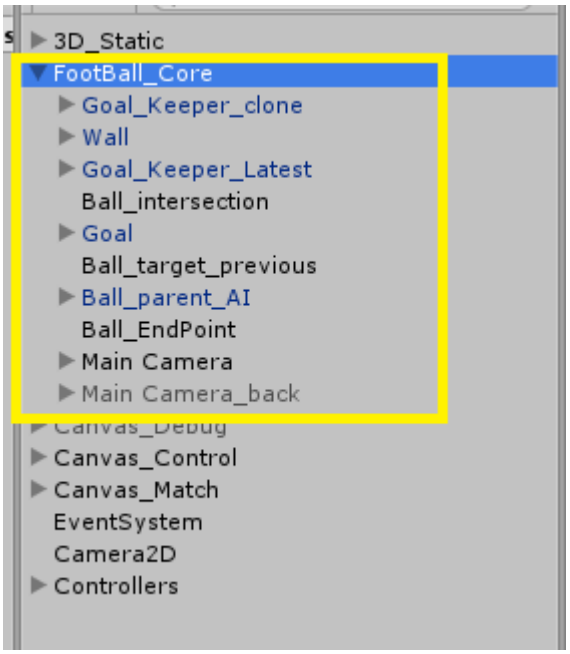
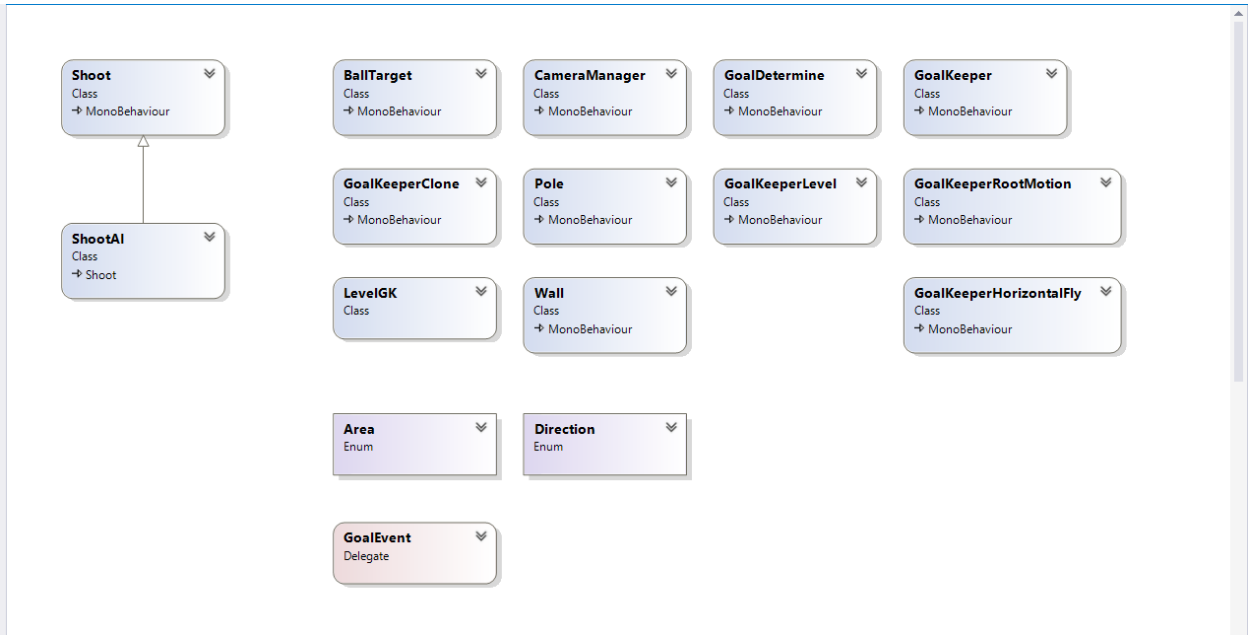
FLICK FOOTBALL

Usually in a scene demo, the core components are grouped into same parent as in picture:



Main components:



- Version 1.0

- Version 1.0.1

- Version 1.0

1. Shoot.cs

This class handle the logic of making the ball flying and curving when receiving touch/mouse input.

2. ShootAI.cs

This is subclass of Shoot.cs, this class handle the logic of making the ball flying and curving from a randomized ball path, i.e. artificial intelligence shooting.

3. GoalKeeper.cs

This class handle GoalKeeper logic:

- Evaluate ball orbit to estimate arrive time
- Update animator controller parameters like direction, distance, and height depend on ball position
- Do jump logic when the ball come
- Provide event where other classes can listen

4. GoalKeeperHorizontalFly.cs

This class handle the logic of moving the goalkeeper to the left/right depend on current ball path, animate the goal keeper position when jumping and after jumped

5. GoalKeeperLevel.cs

This class handle the difficulty data of goal keeper of all levels. Change goal keeper level. There are totally 13 levels, from level index 0 -> 11 is for AI, level index 12 is for user when controlling goalkeeper. The data of 13^{th} level was decided after many tests had been done.

6. GoalKeeperRootMotion.cs

This class override `OnAnimatorMove` method to control goalkeeper movement to the left/right by adjusting its x component of transform.position

7. GoalKeeperClone.cs

This class control the “Ghost” GoalKeeper that is located far away from the play field. This clone GK is used to estimate the distance around GK that he can catch the ball without moving a cm. This help “GoalKeeperHorizontalFly.cs” know whether he should move GK to the left or right.

8. GoalDetermine.cs

This class is responsible for checking whether ball is outside or inside the goal. This can detect which corner the ball was passed, whether the ball hit the poles before it goes into the goal. This also contain logic that animate the little effect at the position the ball cross the line to make a score.

9. Wall.cs

This handle the logic that display the human wall, place it appropriately depend on ball position. Set transparency of the material depend on whether user is controlling the ball or controlling the goalkeeper.

10. CameraManager.cs

This class handle the logic of moving the camera a little bit forward after shooting. It also contains a method that move the camera from the middle of the field to its current position.

11. SwitchCamera.cs

This contains logic switch between main camera and back camera. Back camera is used when user control the goal keeper.

12. SlowMotion.cs

This control the Timescale to give great moment when scoring a goal

13. Pole.cs

This class is attatched to the 3 poles of the goal, fire event when the ball hit it.

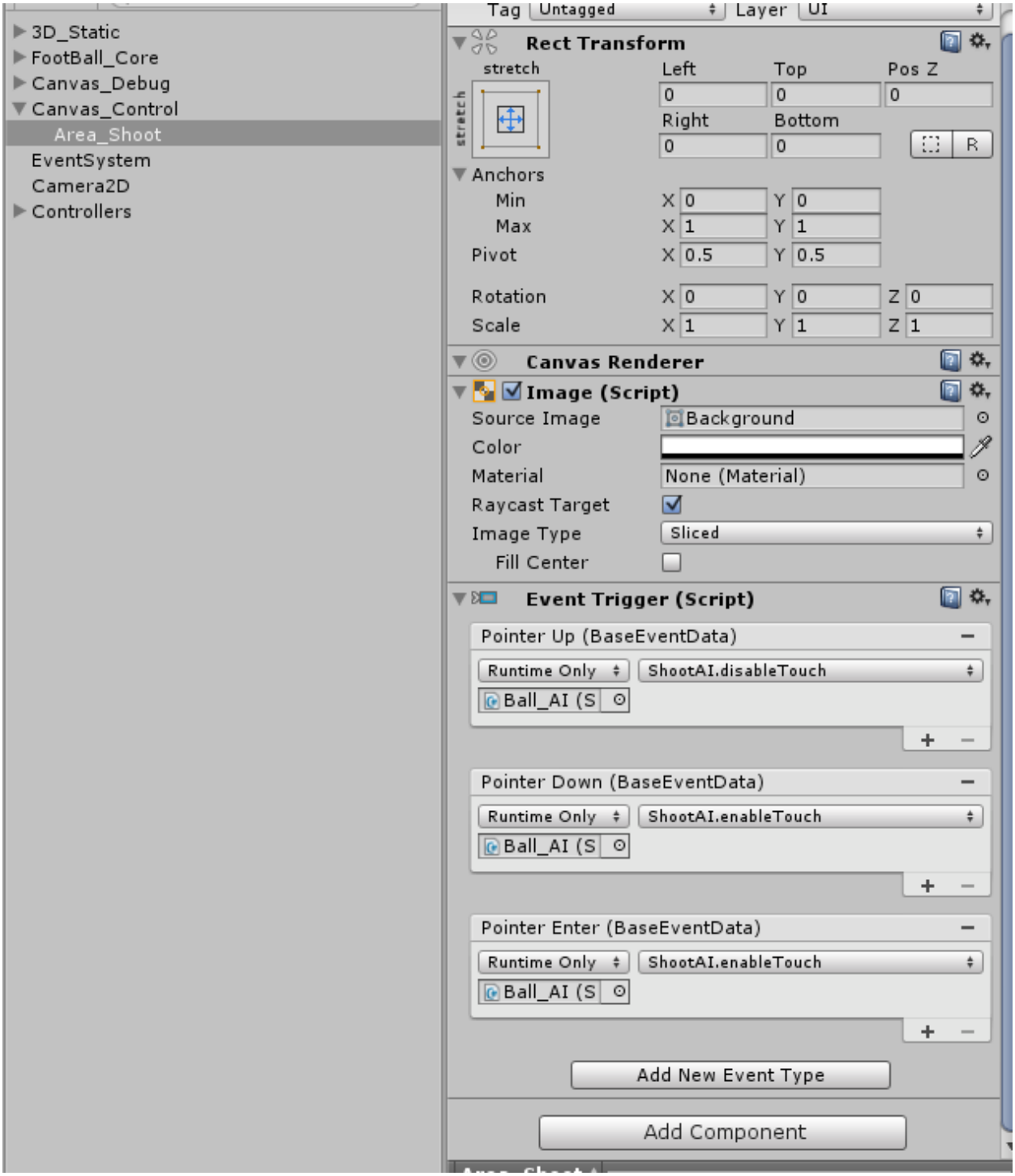
14. BallTarget.cs

This class responsible for evaluating the intersection point of the ball path and goalkeeper position with respect to z-axis. The hand of the goal keeper will try to reach this position to hit the ball.

