University of Szeged

Faculty of Science and Informatics

Implementation of a Virtual Reality-based garden simulation

Bachelor's thesis

Author:
Bársony Daniella
BSc in
Computer Science

Supervisor:
Dr. Varga László Gábor
senior lecturer

Szeged
2019

Contents

Task proposal

The task was to implement a virtual environment where the user can tend to a garden in his own home. Part of the task was to choose a VR device (Google cardboard, HTC Vive, Oculus Rift, etc.) which is capable of visualizing a virtual world. With the chosen device then implement the game itself, which has the features stated below:

- The locale of the game is a garden, which the player can move around in.
- There are plants which reside in the garden.
- Plants in the garden will follow the rules of nature even if not exactly as in reality (For example: growing, they can dry out if the water level is not sufficient, fruits can grow on them, etc.)
- In the garden it is possible to have some interaction with the plants, and tend to them with the given tools.

Summary

• **Denomination of the topic:**
Implementation of a garden that resides in a virtual world

• **Wording of the given task:**
The task is to implement a virtual garden, where players can tend to the plants in 3D space. This includes watering, pruning, using pesticide if the plant gets infected, using fertilizer on the ground. In the case when our plant is happy, which means that the water is sufficient, the ground is fertilized enough and that it is not infected, it will grow into a full-blown plant, break into blossom, and bear fruit.

• **Solution:**
I made a framework for the plants, that made it easier to manipulate them through. I designed a basic User Interface for the player to get and delete plants. I implemented some basic game logic for the garden itself.

• **Devices and methods used:**
The device of my choosing was the HTC Vive. The platform that I used was Unity version 2018.2.10f1. As for the Integrated Development Environment(IDE), I chose Visual Studio Code. C# as the programming language. Furthermore I used 2 plugins from GitHub named VRTK and SteamVR for connecting to the Vive device and managing the VR environment.

• **Accomplishments:**
I finished the game as described. There are x different types of plants, which the player can tend to. X number of plots where the player can place a plant of our liking then tend to them.

• **Keywords:**
Unity, 3D, Virtual Reality, Garden

Magyar nyelvű összefoglaló (Hungarian content summary)

A Virtuális valóság egy olyan élmény, amit egy számítógép generál és egy teljesen szimulált környezetben van. Nem összekeverendő a kiterjesztett valósággal, ami a már létező valósághoz ad hozzá. Egy harmadik lehetőség, a szimulált valóság, ami hasonlít a virtuális valósághoz, de a felhasználó nem tudja megkülönböztetni a szimulációt a valóságtól. Erre láthattunk egy példát az ikonikus Mátrix című filmben, ahol az emberek nem tudták, hogy az egész életük egy szimuláció.

- **VR technológia:**

Ebben a részben a VR történelméről van szó. Kezdve az 1800as évektől, Robert Barker ötletével. Le szerette volna festeni Edinburgh városát egy félkör ív alakú festményen. Először röhelyesnek gondolták ezt az módot, később egy teljes kör-ívet alkotó képet festett, amit elnevezett panorámának.
1838-ban Charles Wheatstone publikált egy tanulmányt, miszerint egy emberi a jobb és a bal szem által látott 2-Dimenziós képből alkot egy 3-Dimenziós rekonstrukciót és így érzékeli a teret. Ezzel a technikával dolgoznak a VR szemüvegek is mostanában, külön képet vetítenek a jobb és a bal szemnek, amik egy picit el vannak tolva, így térhatást érnek el.
Sok kísérlet volt még ezután. Többek között egy repülés szimulátor, amivel a második világháború idején több mint 500.000 pilótát képeztek ki.Ez nem csak képet vetített, hanem imitálta a turbulenciát is motorok segítségével, amik mozgatták a kabint. Morton Heilig is kitalálta a Sensorama-t, ami szintén nyújtott mást is a kép mellett. Ide tartozott a 3D hang, ventillátor és egy szag generátor is. Erre az eszközre több kisfilm is készült, amit ugyúgy Morton Heilig csinált.
Az első HMD (fejen viselt kijelző) ugyanúgy Morton Heilig ötlete alapján látott napvilágot. Később ez alapján Philco vállalatnál két mérnök elkészített egy olyan HMD-t, ami követi is az ember fejmozgását.
Újabb nagy név ezen a területen, Ivan Sutherland, aki letette az alapkövét a virtuális

valóságnak 1965-ben, azzal, hogy megfogalmazta mi az irányelve ennek az ágazatnak. Ebben szerepelt egy olyan kijelző, amit nézve a felhasználó nem tudja, hogy valóság vagy szimuláció van előtte. Továbbá egy olyan virtuális világ, amit a 3-Dimenziós kijelzőn keresztül látunk, valóságos hangzás és interakcióba lehet lépni az ottani világgal. Mai napig ezt a koncepciót követi ez az ágazat.
1987-ben a VR-t felkapta az ipar, a VPL (Visual Programming Lab) több eszközt is kifejlesztett, ami ehhez tartozott, például a DataGlove és egy HMD. Ők voltak az elsők, akik el is adták ezeket.
1991 körül próbáltak elérni nagyobb közönséget ezzel a technológiával és játéktermeket csináltak többjátékos módban játszható gépekkel, amik VR-t használtak. Ezekben valós időben történtek az események és így tudtak vele játszani a játékosok.
A fűnyíró ember egy 1992-es film, ahol még több embert próbáltak elérni a virtuális valóság ötletével. A VPL által kifejlesztett eszközöket használták a filmben is és inspirációt a filmhez is a VPl történetéből merített a rendező, mint ezt bevallotta később.
A SEGA látott fantáziát ebben a technológiában és megpróbálta kifejleszteni a sajátját az akkori konzoljukhoz, amit Genesisnek nevezett el. Be is mutatta 1993-ban a Consumer Electronics Show-n. Sajnos ez a projekt nem jutott túl a prototípuson, pedig 4 játékot fejlesztettek is rá.
Pár évvel később a Nintendo is kapott az alkalmon, és kifejlesztette a Virtual Boy-t. Ez az eszköz ki is került a forgalomba Japánban és Észak Amerikában 180 dollárért, így már elérhető volt az emberek számára is a technológia. A Virtual Boy sajnos nem volt kényelmes így nem sokkal később a cég beszüntette a gyártását a szemüvegnek.

