Corona Killer AR

I. Introduction

This report documents and explains the Augmented Reality (AR) game developed in the course *Augmented and Virtual Reality* at the University of Bordeaux, 2020. In this game, a user is tasked to kill virtual corona viruses that are spawned in a real environment with a spray can. It has been developed with Unity, and the project files can be found online[1] and the demo to game can be found at [2].

The report is organised as followed: First, the game and its objective are described and shown. This is followed by an explanation of the architecture of the projects and its implementation. Afterwards, a discussion follows where potential additions to the applications are discussed and limitations of the HoloLens addressed. In the final section, a conclusion is given.

II. Game Description

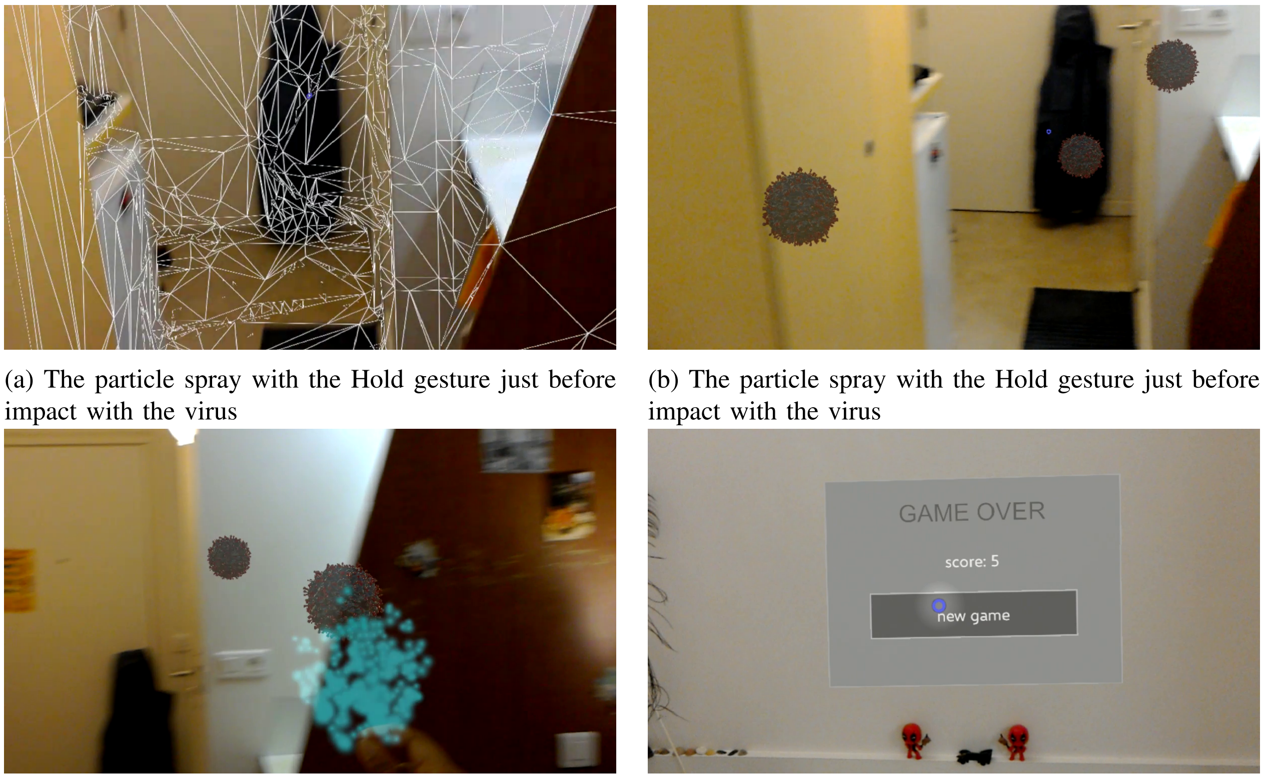
The basic story of the game is that during this pandemic the crazy viruses floating all around the player's world. The HoloLens 1 is a device with the special capability of being able to see these floating menaces. Equipped with the strongest sanitizer spray at the tip of the fingers of the player, the aim of the game is to spray these viruses and kill as many possible, while avoiding getting hit by one of them. All these viruses are on the move to get to the player (they have a motion directed to the player position), and can come from any part of the room randomly. The HoloLens offers the gesture of hold (press the thumb and index finger together for a long time) to start the spray of sanitizer and the moment the spray touches a virus it is successfully killed. Even if one virus gets past a safe distance from the player position the game is over and they need to be immediately quarantined or as this is a game it is possible to begin with a fresh start and a 0 score.

