

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5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。
- ・ AI にとっても興味がある為、ニューラルネットワークで動いている AI を入れてみました。

是非見ていただきたい箇所：（全て CD-ROM/Github 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 (作品_アピール.pdf)
- ・ 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いにく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。

シューティングゲームの AI があります。普通にプレイしているだけで AI は学習しています。AI をつけると宇宙船が自動で敵を狙い撃ちます。

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。
- ・ **AI** をつけたり消したりするために**[o]**キーを使います。

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, trargets 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 }
```

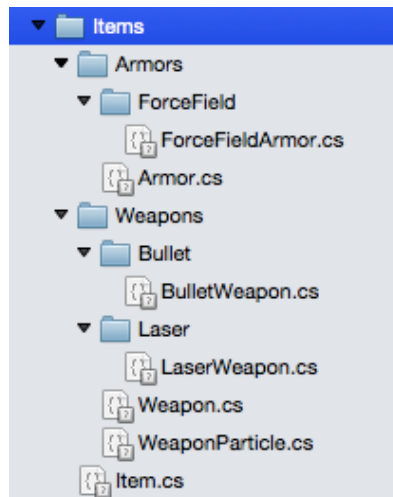
＊

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 }
```

＊

シューティングゲームのため、**ニューラルネットワーク**で動いている人工知能
を作りました。

プレイヤーの動き方から学習して、自動で敵を狙い撃つことができます。

```
24 void Update () {
25     Matrix inLayer = GetEnemiesPositions ().ToLine (); // What the AI can 'see' (enemies on the field)
26
27     AddToTrainSet(inLayer); // The current situation may become an example of the training set for machine learning
28     SaveLoad.Save ();
29
30     // We apply the neural network (1 hidden layer of size 10, logistic gate at hidden layer and out layer)
31     Matrix hiddenLayer = inLayer.Multiply (Game.current.AIMatrix1).ToLogistic ();
32     Matrix outLayer = hiddenLayer.Multiply (Game.current.AIMatrix2).ToLogistic ();
33
34     // May move automatically (if AI is activated) with the result of the Neural Network calculation
35     player.AutoMove (outLayer.Get (0,0), outLayer.Get (0,1));
36
37     // Train 1 epoch of the Neural Network with the current examples of the train set
38     Train (1);
39 }
```

学習のために **Gradient Descent** のアルゴリズムを使っています。

```
58 void Train(int nbIteration){ // Machine Learning
59     for (int i = 0; i < nbIteration; i++) {
60         for (int j=0;j<Game.current.trainSet.Count;j++){
61             TrainOneStep (Game.current.trainSet [j]);
62         }
63     }
64 }
65
66 void TrainOneStep(Pair<Matrix,Matrix> sample){ // Gradient descent
67     float change;
68     Matrix hiddenLayer = sample.First.Multiply (Game.current.AIMatrix1).ToLogistic ();
69     Matrix outLayer = hiddenLayer.Multiply (Game.current.AIMatrix2).ToLogistic ();
70     for (int i = 0; i < Game.current.AIMatrix2.xSize; i++) {
71         for (int j = 0; j < Game.current.AIMatrix2.ySize; j++) {
72             change = Game.current.AIStep * 0.5f * (outLayer.Get (0, j) - sample.Second.Get (0, j)) * (1 + outLayer.Get (0, j)) * (1 - outLayer.
73             Game.current.AIMatrix2.Set (i, j, Game.current.AIMatrix2.Get (i, j) - change);
74         }
75     }
76     for (int k = 0; k < Game.current.AIMatrix1.xSize; k++) {
77         for (int i = 0; i < Game.current.AIMatrix1.ySize; i++) {
78             change = 0;
79             for (int j = 0; j < Game.current.AIMatrix2.ySize; j++) {
80                 change += (outLayer.Get (0, j) - sample.Second.Get (0, j)) * (1 + outLayer.Get (0, j)) * (1 - outLayer.Get (0, j)) * Game.curre
81             }
82             change = Game.current.AIStep * 0.25f * change * (1 + hiddenLayer.Get (0, i)) * (1 - hiddenLayer.Get (0, i)) * sample.First.Get (0,
83             Game.current.AIMatrix1.Set (k, i, Game.current.AIMatrix1.Get (k, i) - change);
84         }
85     }
86 }
```

＊