程式備忘錄

吃子彈修正

目錄

為何做備忘錄

1.快速回憶

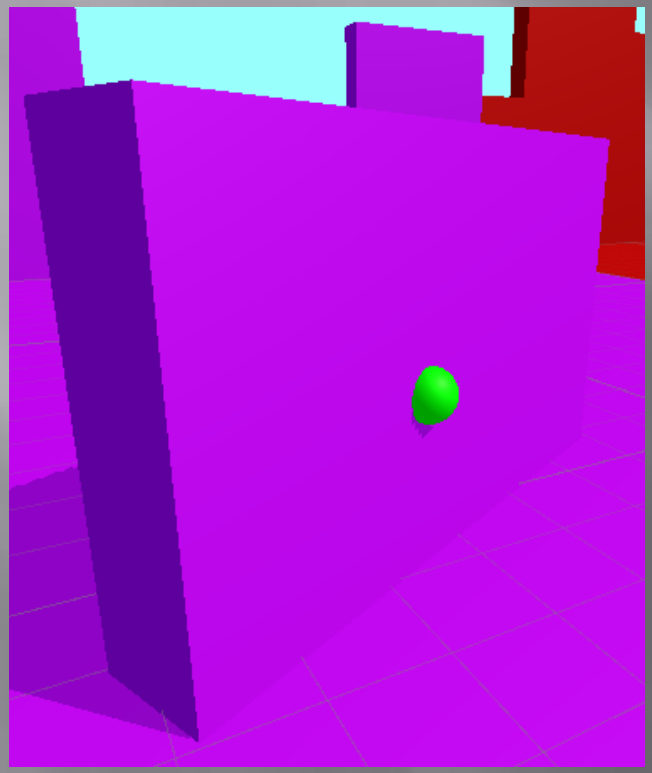
- 誰記得3個月前寫東西(又超複雜)?快速讀懂過去的程式好除錯或添加(Unet就沒做，又要重學，痛定思痛)。

2.自主學習成果

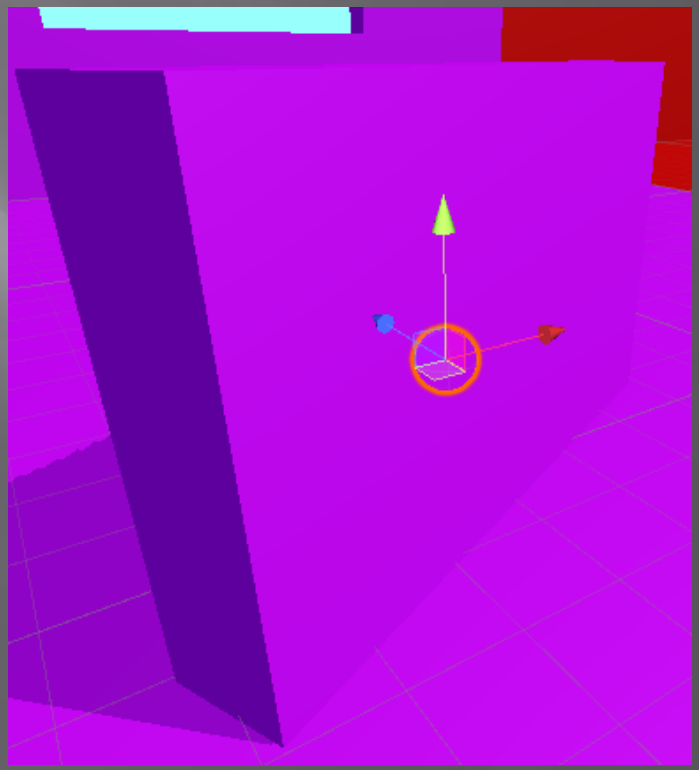
- 如果沒有2.其實不太想寫這個，就繼續前進就對了，反正都除錯完了，而且我都把思考過程、注意事項與曾犯錯誤註解的超清楚，在原代碼前沒有祕密。

要修正什麼

- 子彈撞牆，速度設為零後:



預期(停牆上)



實際(陷進去了)

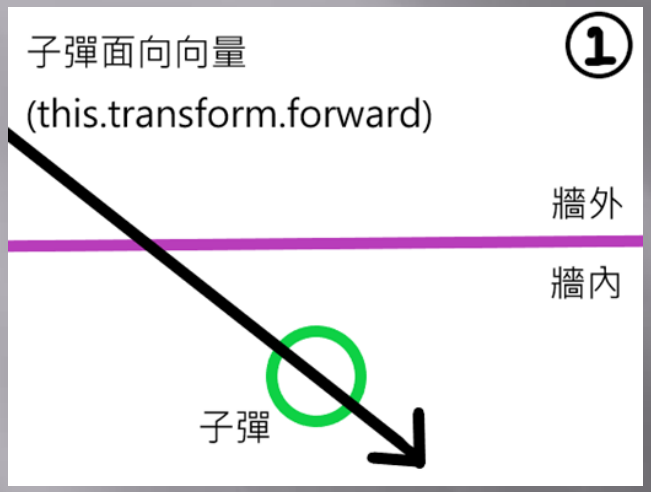
- 修正之路就開始了

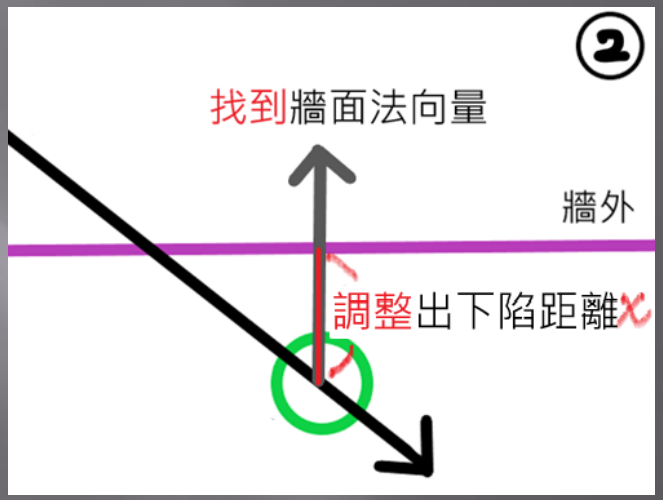
製作過程
(以重大突破分類)