Source: `https://www.vive.com/us/product/vive-virtual-reality-system/`

Figure 1: HTC Vive

A jelenben a HTC Vive, Oculus Rift, Playstation VR és a Google Cardboard uralja a piacot.
Az Oculus-t 2010-ben Palmer Lucky kifejlesztette az első prototísusát az Oculus Rift-

nek, majd 2012-ben elindítottak egy Kickstarter kampányt a Rift fejlesztésére. Később a Facebook felvásárolta őket. 2 PenTile OLED kijelző található benne, 1080x1200-as felbontással kijelzőnként, 90Hz-es képfrissítéssel és 110 fokos látószöggel. A mozgáson kívül a fej mozgását is képes lekövetni, továbbá beépített fülhallgatót is tartalmaznak.
További headset-ek, például a Playstation VR, amit a Sony kifejezetten a Playstation kiegészítőjeként fejlesztett ki. A felbontása kisebb, mint a másik kettőé. 960x1080-as felbontást ad szemenként. A látószög is kisebb, mégpedig 100 fok. Viszont ez a headset a legkényelmesebb a 3 piacvezető közül.űű A HTC Vive, a HTc és a Valve közös projektje, aminek a fejlesztői változatát 2015-ben jelentették be. A felhasználói változat 1026-ban látott napvilágot. Ehhez a HMD-hez jár kettő falraszerelhető egység is, amit Lighthouse-nak hívnak. Ezek segítségével képes követni a HMD-t és a kontrollereket is. Ez a headset 90Hz-es képfrissítési rátával és 110 fokos betekintési szöggel rendelkezik. A headset-ben kettő OLED panel található, szemenként egy, 1800x1200as felbontással. Ebből a szempontból teljesen ugyanolyan, mint az Oculus Rift. amiben több, az a headset elején található kamera, ami a biztonságosabbá tételében segít. Előre konfigurált játékterünk van, aminek ha a széléhez érünk, akkor nem csak a hálót látjuk, ami jelzi a játéktér végét, de a kamera is be kapcsol, így el lehet kerülni baleseteket. Továbbá található benne G-szenzor, ami a gyorsulást figyeli és egy gyroscope, amivel az orientációját lehet meghatározni. Továbbá egy közelség érzékelő is. A kontrollereken sokféle gomb található. Két állású trigger, track pad, oldalán is található gomb. Közel 6 órás üzemidővel rendelkeznek. A kontrollerek elején 24 infravörös érzékelő található meg.
Végül a Google Cardboard, ami egy alsó kategóriás "készülék", ami a felhasználónak csak a telefonját használja semmilyen követés nincs benne. A telefonra le lehet tölteni különböző applikációkat, amik egy sztereoszkópikus képet vetítenek.

- **Unity**

Ebben a fejezetben a Unity-t ismertettem. A Unity egy 2D és 3D játékok fejlesztésére használt játékmotor. A játékmotorok a játék fejlesztőket segítik egy játék elkészítésében. A Unity egy platformfüggetlen és valósidejű játékmotor. 2005-ben került kiadásra. Akkoriban még JavaScript-ben is lehetett fejleszteni hozzá, ami mára már lemorzsolódott és C# lett a támogatott script nyelv. Maga a motor IL2CPP-ben íródott, ami valós C++ teljesítményre képes. Továbbá van egy asset store, ahol különböző játékfejlesztést segítő eszközök találhatóak meg. Ilyenek például az előre elkészített játékelemek.
Pár fontosabb része a Unitynek:

- A Tranform egy olyan komponens, ami minden GameObjecten megtalálható. Ez tárolja el az adott objektum méretét, rotációját, és a térben való elhelyezkedését. Abban az esetben, ha a hierarchiában felül található meg, tehát nincs szülője, akkor a világhoz képest nézi az elhelyezkedést. Azonban, ha van neki, akkor a szülőhöz képest hasonlítja.

- A Scene-ben található meg a környezet, a menü. Ebben a nézetben látszik, hogyan is vannak elhelyezve az objektumok és akár igazíthatóak is. Script segítségével váltogatni is lehet közöttük.

- A Game Object a legfontosabb része a Unity Editor-nak. Minden egyes objektum egy GameObject, például a fények, a kamera és így tovább. Többet egybe lehet építeni, amíg megkreálható bármilyen GameObject.

- A komponensek azok az elemek, amik segítségével a GameObjectek személyre szabhatóak. Nagyon sok különböző komponens van, amiket hozzá lehet fűzni egy GameObjecthez. Például, a Canvas komponens segítségével felhasználói felületet lehet létrehozni.

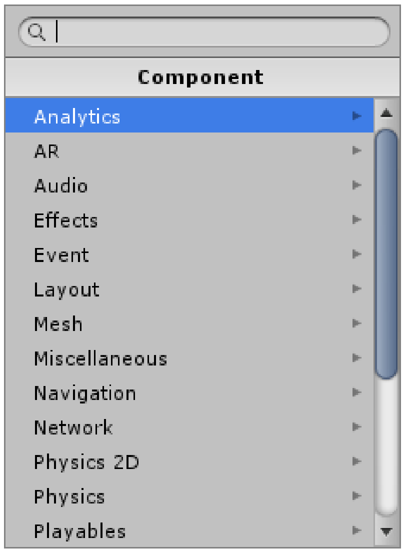


Figure 2: Komponensek

- A scriptek azok a fájlok, amikkel meg lehet határozni a játék logikáját és módosítani a segítségével a GameObjecteket. A Unity ad egy API-t, aminek a felhasználásával el lehet érni, amit a fejleszető szeretne. Továbbá létre hozhatóak saját komponensek , ami a Unity Editorban is látható és módosítható a script átírása nélkül.

- A prefab-ek olyan GameObjectek, amiket már személyre szabtunk script-ekkel, komponensekkel majd behúztuk a project mappába, ezzel elmentve minden beállításával együtt. Ez hasznos lehet tölténykenél, így egyszerűbben tudjuk létrehozni őket snélkül, hogy minden alkalommal újra kéne építeni az adott GameObject-et.

Használtam kettő könyvtárat is, mégpedig a VRTK-t és a SteamVR-t. VRTK megtalálható az asset store-ban és a GitHubon is, attól függ a fejlesztő melyik verziót szeretné. Ez a könyvtár olyan script-eket, prefab-eket és példa scene-ket tartalmaz, amik megkönnyítik a fejlesztő életét, ha az VR játékot szeretne készíteni. A SteamVR a kontrollek érzékelésével és renderelésével foglalkozik. Külön telepítést nem igényel, mert a Steammel együtt elindul, ha érzékeli, hogy egy VR headset van csatlakoztatva.

- **Implementáció**

A fő csapásirány egy olyan játék volt, mint például a Stardew Valley, ahol a játékosnak egy kertje van, amivel foglalkozni tud majd terem. Egy hasonlót szerettem volna VR-ben is létrehozni. Tehát mi is kellene ehhez? Először is, hogy mi kell egy növénynek úgy általában az életben maradáshoz? Víz, tápanyagok és, hogy ne lepjék el a kártékony rovarok. Ennek a mentjén indultam el én is.
A játékmenet annyit tartalmaz, hogy a játékos a játék elején tud növényeket vásárolni, vagy a játék közben azoktól megválni. Miután megvásárolt egy növényt az egy csupasz ág lesz, amit locsolnia kell és megfelelő tápanyaghoz juttatni. Abban az esetben, ha ez sikerül, akkor a növény meg fog nőni és később kivirágzik és teremni fog. Lehetséges az is, hogy nem tudja ezeket teljesíteni a játékos és akkor a növény elkezd elrohadni.
A játék főbb elemei a következők:

- **Kontrollerek,** amik különböző VRTK scriptek segítségével képesek interakcióba lépni a játékelemekkel, például a szerszámokkal és a felhasználói felületekkel, továbbá lehetővé teszik, hogy a játékos tudjon teleportálni a játéktéren.

- **Land,** a föld GameObject, ami összeköti a Pole-t és a növényt, ami megtalálható az adott földben. Továbbá ez tárolja a víz és tápanyag szintet és, hogy fertőzött e bogarak által a növény. A Start nevű függvényen keresztül hívja meg a függvényeket, amik gondoskodnak arról, hogy száradjon a talaj és fogyjon a tápanyag is.