The gameplay follows the simple rules of a First Person Shooter (FPS) game and has been implemented in a similar fashion, with the difference being that the system is an AR system. The viruses and the spray can are virtual objects, the gesture tracking is done on the real hand that is detected by the device, and the environment is the room in which the user stands. The further sections explain the game mechanics, some images of the gameplay can be seen in Fig. 1.

III. Architecture & Implementation

This section describes the architecture of the project, the responsibilities of each module and the interaction between them. For implementation, Unity *2017.4.40f1* is being used, as this version is compatible with the offical Microsoft HoloLens Toolkit[3], version 2017.4.0.0, for the HoloLens 1. This toolkit offers common functionalities for AR, such gesture detection as well as spatial mapping. Note that building of the application is strictly bounded to the above mentioned

[1]https://bit.ly/3pD1nVL
[2]https://youtu.be/XBMjbjCnKas
[3]https://github.com/MSHoloLens/HoloToolkit-Unity



(a) The particle spray with the Hold gesture just before impact with the virus

(b) The particle spray with the Hold gesture just before impact with the virus

(c) The particle spray with the Hold gesture just before impact with the virus

(d) Game Over Screen

Fig. 1: Gameplay

versions. A diagram of the project architecture can be seen in Fig. 2. We only include self-implemented objects and prefabs from the HoloLens Toolkit that are explicitly integrated in the project in this diagram, further programmatical dependencies to other objects in the toolkit are omitted in the visualisation. The following subsections explain the functionalities of each of these objects.

A. *Spatial Mapping & SpatialProcessing*

Spatial Mapping is responsible for scanning the environment, and create a mesh from it. SpatialProcessing allows to process this mesh, remove the noise, and fit planes on them.
Both of SpatialMapping and SpatialProcessing prefabs are borrowed from the HoloLens Toolkit. The Spatial Processing object offers an interface for scanning a room and turning this information into game vertices. Furthermore, it offers some higher order processing functions that allow to detect planes from these vertices. These planes are further classified into subclasses, such as *wall*, *floor*, *tables* etc. These planes are used as a base for spawning the viruses: We do not spawn them randomly in the game space, but on horizontal and vertical worlds. The SpatialMapping object manages the connection between the observed surfaces and the rendering of surface planes and makes the detected planes visible.
Once the SpatialProcessing object is finished scanning the room, which is determined by its *ScanTime*, and enough floor and wall planes are detected, it fires the multiple event. Firstly, it