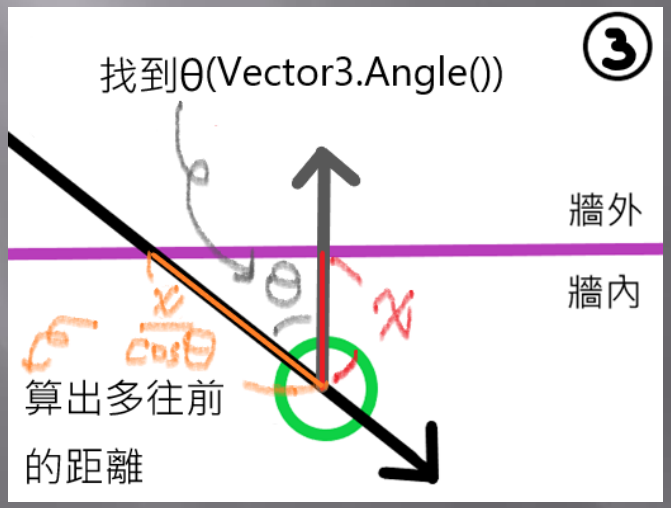
第一版

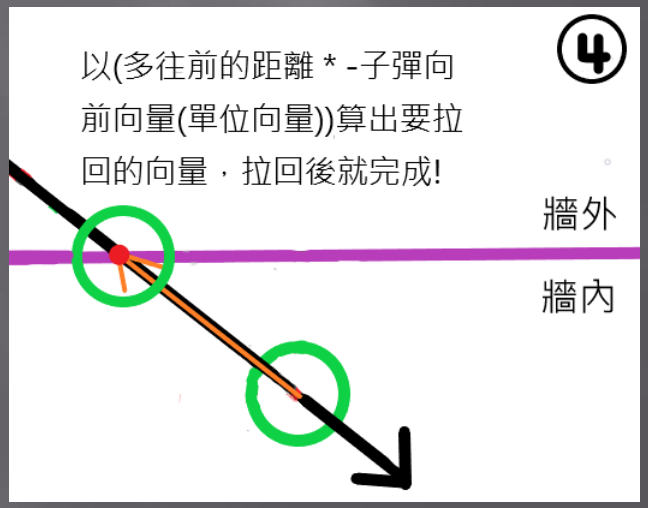
第一版

- 想到物理解運動學最常用的Cos，概念如下：







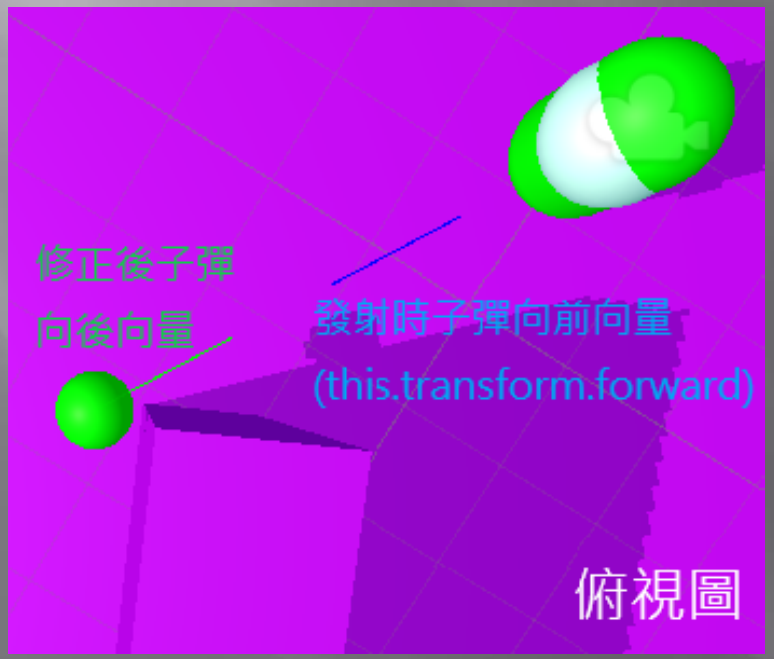


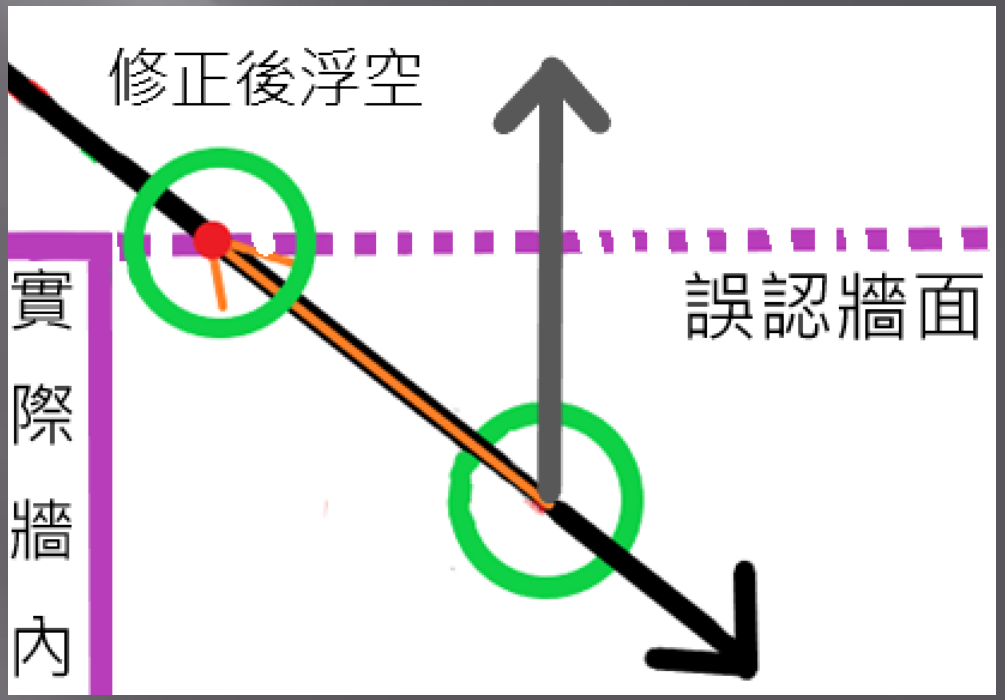
第一版

- 第五周想到，第六周除錯完(只有除程式錯誤，概念錯誤沒有)，我稱之為**拉回修正**！
- 但是!!看完後一定有許多問題，執行下來也有不少問題，等一下一起解釋
- Ps.此程式碼已流失，只有程式片段、文字紀錄與圖片草稿保留，所以以下有原始碼處是之後模擬而成(只有模擬吃子彈部分)，而且我也不確定所有觀念都是此時想到的，但我會盡力回想的!

第一版。問題

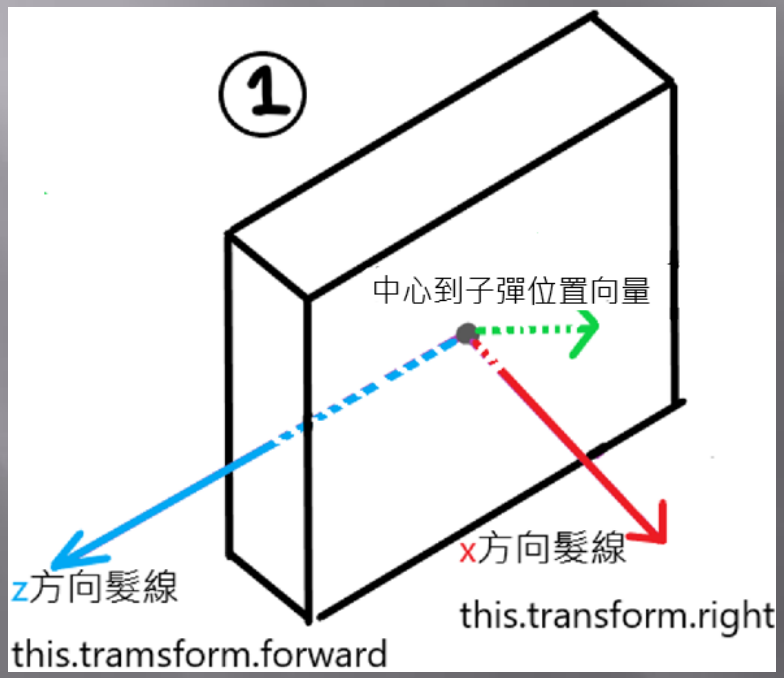
- **Q :** 有測試出什麼問題嗎?
- **Ans :** 有!先看左圖，明明正常運作，為何會懸空?其實是演算法缺失!如右圖，就算此時拉回修正完全符合計畫執行(實際在法向量可能會找錯，參下一頁)，當射向牆角時仍會發生，我稱其**碰邊邊**!此時是直接讓子彈繼續飛(再加一次力)以解決

