Sample Script

Parent and Child Classes

Parent class allows me to create efficient way to code a game without repeating and redundant coding. I just call the parent's functions if the child class can access from parent. It is also important to practice encapsulation in order to prevent from unnecessary accessing the data field.

Every child class related to character an abstract class require to have Rigidbody, animation, shooting damage, and hit point in common. Rather than coding in each file separately, One abstract class file is enough to code in once for every child class.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Character : MonoBehaviour {

    [SerializeField] private float thisHealth = 0.0f;
    [SerializeField] private float thisSpeed = 0.0f;
    [SerializeField] private Vector3 thisDirection = Vector3.zero;

    [SerializeField] private GameObject thisBulletPrefab = null;
    [SerializeField] private GameObject thisBloodPrefab = null;
    [SerializeField] private int thisNumBloods = 0;
    [SerializeField] private float thisBloodRadius = 0.0f;

    [SerializeField] private float setCoolTime = 0.0f;



    private float tickTime = 0.0f;
    private Rigidbody2D myRigid = null;
    private Animator myAnim = null;


    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    protected void StartThing()
    {
        myRigid = GetComponent<Rigidbody2D>();
        MoveInDirection(thisDirection);
        myAnim = GetComponent<Animator>();


    }
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player : Character {
    [SerializeField] private Vector3 thisShootingDirection = Vector3.zero;
    // Use this for initialization
    void Start () {
        StartThing();
        print("Title: RogueLike");
        print("Developer: Jung Hwan Kim");
        print("Instructor: Miles Horak");
        print("Instruction");
        print("Movement: ARROW KEYS");
        print("Shoot: SPACEBAR");
    }

    // Update is called once per frame
    void Update () {
        Vector3 aDirection;
        aDirection = Vector3.zero;



        if (Input.GetKey(KeyCode.LeftArrow))
        {
            aDirection += Vector3.left;
            thisShootingDirection = aDirection;

        }

        if (Input.GetKey(KeyCode.RightArrow))
        {
            aDirection += Vector3.right;
            thisShootingDirection = aDirection;
        }

        if (Input.GetKey(KeyCode.DownArrow))
        {
            aDirection += Vector3.down;
            thisShootingDirection = aDirection;
        }
```

Jump Script

While a main character stays on ground object, the boolean variable called “isGround” is true. After a player press button to jump, the main character is above from a ground. So, the boolean variable is false.

```
private void Jump()
{
    if (isGround)
    {
        verticalVelocity -= gravity * Time.deltaTime;
        if(Input.GetKeyDown(KeyCode.Space))
        {
            verticalVelocity = jumpForce;
            myAnimator.SetTrigger("Jump");

        }
    }
    else
    {
        verticalVelocity -= gravity * Time.deltaTime;

    }
    myRigidBody.velocity = new Vector3(0, verticalVelocity, 0);

}
```

```
private void OnCollisionStay(Collision col){
    if (col.gameObject.CompareTag ("Ground") && !gctlrSCript.CheckPoint) {
        isGround = true;
        myAnimator.SetTrigger("Ground");
    }
    if (col.gameObject.CompareTag ("DeathFall")) {
        healthStat.CurrentVal = 0;
    }
}
```

```
private void OnCollisionExit(Collision col)
{
    if (col.gameObject.CompareTag ("Ground")) {
        isGround = false;

