AR Tower Defense

An exploration of augmented reality on Android using Unity

Mobile Systems Engineering 2017-2018

Gabriele Minneci

AR technologies

The augmented reality market is growing every year, the available libraries are getting better and better. I describe the main libraries, explaining which platform (sw and hw) they support. Note that I've focused only on the android development, so there can be more for the other operating systems.

The more used libraries are **ARKit**, officially supported by apple and tailored on the apple devices only.

ArToolkit works with the Apple and Android systems, but it doesn't fully support the development on windows machines. It has libraries for Android Studio, Unreal Engine and Unity.

ARCore is developed directly by Google, but at the time I write is still in preview and it supports only the newest devices (Samsung Galaxy s8 and Google Pixel). A modified version exists on github (free-for-all), it is built to work with any devices running android 7.0 or greater, but it has problems with some device: the Samsung galaxy s7 is one of them. On the github folder there is a list of all the features that work in the specific devices.
ARCore is based on the Tango project which won't be supported anymore from the 1th March of this year. It supports the development on Android Studio, Unreal Engine and Unity.

Vuforia is one of the biggest AR platforms. It supports different environments: XCode, Visual Studio, Unity, Android Studio.
I've used Vuforia 6.5 on Unity to develop my app.

Vuforia

Vuforia is an Augmented Reality platform and supports iOS, Android and Windows 10. For this project I focused on Android: Vuforia uses the Android NDK and only the camera of the smartphone/tablet for the recognition.

Features (link)

The basis of the image recognition is a database where the images are saved together with the cloud points created. Everything is done in a very simple way in their website where you create a unique key for your project. The key is used by the platform to access to your database saved in the App.

The main features are:

- **Image Targets**: from a 2d image we can create any model or text on it
- **Model targets**: recognizes a pre-existing 3d model where to place AR content
- **Multi Targets**: recognition of 3d objects with flat faces, the AR elements can be model around it and are calculated based on the position of all the faces (fore

example if one face isn't seen but the AR element on it yes, the latter's seen part is rendered)
- **Cylinder Targets**: same as multi targets but with cylindrical or conical shapes (like cans and bottles)
- **Object targets**: can scan a 3d object and create AR elements on it (needs time for scanning)
- **VuMarks**: improvement of the bar code/ qr code. They store information and behave as an image target.
- **Smart Terrain** (deprecated): no more supported form Vuforia 7.0, scan and recognizes a flat area, starting from an already known object (better if 3d).

In the tutorial part of my app I've explored the cylinder target and the image target

AR Tower Defense

I've created a 3D tower defense game in Augmented Reality.

The technologies used are Vuforia 6.5, Unity 2017.2 and Visual Studio. The language used is C#.
I didn't use Android Studio because it isn't a good environment for developing games: I would have developed the entire game engine, with physics and game objects management. Unity has everything already implemented and well supported from years. The development with Vuforia using Android Studio is useful for less interactive projects, for example an Augmented Reality book or a room designer.

The first scene is a menu where we can choose between two different options:

- The real game (Tower Defense)
- A showing of some Vuforia's features

Vuforia Tutorial

Shows some basic feature of the platform:

- **Image Target**: spawn a group of trees on the target
    - **Virtual Button**: there is a button on the image that triggers a particle system. The virtual button is activated when the camera recognizes an occultation of

that part of image. The position and size of the button have limitations (see the official documentation for the details)

- **Multi target**: spawn one sphere per face in a box

There are different image targets to show how much important is the choice of the image: the number and position of cloud the cloud points greatly influence the image recognition. Also, this feature is well explained in the official documentation.

Tower Defense

My game is a simple 3D tower defense with the player and one enemy. Each of them has a tower and they spawn minions that fight the opponent.
The game will automatically start when the image target is recognized, and the map is loaded. The enemy will spawn the minions automatically each 2 second, choosing their type randomly; the player has buttons and an amount of coins that permit him to spawn the minion he wants. The coins are gained when its minions kill the enemy's ones.

The minions will automatically (using a navigator agent) go toward their enemy and they will fight the opposite minions they encounter on the way.

