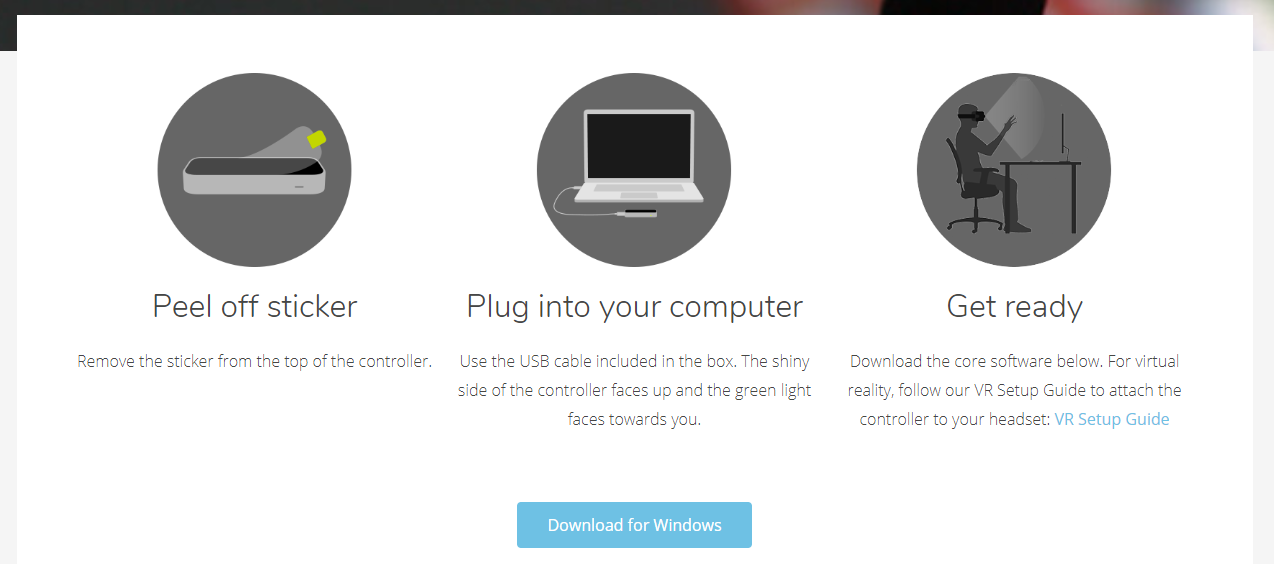
**1. 멤버**

김평강, 신지현, 전예은 홍혜성

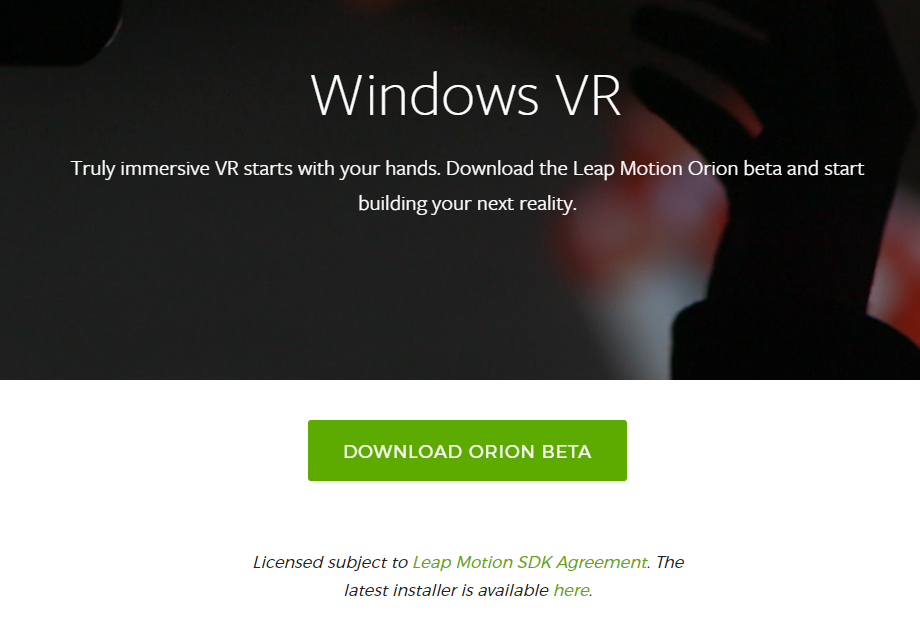
**2. 프로젝트 진행에 필요한 설치**

1. Leap motion

-Leap motion setup을 pc에 다운로드 한다.