- **Pole,** az az objektum, ami minden föld mellett megtalálható. Amikor a játékos hozzá érinti a kontrollert a tetején található gömbhöz, akkor az felhoz egy felhasználó felületet. Attól függően, hogy van e az adott földben növény hozza fel a felületet, hogy eladjunk vagy vegyünk egy újabb növényt.

- **Növények,** a fő elemei a játéknak. Mindegyik előre készített GameObject, tartalmaz egy palátát és egy felnőtt fát. Ezt a kettőt váltja le a script ami rá van rakva, ha bizonyos feltételeknek megfelel a növény. Mondhatni úgyis, hogy megnő. Itt is történik a kivirágzás és a termés kezelése.

- **Eszközök,** azon GameObjectek, amiket arra használhat a játékos, hogy interakcióba lépjen a növénnyel. Öntözze, visszavágja, permetezze és tápanyagot adjon neki. Egy VRTK script az alapja az összes eszköznek, amit felül kellet definiálnom, és testeszabnom külön-külön.

Introduction

In this thesis I will introduce the Virtual Reality technology. Starting with some history that helped humanity reach what we call Virtual Reality nowadays. All the way from panoramic paintings, through movies playing with this concept, until we reached HTC Vive.
There are many video games out there for VR, but most of those are horror themed, which might not be favorable for everyone, so I took another approach with this game, and ended up making a small garden. I will introduce the technologies used while implementing my game. Furthermore the environment used to develop the game itself and how it is built.

Chapter 1

VR technology

Virtual reality is an experience that is generated by a computer, in a simulated environment. It is not to be confused with augmented reality that adds nonexistent things to the real world, neither to be confused with simulated reality that is a dream which is far away with today's technology. It would be a reality which is 100% simulated and can not be differentiated from the real world hence fooling the person in it to thinking that it is in fact reality. We've seen it before in movies(to mention one: The matrix).
In my humble opinion virtual reality could eventually lead up to the simulated one, even if it looks impossible to reach for now. Obviously the technology for it is still light-years away but it could be a beginning. If this is a good or a bad thing that's something that only time can tell.

1.1 Beginnings

Humans played with the thought of virtual reality and how to achieve it way before we had computers. Starting from the year of 1800 until now there were countless experiments. With some hiatus every now and then but from starting at 1965, development has really sped up. At that time it was Ivan Sutherland who set the basis of what virtual reality should be. That was the point when people got really interested and invested in this topic, researching it even more than before. [1]

1.1.1 Panoramic paintings

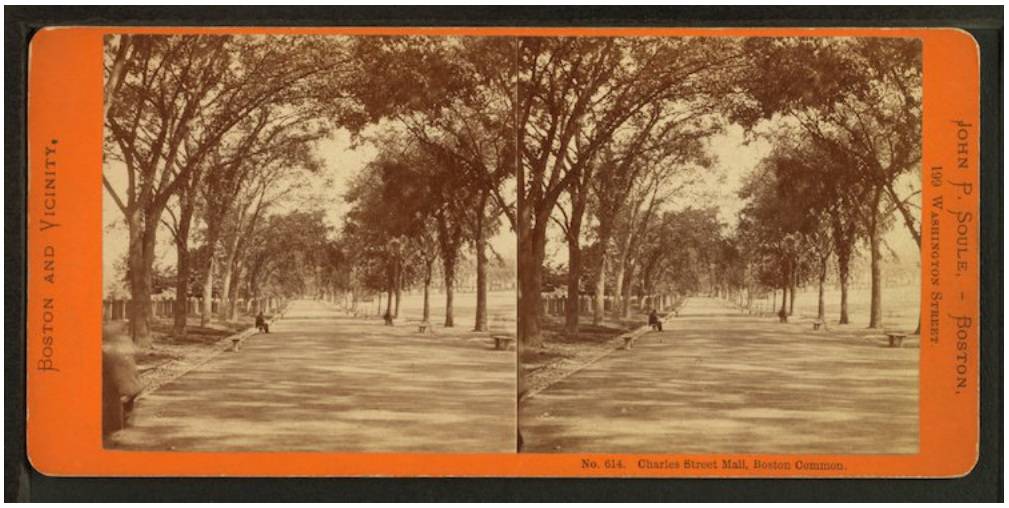
Robert Barker(1739 - 1806) who was an English painter with Irish ancestors, came up with the idea of painting Edinburghs city on a half-circle view painting. He then shown these pictures to Sir Joshua Reynolds, who said these pictures were not practical. Robert

[1] $https://www.vrs.org.uk/virtual-reality/history.html$

Barker did not give up and later on he made a full-circle view picture too and put it up for exhibition in Archer's Hall. The name of panorama came from the Greek "pan" which means "all" and "horama" meaning "view" in 1792. [2] [3]

1.1.2 Stereoscopic pictures

In 1838 Charles Wheatstone published a study, that our brain fuses two 2-Dimension pictures into a 3-Dimensional one. That's how it perceives depth and space. Later on in 1839, Willian Gruber, then in 1849 David Brewster made glasses for the stereoscopic pictures. These glasses could be familiar already from the cheaper alternatives to today's VR, just look at Google cardboard for example which just places the pictures of the left and right eye next to each other.



Source: `https://www.vrs.org.uk/virtual-reality/history.html`

Figure 1.1: Stereoscopic picture

1.1.3 Flight simulator

In 1929 Edward Link invented the "Link trainer", which wasn't only using picture to stimulate senses, but used motors to simulate turbulence and other disturbances that could occur mid flight. In the beginning only theme parks ordered the device from him. In the end, United States Army Air Corps bought 6 of these and trained over half a million people with it to be an airmen during World War II.

[2] $https://www.libraryireland.com/irishartists/robert-barker.php$

[3] $https://en.wikipedia.org/wiki/Robert_{B}arker_{(}painter)$

1.1.4 Morton Heiligs Sensorama

In the mid 50s Morton Heilig, director, came up with the idea of a theater which was named the Sensorama. He could build a prototype of it by 1962. His idea was to not only stimulate the eyes through pictures and ears through sound but to involve all our senses so the person could get really absorbed in the experience. It came with stereo-sound system, scent generating device, fans, vibrating chair and stereoscopic colored displays. They even made 6 short movies for the Sensorama including a Coca Cola advertisement that he shot and edited himself. He patented the Sensorama under US Patent #3,050,870.

1.1.5 Head Mounted Display/HMD

Another thing that we can thank Morton Heilig for,is that in 1960 he had a project called Telesphere mask. This was the first time that we've seen a Head Mounted Display(HMD). There was no interaction or movement tracking. The device had a 3-Dimensional wide-screen display and stereo-sound. He patented it under Patent #2,955,156.

1.1.6 Movement tracking HMD, Headsight

In 1961 two Philco Corporation employees who were engineers developed the first movement tracking HMD and called it Headsight. It had a display for each eye and magnetic movement tracking, that was linked to a closed circuit camera. It was designed, so that the army could observe dangerous situations with its help so this was a surveillance system. The motion tracking was that if the person rotated his head left or right the camera was following the movements.

1.1.7 The concept of Virtual Reality

Ivan Sutherland described a concept in 1965, that was about a display which would represent reality so closely that a normal human being will not be able to differentiate reality and simulation. In it was virtual world, that can be perceived through a 3D display and sounds effects that made the person feel more like it's reality and tactile feedback. A computer that is generating that world and keeping it up to date in real time. Moreover, the ability to interact with whatever is inside that reality in a way that does indeed feel realistic.
This was the concept that to this day people keep in mind when designing devices for virtual reality.

