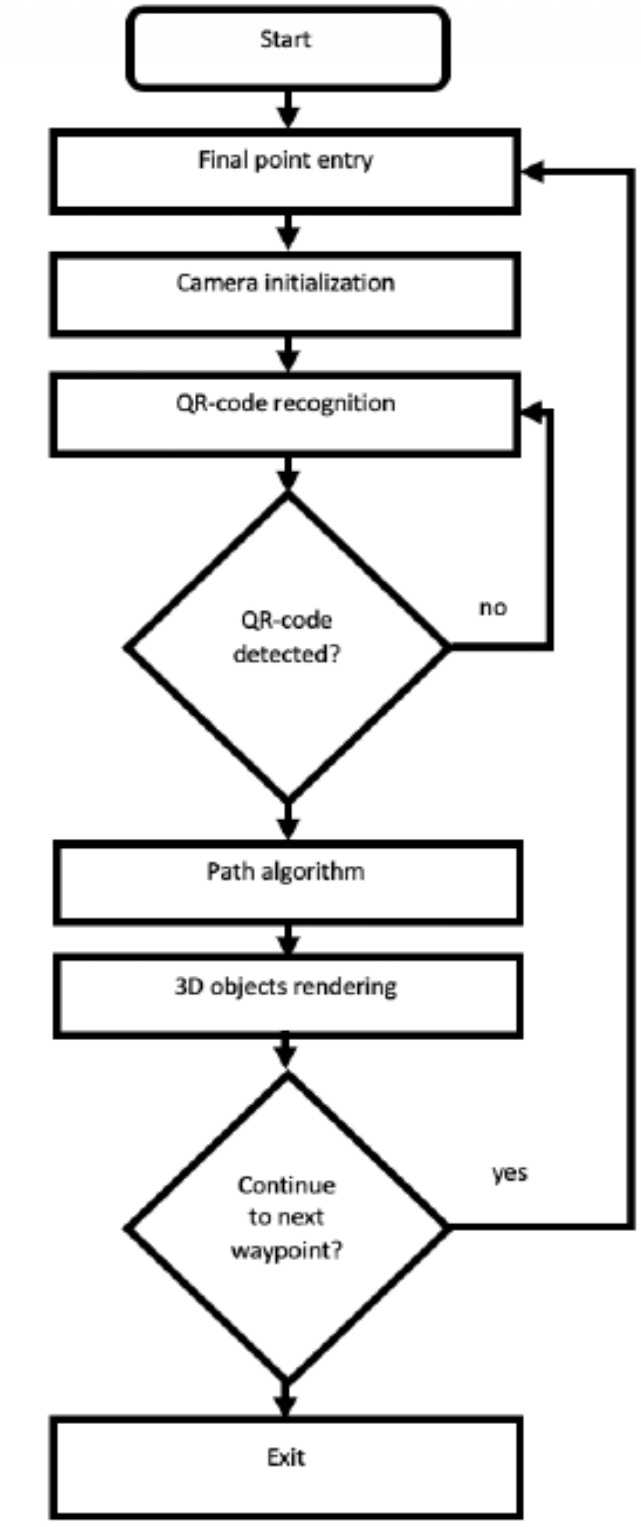

Figure 2.3.1: Flowchart of the QR Codes based Indoor Navigation System

The major drawback we found in this project is that for a user to go to the destination from his location he/she has to scan each and every QR code that comes in the path to the destination.

3. SYSTEM REQUIREMENT SPECIFICATION

3.1 FUNCTIONAL REQUIREMENTS

A Functional Requirement (FR) is a description of the service that the software must offer. It describes a software system or its component. A function is nothing but inputs to the software system, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

Our Project Indoor Navigation using Augmented Reality operates as described below:

1. The user has to scan a QR code to let the application know the current location of the user.
2. The user has to select the destination that he/she needs to go to.
3. The application needs to be open, which shows the user the path.
4. The path displayed in their smartphone as an augmented arrow, the user needs to follow the path shown, to reach their destination.
5. If the user chooses the wrong path, the application will update the path and will show the user the updated path.

3.2 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements are the requirements which specify criteria that can be used to judge the operation of a system, rather than specific behaviors. This should be contrasted with functional requirements that specify behavior or functions. Typical non-functional requirements responsiveness, scalability, security, usability.

Responsiveness: As soon as the QR is scanned, the application should display all the possible destinations that he/she can go to.

Scalability: The system should be scalable and must be capable of handling any sized building that the application is made for.

Usability: The system should be easy to use so that the users do not face any kind of difficulty while operating it.

Accurate: The path that is displayed, needs to be accurate enough so that the users should not have any trouble following it.

3.3 SOFTWARE REQUIREMENTS

S. No.	Software	Description
1.	Main Platform	Unity
2.	Main Plugin	Google AR Core
3.	Programming Language	C#
4.	Other Tools	Blender

Table 3.3.1: Software Requirements

3.3.1 UNITY

Unity is a cross-platform game engine developed by Unity Technologies. It packs a ton of features together and is flexible enough to make almost any game you can imagine. The engine can be used to create three-dimensional (3D) and two-dimensional (2D) games, as well as interactive simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering and construction.

Unity provides a workspace that combines artist-friendly tools with a component-driven design that makes game development pretty darn intuitive. Both 2D and 3D development is possible in Unity, with 2D physics handled by the popular Box2D engine. Unity uses a component-based approach to game dev revolving around prefabs. With prefabs, game designers can build objects and environments more efficiently and scale faster.

3.3.2 GOOGLE AR CORE

Google AR Core is a plugin that brings AR functionalities to unity. In order to augment reality, our devices need to understand it. ARCore provides a variety of tools for understanding objects in the real world. These tools include environmental understanding, which allows devices to detect horizontal and vertical surfaces and planes. They also include motion tracking, which lets phones understand and track their positions relative to the world. As ARCore continues to improve and expand, it will add more contextual and semantic understanding about people, places and things.

3.3.3 C#

A script must be attached to a GameObject in the scene in order to be called by Unity. Scripts are written in a special language that Unity can understand. And, it's through this language that we can talk to the engine and give it our instructions.

The language that's used in Unity is called C# (pronounced C-sharp). All the languages that Unity operates with are object-oriented scripting languages. Like any language, scripting languages have syntax, or parts of speech, and the primary parts are called variables, functions, and classes.

3.3.4 BLENDER

Blender is a free and open-source 3D computer graphics software toolset used for creating animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, virtual reality, and computer games.

Unity natively imports Blender files. This works under the hood by using the Blender FBX exporter.

3.4 HARDWARE REQUIREMENTS

S. No.	Hardware	Description
1.	Operating System	Windows 7/8.1/10
2.	Processor	Intel(R) Core(TM) i3/i5/i7 CPU M 350 @2.27GHz
3.	RAM	2 GB or more
4.	Disk Space	10GB or more

Table 3.4.1: Hardware Requirements

4. METHODOLOGY

After going through the above papers and their drawbacks we would now like to develop an Indoor Navigation Application using Augmented Reality without any limitations. The first step is to develop this application is to build a 3D model of the building along with the interiors in which we want to deploy this application. We are going to use the Blender tool to develop the 3D model of the building.

To navigate a user to a destination from his/her current location, the system needs to understand the user's location and it has to map it to the 3D model of the building. To achieve this we are going to install QR codes at all possible destinations in the building assuming any destination can be the starting point of the user. QR codes are being used in many areas such as: education, marketing, the gaming industry and businesses. In our daily lives QR codes provide an opportunity to share or receive additional information about a place or product. Coupled with the fact that Apple's iOS 11 update added QR code scanning capability in the camera app along with the latest Android smartphones being able to do the same, QR codes have become an integral part of everyday life.

In an effort to deliver a great indoor-navigation experience, we decided to use QR codes. Each QR code is pegged to the specific graph node but not every node contains QR code. Using QR codes, the navigation map identifies the user's location and places a 3D object on the smartphone's screen. 3D objects are presented by arrows which set the direction to the next point. Once the user scans any QR code, the system understands his/her current location and asks him to select the destination.

After the user selects the destination, the user's camera is opened. Google AR Core does simultaneous localization and mapping by taking live feed from the user's camera which means the live feed of the camera is compared with the 3D model of the building to get the exact location of the user. The user's location is updated very seamlessly similar to GPS as the user moves across the building. Depending on the chosen path, AR (Augmented Reality) arrow points toward the next node along the path. Thus, the user knows exactly where it is needed to make right turns or left turns.

As the system gets the user's location, it finds the shortest path to the chosen destination using A * shortest path finding algorithm. A * algorithm is much faster,

efficient and reliable than Dijkstra's algorithm. After the system gets the path to the destination, it places a virtual 3D arrow object in the user's camera screen which will assist the user to go through the shortest path to reach the destination. The shortest path keeps on updating as the user's location is updated. So, the directions are updated even if the user moves in the wrong direction and there is no need for the user to scan QR codes along the path to destination.

4.1 Google ARCore

ARCore is Google's platform for building augmented reality experiences. Using different APIs, ARCore enables your phone to sense its environment, understand the world and interact with information. Some of the APIs are available across Android and iOS to enable shared AR experiences.

ARCore uses the below key capabilities to integrate virtual content with the real world as seen through your phone's camera.

4.1.1 Motion tracking

As your phone moves through the world, ARCore uses a process called simultaneous localization and mapping, or SLAM, to understand where the phone is relative to the world around it. ARCore detects visually distinct features in the captured camera image called feature points and uses these points to compute its change in location. The visual information is combined with inertial measurements from the device's IMU to estimate the pose (position and orientation) of the camera relative to the world over time.

By aligning the pose of the virtual camera that renders your 3D content with the pose of the device's camera provided by ARCore, developers are able to render virtual content from the correct perspective. The rendered virtual image can be overlaid on top of the image obtained from the device's camera, making it appear as if the virtual content is part of the real world.

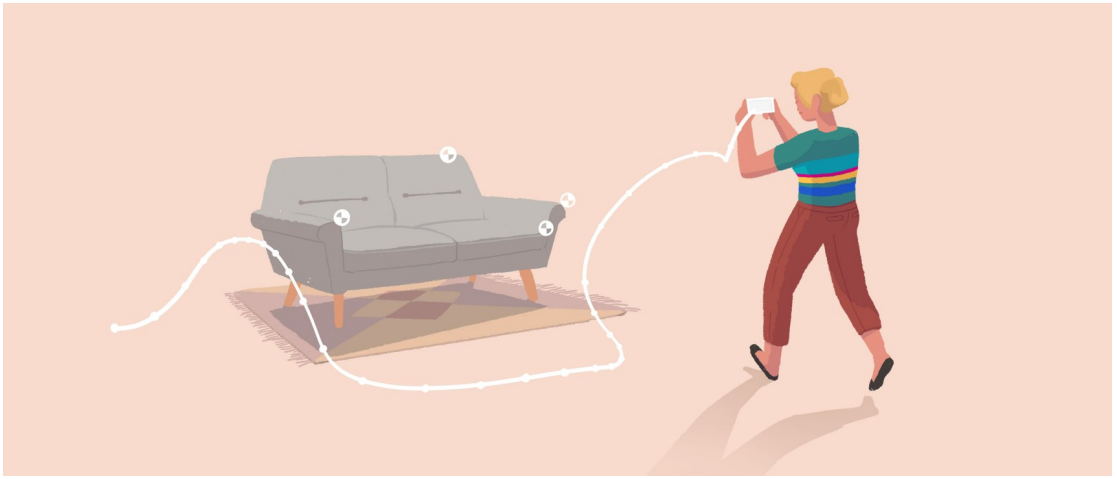

Figure 4.1.1: Motion Tracking with Google ARCore

4.1.2 Environmental understanding

ARCore is constantly improving its understanding of the real world environment by detecting feature points and planes.