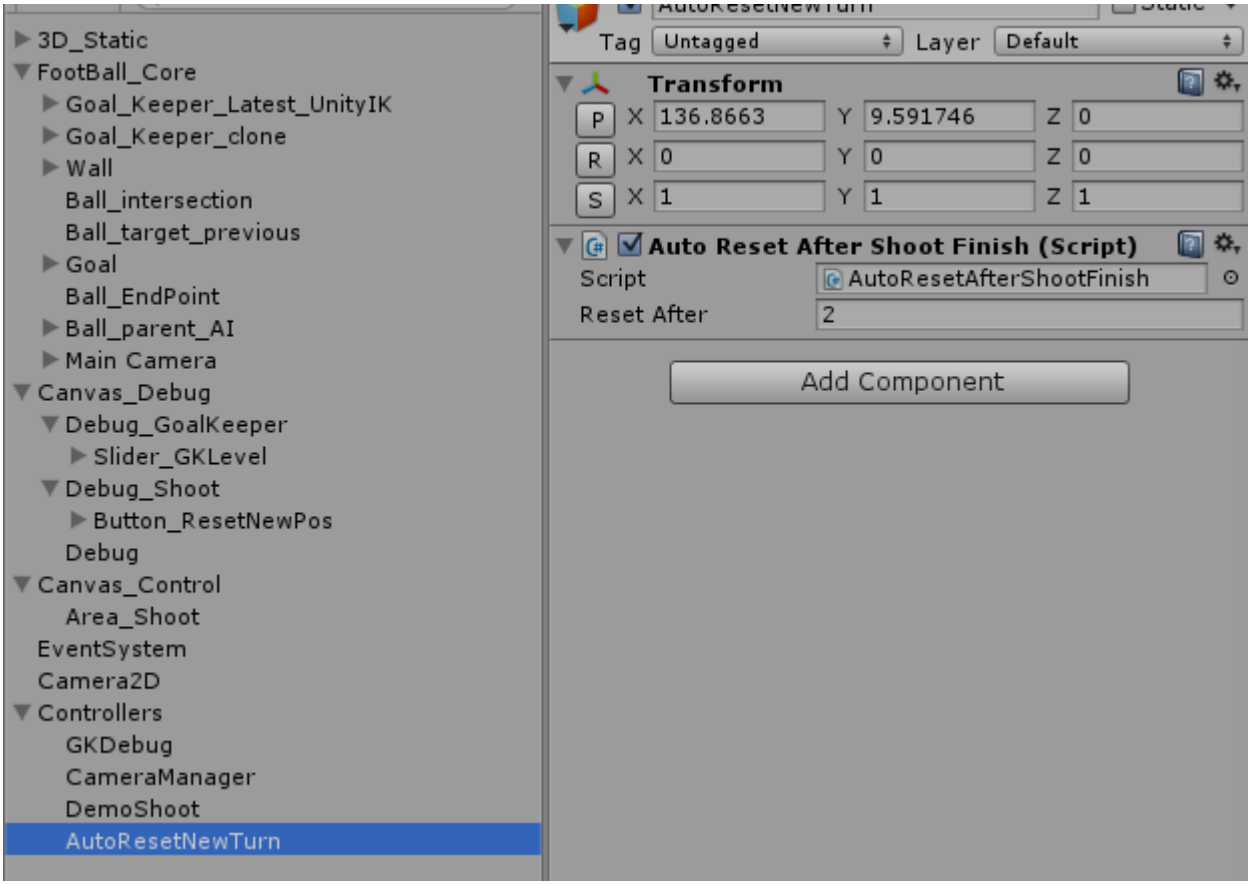
15. Demo 1: Shoot

This demo is for getting to know about the shooting mechanism. In this kind of flick football, you can control the ball even after it has been kicked. Thus you can create the curved ball path very easily.

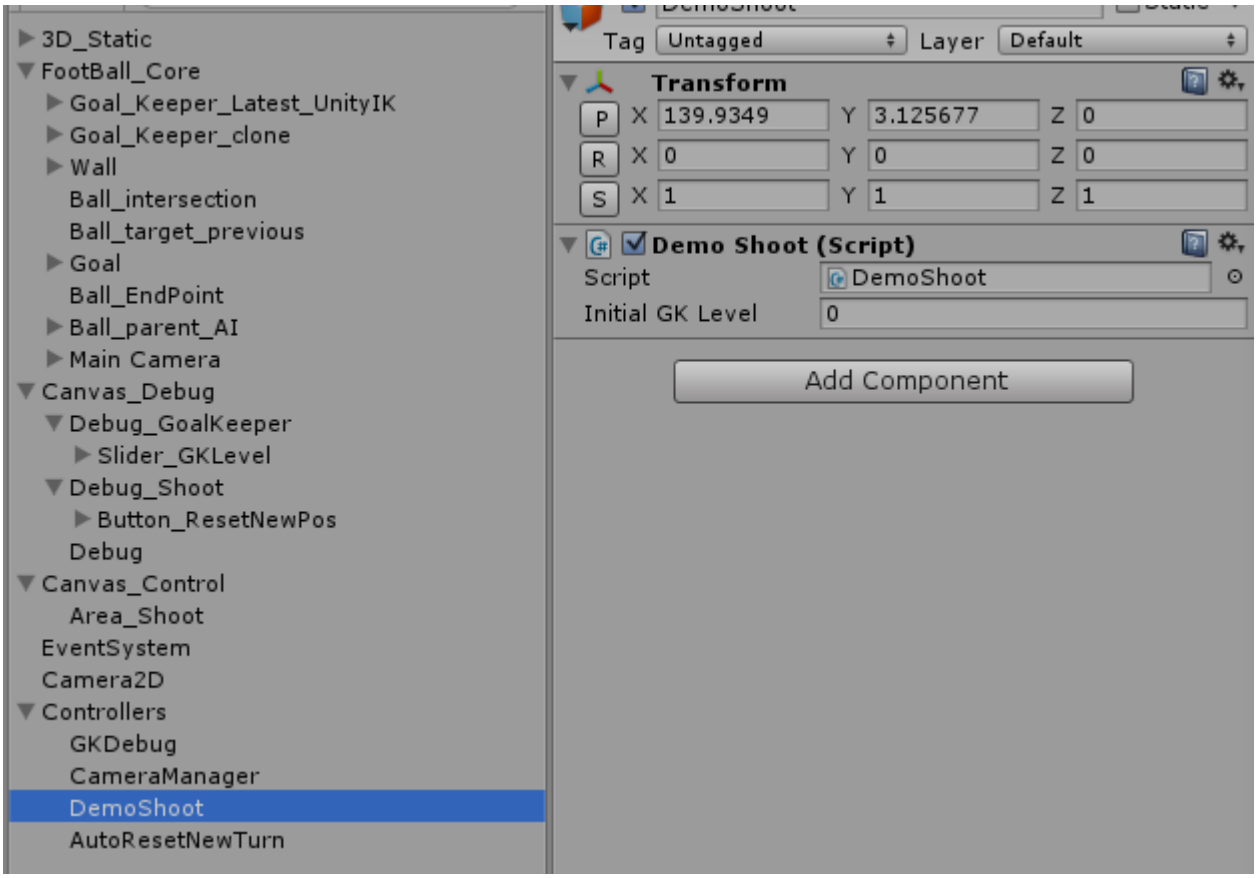
- The slider on top of the screen is for setting goal keeper level, the higher the more challenge the goal keeper is, try to adjust it and see.
- Button “Random Position”: random new ball position of the kick



Area_Shoot: this button will receive touch input and call ShootAI.enableTouch(), this ask ShootAI.cs, more specifically Shoot.cs to capture touch input and make the ball flying, curving correspondingly.