1.1.8 Sword of Damocles

Another thing by Ivan Sutherland and his student Bob Sproull, in 1968 was the first AR/VR HMD that was linked to a computer and not a camera. This device was huge and scary looking and it was too heavy to wear comfortably so they came up with the idea to suspend it from the ceiling. The user needed to be strapped into the device. The computer graphics were primitive wire-frame rooms and objects.

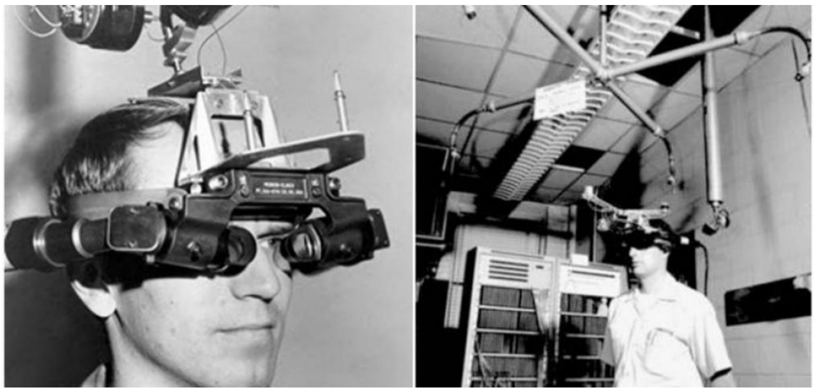


Figure 1.2: Sword of Damocles [1]

1.1.9 The VR is born

It was 1987, when the founder of VPL(Visual Programming Lab), Jaron Lanier came up with the term virtual reality. Through his company he developed many devices for this new field, like DataGlove and EyePhone HMD. They were the first company who sold VR goggles. This was a really big step for this field.

1.1.10 Virtual, multiplayer arcade machines

In 1991 VR could reach a wider audience, even if it wasn't available for home usage yet. They popped up in arcades. The players wore HMDs, that had 3D stereoscopic displays. The games themselves were running in real time, which meant that their latency was less than 50ms. There were even games where they could implement multiplayer mode, for example Dactyl Nightmare.

1.1.11 The Lawnmower Man

This movie came out in 1992 that introduced the concept of virtual reality to even more people. The movie was based on the work on Jaron Lanier who was the founder of VPL.

[1]Source: `https://vrroom.buzz/vr-news/guide-vr/sword-damocles-1st-head-mounted-display`

Doctor Angelo who was played by Pierce Brosnan a big name at the time, was playing with therapy that was based on virtual reality to help a mentally impaired boy. They used the devices that were made by VPL itself and the director Brett Leonard did admit that he drew inspiration for this movie from companies like VPL.

1.1.12 SEGA

SEGA announced a VR headset, named Genesis for its console in 1993, at the Consumer Electronics Show. This prototype was able to track the movements of the head, had stereo sound and had LCD displays built in it. The company wanted to release it but sadly they hit a wall that they could not overcome, so it never reached customers.
There were even 4 games developed just for this HMD. 2 years later even Nintendo tried to come up with something like this.

1.1.13 Nintendo Virtual Boy

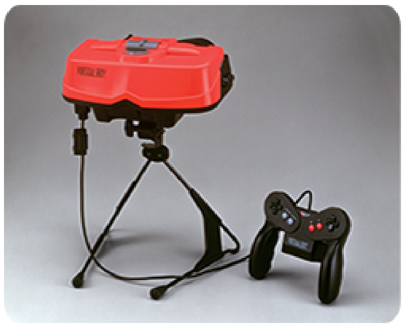


Figure 1.3: Virtual Boy [2]

It was a 3D video game console, which was advertised to be the first portable game console, that was capable of rendering true 3-Dimensional graphics. First came out in Japan then North-America for 180$. Despite the efforts of Nintendo, trying to lower its price, this project proved to be a failure. It was not comfortable and it had problems with the colors too. These were the causes that made the company stop manufacturing the said console.

[2]Source: https://www.nintendo.co.uk/Iwata-Asks/Iwata-Asks-Nintendo-3DS/Vol-1-And-That-s-How-the-Nintendo-3DS-Was-Made/2-Shigeru-Miyamoto-Talks-Virtual-Boy/2-Shigeru-Miyamoto-Talks-Virtual-Boy-229419.html

1.1.14 The Matrix

It was 1999 when Matrix the movie came out, which later on grew to be a cult film. In it the characters lived in a simulated world, which looked like just our everyday life, where almost no one knew that they lived their whole life in a simulated world. Even before this film there were ones who were playing with the idea of virtual or simulated worlds, just like Tron or The Lawnmower Man, but this one was such a big hit that it introduced the idea of virtual worlds to people.

1.2 Present

This generation is dominated by HTC Vive, Oculus Rift, Project Morpheus on the high-end, while the low-end has devices as the Google cardboard and other headsets that use ones phone as a display.

1.2.1 Oculus Rift

In 2010 Palmer Lucky designed the first prototype of the Oculus Rift, which was capable of tracking the movements of the head and having a 90 degree field of vision. 2012 they started a Kickstarter campaign for the development of Rift. Later on Facebook purchased the whole company. The Rift right now has 2 PenTile OLED displays, 1080x1200 resolution per eye, 90Hz refresh rate and 110 degree field of view.
It's not only capable of tracking the movements of the head but position too. Moreover, it has integrated speakers which provide a 3D audio effect. It has multiple accessories for example the controllers and sensors. The movement tracking sensor is named Constellation, which is not only capable of tracking the headset, but the accessories too. External infrared sensors are there to optically track compatible VR devices.
The Oculus supports third-party peripherals too, so it provides an API that people can use.

1.2.2 HTC Vive

Figure 1.4: HTC Vive [3]

By 2013 Valve joined in on the virtual reality business and they had a break through with low-persistence displays, that had almost no latency and smear-free. Oculus used this technology too and used it in later developments.
HTC and Valve announced the development kit of HTC Vive in 2015.
This headset uses "base stations" for tracking which are called Lighthouse, that one can mount on walls and uses infrared light too. The consumer version came out in June 7th of 2016.
The **headset** has 90Hz refresh rate and 110 degree field of view. The device is packed with two OLED panels, one for each eye, that has 1800x1200 resolution for each of the displays just like the Oculus Rift. The headset comes with a front facing camera for safety reasons, so the user will not bump into moving or static objects in the room. It has a G-sensor in it, which tracks the acceleration and a gyroscope so it can determine the orientation, furthermore a proximity sensor too.
The **contollers** have different input methods. Dual stage trigger, track pad, grip buttons. It has a battery time of 6 hours. On the front of the controllers, across the rings there are 24 infrared sensors that the base stations use to detect the location of said devices. SteamVR tracking system is used to track the controller location to a fraction of a millimeter, with update rates of 250Hz to 1kHz.
Vive Tracker is an accessory used for motion tracking. They designed it, so the user can attach it to anything and the Lighthouses will detect them and track their movements.