ARCore looks for clusters of feature points that appear to lie on common horizontal or vertical surfaces, like tables or walls, and makes these surfaces available to your app as planes. ARCore can also determine each plane's boundary and make that information available to your app. You can use this information to place virtual objects resting on flat surfaces.

Because ARCore uses feature points to detect planes, flat surfaces without texture, such as a white wall, may not be detected properly.

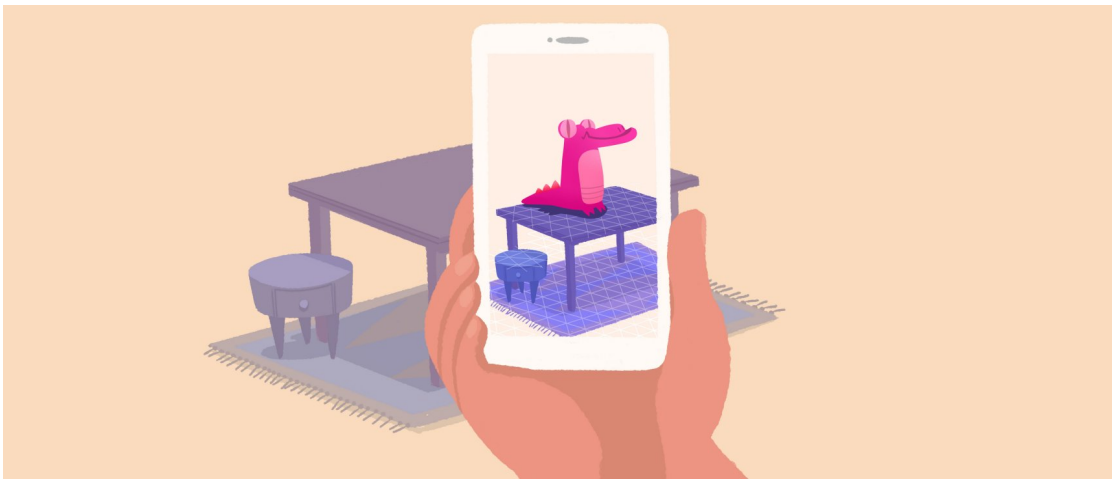

Figure 4.1.2: Environmental Understanding with Google ARCore

4.1.3 Depth understanding

ARCore can create depth maps, images that contain data about the distance between surfaces from a given point, using the main RGB camera from a supported device. You can use the information provided by a depth map to enable immersive and realistic user experiences, such as making virtual objects accurately collide with observed surfaces, or making them appear in front of or behind real world objects.

4.1.4 Oriented points

Oriented points lets you place virtual objects on angled surfaces. When you perform a hit test that returns a feature point, ARCore will look at nearby feature points and use those to attempt to estimate the angle of the surface at the given feature point. ARCore will then return a pose that takes that angle into account.

Because ARCore uses clusters of feature points to detect the surface's angle, surfaces without texture, such as a white wall, may not be detected properly.

4.1.5 Anchors and trackables

Poses can change as ARCore improves its understanding of its own position and its environment. When you want to place a virtual object, you need to define an anchor to ensure that ARCore tracks the object's position over time. Oftentimes you create an anchor based on the pose returned by a hit test, as described in user interaction.

The fact that poses can change means that ARCore may update the position of environmental objects like planes and feature points over time. Planes and points are a special type of object called a trackable. Like the name suggests, these are objects that ARCore will track over time. You can anchor virtual objects to specific trackables to ensure that the relationship between your virtual object and the trackable remains stable even as the device moves around. This means that if you place a virtual Android figurine on your desk, if ARCore later adjusts the pose of the plane associated with the desk, the Android figurine will still appear to stay on top of the table.

4.1.6 Augmented Images

Augmented Images is a feature that allows you to build AR apps that can respond to specific 2D images such as product packaging or movie posters. Users can trigger AR experiences when they point their phone's camera at specific images - for instance, they could point their phone's camera at a movie poster and have a character pop out and enact a scene.

ARCore also tracks moving images such as, for example, a billboard on the side of a moving bus.

Images can be compiled offline to create an image database, or individual images can be added in real time from the device. Once registered, ARCore will detect these images, the images' boundaries, and return a corresponding pose.

4.2 A* Shortest Path Finding Algorithm

A* Search algorithm is one of the best and popular techniques used in path-finding and graph traversals.

4.2.1 Explanation

Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithm comes to the rescue.

What A* Search Algorithm does is that at each step it picks the node according to a value-‘f’ which is a parameter equal to the sum of two other parameters – ‘g’ and ‘h’. At each step it picks the node/cell having the lowest ‘f’, and process that node/cell.

We define ‘g’ and ‘h’ as simply as possible below

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.

h = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don’t know the actual distance until we find the path,

because all sorts of things can be in the way (walls, water, etc.). There can be many ways to calculate this ‘h’ which are discussed in the later sections.

4.2.2 Algorithm

We create two lists – Open List and Closed List (just like Dijkstra Algorithm)

```
// A* Search Algorithm
1.  Initialize the open list
2.  Initialize the closed list
    put the starting node on the open
    list (you can leave its f at zero)
3.  while the open list is not empty
    a) find the node with the least f on
       the open list, call it "q"
    b) pop q off the open list
    c) generate q's 8 successors and set their
       parents to q
    d) for each successor
        i) if successor is the goal, stop search
          successor.g = q.g + distance between
                              successor and q
          successor.h = distance from goal to
          successor (This can be done using many
          ways, we will discuss three heuristics-
          Manhattan, Diagonal and Euclidean
          Heuristics)

          successor.f = successor.g + successor.h

        ii) if a node with the same position as
            successor is in the OPEN list which has a
           lower f than successor, skip this successor
        iii) if a node with the same position as
            successor  is in the CLOSED list which has
```

```
        a lower f than successor, skip this successor

        otherwise, add  the node to the open list

 end (for loop)

e) push q on the closed list
end (while loop)
```

So suppose as in the below figure if we want to reach the target cell from the source cell, then the A* Search algorithm would follow the path as shown below. Note that the below figure is made by considering Euclidean Distance as a heuristics.

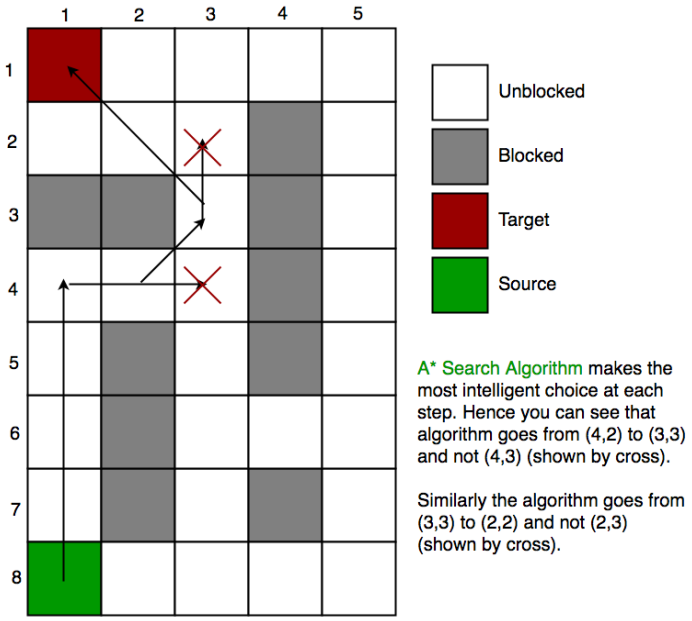

Figure 4.2.1: A* Search Algorithm

4.2.3 Heuristics

Exact Heuristics –

We can find exact values of h, but that is generally very time consuming.

Below are some of the methods to calculate the exact value of h.

1) Pre-compute the distance between each pair of cells before running the A* Search Algorithm.

2) If there are no blocked cells/obstacles then we can just find the exact value of h without any pre-computation using the distance formula/Euclidean Distance

Approximation Heuristics –

There are generally three approximation heuristics to calculate h –

1) Manhattan Distance –

- It is nothing but the sum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively, i.e.,

```
h = abs (current_cell.x – goal.x) +
    abs (current_cell.y – goal.y)
```

- When to use this heuristic? – When we are allowed to move only in four directions only (right, left, top, bottom)

The Manhattan Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).

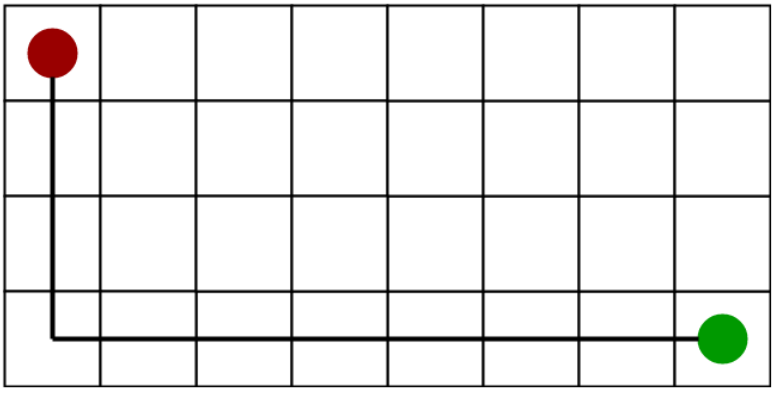

Figure 4.2.2: Manhattan Distance Heuristics

2) Diagonal Distance-

- It is nothing but the maximum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively, i.e.,

```
h = max { abs(current_cell.x – goal.x),
          abs(current_cell.y – goal.y) }
```

- When to use this heuristic? – When we are allowed to move in eight directions only (similar to a move of a King in Chess)

The Diagonal Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).

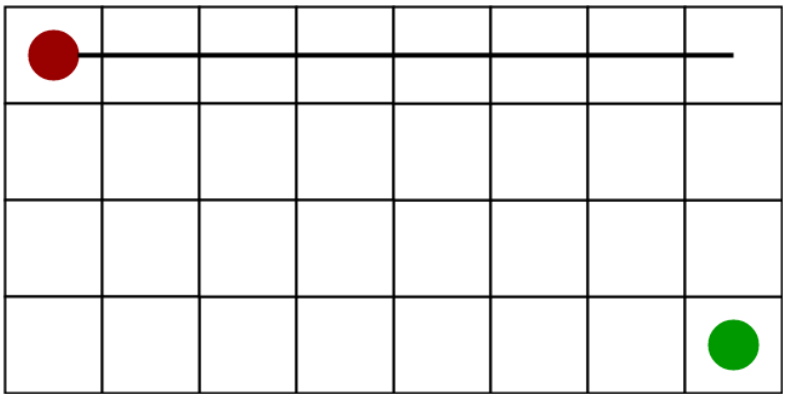

Figure 4.2.3: Diagonal Distance Heuristics

3) Euclidean Distance-

- As it is clear from its name, it is nothing but the distance between the current cell and the goal cell using the distance formula

```
h = sqrt ( (current_cell.x – goal.x)2 +
           (current_cell.y – goal.y)2 )
```

- When to use this heuristic? – When we are allowed to move in any direction.