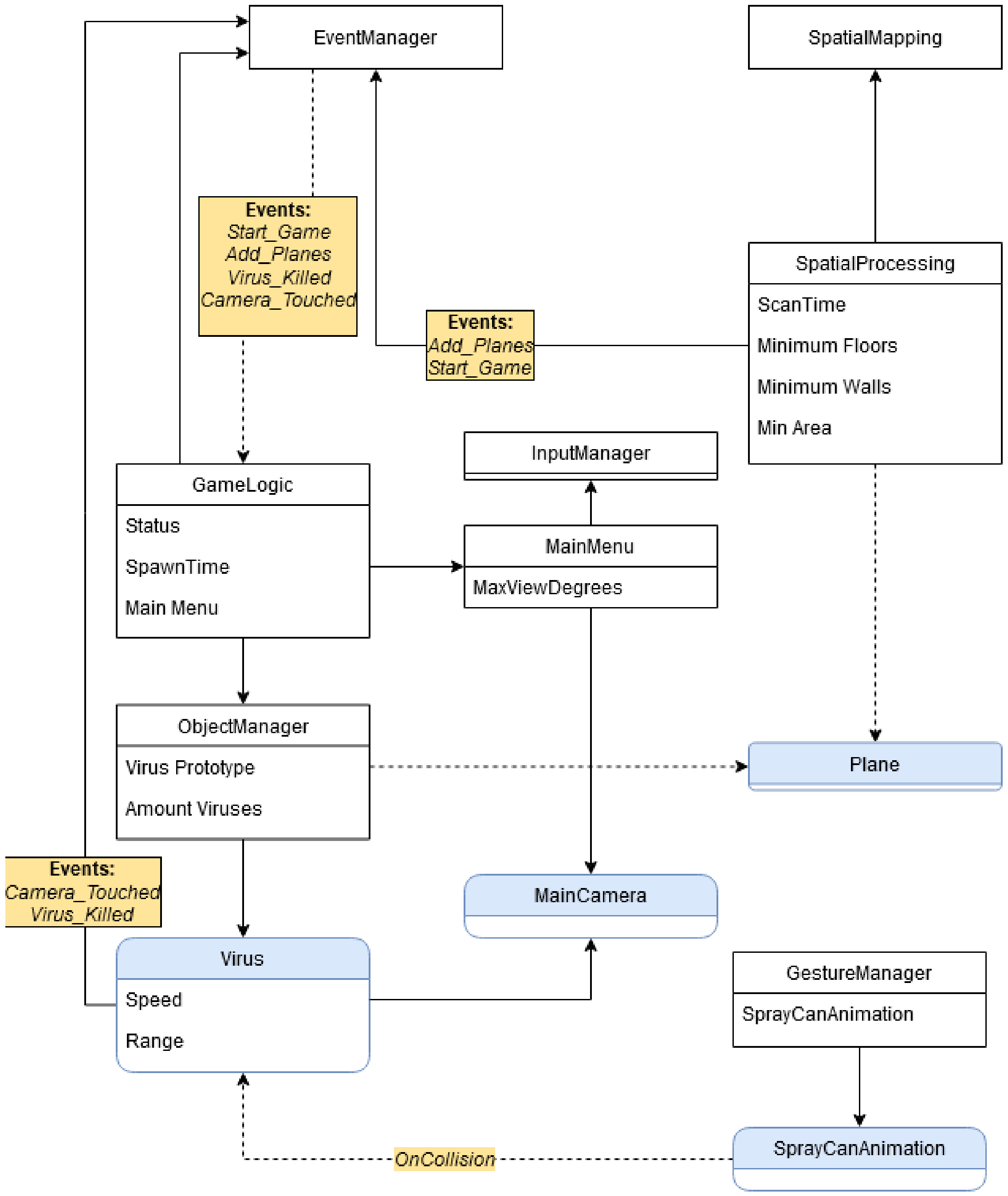


Fig. 2: UML Diagram of Project
White boxes show GameObjects or Prefabs, responsible for enclosing a programmatical unit, without any relevant physical position in the game world. Blue boxes show GameObjects with a physical position in the world which involve some sort of interaction with the player. Yellow boxes show events. Only for us relevant field variables are shown. Each of the objects may consists of multiple scripts. For simplicity, we omit the differentiation into multiple scripts.

lets the GameLogic object know that planes are detected. Secondly, it sends the event that the game is ready to start. If not enough planes have been detected, the scanning procedure restarts.