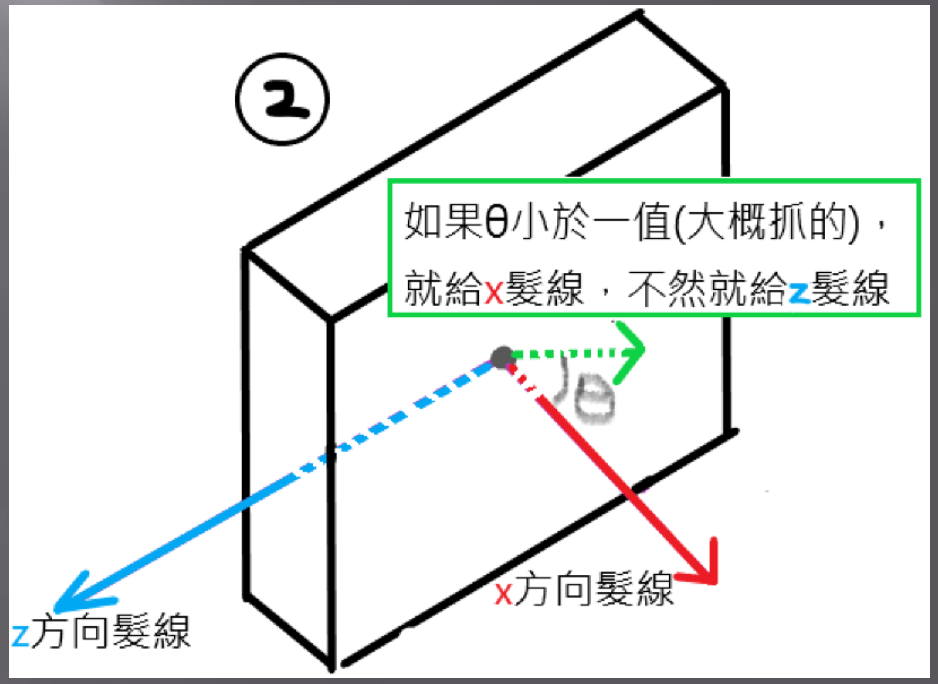




第一版。問題

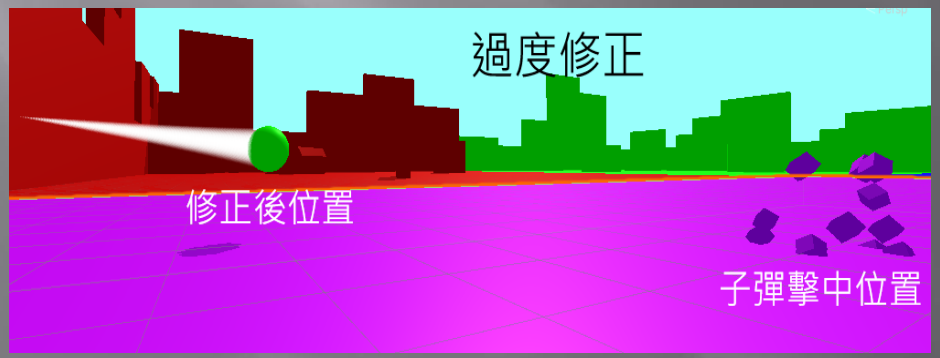
- **Q :** 如何找到牆面法向量?
- **Ans :** 此時，我只想到打地板就用(0f ,1f ,0f)為法向量，打牆壁就判斷(如下)打在x或z方向給牆面法向量 ，其他處仍想不到，所以目前只適用在方形碰撞體，且沒考慮到y方向，非常不準，也沒有通用性





第一版。問題

- **Q :** 調整出下陷距離，那麼隨便?
- **Ans :** 老實說，我真的回憶不起那時候是怎麼想的，我只找到唯一的概念圖，上面寫下陷距離為1，右邊還有式子以1為下陷距離去計算拉回修正，所以當時應該是認為下陷距離為1，原因不明。但是!由於實際下降高度不定且通常<1，所以重現原代碼時發現修正極為不準，且會重複觸發碰邊邊後再次碰撞，因此推測當時因該是以手動調整出下陷距離解決



第一版。原始碼

- 宣告 + HasFirstEnterCollider(有無碰過Collider並未離開，怕一次碰撞到2個物件而出錯，順便用於碰邊邊判斷)

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5
6   public class OfflineBullet : MonoBehaviour
7   {
8       public GameObject LittleCrack;
9       public GameObject Explosion;
10      public GameObject AttackHotExplose;
11      public GameObject BulletDestory;
12
13      private float StartTime;
14      private float FlyTime;
15
16      private bool HasFirstEnterCollider = false;
17
18      private Vector3 ReflectVector;
19
20      private void OnTriggerEnter(Collider other)
21      {
22          if (!HasFirstEnterCollider)
23          {
24              HasFirstEnterCollider = true;
25              //只執行一次ColliderEnetr(可能一次碰到多個Collider)
```

第一版。原始碼

- 粒子特效

```
28          switch (other.tag)
29          {
30              case "BattleField":
31                  GameObject NewLittleCrack = Instantiate(LittleCrack, this.transform.position, other.transform.rotation);
32                  NewLittleCrack.GetComponent<Renderer>().material.color = other.GetComponent<Renderer>().material.color;
33                  Destroy(NewLittleCrack, 0.8f);
34                  break;
35              case "Player":
36                  GameObject NewExplosion = Instantiate(Explosion, this.transform.position, Quaternion.identity);
37                  NewExplosion.GetComponentsInChildren<Transform>()[1].GetComponent<Renderer>().material.color = other.GetComponent<Renderer>().material.color;
38                  Destroy(NewExplosion, 0.8f);
39                  break;
40              case "AttackStuff":
41                  GameObject NewAttackHitExplose = Instantiate(AttackHotExplose, (this.transform.position + other.transform.position) / 2, Quaternion.identity);
42                  Destroy(NewAttackHitExplose, 0.8f);
43                  break;
44          }
45          //粒子特效
```

第一版。原始碼

- 停子彈 + 找法向量