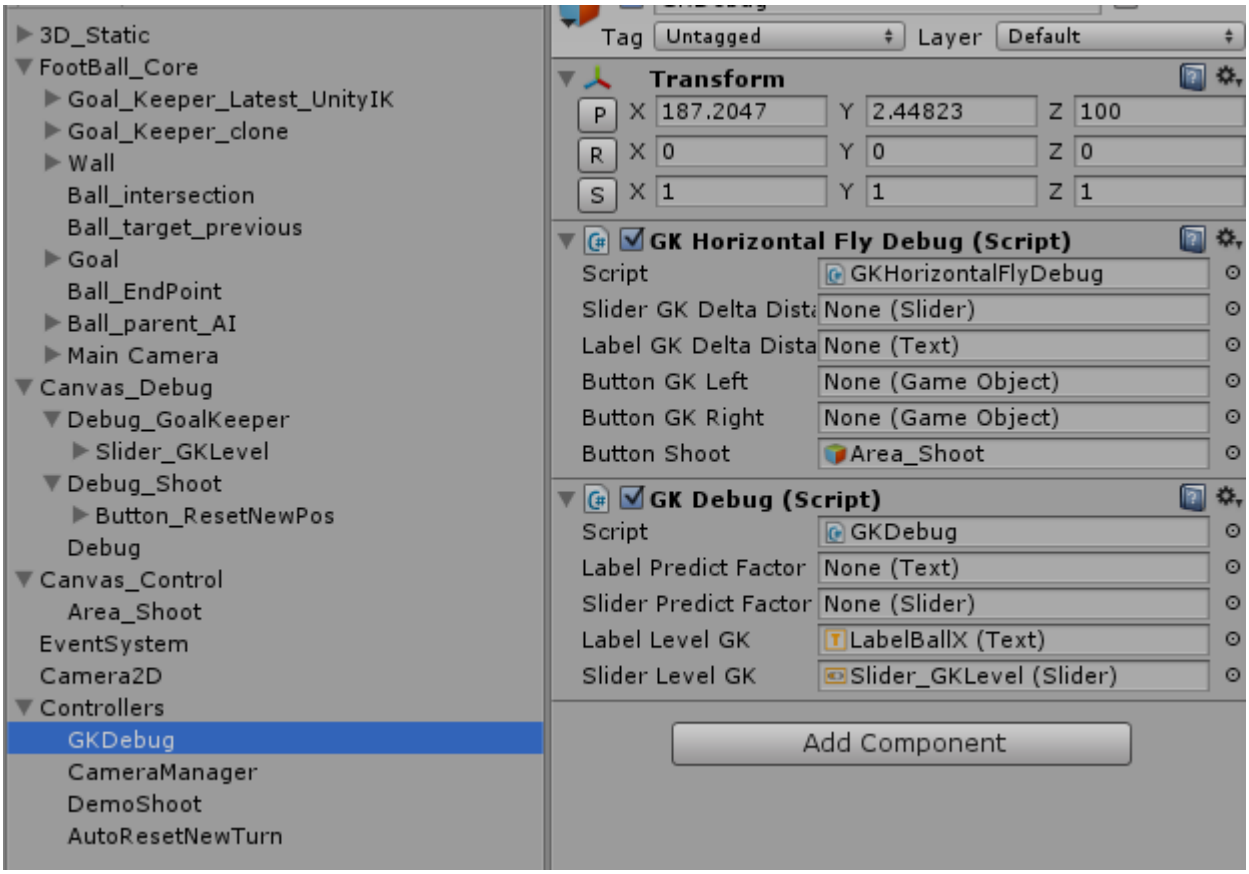


There is a simple script “AutoResetAfterShootFinish”. As the name of the script, it contains a very simple logic that help to reset new turn after an interval (2 seconds by default).



DemoShoot.cs:

`public void Reset(bool shouldRandomNewPos)` : this method handle the necessary steps need in order to correctly reset a new turn.



GKHorizontalFlyDebug.cs: this class is a helper class to debug the engine. It contains events which are called from the UI elements. In this example, it doesn't do much of it designed purposes, so just leave it for the coming demos.

GKDebug.cs: this control the GKLevel slider, listen to event from the slider and adjust GoalKeeper level accordingly. Here is the exact code that does the logic:

```
public void onValueChanged_LevelGK(float val)
    {
        if (GoalKeeperLevel.share != null)
        {
            int level = (int)Mathf.Lerp(0, GoalKeeperLevel.share.getMaxLevel(), val);
            GoalKeeperLevel.share.setLevel(level);
```

val vary from 0 to 1, we used it as t in Mathf.Lerp method to get the GK level, call setLevel() method of GoalKeeper to set the level.

16. Demo 2: Shoot Wall

Button Random Position: click this to random new ball position

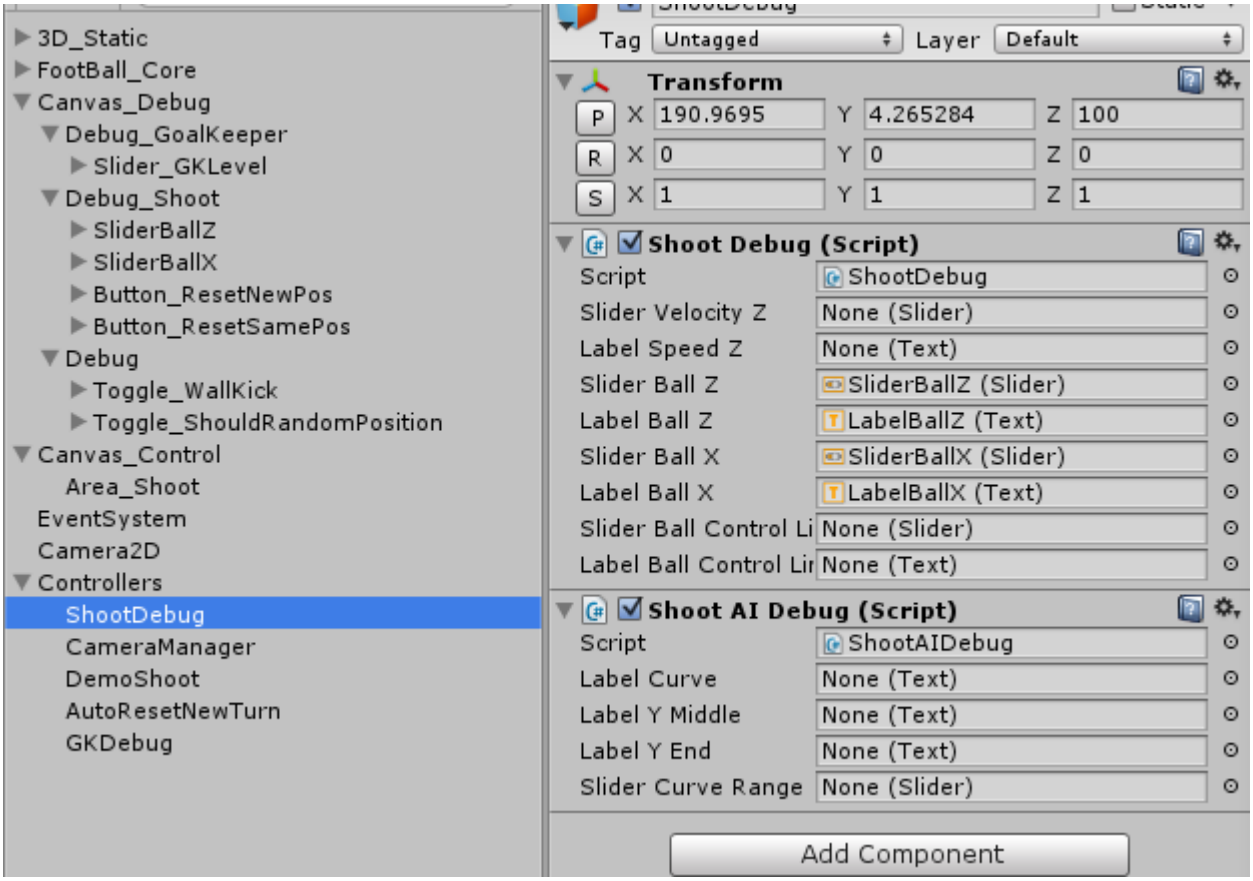
Button Same Position: click this to move to next turn and keep the same ball's position

Slider Ball X: this slider tell us about x component of ball's position, adjust this and click "Same Position" to have it affected immediately.

Slider Ball Z: this slider tell us about z component of ball's position, adjust this and click "Same Position" to have it affected immediately.

Toggle Random Pos On Reset: check this if you want to randomize ball's position every new turn.

Toggle Wall Kick: check this to experience wall kick as the demo's name suggested



ShootDebug.cs: this class help to debug the shoot mechanism, it contain methods to listen to changes from UI element and act accordingly.

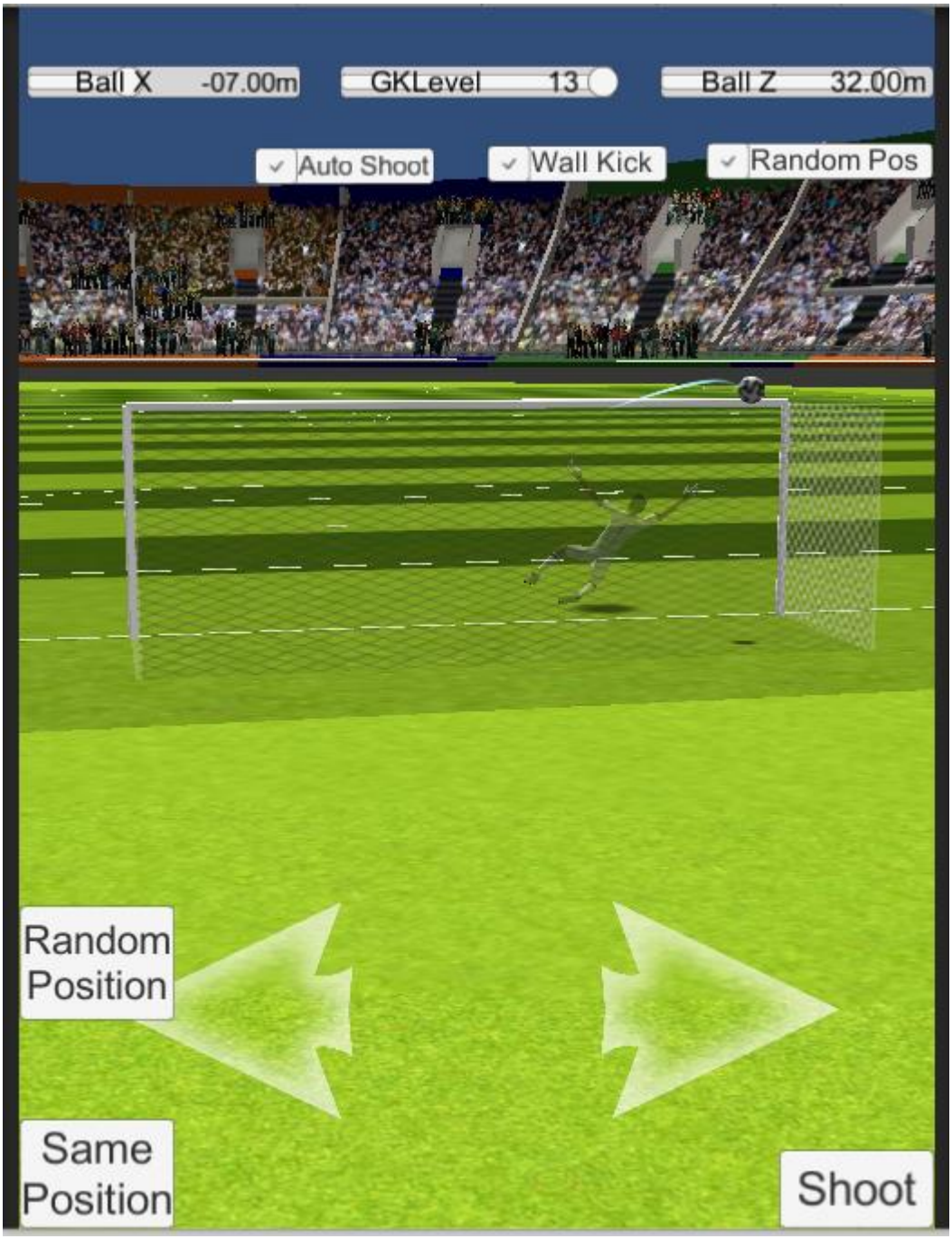
```
  public void OnChange_SliderBallZ(float val)
    {
        if (val == 0)
            Shoot.share.ballPositionZ = -16.5f;
        else
        {
            Shoot.share.ballPositionZ = (int)-Mathf.Lerp(Shoot.share._distanceMinZ,
Shoot.share._distanceMaxZ, val);
        }
```

