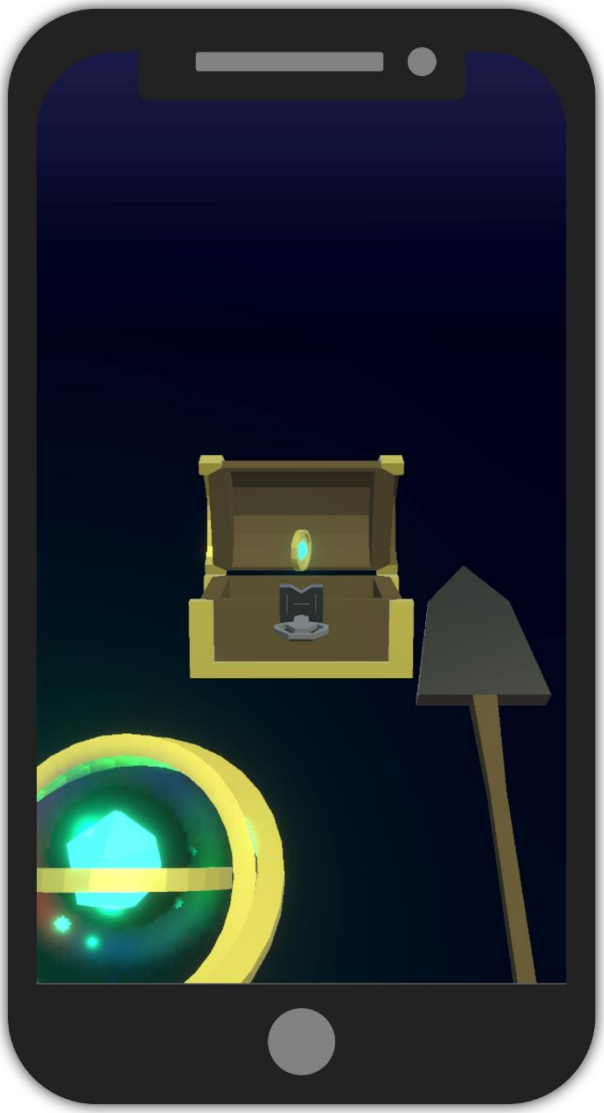


Treasure Hunter App

DESIGN DOCUMENT

Author:
Hendrik Lichtenberg

Abstract

This document outlines the development of a GPS based mobile game, potentially running on all mainstream mobile operating systems (Android, iOS) and powered by the Unity engine.

The design document is intended as working sample, for developers working on the project and for generally interested people who want to develop their own GPS based games.

After reading this document you should have a rough understanding of the inner workings of the app, the software architecture and the underlying concepts used to develop this project.

The first chapter deals with the projects context i.e. why the project is necessary, what purpose it serves etc.

The second chapter is about the goals and non-goals of the project or in other words: what the project is made for and what it isn't made for. Furthermore this chapter contains a list of user stories which should give the reader a brief overview of the apps features.

The third chapter deals with the apps gameplay and gives an overview of the game design.

In the fourth chapter you'll find the projects milestones, to give you an overview about the development process.

The fifth chapter is all about solutions and further divided in three subsections, which are:

- Current solution
- Proposed future solution

The sixth chapter of this document proposes testing methods to guarantee all features are running as intended.