```
50          switch (other.tag)
51          {
52              case "BattleField":
53                  this.GetComponent<Rigidbody>().velocity = new Vector3(0f, 0f, 0f);//停子彈
54
55                  if (other.name == "Floor")
56                      ReflectVector = new Vector3(0f, 1f, 0f);
57                  else
58                  {
59                      Vector3 WallToBullet = this.transform.position - other.transform.position;
60
61                      float ReflectAngle = Vector3.Angle(WallToBullet, other.transform.right);
62                      if (ReflectAngle > 90)
63                          ReflectAngle = 180 - ReflectAngle;
64                      if (Vector3.Angle(WallToBullet, other.transform.right) <= 80)
65                          ReflectVector = other.transform.right;
66                      else
67                          ReflectVector = other.transform.forward;
68                  }
69                  //找到牆面法向量
```

第一版。原始碼

- 拉回修正 + 呼叫碰邊邊函式
- Ps. 為了不讓圖片太長，犧牲了一些註釋，抱歉…

```
73                  float CalculAngle = Vector3.Angle(this.transform.forward, ReflectVector);//找到牆面法向量與子彈向
74                  if(CalculAngle > 90)
75                      CalculAngle = 180 - CalculAngle;//去除廣義角
76                  Vector3 BulletMoveForward = this.transform.forward * 0.2f / Mathf.Cos(CalculAngle / 180 * 3.14f);
77                  this.transform.position = this.transform.position - BulletMoveForward;//吃子彈修正完成
78
79                  DetactBulletOverFly();//確認卡子彈調整後是否不在牆上了
80                  break;
81              }
82          }
83      }
```

第一版。原始碼

- 解除HasFirstEnterCollider + 碰邊邊函式(OverFly)

```
87      private void OnTriggerExit(Collider other)
88      {
89          HasFirstEnterCollider = false;
90      }
91      //離開Collision回復HasFirstEnetrCollider
92
93      public void DetactBulletOverFly()
94      {
95          StartCoroutine(OverFliedFixed());
96      }
97      IEnumerator OverFliedFixed()
98      {
99          yield return new WaitForSecondsRealtime(0.05f);
100
101         if (!HasFirstEnterCollider)
102             this.GetComponent<Rigidbody>().AddForce(this.transform.forward * 1500f);
103         else
104             Destroy(this.GetComponentsInChildren<Transform>()[1].GetComponent<ParticleSystem>());
105     }
106     //OverFly時就繼續飛，回復原本模式
```

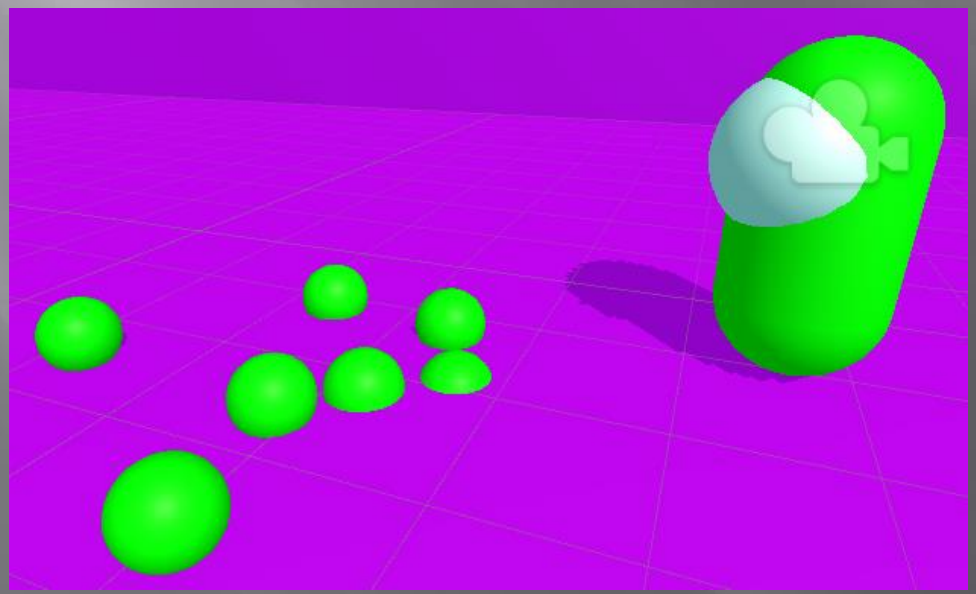
第一版。原始碼

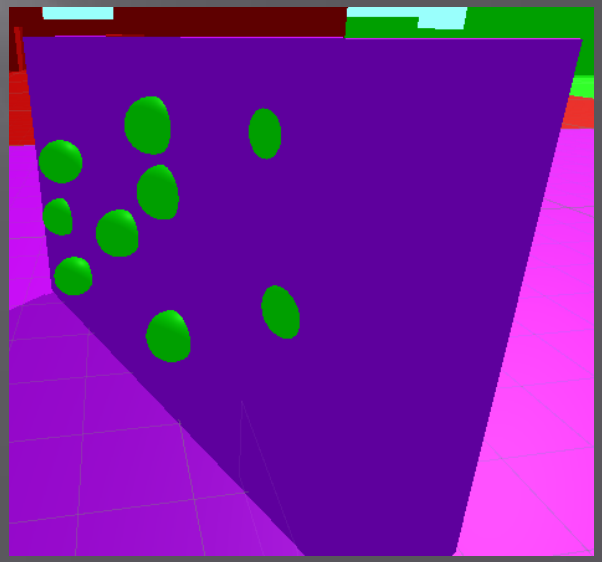
- 子彈飛行時間判斷 + 子彈摧毀粒子特效