B. GameLogic

The GameLogic object implements the game logic by connecting the functionalities of multiple objects, reacting on certain events and keeping track of the game status and game score. It handles the functionality of the following events:

- *Start_Game*: It sets the game status to *running* and starts spawning viruses by calling *SpawnVirus()* on the ObjectManager. How often this function is called is regulated by the *SpawnTime* field.
- *Add_Planes*: Handles explicit casting of the event payload to GameObjects and passes them to the ObjectManager.
- *Virus_Killed*: Increases the game score.
- *Camera_Touched*: Signalises that the camera has been touched by a virus. This sets the game status to *game over* and sets the game over screen visible.

C. ObjectManager

The ObjectManager handles spawning of viruses. Once planes have been detected and added, the ObjectManager computes the area of each plane which is used to further calculate spawning probabilities for the viruses. The probability of a viruses spawning at a specific position on a plane directly corresponds to its area, so that that the viruses are evenly distributed across a room. Furthermore, the maximum amount of active viruses at any given time can be limited by the *Amount Viruses* field variable.

The Virus object, once spawned, consistently moves towards the camera, i.e. the user. The speed of the virus can be varied by its field variables. In addition, the virus is associated with a range. This range determines at what distance the virus comes too close to the camera, this triggers the *Camera_Touched* event which further signalises that the game has been lost. If an a collision with the spray can particles is detected, the virus GameObject is destroyed and triggers the *Virus_Killed* event.

1) Spatial Sound: To enhance the experience of the Gameplay spatial sound has been added to the Object Manager. As it is the object that controls the objects in the game, a 3D sound spatializer has been used to give feedback to the player that the Game has begun. It plays on loop and based on movement around in the room has a slight Doppler effect along with a logarithmic roll-off to give the feeling of moving around the room farther from the point where they began playing the game.

D. GestureManager

The Gesture Manager handles the main gestures recognized by the HoloLens, and associates appropriate callback functions to each, allowing the user to interact with the game.

The first is the *Air Tap* gesture, as shown in Fig. 3. This gesture is commonly used in conjunction with the *gaze pointer* to select and click menu items an option. For example, this is used for interacting with the menu as shown in Fig. 1d.

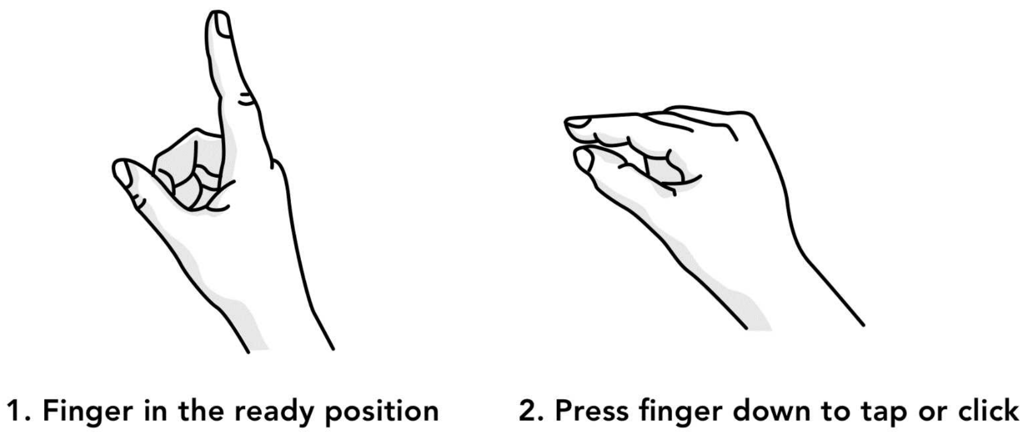

1. Finger in the ready position
2. Press finger down to tap or click

Fig. 3: Air Tap gesture

The next gesture that is used to play the game is the *Tap and Hold* gesture. If one were to *Tap* as shown in Fig. 3, and then hold the finger in the downwards position, it produces the *Tap and Hold* gesture. There are three associated callbacks for the *Tap and Hold* gesture. The *Hold Started* callback, starts the spray particle systems which are used to kill the viruses. The *Hold Completed* (called when *Hold* is released) and *Hold Cancelled* (called when hand exits the view during *Hold*) gestures both stop the particle system. To account for player's gaze, the manager also sets the orientation of the particles to the same as the gaze so that the direction can be controlled in that way. The update interaction function ensures that every frame the position of the particles is the same as the hand. The Air Tap works for the Game over but is handled separate to the Gesture Manager.

