プログラマ作品：Unity(3D)で作ったRhythmShooterというゲーム

プロジェクトの状態：

- このゲームは一人でUnity(3D)を使って作っているゲームです。
- 現在、制作期間は４週間程度です。
- 遊べる状態になっていますがまだ完成していません。その為、バッグが出る可能性がありますのでご理解下さい。
- ゲームに使われている音楽とテクスチャーとスプライトとモデルはインターネットに乗せてあった無料で使えるリソースです。自分で作ったものはスクリプトとアニメーションとプレハブとシーンです。

注意したこと：

- ゲームを面白くする為に音楽やアニメーションをつけました。
- 任意スクリーンの解像度（16:9，16:10，4:3など）で使えるUIを作る為に苦労しました。
- リズムが厳しいのでリズムがちゃんと音楽に合わせるようなやり方を考えました。
- ステージによって、敵が変わったり音楽のリズムが変わったりするので、新しいステージを作ったりもう作られているステージを変えたりすることが簡単にできるように考えました。
- 一人で作っても、分かりやすいコードを書くことに注意しました。

是非見ていただきたい箇所：（全てCD-ROMに乗せてあります）

- ゲーム（下記参照、ゲームの使い方）
- ゲームのアピールPDF(プロモーション)
- ゲームのソースコード（下記参照、ソースコードの見せたい所）

ゲームの使い方

ゲームの説明（工夫したこと）：
このゲームはシューティングゲームとリズムゲームのミックスです。
同じ時にシューティングゲームとリズムゲームをやります。
リズムゲームのコンボによって、シューティングゲームの宇宙船の武器のパワーが変わります。
シューティングゲームに使っている武器のパワーによって、リズムゲームの難しさが変わります。
武器はショップで選びます。強い武器を選ぶと、リズムゲームが難しくなっているのでコンボを上げるのは難しくなるので強い武器が使えなくなりますからバランスが必要です。
ステージの終わりにボスが出るので、その時までコンボを上げて武器を強くした方が簡単に倒せます。
スコアが保存されているのでベストスコアが見られます。

ゲームのコントロール：
メニューはキーボードの矢印キーと[Return]キーで使えます。
シューティングゲームの宇宙船はキーボードの矢印キーで動かします。
リズムゲームは[1]、[2]、[3]キーで使えます。又は、[q,w,e]キーか[a,s,d]キーでも使えます（[1]、[2]、[3]キーがない場合）。
ポーズメニューは[p]キーでディスプレイできます。

ソースコードの見せたい所

プログラミングの基礎ができます（Structures, Static Methods, Dictionaries, Queues, Arrays など）：

```
18    private Queue<BeatInformation> beats; // Contains the beats pattern of the stage
19
20    private struct BeatInformation{
21        public BeatInformation(float time, int type, int target, int diffMin, int diffMax){
22            this.time = time;
23            this.type = type;
24            this.target = target;
25            this.diffMin = diffMin;
26            this.diffMax = diffMax;
27        }
28        public float time;
29        public int type;
30        public int target;
31        public int diffMin;
32        public int diffMax;
33    }
34
35    void Start(){
36        stageDuration = GameValuesContainer.container.stageDuration;
37        fillBeats ();
38    }
39
40    void fillBeats(){ // Defines the times, types, tragets and condition of apparition of each beat
41        beats = new Queue<BeatInformation> ();
42        float[] beatsTimes = GameValuesContainer.container.currentStage.BeatsTimes();
43        int[] beatsTypes = GameValuesContainer.container.currentStage.BeatsTypes();
44        int[] beatsTargets = GameValuesContainer.container.currentStage.BeatsTargets();
45        int[] beatsDiffMin = GameValuesContainer.container.currentStage.BeatsDifficultiesMin();
46        int[] beatsDiffMax = GameValuesContainer.container.currentStage.BeatsDifficultiesMax();
47        float time = 0;
48        int loop = -1;
49        int i = 0;
50        while (time < stageDuration) {
51            if (i == 0) {
52                loop += 1;
53            }
54            time = beatsTimes [i] + loop * GameValuesContainer.container.audioHandler.stageMusic.length;
55            BeatInformation info = new BeatInformation (time, beatsTypes [i], beatsTargets [i], beatsDiffMin [i], beatsDiffMax [i]);
56            beats.Enqueue (info);
57            i = (int)Mathf.Repeat (i + 1, beatsTimes.Length);
58        }
59
60    }
```