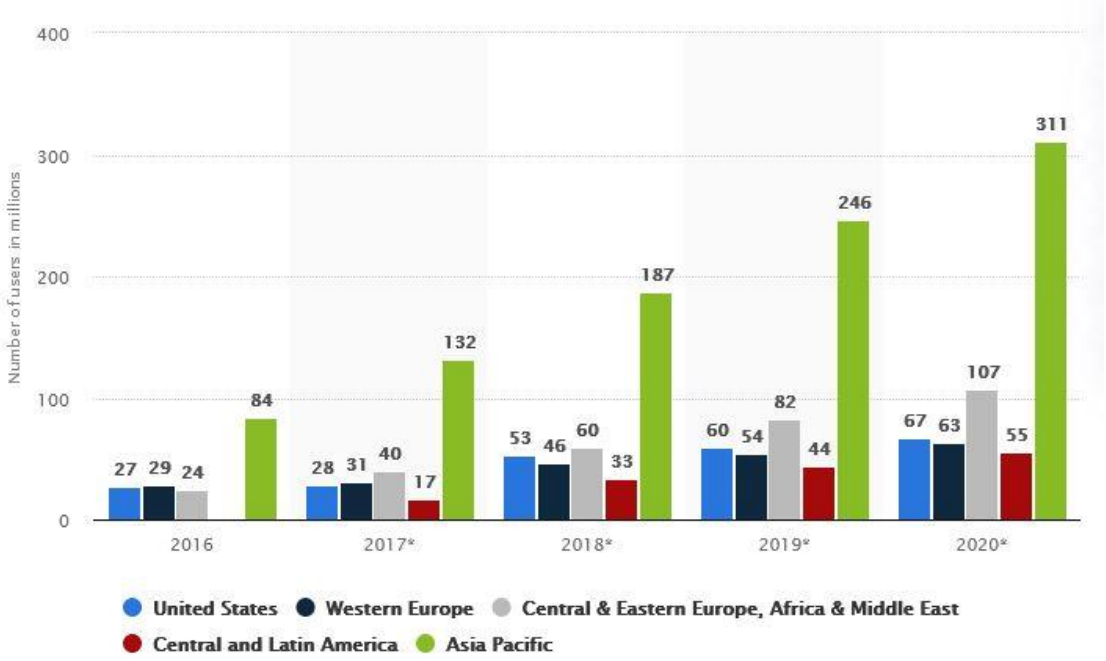
In the seventh chapter you can read about the projects underlying architecture and how the smaller parts come together to a whole. This chapter also contains a short explanation to each class and explain what are they used for and what's their purpose.

Context

HISTORICAL BACKGROUND

In 2016 the game development studio Niantic released an GPS based augmented reality game called „Pokémon Go“ which quickly grew to an international phenomenon, exceeding 45 million users only days after the initial release[1].

While user growth quickly stagnated, a positive trend can be observed in the years 2016-2020, with growing user numbers until today [2]



Number of active users of Pokémon Go worldwide [2]

According to the survey, mentioned above, the market for augmented reality apps is constantly growing and while Niantic undoubtly has a strong grasp of it, it is still mostly uncontested.

PURPOSE OF THIS PROJECT

While the main purpose of this project is still being a working sample, the fact that augmented reality gaming is still a young market can’t be ignored and developing an app that contests Niantics market position is a goal that is probably worth it.

Anyway, it’s tempting to only focus on the financial & marketing perspective but let’s be honest: making a GPS based app is both fun and cool and it’s a great opportunity to explore the technical skills needed to do this.

Long story short, the goal of the project is the development of a multi-platform app, which gives the user the opportunity to go on a mixed reality treasure hunt wherever he/she might be.

While searching for treasures the user will encounter different objects to interact with and will be ultimately rewarded with ingame-currency for completing a treasure hunt called „Quest“.

Furthermore the project is meant to be a proof of concept for the author and basic groundwork for developing more complicated GPS apps.

Goals and non-goals

GOALS

First and foremost this project is a working sample to illustrate the authors skills with C#, software-development, architecture, documentation and familiarity with the Unity framework.

The second goal is to develop a working, GPS based game which runs at least on Android but can be ported easily to other operating systems and that’s where Unitys strengths come into play as it grants easy portability to a multitude of platforms.

While not the main goal, the app should entertain the user to a certain degree which means it should implement at least a basic game-cycle to keep the user engaged.

The app should be license-legal which means only libraries, tools etc. with licenses available to the author are to be used.

To conclude this section: A solid and efficient application is definitely the priority, while keeping the architecture small and simple. It is not supposed to have a lot of features but few, good working features which can be build upon in later iterations.

NON-GOALS

As mentioned before, the app is going to be a working sample, not a finished product and while graphics and design are nice to have, they’re definitely not a priority.

The app isn’t going to be a „Pokémon Go Killer“ because the effort required to build an app like that would require a whole team of developers, designers and at least some budget, which isn’t available right now.

