The Wanderer: Implementing Markerless Augmented Reality with Object Position Awareness

Mina Samir, Ahmed Hanie, Aly Aboulgheit, Karim Hossam
Supervised by: Dr. Ayman Ezzat and Eng. Noha Elmasry

February 13, 2018

1 Introduction

1.1 Purpose

In This document, we will describe the Wanderer's core flow of data. We will present our class diagram that will define the construction of our game in an object oriented manner. The database will map the stored data inside our project. The sequence diagram defines the roles of the player and the admin. We will also talk about the server and how it functions, what are the algorithms that are in use.

1.2 Scope

The Wanderer aims to entertain out-going people that enjoy moving around and having fun. The game is a location-based augmented reality game that everyone should enjoy. The user will feel that they are inside the game, taking adventures and completing tasks that will help them level up and become more adaptive toward the gameplay. The wanderer will also help by adding quests in points of interests like charity work, and blood donations vans. These locations can also be added into hot sales for advertisements to certain shops.

1.3 Overview

Augmented reality games has taken over the world by a surprise starting with Ingress to Pokemon GO [1]. This field makes the user entertained and help developers understand the use the markerless augmented reality algorithms. The idea of The Wanderer is that the user go to a specified real time place to accomplish a certain quest, this quest could be fetching something from a place and returning it, delivering a certain item from a place to another, or combating some enemies. The quest will be located using the smartphones sensors fused together. GPS to locate the quest in the map, using gyroscope to locate the orientation on the quest, and the compass to locate the quests direction relative to the geographic cardinal directions , once the user enters the field, the sensor fusion will activate to locate the quest. The quest will be superimposed by using the markerless augmented realitys algorithm object recognition and with the help of the smartphones sensor fusion, the item will be augmented on real time objects. Same strategy will be applied on the enemy, or items.

1.4 Definitions and Acronyms

1. Firebase: A mobile and web application platform.
2. OpenCv: a library of programming functions mainly aimed at real-time computer vision.
3. Unity Game Engine: a cross-platform game engine, which is primarily used to develop video games and simulations for computers, consoles and mobile devices.
4. Geolocation: estimation of the real-world geographic location of an object.
5. RPG: role-playing game is a game in which players assume the roles of characters in a ctional setting.
6. NPC: Non player character.
7. Buff: a temporary benecial status eect in some video games on a character or an enemy.
8. DeBuff: a temporary detrimental status eect in some video games on a character or an enemy.
9. Lure Modules: items players can use at a Pokestop to increase Pokemon Spawn Rates.

2 System Overview

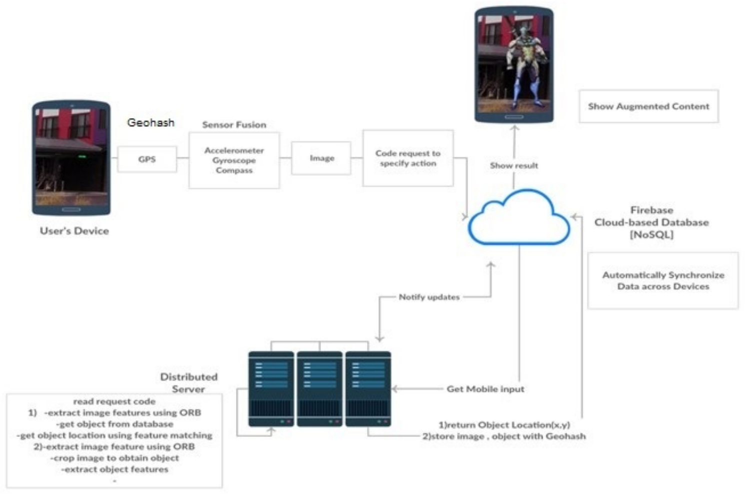

Geohash
Sensor Fusion
GPS
Accelerometer
Gyroscope
Compass
Image
Code request to specify action
User's Device
Show result
Show Augmented Content
Firebase
Cloud-based Database
[NoSQL]
Automatically Synchronize Data across Devices
Notify updates
Distributed Server
read request code
1) -extract image features using ORB
-get object from database
-get object location using feature matching
2)-extract image feature using ORB
-crop image to obtain object
-extract object features
Get Mobile input
1)return Object Location(x,y)
2)store image , object with Geohash

Fig 1: System Overview

2.1 Pre-processing

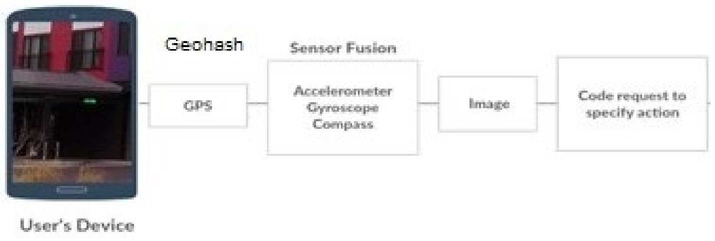


Fig 2: System Overview: Pre-processing

The mobile phone captures a frame from the camera, and the location of the user. The location is encoded by using Geohash. Geohash is a public domain geocoding, which encodes a geographic location into a short string of letters and digits. It is a hierarchical spatial data structure which subdivides space into buckets of grid shape, which is one of the many applications of what is known as a Z-order curve, and generally space-filling curves. The mobile device re-sizes the captured frame to be more manageable for processing and sending.

2.2 Database Management

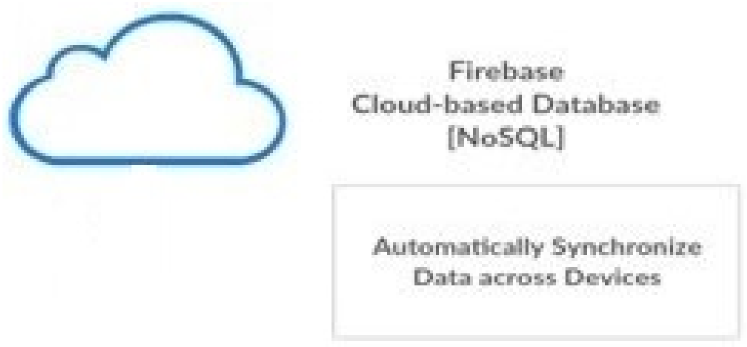


Fig 3: System Overview: Database Management

These information is sent to Firebase database for storage, and the server receive the captured data for further processing. Firebase is an intermediate between the server and the mobile device, which ensures the correct movement of the data from one to another. When the database gets any updates, it sends a notification to the other party notifying it of the new arrival of data and sending it if necessary.

2.3 Processing

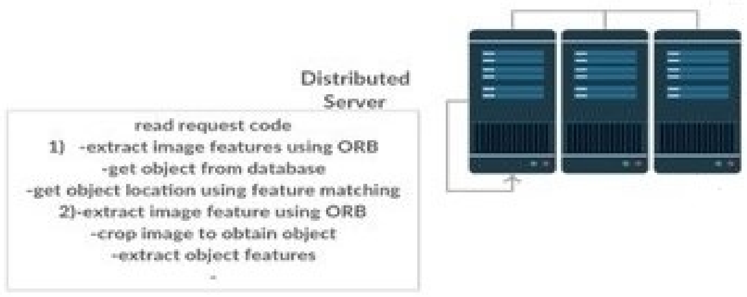

Distributed
Server
read request code
1) -extract image features using ORB
-get object from database
-get object location using feature matching
2)-extract image feature using ORB
-crop image to obtain object
-extract object features
-

Fig 4: System Overview: Processing

According to the logged token, if the logged in is an admin then the server will invoke response 2, else the server will invoke response 1. The admin response will get the frame, crop the frame to obtain the object and run feature extraction. The user response, the server gets the response and starts to apply feature extraction on the sent frame. After that, the server starts to compare between the stored images in the database and the sent image. If there was a match then the server sends the response to the database.

2.4 User Interface

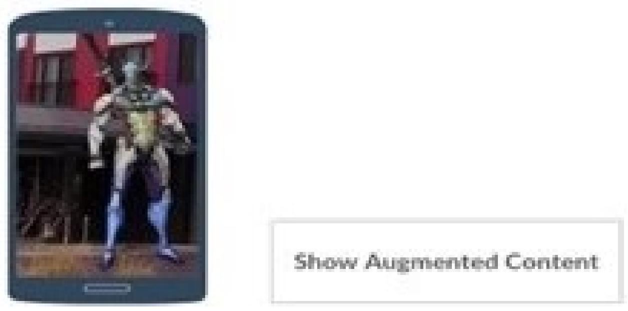


Fig 5: System Overview: User Interface

The database then sends the code for the object that is attached to the image according to the location and the sent processed frame from the server, the mobile selects the object from the game's files and superimpose the virtual object on the real object.

3 System Architecture

3.1 Architectural Design

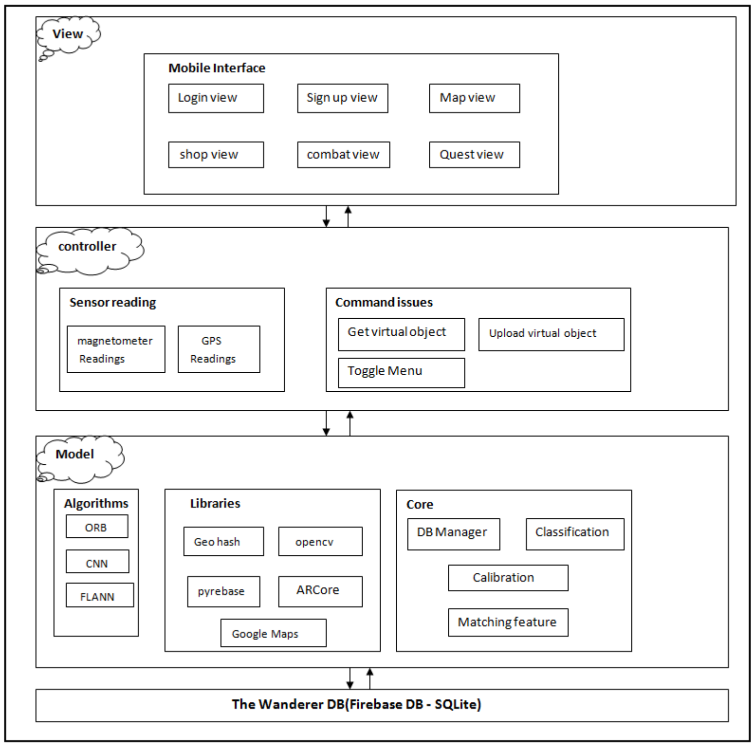


Fig 6: Architectural Design

3.1.1 Model

1. Algorithm:

   - ORB: The Oriented and Fast rotated Brief algorithm is a fast feature detector that uses keypoint detection. First it finds the keypoints and then computes the intensity weighted centroid of the patch with located corner at center. These keypoints marks the important parts inside a frame, which is small in size and more easy to compute when wanting to compare frames.
   - CNN [2]: In order to invoke object detection, the server should have some kind of intelligence to contain the incoming dataset. In this scenario, we are using CNN deep learning algorithm. The server takes a frame resize and transpose to match current dataset, then the server pushes the processed data inside an array to adjure into a

classifier. The incoming frame will be compared with the classifiers in the database, if the frame passes the threshold then the object exists and the server will send the response to the database in order for the mobile to superimpose the virtual object.