1) Particle System: The spray is defined using a Unity particle system, with collider defined around each individual particle. The particle system takes a cone shape and is set to loop over indefinitely when instantiated.

The over-all particle system is kept at a low emission, allowing for around 750 particles at any given moment. This ensures that the particle cloud is translucent enough that the user can still look through it, and also ensures that the particle system does not become computationally expensive.

E. EventManager

To enforce loose coupling between objects, an event managment system has been adopted. It is mainly based on two online resources [4][5]. It allows objects to register itself to listen to certain events and bind specific actions to these events. These events can be accompanied by a payload object, which can carry type of data. Furthermore, it allows objects to trigger certain events, without having a direct dependency to other objects. These functionalities are widely adopted across the application.

[4]https://john-tucker.medium.com/discovering-unity-eventmanager-a040285d0690

[5]https://stackoverflow.com/questions/42034245/unity-eventmanager-with-delegate-instead-of-unityevent

F. MainMenu & InputManager

The MainMenu and InputManager is responsible for handling the graphical interface of the game, that can be seen after the player is killed, and the game is over, and it can be seen of Figure 1 (d).

This interface contains, the score of the user and a button to restart the game. $MainMenu$ is a prefab from the HoloToolkit examples, and served as a base, to be modified according to the game. It is responsible for the visualization, and behaviour of the Game Over windows.
The $InputManager$ is a bridge between the user and the MainMenu, receiving user interactions, such as gaze and gesture, and triggering the corresponding event in the MainMenu.

1) Billboarding and Tag-Along: Billboarding is a behavioral concept, and is used to have holograms always face towards the player. Tag-along makes sure that the objects always attempts to stay in range, and never fully leave the players field of view. As the user moves its head, the object with tag-along will always slide towards the edge of the field of view. As the users glances towards it it enters the whole view. To add Billboarding and Tag-alonging to our window, SolverRadialView was added to the MainMenu GameObject. It has several parameters. The MaxViewDegrees were set to 15 degree (was lowered from 30 degrees). It is responsible how far the widows can leave the center of view, and it has to be lowered, since the HoloLens has a small view.

2) Behaviour of Main Menu: At the start of the game, the GameManager disables the main menu. Once the game is over, and event is triggered, and the GameManager sets the window active. It loads the score from the GameLogic, and window becomes visible.
The Gaze is activated, and the user has to focus on the restart button. Meanwhile the button is focues, a hold gesture shall be performed for 0.5 seconds, and a new game will be started. When the MainMenu is activated the InputManager starts to trace the gaze of the user. As it hits the collider of the $newgame$ button, the InputManagers' GestureManager starts tracing the gestures. As the hold event appears, a timers starts, and after 0.5 seconds, a new game is started, new virus starts to spawn, the $MainMenu$ window deactivates.

IV. Discussion & Limitations

This section illustrates the current shortcomings and limitations of the game. Some of these limitations could be addressed in further iterations of the game, however, a few of them directly arise by the restrictions of the HoloLens 1. Some of the limitations that exist currently are:

- **Virus Spawning**: The viruses currently can go through walls. This can be handled by applying some form of path finding for the virus. One observation while playing is that sometimes the viruses might spawn quite close to the player. This is troublesome due to the small Field of View (FOV) of the HoloLens and so some functionality of spawning viruses farther from the camera could be useful to improve the gameplay experience.
- **Virus Localisation:** Currently, it might be troublesome to spot viruses due to the small FOV. Hence, an arrow on the screen showing the direction towards the nearest virus could make it easier for the player to spot viruses or add some sound when a virus reaches close to the player could provide the feedback. Alternatively, a 2D map with the virus locations should be displayed on the screen.