This method listen to event from the Slider Ball Z, and update the value to “ballPositionZ” of Shoot.cs

```
public void OnChange_SliderBallX(float val)
    {
        Shoot.share.ballPositionX = (int)Mathf.Lerp(Shoot.share._distanceMinX,
Shoot.share._distanceMaxX, val);
        _labelBallX.text = "" + Shoot.share.ballPositionX.ToString("00.00") + "m";
    }
```

This do set “ballPositionZ” of Shoot.cs

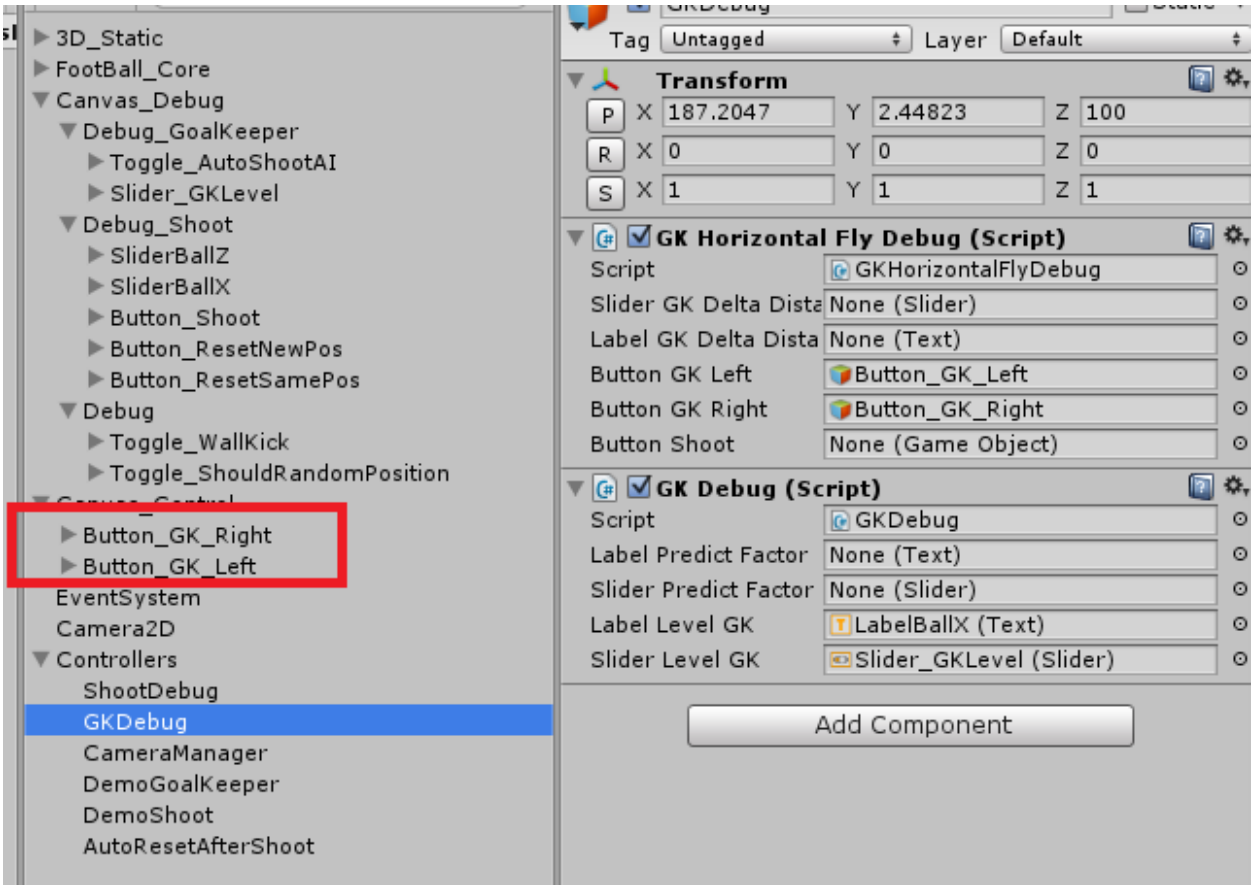
17. Demo 3: GoalKeeper



Toggle Auto Shoot: check this to have the ShootAI automatically do the shoot

Button Shoot: this button is to manually make a shoot.

2 arrows: means touch on half left of the screen to move the GK to the left and vice versa. Beside in editor mode you can use arrow keys on keyboard to control him.

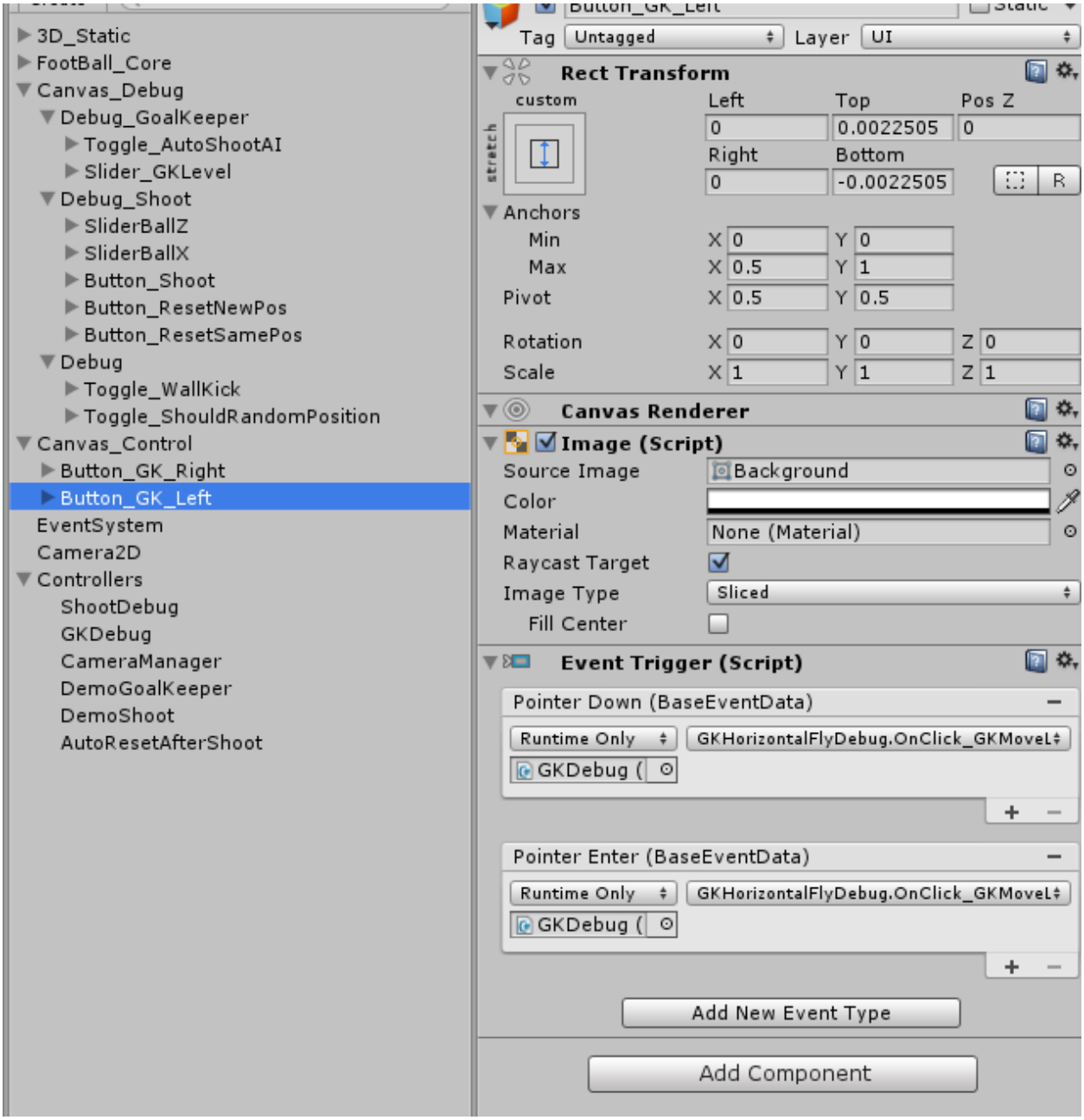


GKHorizontalFlyDebug.cs: in this example, it references to 2 gameobjects in red rectangle as in above piture. The 2 gameobjects are the button to receive event when user touch on the screen. They they will call methods :

```
public void OnClick_GKMoveLeft()
    {
        GoalKeeperHorizontalFly.share.MoveGKToLeft();
    }

    public void OnClick_GKMoveRight()
    {
        GoalKeeperHorizontalFly.share.MoveGKToRight();
    }
```

