3D Augmented Reality: Project #1

Due on April 2020

Prof. Simone Milani

Elena Camuffo 1234370

Contents

1 Overview

1.1 Project

As far as Augmented Reality has become very popular in the last few years, gaining spread and diffusion all over the world, it can be supported by a number of different devices and may assume consistents behaviours. We want to provide with this project an example of such application for people's entertainment.
The presented project is a **game**, initially built as a standalone to correctly develop the whole environment, game logic and interface, and it is then transported to a mobile platform to serve as an Augmented Reality application for Android Systems.

The game consists in a **recreated and modified version of "fruit ninja"**, the popular and widespread game for smartphones and tablets where the player, that is a *ninja girl*, can jump and move around the 3D world in order to punch or kick the falling objects (i.e. fruits) and gain points paying attention to not loose her lives.
Some variants have been made w.r.t. the original game to make this version fit better the 3D representation and also the adaptivity to Augmented Reality as well.

Figure 1: The ninja girl model.

1.2 Softwares

The whole implementation is done using **Unity 3D 2019.2.15.f1** and the most of the game logic is developed exploiting **Visual Studio 2019** for scripting, the software attached to Unity by default.

The model of the ninja girl has been provided. The other models and all the animations were taken from the internet[1], and many of them refined by means of a computer graphics tool, i.e. **Blender 2.82a**.
The sounds were taken from the internet too[2].

2 Project inside

We are going now deep into the Project structure in order to investigate its main features and aspects.

2.1 Interface

As the game starts, the **Game User Interfaces (GUI)** is shown over the screen of the device.
It is composed of the following *static panels* (fig. 2):

[1]Models taken from many websites i.e. `https://www.turbosquid.com/`,`https://www.mixamo.com/`, `https://https://www.cgtrader.com/`, `https://free3d.com/it/`

[2]Sounds taken from `https://www.freesoundeffects.com/` and `http://soundbible.com/`

- **Name.** The name of the player. It is *inserted by the user* through a dynamic window at the beginning of the game.
- **Score.** The score is set *initially to zero*. Its increment is controlled via scripting and occurs as soon as the player hit dedicated objects (fig. 6).
- **Lives.** The lives are *initially three*. Their increment and decrement is controlled via scripting and occur as soon as the player hit dedicated objects (fig. 6).
- **Level.** Starting from *level 1*, as the time proceeds, the level grows.
- **Time.** Total amount of seconds from the beginning of the game.

In addition, in the middle-up screen two little buttons are situated:

- The **pause** button freezes the game in the moment it is pushed. Pushing again the game is resumed.
- The **exit** button closes the application.

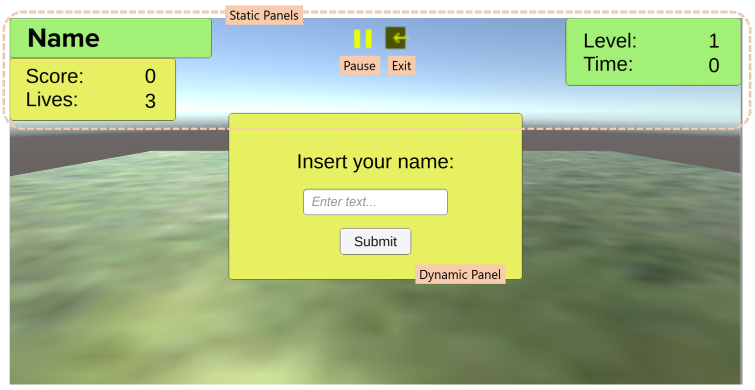
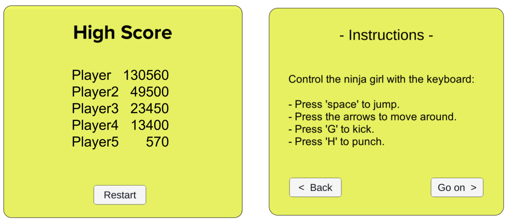


Figure 2: The Game User Interface (GUI).



```
Player   130560
Player2   49500
Player3   23450
Player4   13400
Player5     570
```

Control the ninja girl with the keyboard:

- Press 'space' to jump.
- Press the arrows to move around.
- Press 'G' to kick.
- Press 'H' to punch.

Figure 3: Dynamic panels.