USER STORIES

1 Geolocation

1.1
“As user I need a way to track my devices location to get my current LatLong Position.“

1.2
“As developer I need a way to calculate distance between two LatLongs to check if the user reached a target coordinate.“

1.3
“As developer I need a way to easily initialize and start the Geolocation tracking service to implement further features.“

1.4
“As developer I need a way to get a random LatLong coordinate inside a certain radius around a predefined LatLong to randomly place target coordinates around the users position.“

2 Quest

2.1
“As user I need a feature to start a new quest based on my current location when starting the app to play the game.“

2.2
“As developer I need a way to randomly generate quest targets to generate a full quest.“

2.3
“As developer I need some functions to randomly distribute loot, between quest targets to generate a full quest.“

2.4
“As developer I need functions to keep track of the devices distance to each quest target to determine whether the user reached a target.“

2.5
“As developer I need a function to get the nearest target to know which target the user is approaching.“

2.6
“As user I need a way to end a quest to finish the game cycle.“

3 QuestTarget

3.1
“As a developer I need a function to initialize a target with individual locations to place them randomly.”

3.2
“As a user I need quest targets to react when I come near them or leave them, to interact only with a single target at once.”

3.3
“As a designer I want functions to hide and show objects (piles, rewards) related to a quest target to make them pop up once a user gets near them and hide objects once the user leaves the area.”

4 QuestItem

4.1
“As a developer I need a base class for interactable items to use them in multiple contexts.”

5 Astrolabium

5.1
“As a user I need a way to determine the distance to the next quest target to find it.”

5.2
“As a designer I need functions to control animation speed and effects of the Astrolabium to give the user hints where the next target is located.”

6 Pile

6.1
“As a developer I need a dirt pile where I can hide keys and treasure chests.”

7 Shovel

7.1
“As a user I need a shovel to dig up dirt piles.”

8 Key

8.1
“As a user I need a key to open chests.”

9 Chest

9.1
“As a user I need a treasure chest to find and open with a key.”

10 Coin

10.1
“As a user I need a coin which is spawned by a treasure chest to collect as a quest reward.”

11 Player

11.1
“As a developer I need a class that holds informations on the current status of the player.”

12 Fog

12.1
“As a designer I need a way to control fog displayed on the screen like clearing and closing the fog to hide sudden spawning of items.”

13 Math

13.1
“As a developer I need a function to check if a certain value is between two other values to check distances to targets with a certain tolerance.”

14 DebugUI

“As a tester I need a UI which displays debug messages to test the app outside in public where a desktop computer isn’t available.”

Gameplay

GAMPLAY OBJECTS

Fig 1. Gameplay objects

GAME LOOP

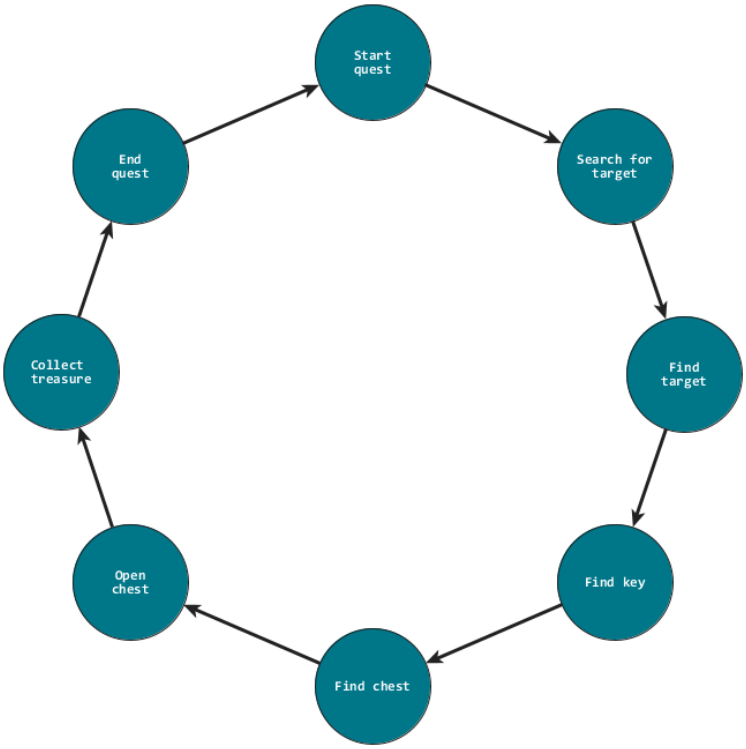


Fig 2. Game loop

The apps game loop consists of eight steps which are explained below.