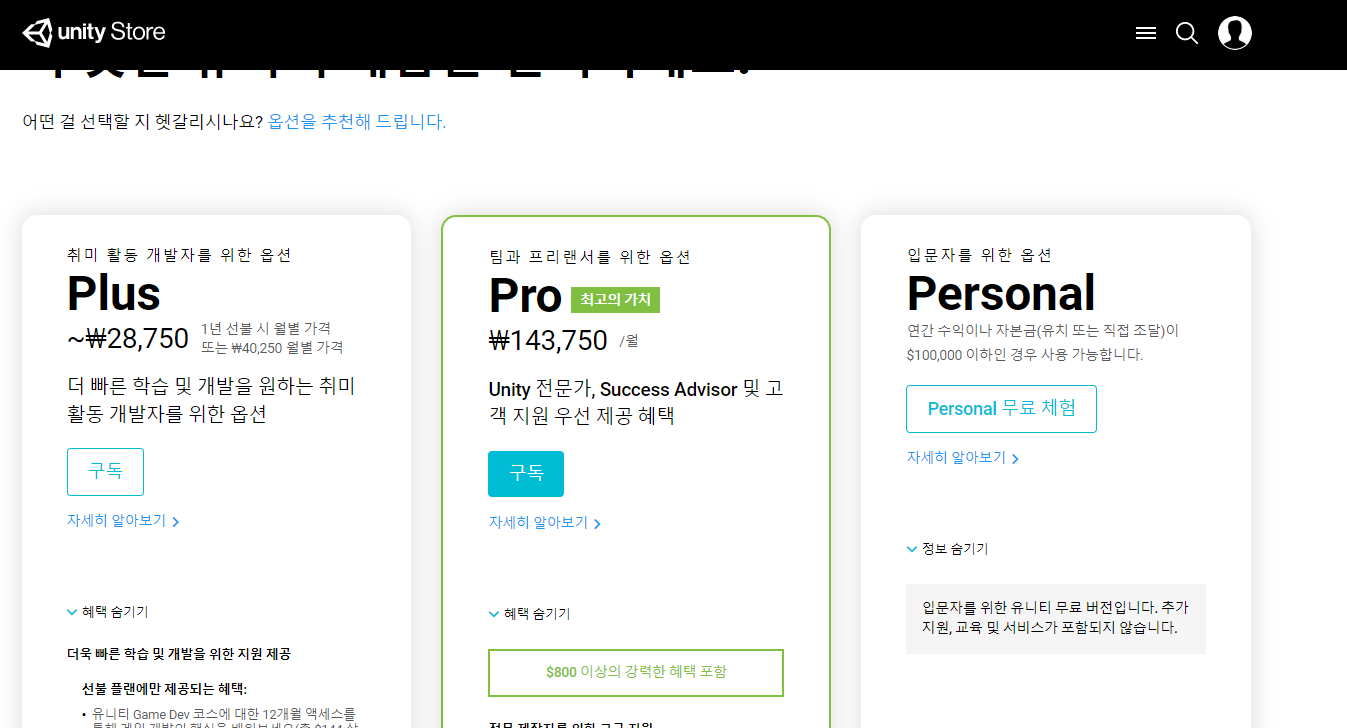
-아래 파일들을 다운로드한다.