The Euclidean Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).

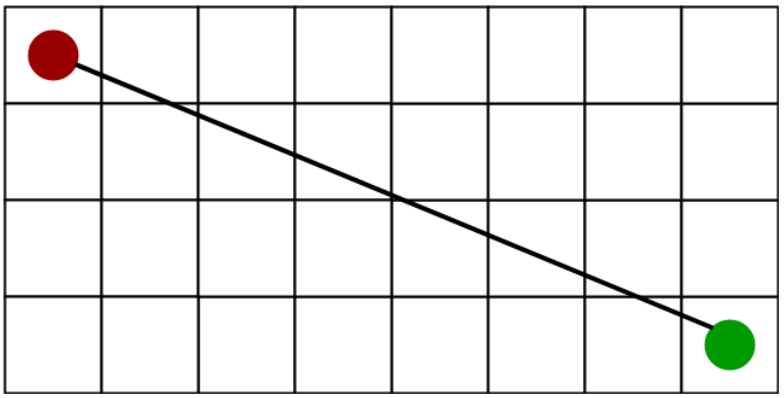

Figure 4.2.4: Euclidean Distance Heuristics

4.3 Navigation in Unity

The Navigation System allows you to create characters which can navigate the game world. It gives your characters the ability to understand that they need to take stairs to reach the second floor, or to jump to get over a ditch. The Unity NavMesh system consists of the following pieces:

- **NavMesh** (short for Navigation Mesh) is a data structure which describes the walkable surfaces of the game world and allows one to find a path from one walkable location to another in the game world. The data structure is built, or baked, automatically from your level geometry.
- **NavMesh Agent** component helps you to create characters which avoid each other while moving towards their goal. Agents reason about the game world using the NavMesh and they know how to avoid each other as well as moving obstacles.
- **Off-Mesh Link** component allows you to incorporate navigation shortcuts which cannot be represented using a walkable surface. For example, jumping over a ditch or a fence, or opening a door before walking through it, can be all described as Off-mesh links.
- **NavMesh Obstacle** component allows you to describe moving obstacles the agents should avoid while navigating the world. A barrel or a crate controlled by the physics system is a good example of an obstacle. While the obstacle is

moving the agents do their best to avoid it, but once the obstacle becomes stationary it will carve a hole in the navmesh so that the agents can change their paths to steer around it, or if the stationary obstacle is blocking the path way, the agents can find a different route.

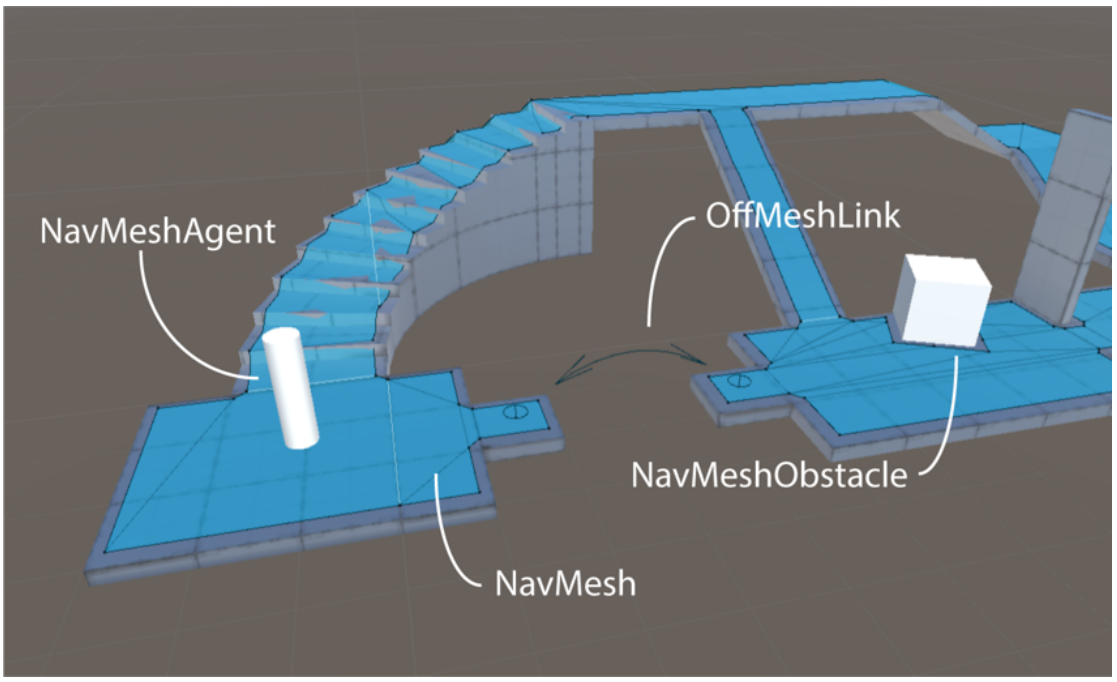

Figure 4.3.1: Navigation in Unity

4.3.1 NavMesh Agent

NavMeshAgent components help you to create characters which avoid each other while moving towards their goal. Agents reason about the game world using the NavMesh and they know how to avoid each other as well as other moving obstacles. Pathfinding and spatial reasoning are handled using the scripting API of the NavMesh Agent.

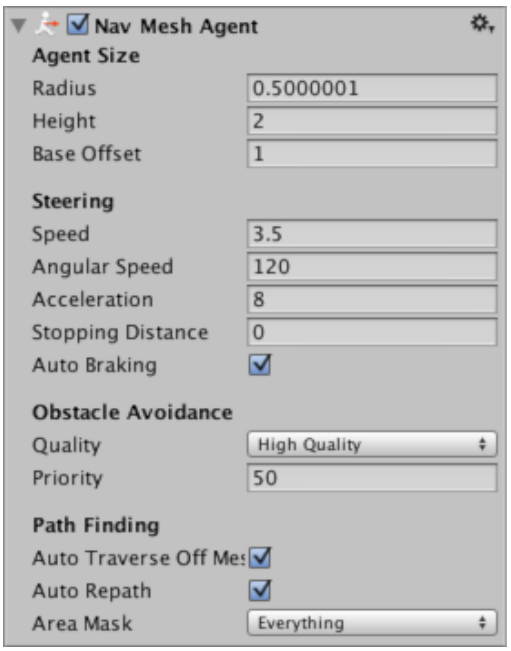

Figure 4.3.2: NavMesh Agent Component

Properties

Property	**Function**
Agent Size	
Radius	Radius of the agent, used to calculate **collisions between** obstacles and other agents.
Height	The height clearance the agent needs to pass below an obstacle overhead.
Base offset	Offset of the collision cylinder in relation to the transform pivot point.
Steering	
Speed	Maximum movement speed (in world units per second).
Angular Speed	Maximum speed of rotation (degrees per second).
Acceleration	Maximum acceleration (in world units per second squared).
Stopping distance	The agent will stop when this is close to the goal location.
Auto Braking	When enabled, the agent will slow down when reaching the destination. You should disable this for behaviors such patrolling, where the agent should move smoothly between multiple points

Obstacle Avoidance	
Quality	Obstacle avoidance quality. If you have a high number of agents you can save CPU time by reducing the obstacle avoidance quality. Setting avoidance to none, will only resolve collision, but will not try to actively avoid other agents and obstacles.
Priority	Agents of lower priority will be ignored by this agent when performing avoidance. The value should be in the range 0–99 where lower numbers indicate higher priority.
Path Finding	
Auto Traverse OffMesh Link	Set to true to automatically traverse off-mesh links. You should turn this off when you want to use animation or some specific way to traverse off-mesh links.
Auto Repath	When enabled the agent will try to find a path again when it reaches the end of a partial path. When there is no path to the destination, a partial path is generated to the closest reachable location to the destination.
Area Mask	Area mask describes which area types the agent will consider when finding a path. When you prepare meshes for NavMesh baking, you can set each meshes area type. For example you can mark stairs with a special area type, and forbid some character types from using the stairs.

Table 4.3.1: NavMesh Agent Component Properties

Details

The agent is defined by an upright cylinder whose size is specified by the Radius and Height properties. The cylinder moves with the object but always remains upright even if the object itself rotates. The shape of the cylinder is used to detect and respond to collisions between other agents and obstacles. When the GameObject ’s anchor point is not at the base of the cylinder, you can use the Base Offset property to take up the height difference.

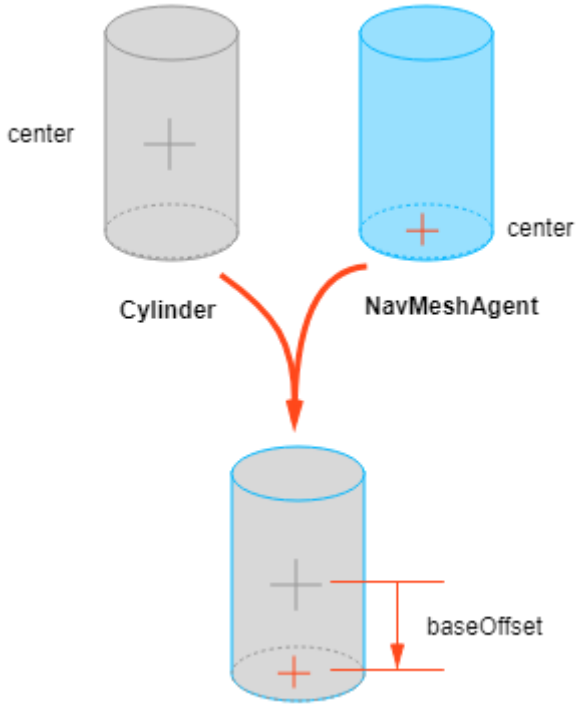

Figure 4.3.3: An Example of NavMesh on a Cylinder

The height and radius of the cylinder are actually specified in two different places: the NavMesh bake settings and the properties of the individual agents.

NavMesh bake settings describe how all the NavMesh Agents are colliding or avoiding the static world geometry. In order to keep memory on budget and CPU load in check, only one size can be specified in the bake settings.

NavMesh Agent properties values describe how the agent collides with moving obstacles and other agents.

Most often you set the size of the agent the same in both places. But, for example, a heavy soldier may have a larger radius, so that other agents will leave more space around him, but otherwise he'll avoid the environment just the same.

4.3.2 NavMesh Obstacle

The NavMesh Obstacle component allows you to describe moving obstacles that NavMesh Agents should avoid while navigating the world (for example, barrels or crates controlled by the physics system). While the obstacle is moving, the NavMesh Agents do their best to avoid it. When the obstacle is stationary, it carves a hole in the

NavMesh. Nav Mesh Agents then change their paths to steer around it, or find a different route if the obstacle causes the pathway to be completely blocked.