- FLANN: FLANN algorithm takes each keypoint and starts to apply a brute force algorithm to compare each keypoint, and returns a list of common features. According to these common features, the algorithm decides whether the image is the same or not by adding a threshold number that filters the bad percentage from the good one.

2. Libraries:

    - Geohash[3]: Geohash is a public domain geocoding, which encodes a geographic location into a short string of letters and digits. It is a hierarchical spatial data structure which subdivides space into buckets of grid shape. We use these grids to filer the objects when the user enters them, this will help on superimposing the objects not only with image detection algorithm, but with the use of GPS location.
    - Google Maps: Google Maps is a web mapping service developed by Google. It offers satellite imagery, street maps, real-time traffic conditions, and route planning for traveling by foot, car, bicycle, or public transportation. The use of google maps helps the admin on adding an object for the user by uploading the frame along with the geohash from the google maps library for filtration.
    - Pyrebase: Pyrebase is a Python interface to Firebases REST API. It allows you to use Python to manipulate your Firebase database.
    - OpenCV: OpenCV is a library of programming functions mainly aimed at real-time computer vision. It was designed for computational efficiency and with a strong focus on real-time applications. It is also used with ORB and FLANN, along side converting the frame to gray scale and re-sizing the frame.
    - ARCore: ARCore's motion tracking technology uses the phone's camera to identify interesting points, called features, and tracks how those points move over time. With a combination of the movement of these points and readings from the phone's inertial sensors, ARCore determines both the position and orientation of the phone as it moves through space.

3. Core:

    - DBManager: The Database Manager takes control by notifying the server with recent updates, like if there's a new frame arriving or if the connection is closed. The server also uses it when finishing processing by either sending the location of an object inside a frame, or storing the processed image along with its geohash.

- Classification: It's used for classify the objects captured using the mobile device camera. These objects are processed and then added to a dataset that is later used for comparison and identifying similar frames later
- Matching Features: After extraction features, the algorithms takes the features of the new frame and compare it with the dataset stored in the database.
- Calibration: Calibration is the process that combines the library ARCore with our project, using ARCore will help superimposing virtual objects in the real world easily with the integration of Unity in it.

3.1.2 Controller

1. Sensor Readings:
   - Compass Readings: The compass helps getting the heading of the mobile device using the north pole. These reading can help getting the orientation of the mobile device when the user is searching for the virtual object
   - GPS Reading: The GPS is used to filter the objects for the user. When the user enter a certain place, the server will alert the user for an object existing in this place, and the user will search for it.
2. Command Issues
   - Get virtual object: this command is used to get the exact location of an object inside a frame. This location will be snapped on a real object by putting the virtual object accordingly to the pixels of the frame.
   - Upload virtual object: When the admin wants to add an object, the admin will capture the frame and then adds the coordinate of the virtual object inside the frame by pressing on the intended place. The controller will then take the location of the object inside the frame (pixels) for the user when searching will appear in the intended place .
   - Toggle Menu: This command will assist with the user interface. According to the object when appearing, this function will activate the proposed interface and closing it. For example, when the user finds a quest, this function will open the quest menu for the user to accept or refuse the quest. Or when the user stumbles in a monster, this function will open the interface for the combat mechanism.

3.1.3 View

1. Mobile Interface

- Login view: This interface is meant for the users to login the game. Logging in will determine the user's information as well as the progress that the user has reached, the user's quests, and inventory.
- Shop view: The shop view will have the trader's inventory. Each trader has an inventory containing items for the user to use. These items are weapons, armor, potions, or poisons to use inside the combat.
- Combat view: The combat view contains the combat mechanism. This will interacting with the monster the user faced, either by attacking using weapons or magic, defending incoming attacks, using a potion or poison, or fleeing from the monster.
- Sign up view: This interface allows new users to play the game, by entering their information for the database to store.
- Quest view: This menu contains all the user's quest; either current quests, completed quests or active quests. Also having the information of each quest as well as teh objectives inside each quest.
- Map View: This is a map that contains nearby shops or quests for the user to go to these locations to interact with them.

3.2 Class Diagram

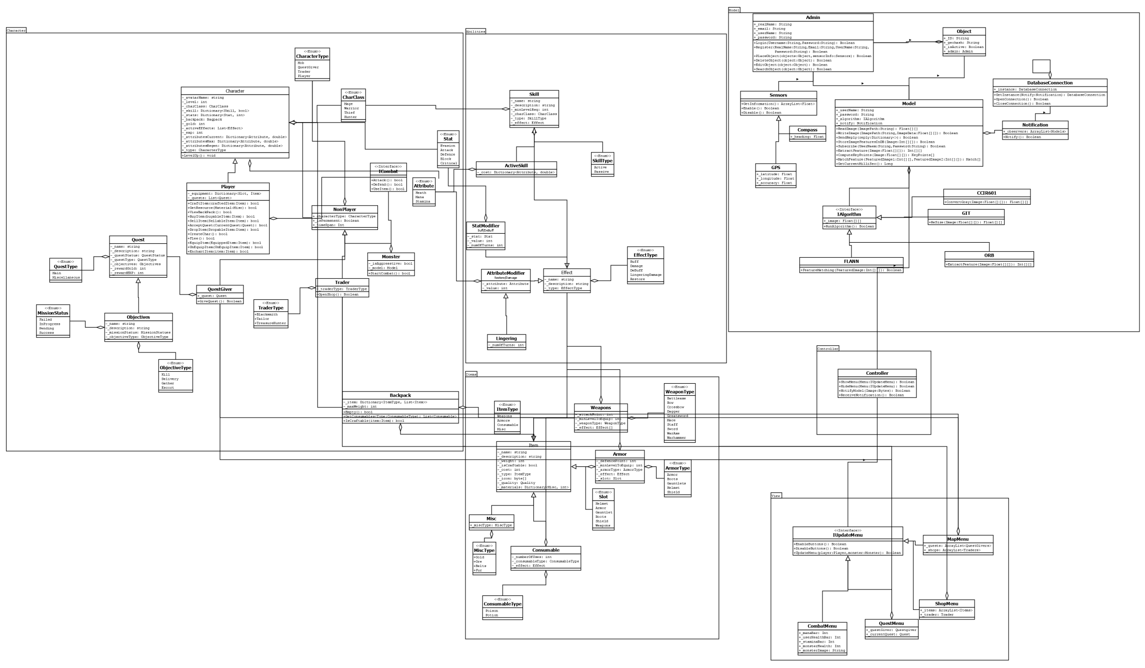


Fig 6: The Wanderer's Class Diagram

3.2.1 Character

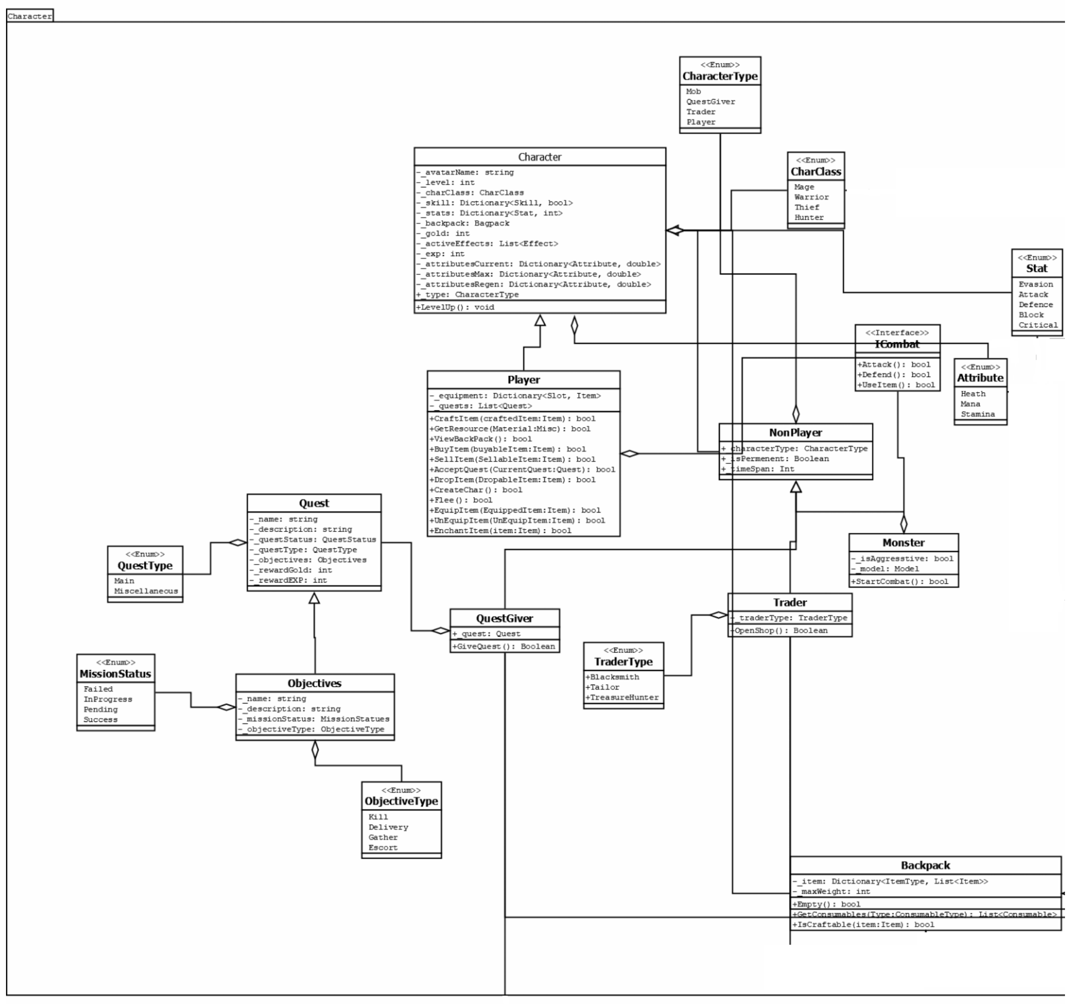


Fig 7: Class Diagram: Character Package

This package describes all the characters that appear inside the game. First we have the master class that defines all the attributes that any character will have. Secondly we have the player, or the user, and their capacity inside the game. The NPCs are what guids the user to play through, like the Quest giver who gives the quest to the player, or the monster that the user attacks. The Quest is the main component that contains the quest of each user and NPC.