This is the exact meat of code that move the goalkeeper to the left/right

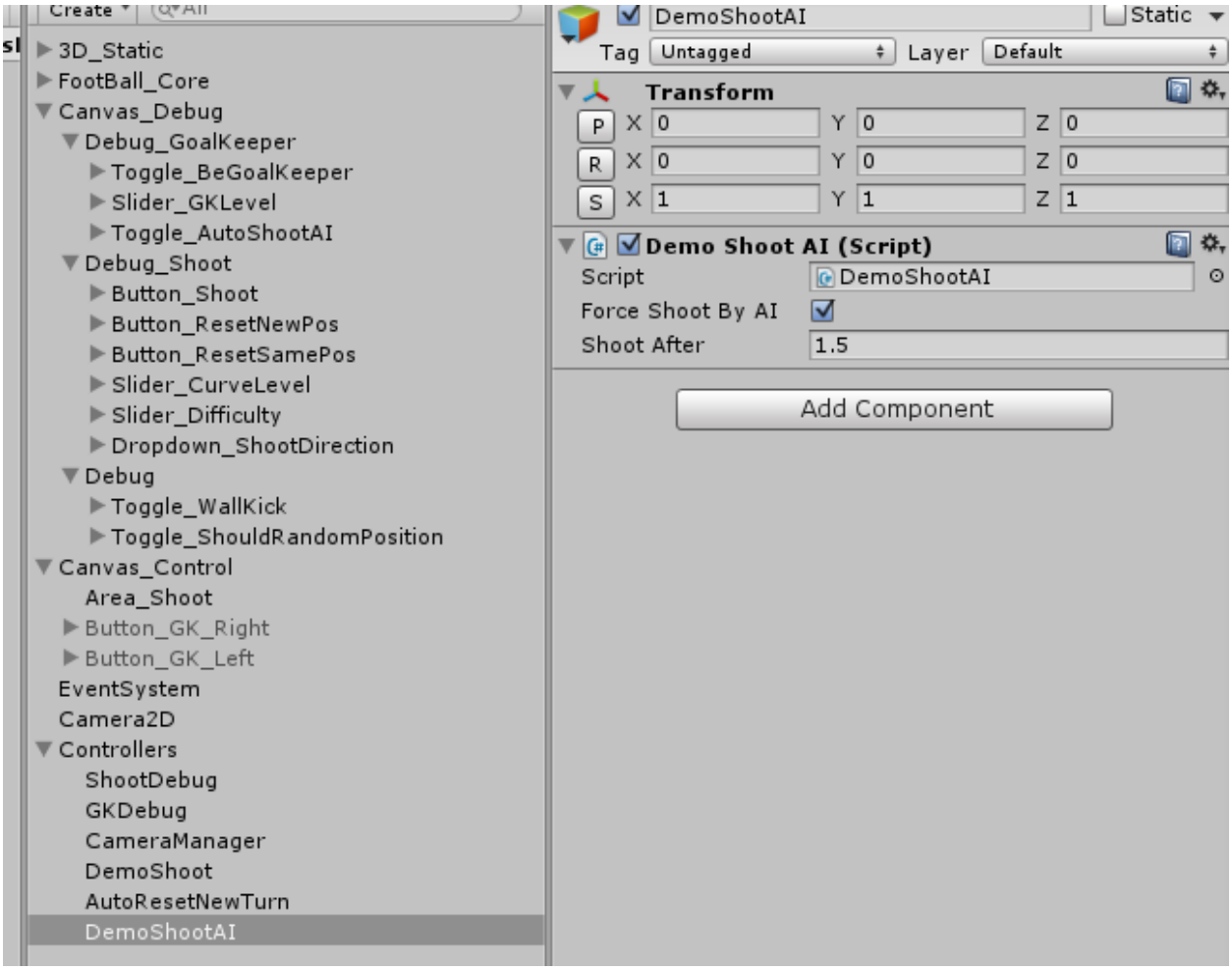

3D_Static
FootBall_Core
Canvas_Debug
Debug_GoalKeeper
Toggle_AutoShootAI
Slider_GKLevel
Debug_Shoot
SliderBallZ
SliderBallX
Button_Shoot
Button_ResetNewPos
Button_ResetSamePos
Debug
Toggle_WallKick
Toggle_ShouldRandomPosition
Canvas_Control
Button_GK_Right
Button_GK_Left
EventSystem
Camera2D
Controllers
ShootDebug
GKDebug
CameraManager
DemoGoalKeeper
DemoShoot
AutoResetAfterShoot
Tag
Untagged
Layer
UI
Rect Transform
custom
Left
Top
Pos Z
0
0.0022505
0
Right
Bottom
0
-0.0022505
stretch
Anchors
Min
X 0
Y 0
Max
X 0.5
Y 1
Pivot
X 0.5
Y 0.5
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Canvas Renderer
Image (Script)
Source Image
Background
Color
Material
None (Material)
Raycast Target
Image Type
Sliced
Fill Center
Event Trigger (Script)
Pointer Down (BaseEventData)
Runtime Only
GKHorizontalFlyDebug.OnClick_GKMoveL
GKDebug (
Pointer Enter (BaseEventData)
Runtime Only
GKHorizontalFlyDebug.OnClick_GKMoveL
GKDebug (
Add New Event Type
Add Component

18. Demo 4: Shoot AI

Popup AI Shoot Direction: this allow to control the direction, whether ShootAI will shoot the ball the left only, right only or both

Slider Curlevel: the higher the value, the higher curve the ball path is

Slider Difficulty: the higher the value, the higher probability that the ball will go exactly into 2 corners of the goal.



DemoShootAI.cs: in the Start() method

```
IEnumerator Start()
    {
        yield return new WaitForEndOfFrame();       // wait for the execution of Start
method of DemoShoot

        GoalKeeperHorizontalFly.share.IsAIControl = true;
        ShootAI.shareAI.willBeShootByUser = false;
        SwitchCamera.share.OnToggle_BeGoalKeeper(false);
    }
```

Initially set the goal keeper to be control by AI, this can be optional, either you or AI can control the GoalKeeper

Tell ShootAI that AI will take the shoot also, this is a must to have the shoot performed by AI

Ask the camera to be front camera.

```
  void Awake()
    {
        AutoShoot = forceShootByAI;
        Shoot.EventDidPrepareNewTurn += OnNewTurn;
    }
```

We listen to event from Shoot.cs to know when new turn has been prepared.

```
  void OnNewTurn()
    {
        RunAfter.removeTasks(gameObject);

        if (AutoShoot)
        {
            RunAfter.runAfter(gameObject, () =>
            {
                ShootAI.shareAI.shoot();
            }, shootAfter);
        }
    }
```

ShootAI.shareAI.shoot() is the exact meat of code that ask ShootAI to perform the shoot.

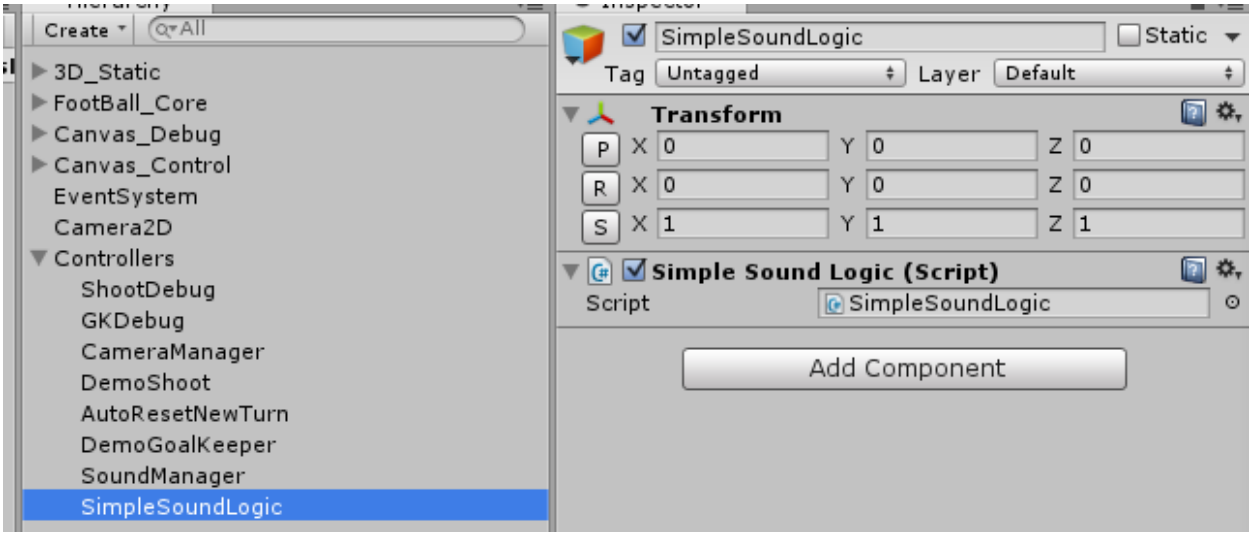
RunAfter.cs is a very little class that help me to run some action after a period of time.