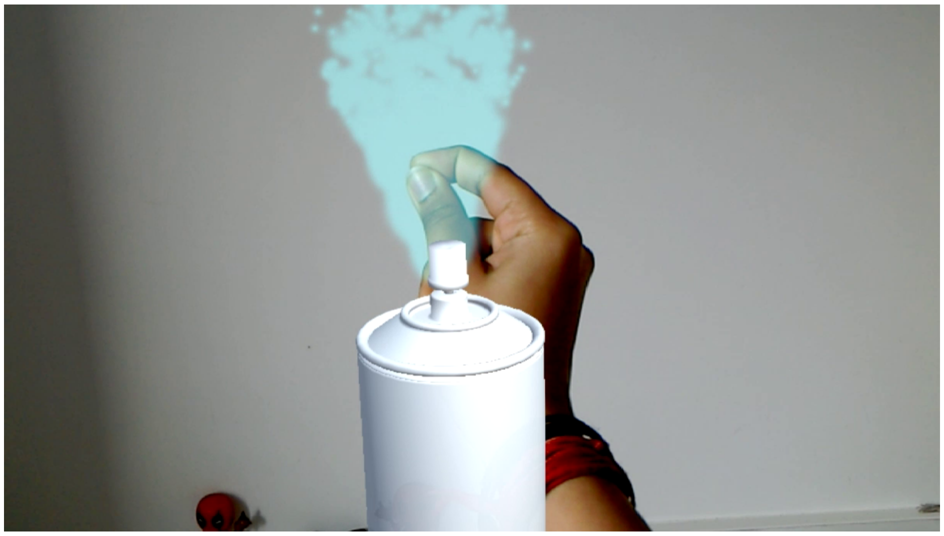


Fig. 4: The particle spray and the spray can with the Hold gesture

- **GUI Improvements**: Adding a health bar for the virus, Animations when the virus is destroyed, sound effects to improve experience are some embellishments that can make the game more interactive. Adding the current score on screen was avoided so that it does not get in the way of playing because of the small FOV. Also add a starting screen, scanning room screen and then a game start message would let the player know the stages in the game.
- **Trouble with Spray Can**: In addition to the particle system, a spray can was also part of the prefab to be displayed. However, while it displayed correctly when tested separately with just the gestures, the spray would not appear once integrated with the full project. Different kind of materials were used on the spray can which didn't solve the issue. Fig 4 shows the spray can object which was tested separately.

We can see that some of these limitations are a direct result of the limited FOV of the HoloLens 1. Due to the fact that it is relatively limited, big objects often are cut at the edges of the screen, such as approaching viruses. To prevent this, the clipping plane can be potentially adapted, however, this results in objects not being rendered once they come to close which can be problematic with the current game mechanics. Both variants influence the immersion negatively. An alternative could be to have stationary viruses that shoot towards the player, some HoloLens 1 games solve the FOV problem this way.

Another problem is also some time delay in recognising the start of the gesture hold. This makes the game at times feel unresponsive, as it times to register the human intent.

Furthermore, the computational resources are limited. This is especially noticeable when dealing with particles, where we had to fine-tune the particle amount so that the frame rate does not suffer. In addition, the quality of the 3D objects is set to low for the same reason. The most realistic option for particles, *mesh rendering*, causes significant lag in the game. Therefore the lower quality, *billboard rendering*, was used instead.

V. Summary

In the course of this module, a functioning AR game prototype has been developed for the HoloLens 1 with Unity. It implements a basic game loop with various interaction elements. The game has been implemented in such a way that it allows for easy addition of extra features. However, some of the functionalities are limited due to the restriction of the HoloLens 1 itself. Therefore, it will be interesting to see the capabilities of the HoloLens 2.