In the center of the screen we can find a *dynamic window panel* (figg. 2 and 3). It allows to:

- **Insert your name** at the beginning of the game.
- Visualize the **instructions** at the beginning of the game.
- Visualize the **highscore table** at the end of the game (i.e. when your lives decrease to zero).

All these buttons' control is managed via scripting as well.

2.2 Game Logic

The *objective of the game* is to collect as much points as possible collecting objects, moving the ninja girl in the world and trying to preserve and not to loose lives.

2.2.1 Player's movement

To move the player in the 3D world, an animator controller is needed (fig. 4).
The animator controller and the related scripts were taken from the *standard assets library* (3^{rd} person controller) and then modified to fit the requirements of the game. The basic controller includes:

- The **idle** animation, i.e. the one running by default.
- The possibility to **move around** the 3D world, capturing the intensity of the walk (i.e. little movement makes her walk, hard one makes her run).
- The **jump** action, supplied with “airbone” and “crouch” actions to make it more realistic.

For the game's purpose the following actions were addeed:

- The **kick** action.
- The **punch** action.

Some parameters, controlled via script, governs the animation change (fig. 5). The input commands change according to the device possibilities (section 3).

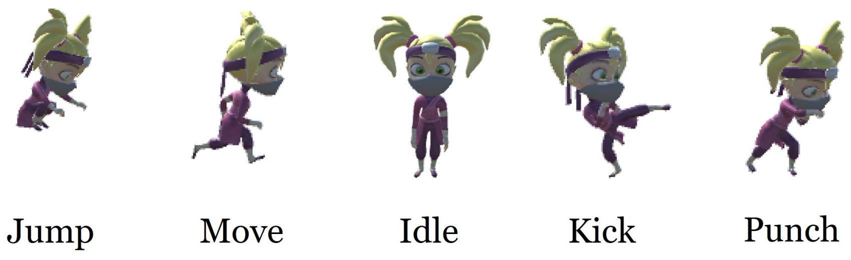


Figure 4: The different animations of the player.

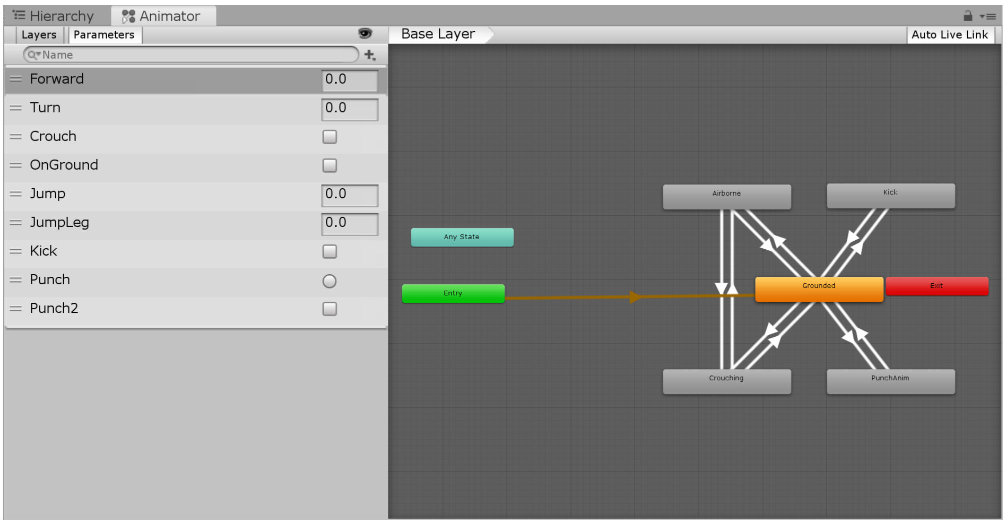
2.2.2 Objects type

The collectable objects are of many types with different behaviours and functions in the game. At the beginning of the game the set of object that the player can find includes:

- **Coin.** Adds 10 points to the score.
- **Fruits** (orange, apple, banana, cherry). Each fruit adds 100 points to the score if hit (punch), 200 points if kicked (special command).
- **Bomb.** Lose a life.
- **Cupcake.** Gain a life.

As the game proceed, the level grows and other objects appear. In particular:

- **Box.** (level 10) Useless object. It is cumbersome but you can jump on it.