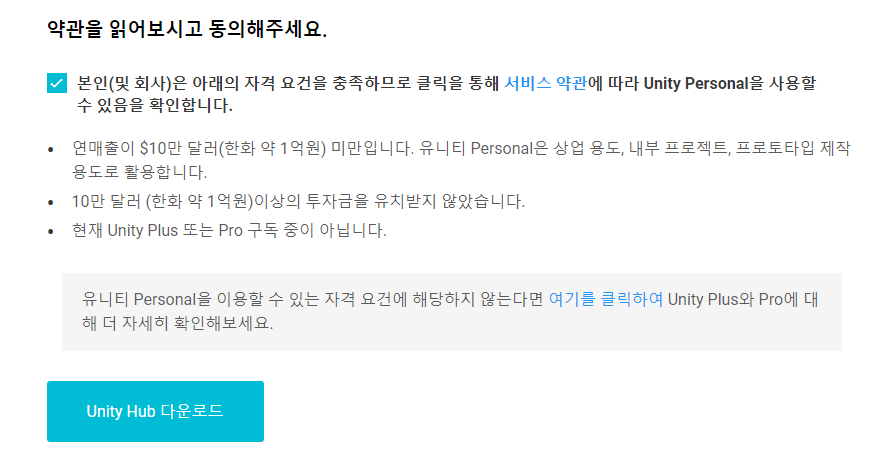


1. Unity

- Personal 선택



- 약관 동의 후 Unity Hub 다운로드를 클릭한다. 로그인 후 사용할 수 있다.



**3. Research**

Battle ground in handing 게임은 FPS 게임에 익숙하지 않은 사람들이 게임을 직관적으로 이해하고 쉽게 접근 할 수 있고, 기존에 사용자들 또한 게임을 흥미로운 관점으로 접근 할 수 있도록 돕는 것을 목표로 하였다. 따라서 사용자들의 게임 이해도, 게임 시간, 게임 흥미에 대한 정보를 수집하고자 하였다. 다음 단계는, Unity 와 leap-motion간의 상관관계를 이해하고 연동할 수 있는 연결점을 찾는 것을 조사의 목적으로 하였다. 실제로 게임에 등장하는 asset 을 구현하기 위해서는 Unity asset score, google 사이트를 참조하여 게임에 활용할 수 있는 정보와 자원들을 적극 수집하였다.

3-1) 용도

Battle ground in handing 는 기존의 FPS게임과의 방법과 달리 사용자가 직관적으로 게임을 이해하여 흥미로운 방법으로 게임을 즐길 수 있도록 한다.

3-2) 목표

Leap motion 기술과 unity engine을 활용하여 사람의 손을 인식하고 총을 쏘는 상황을 직관적으로 이해할 수 있도록 프로그램을 구현하여 게임을 할 수 있도록 한다.

3-3) 대상

FPS게임을 쉽게 즐기지 못하는 사람을 주로 대상으로 하며, 기존에 FPS게임을 하고 있는 사람을 대상으로도 새로운 게임 환경을 제공하여 그 발전 가능성을 확인하도록 한다.

**4. 게임 첫 화면**



**5. Source Code**

Using C# Script

“SourceCode” 폴더에 있음

[총 쏘기 구현-Weapon.cs]

using UnityEngine;

using UnityEngine.UI;

public class Weapon : MonoBehaviour {

// Weapon Specification

public string weaponName;

public int bulletsPerMag;

public int bulletsTotal;

public int currentBullets;

public float range;

public float fireRate;

// Parameters

private float fireTimer;

private bool isReloading = false;

// References

public Transform shootPoint;

public ParticleSystem muzzleFlash;

public Text bulletsText;

private Animator anim;

// Sound

public AudioClip shootSound;

public AudioClip reloadSound;

public AudioSource audioSource;

// Use this for initialization

private void Start () {

currentBullets = bulletsPerMag;

anim = GetComponent<Animator> ();

bulletsText.text = currentBullets + " / " + bulletsTotal;

}

// Update is called once per frame

private void Update () {

AnimatorStateInfo info = anim.GetCurrentAnimatorStateInfo (0);

isReloading = info.IsName ("Reload");

if (Input.GetButton ("Fire1")) {

if (currentBullets > 0) {

Fire ();

} else {

DoReload ();

}

}

if (Input.GetKeyDown (KeyCode.R)) {

DoReload ();

}

if (fireTimer < fireRate) {

fireTimer += Time.deltaTime;

}

}

private void Fire(){

if (fireTimer < fireRate) {

return;

}

Debug.Log ("Shot Fired!");

RaycastHit hit;

if(Physics.Raycast(shootPoint.position, shootPoint.transform.forward, out hit, range)){

Debug.Log("Hit!");

}

currentBullets--;

fireTimer = 0.0f;

audioSource.PlayOneShot (shootSound); // shoot sound

anim.CrossFadeInFixedTime ("Fire", 0.01f); // fire animation

muzzleFlash.Play();

bulletsText.text = currentBullets + " / " + bulletsTotal;