    }

}
```

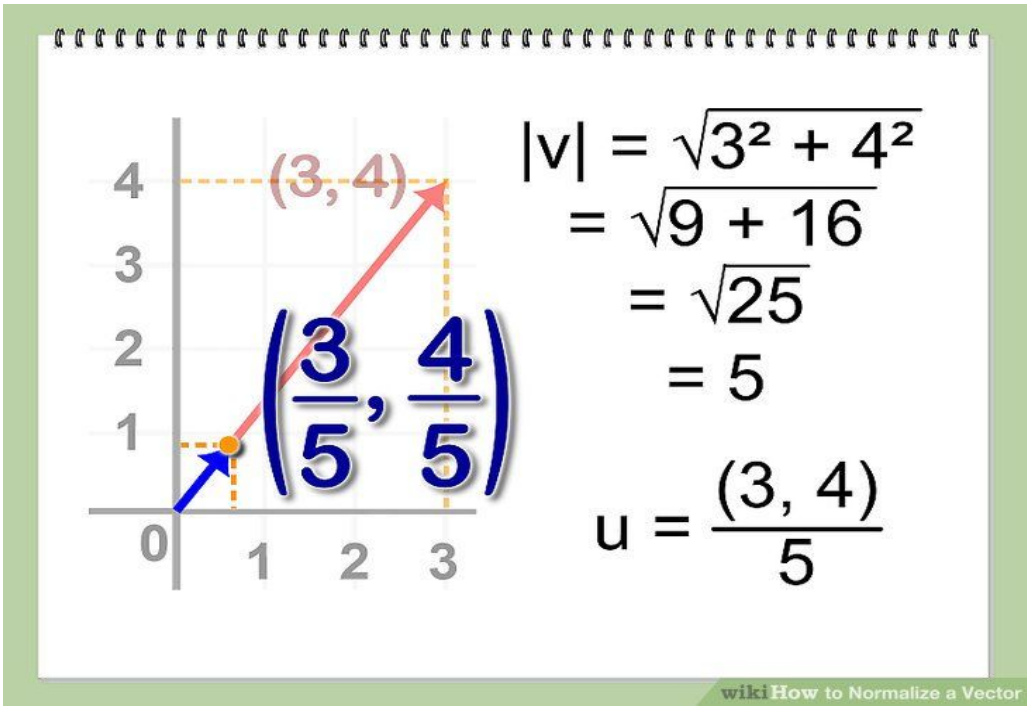
Turret’s Targeting and Rotation

We can’t instantly move turret in real life. For our game mechanic, we delay the Artificial Intelligent Object requires a time to aiming the target as it rotates.

To solve the problem for turret rotation as aiming a target, I normalize a vector from mathematical solution. For example, if the our target is on vector (3,4) and our turret is on (0,0), the turret. This picture shows:

```
Vector3 dir = (transform.position - target.position).normalized;

Quaternion newRot = Quaternion.LookRotation(dir, transform.forward);
transform.rotation = Quaternion.Slerp(transform.rotation, newRot, 2f * Time.deltaTime);
transform.localEulerAngles = new Vector3(0f, 0f, transform.localEulerAngles.z);
```



Generating Web Flash Game from Webgl

After Generating the my flash game file, we sometimes moves our html folder and build folder are in separate location. As a result, our html unity complains that it cannot access our build file. So, the yellow highlighted lines must be changed when directory specifically locates the build file.

```html
<!DOCTYPE html>
<html lang="en-us">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Unity WebGL Player | Racing</title>
    <link rel="shortcut icon" href="TemplateData/favicon.ico">
    <link rel="stylesheet" href="TemplateData/style.css">
    <script src="TemplateData/UnityProgress.js"></script>
    <script src="RaceFlash/Build/UnityLoader.js"></script>
    <script>
      var gameInstance = UnityLoader.instantiate("gameContainer", "RaceFlash/Build/RaceFlash.json", {onProgress: UnityProgress});
    </script>
  </head>
  <body>
    <div class="webgl-content">
      <div id="gameContainer" style="width: 1024px; height: 768px"></div>
      <div class="footer">
        <div class="webgl-logo"></div>
        <div class="fullscreen" onclick="gameInstance.SetFullscreen(1)"></div>
        <div class="title">Racing</div>
      </div>
    </div>
  </body>
</html>
```

Projectile Instantiation and Projectile Targeting

In 2D game environment, the projectile must not be instantiated from center of shooter. We must create offset “posBullet” which gives additional offset from shooter position.

```
if (thisFacingRight)
{
    posBullet = transform.position + new Vector3(10f, 0f, 0f);

}
else
{
    posBullet = transform.position + new Vector3(-10f, 0f, 0f);
}
```

And this is for instantiating a projectile

```
GameObject bulletClone = Instantiate(thisBullet, posBullet, Quaternion.identity);// instantiate swing!
```

Each bullet must have a path toward the target.