Forward	0.0
Turn	0.0
Crouch	☐
OnGround	☐
Jump	0.0
JumpLeg	0.0
Kick	☐
Punch	○
Punch2	☐

Figure 5: The animator controller of the player and the relater parameters involved.

- **Book.** (level 15) If hit, it makes all bombs present in the game disappear.
- **Star.** (level 15) Adds 1000 points to the score, but it holds for few seconds.
- **Ghost.** (*coming soon*) It follows you and if hit, makes you loose 3 lives at a time.

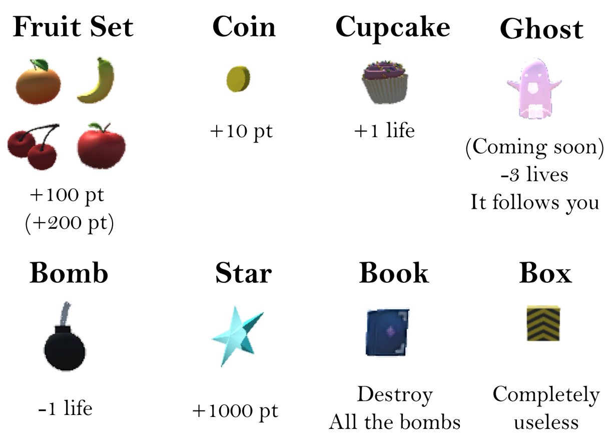


Figure 6: The different object involved in the game and their specifications.

2.2.3 Falling Objects mechanism

The collectable objects *fall from the sky* following a *logarithmic law* as a function of time, i.e. the interarrival time (the time between an object and the following) is:

$$falling(t) = \frac{k}{\ln\{level(t)\}},$$

where $k = 2s$ and $level(t) = \lceil \frac{t}{20} \rceil$.
The next arrival (i.e. next object that will fall) is chosen at random according to a probability distribution

over the choice of the possible object at that level. In particular the different probabilities associated to the objects, were assigned according to their power, and they change as the level grows as figure 7 shows.

This mechnanism works because Unity has an integrated system which can simulate forces and in paricular *the gravity* and treat the objects as *rigidbodies*.
This system is also responsible to detect and manage *collisions* between the objects.

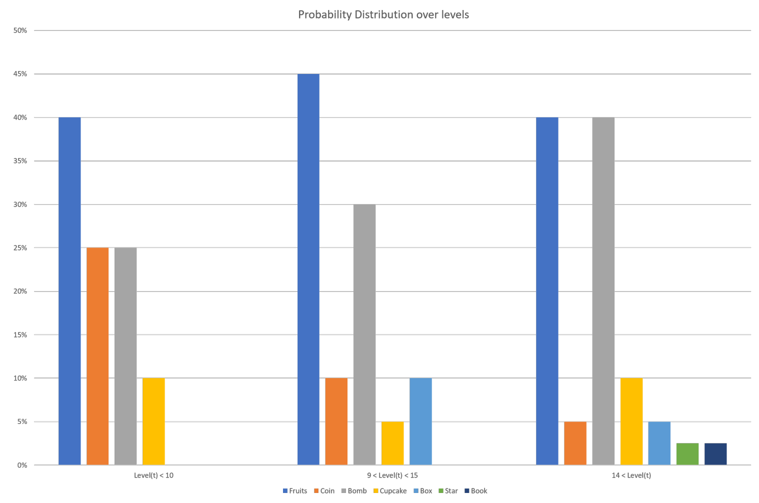
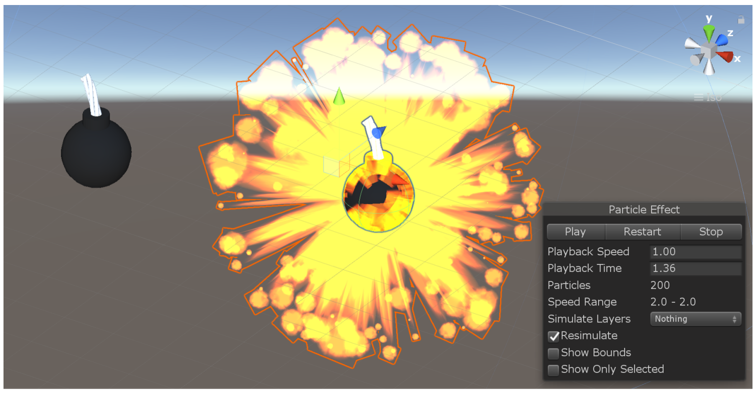


Figure 7: The probability of the fall of the different object through the levels. The *fruits* includes the four types, equiprobable.