3.2.2 Abilities

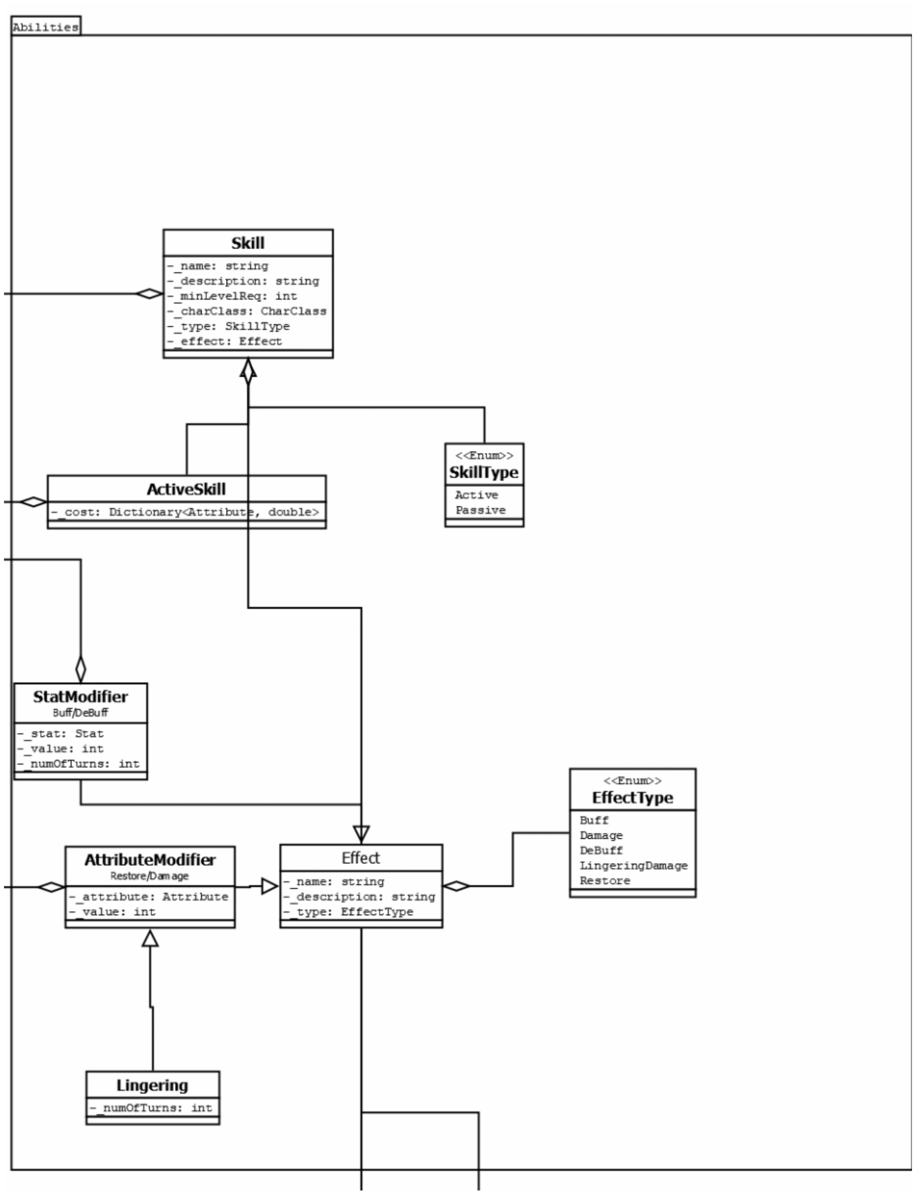


Fig 8: Class Diagram: Abilities Package

This package manages all of the player's special abilities like if there's an actives skill active on a player like magic attacks. These skills are decided by the player whether if they want to specialize in an attribute. The Effects class manages certain attributes like recover health, damage monster's health, increase attack point.

3.2.3 Items

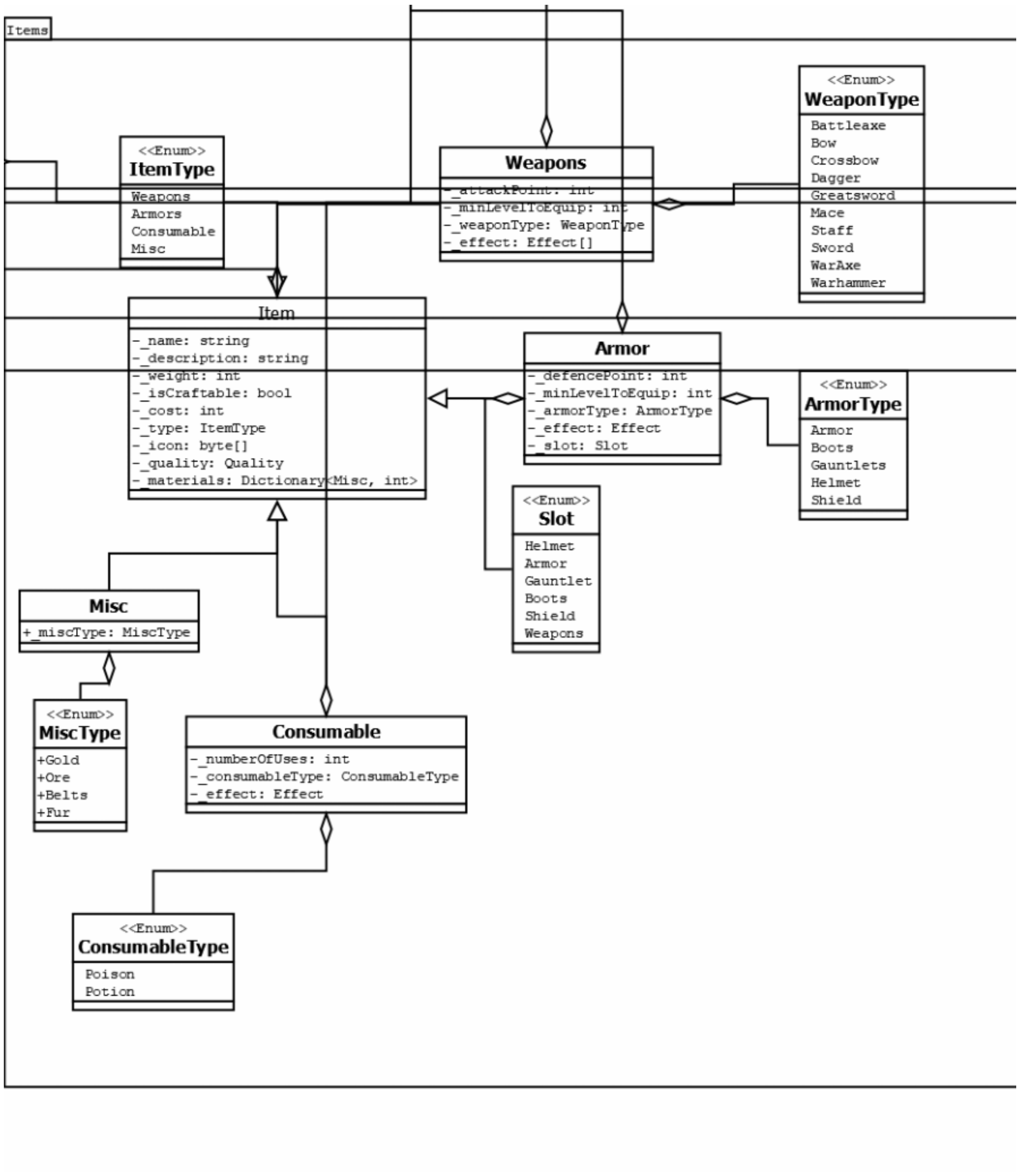


Fig 9: Class Diagram: Items Package

This package manages all the items inside the game. Weapons for the player to attack monsters, the higher level the more attack point. Armor for defense and to decrease the incoming attacks. Consumables for the player to use, either potion to reviving your attributes, or increase hit point, or poison to decrease monster's attributes. And misc for the player to trade for gold or craft items with them.