1.2.3 Project Morpheus/Playstation VR

Project Morpheus was the code name for this project during development. It was released in October 2016 for the Playstation 4.
The latest release of the headset comes with OLED displays, a resolution of 960x1080 per eye, the refresh rate can be switched depending on the developers to native 90Hz, native 120Hz and there is a third, where the 60Hz would be displayed as a 120Hz using a motion interpolation technique(new frames are generated between existing ones). Field of view is only 100 degrees. As for tracking it has six-axis motion sensing system (three-axis gyroscope, three-axis accelerometer). [4]

1.2.4 Google Cardboard

Last there is the Google cardboard. This is one of the low-end "devices", as the name says it is a piece of cardboard, even though there are more stylish ones. All it does is holds

[3]Source: `https://www.vive.com/us/product/vive-virtual-reality-system/`
[4]`https://www.playstation.com/en-us/explore/playstation-vr/tech-specs`

the phone, on which an app can be downloaded. The result is a stereoscopic image with a wide field of view. Developers can use VR View, an expansion of the Cardboard SDK allowing developers to ember 360-degree VR content on a web page or in a mobile app on pc, android and iOS. The HTML and JavaScript code from web publishing VR content is open source and available on GitHub. [5]

[5] https://en.wikipedia.org/wiki/Google_Cardboard

Chapter 2

Unity

A game engine is a software-development environment designed for people to build video games[1]. The core functionality being rendering 2D and 3D graphics, physics and collision detection, sound, scripting and a few more. These tools are to help the developers by simplifying the development process. There are quite a few game engines out there and they use different primary programming languages like C++, C, Lua, C# and so on. The bigger ones being Unity, Unreal Engine and CryEngine.

2.1 What is Unity

Unity is a cross-platform real-time game engine developed by Unity Technologies which was announced first in June 2005. It used to support JavaScript as its programming language but now it's deprecated and only supports C#. Supported platforms are iOS, Android, Windows, Mac, Linux, WebGL, Playstation4, Xbox One, Nintendo 3DS and much more adding up to 25+. The engine is written in IL2CPP (Intermediate Language To C++), which provides native C++ performance. Unity has XR, which is support for VR and AR and MR development. This game engine support both 2D and 3D game development. There is an asset store, where the developers can either purchase or use free premade scenes, prefabs, models and so on that anyone could need in the future for development.

Starting with 2016 Unity started to offer cloud based services for the developers, which are: Unity Ads, Unity Analytics, Unity Certification, Unity Cloud Build, Unity Everyplay, Unity In app purchase, Unity Multiplayer, Unity Performance Reporting, Unity Collaborate and Unity Hub. [2] [3]

[1] `https://en.wikipedia.org/wiki/Game_engine`
[2] `https://unity3d.com/unity`
[3] `https://en.wikipedia.org/wiki/Unity_(game_engine)`

2.1.1 Transform

The Transform component determines the Position, Rotation, and Scale of each object in the scene. Every GameObject has a Transform. The game objects Transform values are measured relative to its parent element, in the case that there is no parent element, then it is measured in world space.
This component has quite a few elements we can use, it has properties and public methods that helps either manipulate the members mentioned above or we can perform various actions with their children or parents. The three basic members of a transform can be manipulated via the Unity Editors component window.
Position is stored in a Vector3, which is the representation of 3D vectors and points. Basically it is storing x, y, z coordinates. This struct has properties and methods too, so it can be manipulated through them. **Rotation** is represented as a Quaternion, which too has properties that should not be modified directly unless one knows how Quaternions work. **Scale**, is stored as a Vector3, one can either use `localScale` which is relative to the parent component or `lossyScale` which is the global scale of given object.

2.1.2 Scenes

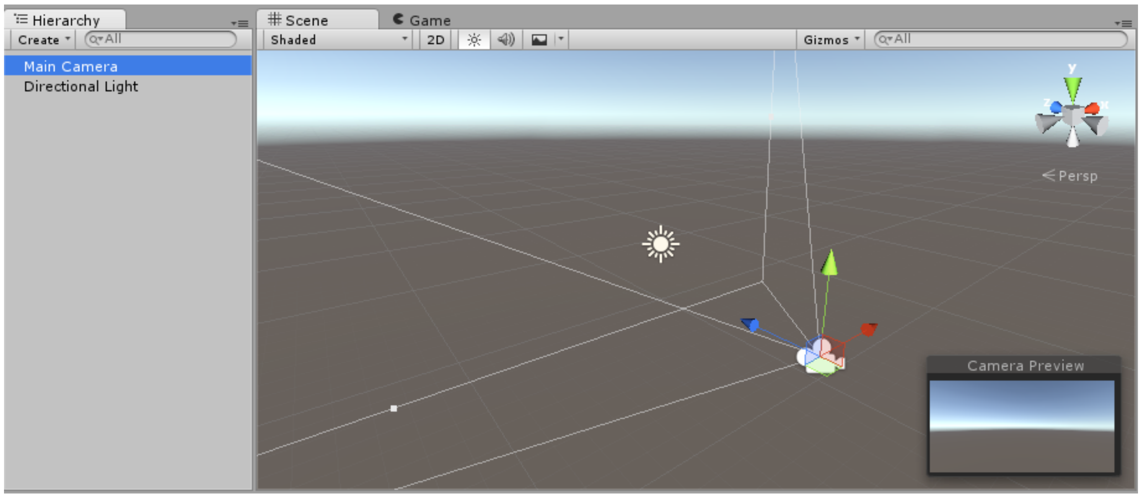


Figure 2.1: Unity Scene [4]

A scene contains the environment and the menu of a game similar to a different level. One can place their decorations, player character, User Interface and whatever they like helping to build the game in pieces. One can make multiple scenes and switch between them using the Unity API in their own scripts. Multiple scenes can be loaded too at the same time if need be.

[4]`https://docs.unity3d.com/Manual/CreatingScenes.html`

2.1.3 Game Objects

A GameObject is the most important concept in the Unity Editor. Every object in a game is a GameObject starting from characters to lights and cameras and so. They can't do anything on their own, one needs to give them properties. To give them these, one needs to add components to it. The different elements can be combined until they add up to the object that the developer is working towards. For example if one wants a Light object,a Light component needs to be attached to the GameObject. It can be said that a GameObject is a container for many different components.
GameObjects have transforms. The transform component determines the Position, Rotation and Scale of the given object in the scene. Every GameObject has it, as mentioned above. [5] [6]

2.1.4 Components

As mentioned above, GameObjects have components attached to them. These can bee seen in the inspector window. When a new GameObject is created a Transform component is attached to all of those, which is a must have. Components are the functional pieces of a GameObject. They can be added through the `Add Component` button at the bottom of the inspector. Components provide great flexibility. When attached to a GameObject, it has multiple values and properties that can be adjusted, while building the game.
Scripts are components too that are attached to GameObjects. Some of the components I have used:

- **Colliders:** This component is for defining the shape of an object for the purpose of physical collision.
- **Rigidbody:** These enable the GameObjects to act under the control of physics. Some of its properties are:
  - `Mass`, to set the mass of the object
  - `Drag`, to set the linear drag coefficient
  - `Use Gravity`, if the object should be afflicted by it
  - `Is Kinematic`, if the object would not be driven by the physics engine and can only be manipulated through its **Transform**
- **Fixed Joint:** This component restricts an object's movement to be dependent on another object.