2.3 Effects

In order to improve the game quality, some visual and audio effects were added.

The **Visual Effects** include the animations of the player as well as the *explosion* of determinate objects. The animations were taken from `www.mixamo.com`, while the explosions were created exploiting the build-in **particle system** effects in Unity. The bomb object and the fruits can explode, but their animations are different, obtained wisely *tuning the parameters* of the particle system component (fig. 8).



Playback Speed 1.00
Playback Time 1.36
Particles 200
Speed Range 2.0 - 2.0
Simulate Layers Nothing
☑ Resimulate
☐ Show Bounds
☐ Show Only Selected

Figure 8: The bomb explosion particle system.

The **Audio Effects**, taken from the internet (see section 1.2) occur when an object is hit, and they are different according to the selected object. An audiotrack in background is added as well.

3 Platforms

The game was exported for three different platforms, taking care of **adapting** the scripts for input commands and such kind of stuff according to the modalities of the different devices.

3.1 Windows Standalone

The first exported version of the game is the one for Windows desktop, because it is the easiest to develop and to use in the testing phase.
This application, once built, runs autonomously on every Windows-based device and takes as an input the commands given by the user by means of the **keyboard** (fig. 9). In particular:

- Use the **arrows** to move around.
- Use the **spacebar** to jump.
- Use the **G key** to kick.
- Use the **H key** to punch.

As it is not yet an AR application, the girl stays on a green plane. If the player falls down from it, *looses a life.*
In addition the **virtual camera**, having a skybox background set, is handled via script and *follows the player's movement* around the 3D world, keeping a fixed offset distance.

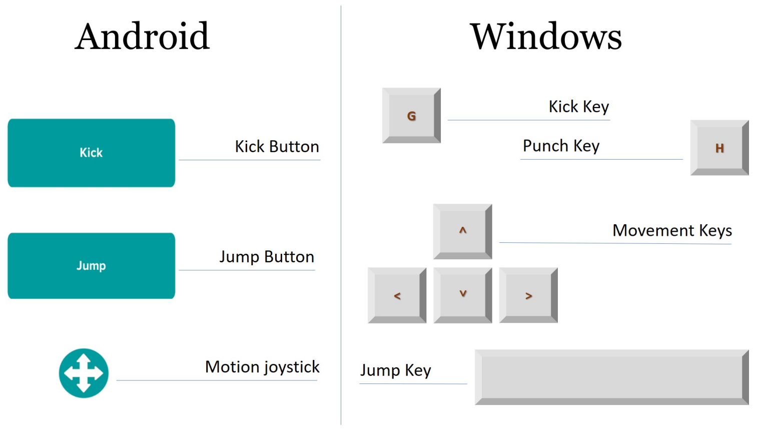


Figure 9: Control keys for Windows and Android systems.

3.2 Android Standalone

Then the application is exported on Android-based moblie devices (the oldes supported version is *Kit-Kat*, i.e. *Android API 19*).
In order to fit this platform, the commands need to be changed (fig. 9). In particular, the **touchscreen** input method must be exploited as follows:

- A **kick** button is shown on the screen to make the player kick.
- A **jump** button is shown on the screen to make the player jump.
- A **joystick** is shown on the screen. Dragging it in a direction makes the player move in that direction.

The setting is the same of Windows version and also here the **virtual camera** follows the player's movement.