```
109     private void Start()
110     {
111         StartTime = Time.time;
112     }
113     private void Update()
114     {
115         FlyTime = Time.time - StartTime;
116 
117         if(Time.time - StartTime > 5f)
118         {
119             if (HasFirstEnterCollider)
120                 Destroy(this.gameObject);
121         }
122 
123         if (Time.time - StartTime > 20f)
124             Destroy(this.gameObject);
125         //預防射向天空，要一直計算子彈，不會被消滅
126     }
127     //計算FlyigTime
128 
129     private void OnDestroy()
130     {
131         GameObject NewLittleCrack = Instantiate(BulletDestory, this.transform.position, this.transform.rotation);
132         NewLittleCrack.GetComponent<Renderer>().material.color = this.GetComponent<Renderer>().material.color;
133         Destroy(NewLittleCrack, 0.8f);
134     }
135 }
```

第一版。小結

- 執行結果如下兩圖，真的非常不準阿！雖然說是第一次自行解決問題(第一次在網路上查不到有一樣問題的解決方法)，能想到這樣有一點成就感，但花了時間有點多了，而且有許多不確定性。因此沒有休息，繼續思考更好的解決方法，目標是達到 精準 + 通用 兼具的程式！





第二版

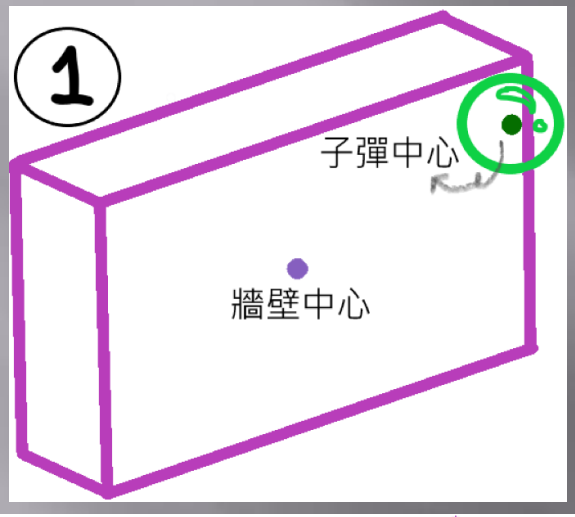
第二版。前言

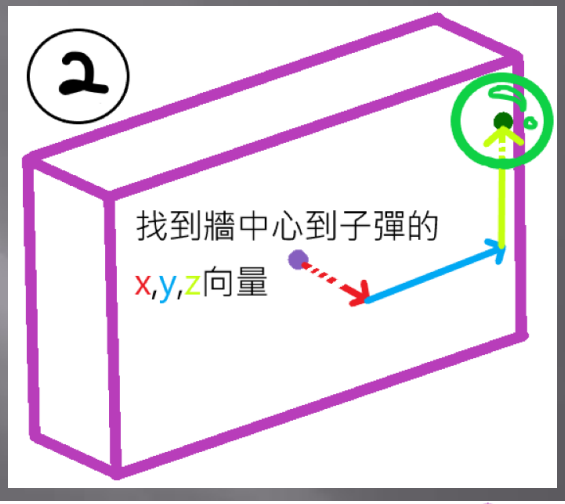
- 要做到第二版觀念前，必須知道Collision與Trigger的差別才行，因為第一版以Trigger為主的碰撞，第二版則以Collision為主的碰撞，看下圖後:

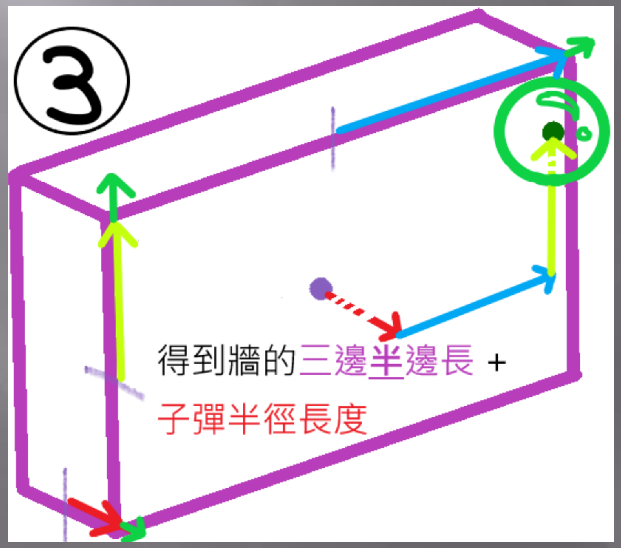
- 可知Collision是有碰撞反應(Unity的物理運算)，才會沒陷入牆中，但Trigger就沒有物理運算了
- Ps.子彈有反彈材質，才會有右圖的反彈軌跡出現

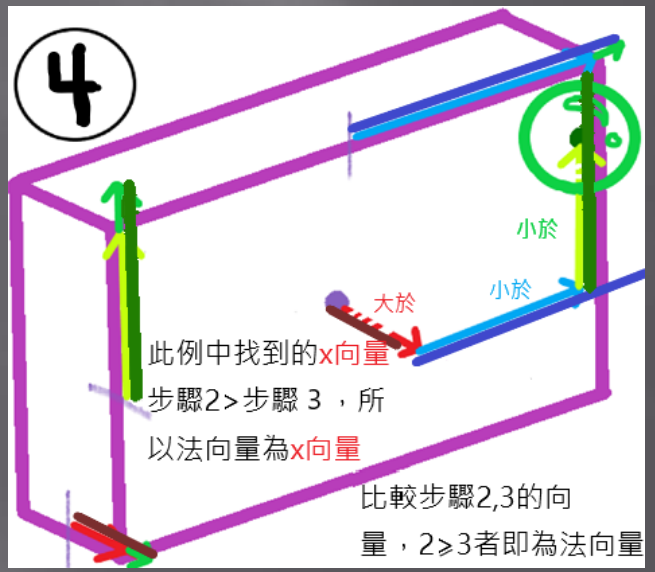
第二版

- 在找法向量上得到改進，概念如下：



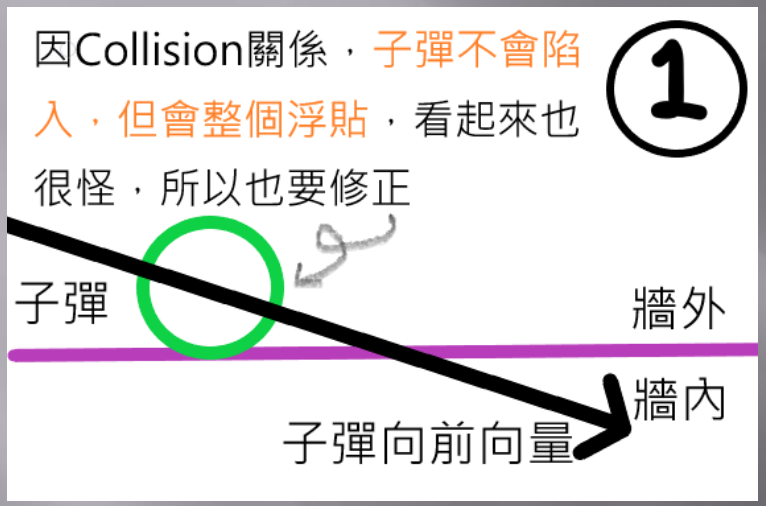


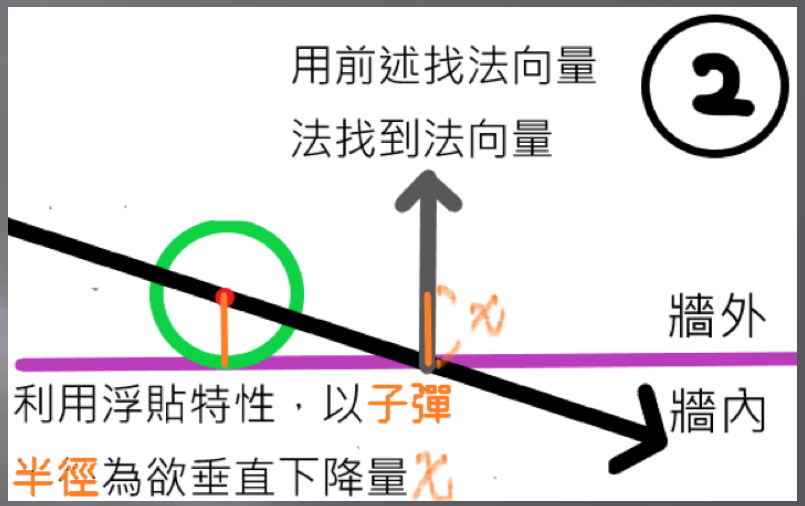


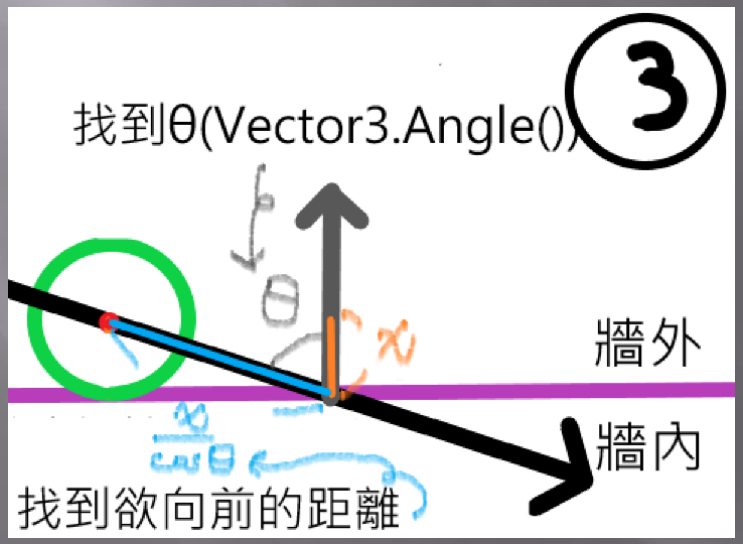


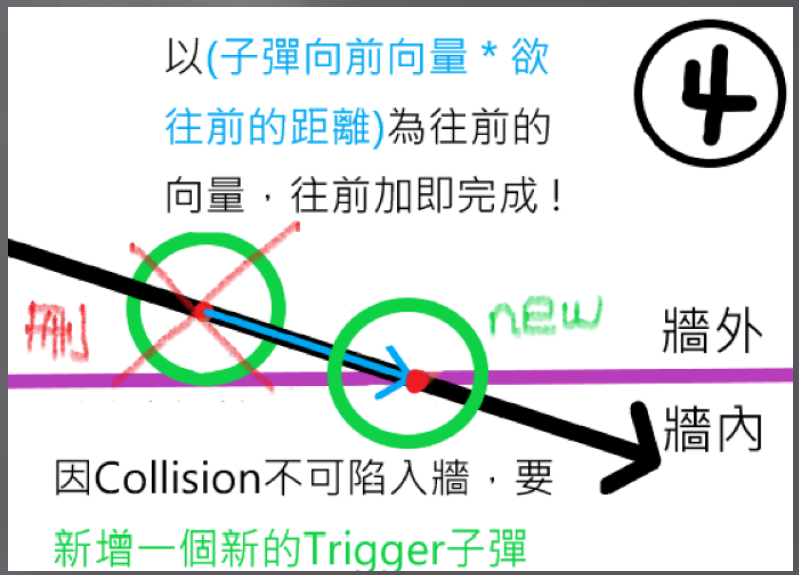
第二版

- 改用Collision後，拉回修正也要修改成Collision版：









第二版

- 第七周想到，第八周除錯完，第九周了解它的不足，因為後續這兩個算法有更好的取代，因此不取名
- 這版相較第一版，無論是精準度(下降高度一致)或通用性(只要是箱形碰撞體皆可)都提升了，概念上也沒太大問題，但是執行後仍有問題，也有一些概念想說明，等一下一次講明白