[5]`https://docs.unity3d.com/Manual/GameObjects.html`
[6]`https://docs.unity3d.com/Manual/class-Transform.html`

- **Canvas:** Represents the abstract space in which the UI is laid out and rendered. All UI elements must be children of a GameObject which has this component attached to it.
  Render modes and their differences:
  - `Overlay`, the Canvas is scaled to fit the screen and then rendered directly without reference to the scene or the camera.
  - `Camera`, the Canvas is rendered as if it was drawn on a plane object set distance from the camera.
  - `World Space`, with this mode the Canvas is rendered as if it was drawn on a plane object too, the difference with Camera mode is that the plane does not need to face the camera itself. In my project i used World Space rendering for all the UI-s.

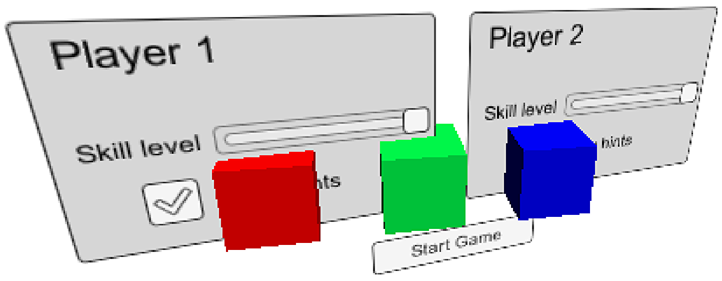


Figure 2.2: World Space mode for Canvas [7]

2.1.5 Scripts

The most important part of Unity are scripts, which hold the logic of the game itself, handle user input, manipulate with GameObjects, components, scenes and pretty much everything Unity has to offer can be modified by these. Unity does provide a well documented API, for the developers to use. These scripts should be written in C#. Not that long ago one could write their scripts in JavaScript too, but as of now it is deprecated.
After attaching a finished script to a GameObject, it can be seen in the Inspector window as a component. The developer can write his script in a way, that he can modify certain values through the Unity Editor without rewriting the script, thus fine tuning it. A good example could be, when there are certain monsters in a game, that have a health point property. The developer can fine tune how much health points should they have without opening the script and rewriting it every single time. Even better, they can be manipulated

[7]Source: `https://docs.unity3d.com/Manual/class-Canvas.html`

when the game is running.
When creating a script, one can use an IDE (Integrated Development Environment) of their choosing. As for myself, I have used Visual Studio Code and the default for Unity is MonoDevelop.

2.1.6 Prefabs

This system allows the developer to create, configure and store a set GameObject for later use with all the settings and changes made on it. Any edits made on the Prefab will be reflected to the instances of that given object. The prefabs can be nested inside other Prefabs to create complex hierarchies of objects that are easy to edit.
However the instances of a given Prefab don't need to be identical, small edits can be made on each of them that need to be different somehow and it won't be reflected on the parent. In case one wants to edit a prefab, it can be pulled to scene view, change whatever is needed, then press the `Apply` button which saves it to the Prefab and changes all instances of it accordingly.
A good way to use a prefab could be a player character that is the same in different scenes so it doesn't need to be remade from scratch or a weapon with projectiles and unique scripts attached to it that would be too much of a hassle to remake every time it is needed.

2.2 VRTK - Virtual Reality Toolkit SDK

This asset can be found either on the Asset store or on GitHub. Depending on which SteamVR version the developer is using, he might need to use the GitHub master version. This asset is a collection of useful scripts, prefabs and example scenes to aid building VR games, for example to help the player interact with GameObjects. After the developer imports the VRTK asset, it has a `VRTK_SDKManager` script that can be attached to a GameObject in the scene. It supports almost all VR devices. [8]

2.3 SteamVR

This plugin is made and maintained by Valve. They want to help developers so that they can develop games with one API that supports all the popular VR headsets. It manages three main things for the controllers:

1. Loading 3D models for them

[8] `https://vrtoolkit.readme.io`

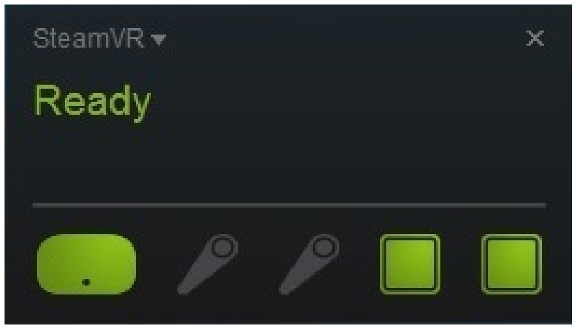


Figure 2.3: Steam VR UI

2. Handling the input from those
3. Estimating what the hands look like while using them

SteamVR has an Interaction System example to show the developer how to set up a world that can be interacted with. To use it the player needs to have SteamVR runtime installed, which can be download from Steam. This will be started up whenever we switch to VR mode. [9]

[9] `https://valvesoftware.github.io/steamvr_unity_plugin/`

Chapter 3

Overview and Implementation

As I have mentioned in my earlier chapter I chose Unity Engine to make my thesis. This chapter will describe the details of coding, getting familiar with SteamVR and VRTK Software Development Kits, testing and experimentation.

3.1 The main idea

I wanted to create a somewhat realistic garden which is to show the potential of a VR headset. The main reason why I decided to make a garden was because of games like Stardew Valley (which is a farming simulator where the player can grow their crops and tend to their farm animals) and I wanted to create an experience that was similar to it.

Figure 3.1: Stardew Valley[1]

[1]https://www.stardewvalley.net/

It was to be a place where the player can just turn off and have some fun with the plants. As for the structure I was aiming at making a framework that is easy to extend and use if more plants or tools are to be added in the future.

3.2 Basic needs of the plants

What are needed to grow a plant in real life? Basically water, sun, sufficient fertilization and to not get infected by insects. There are more than that, but those were the things I kept in mind when implementing my framework.

3.3 Flow of the game

In the beginning the player starts with a few empty lands and the tools needed. Each of them containing a fence in the back, a pole and basic UI. The player can buy a plant that they want, then a small tree will appear and the statistics will be shown on the UI. At that point the player should pick up the tools necessary and water, fertilize the tree. Information will be displayed always. Blue line showing the water level, green line showing the fertilization. In case the tree gets infected, the UI will show a small picture of a bug. That's when the player should grab the pesticide tool and get rid of the bugs. There are two ways this can go at this point.
One, when the player neglects the sprout and it starts to rot. Then it's time to grab the scissors and get ahead of the infection by cutting down the infected branches. If he is successful, then he can keep tending to it.
Two, when the player is taking good care of the plant and keeping infection away and well hydrated, fertilized, then it will grow into an adult tree. Even after it grew, still needs attention. At this point the tree can't rot anymore, but still needs to be cultivated. With some time it will bring flowers if everything is sufficient, then as the last step to a full-grown plant it will yield fruit.

3.4 Game Elements

3.4.1 Controllers

I did not need to take care of detecting the controllers, SteamVR does it for me. What i needed to take of is to make them be able to interact with game elements. So my Scene holds 3 different GameObjects , two of them are identical.