RunAfter.removeTasks() is called to make sure all delay action that is currently ongoing regarding this gameobject will be cancelled or stopped.

19. Demo 5: composition of demo 1 to 4

This demo contains all the feature, debug UIs from all 4 demo, beside it has a button (in red rectangle) that allow us to switch camera angle.

20. Demo 6: Simple sound logic



SimpleSoundLogic.cs: this implement a very simple logic of sound based upon listening specific events from core components.

```
void Start()
    {
        Shoot.share.eventShoot += OnShoot;
        GoalKeeper.EventBallHitGK += OnBallHitGK;
        GoalDetermine.share.eventFinishShoot += OnGoal;
    }

    void Destroy()
    {
        Shoot.share.eventShoot -= OnShoot;
        GoalKeeper.EventBallHitGK -= OnBallHitGK;
        GoalDetermine.share.eventFinishShoot -= OnGoal;

    }
```

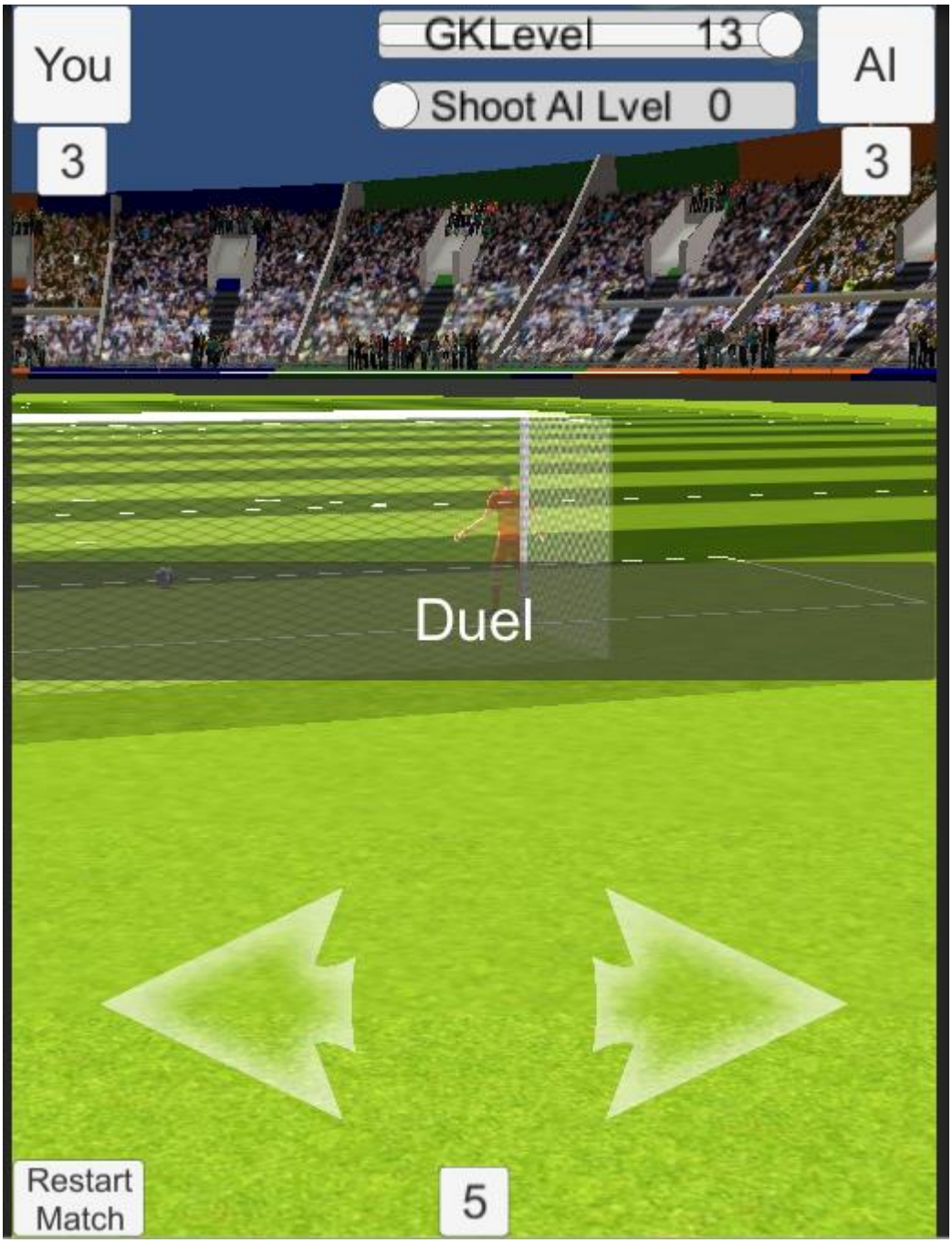
eventShoot: will be fired when the ball has just been kicked

EventBallHitGK: will be fired when the ball hit the goalkeeper

eventFinishShoot: will be fired when GoalDetermine finish its evaluation of detecting a goal.

Based on these events, the corresponding listener method will play a suitable sound effect.

21. Demo 7: Simple Match

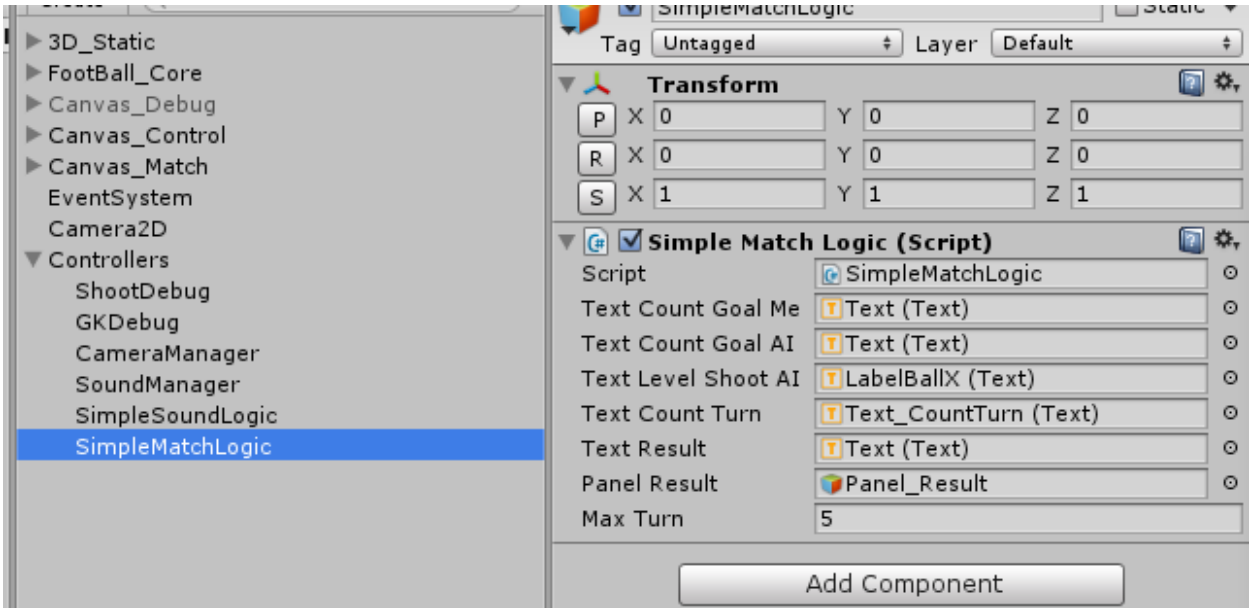


This demo implement a very simple logic of a competition match.

Each player to have 5 turn of being shooter and goalkeeper, the one who score more goals after 5 turns will win.

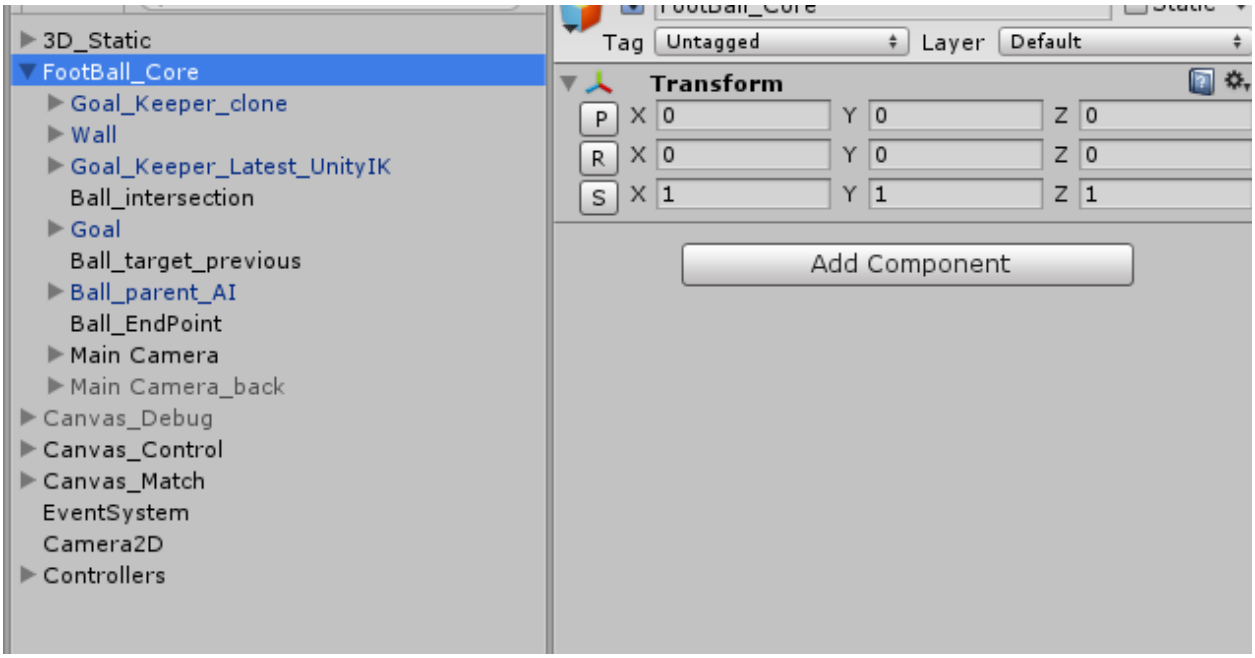
Slider GKLevel: this adjust the level of goalkeeper, the higher the more superman the goalkeeper is.