- Ps.這次就有原始碼了！但是當時仍有製作子彈停下再被打到後會反彈的程式(其實Unity的物理就做得到)，因沒太大關係而刪除，只保留吃子彈程式。

第二版。問題

- **Q :** 為什麼要換到以Collision為主的計算?
- **Ans :** 因為這個法向量找法不允許子彈中心陷入牆內，如果陷入，會導致三邊邊長都>牆中心到子彈中心的x,y,z向量，就會出錯，Trigger為主沒有Unity物理計算，可能發生上述事件 ；再加上Collision版拉回修正中也用到子彈浮貼，可以固定欲下陷距離，讓拉回修正穩定。由上優點，所以換到Collision。

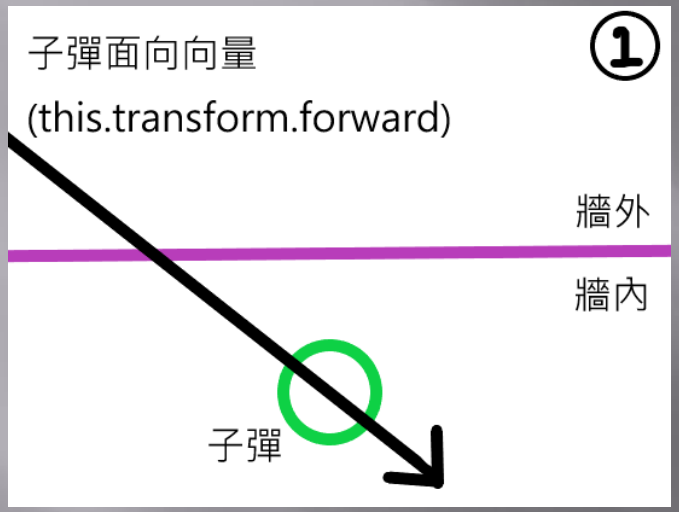
- **Ps.** 其實這個法向量找法是從人類如何分辨一個點在箱子的裡面、x面、y面或z面想到的，我個人是這樣分的啦，我想大部分應該是一樣的吧！

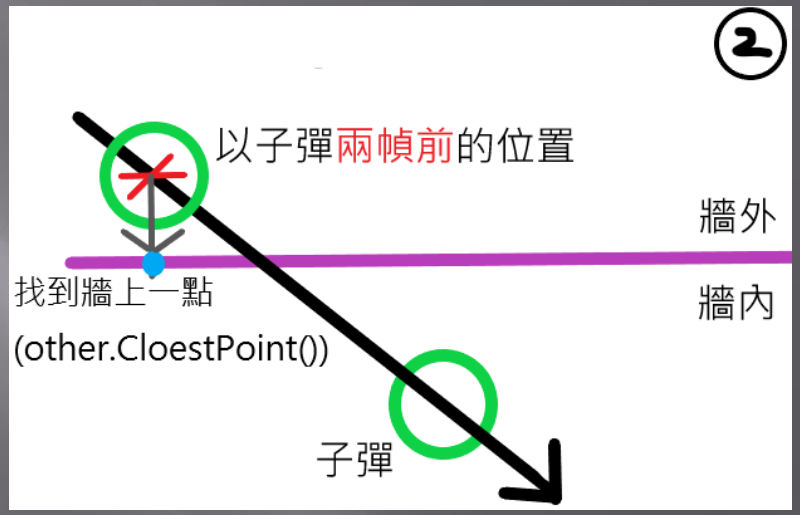
第二版。問題

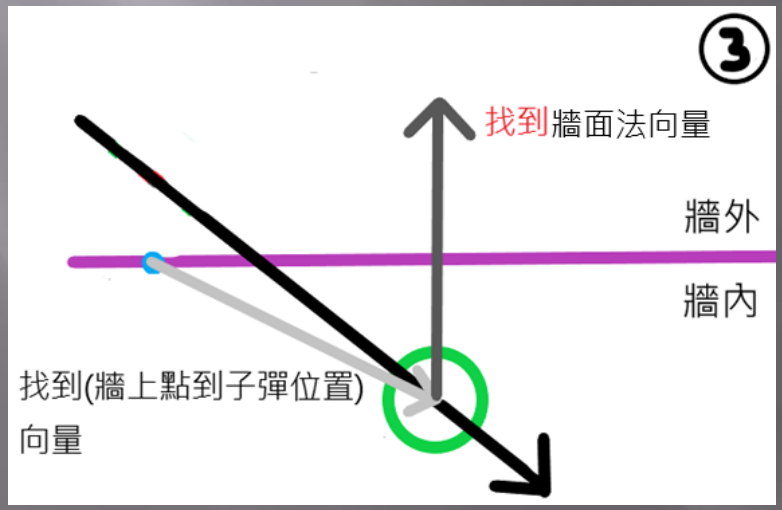
- **Q :** 為什麼找法向量時要加子彈半徑?
- **Ans :** 用2D比較好解釋，看右圖

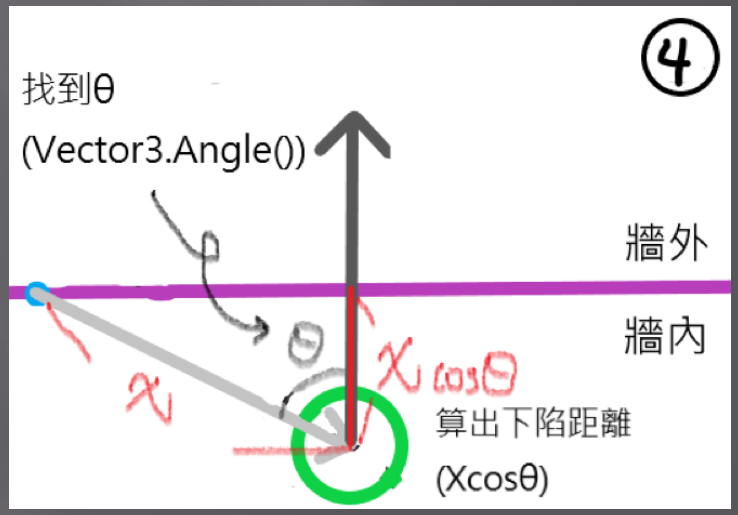
第三版

第三版

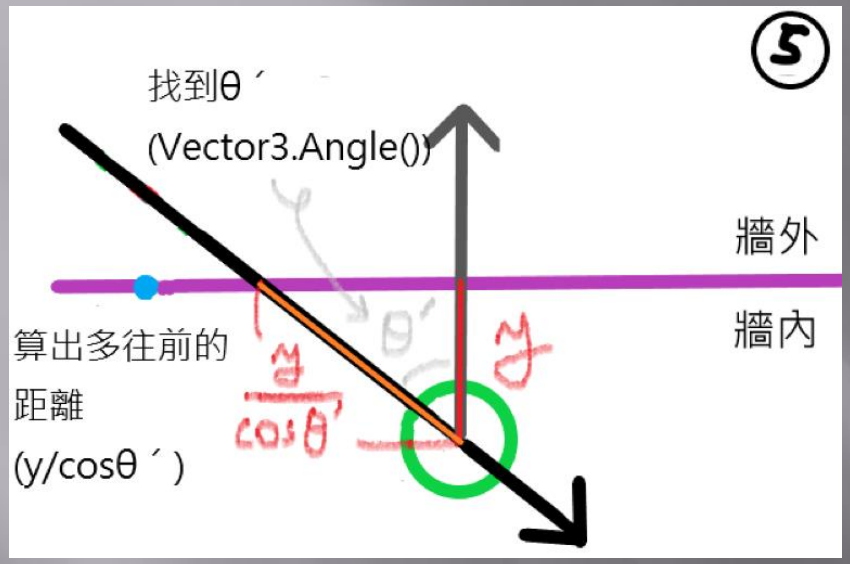

子彈面向向量
(this.transform.forward)
①
牆外
牆內
子彈

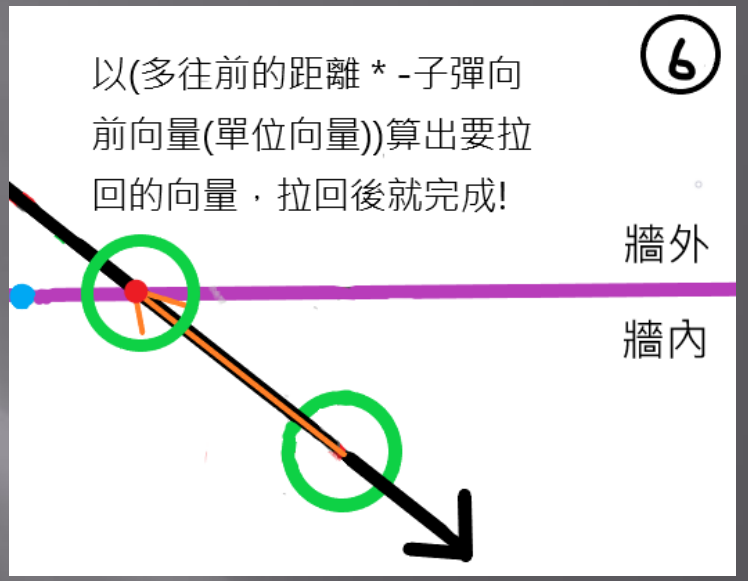

②
以子彈兩幀前的位置
牆外
牆內
找到牆上一點
(other.CloestPoint())
子彈


③
找到牆面法向量
牆外
牆內
找到(牆上點到子彈位置)
向量


④
找到θ
(Vector3.Angle())
牆外
牆內
θ
X
X cosθ