3.2.4 Design Patterns

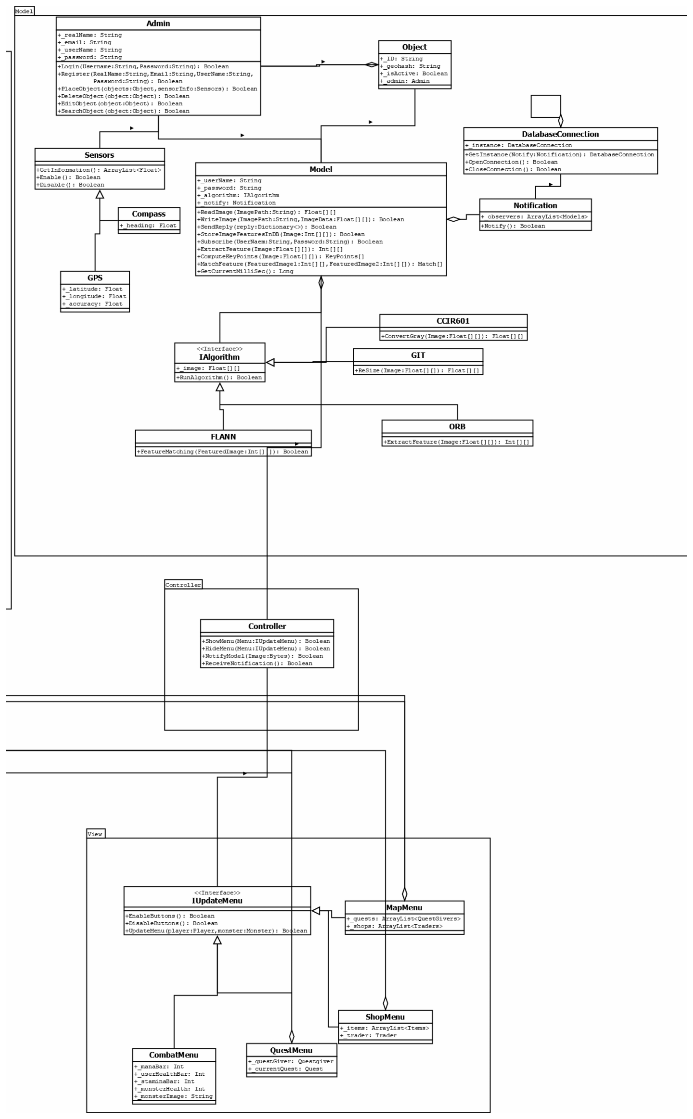

Model
Admin
+_realName: String
+_email: String
+_userName: String
+_password: String
+Login(Username:String,Password:String): Boolean
+Register(RealName:String,Email:String,UserName:String, Password:String): Boolean
+PlaceObject(objects:Object,sensorInfo:Sensors): Boolean
+DeleteObject(object:Object): Boolean
+EditObject(object:Object): Boolean
+SearchObject(object:Object): Boolean
Object
+_ID: String
+_geohash: String
+_isActive: Boolean
+_admin: Admin
DatabaseConnection
+_instance: DatabaseConnection
+GetInstance(Notify:Notification): DatabaseConnection
+OpenConnection(): Boolean
+CloseConnection(): Boolean
Sensors
+GetInformation(): ArrayList<Float>
+Enable(): Boolean
+Disable(): Boolean
Model
+_userName: String
+_password: String
+_algorithm: IAlgorithm
+_notify: Notification
+ReadImage(ImagePath:String): Float[][]
+WriteImage(ImagePath:String,ImageData:Float[][]): Boolean
+SendReply(reply:Dictionary<>): Boolean
+StoreImageFeaturesInDB(Image:Int[][]): Boolean
+Subscribe(UserNaem:String,Password:String): Boolean
+ExtractFeature(Image:Float[][]): Int[][]
+ComputeKeyPoints(Image:Float[][]): KeyPoints[]
+MatchFeature(FeaturedImage1:Int[][],FeaturedImage2:Int[][]): Match[]
+GetCurrentMilliSec(): Long
Notification
+_observers: ArrayList<Models>
+Notify(): Boolean
Compass
+_heading: Float
GPS
+_latitude: Float
+_longitude: Float
+_accuracy: Float
CCIR601
+ConvertGray(Image:Float[][]): Float[][]
<<Interface>>
IAlgorithm
+_image: Float[][]
+RunAlgorithm(): Boolean
GIT
+ReSize(Image:Float[][]): Float[][]
ORB
+ExtractFeature(Image:Float[][]): Int[][]
FLANN
+FeatureMatching(FeaturedImage:Int[][]): Boolean
Controller
Controller
+ShowMenu(Menu:IUpdateMenu): Boolean
+HideMenu(Menu:IUpdateMenu): Boolean
+NotifyModel(Image:Bytes): Boolean
+ReceiveNotification(): Boolean
View
<<Interface>>
IUpdateMenu
+EnableButtons(): Boolean
+DisableButtons(): Boolean
+UpdateMenu(player:Player,monster:Monster): Boolean
MapMenu
+_quests: ArrayList<QuestGivers>
+_shops: ArrayList<Traders>
ShopMenu
+_items: ArrayList<Items>
+_trader: Trader
CombatMenu
+_manaBar: Int
+_userHealthBar: Int
+_staminaBar: Int
+_monsterHealth: Int
+_monsterImage: String
QuestMenu
+_questGiver: Questgiver
+_currentQuest: Quest

Fig 10: Class Diagram: Model, Controller, and View Packages

- Model: The Model manages the database and server processes. The processes are like feature extraction, frame resize, convert grey scale, or match features. The database manages the storage of frames, or processed data. The admin is the main class for the backend, where the admin place objects for the players to interact with them. These objects can be a NPC, item, quest, or a shop. The admin can also remove, edit and search for object using the database and the server.

- Singleton Design Pattern: A software design pattern that restricts the instantiate of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system

- Strategy Design Pattern: A software design pattern that enables selecting an algorithm at runtime. Our project uses the strategy pattern to select from one algorithm to another. (IAlgorithm Interface, ICombat Interface, IUpdateMenu)

- Observer Design Pattern A software design pattern in which an object maintains a list of its dependents and noties them automatically of any state changes. (Notication Class)

- Controller: The Controller's role is an intermediate between the backend server and the Interface and vice-versa. The controller receives notification from the server on which frame is incoming, according to the frame the controller shows to the user which menu to show. The controller also takes the events from the event listeners in the interface and sends these events to the server.

- View: The Interface IUpdateMenu is an interface that manages all the menus inside the game. There are 4 menus that the player may interact with. The shop menu containing the items that the player can purchase and the trader managing these items. The quest menu that contains all the player's quests and their information and the quest giver. The combat menu that have all the combat mechanism the player uses to kill the monster. The maps menu that the player access to know the location of shops and quests for the player go in a real location to interact with.

3.3 Activity Diagram

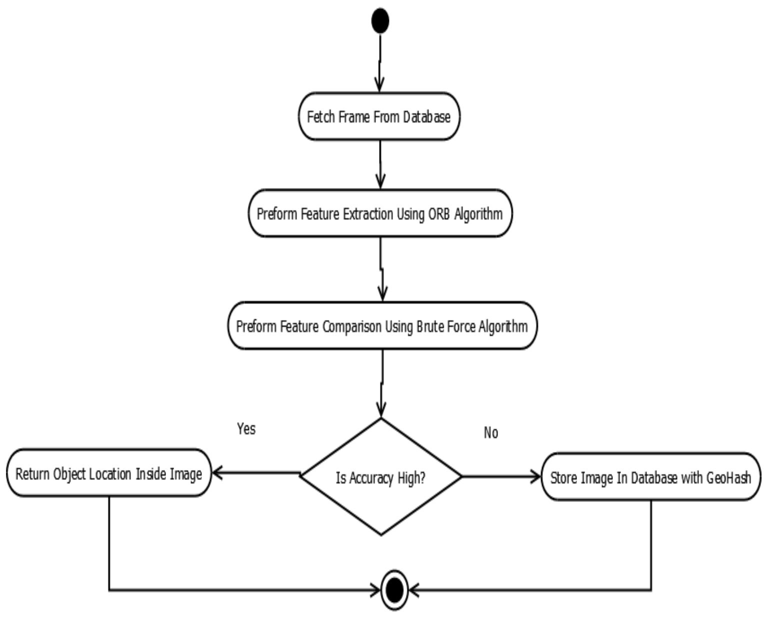

Fetch Frame From Database
Preform Feature Extraction Using ORB Algorithm
Preform Feature Comparison Using Brute Force Algorithm
Yes
No
Is Accuracy High?
Return Object Location Inside Image
Store Image In Database with GeoHash

Fig 11: Activity Diagram

In this diagram, the flow of the server function is being shown as first the server will retrieve the frame from the database. Then the server will run the ORB algorithm which extracts the main features of the frame, by detecting important keypoint using edge detection algorithm. These keypoints defines the frame and they are added into a 2D array. The server takes these features and compare the new extracted frame with every frame captures in the database using FLANN algorithm that uses a brute force approach. If the accuracy is high, then that means that the frame exists in the database and has an object attached with it, the server will then send that location of the object inside the frame to the database. If the accuracy was low, then that means that this frame is new, the server will then store the frame for future references along with the Geohash in the database.

3.4 Sequence Diagram

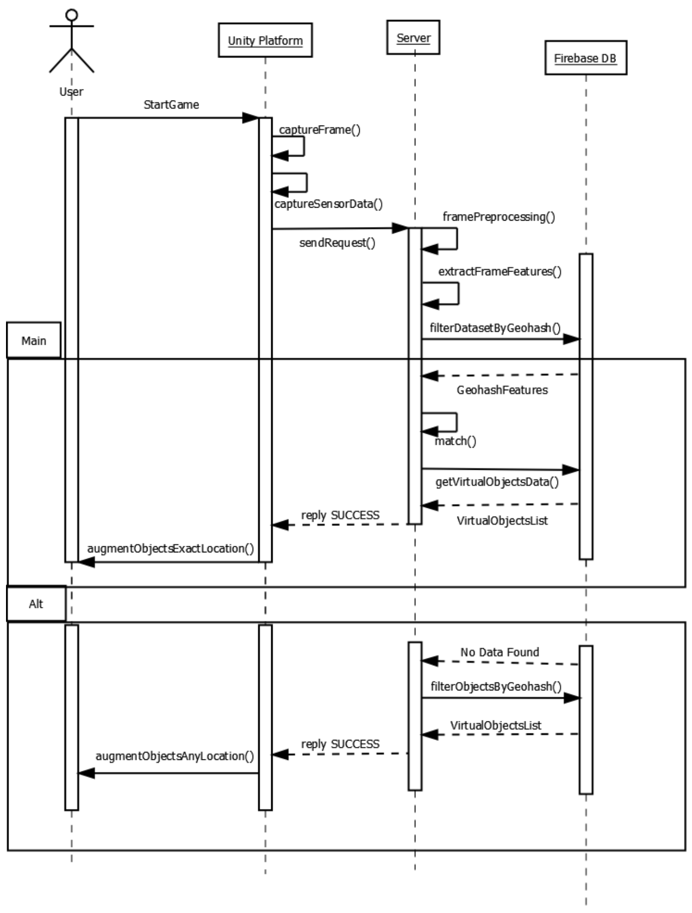


Fig 12: Sequence Diagram User's Side

In this diagram, it shows the flow of the user during gameplay. Unity sends the captured frame to the server to preform feature extraction, then request filtration using geohash and retrieve the current objects inside the given geohash. The server then runs FLANN algorithm for feature extraction on the incoming frames from the database, and reply to unity platform on the mobile device. Unity platform will then correspond the reply with current objects inside the mobile device and superimpose the object. If there wasn't a frame found inside the database then the server will superimpose the object using geohash.