}

private void DoReload(){

if (!isReloading && currentBullets < bulletsPerMag && bulletsTotal > 0) {

anim.CrossFadeInFixedTime ("Reload", 0.01f); // Reload

audioSource.PlayOneShot (reloadSound);

}

}

public void Reload(){

int bulletsToReload = bulletsPerMag - currentBullets;

if (bulletsToReload > bulletsTotal) {

bulletsToReload = bulletsTotal;

}

currentBullets += bulletsToReload;

bulletsTotal -= bulletsToReload;

bulletsText.text = currentBullets + " / " + bulletsTotal;

}

}

[장전-ReloadState.cs]

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ReloadState : StateMachineBehaviour {

public float reloadTime = 0.7f;

private bool reloaded = false;

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

//override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {

//

//}

// OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks

override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {

if (reloaded)

return;

if (stateInfo.normalizedTime >= reloadTime) {

animator.GetComponent<Weapon> ().Reload ();

reloaded = true;

}

}

// OnStateExit is called when a transition ends and the state machine finishes evaluating this state

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {

reloaded = false;

}

// OnStateMove is called right after Animator.OnAnimatorMove(). Code that processes and affects root motion should be implemented here

//override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {

//

//}

// OnStateIK is called right after Animator.OnAnimatorIK(). Code that sets up animation IK (inverse kinematics) should be implemented here.

//override public void OnStateIK(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {

//

//}

}

[총알 맞은 곳에 흔적내기(탄흔)- WeaponSway.cs]

using UnityEngine;

public class WeaponSway : MonoBehaviour {

public float swayAmount = 0.02f;

public float smoothAmount = 6f;

public float maxAmount = 0.06f;

private Vector3 originalPosition;

// Use this for initialization

private void Start () {

originalPosition = transform.localPosition;

}

// Update is called once per frame

private void Update () {

float positionX = -Input.GetAxis ("Mouse X") \* swayAmount;

float positionY = -Input.GetAxis ("Mouse Y") \* swayAmount;

Mathf.Clamp (positionX, -maxAmount, maxAmount);

Mathf.Clamp (positionY, -maxAmount, maxAmount);

Vector3 swayPosition = new Vector3 (positionX, positionY, 0);

transform.localPosition = Vector3.Lerp (transform.localPosition, originalPosition + swayPosition, Time.deltaTime \* smoothAmount);