Slider ShootAI level: this adjust the difficulty of the ShootAI mechanism, the higher the better shoot.



SimpleMatchLogic.cs: implement the logic of a simple football match, count the score of 2 player, check the winner at the end. Reset next turn, make freekick and wallkick turn after turn.

22. So how to use this template



Just copy FootBall_Core to your scene and place it at the origin (0, 0, 0), then you are good to call methods of core components and listen to their events to control the game flow as you like, when the ball is kick, when I know there is a goal or not, when the ball hit something, etc…

23. Useful events

GoalDetermine.cs:

```
public static GoalEvent EventFinishShoot;
```
This event is fired when there is decision about whether there was a goal or not

GoalKeeper.cs:

```
public static Action EventBallHitGK
```
: This event is fired when the ball hit the goalkeeper

GoalKeeperHorizontalFly.cs:

`public static Action<bool> EventChangeIsAIControl` : this event is called when there is a change in goalkeeper control permission, “true“ means AI will control the GK, “false” means user will control the ball

GoalKeeperLevel.cs:

`public static Action<LevelGK, int> EventChangeLevel` : this event is fired when goal keeper change his level.

Shoot.cs:

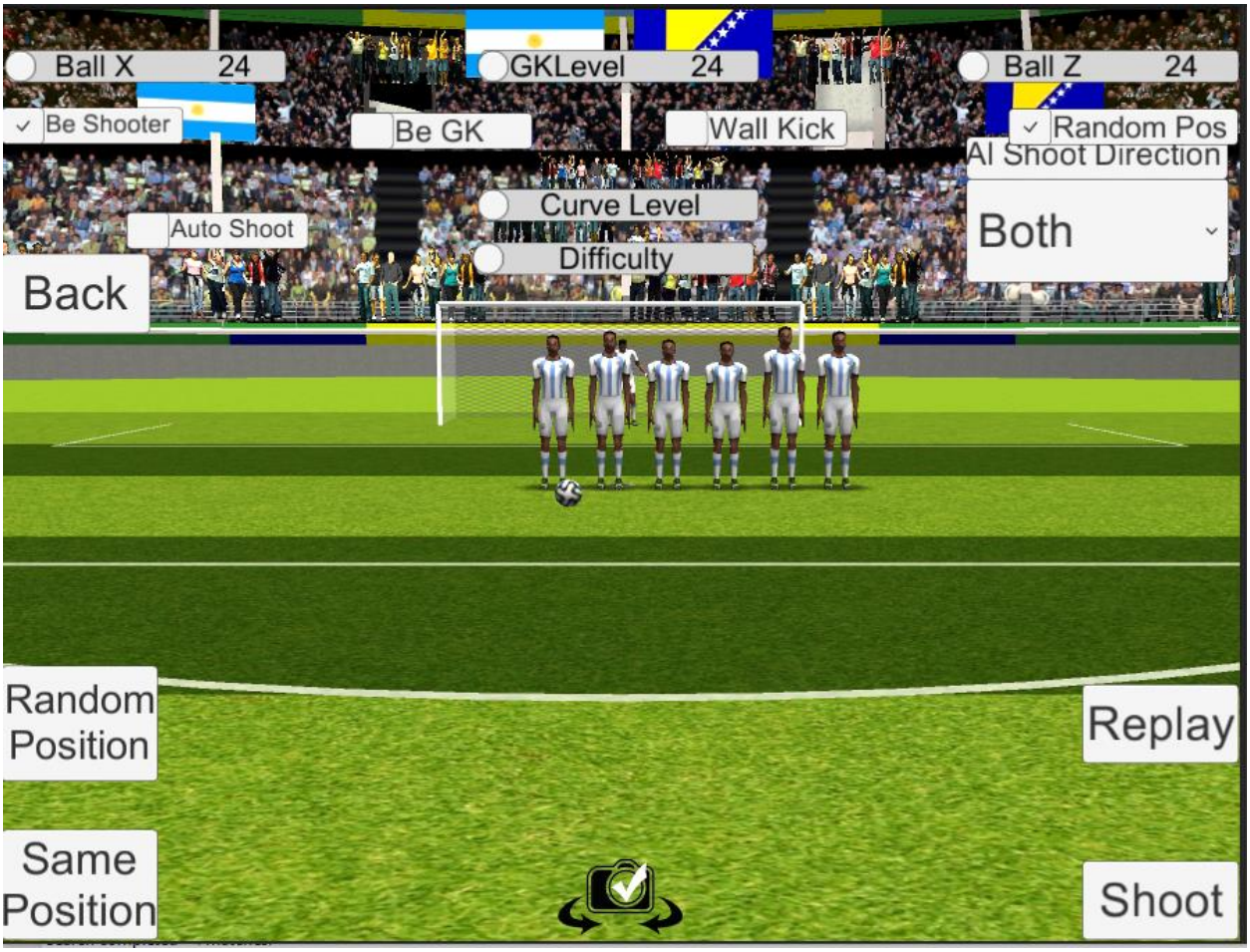
`public static Action EventShoot` : this event is fired after the ball is shoot

`public static Action EventDidPrepareNewTurn` : this event is fired after the ball is prepared for the next turn

`public static Action<Collision> EventOnCollisionEnter` : this event is fired when there is collision between the ball and something else

- Version 1.0.1

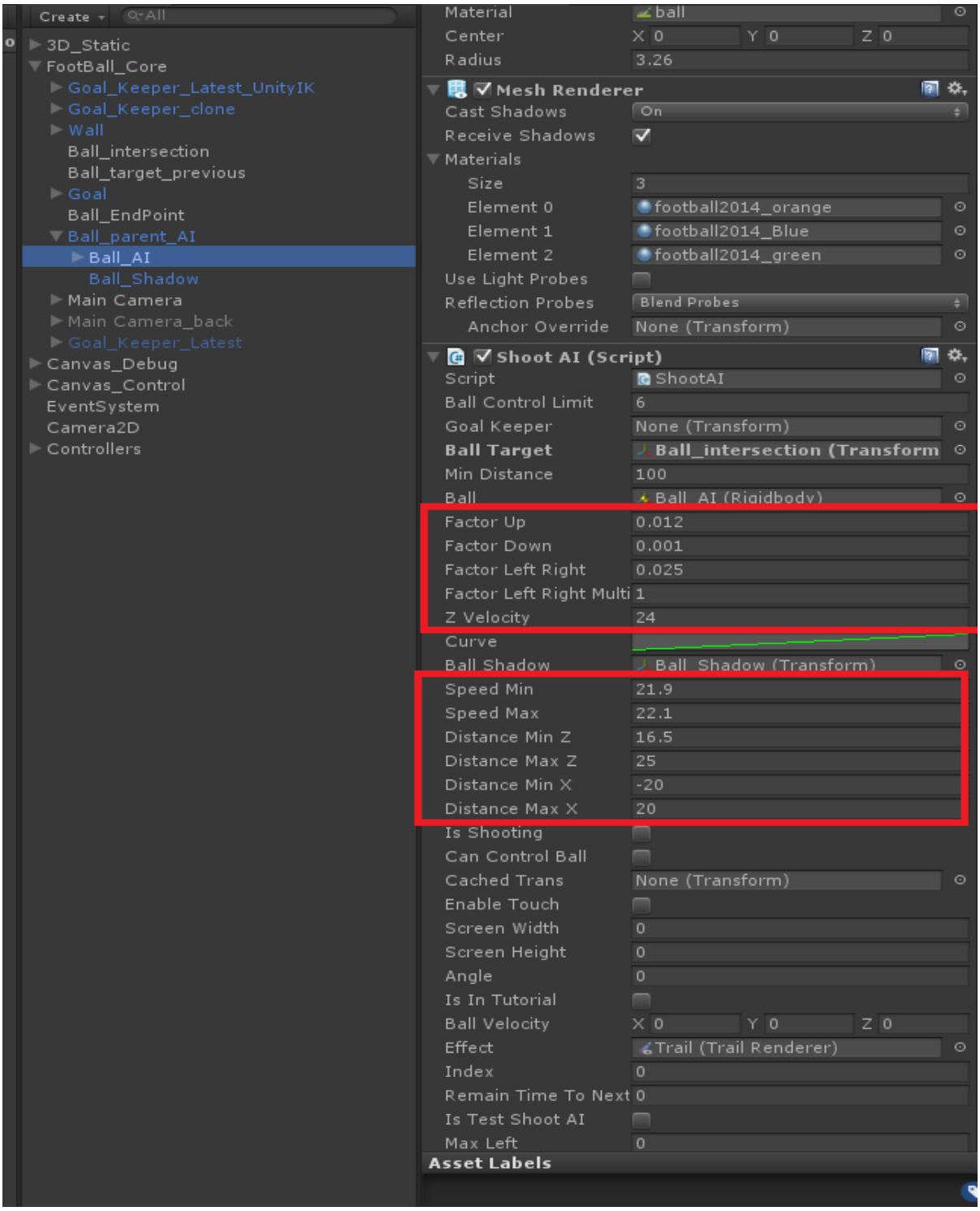
1. Demo 8: Recording gameplay



This is the new scene demo to show the Recording function.