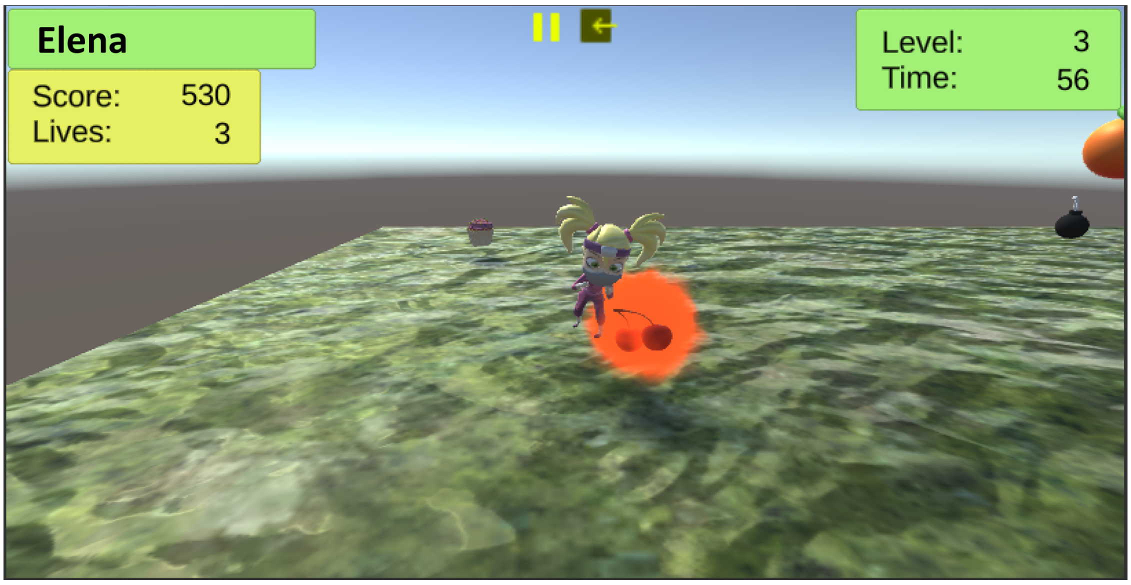


Figure 10: Example of the desktop application.

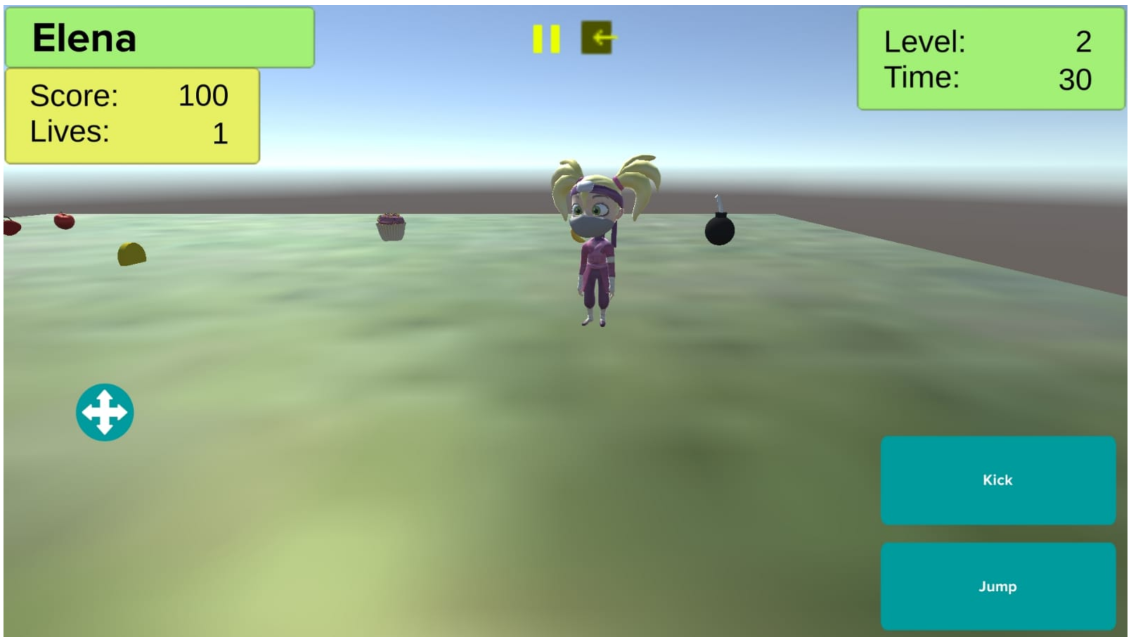


Figure 11: Example of the mobile application.

3.3 Android Augmented Reality

The final and goal version of the application is the Augmented Reality one, supported by Android-based mobile devices.

