Rational

Vol.36

目次

顧問挨拶

貴重な経験2

ソフトウェア研究部会　　顧問　　北島　宏之

一昨年から続くコロナの影響から，ようやく高専大会やコンテスト等が予定通りの日程で開催され，学生の様々な活動にも活気が戻りつつあるように思われます．そこで，過ぎ去りし苦労話しとして，「貴重な経験2」を記すことにしました．以下ご笑納頂ければ幸いです．

2019 年 10 月 13 日(日)～14 日(月)に，第 30 回全国高等専門学校プログラミングコンテストが，宮崎県の都城市総合文化ホールを会場に行なわれた．本校からは部員 4 名，競技部門への参加であった．結果は惜しくも準々決勝敗退，ベスト 8 であった．が，参加部員も顧問も“奮闘”した大会であった．

その予兆はプロコンへの出発日である 10 月 12 日(土)の数日前に台風 19 号が発生したことにあった．12 日(土)に近づくにつれ交通機関の計画運休の情報が増加，当参加メンバの搭乗予定である飛行機も欠航に．一方でプロコンの実施は正式に決定され，行くしかないと覚悟を決め急遽移動を新幹線に変更した．が，さらに 12 日(土)の各所新幹線も部分区間で運休との情報が出始め，悩み込んだのが 10 日(木)の深夜，11 日(金)に入ってからであった．そこから，12 日(土)中の移動手段を調べに調べたものの不可能ということが判明，最終的に 11 日(金)中の移動を決心しつつ参加メンバへの連絡や対応可能性など不安なまま仮眠を取った．

11 日(金)早朝に参加メンバに携帯電話やショートメール等で連絡を取り仙台駅に集合可能か確認したところ，幸いにも全員が迅速に対応でき，仙台駅みどりの窓口の混雑に卒倒しそうになりながらも東北新幹線の指定席を確保，東海道新幹線や山陽新幹線は自由席になること諦めつつ博多までの移動となった．残るは，想定外の 11 日(金)の博多 1 泊であったが，東北新幹線の中でスマホを駆使して様々なホテル予約サイトを検索し続け，幸いにも博多駅に近いホテルを予約確保できた．その間，学校や参加メンバの担任の先生方にもメール連絡を入れ続けた．結局，7 時間余りの移動の末に博多に 1 泊，翌 12 日(土)は九州の穏やかな気候の下に都城まで九州新幹線と JR 日豊本線を乗り継ぎ 3 時間半ほどの移動しようやく都城に到着できた．

思い返せば 2006 年，第 17 回茨木大会も相当な経験であったが(Rational-Vol.20(?)を参照)，今回も貴重な経験となった．2006 年当時と大きく異なるのは，スマホとインターネットの発達やコモディティ化であって，それらの恩恵がなければ今回のプロコン参加は不可能であったと言える．便利になったものです，ありがたやありがたや．

追伸．復路は飛行機にて，鹿児島～神戸経由～仙台と苦もなく帰ってこれました．

代表挨拶

会長挨拶

ロボティクスコース　3年　佐藤　至

まずは、「Rational」を手にとってくださってありがとうございます。昨年、一昨年と新型コロナウイルスの影響により高専祭が中止になってしまったり、外部の方が高専祭に来場できなかったりと、あまり活発に活動できない一年でした。しかし今回は外部の人も来場可能ということで、今までと違う高専祭に戸惑いながらも、とても嬉しく思っています。

今回のプログラミングコンテストですが、今回は重ね合わさった音声をコンピューターで分析して行う「かるた」でした。しかし残念ながら予選で落選してしまったためプログラミングコンテストに関する記事はありません。楽しみにされていた方にはこの場を借りてお詫び申し上げます。しかし、部員それぞれの個性が輝く部誌にはなっていると思うので、最後まで読んでいただけると幸いです。

最後にこの「Rational」の作成に携わってくださった方、並びに顧問の北島先生、佐藤先生、そして今この部誌を読んでくださっている方に感謝を込めて、会長挨拶とさせていただきます。

副会長挨拶

ロボティクスコース　2 年　鈴木　佑

「Rational」を手に取っていただき、ありがとうございます。今年は昨年とは異なり、新型コロナウイルスの流行が落ち着いて外部から高専祭に来て頂けるようになりました。

これは、高専祭の「Rational」の発表を活動の一部としているソフトウェア研究部会にとっては感慨深いです。今年度の「Rational」はどの部員も心を込めて製作しましたので、ぜひ最後までお楽しみください。

最後に顧問の北島先生と佐藤先生、そして今手に取っていただいている方々、全ての皆様にお礼を申し上げます。

C#で Todo リストを製作する

ロボティクスコース 2 年　鈴木 佑

1. はじめに

私は今回の C#という言語を使用して予定を書き留める Todo リストを製作しました。開発環境として Microsoft Visual Studio(バージョン.1.7.3.4)です。この記事を最後まで読んでいただけたら幸いです。

2. 実際の動作とプログラム

2.1. 動作解説

プログラムを実行した画面が図 1 です。何か追加したい予定があった場合 3 つのコンボボックスの左から順番に年と月と日を決めて、下のテキストボックス内に予定を書き込みます。その後、[予定を追加する]を押すと、1 番上のテキストボックス内に予定が追加されます。ですがこのままタブを閉じると予定が保存されずに消去されてしまうので、[予定を保存する]を押すと指定したファイル内に予定が保存されます。逆に予定を取り消したい場合は、消去したい予定の横のチェックボックスにチェックを入れて[予定を取り消す]を押します。そうしたら予定が取り消されます(図 2)。

