Architecture Design

Group: House Gryffindor (1)

Date: 17-06-2016

Viktor Wigmore	V.Wigmore@student.tudelft.nl	4279638
Magdalena Simidzioski	M.simidzioski-1@student.tudelft.nl	4383036
Maria Simidzioski	M.simidzioski@student.tudelft.nl	4381319
Matthijs Klaassen	M.klaassen@student.tudelft.nl	4273796
Wing Nguyen	t.n.nguyen@student.tudelft.nl	4287118

Table of Contents

1. Introduction

1.1. Design goals

During the project, several design goals are to be taken into account. Although some of these goals may already be discussed in the former part of the report, all goals will be stated explicitly underneath.

- Performance
  One of the hardware components in the system is the VR HMD. This means that high performance is a necessity. Without a frame-rate of at least 90 FPS, the user can experience dizziness and nausea.

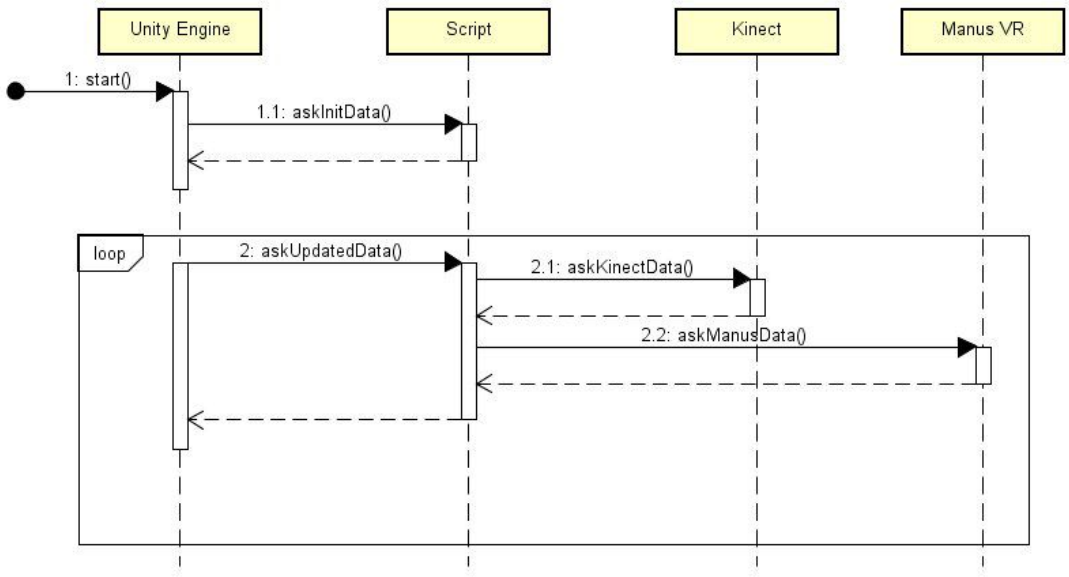
- Reality
  In order to treat patients successfully, objects in the virtual world should behave as they would in real world situations. This means for example that objects cannot fly through other objects. If it would be possible, patients may not be helped at all because of the unreal situations.

- Scalability
  The setting in which additional functionality is developed is not the scenario in which it eventually will be used. The development environment is a less computationally exhaustive setting, therefore it is crucial that the performance of the system is still acceptable in the final environment.