The projects must be equipped with an additional package, **Vuforia Engine**, which provides a tool to manage and some instruments to implement Augmented Reality functions in the Unity environment.
Vuforia provides a system of Augmented Reality marker-based, i.e. to set an image as a target to be recognized by the device's camera (fig. 12).
The game starts as soon as the target image is framed, and the objects appear over it, laying on an invisible infinite plane, *coplanar with the marker* and having the *same origin*[3].
It is important to remark that the phisical system of Unity is enabled (see section 2.2.3), so the marker must be put on an *horizontal surface* to make it work properly (figure 14).

The insertion of the image target leads to modifications in the scene structure, in particular the *hierarchy of the scene*. In this case all the elements of the scene have to be set as *children* of that marker (fig. 13), made exception for the camera object. The camera is here set to the default **AR camera** of the Vuforia package, which allows to see the virtual objects overlay on the real world.

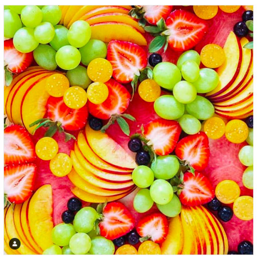
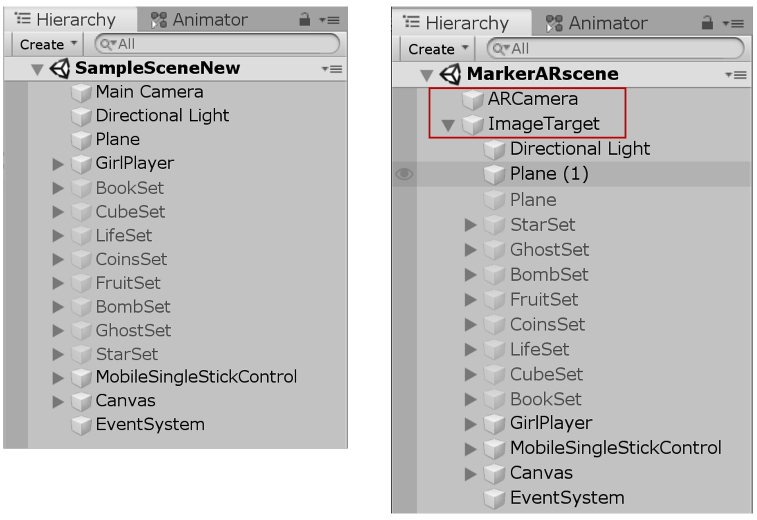


Figure 12: Image target chosen as a marker.



```
Hierarchy  Animator
Create  All
▼ SampleSceneNew
    Main Camera
    Directional Light
    Plane
  ▶ GirlPlayer
  ▶ BookSet
  ▶ CubeSet
  ▶ LifeSet
  ▶ CoinsSet
  ▶ FruitSet
  ▶ BombSet
  ▶ GhostSet
  ▶ StarSet
  ▶ MobileSingleStickControl
  ▶ Canvas
    EventSystem
```

```
Hierarchy  Animator
Create  All
▼ MarkerARscene
    ARCamera
  ▼ ImageTarget
      Directional Light
      Plane (1)
      Plane
    ▶ StarSet
    ▶ GhostSet
    ▶ BombSet
    ▶ FruitSet
    ▶ CoinsSet
    ▶ LifeSet
    ▶ CubeSet
    ▶ BookSet
    ▶ GirlPlayer
    ▶ MobileSingleStickControl
    ▶ Canvas
      EventSystem
```

Standard Scene Augmented Reality

Figure 13: Hierarchy of the scene (a) without AR (b) with AR.

[3]As the plane is here invisible it is set also infinite (very very big), in order to prevent an accidental falling.