After make a shoot, you can click Replay button to watch the replay.

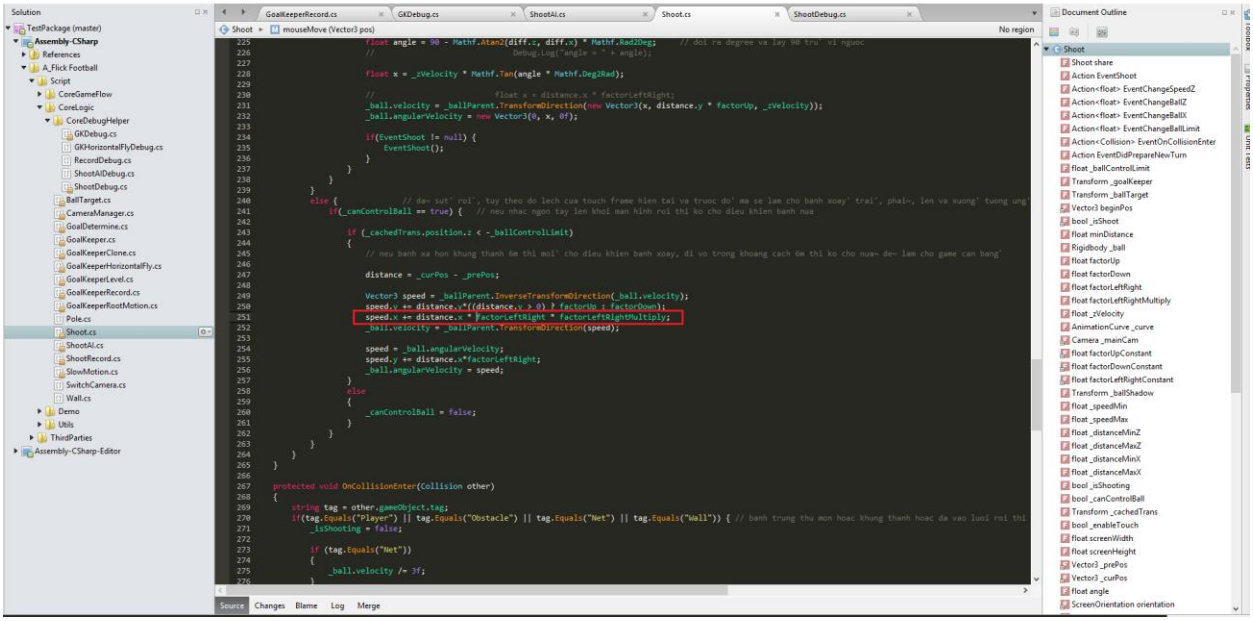
2. Adjust public variables for landscape mode



From the picture below, we can see numbers of variables bounded in red rectangles.

You can play with them to find the unique shoot experience of your own.

For the sake of changing the shoot mechanism to landscape mode, “Factor Left Right Multiply” was modified from 0.7 to 1.0.



The reason is landscape view has bigger width, so it take longer time for the finger to be able to move from the left side to the right side of the screen, so “Factor Left Right Multiply” is set from 0.7 to 1.0 to help user gain a better sensitivity of controlling the ball to the left/right on the fly.

The 3 factors “Factor Up”, “Factor Down”, “Factor Left Right” were set to balance each other in portrait mode, so I prefer to keep them as is and change only “Factor Left Right Multiply”.

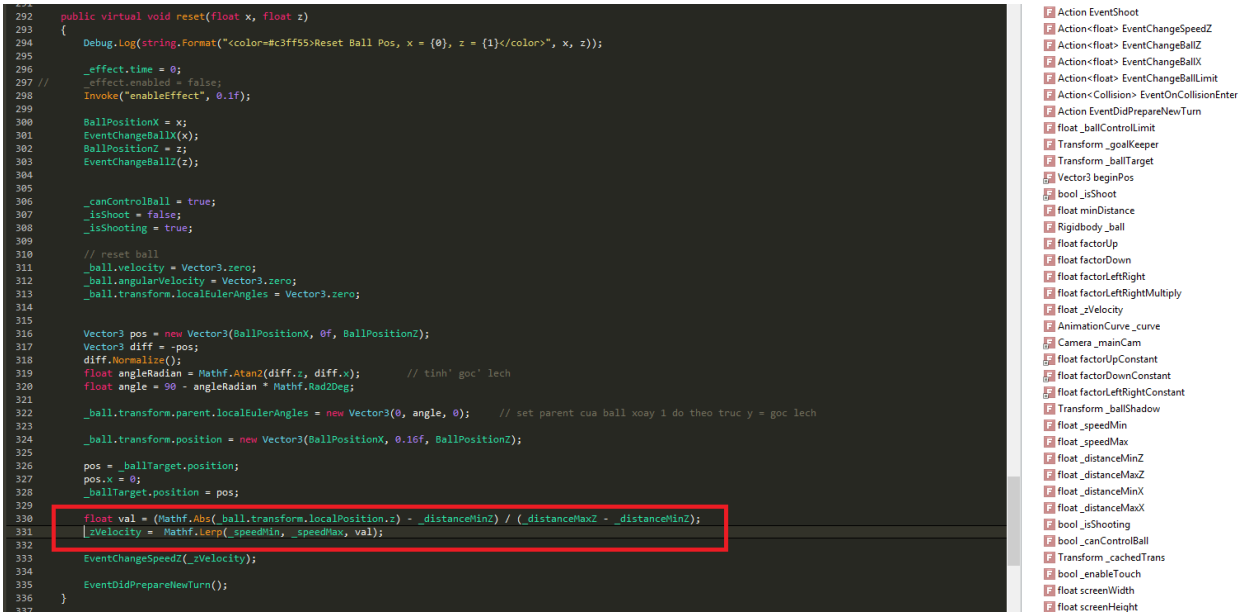
It’s up to you if you want to modify these factors to create unique shoot experience.

There are other factors like:

Distance min Z -> Distance max Z: the z of ball’s position will be randomized in this range

Distance min X -> Distance max X : the x of ball’s position will be randomized in this range

Speed min -> speed max: the speed of the ball along its forward direction (_zVelocity variable) will be interpolated form speed min -> speed max according to the z of ball’s position.



```
public virtual void reset(float x, float z)
{
    Debug.Log(string.Format("<color=#c3ff55>Reset Ball Pos, x = {0}, z = {1}</color>", x, z));

    _effect.time = 0;
//      _effect.enabled = false;
    Invoke("enableEffect", 0.1f);

    BallPositionX = x;
    EventChangeBallX(x);
    BallPositionZ = z;
    EventChangeBallZ(z);


    _canControlBall = true;
    _isShoot = false;
    _isShooting = true;

    // reset ball
    _ball.velocity = Vector3.zero;
    _ball.angularVelocity = Vector3.zero;
    _ball.transform.localEulerAngles = Vector3.zero;


    Vector3 pos = new Vector3(BallPositionX, 0f, BallPositionZ);
    Vector3 diff = -pos;
    diff.Normalize();
    float angleRadian = Mathf.Atan2(diff.z, diff.x);        // tinh' goc' lech
    float angle = 90 - angleRadian * Mathf.Rad2Deg;

    _ball.transform.parent.localEulerAngles = new Vector3(0, angle, 0);     // set parent cua ball xoay 1 do theo truc y = goc lech

    _ball.transform.position = new Vector3(BallPositionX, 0.16f, BallPositionZ);

    pos = _ballTarget.position;
    pos.x = 0;
    _ballTarget.position = pos;

    float val = (Mathf.Abs(_ball.transform.localPosition.z) - _distanceMinZ) / (_distanceMaxZ - _distanceMinZ);
    zVelocity =  Mathf.Lerp(_speedMin, _speedMax, val);

    EventChangeSpeedZ(_zVelocity);

    EventDidPrepareNewTurn();
}
```

```
Action EventShoot
Action<float> EventChangeSpeedZ
Action<float> EventChangeBallZ
Action<float> EventChangeBallX
Action<float> EventChangeBallLimit
Action<Collision> EventOnCollisionEnter
Action EventDidPrepareNewTurn
float _ballControlLimit
Transform _goalKeeper
Transform _ballTarget
Vector3 beginPos
bool _isShoot
float minDistance
Rigidbody _ball
float factorUp
float factorDown
float factorLeftRight
float factorLeftRightMultiply
float _zVelocity
AnimationCurve _curve
Camera _mainCam
float factorUpConstant
float factorDownConstant
float factorLeftRightConstant
Transform _ballShadow
float _speedMin
float _speedMax
float _distanceMinZ
float _distanceMaxZ
float _distanceMinX
float _distanceMaxX
bool _isShooting
bool _canControlBall
Transform _cachedTrans
bool _enableTouch
float screenWidth
float screenHeight
```

3. Shoot Record

Capture ball position frame by frame, replay ball position frame by frame

4. Goalkeeper record

Capture goalkeeper animation, capture Time.deltatime at the time of recording and use that data for playing back the animation.