2. Software architecture

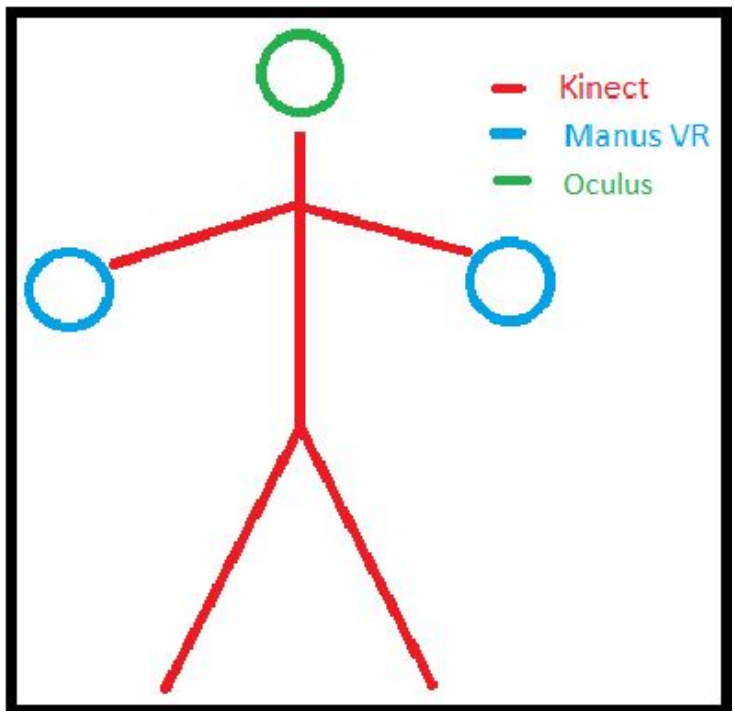
2.1. Interaction between Scripts and the Unity Engine

The system created so far uses the Kinect and Manus VR hardware. For every frame, the unity engine calls the update function in the different scripts for determining the new positions of the component in the armature.