Start quest

When the app is started the user receives a “quest” which is completed in the next steps of the loop. Key and chest are hidden at random GPS targets near the user.

Search for target

The user has to walk around and observe the Astrolabium (see Fig. 1) until it starts spinning which indicates, there’s a target in close range.

Find target

Once the Astrolabium started spinning, the user has to decrease the distance between him/her while the Astrolabium spins faster the nearer the user gets to the target.

Find key

The user has to find targets until he/she found a target where a key is hidden. The user picks up the key and proceeds.

Find chest

One target contains a chest, which can be opened given the user has already found a key.

Open chest

If the user has found a key, the chest can be opened and spawns a reward i.e. a coin.

Collect treasure

The chest spawns a coin that can be collected by the user which ends the quest.

End quest

Once the reward has been collected, the quest ends and a new quest starts (Game loop repeats).

Project milestones

ROADMAP

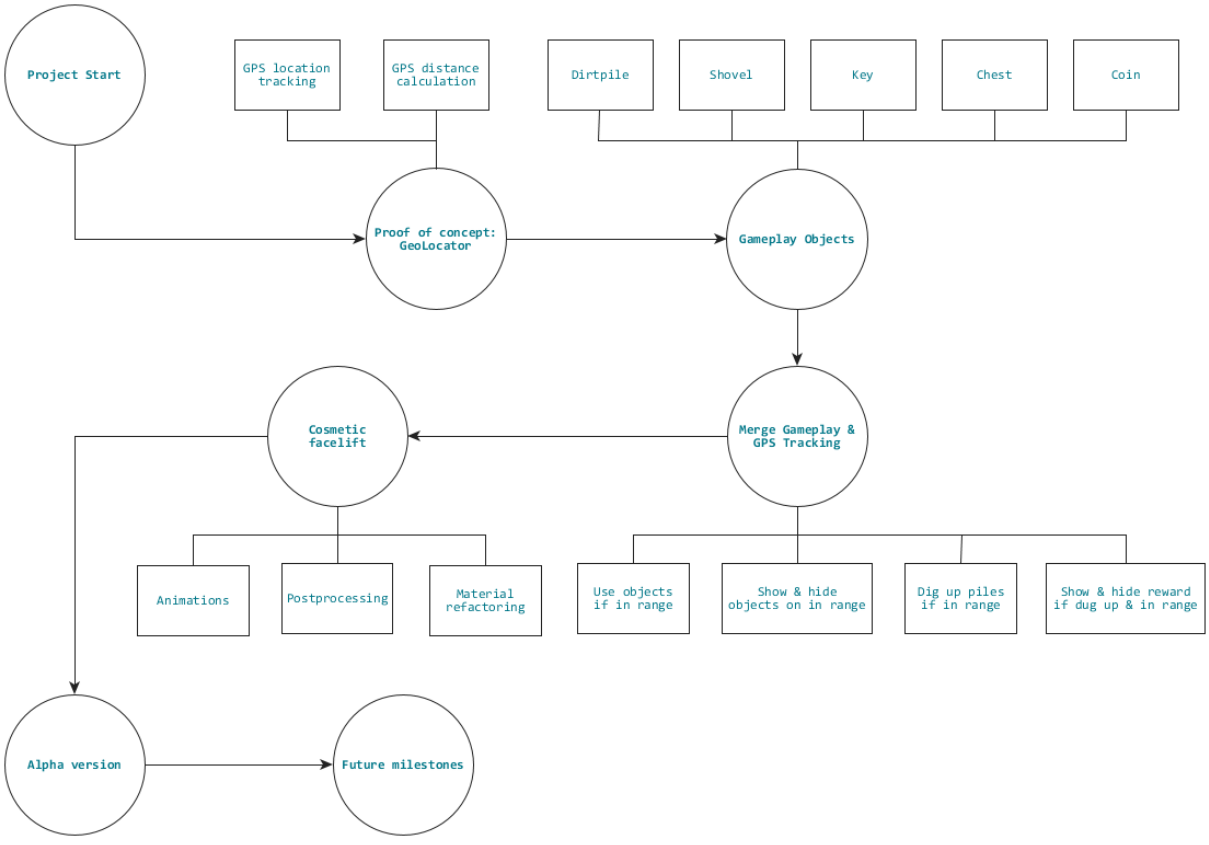


Fig 3. Project Roadmap

DETAILED MILESTONE EXPLANATION

Milestone 1 – Proof of concept: GeoLocator

The first milestone of the project lays the basic groundwork for the GPS tracking feature of the app as any further development is based on it. This milestone exposes two key features that need to be implemented before development can be continued:

- GPS location tracking
  A GPS based app obviously needs a way to continuously track the users location. Luckily Unity provides some functions to interface with the devices GPS module.[3]
- GPS distance calculation
  As the app relies on the users distance to certain coordinates, there needs to be a way to calculate the distance between two GPS coordinates.
  The biggest challenge to implement this feature is that a distance is interpreted in a planar coordinate system with linear metrics, while latitudes and longitudes are interpreted in spherical space.
  Luckily, the "Law of Haversines" [4] gives us an easy way to calculate a linear distance between two LatLongs.

The ultimate goal of this milestone is the development of a modular GeoLocator class which exposes the relevant functions to achieve the goals mentioned before. The GeoLocator is to be implemented as modular as possible, which means it can be reused in future projects in a "Copy & Paste" manner.

Milestone 2 – Gameplay Objects

The second milestone focuses completely on gameplay and features the development of all objects, needed to implement the "rules" of the game without taking location tracking into concern at this point. This ensures the correct implementation of every game rule while the inherit margin error of GPS isn't a concern at the moment.

This milestone is completed once the following tasks are done:

- Implementation of base classes for every gameplay object.
- Concrete implementation of object classes.
- Design and animation of the relevant 3D models for every gameplay object.
- Implementation of game rules or "how an object interacts with other objects"

For more information on the concrete features see chapter "User Stories 4-11".

Milestone 3 – Merge gameplay & GPS tracking

The third milestone deals with “merging” the product of the first and second milestone which means adding some additional game rules i.e. adding on quest targets and interaction of gameplay objects with them.

After this milestone is completed the app should be playable and all key features should be working as intended (see chapter “User Stories” for a brief overview of the intended features).

Milestone 4 – Cosmetic facelift

In Germany we got the saying “Das Auge isst mit” which roughly translates to “the eye eats too”, the same goes for applications. For that purpose, the app gets a cosmetic facelift.

While a beautiful design is a minor goal of the project, there’s no reason to not use the post-processing effects of Unitys universal render pipeline (URP), which gives us the opportunity to easily implement nice-looking effects like bloom and HDR-lighting as the URP is optimized for mobile devices and should make the app look way better for a minor performance tradeoff.

Milestone 5 – Alpha version

Once milestones 1-4 are completed the project is ready for being compiled into a first, working Alpha product, which could potentially be deployed to an app-store.

While deployment isn’t a goal right now because there’s still room for improvement, the app is supposed to be working as intended, with a complete game-loop, all user stories implemented and free of errors.

Once this happened, the app can be tested in a real-life environment an published to Github so others can try it and elaborate on how it can be improved, which features could be added in the future and voice their general opinion on the product.

Future Milestones

As mentioned before, once all milestones are completed, the app isn’t ready for deployment and of course an Alpha version isn’t a releasable product. That’s why this milestone is a placeholder for every feature imaginable in future iterations of the product. See the chapter “Current & future solution” for some examples of what could be added in the future.

Current & future solution

CURRENT SOLUTION

In its current state the app implements all features mentioned in the chapter “User Stories” and is in its Alpha state as mentioned in chapter “Milestones, Milestone 5 – Alpha Version”.

The app is playable, features a simple game-loop and GPS tracking works with an acceptable margin error, which is inherit to all GPS-based applications.

The project fulfills all requirement as laid out before and in document “Specification Book” and should fulfill its role as a working sample.