}

}

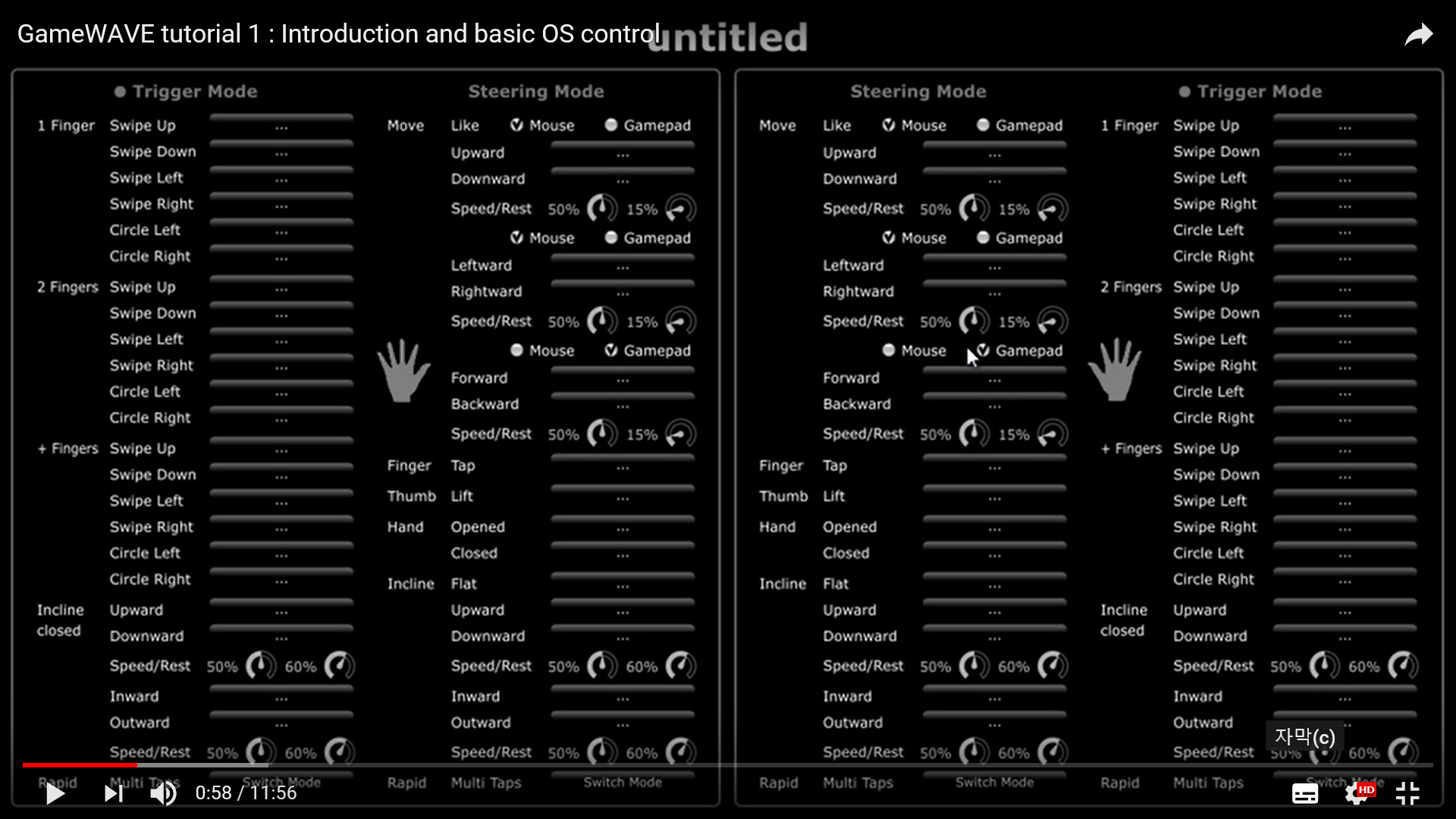
*출처:* [*https://m.blog.naver.com/choish1919/221257826199*](https://m.blog.naver.com/choish1919/221257826199)

**6. 구동방식**

우리 팀은 동일한 게임에 대해 두 가지 방법의 interaction을 구현하고자 한다. 두 가지 방법을 구현한 후 그 둘을 비교할 것이다.

첫 번째는 기존의 FPS게임방식과 유사하게 마우스와 키보드를 활용한 interaction이다. 마우스의 이동에 따라 시점이 움직이게 한다. 정전은 자동으로 되며(총 1500발, 30발씩 장전됨), 사람이 이동하는 것은 키보드 방향키로 조작한다.

두 번째는 leap motion device을 활용한 interaction이다. 손가락 모션을 이용하여 총알을 발사하고, 다른 한쪽 손의 손바닥을 움직여 시점과 움직임을 조절한다. 이러한 leap motion을 이용한 interaction을 구현하기 위해 GameWAVE라는 툴을 이용하였다.



GameWAVE는 마우스와 키보드를 통한 interaction을 leap motion device을 통한 interaction으로 변환하여 주는 툴이다. 위의 사진과 같이 개발자가 직접 마우스와 키보드의 속도, 방향, 클릭 등을 원하는 손동작과 대응시킬 수 있으며, 이를 통해 다양한 interaction 조작이 가능하다.

아래의 링크는 GameWAVE tutorial영상이다.

<https://www.youtube.com/watch?v=bsQK3ab3YhQ>

**7. 향후 계획**

1) 지금까지 구현한 에셋에 더해서 게임을 더 액티브하게 만들고자 한다. 게임에 등장하는 적의 방향과 움직임에 자유도를 더해주는 것이 첫번째 목표이다. 또한 사용자와 적의 전투 상황 시 구현 조건을 디테일하게 조정해 자연스러운 상황을 연출시키고자 한다.

2) 게임은 사용자가 1인용 게임만을 추구하지 않고, 네트워크를 가능하게 하여 멀티유저가 게임을 즐길 수 있도록 한다.

**8. Reference**

https://m.blog.naver.com/PostList.nhn?blogId=choish1919