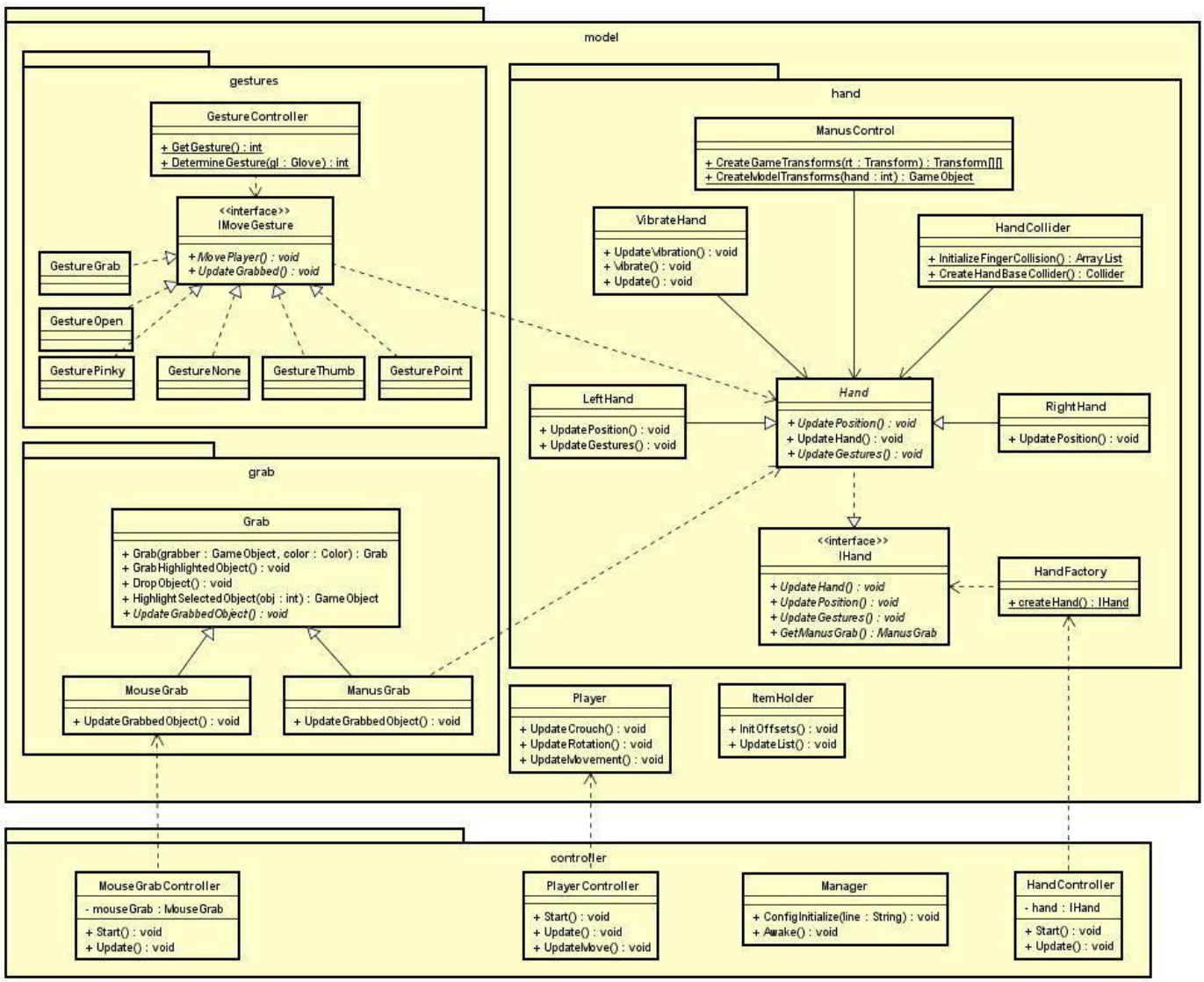
2.2. Hardware responsibilities

The above model describes how the different components relate to one another. There are 3 input devices; the Manus VR, the Kinect and the Oculus. Each of them control a different part of the simulated body. In the image below a simple stick figure provides an overview of the body parts and their controllers.



2.3. System design visualisation

The UML underneath describes how the different classes in the system work together. We have two different grabbing classes. One is for grabbing objects with the Manus, and the other one for grabbing objects with a simple mouse click. And we have an apart class for controlling the player. The hand functions are implemented by using the Inversion of Control pattern. And the Gesture controls are implemented by using the Strategy pattern. Both the GestureController and the HandFactory class are factories which decide which instance has to be created. The Manager class is a singleton since it contains global variables read from a config file.



3. Solutions

The client's primary wish was a realistic simulation, specifically a simulation where in a user could grab virtual objects.

The simulation uses a virtual hand that is connected to a real life glove, and the virtual hand moves like the real life glove. To control the virtual body we use the kinect, a camera device that tracks the user's body and maps the movement to the virtual body. The user can easily move in the supermarket simulation and add the picked up groceries to the basket. The groceries move smoothly with the basket and can be removed from it. When the user is at a predetermined distance form the desired item, the item is highlighted and the user can pick it up. The items have physics properties so the user can easily push away or throw an undesired item.

3.1. Unity

The simulation is being run in Unity, so in order to have a full understanding of our product, it is important to understand the relevant features of Unity that are being used.

3.2. GameObjects

All objects in Unity are GameObjects. These contain information called *components* about the in-game objects such as *Transform* information, *colliders*, *scripts*. In order to control the environment, GameObjects are used.

3.3. Components

Components define the behavior of GameObjects in the Unity Engine. They are always linked to a certain game object.

3.4. Transform

The transform property is used to change the rotation and position of an object. All objects have a transform property. The transform property can be manipulated by changing the properties in the Inspector.

3.5. Physics

There are 2D and 3D physics properties available in unity. We mostly use the 3D physics which consists of several options such as *colliders, rigidbodies* and *physics materials.*

3.6. Colliders

Colliders are used for physical collisions. They define an invisible shape around the object.
Colliders added to objects without a rigidbody are static colliders because they do not move. Colliders added to objects with a rigid body are called dynamic colliders. Static and dynamic colliders can also interact with each other.