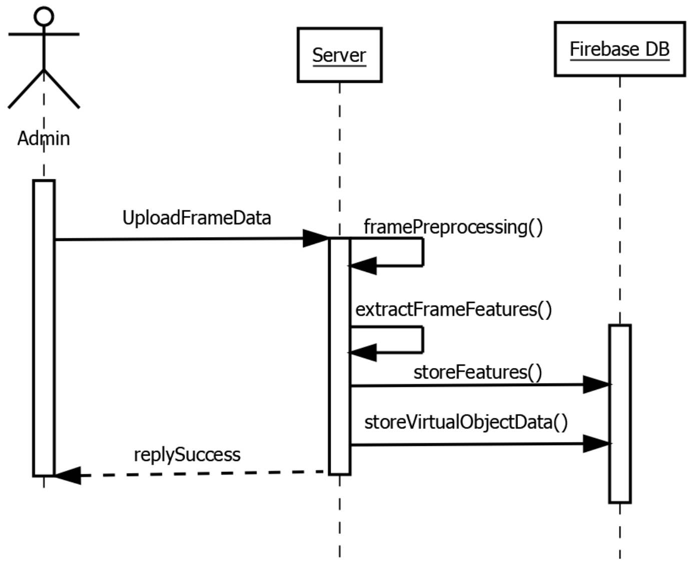


Fig 13: Sequence Diagram Admin's Side

In this diagram, it shows the flow of the admin when uploading an object. The admin upload the captured frame to the server, the server extract the features and store it in the database for future references. The server also stores the object that the admin wants to superimpose when the user enters the location. The server then reply to the admin for the success of the operation.

3.5 Design Rationale

The Wanderer is based in MVC design pattern, where the model (server) handles the backend processing like storage management and image processing, the controller takes the information from the model and deliver it to the view for the user interface, the view takes the event from the user using the event listeners and deliver it to the controller, then the controller gives it to the model for further processing. Using the MVC design pattern ensure the re-usability of the Wanderer for other projects, and the easy maintenance without going deep inside the core.

The model will use the ORB algorithm, which takes the frame and extract features from it making sure that no other image will be the same as the last, if there were a different object inside the image. The FLANN algorithm takes the processed image and start matching the features from other frames to know if there is a similar frame in the database.

For further processing, the server should have some kind of intelligence to contain the incoming dataset. In this scenario, we are using CNN deep learning. The server takes a frame resize and transpose to match current dataset, then the server pushes the processed data inside an array to adjure into a classifier. The incoming frame will be compared with the classifiers in the database, if the frame passes the threshold then the object exists and the server will send the response to the database in order for the mobile to superimpose the virtual object.

4 Data Design

4.1 Data Description

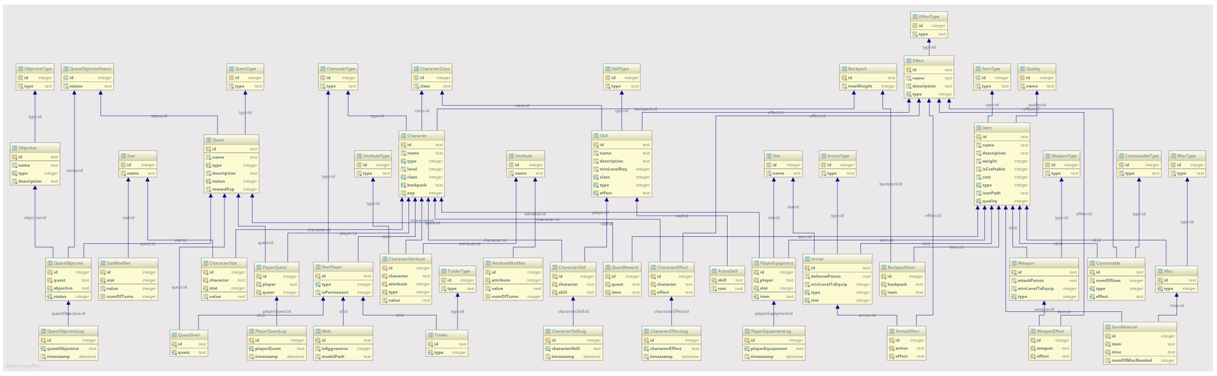


Fig 14: The Wanderer's Database

4.2 Data Dictionary

- Item: This table describes all the items in our game, where each item has it's own information. Its relations are Misc, Consumables, Weapons, Armor, BackpackItem, PlayerEquipment, QuestReward, ItemType, Quality, and ItemMaterials. Its attributes are name, description, weight, isCraftable, cost, type, iconPath, quality.

- Weapon: This table describe the weapon used by the player to attack monsters. Its relations are WeaponEffect, and WeaponType. Its attributes are attackPoints, minLevelToEquip, and type.

- WeaponEffect: This table describes the weapons that have effects on them. These effects will affect the monster in different ways. Its relations are Weapons, and Effects. Its attributes are weapon and effect.

- Consumable: This table describes the potions or poisons used in battle, which will put an effect on the player or the monster. Its relations are ConsumableType and Effects. Its attributes are numOfUses, type, effect.

- Misc: This table describes other items that the player may sell or craft with them. Its relations are MiscType and ItemMaterials. Its attribute is type.
- ItemMaterials: This table describes the material that the items are made of. Its relations are Misc and Items. Its attributes are item, misc and numOfMiscNeeded.
- MiscType: This table describes the contains of the misc, like gold, ore, and fur that the user may craft or sell for gold. Its relation is Misc. Its attribute is type.
- ConsumableType: This table describes the type of the item that the user will use. Either a poison or potion, poison will decrease the monster's attributes, and potion will increase the player's attributes. Its relation is Consumable. Its attribute is type.
- WeaponType: This table describes the types if the Weapons, like swords or axes . Each weapon has a different attack point and stamina consumption.Its relation is Weapons. Its attribute is type.
- Quality: This table describes the quality of the item, the quality is defined by set of names like fine, superior, legendary, or epic they describes the attack point of a weapon. Its relation is Item. Its attribute is name.
- ItemType: This table describes the type of the item, if the item is a weapon, armor, consumable, or misc. Its realtion is Item. Its attribute is type.
- Armor: This table describes the armor that the player is wearing. An armor will help decrease the incoming attacks from the monster. Its relations are ArmorEffect, ArmorType, Slot, and Item. Its attributes are defensePoints, minLevelToEquip, type, and slot.
- ArmorEffect: This table describes the effects on an armor. This effects can increase the player's ratings. Its relations are Armor and Effect. Its attributes are armor and effect.
- ArmorType: This table describes the type of a worn armor, where it's wearable like a helmet or a shield. Its relation is type. Its attribute is type.
- BackpackItem: This table describes the items inside a Backpack that the player uses during their gameplay. Its relations are Backpack and Item. Its attributes are backpacka and item.
- BackPack: This table describes the items the player carries with them, like weapons, armors, consumables, reward items from quests. Its relations are BackPackItem and Character. Its attributes is maxWeight.

- Effect: This table puts special effect on weapons and armors that either will help the player gain special status or will put the enemy at a temporary disadvantage. Its relations are Character, Skill, ArmorEffect, WeaponEffect, and Consumable. Its attributes are name, description, and type.
- EffectType: This table describes the type of the added effects, like Buff, DeBuff, Damage, Restore. These effects works differently like buff adds for an extanded time an advantage for the player. Its relation is Effect. Its attribute is type.
- Slot: This table assists with the equipped items, if the player has a weapon which hand holds the weapon, or where is the armor worn. Its relations are Armor, and PlayerEquipment. Its attribute is name.
- PlayerEquipment: This table clarify the items that the player has equipped, like weapons and armor to update the player's ratings. Its relations are Slot, Item, Character and PlayerEquipmentLog. Its attributes are player, slot, and item.
- PlayerEquipmentLog: This table keeps record of what the player is wearing or worn, and also logs the effects applied on the player. Its relation is PlayerEquipment. Its attributes are playerEquipment, and timestamp.
- Skill: This table's purpose is to map the player's skills, which will help them with surviving the game. Skills can help the user unlock new attacks, craft new items, or increase carry weight in backpack. Its relations are SkillType, CharacterClass, Effect, CharacterEffect, and CharacterSkill. Its attributes are name, description, minLevelReq, class, type, and effect.
- SkillType: This table describes the type of the skill effect used by the player, if it's an active or passive. Active means that the player uses them in battle, Passive means it works in the background like increase max weight. Its relation is Skill. Its attribute is type.
- ActiveSkill: This table describes the active skills that the player has unlocked. Active means that the player uses them in battle as weapons, each skill has special affect on the monster. Its relation is Skill. Its attribute is cost.
- CharcterEffect: This table describes the effects that are added on the player. The long term effects like restore attributes. Its relations are Character, Effect, and CharacterEffectLog. Its attributes are character and effect.
- CharacterEffectLog: This table logs the added effects on the player. Its relation is CharacterEffect. Its attributes are characterEffect and timestamp.

- CharcterSkill: This table describes the skills that are added on the player. The long term skills like increase carry weight. Its relations are Skill, Character, and CharacterSkillLog. Its attributes are character and effect.
- CharacterSkillLog: This table logs the added skills on the player. Its relation is CharacterSkill. Its attributes are characterskill and timestamp.
- QuestReward: This table are special items that given when completing a quest, they often have rare effects or high ratings. Its relations are Item and Quest. Its attributes are quest and item.
- Attribute: This table describes certain ratings on the player. These ratings are Buffs and DeBuffs that either help the player or not. Its relations are AttributeModifer and CharacterAttributes. Its attribute is name.
- AttributeModifer: This table is a detailed version of Attribute. It maps the added attributes on the player. Its relation is Attribute. Its attributes are attribute, value, and numOfTurns.
- Character: This table's purpose is to handle any character in the games information from player to NCPs that interacts with the player. Its relations are PlayerEquipment, CharacterClass, Effect, CharacterTypr, CharacterEffect, CharcterSkill, CharacterAttribute, NonPlayer, PlayerQuest, CharacterStats, and Backpack. Its attributes are name, type, level, class, backpack, exp.
- CharacterClass: This table describes each character is which kind of class, like a mage, a warrior, a hunter. Its relations are Skill and Character. Its attribute is class.
- CharcterType: This table is a detailed type of a character like a trader, a monster, a player. Its relations are character and Nonplayer. Its attribute is type.
- AttributeType: This table defines the attribute thate is being used, either mana, health, or stamina. Its relations are Character and CharacterAttribute. Its attribute is type.
- CharacterAttribute: This table describes each attribute health, mana, and stamina for each character. Its relations are Its relations are Character and CharacterAttribute. Its attributes are type, attribute, character, and value.
- NonPlayer: This table stores all of the NPCs that are active inside the game. The Non player characters interacts with the player in many ways aggressively or not. Its relations are CharcterType, Character, Mob, Trader, QuestGiver. Its attributes are type and IsPermanent.