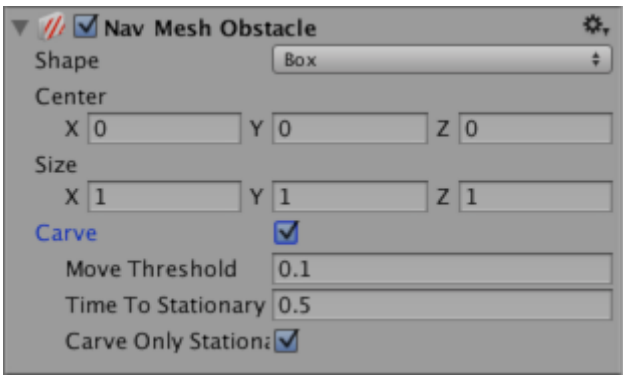

Figure 4.3.4: NavMesh Obstacle Component

Properties

Property	Function
Shape	The shape of the obstacle geometry. Choose whichever one best fits the shape of the object.
Box	
Center	Center of the box relative to the transform position.
Size	Size of the box.
Capsule	
Center	Center of the capsule relative to the transform position.
Radius	Radius of the capsule.
Height	Height of the capsule.
Carve	When the Carve checkbox is ticked, the Nav Mesh Obstacle creates a hole in the NavMesh.
Move Threshold	Unity treats the Nav Mesh Obstacle as moving when it has moved more than the distance set by the Move Threshold. Use this property to set the threshold distance for updating a moving carved hole.
Time To Stationary	The time (in seconds) to wait until the obstacle is treated as stationary.

Carve Only Stationary	When enabled, the obstacle is carved only when it is stationary. See Logic for moving NavMesh Obstacles, below, to learn more.

Table 4.3.2: NavMesh Obstacle Properties

Details

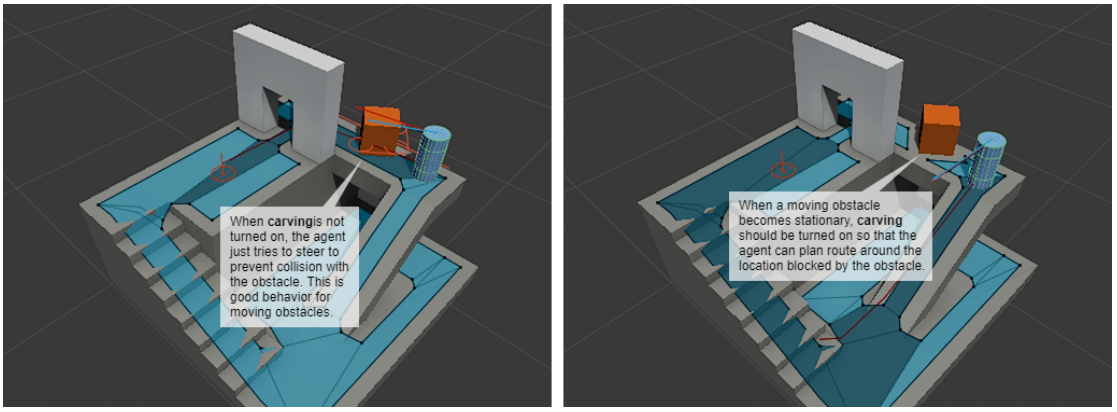

Figure 4.3.5: An Example for NavMesh Obstacle

Obstructing

When Carve is not enabled, the default behavior of the Nav Mesh Obstacle is similar to that of a Collider. Nav Mesh Agents try to avoid collisions with the Nav Mesh Obstacle, and when close, they collide with the Nav Mesh Obstacle. Obstacle avoidance behaviour is very basic, and has a short radius. As such, the Nav Mesh Agent might not be able to find its way around in an environment cluttered with NavMesh Obstacles. This mode is best used in cases where the obstacle is constantly moving (for example, a vehicle or player character).

4.3.3 Inner Workings of the Navigation System

When you want to intelligently move characters in your game (or agents as they are called in AI circles), you have to solve two problems: how to reason about the level to find the destination, then how to move there. These two problems are tightly coupled, but quite different in nature. The problem of reasoning about the level is more global and static, in that it takes into account the whole scene Moving to the destination is more local and dynamic, it only considers the direction to move and how to prevent collisions with other moving agents.

Walkable Areas

The navigation system needs its own data to represent the walkable areas in a game scene. The walkable areas define the places in the scene where the agent can stand and move. In Unity the agents are described as cylinders. The walkable area is built automatically from the geometry in the scene by testing the locations where the agent can stand. Then the locations are connected to a surface laying on top of the scene geometry. This surface is called the navigation mesh (NavMesh for short).

The NavMesh stores this surface as convex polygons. Convex polygons are a useful representation, since we know that there are no obstructions between any two points inside a polygon. In addition to the polygon boundaries, we store information about which polygons are neighbours to each other. This allows us to reason about the whole walkable area.

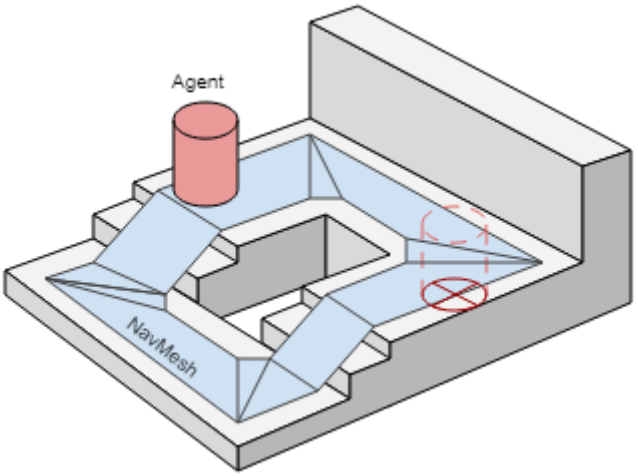

Figure 4.3.6: NavMesh Walkable Areas

Finding Paths

To find a path between two locations in the scene, we first need to map the start and destination locations to their nearest polygons. Then we start searching from the start location, visiting all the neighbours until we reach the destination polygon. Tracing the visited polygons allows us to find the sequence of polygons which will lead from the start to the destination. A common algorithm to find the path is A* (pronounced “A star”), which is what Unity uses.

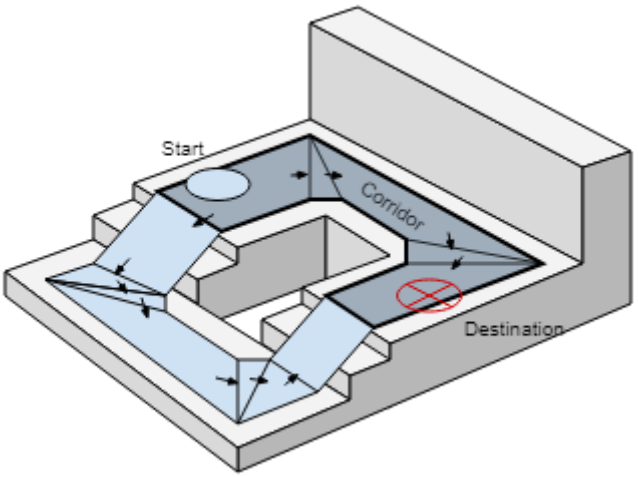

Figure 4.3.7: Finding Paths using NavMesh

Following the Path

The sequence of polygons which describe the path from the start to the destination polygon is called a corridor. The agent will reach the destination by always steering towards the next visible corner of the corridor. If you have a simple game where only one agent moves in the scene, it is fine to find all the corners of the corridor in one swoop and animate the character to move along the line segments connecting the corners. When dealing with multiple agents moving at the same time, they will need to deviate from the original path when avoiding each other. Trying to correct such deviations using a path consisting of line segments soon becomes very difficult and error prone.

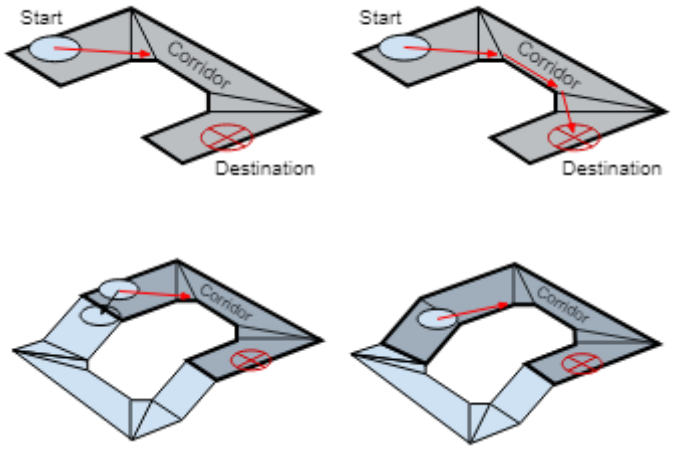

Figure 4.3.8: Following the Path

Since the agent movement in each frame is quite small, we can use the connectivity of the polygons to fix up the corridor in case we need to take a little detour. Then we quickly find the next visible corner to steer towards.

Avoiding Obstacles

The steering logic takes the position of the next corner and based on that figures out a desired direction and speed (or velocity) needed to reach the destination. Using the desired velocity to move the agent can lead to collision with other agents.

Obstacle avoidance chooses a new velocity which balances between moving in the desired direction and preventing future collisions with other agents and edges of the navigation mesh. Unity is using reciprocal velocity obstacles (RVO) to predict and prevent collisions.

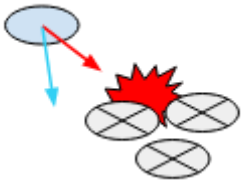

Figure 4.3.9: Avoiding Obstacles

Moving the Agent

Finally after steering and obstacle avoidance the final velocity is calculated. In Unity the agents are simulated using a simple dynamic model, which also takes into account acceleration to allow more natural and smooth movement.

At this stage you can feed the velocity from the simulated agent to the animation system to move the character using root motion, or let the navigation system take care of that.

Once the agent has been moved using either method, the simulated agent location is moved and constrained to NavMesh. This last small step is important for robust navigation.

4.4 System Architecture

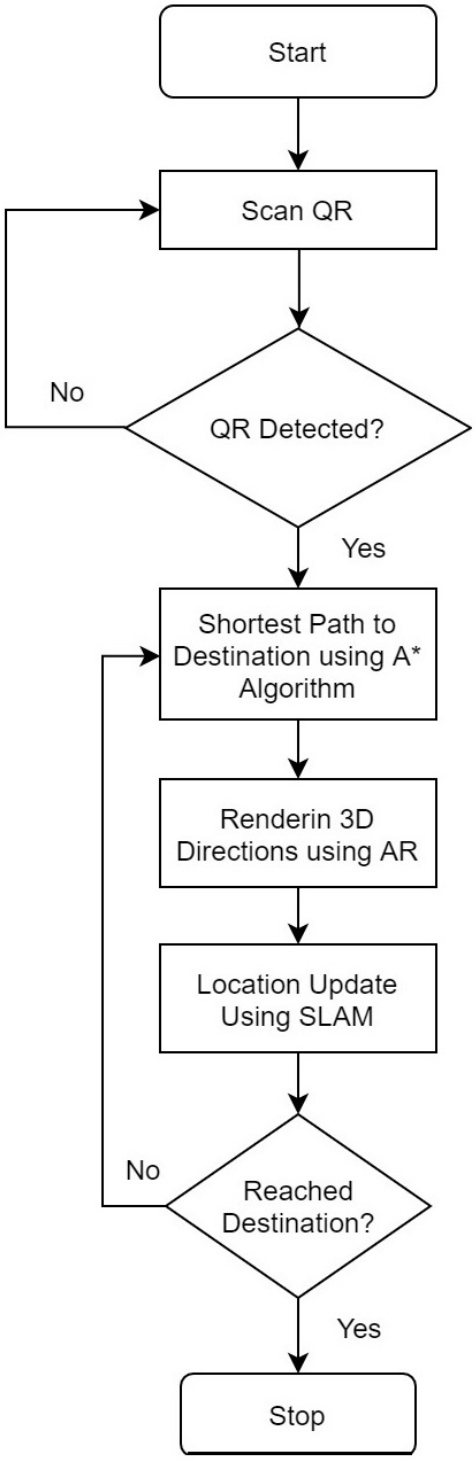

Figure 4.4.1: Flowchart of the Proposed System

5. IMPLEMENTATION

In order to run the project Unity is required.The following steps have to be followed in order to install and set up the Unity Engine. First we need to download the Unity Download Assist from the Unity Download page.