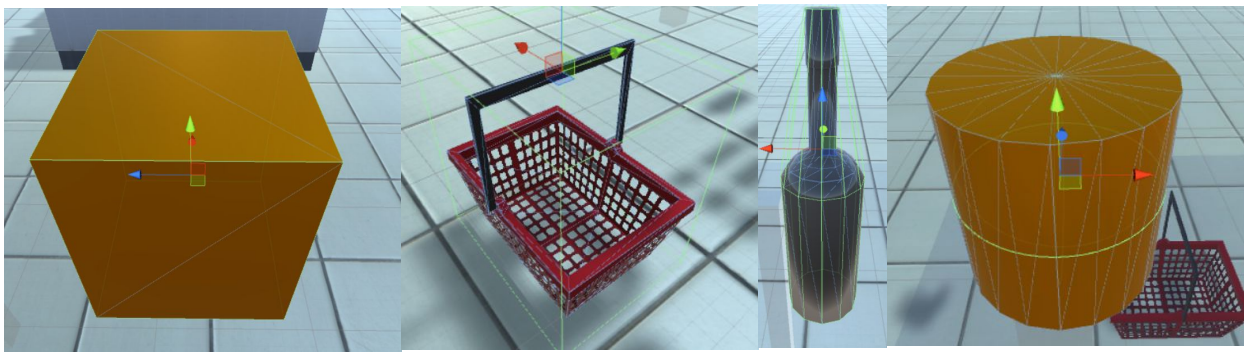
3.7. Scripts

Scripts are created in unity and they can be used to respond to a certain user's input and trigger game events. Each time a script is attached to a GameObject, a new instance of the object is created.

3.8. Tags

In Unity, tags have a purpose of identifying GameObjects. They are words which have a connection to a certain GameObject (a GameObject can have only one tag connected to it). This is a very useful feature since it saves a lot of time when working with more GameObjects.