- Mob: This table describes the monsters inside the game. The monster's intent is to attack the player, and the player initiates combat with them. Its relation is NonPlayer. Its attributes are isAggressive and modelPath.
- Trader: This Table is the trader where the player goes to to buy or sell items. Each trader has their own inventory where the player inspects them and purchase items. Its relations are NonPlayer and TraderType. Its attribute is type.
- TraderType: This table defines the trader's inventory, either it's a blacksmith then purchase weapons and armors, a Treasure Hunter, then purchase maps for special items. Its relation is Trader. Its attribute is type.
- QuestGiver: This table is the NPC that gives teh quest to the player, the player is to accept or refuse the quest from the questgiver. Its relations are NonPlayer and Quest. Its attribute is quest.
- PlayerQuest: This table is the relation of the quests that the player took or in process of. Its relations are Character and Quest. Its attributes are player and quest.
- Quest: This table is the mission that the player takes from the questgiver. The quest could be fetching an item, killing monsters or crafting certain items. Each quest then rewards the player with a special item. Its relations are QuestType, PlayerQuest, QuestReward, QuestGiver, and QuestObjective. Its attributes are name, type, description, status, and rewardExp.
- QuestType: This table defines each quest, either fetching items, killing monsters or crafting items. Its relation is Quest. Its attribute is type.
- CharacterStat: This table defines the stats that the player has unlocked when leveling up. Its relation is Stat and Character. Its attribute is character, stat, and value.
- Stat: This table describes the ratings that the player has. Ratings are some advantages that the player gets when leveling up. Advantages like adding attack points to weapons, decrease incoming attacks. Its relations are StatModifier and CharacterStat. Its attribute is name.
- statmodifier: This table defines the stats that the player receives like recover attributes, or damage monsters or player. Its relation is Stat, its attributes are stat, value, and numOfTurns.
- Objective: This table describes the objectives which are sub-quests that the player completes. On the completion of an objective, another one comes until the quest is finished. Its relations are ObjectiveType and QuestObjective. Its attributes are name, type, and description.

- ObjectiveType: This table define the objective type, either fetching items, killing monsters or crafting items. Its relation is Objective. Its attribute is type.
- QuestObjectiveStatus: This table describes the status of an objective, either completed, failed or in-process. Its relations are QuestObjectives and Quests. Its attribute is status.
- QuestObjective: This table related between each quest and its objective. Its relations Objective and Quest. Its attributes are quest, objective, and status.
- QuestObjectiveLog: This table logs each quest and its objective for storage and for future references. Its relation is QuestObjective. Its attributes are questobjective and timestamp.

5 Component Design

The server uses the ORB algorithm [4] to extract the important features of the sent frame, this will ensure if there was another frame sent of the same object, the features would stay the same.

5.1 ORB Algorithm

The Oriented and Fast rotated Brief algorithm is a fast feature detector that uses keypoint detection. First it finds the keypoints and then computes the intensity weighted centroid of the patch with located corner at center. The direction of the vector from this corner point to centroid gives the orientation. These keypoints marks the important parts inside a frame, which is small in size and more easy to compute when wanting to compare frames. This will ensure that even if the orientation of a frame is changed, the important objects and corners are effected.

5.2 Dataset

The dataset that has been stored inside the database are the object on which the admin wants to superimpose the object. When the admin wants to add a virtual object on a real object, the admin will capture the object in more that one position, this will ensure the increased accuracy of the results the more frames the more accuracy. The admin will then upload the frames to the database for the server to process it and compare them with the new frame that the user will then sends.

5.3 FLANN Algorithm

For the frame comparison, the server uses Brute Force algorithm called FLANN that compares the features of the frames together by comparing the keypoints that the ORB algorithm has extracted. The result of the brute force algorithm gives a percentage if the frames are closer to each others, this percent is compared with a threshold which then determines if the frames are identical or not. If the frames are identical, then the server sends the object location on the frame that will be superimposed on the mobile device, if there were a mismatch, then the server stores the new frame along side with the geohash. That will ensure if the same frame is sent gain, then the object will then appear.

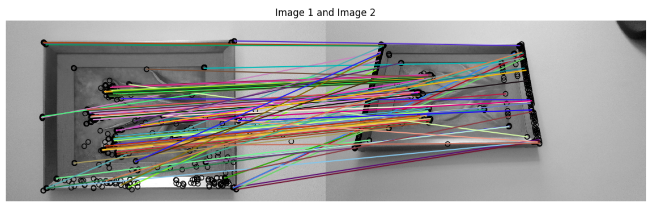


Fig 15: FLANN algorithm Matching Keypoints

In Figure 10, ORB algorithm has extracted keypoints inside each image, these keypoints are important features that defines the image. FLANN algorithm takes each keypoint and starts to apply a brute force algorithm to compare each keypoint, and returns a list of common features. According to these common features, the algorithm decides whether the image is the same or not by adding a threshold number that filters the bad percentage from the good one.

5.4 CNN Using Keras Library

In order to invoke object detection, the server should have some kind of intelligence to contain the incoming dataset. In this scenario, we are using CNN deep learning using keras library. The server takes a frame resize and transpose to match current dataset, then the server pushes the processed data inside an array to adjure into a classifier. The incoming frame will be compared with the classifiers in the database, if the frame passes the threshold then the object exists and the server will send the response to the database in order for the mobile to superimpose the virtual object.

The merge of the core with the game has been implemented. The core is used to apply virtual objects by using geolocation for general filtration, which means that the geolocation helps pinpoint certain objects inside the geohash that the user is standing inside. The object recognition is used to superimpose the virtual object on the real object, these objects can be several things like quests NPCs, Monsters or items. The user would go to a NPC that will give them a quest, accordingly to the quest, the user will go to the given location to fetch an item by fighting a monster, upon defeat the user will acquire the item and complete the quest.

6 Human Interface Design

6.1 Overview of User Interface

The Wanderer works on two different ends, the admin end and the user end.

- The admin ends focus around adding a virtual object to the user. The admin will go to the place where the object will be placed, then capture a few frames, the more frames the more accurate the reading will be. The admin will then go to the upload place, then the user will upload the captured frame along with the geohash using google maps, the name of the object, and the purpose of the object, like a monster or a questgiver. These information will be uploaded to Firebase database for storage, and for the server to process them.
- The user side will contain several menus they are as follows:
  - Shop menu: The shop menu will have the trader's inventory. Each trader has an inventory containing items for the user to use. These items are weapons, armor, potions, or poisons to use inside the combat.
  - Combat menu: The combat menu contains the combat mechanism. This will interacting with the monster the user faced, either by attacking using weapons or magic, defending incoming attacks, using a potion or poison, or fleeing from the monster.
  - Quest menu: This menu contains all the user's quest; either current quests, completed quests or active quests. Also having the information of each quest as well as teh objectives inside each quest.
  - Map menu: This is a map that contains nearby shops or quests for the user to go to these locations to interact with them.

6.2 Screen Images

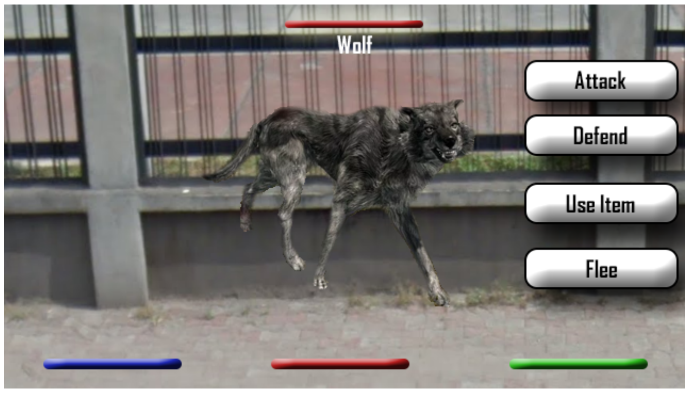


Fig 16: The Combat Menu

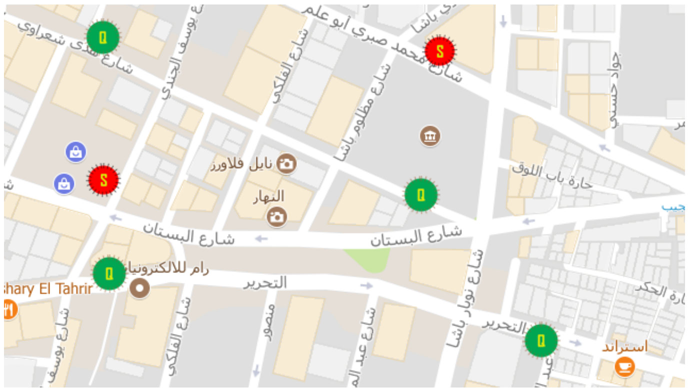
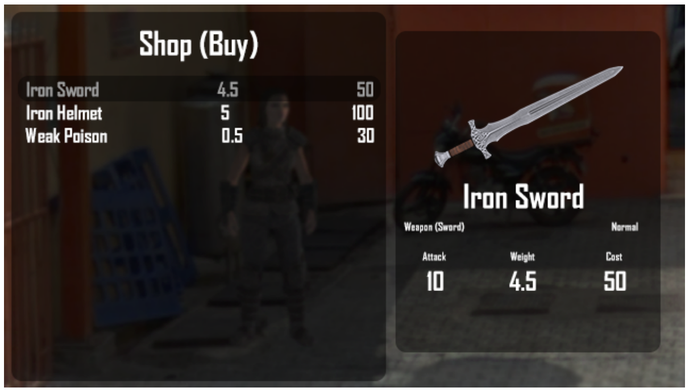


Fig 17: The Maps Menu


Shop (Buy)