算出下陷距離
(Xcosθ)

第三版

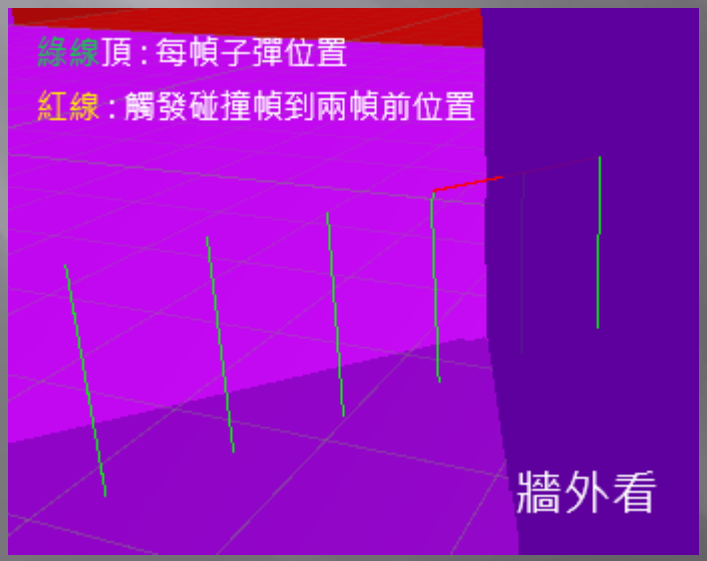

5
找到θ´
(Vector3.Angle())
牆外
牆內
算出多往前的
距離
(y/cosθ´)
y
cosθ´
y

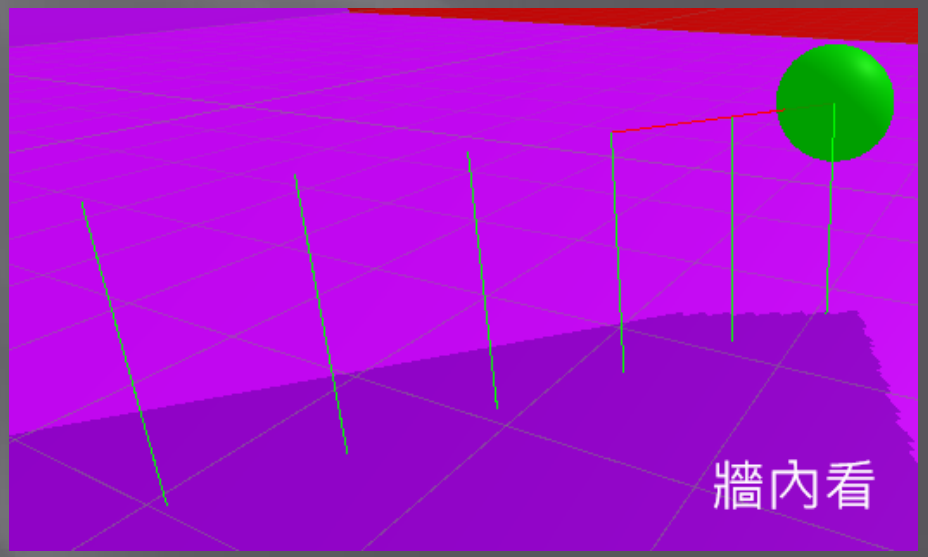

6
以(多往前的距離 * -子彈向
前向量(單位向量))算出要拉
回的向量，拉回後就完成!
牆外
牆內

第三版。問題

- **Q :** 步驟二中，為什麼取兩幀前位置?
- **Ans :** 因為ClosetPoint()給的位置如果在碰撞體內，就不會回傳牆上一點，只會回傳(0f ,0f ,0f)，我們不能確定子彈在牆內or外，但兩幀前必在牆外。
- **Q :** 為什麼兩幀前可以確定不在碰撞體內?
- **Ans :** 實驗結果(兩幀前:未碰撞、一幀前:Unity物理引擎計算、此幀:碰撞偵測，停子彈)　Ps.此處幀皆是FixedUpdate

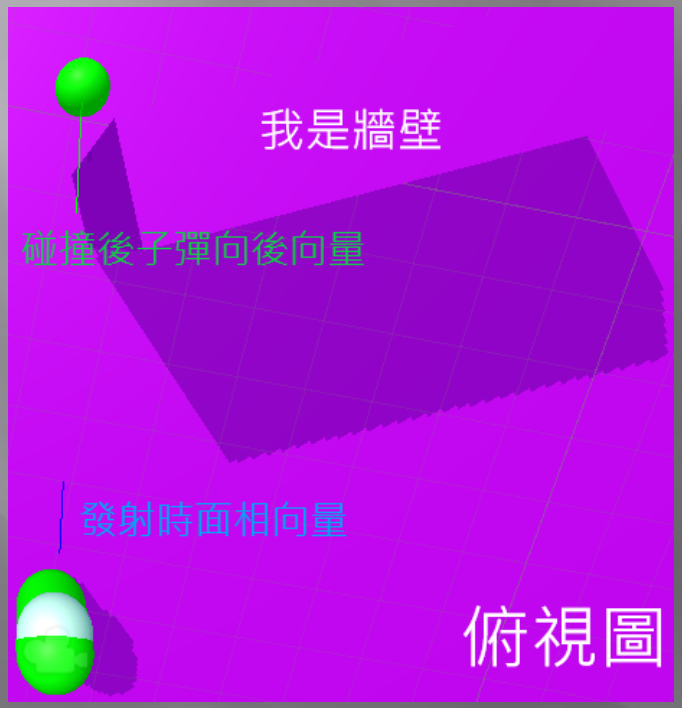
因物理計算皆在FixedUpdate上

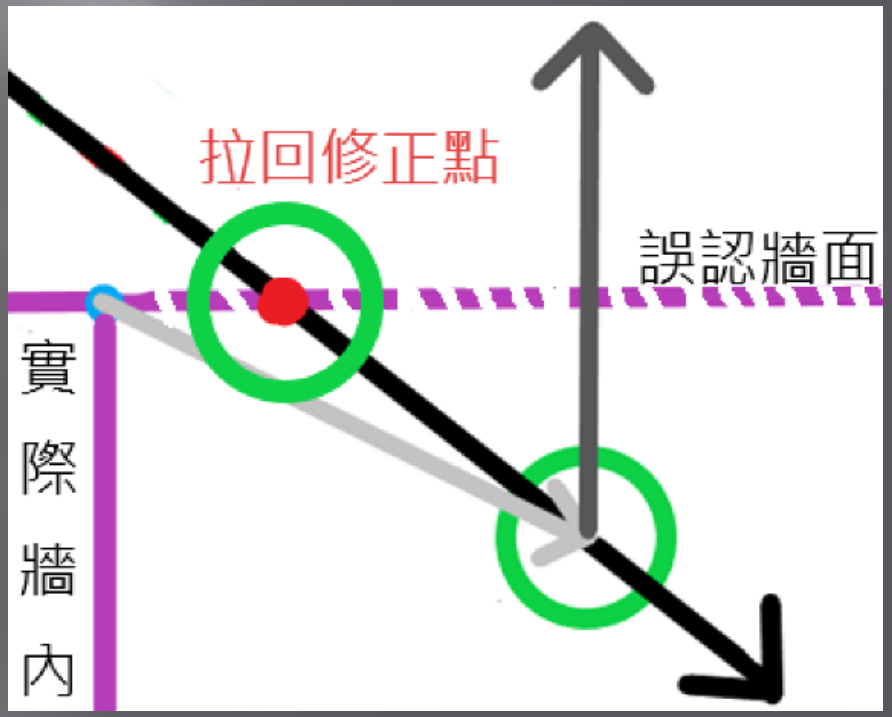




第三版。問題

- **Q :** 還有其他問題嗎?
- **Ans :**





第三版。心得

- 雖然我花了3到4周才完全完成這半成品，但是!!這是我第一次完全自創程式碼，沒有網路教學，也沒抄教學書上的程式，而且沒有人願意且能夠幫我，從思考、製作到除錯都是一手做起，所以蠻有成就感的!!但是，做出來的東西仍要加強!
- 當然要是沒有Unity的幫忙與賴佑吉、姚智原、陳國瑋先生做的Unity教學書，加上無數網路上的熱心人士教了我一年多的Unity與其程式，還有數學、物理老師教我三角函數(向量觀念我已經不確定是如何學會的了，數學還沒教，不知道感謝誰)，我也做不出這個程式，非常感謝他們!!