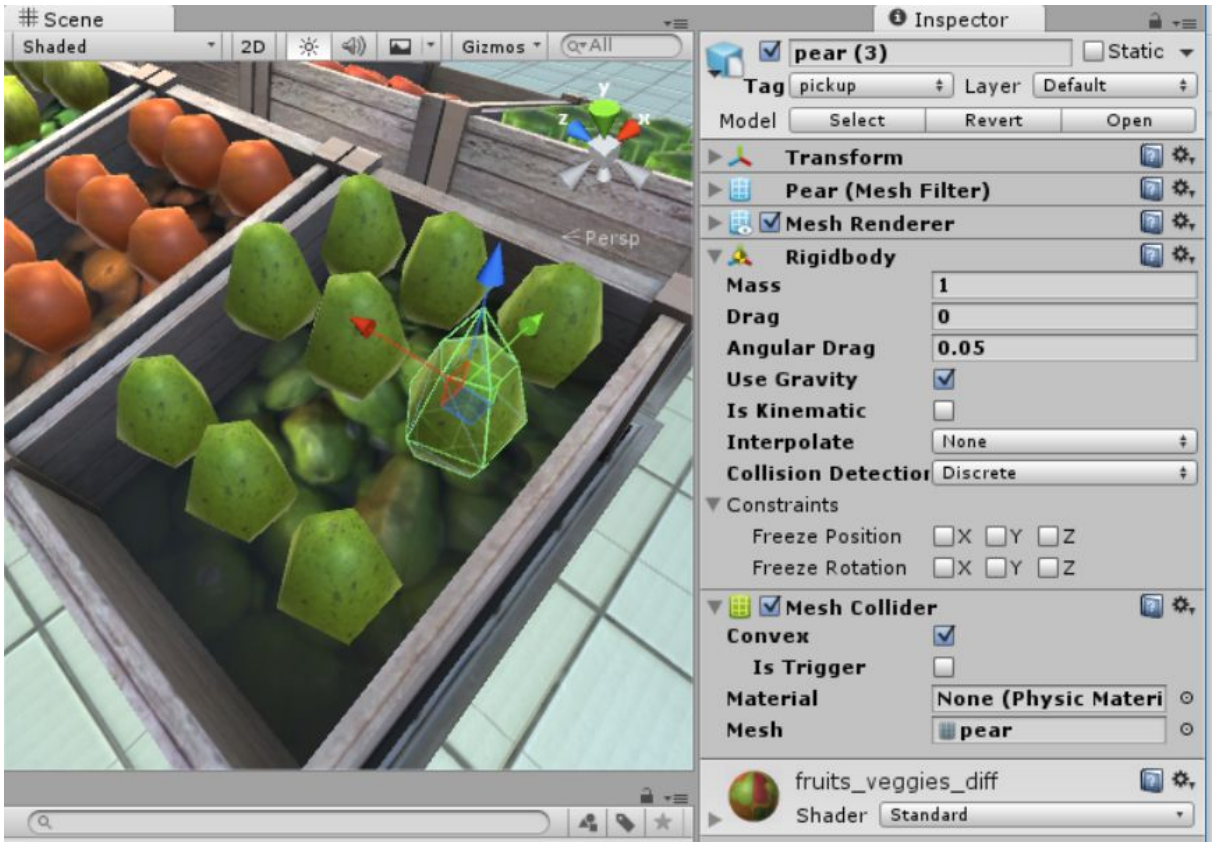
3.9. Objects



There are many types of objects in the simulation. In order to keep the environment as stable as possible, large layout influencing objects such as walls, floor, ceiling are static objects while the bottles, vegetables, basket and other groceries are dynamic objects which can be picked up, moved around and dropped.

Object requirements
Objects that are grabbable have to be tagged as **pickup** in the Unity Editor. This makes sure that you can only grab objects that are intended to be grabbed.

Pickup objects also have to contain the Physics Component "Rigid Body" and have to contain a Collider Component.

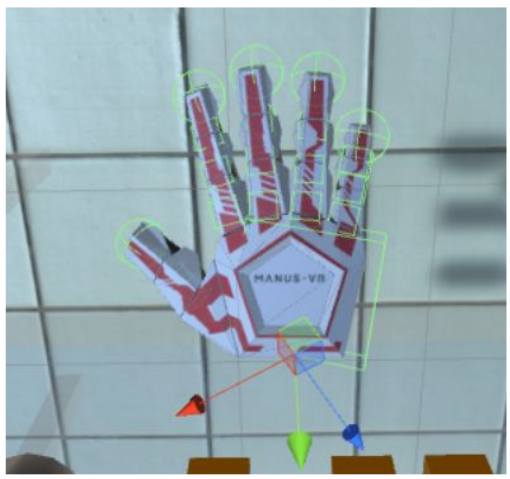


3.10. The Hand

Positioning

For our product, we used the 3D models and parts of the scripts of the Manus-Unity SDK readily available on GitHub. For the right positioning of the hand, we use the wrist's position of the Kinect model and translate the hand accordingly, to have the hand follow the wrist. The hand also is oriented such that it correctly reflects the user's real life hand.

Colliders



In order to enable grabbing with the Manus-VR, we have implemented methods to detect collisions between the fingers of the hand and the virtual objects. For each part of the hand, a bounding box is created. For the fingertips, spherical bounding boxes are created to provide better collision with other objects. These colliders are saved in an arraylist so that they can be referenced later whenever a collision occurs. This can be used to stop the bending of the 3D model so that fingers don't go through the objects, making the simulation less believable. Our implementation of this concept does work most of

the time, but we're not really satisfied as sometimes the hands do go through because of collisions not being detected at certain times by the Unity Engine.