Iron Sword	4.5	50
Iron Helmet	5	100
Weak Poison	0.5	30

Iron Sword

Weapon (Sword) Normal

Attack	Weight	Cost
10	4.5	50

Fig 18: The Shop Menu

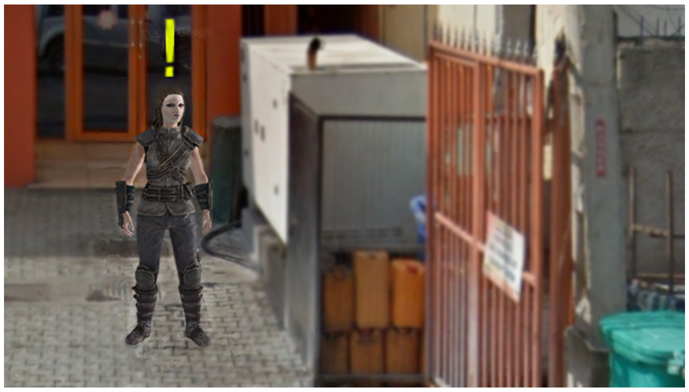


Fig 19: The Quest Giver For The User To Interact With It

6.3 Screen Objects and Actions

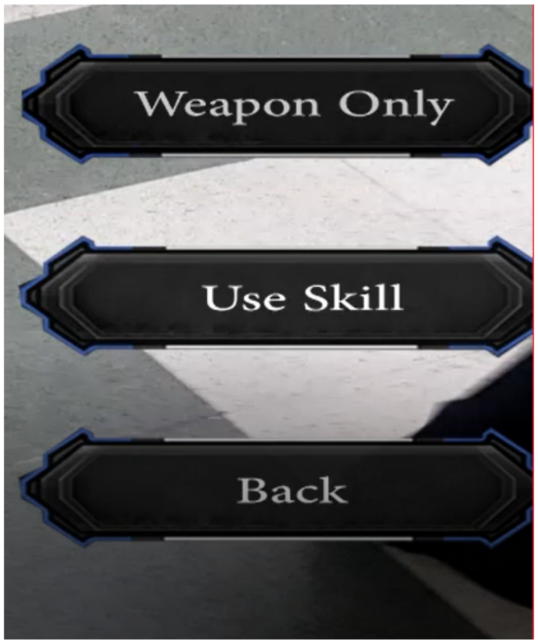


Fig 20: The Attack Menu

Fig 21: The Weapon Button

Initiating attack on a monster will enter into a selection, either choosing to attack with a weapon which will consume from the stamina bar, and according to the equipped weapon the attack point will decrease the monster's health.

Fig 22: The Sills Button

The player can choose skills, if the player have skills, it will consume from the mana bar. The skills are added dynamically according to the unlocked skills

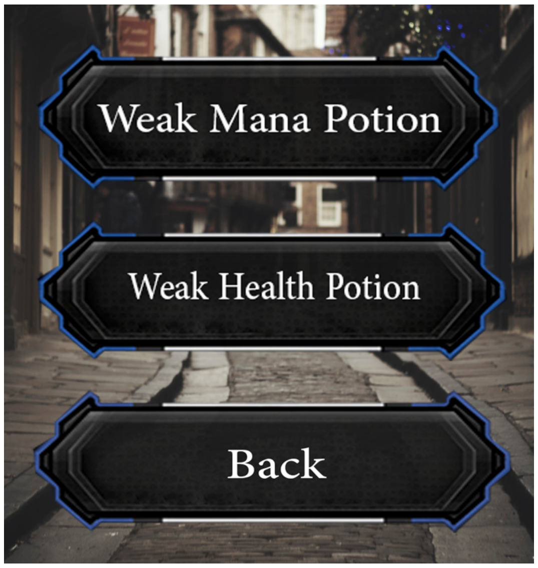
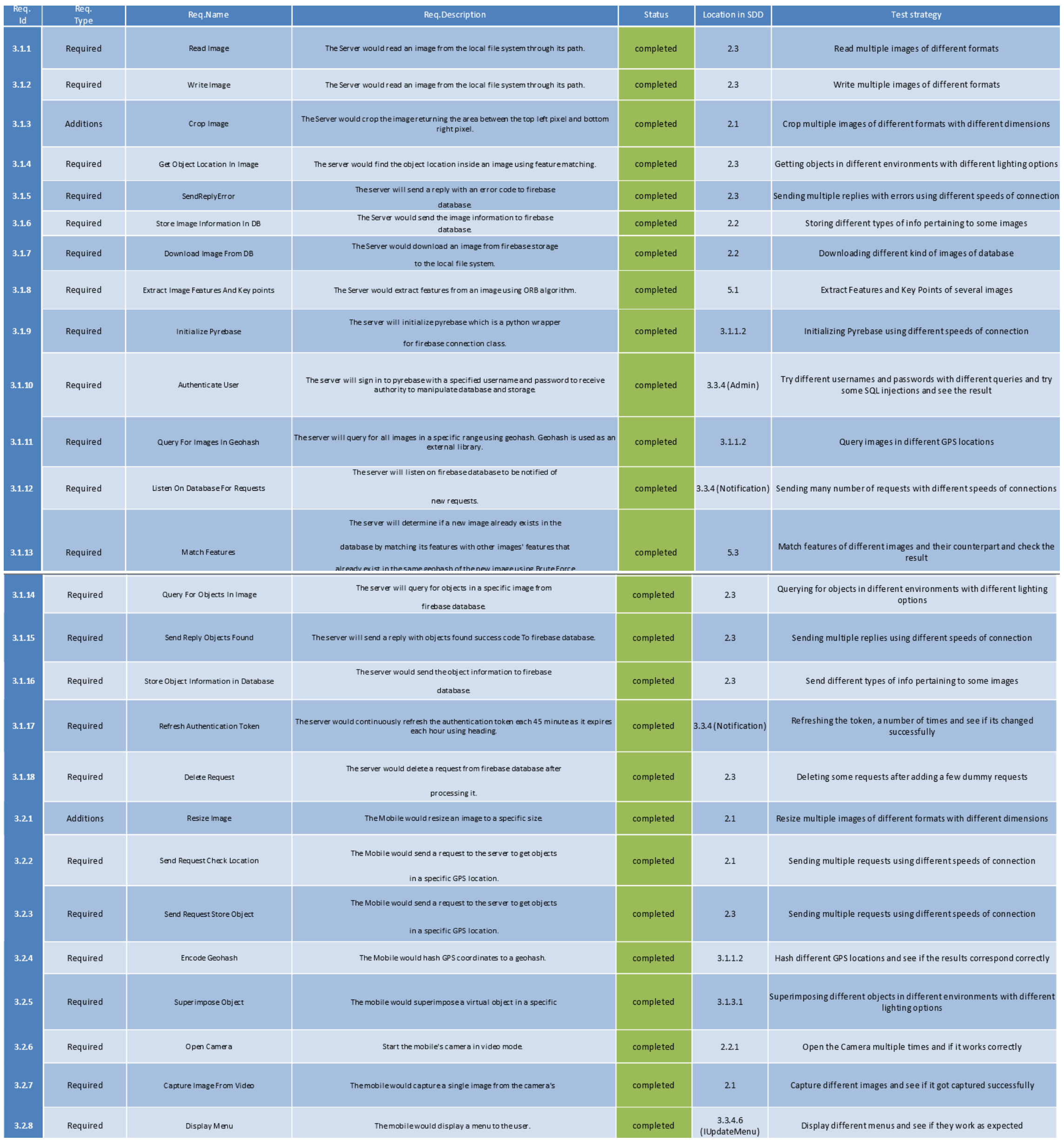


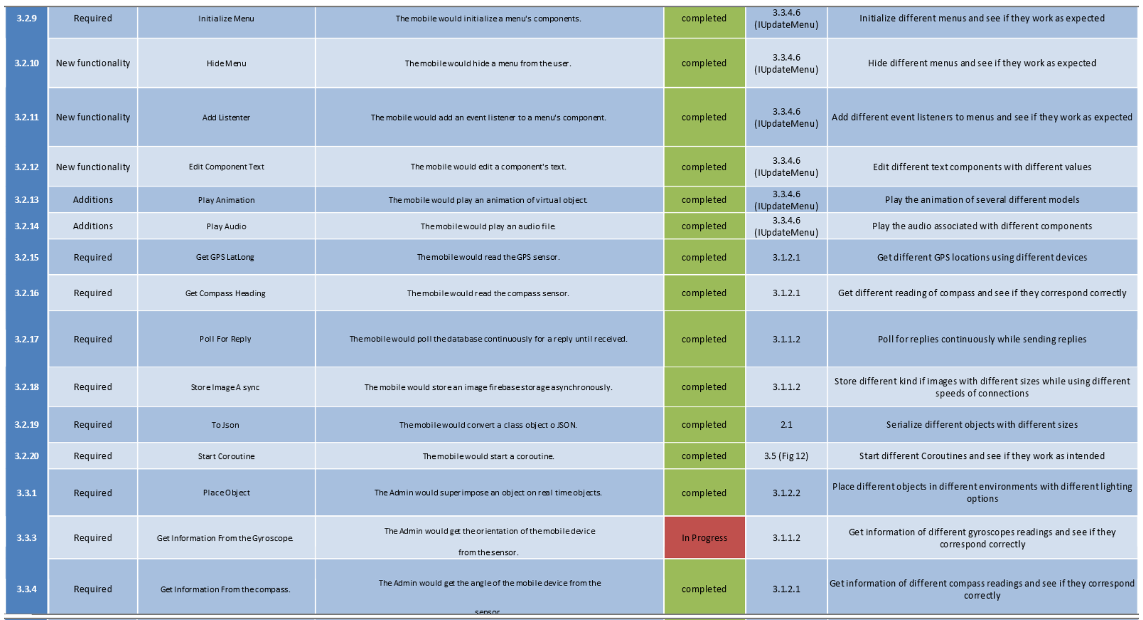
Fig 23: The Potion Menu

The potion menu will be used for the user to increase his attributes (health, mana, and stamina). If the player has consumables then it will appear dynamically from the player's inventory. The potion after selection will cost a turn, meaning the monster will attack you. There are other potions like increase defense rating, attack points, critical points, ...etc.