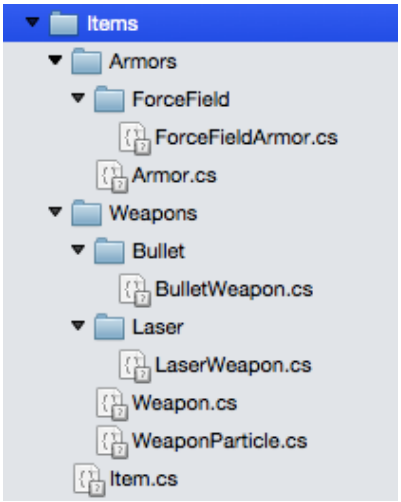
Coroutine をたくさん使っています。必ず Coroutine が止められるようにしています：

```
30    public void StrongShoot(){
31        if (coroutine == null) {
32            coroutine = CoroutineShoot (10);
33            StartCoroutine (coroutine);
34        }
35    }
36
37    IEnumerator CoroutineShoot(int power){
38        for (int j = 0; j < power; j++) {
39            InstantiateParticle (new Vector3 (0.0f, 0.0f, 0.0f), Quaternion.identity);
40            yield return new WaitForSeconds (0.15f);
41        }
42        coroutine = null;
43    }
44
45    public void StopShoot(){
46        if (coroutine != null) {
47            StopCoroutine (coroutine);
48            coroutine = null;
49        }
50    }
```

リズムがちゃんと音楽に合わせる方法を考えました：

```
107     void SpawnInRhythm(){
108         float realTime = currentAudioTime + numberLoops * GameValuesContainer.container.audioHandler.stageMusic.length;
109         while(beats.Count>0 && beats.Peek().time<realTime+reachTimesByDifficulty[difficulty-1]){
110             BeatInformation info = beats.Dequeue ();
111             if (info.diffMin <= difficulty && info.diffMax >= difficulty) {
112                 Spawn (targets [info.target], info.time - realTime, info.type);
113             }
114         }
115     }
```

同じコードを何回も書かないように派生クラスを使っています（abstract なクラスも）： Item -> Weapon -> LaserWeapon



UI が任意スクリーンの解像度で使えるようにしました、それに苦労しました：

```
88      void UpdateLayout(){
89          Vector2 sizePanel = GetComponent<RectTransform> ().sizeDelta;
90          Vector2 pivot = GetComponent<RectTransform> ().pivot - new Vector2 (0.5f, 0.5f);
91          if (graph != null && label != null) {
92              graph.gameObject.GetComponent<RectTransform> ().sizeDelta = new Vector2 (sizePanel.x, sizePanel.y * 0.8f);
93              graph.gameObject.GetComponent<RectTransform> ().localPosition = new Vector3 (-sizePanel.x * pivot.x, sizePanel.y * 0.1f - sizePanel.y * pivot.y, 0);
94              label.gameObject.GetComponent<RectTransform> ().sizeDelta = new Vector2 (sizePanel.x, sizePanel.y * 0.2f);
95              label.gameObject.GetComponent<RectTransform> ().localPosition = new Vector3 (-sizePanel.x * pivot.x, -sizePanel.y * 0.4f - sizePanel.y * pivot.y, 0);
96          }
97      }
```

Unity の物理エンジンにはなかったが必要だったメソッドを自分で書きました（物理学の計算を自分でやりました）：

```
6     static public Vector3 GetDirection(Transform transform, Vector3 vectorForward){
7         float sX = Mathf.Sin (transform.eulerAngles.x * Mathf.PI / 180);
8         float cX = Mathf.Cos (transform.eulerAngles.x * Mathf.PI / 180);
9         float sY = Mathf.Sin (transform.eulerAngles.y * Mathf.PI / 180);
10        float cY = Mathf.Cos (transform.eulerAngles.y * Mathf.PI / 180);
11        float sZ = Mathf.Sin (transform.eulerAngles.z * Mathf.PI / 180);
12        float cZ = Mathf.Cos (transform.eulerAngles.z * Mathf.PI / 180);
13        float a = vectorForward.x;
14        float b = vectorForward.y;
15        float c = vectorForward.z;
16        float vx = a * (cY * cZ + sX * sY * sZ) + b * (sX * sY * cZ - cY * sZ) + c * (cX * sY);
17        float vy = a * (cX * sZ) + b * (cX * cZ) - c * (sX);
18        float vz = a * (sX * cY * sZ - sY * cZ) + b * (sY * sZ + sX * cY * cZ) + c * (cX * cY);
19        Vector3 result = new Vector3 (vx, vy, vz);
20        result.Normalize ();
21        return result;
22    }
```