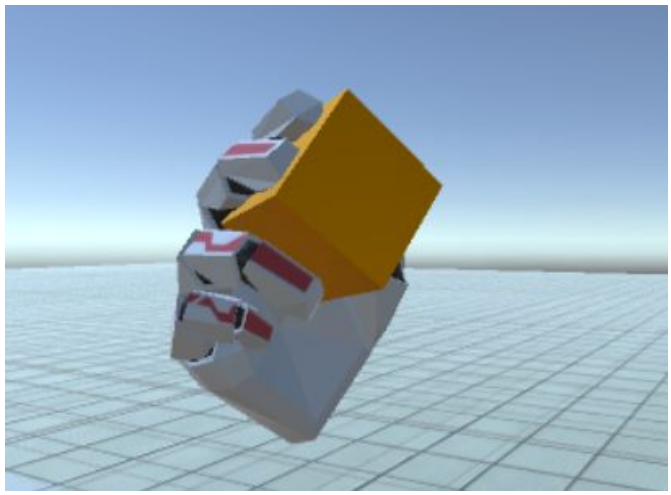
Gestures

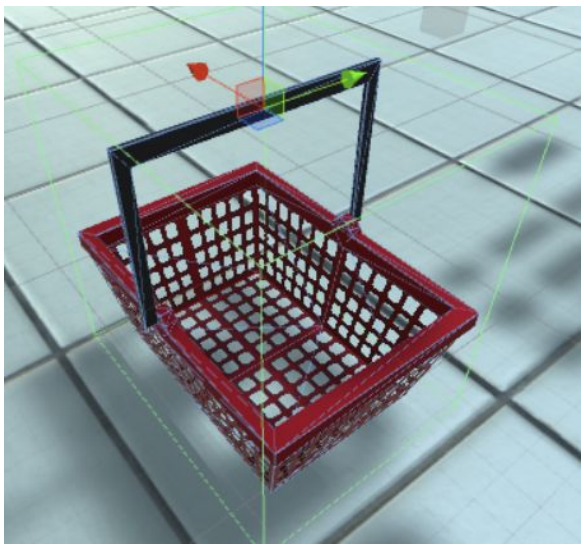
In order to make it easier to control the virtual body (without the use of controllers like a mouse or keyboard) we implemented gestures. Gestures look at what fingers are stretched or bend and according to this move the user' virtual body forward or backward, or rotate the user.

3.11. Grabbing

Detecting collisions

To detect collisions, the colliders of the fingertips of the hands are used. An object is grabbed if the thumb and at least one other finger are touching it. If the **open** gesture is triggered, the object is then dropped. Whenever a collision is detected by the Unity Engine, a function *OnCollisionEnter()* is called which calls the *Grab* methods. The selected object can then be translated and rotated accordingly at each update (this happens every frame).





3.12. Shopping Basket

To make it possible for the patient to collect all the items he wants to buy we have a shopping basket in the scene. The patient can hold the shopping basket in one hand and put items in it with the other hand. Like other pickup objects, the shopping basket has to be tagged as **pickup**, contain rigid body component and contain a collider component.

4. Running the project

Requirements:

- Visual Studio 2013.
- Unity 5.3.4 P1 or newer stable releases.

The project needs to be downloaded or cloned from the repository. Then the project needs to be imported into Unity , specifically the folder “Marketsim”. To run the project the play button in the editor of Unity needs to be pressed. Finally, to test the simulation with using keyboard and mouse, the keys (direction keys) are used for moving around, the mouse is used to look around all directions and to select an object (bottles, cubes or the basket) the user needs to stand near object and left-click it. To jump space needs to be pressed, to crouch down left ctrl needs to be pressed and to stand up left shift needs to be pressed.

4.1. Configuration File

We created a configuration file to make it easy to enable certain parts of the simulation (for instance the vibration). The configuration file is found in the location:

<Repository> \MarketSim\Assets\src\config.gryffindor

```
################################################
#############CONFIGURATION FILE#################
################################################
#                    __          _  _          #
#|_| _     _  _     /__ __ \/_|__|_ o __  _| _  __#
#| |(_)|_|_> (/_   \_| |  /  |  |  | | |(_|(_) | #
################################################

# Lines starting with # are ignored.
# Format: <attribute> = <val>
# Not case sensitive.

################## [GENERAL] ##################
# HIGHLIGHTON: decides whether an object will be highlighted
# GESTUREMOVEMENTON: control the player character controller with the left glove
# MKBONLY: Controlling the simulation without VR hardware
###############################################

HIGHLIGHTON             = True
GESTUREMOVEMENTON       = FALSE
MKBONLY                 = true

################## [GRABBING] ##################
# PROXDIST: the 'grabber' (hand/camera) has to be within this distance to grab
#       Default: 2.0 (float)
# THROWFORCE: force multiplied with throw vector decides how far objects are thrown
#       Default: 500 (float)
# ENABLEVIBRATION: enables haptic feedback
# VIBRATIONFORCE: the force of the haptic feedback [0, 1]
# VIBRATIONTIME: the time of the haptic feedback in ms.
# GLOVEORIENTATION: used to align the glove in the z axis -1 or 1
###############################################

PROXDIST                = 0.5
THROWFORCE              = 500

ENABLEVIBRATION         = true
VIBRATIONFORCE          = 0.1
VIBRATIONTIME           = 100

GLOVEFORWARD            = -1
```