```
bulletClone.transform.rotation = transform.rotation;
bulletClone.transform.LookAt(target.transform);
bulletClone.transform.eulerAngles = new Vector3(bulletClone.transform.eulerAngles.x, bulletClone.transform.eulerAngles.y, bulletClone.transform.eulerAngles.z);
bulletClone.GetComponent<Rigidbody>().AddForce(bulletClone.transform.forward * thisBulletSpeed);
```

Save and Load

"SavePlayer" function is for creating a save file or overwriting save file.
"LoadPlayer" function is received data from saved file.

```
public static void SavePlayer(TheOutPost aOutPost)
{
    BinaryFormatter formatter = new BinaryFormatter();
    string path = Application.persistentDataPath + "/PlayerData.dat";

    Debug.Log(path);
    FileStream stream = new FileStream(path, FileMode.Create);
    OutPostData data = new OutPostData(aOutPost);

    formatter.Serialize(stream, data);
    stream.Close();

}

public static OutPostData LoadPlayer()
{
    string path = Application.persistentDataPath + "/PlayerData.dat";

    if (File.Exists(path))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(path, FileMode.Open);

        OutPostData data = formatter.Deserialize(stream) as OutPostData;
        stream.Close();
        return data;
    }else
    {
        Debug.LogError("Save file not found in " + path);
        return null;
    }
}
```

Only int, float, string, boolean, and list can be encrypted for saving and decrypted for loading. However Game object generally cannot be saved.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


[System.Serializable]
public class OutPostData {

    private int daysPassed;
    private int currentExtraMorale;
    private int ammoPoint;
    private int foodPoint;
    private int researchPoint;
    private int attackDays;


    private List<string> thisSoldierRanks;
    private List<string> thisSoldierLastName;
    private List<string> thisSoldiersFirstName;
    private List<string> thisSoldiersStatus;
    private List<int> thisSoldierInjuredDays;
    private List<bool> thisSoldierIsInjured;



    public OutPostData(TheOutPost aOutPost)
    {

        daysPassed = aOutPost.ThisDayPassed;
        currentExtraMorale = aOutPost.ThisCurrentExtraMorale;
        ammoPoint = aOutPost.ThisNumAmmo;
        foodPoint = aOutPost.ThisNumFood;
        researchPoint = aOutPost.ThisResearchPoint;
```

List

List is very useful for many strategy game such as inventory system, tracking system for all objects in game-play scene, and instanting each unique object. I use list when new object is created and removed when object is no longer in active.

```
private List<string> thisSoldierRanks;
private List<string> thisSoldierLastName;
private List<string> thisSoldiersFirstName;
private List<string> thisSoldiersStatus;
private List<int> thisSoldierInjuredDays;
private List<bool> thisSoldierIsInjured;
```

```
thisSoldierRanks = new List<string>();
thisSoldierLastName = new List<string>();
thisSoldiersFirstName = new List<string>();
thisSoldiersStatus = new List<string>();
thisSoldierInjuredDays = new List<int>();
thisSoldierIsInjured = new List<bool>();

thisSoldiersAllList.Clear(); // initialized if there are remained list;

thisSoldiersAllList.AddRange(aOutPost.ThisSoldiersinBarack);
thisSoldiersAllList.AddRange(aOutPost.ThisSoldiersinHospital);
thisSoldiersAllList.AddRange(aOutPost.ThisSoldierinDepot);
thisSoldiersAllList.AddRange(aOutPost.ThisSoldierinFactory);
thisSoldiersAllList.AddRange(aOutPost.ThisSoldierinScout);
```

```
if (thisSoldiersAllList.Count > 0)
{
    for (int i = 0; i < thisSoldiersAllList.Count; i++)
    {
        thisSoldierRanks.Add(thisSoldiersAllList[i].GetComponent<SoldierSlot>().SoldierRank);
        thisSoldierLastName.Add(thisSoldiersAllList[i].GetComponent<SoldierSlot>().LastName);
        thisSoldiersFirstName.Add(thisSoldiersAllList[i].GetComponent<SoldierSlot>().FirstName);
        thisSoldiersStatus.Add(thisSoldiersAllList[i].GetComponent<SoldierSlot>().SoldierStatus);
        thisSoldierInjuredDays.Add(thisSoldiersAllList[i].GetComponent<SoldierSlot>().InjuredDays);
        thisSoldierIsInjured.Add(thisSoldiersAllList[i].GetComponent<SoldierSlot>().Injured);

    }
}
```