PROPOSED FUTURE SOLUTION

As written in previous chapters, the app in its current state isn’t releasable because it isn’t able to compete with comparable apps and needs a lot of further refinement to achieve a noticeable market-share.

There’s also room for improvement when looking into GPS-tracking accuracy as in its current state the app achieves accuracy of ~1-10 meters which is good enough but can still be improved.

The following sub-chapters deal with options on how to improve the product.

Improvement of gameplay

As mentioned before the app isn’t ready for competing with available GPS-based games, available on various app-stores because it still lacks features which enable a positive user experience. To achieve this, a couple of features could be added as listed below:

- Expansion of the game-loop:
  In its current state the app doesn’t give the user any motivation to play except for walking around outside and finding coins.
  For this reason the coins need a purpose which could be serving as in-game currency to buy upgrades for the tools available to the player i.e. a faster digging shovel, an upgraded Astrolabium which gives better hints to find targets like displaying the rough direction of the target next to the user.
- Action based gameplay elements:
  In addition to just searching treasure, the player could encounter and fight monsters which guard additional treasure.

Improvement of tracking accuracy

GPS tracking on mobile devices is less than perfect. For this reason outlined below is a proposal for an improved tracking technique.

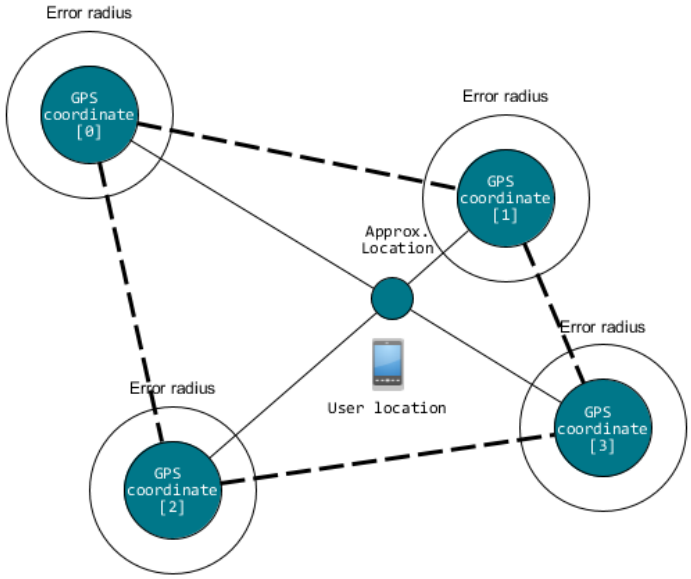


Fig 4. Cubic brute force tracking approach

Instead of simply requesting a single location from the devices GPS service, the application instead requests multiple coordinates, shaped in a cuboid pattern around the users location and models diagonal lines between the cubes corners. Theoretically, the point where these lines cross should approximate the users location more accurate than a single GPS request.

This approach is a brute force based method as it trades off system resources for better tracking results and should be evaluated thoroughly whether it is worth it or not.

Testing

UNIT TESTING

Automatic unit testing greatly streamlines testing procedures and ensures all tested functions are running as intended, therefore unit tests are to be implemented for core features which are the following:

- GeoLocator distance calculation
- Target generation around the user

As this project is intended as a working sample and is still pretty lightweight, unit testing for every little function is considered an

overkill as most functions are simple enough to test them manually and to be frank unit testing is pretty boring and doesn’t contribute to the projects progress that much.

USER TESTING

As this project is a mobile, GPS based application user testing is a way more effective method to ensure good functionality which means testing the app by actually using it.

This method is considered better because it gives an idea how good the product will actually work and it gives the developer the opportunity to go for a walk.

Architecture overview

The project uses Unitys inherit component-based architecture which basically means that each interactable object has a C# class to control its behavior and its properties. Every class that needs to interact with other classes defines the needed interfaces as public functions while keeping internally used functions private.

The design of classes is inspired by real-life correlations as far as possible. I.e. the class Shovel defines functions to dig up a pile while the class Pile defines a variable which represents how much dirt remains in the pile and Shovel calls Piles “reduceDirt(…)” function to represent a real shovel, removing dirt from a pile.