The Unity Download Assistant is a small executable program (approximately 1 MB in size) which lets you select which components of the Unity Editor you want to download and install.

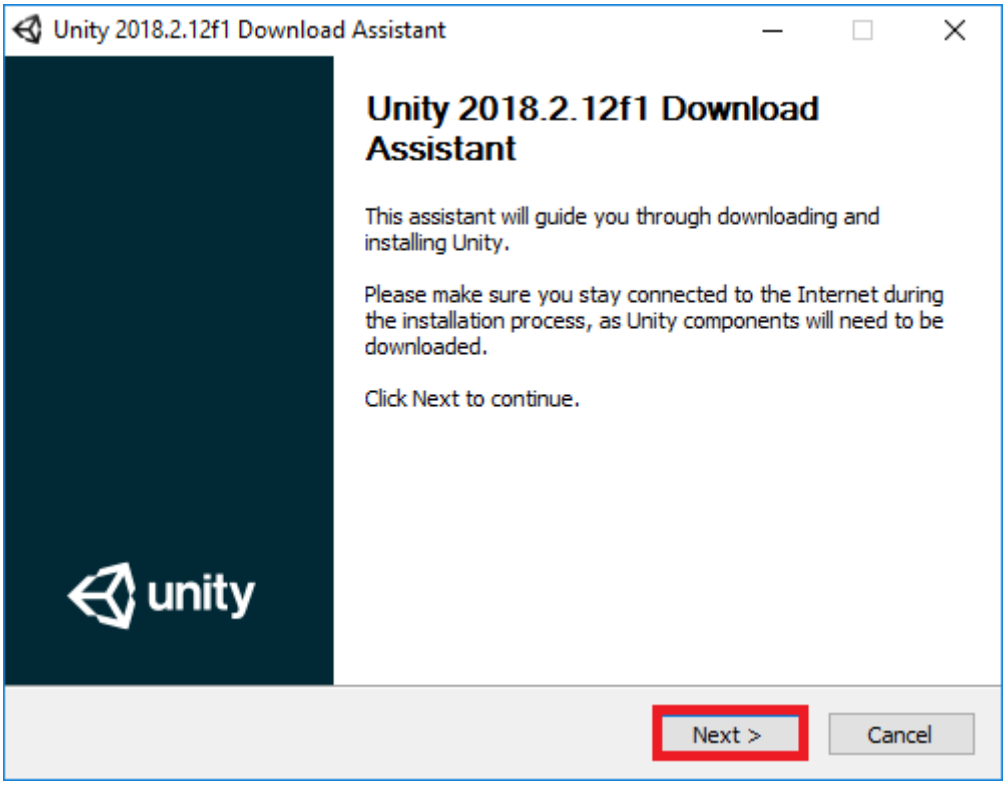

Figure 5.1: Unity Download Assist window

After clicking continue, a dialog box appears as shown in Figure 5.2

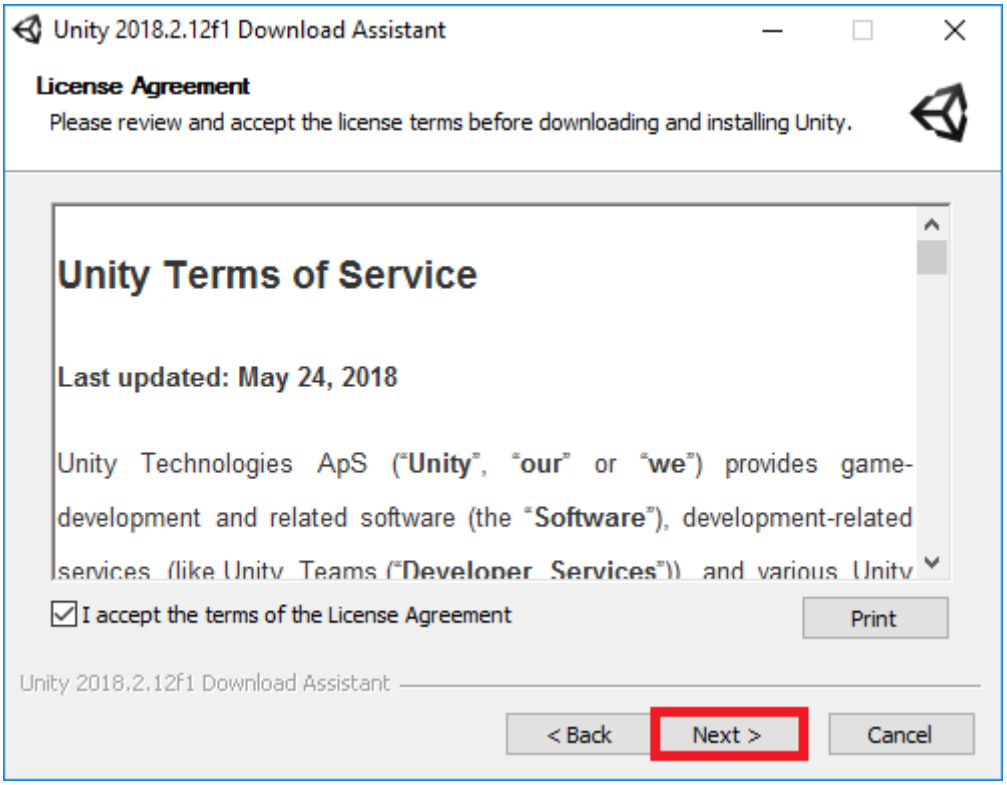

Figure 5.2: Unity License Agreement window

Accept the license and terms and click Next, will let you choose components to install.

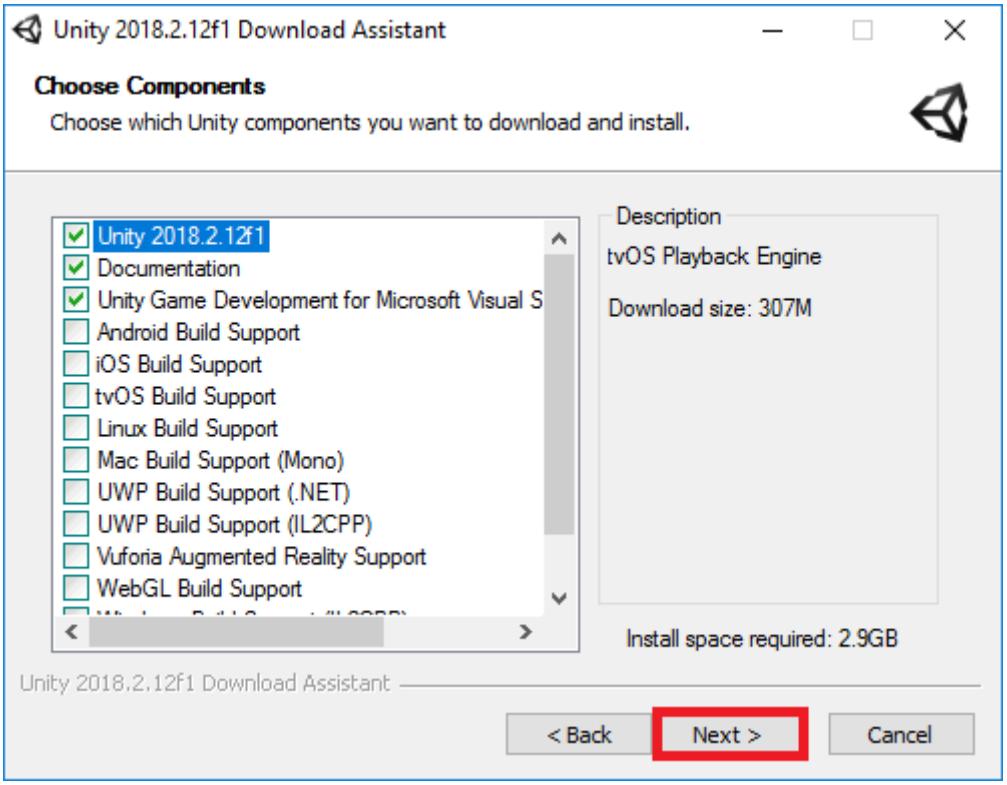

Figure 5.3: Unity components selection window

Selecting the components you would like to be installed with Unity and click “Next”, will let you change the location where unity is installed.

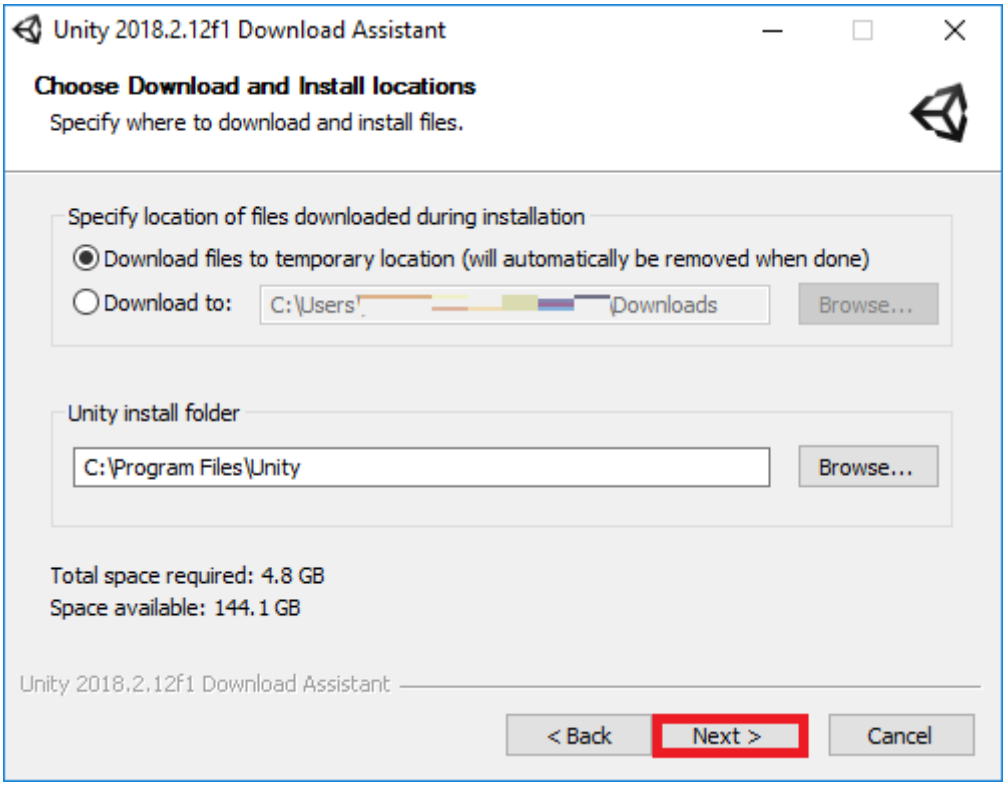

Figure 5.4: Unity Install location selection window

You can change where you want Unity installed, or leave the default option and click “Next”. Depending on the components you selected, you may see additional prompts before installing. Follow the prompts and click “Install”. Installing Unity may take some time. After the installation is finished, Unity will be installed on your computer.