- **Right/Left Controller:** These are the two identical GameObjects. They are holding different scripts, that are from the VRTK library.
  - `Controller Events`: This is responsible for handling the different button inputs that the player can use. This component can be left at default or can be fine tuned through its settings, like how hard should the trigger be squeezed until it registers.
  - `Pointer`: The script handling the ray that can be used for different interactions. It needs a reference to the GameObject itself which the ray will shoot out of. Includes various settings to customize the pointer. There is a setting for enabling teleportation through the ray, so if the player points somewhere with it and presses a predefined button he will get teleported to that exact place.
  - `Straight Pointer Renderer`: This is rendering the pointer ray as the name suggest. This too has various settings that can be used to customize the ray, like maximum distance, valid/invalid collision color. I left these at default.
  - `UI Pointer`: This was needed so the player can interact with User Interface elements, through the pointer. For him to be able to do that, I attached `VRTK_UICanvas` script to every interactable Canvas. There are various settings here too, so I was able to set which button press should trigger interaction with the canvas.
  - `Interact Grab`: This script holds the methods to make the player able to interact with GameObjects. It has option to set the key which should be held for the interaction to trigger. I set it to grip press, which can be found on the side.
  - `Interact Touch`: This is a simple script that needs to be attached to every controller. It does not contain any settings.
  - `Interact Use`: With this script you can trigger a set action when holding an object. This plays a critical role in the tools that the player can tend to his plants with.
- **PlayAreaScripts:** All this GameObject hold is a script named `Basic Teleport`, which so the player can teleport. The settings in here include what color should it show for the split second when the player changes position.

3.4.2 Land

The base of the process is the Land.
There are several lands which hold the script to check for water, fertilization and the

insects themselves. When the game is started it calls the functions I created so that the land will dry out, will eat the fertilizer and call insects on itself and repeats calling it every now and then so the player will need to keep an eye out for the information shown in the User Interface for each land. For this I have used the `InvokeRepeating` (Invokes the method methodName in time seconds, then repeatedly every repeatRate seconds[2]) function of Unity which got placed inside the Start function, that runs when the player starts the game.

```
void Start() {
  /* Calls the dryOut function after 120 seconds
  ** then every 5 seconds */
  InvokeRepeating("dryOut", 120.0f, 5.0f);
}
```

The land prefab has a collider on it just like the tools so that is how you interact with it. Holds a reference to the plant Game Object which is null until you buy a plant through the User Interface. The User Interface shows the player information that given land is holding about water, fertilization and the insects. Water and fertilization can be an integer value between 0 and 100 while insects is a boolean whether is your plant infected or not which gets set in the `callInsects` method that will random a value and set it based on that. The land will use up the fertilizer and water with time. Decreasing their value by 10 every time the relevant function gets called.
The canvas attached to the `Land` holds a VRTK script, named `VRTK_UICanvas`. Its only purpose is to make the canvas interactable with the pointer.
This GameObject is the middleman connecting a Pole with the plant residing in the land.

3.4.3 Pole

The pole serves the sole purpose of buying and deleting your plant. When you touch the big red ball on the top of it with a controller it brings up the User Interface. For an object to be interactable I needed to use given VRTK scripts on it. The first script was the `VRTK_Interactable Object`, with its help I was able to edit whether the player can grab the given Game Object or use it or both. There are other options, like a button needs to be held for usage or that the given object can only be used when grabbed.
Given that a controller is touching the sphere on the top, it bring up said user interface for buying and deleting the plant. Interaction with the UI was implemented using buttons

[2]https://docs.unity3d.com/ScriptReference/MonoBehaviour.InvokeRepeating.html

and another VRTK script, named `VRTK_UI_Canvas`. It handles interaction with the controller pointer, so when the player, points on a button and pulls the trigger, a given `onClick()` function will be called. The script that handles the User Interface functions for the pole is called `PoleUIController`. Its properties are:

- `DeletePlantUI`, holds a reference to the canvas for deleting a plant. All it has is a "yes" or "no" button.
- `BuyPlantUI`, another reference to the canvas that has the option to buy one from the different plants.
- `Land`, is simply a reference to the land that is connected with the pole.
- `CherryTree/CatBush`, these two are the plant GameObjects that are in the game for now.

When the sphere on top of the pole is touched, then the script runs a check through the reference of land whether its empty or not. Depending on the answer, it brings up the User Interface to buy or sell. On the interface to buy a plant there are two different buttons, one for the cherry tree and one for the cat bush. In case the player clicks on one the relevant function gets called that instantiates the plant to the given transform, then sets it to the land and vice versa. That function is needed, so when the player interacts with the sphere again on top of the pole, it will know that the land is not empty.
When the player decides to sell the plant, the relevant User Interface will be brought up. Clicking on the sell button will call the `onClick()` function, that is set to `deletePlant()`. That method is responsible for destroying that instance of the game object, then setting the plant to null on the land GameObject.
I took care of the UI showing up in times it shouldn't have, by creating a `CloseUI()` function, which sets both canvas to inactive, and calling it after every interaction, so the player can't buy more than one plant to the same piece of land.

```
public void buyCatBush() {
  CloseUI();
  var bush = Instantiate(catBush, land.transform);
  bush.GetComponent<Grow>().setLand(land);
  land.setPlant(bush);
}
```

3.4.4 The plants

The plants are the GameObjects the player interacts with, tends to and grows. Each prefab for a given plant holds a version of the small tree and the grown one. When the player purchases a plant through the pole User Interface, the plant will only contain bare branches.

Small trees are identical for every plant. I created these by cutting the base model into branches and connecting them with `FixedJoints` and used the hierarchy to show the parent and child connection which was an important part for the rotting script.

Grown tree is different for each plant. In the first step when reaching this point, there will be a bare tree with green leafs, when the player keeps tending to the given plant it will break out into blossom. The blossoms are already placed on the model, just inactive. After the player reaches the blossom part and still takes care of it, with some time it will yield fruit. There is a set Transform for each tree, where the fruit Prefab will be instantiated. The fruits are kept as a prefab too, even though they don't hold a script on their own, I only set their scale on the prefab.

The **scripts** handling these actions and the communication with other GameObjects, is called `Grow`. That is the base for every plant. Its properties are:

- `Small`, is a GameObject that holds a reference to the small tree.
- `Big`, is the referenced prefab of the grown tree.
- `Blossom`, holds the prefab that will be active when the tree blooms.
- `Rottable`, is another GameObject, that is the starting point for the rotting function.
- `Prefab`, is a Transform, where the fruits will spawn.
- `HappyText`, is a GameObject too, which holds the Canvas with a text saying when the plant has sufficient water, fertilization and not infected by bugs.
- `Adult`, a boolean to check whether the tree is grown up or not.
- `Blossoming`, is another boolean value, that hold if the tree is blooming or not.
- `Land`, holds a Land GameObject. This was needed so the plant can communicate with the Land and Pole GameObjects.

All the methods I wrote gets called with `InvokeRepeat` in the Start function that runs when you instantiate a plant.

The first one it starts calling, is named **rotting**. That function has a check if the given

instance of the plant is happy and not an adult, in that case it gets the branch component of the rottable prefab I gave a reference to and keeps calling it while there are branches left. The rot function can be found in another script named `Branch`. It holds a boolean value if it infected and a Material, which is to show the infection to the player. All this script does is get the child branches, which is done through the hierarchy I built in the Unity Editor itself and builds a tree of those. Then starting from the outer branches it randomly starts changing the material on them. This is a simple implementation of a tree traversal on the branches. In case the infected branch does not get cut in a set amount of time then that branch will give the blight to its parent branches. If a branch rots, instantaneously its children will start to rot too. This way I was able to make sure that a sub-tree with decaying root will be infected completely. This whole process can be stopped via cutting said branches.
The check if the plant is **happy**, uses the reference to its `Land` GameObject and gets the water, fertilization, these should be higher than a set value, furthermore is not infected by insects, then it returns with true and sets the `HappyText` canvas to be active. This method gets used for every other function in this script.
The **grow** method runs a simple check if the instantiated plant is happy, if so then sets the `small` version to inactive, and the `big` one to active. Thus just changing the model for the plant and sets the value of adult to be true. The **blossom** method does the same, it sets the prefab to be visible and sets the value of `blossoming` to be true.
spawnFruit function is similar to the others, with first running a check if the given plant is happy and blooming, then creates a random value that is used when instantiating the fruits aren't falling from the exact same position all the time, which the Transform dictates.