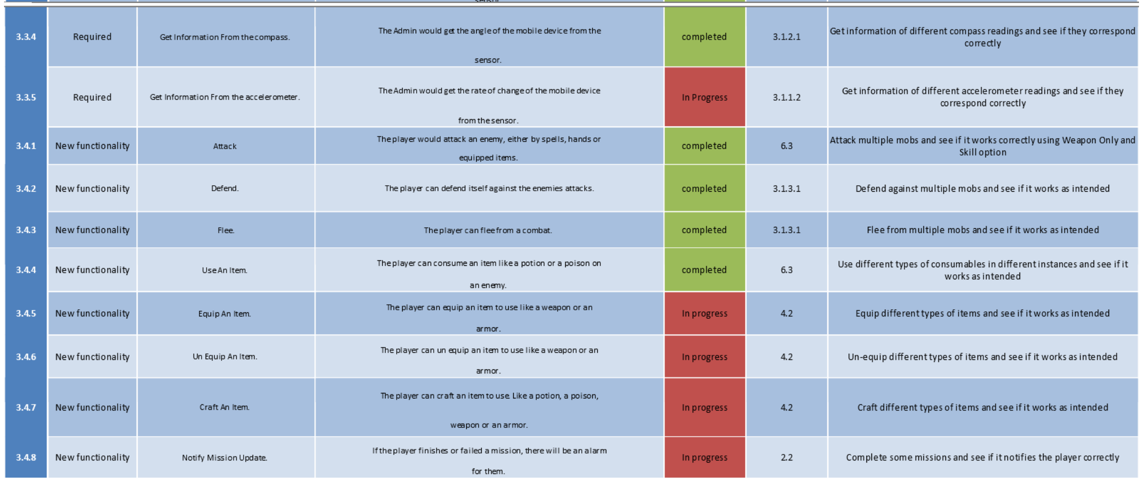
7 Requirements Matrix



Req. Id	Req. Type	Req.Name	Req.Description	Status	Location in SDD	Test strategy
3.1.1	Required	Read Image	The Server would read an image from the local file system through its path.	completed	2.3	Read multiple images of different formats
3.1.2	Required	Write Image	The Server would read an image from the local file system through its path.	completed	2.3	Write multiple images of different formats
3.1.3	Additions	Crop Image	The Server would crop the image returning the area between the top left pixel and bottom right pixel.	completed	2.1	Crop multiple images of different formats with different dimensions
3.1.4	Required	Get Object Location In Image	The server would find the object location inside an image using feature matching.	completed	2.3	Getting objects in different environments with different lighting options
3.1.5	Required	SendReplyError	The server will send a reply with an error code to firebase database.	completed	2.3	Sending multiple replies with errors using different speeds of connection
3.1.6	Required	Store Image Information In DB	The Server would send the image information to firebase database.	completed	2.2	Storing different types of info pertaining to some images
3.1.7	Required	Download Image From DB	The Server would download an image from firebase storage to the local file system.	completed	2.2	Downloading different kind of images of database
3.1.8	Required	Extract Image Features And Key points	The Server would extract features from an image using ORB algorithm.	completed	5.1	Extract Features and Key Points of several images
3.1.9	Required	Initialize Pyrebase	The server will initialize pyrebase which is a python wrapper for firebase connection class.	completed	3.1.1.2	Initializing Pyrebase using different speeds of connection
3.1.10	Required	Authenticate User	The server will sign in to pyrebase with a specified username and password to receive authority to manipulate database and storage.	completed	3.3.4 (Admin)	Try different usernames and passwords with different queries and try some SQL injections and see the result
3.1.11	Required	Query For Images In Geohash	The server will query for all images in a specific range using geohash. Geohash is used as an external library.	completed	3.1.1.2	Query images in different GPS locations
3.1.12	Required	Listen On Database For Requests	The server will listen on firebase database to be notified of new requests.	completed	3.3.4 (Notification)	Sending many number of requests with different speeds of connections
3.1.13	Required	Match Features	The server will determine if a new image already exists in the database by matching its features with other images' features that already exist in the same geohash of the new image using Brute Force	completed	5.3	Match features of different images and their counterpart and check the result
3.1.14	Required	Query For Objects In Image	The server will query for objects in a specific image from firebase database.	completed	2.3	Querying for objects in different environments with different lighting options
3.1.15	Required	Send Reply Objects Found	The server will send a reply with objects found success code To firebase database.	completed	2.3	Sending multiple replies using different speeds of connection
3.1.16	Required	Store Object Information in Database	The server would send the object information to firebase database.	completed	2.3	Send different types of info pertaining to some images
3.1.17	Required	Refresh Authentication Token	The server would continuously refresh the authentication token each 45 minute as it expires each hour using heading.	completed	3.3.4 (Notification)	Refreshing the token, a number of times and see if its changed successfully
3.1.18	Required	Delete Request	The server would delete a request from firebase database after processing it.	completed	2.3	Deleting some requests after adding a few dummy requests
3.2.1	Additions	Resize Image	The Mobile would resize an image to a specific size.	completed	2.1	Resize multiple images of different formats with different dimensions
3.2.2	Required	Send Request Check Location	The Mobile would send a request to the server to get objects in a specific GPS location.	completed	2.1	Sending multiple requests using different speeds of connection
3.2.3	Required	Send Request Store Object	The Mobile would send a request to the server to get objects in a specific GPS location.	completed	2.3	Sending multiple requests using different speeds of connection
3.2.4	Required	Encode Geohash	The Mobile would hash GPS coordinates to a geohash.	completed	3.1.1.2	Hash different GPS locations and see if the results correspond correctly
3.2.5	Required	Superimpose Object	The mobile would superimpose a virtual object in a specific	completed	3.1.3.1	Superimposing different objects in different environments with different lighting options
3.2.6	Required	Open Camera	Start the mobile's camera in video mode.	completed	2.2.1	Open the Camera multiple times and if it works correctly
3.2.7	Required	Capture Image From Video	The mobile would capture a single image from the camera's	completed	2.1	Capture different images and see if it got captured successfully
3.2.8	Required	Display Menu	The mobile would display a menu to the user.	completed	3.3.4.6 (IUpdateMenu)	Display different menus and see if they work as expected



3.2.9	Required	Initialize Menu	The mobile would initialize a menu's components.	completed	3.3.4.6 (IUpdateMenu)	Initialize different menus and see if they work as expected
3.2.10	New functionality	Hide Menu	The mobile would hide a menu from the user.	completed	3.3.4.6 (IUpdateMenu)	Hide different menus and see if they work as expected
3.2.11	New functionality	Add Listenter	The mobile would add an event listener to a menu's component.	completed	3.3.4.6 (IUpdateMenu)	Add different event listeners to menus and see if they work as expected
3.2.12	New functionality	Edit Component Text	The mobile would edit a component's text.	completed	3.3.4.6 (IUpdateMenu)	Edit different text components with different values
3.2.13	Additions	Play Animation	The mobile would play an animation of virtual object.	completed	3.3.4.6 (IUpdateMenu)	Play the animation of several different models
3.2.14	Additions	Play Audio	The mobile would play an audio file.	completed	3.3.4.6 (IUpdateMenu)	Play the audio associated with different components
3.2.15	Required	Get GPS LatLong	The mobile would read the GPS sensor.	completed	3.1.2.1	Get different GPS locations using different devices
3.2.16	Required	Get Compass Heading	The mobile would read the compass sensor.	completed	3.1.2.1	Get different reading of compass and see if they correspond correctly
3.2.17	Required	Poll For Reply	The mobile would poll the database continuously for a reply until received.	completed	3.1.1.2	Poll for replies continuously while sending replies
3.2.18	Required	Store Image A sync	The mobile would store an image firebase storage asynchronously.	completed	3.1.1.2	Store different kind if images with different sizes while using different speeds of connections
3.2.19	Required	To Json	The mobile would convert a class object o JSON.	completed	2.1	Serialize different objects with different sizes
3.2.20	Required	Start Coroutine	The mobile would start a coroutine.	completed	3.5 (Fig 12)	Start different Coroutines and see if they work as intended
3.3.1	Required	Place Object	The Admin would superimpose an object on real time objects.	completed	3.1.2.2	Place different objects in different environments with different lighting options
3.3.3	Required	Get Information From the Gyroscope.	The Admin would get the orientation of the mobile device from the sensor.	In Progress	3.1.1.2	Get information of different gyroscopes readings and see if they correspond correctly
3.3.4	Required	Get Information From the compass.	The Admin would get the angle of the mobile device from the sensor.	completed	3.1.2.1	Get information of different compass readings and see if they correspond correctly



3.3.4	Required	Get Information From the compass.	The Admin would get the angle of the mobile device from the sensor.	completed	3.1.2.1	Get information of different compass readings and see if they correspond correctly
3.3.5	Required	Get Information From the accelerometer.	The Admin would get the rate of change of the mobile device from the sensor.	In Progress	3.1.1.2	Get information of different accelerometer readings and see if they correspond correctly
3.4.1	New functionality	Attack	The player would attack an enemy, either by spells, hands or equipped items.	completed	6.3	Attack multiple mobs and see if it works correctly using Weapon Only and Skill option
3.4.2	New functionality	Defend.	The player can defend itself against the enemies attacks.	completed	3.1.3.1	Defend against multiple mobs and see if it works as intended
3.4.3	New functionality	Flee.	The player can flee from a combat.	completed	3.1.3.1	Flee from multiple mobs and see if it works as intended
3.4.4	New functionality	Use An Item.	The player can consume an item like a potion or a poison on an enemy.	completed	6.3	Use different types of consumables in different instances and see if it works as intended
3.4.5	New functionality	Equip An Item.	The player can equip an item to use like a weapon or an armor.	In progress	4.2	Equip different types of items and see if it works as intended
3.4.6	New functionality	Un Equip An Item.	The player can un equip an item to use like a weapon or an armor.	In progress	4.2	Un-equip different types of items and see if it works as intended
3.4.7	New functionality	Craft An Item.	The player can craft an item to use. Like a potion, a poison, weapon or an armor.	In progress	4.2	Craft different types of items and see if it works as intended
3.4.8	New functionality	Notify Mission Update.	If the player finishes or failed a mission, there will be an alarm for them.	In progress	2.2	Complete some missions and see if it notifies the player correctly

Fig 24: Requirements Matrix

8 References

[1] Tateno, Masaru, et al. New game software (Pokmon Go) may help youth with severe social withdrawal, hikikomori. Psychiatry research 246 (2016): 848849.

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580587.

[3] ThomasSandholmandHangUng, Real-time,locationaware collaborative lter-ing of web content, in Proceedings of the 2011 Workshop on Context-awareness in Retrieval and Recommendation. ACM, 2011, pp.14 18.

[4] Frazer K Noble, Comparison of opencvs feature detectors and feature matchers, in Mechatronics and Machine Vision in Practice (M2VIP), 2016 23rd International Conference on. IEEE, 2016, pp. 16.