Step 1: Created a 3D model of the building using blender:

Blender’s comprehensive array of modeling tools make creating, transforming, sculpting and editing your models a breeze. First, open Blender. You should see the blender window, with a single cube helpfully available. The default blender setup is insanely customizable. Perhaps too much so any panel can become any menu, menus can be located anywhere on the screen, and so on. Then we can change it to edit mode. Once we are in edit mode, we can edit vertices, edges or

faces. Then we can change the vertices, edges, or faces and add more objects to make the 3D Model that we need(in our case, the model of the building).

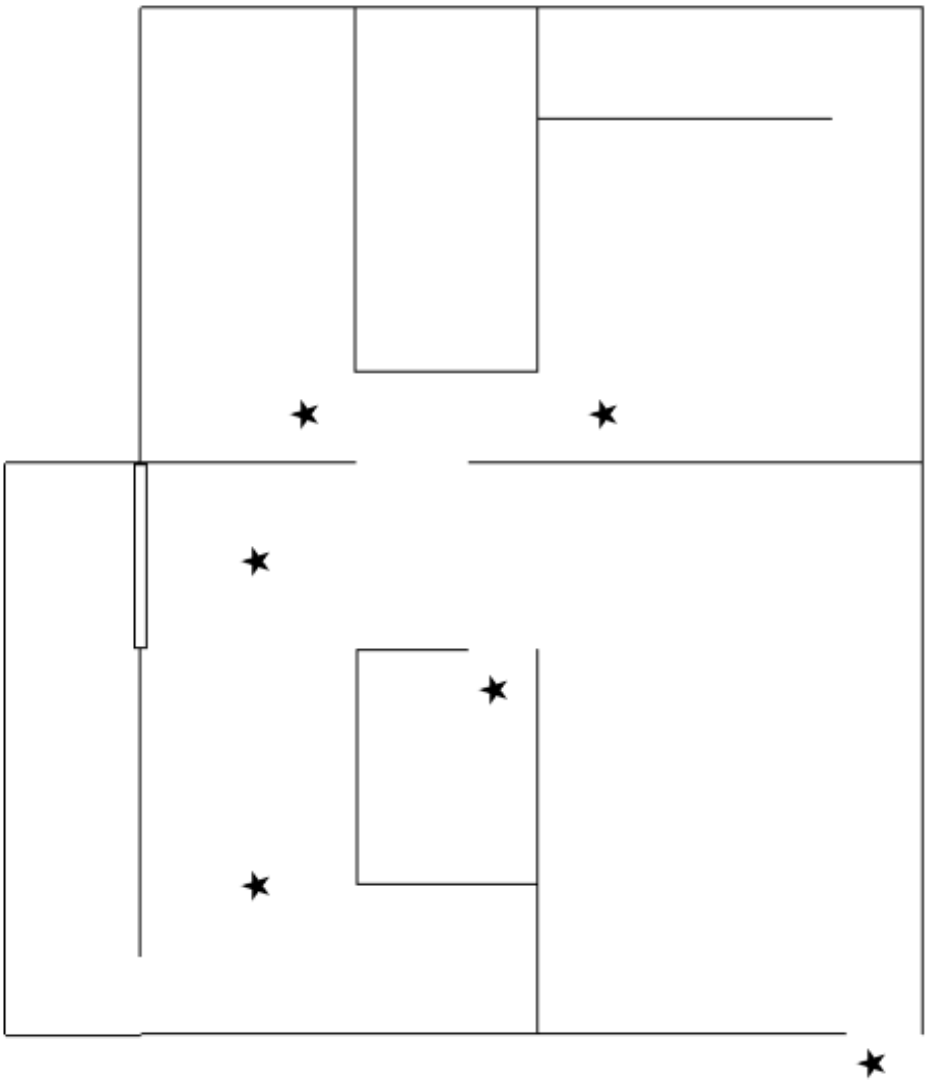

Figure 5.5: Floor plan for building the 3D model.

Step 2: Integrated the model with unity and added obstacles:

There are two formats that Unity can deal with in regards to Blender: native Blender files (.blend) and FBX. Importing a Blender file to Unity can be as simple as drag and drop. For multiple models in a single Blender file, it's probably easier to export each one as an FBX file and then import them one at a time.When a Blender file is imported into Unity, it will call Blender's export scripts to automatically create

an FBX file before actually being imported. This action obviously requires Blender to be installed on the machine, and that might not be the case for some users.

There were some inconsistencies between the coordinate systems of Blender and Unity, as the X-, Y-, and Z-axes don't match (right-handed vs. left-handed coordinate systems) and the origins weren't accurately imported. The model's scale was also changed by default during importation.

We Followed these steps to properly prepare the model for Unity:

1. With the model opened, in the right side-bar region, going to the "Transform" tab.

2. Set all the rotation axes (X, Y, Z) to 0°.

3. Set all the axis scales (X, Y, Z) to zero 1,0.

4. Select the "Cursor Tool" in the top left.

5. Select the new origin on the model by clicking where you want to place it.

6. Go to "Object > Set Origin > Origin to 3D Cursor".

7. Go to "Edit Mode" and select all faces.

8. Go to "Mesh > Normals > Recalculate Outside".

9. Save your file.

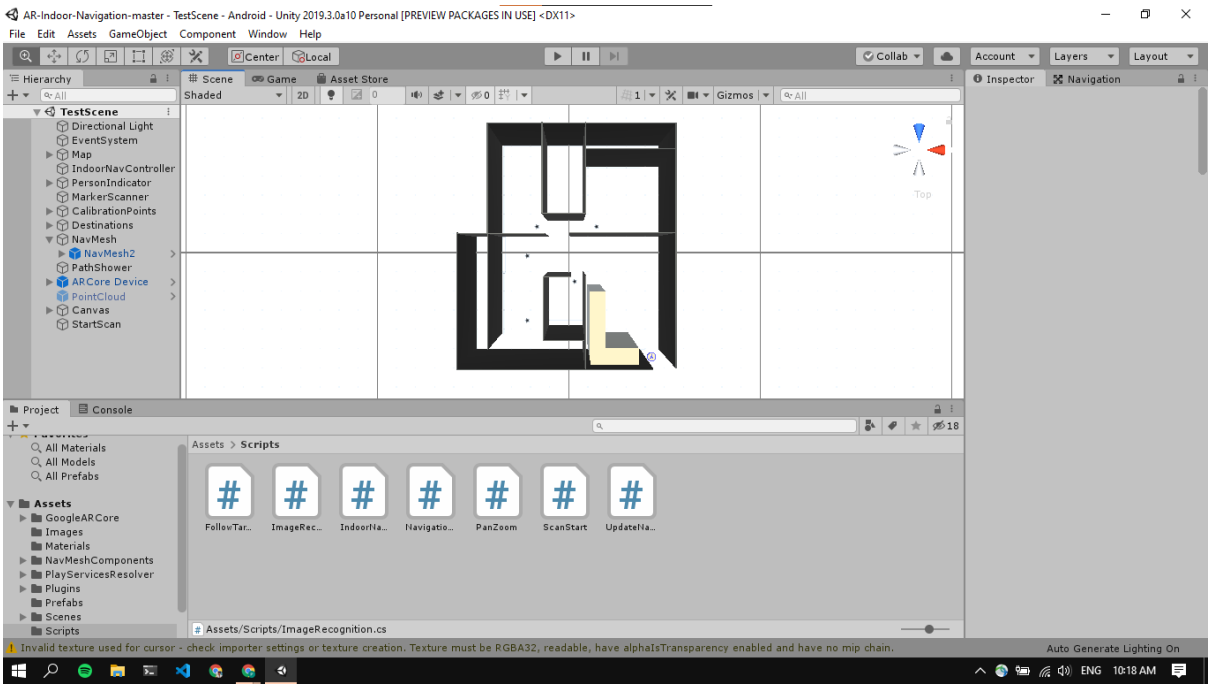

Figure 5.6: 3D Model of the building imported to Unity.

Step 3: Created a 3D Model of the Arrow to display using Augmented Reality in real time

Figure 5.7: 3D Model of the Arrow

Step 4: Implemented a relation of AR Camera in the application with the device's camera:

After importing the 3D Model of the building to Unity, We selected all the possible destinations that the user can go to. And then we added an AR camera to the scene. Then, to move the AR Camera as the user moves in the real world, we made the Camera of the smartphone as a child to the AR Camera. We also added SLAM(Simultaneous Localization and Mapping) to update both the cameras in real time.

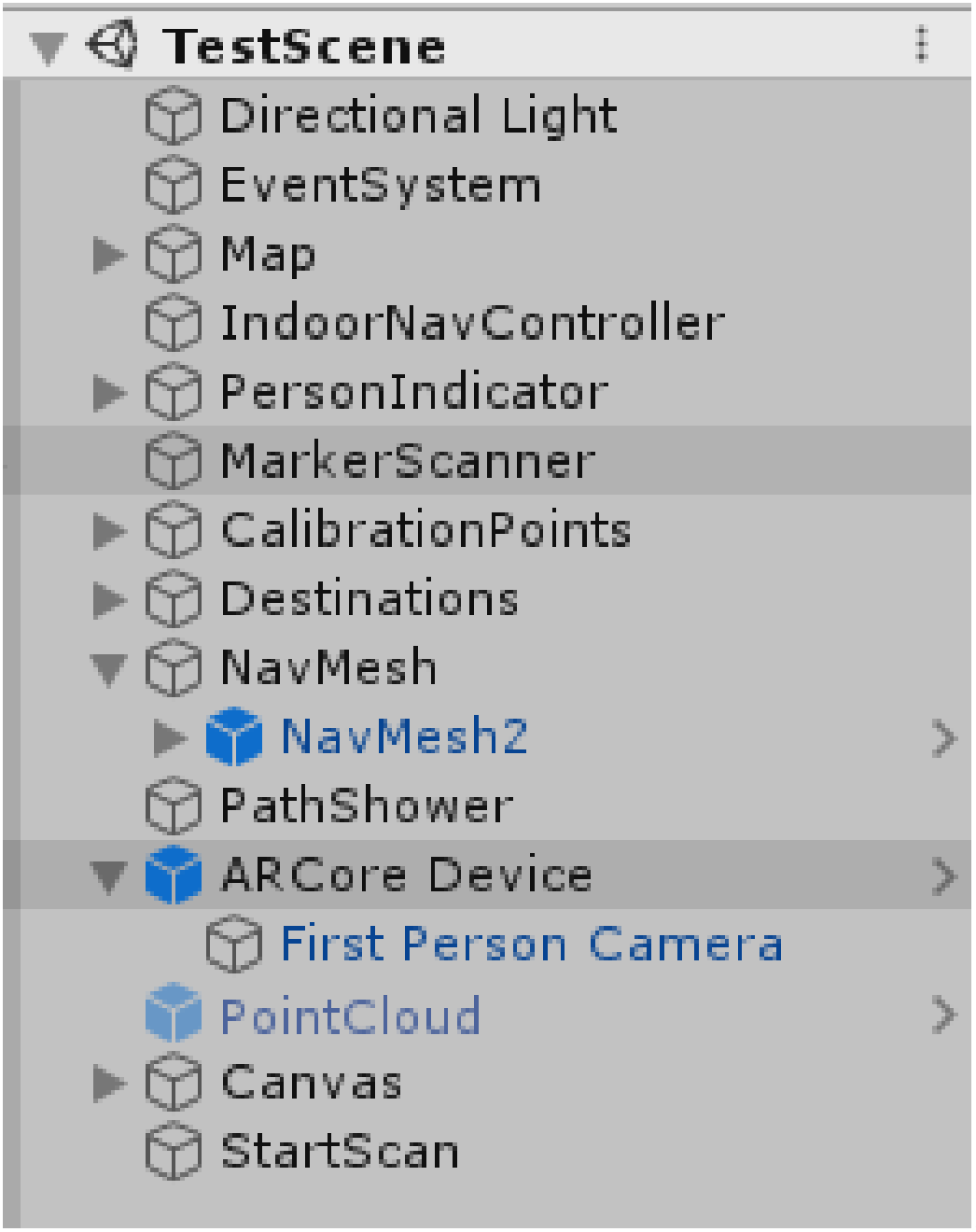

Figure 5.8: Device's camera as the child for the AR Camera

Step 5: Built a Navmesh Agent for getting the directions:

While calculating the path from the source to any given destination, we need to consider all the obstacles like walls, doors, tables and any other considerable obstacles. To overcome this we created a mesh around all the obstacles using NavMesh and the script for navigation considers these obstacles to generate the shortest path.

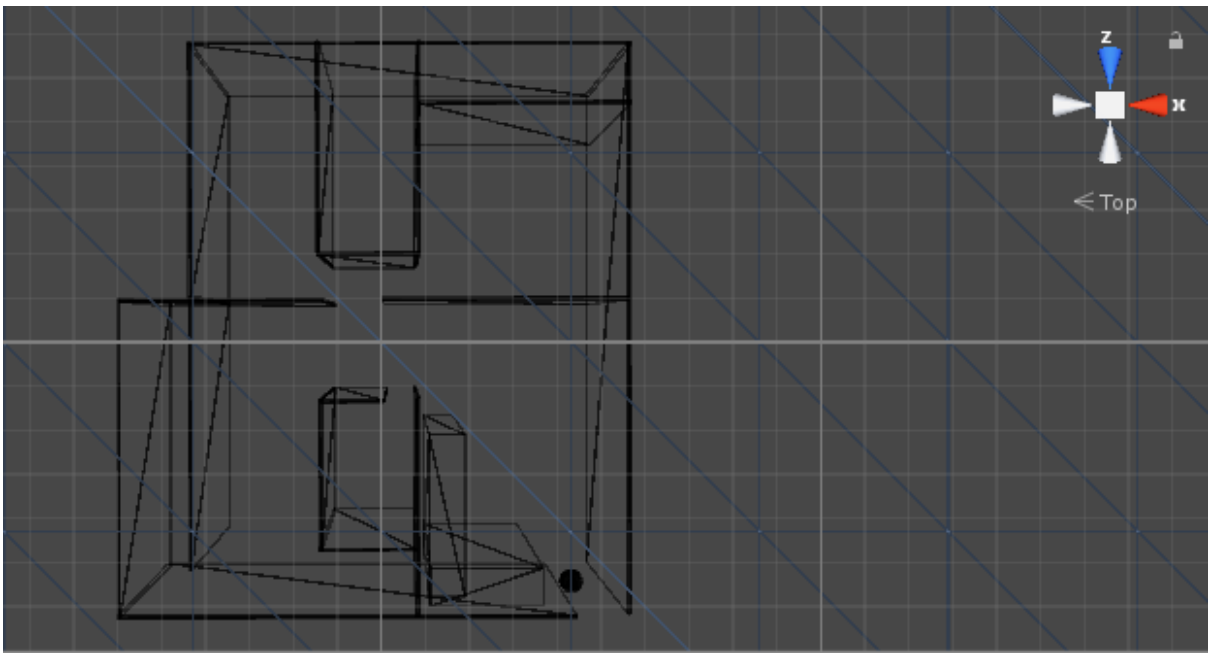

Figure 5.9: NavMesh that is built around the obstacles

Step 6: Getting to know the location of the User from scanning a QR

QR codes are being used in many areas such as: education, marketing, the gaming industry and businesses. In our daily lives QR codes provide an opportunity to share or receive additional information about a place or product. Coupled with the fact that Apple's iOS 11 update added QR code scanning capability in the camera app along with the latest Android smartphones being able to do the same, QR codes have become an integral part of everyday life.

Figure: 5.10: QR Code sample used for getting user's initial location

Using QR codes, the navigation map identifies the user's location and places a 3D object on the smartphone's screen. 3D objects are presented by arrows which set the direction to the next point. We ask the user to scan a QR on initializing the application. Once the QR Code is properly detected, we use that to get the current location of the user.

Script for Scanning QR:

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

public class ScanStart : MonoBehaviour
{
    public GameObject ardevice;
    private bool camAvailable;
    private WebCamTexture backCam;
    private Texture defaultBackground;
    public RawImage background;
    public AspectRatioFitter fit;
    public ImageRecognition imgRec;
    public GameObject text;
```

```
        public GameObject scanOverlay;
        private void Start()
        {
            defaultBackground = background.texture;
            WebCamDevice[] devices = WebCamTexture.devices;

            if(devices.Length == 0)
            {
                Debug.Log("No camera detected");
                camAvailable = false;
                return;
            }
            for(int i = 0; i < devices.Length; i++)
            {
                if (!devices[i].isFrontFacing)
                {
                    backCam = new WebCamTexture(devices[i].name,
Screen.width, Screen.height);
                }
            }
            if(backCam == null)
            {
                Debug.Log("unable to find backcam");
                return;
            }
            backCam.Play();
            background.texture = backCam;
            camAvailable = true;
        }
        private void Update()
        {
            if (!camAvailable)
            {
                return;
            }
            float ratio = (float)backCam.width / (float)backCam.height;
            fit.aspectRatio = ratio;
            float scaleY = backCam.videoVerticallyMirrored ? -1f: 1f;
            background.rectTransform.localScale = new Vector3(1f, scaleY,
1f);
            int orient = -backCam.videoRotationAngle;
            background.rectTransform.localEulerAngles = new Vector3(0, 0,
orient);
```

```
            bool result = imgRec.StartPosition(backCam);
            if (result)
            {
                ardevice.GetComponent<ARCoreSession>().enabled = true;
                background.gameObject.SetActive(false);
                text.SetActive(false);
                scanOverlay.SetActive(false);
                this.gameObject.SetActive(false);
            }
        }
        public void OnClick()
        {
            ardevice.GetComponent<ARCoreSession>().enabled = true;
            background.gameObject.SetActive(false);
            text.SetActive(false);
            scanOverlay.SetActive(false);
            this.gameObject.SetActive(false);
        }
    }
```

Step 7: Implementing a script to navigate user to target:

We developed a script for calculating the shortest path between the current position of the user and the destination, and displays the path using Augmented Reality Arrows on screen, which also gets updated in real time. This script is run until the user reaches his destination. Upon reaching the destination, a 3d marker is displayed which implies that the destination is reached.

Script for following the target:

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class FollowTarget : MonoBehaviour
{
    public Transform targetToFollow;
    public Quaternion targetRot;
    public RawImage minimap;
    public Camera fullscreenCamera;
    public GameObject arrow;
    public GameObject scanButton;
    public GameObject switchButton;
    public float rotationSmoothingSpeed = 1.5f;
```

```
        private bool map = false;
        private bool pressed = false;
        private RenderTexture texture;
        void LateUpdate()
        {
            if (!targetToFollow)
                return;
            Vector3 targetEulerAngles = targetRot.eulerAngles;
            if(targetRot.x > 0.5)
            {
                if(!map)
                {
                    map = true;
                    gameObject.GetComponent<Camera>().enabled = false;
                    minimap.gameObject.SetActive(false);
                    texture = fullscreenCamera.targetTexture;
                    fullscreenCamera.targetTexture = null;
                    fullscreenCamera.orthographicSize = 15;
                    scanButton.SetActive(false);
                    switchButton.SetActive(false);
                }
            } else
            {
                if(map)
                {
                    map = false;
                    gameObject.GetComponent<Camera>().enabled = true;
                    minimap.gameObject.SetActive(true);
                    fullscreenCamera.targetTexture = texture;
                    fullscreenCamera.orthographicSize = 7;
                    scanButton.SetActive(true);
                    switchButton.SetActive(true);
                }
            }
            float rotationToApplyAroundY = targetEulerAngles.y;
            float newCamRotAngleY =
Mathf.LerpAngle(arrow.transform.eulerAngles.y, rotationToApplyAroundY,
rotationSmoothingSpeed * Time.deltaTime);
            Quaternion newCamRotYQuat = Quaternion.Euler(0,
newCamRotAngleY, 0);
            if(targetEulerAngles.x < 65)
            {
                arrow.transform.rotation = newCamRotYQuat;
```

```
        }
    }
    public void Switch()
    {
        if (!pressed)
        {
            pressed = true;
            gameObject.GetComponent<Camera>().enabled = false;
            minimap.gameObject.SetActive(false);
            texture = fullscreenCamera.targetTexture;
            fullscreenCamera.targetTexture = null;
            fullscreenCamera.orthographicSize = 15;
            scanButton.SetActive(false);
        }
        else
        {
            pressed = false;
            gameObject.GetComponent<Camera>().enabled = true;
            minimap.gameObject.SetActive(true);
            fullscreenCamera.targetTexture = texture;
            fullscreenCamera.orthographicSize = 7;
            scanButton.SetActive(true);
        }
    }
}
```

Step 7: Tested and Deployed the application on an Android device:

For Testing the application, we chose Android platform, which requires the exported file to be in APK format. For that we needed to configure the Build settings in Unity. To configure and build apps for Android, access the Build Settings window (File > Build Settings). In Platform, select Android. To set Android as our default build platform, click the Switch Platform button. After we specify our build settings, click the Build button to create our build. To build the app, click Build And Run to create and run your build on the platform we have specified. The results from our testing are in the following section.