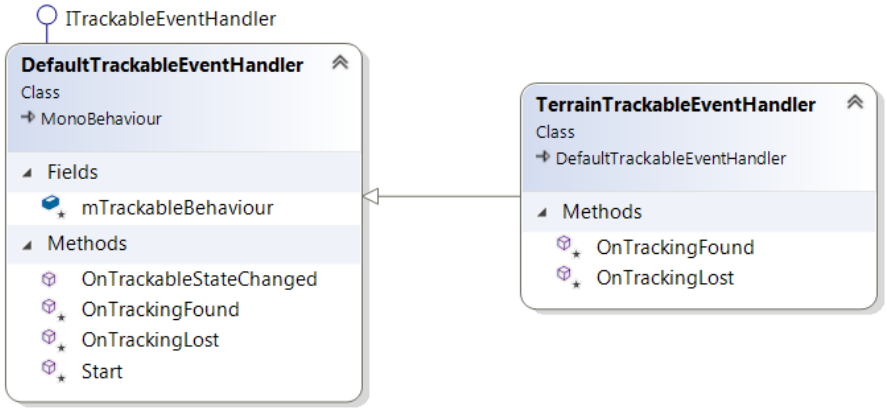
Features

1. **Image recognition and scene loading on it**
   When the image is tracked, the game is loaded and begins automatically. The map is shown also if the image target is no more on the screen, until some part of the scene can be seen.
2. **Automatic play/pause game with the loss of the image tracking**
   When the scene is no more visible on the screen the game pauses automatically until the image target is recognized again. At this point the game will resume immediately.
3. **Minions' spawn according to the available coins**
   The Enemy spawns his minions each 2 second for simplicity, but the design is thought for a more sophisticated implementation.
   The user Player can spawn minions based on the amount of coins he has. Any button is automatically disabled if the user can't spawn the specific associated minion.
4. **Minions' movement with Navigation agent**
   Minions know their enemy and go directly toward it changing the path if they encounter obstacles, stopping if they are in range for attacking an enemy.
   Each type of minion can have different speed.
   The path in my map is straight for simplicity, but the implementation would work also if the minions found a more difficult path.
5. **Minion's combat system with enemies' recognition**
   When a minion is in range for an attack against an enemy (minion or tower), it stops the movement and fight until no other enemy is in range or it dies.
   When a minion dies, it is destroyed and the information is sent to the listener objects (if they exist)
   Each type of minion has a specific range, attack, time between attacks and range.

6. **Coins update depending on specific events**
   The CoinsManager listens to the use of coins for spawning and the gaining of them from the death of the enemy's minions. It triggers the specific method and broadcast the change to the listeners: UI coin text and spawning buttons.

Architecture

Tracking System



DefaultTrackableEventHandler: Vuforia class that manages the tracking of the imageTarget

TerrainTrackableEventHandler: Derived class that overrides the OnTrackingFound and OnTrackingLost. Every GameObject inside the ImageTarget is rendered/disabled according to the gameplay necessities (see the problems encountered).

Players

Enemy: The computer player

Player: The user player

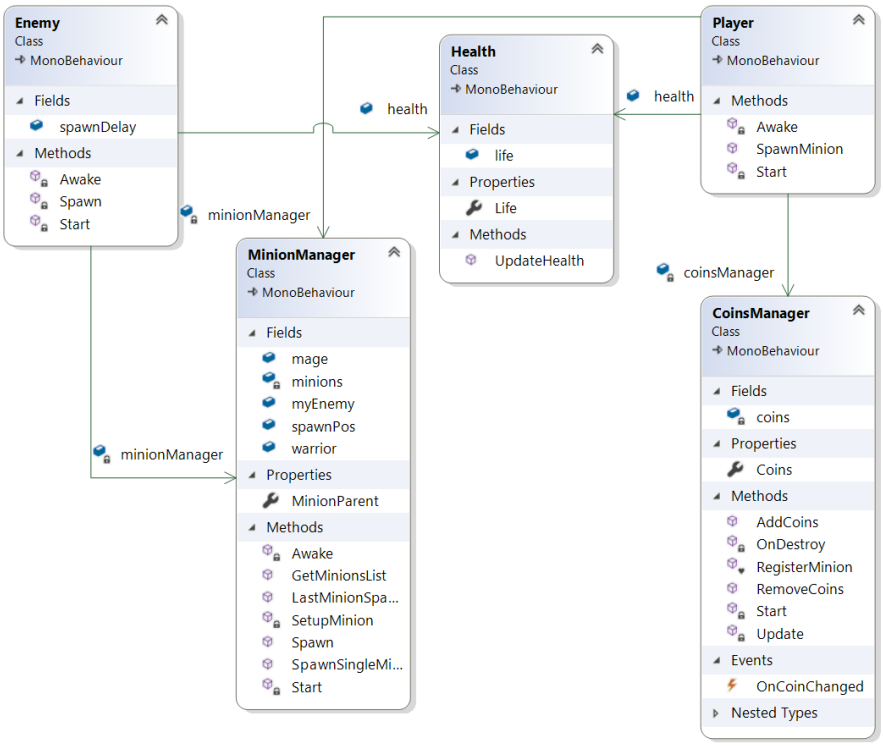
Health: Stores the player's health and its update. When the health is 0, it will ask (GameState class) for the end of the game

MinionManager: has the responsibility to spawn and store all the player's minions. It's the same class for both the players, only the parameters passed change