For a more detailed description of the architecture and inner workings of the app, please refer to the attached UML and BPMN documents.

Class overview

The following chapter gives a brief overview over all classes featured in the project, for a more detailed description please refer to the docs and the source code.

GEOLOCATOR

The class GeoLocator is probably the most important piece of the whole app, its purpose is to track the devices location and make it accessible to other classes. The class is implemented as a singleton which means that only a single instance exists at all times.

Updating of the devices position is called every n seconds which is controlled by a variable.

Furthermore the GeoLocator exposes a function to calculate the distance between two LatLongs and a function to get a random coordinate in a certain radius around a specified coordinate.

QUEST

The class Quest is used to manage all targets (QuestTarget), the user can reach and investigate when playing the game. It generates the related targets and treasure around the users initial location, regularly updates the devices distances to them and keeps track about its completion state.

QUESTTARGET

The class QuestTarget represents a single location, related to a quest and manages its behavior in relation to the users actions.

Targets are generated whenever a quest is initialized and receives a random location.

This class exposes functions for showing and hiding UI objects as well as functions to be called when a target is in range or out of range.

QUESTITEM

The class QuestItem is a simple parent class for gameplay objects to inherit from, like keys and dirt piles. It defines only a single function called “onUse()” which is implemented by child classes and determines the behavior when the player uses the item.

PLAYER

This simple class is used to represent the state of the player in the app. It just stores some objects like the players shovel, his/her current quest and a key once it has been found as well as some markers so the game engine knows where to place objects in the scene.

ASTROLABIUM

The class Astrolabium represents a device, rendered in the scene which gives hints to the user on how far away he/she is away from the next target by increasing the speed of its animation respectively.

This class basically features functions to control its animation speed.

CHEST

This class represents the chest containing a reward for every quest. Once the user has found the respective key, he/she is able to open

the chest. Chest defines variables to hold the key, related to the chest instance and the loot, the user can collect once opened as well as some functions used to control its behavior i.e. what happens when the user touches it.

COIN

This class represents the reward contained in chests. At the moment it can’t do much more than disappearing when collected. It’s imaginable that collecting a coin would increase a player score or currency value in later iterations of the project.

KEY

The class Key is another simple class and just sets the players key variable to itself when collected. It also defines a function to control its animation state.

PILE

This class represents a dirt pile which is shown when the user gets near a target coordinate. The Pile class defines a variable to track how long the user has to dig until the reward for digging at the location is shown and a function to decrease this value and triggering the spawning of the aforementioned reward.

SHOVEL

This class represents the users main tool to dig up rewards, hidden in dirt piles. It defines functions to control its animation state and calls of the piles dirt reducing function.

FOG

This class is used to control the behavior of the fog, rendered in the scene if the user isn’t near a target to hide “popping-up” of objects when the user gets in range.

It defines functions to clear and close the fog curtain and definitions of the different states the fog can have like idle, closing and clearing.

MATH

This class simply provides static functions to be used by other functions. At the moment it defines a single function which determines whether a value is between two other values.

MOCKPROVIDER

This class is used for testing the app. It’s supposed to provide sample objects to guarantee consistency while testing the app. I.e. it defines a function which returns a list of pre-set coordinates

for generating quest targets. More functions are defined when needed.

DEBUGUI

This class controls the behavior of a scene overlay which displays debug messages on mobile devices for easy debugging on the run.

Sources

[1] Statista Research Department „Number of daily Pokémon GO users in the United States as of August 2016“ 2016 [Online, accessed 25.06.2020]. Available: https://www.statista.com/statistics/589213/pokemon-go-user-number-us/

[2] Statista Research Department „Number of active users of Pokémon Go worldwide from 2016-2020, by region“ 2016 [Online, accessed 25.06.2020]. Available: https://www.statista.com/statistics/665640/pokemon-go-global-android-apple-users/

[3] Unity Manual [Online, accessed 26.06.2020]. Available: https://docs.unity3d.com/ScriptReference/Input-location.html

[4] Haversine formula [Online, accessed 26.06.2020]. Available: https://en.wikipedia.org/wiki/Haversine_formula

Figures