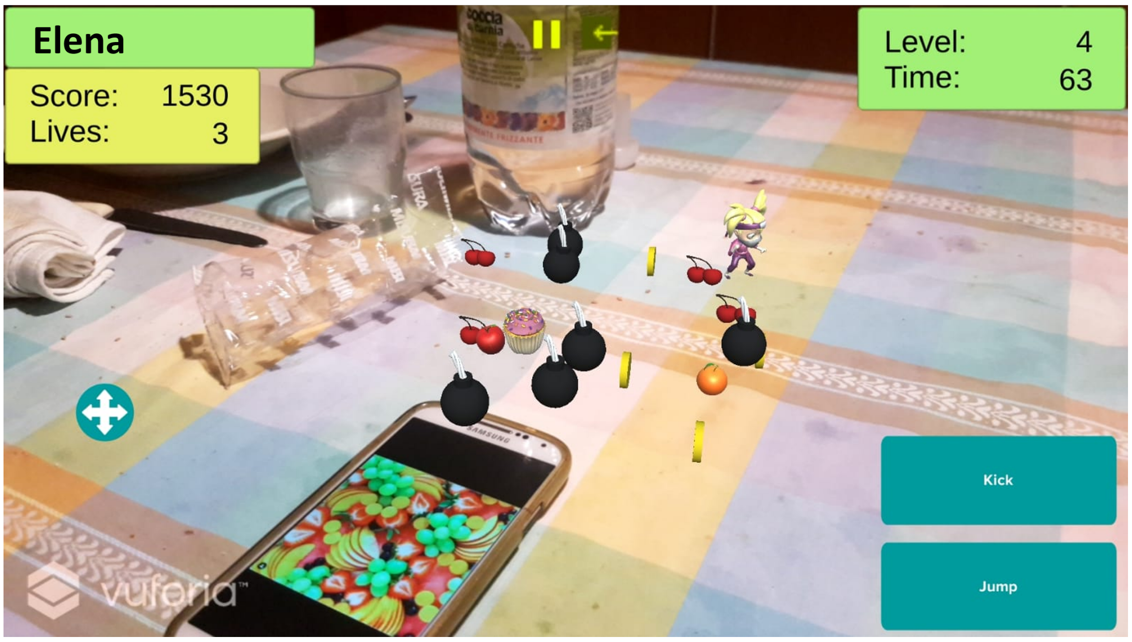


Figure 14: Augmented Reality game over my kitchen's table.

3.4 Towards HoloLens

In order to deepy exploit the potential of the Augmented Reality in all its aspects, an *headset platform* like HoloLens has to be chosen.
The goal is to change the Augmented Reality into **Mixed Reality** in such a way to make possible the interaction between the virtual objects and the real world. HoloLens for example has a buid-in system which allows such kind of integration.

As far as an headset device is used, all the input commands methodology has to be changed. HoloLens has a system of **gestures recognition** and **speech recognition**, which can be exploited to gather the input of the user. In partucular:

- A **virtual pointer** can be shown, to allow the positioning of the ninja girl player in the real world.
- The **"jump"** command is achieved by making the system recognizing that same word.
- The **"kick"** command is achieved by making the system recognizing that same word.

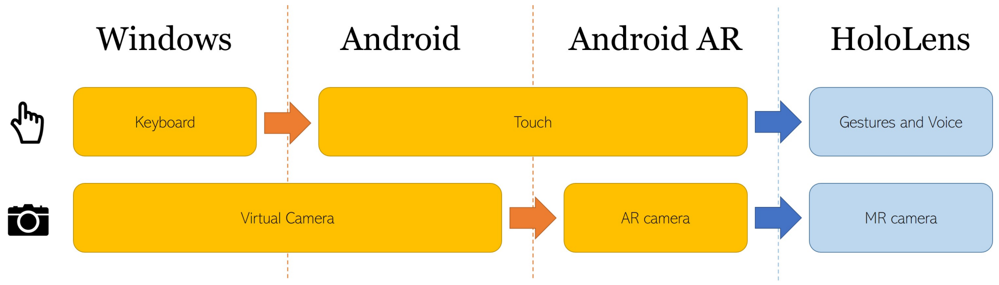


Figure 15: Scheme of the changements made up to the various platforms.

4 Conclusions

As reported up to this point, the game briefly follows some guidelines of the orginal *fuit ninja* game, but it brings lots of innovation in each of its aspects. In particular:

- The **3D world space** and the 3^{rd} person player represent the first clear advance w.r.t. the original 2D version, bringing depth and a new perspective to the game environment.
- The introduction of **Augmented Reality** represents another great improvement in the game visualization, bringing novelty and such kind of step towards new technologies.
- The **adaptation to HoloLens** and/or to other headsets is definitely the best changement introduced w.r.t. the traditional game conception. It completes the integration of the world with virtual objects and provides a new game modality for the user.

It is to notice in advance that the projectual choices made reflect the creator's own taste and experiments, but they can be change, improved or developed according to a deep testing or invesigation, as well as the users' preferences.