CoinsManager: Manages the coins of the user player, triggering all the events regarding it:

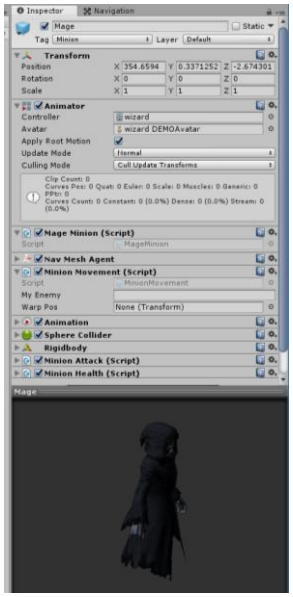
- UI text
- Spawning Buttons

Is registered in the event of the user player's minions that kill someone, to gain the coins.

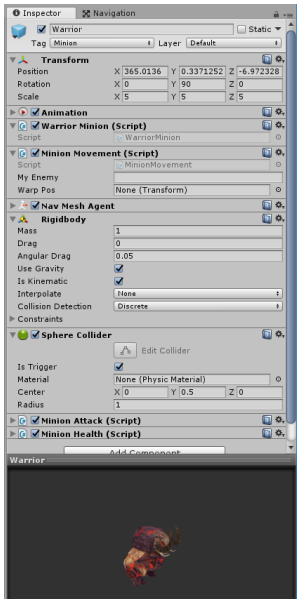


Minions

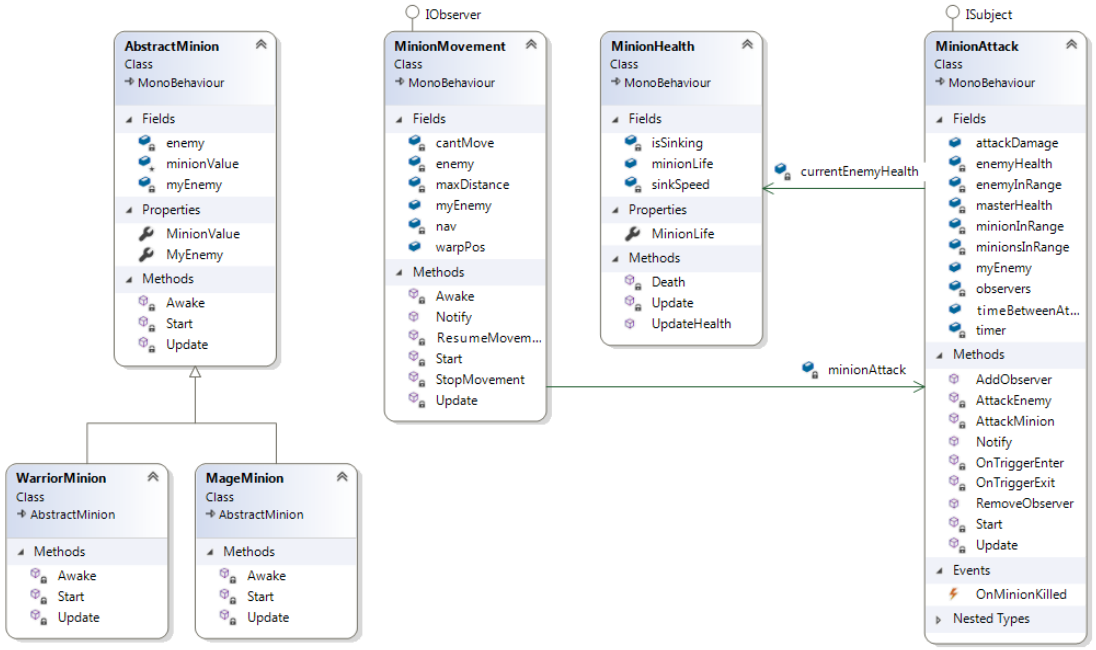
Each minion is a Unity GameObject where there are Unity specific elements (I won't go in the details here) and the following classes.



Mage



Warrior



AbstractMinion: Abstract class of the possible minions, stores common fields

MageMinion: The Mage minion, with specific statistics

WarriorMinion: The Warrior minion, with specific statistics

MinionMovement: Manages the movement toward the enemy. It has a NavMeshAgent that is the Unity class for managing the navigation in a predefined map, creating the best paths and updating them if there are collisions. It listens to the collisions managed by the MinionAttack

MinionHealth: It's responsible for updating the minion's health and its death

MinionAttack: manages the collisions and the combat system, storing all the minions in range for the fight and choosing the current opponent. To do this it overrides the Collider methods OnTriggerEnter and OnTriggerExit.

GameState

GameState: A Singleton class, visible from every other class of the project. It manages the pause/resume and end of the game. There is a button that change the state of the game and it's automatically triggered according to the image recognition.