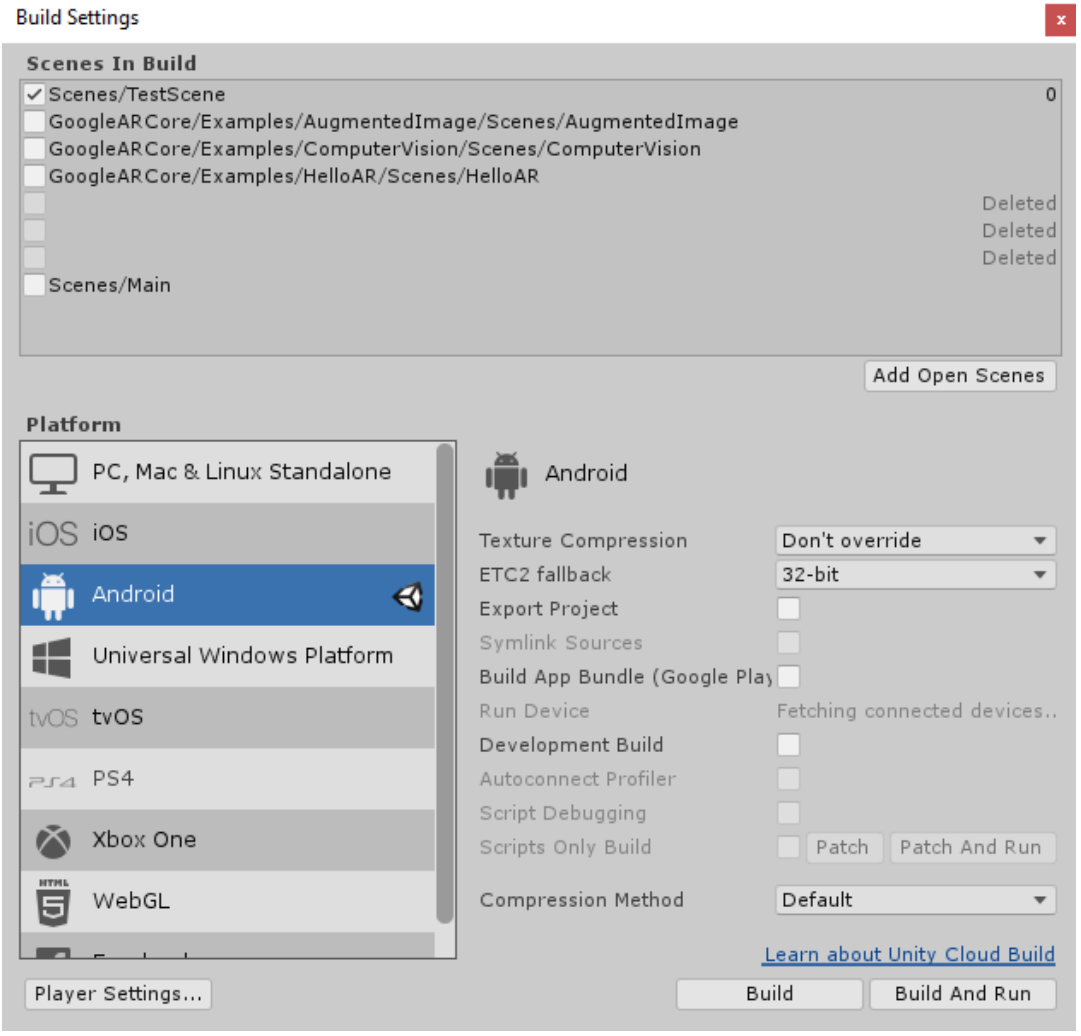

Figure 5.11: Build Settings window

In order to make the smartphone available to run the application, we need to enable the debugging by doing the following: The mobile has to be connected to the system using a usb cable and then the usb debugging options has to be enabled in order to deploy the mobile

application in the actual physical mobile. To enable the usb debugging options, first settings in mobile have to be opened and their build number as shown in figure below has to be tapped 7 times in order to get the developer options. After that in developer options usb debugging should be turned on. Once the usb debugging is enabled, the device name can be seen in the Build settings window. Then click on Build and Run to get the application on your device.

6. RESULTS

The Application starts off with asking the user to scan a QR code as shown in figure 6.1 for the application to know the users location. The user shall scan the nearest QR code. After the user scans a QR code, a list of destinations are given to choose one. Once the user selects a destination, the path to the destination from the user's current location is displayed in the map view and as shown in figure 6.2.

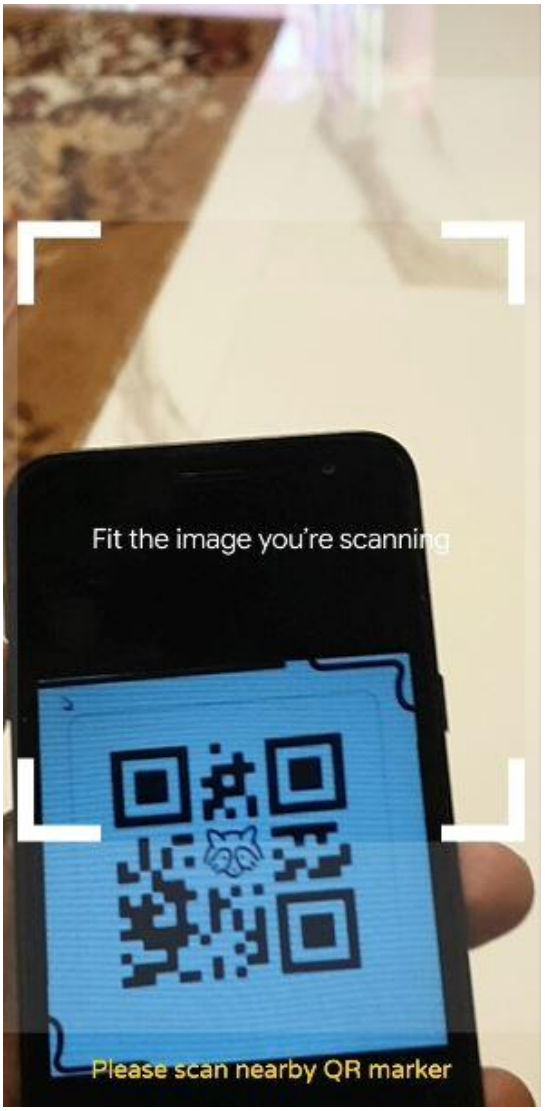

Figure 6.1: Result Screenshot of application scanning the QR

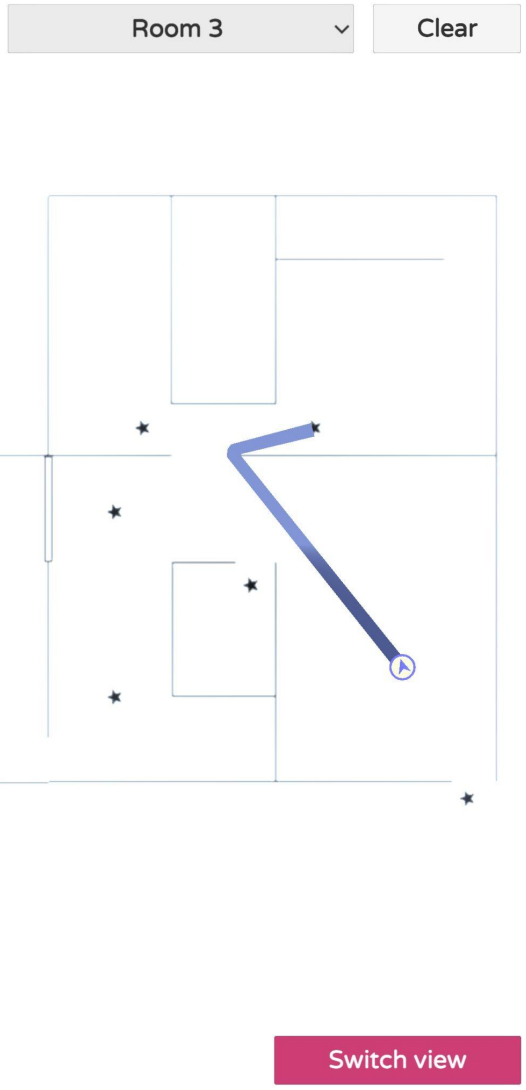

Figure 6.2: Result Screenshot of the map showing path to destination

When the user switches from map view to camera view, a 3D arrow object will be visible showing the directions to the destination as shown in figure 6.3. The user will reach the destination following the directions given by the arrow object. Once the user reaches the destination, a 3D pin object is displayed to indicate that the destination has arrived as shown in figure 6.4.

Figure 6.3: Result Screenshot of the AR view displaying the route to destination

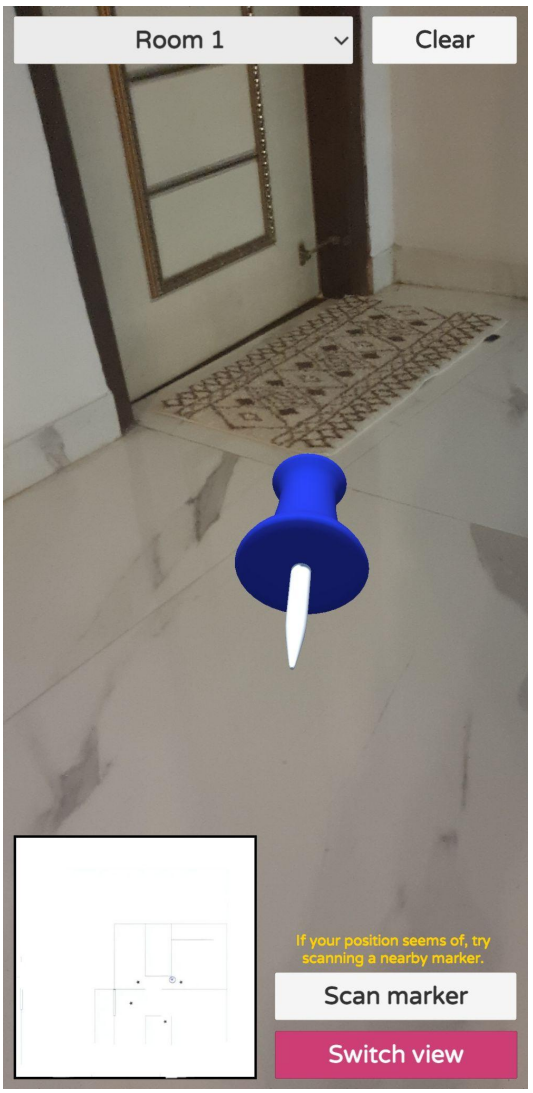

Figure 6.4: Result Screenshot of the Pin Marker shown after reaching the destination

7. CONCLUSION AND FUTURE SCOPE

In this project, we have developed an Indoor Navigation Application with Augmented Reality using QR codes. This application can be installed in any Android/iOS smartphone. The five major steps to achieve this are:

1. Developing the 3D map using the building floor plan.
2. QR-code based positioning of the user.
3. Google ARCore based simultaneous localization and mapping.
4. Finding shortest path to chosen destination using NavMesh (A * algorithm).
5. Navigation in the AR view.

Due to the current scenarios we could only make the application for our house, so in future when things get normal, we would like to build this application on a bigger scale, i.e, for a Shopping Mall or an Organization.

BIBLIOGRAPHY

1. Bluetooth-based Indoor Navigation Mobile System by Adam Satan
   Source: IEEE
2. Indoor Navigation System Using Visual Positioning System with Augmented Reality by Ravinder Yadav, Vandit Jain, Himanika Chugh, Prasenjit Banerjee
   Source: IEEE
3. A multi-functional method of QR code used during the process of indoor navigation by Daria Mamaeva, Mikhail Afanasev, Vitaliy Bakshaev, Mark Kliachin
   Source: IEEE
4. Navigation System in Unity. Source: Unity Documentation.
5. Google ARCore Fundamental Concepts. Source: Google Developers.
6. Indoor positioning system. Source: Wikipedia
7. Indoor Navigation: The Complete Guide. Source: Nearmotion