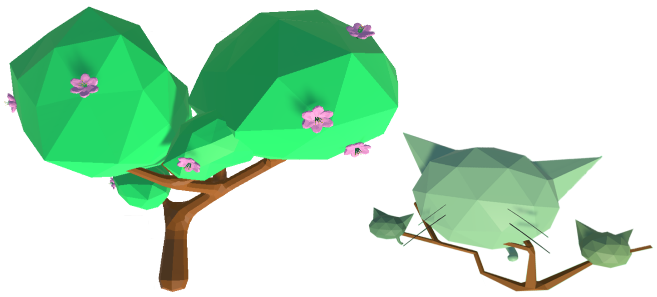


Figure 3.2: Available plants when they are in full blossom

3.4.5 Tools

The tools are the GameObjects that helps the gamer interact with plants and control the flow of game experience. I have created a tool for each of the needs that the plants have. Each tool is based on the same idea. There is the model itself, and a capsule GameObject

attached to it pointing the way it is facing and the Mesh Renderer component disabled, so it can't be seen, only exception being the scissors which doesn't need a Capsule. The attached Capsules hold a `RigidBody` component, a `Capsule Collider` and a custom script named `Held Object Collider` 5.1.
All it does is check if the collider attached to the tool is entering or exiting another collider. Uses for the mentioned script was checking collision with the Land so the player can interact with only the land he wants and to check if the scissors collide with branches. As for the other GameObject that is bundled together with the Capsule, that is the model itself. Those two GameObjects are connected with a FixedJoint component. The components that I attached to the base model are, various VRTK scripts and a collider so it will interact with its surroundings and lastly I added a custom script to it that controls the interactions of the given tool. These custom scripts are using the VRTK namespace and inherited from `VRTK InteractableObject` so I was able to use and override specific methods and got a half finished component with settings. These gave me the opportunity to set if the object is grabbable/usable or in fact both, whether the player needs to hold the button to use the object and that it will not fall from the players grasp when he teleports.

```
/* The VRTK method that gets the GameObject
** the player is interacting with */
public override void
StartUsing(VRTK_InteractUse usingObject) {}
```

The VRTK scripts that I attached are:

- **VRTK Fixed Joint Grab Attach** which is responsible for the grab action, so when the player grabs the tool it will connect to the controller through a FixedJoint. Settings include `Precision Grab`, if it is checked, then when the player grabs this GameObject, it will grab it with precision and pick it up at the particular point on the GameObject that the controller is touching. I turned this particular setting on, so the player can grab the said GameObject wherever.

- **VRTK Swap Controller Grab Action** is a script with no settings. All I needed to do was attach it to the tools, so the player is able to interact with both controllers.

The tools I have created are as follows:

- **Watercan** is responsible for keeping the water level of the plant sufficient. `WatercanHeldObject` is handling the interaction with the plant through land.

First it checks if the Capsule Collider of the capsule GameObject is inside the a collider which must be a land collider, then gets the Land GameObject itself through it. In the end it simply raises the water level by 30 if it is under 100, else it just stays at 100.

- **Fertilizer** is pledged to manage the fertilization level of the plant. `FertilizerHeldObject` is the scripts name. Basically it is the same as `WatercanHeldObject` script. It gets the attached capsule GameObject through the `HeldObjectCollider`, then runs the same checks as the `WatercanHeldObject` with the difference, that this scripts adds to the fertilization and not the water level.

- **Pesticide** is the one killing the insects on the plant if it gets infected. The base of this script is identical to the `WatercanHeldObject`, the only difference being that after passing the checks, it tests if the plant is indeed infected or not. In that case sets the value of insects to false.

- **Scissors** are somewhat unique compared to the other tools mentioned above. It checks if the target of the collision is with a branch, in that case it gets the GameObject then runs a check if there is a branch and it's infected. Only then can the player cut the branch by destroying the `FixedJoint` holding them together. which calls the DestroyBranch method, that waits for 5 seconds then destroys the branch GameObject itself.

Chapter 4

Conclusion

Working with VR wasn't as easy as I first visualized it would be. If I were to create another VR game, now I know where should I start and what I should/shouldn't do. The testing of the game itself was a real hassle, equipping the headset and turning on the controllers for every single tweak. Other than that it was a real fun project and I was able to learn a lot about this technology.
I learned a lot about Unity too. Sometimes it caused me hardships throughout the development of this project, but now I know never to update version mid development.
Even though I reached the end of this project, I can still see how I could have improved it. Including features to sell the fruit of plants and spending those to buy new ones, or anything alike.
All in all it was an amazing experience and I improved a lot during it.

Chapter 5

Attachments

5.1 Held Object Collider

```
public class HeldObjectCollider : MonoBehaviour {
bool inside = false;
Collision col;

void OnCollisionEnter(Collision collision){
  col = collision;
  inside = true;
}

void OnCollisionExit(Collision collision){
  col = collision;
  inside = false;
}

public bool getInside(){
  return inside;
}

public Collision getCollision(){
  return col;
}
}
```

Statement (Nyilatkozat)

Alulírott Bársony Daniella, Programtervező Informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Képfeldolgozás és Számítógépes Grafika Tanszékén készítettem, Programtervező Informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, May 1, 2019 . . . .

aláírás

Acknowledgements (Köszönetnyilvánítás)

Ezúton szeretnék köszönetet mondani **Dr. Varga László Gábornak**, amiért nagyon sok hasznos tanácsot adott és támogatott, hogy elkészüljön a program.

. . .

I would like to thank my supervisor **Dr. Varga László Gábor**, for the useful advice and encouragement that I got from my supervisor.

Bibliography

[1] https://www.vrs.org.uk/virtual-reality/history.html

[2] https://www.libraryireland.com/irishartists/robert-barker.php

[3] https://en.wikipedia.org/wiki/RobertBarker(painter)

[4] https://vrroom.buzz/vr-news/guide-vr/sword-damocles-1st-head-mounted-display

[5] https://www.vive.com/us/product/vive-virtual-reality-system/

[6] https://www.playstation.com/en-us/explore/playstation-vr/tech-specs/

[7] https://en.wikipedia.org/wiki/Google_Cardboard

[8] https://en.wikipedia.org/wiki/Game_engine

[9] https://unity3d.com/unity

[10] https://en.wikipedia.org/wiki/Unity_(game_engine)

[11] https://docs.unity3d.com/Manual/CreatingScenes.html

[12] https://docs.unity3d.com/Manual/GameObjects.html

[13] https://docs.unity3d.com/Manual/class-Transform.html

[14] https://docs.unity3d.com/Manual/class-Canvas.html

[15] https://vrtoolkit.readme.io/

[16] https://valvesoftware.github.io/steamvr_unity_plugin/

[17] https://docs.unity3d.com/ScriptReference/MonoBehaviour.html