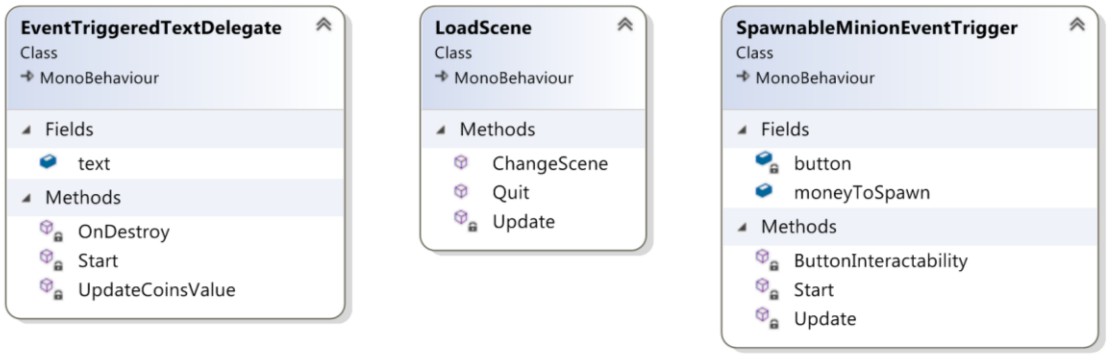
UI

EventTriggeredTextDelegate: update the UI text showing the current amount of coins

LoadScene: manages the moving between the different scenes of the application:

- Start Menu
- Vuforia Tutorial
- Tower Defense

SpawnableMinionEventTrigger: enables/disables the button according to the current amount of money



Problems encountered

Virtual buttons

Virtual buttons can be recognized by Vuforia only if they are placed inside the image target boundaries. According to the setup of my application, I couldn't create the purposed buttons for spawning the minions; after some trials to avoid the problem, instead, I've used a standard UI. As an example, the forest object in the Vuforia Tutorial scene has a virtual button.

Map Tracking recognition and map spawning

The default tracking wasn't enough for my purpose, because it's supposed to be used for static elements and not for a game scene. I have implemented the derived class TerrainTrackableEventHandler for this purpose. It enables and disables all the components of the game according to the tracking recognition. The hardest element to understand was the map with the Terrain component, because I had to disable the element visibility, but without deleting it completely, otherwise the position of the game objects (minions and towers) would have been affected: the physics would have been triggered and everything would have fallen. For this purpose, this class uses the GameState singleton to pause and resume the game.

Minion teleport on spawning

The enemy minions teleported in a different position after spawning. I figured out that the problem was for the position in the map, but I couldn't easily fix. It is a common problem in Unity (I've found many questions about it online) and the only solution I've found is telling the minion to teleport on the spawning position the frame after it is instantiated. The user can't see the difference.

Navigation Agent

The navigation agent has been tricky to implement for two reasons:

- Understand how to let the agent know about the terrain where it could navigate.

- Let the agent stop when the map is not visible anymore (tracking lost) and resume when it is visible again in the same position as before. The solution has been the automatic pausing and resuming of the game according to the image tracking: GameState is a singleton class that manages this feature.

Collisions

The collision between the minions has been easy, the problem has been how to let them recognize if the other was an enemy or not. I've used the Unity tags to let them know in which side they belong to.
The other problem has been how to let them stop when they arrive in front of the enemy tower. I've added a collider also to the tower, but the navigation agent sometimes enabled itself again and the result was that the minion entered in the tower through the walls. As a solution I've used a Boolean to block completely the navigator when it arrives there, in addition I've put a maxDistance to the destination point for the Navigator Agent: it will stop at that distance, but it's only for being sure that the problem won't happen if the Boolean doesn't work. The maxDistance is not related to the position and dimension of the tower.

Image focus tracking

I couldn't understand how the camera focus could be managed in the application. The default one is automatic, so if the image is too little, or the camera "decides" to focus another part of the screen (like the table behind the image), then the image tracking will be difficult. The degree of this problem depends a lot on the image chosen, I've seen that an image with good cloud points is not enough, also the distribution of them is important: if an image has equally distributed cloud points, tracking it without the proper focus will be hard anyway, instead if the points are more concentrated in some part of the image, the recognition will be easier.

Physics change in 3d world

Since this is a 3D game, the physics problem I've encountered are many, the most part were simple things to manage, but two of them have been hard:

- Blocking the physics when the game is in pause (See the navigation agent problem)
- When a minion stops its movement, I've seen that there still is something that let it move a little bit more. Searching for the problem I've found out that when an element stops moving, there is a force component that is still active and let the minion continue to move until this force is finished. Unfortunately, I didn't find a good solution. This problem is easily visible if the minions move fast (I've tried with speed 20), instead it is negligible if they move slowly as I have set in my game.