図 1 実際の画面

図 2 この状態でボタンを押すと消える

2.2. 予定を消去するプログラムの解説

リスト 1 は予定を消去するときのプログラムです。このプログラムでは[予定を消去する]ボタンを押したときにもしチェックボックスがチェックされていたら、一番上の予定を消去する動作を行っています。

これはイベントハンドラと if 文を組み合わせて使用しています。イベントハンドラとはユーザーがウィンドウを操作したときに何か処理するプログラムです。if 文は条件を満たした場合に処理するプログラムです。

リスト 1 プログラム

```
1   public void BtO_Click(Object
    sender, EventArgs e){
2   if (cb.Checked == true){
3               tbO.Text = "";
4                           }
```

3. プログラムの大まかな流れ

予定を追加する際の大まかな処理を図 3 に示します。これはプログラムを実行した際に年、月、日、内容を入力します。その後、記入されていないテキストボックスがあるか判断します。もしある場合は、記入されていないテキストボックスに入力した文字が出力されます。ない場合には出力出来ないので、一度予定を消去してもう一度入力する流れになります。

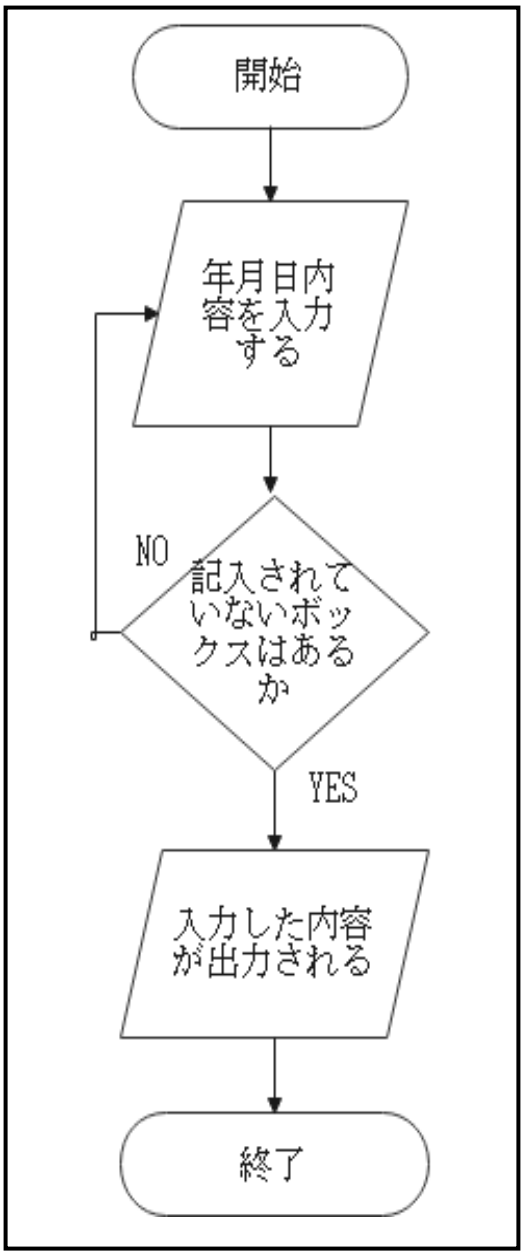

図 3 予定を追加する際のフローチャート

4. 終わりに

私は今回 C#で Todo リストを製作しましたが、一番苦戦したのは、データの保存です。このレーショナルには詳しく説明出来ませんでしたが、どういった方法で出力した予定を保存するかがしばらくわからず、苦慮しました。最終的に自分が愛用している参考書に詳しく記載されていたのを参考にして完成させました。来年は高専プロコンに参加する予定です。高専プロコンに備えプログラミング言語を更に学習していきたいと思います。最後まで読んでいただきありがとうございました。

参考文献

高橋麻奈．やさしい C#．SB クリエイティブ 2019

2D アクションゲームの作り方

ロボティクスコース　3年　佐藤　至

1.　はじめに

今回私は Unity を用いて 2D アクションゲームを制作した。この記事では、その大まかな作り方について解説する。

2.　プレイヤーの作り方

2.1.　プレイヤーを移動させる

まずはプレイヤーの移動を管理するプログラムを解説する。リスト 1 を見てほしい。これはプレイヤーの横移動を管理するプログラムである。

このプログラムの内容は、4 行目で Horizontal の値を取得し、6～18 行目で正の数なら右に移動、負の数ならば左に移動するといったものになっている。

Horizontal とは Unity の機能で大まかに説明すると、「d」キーを押すと正の数になり、「a」キーを押すと負の数になるように設定されている。また 3 行目のプログラムでスピードを管理できるような仕組みにしている。

リスト 1　横移動のプログラム

```
1.  private float GetXSpeed(){
2.    //キー入力されたら行動する
3.      public float speed;
4.      float horizontalKey
          =  Input.GetAxis("Horizontal");
5.      float xSpeed = 0.0f;
6.       if (horizontalKey > 0)
7.       {
8.            transform.localScale  =  new
    Vector3(-1, 1, 1);
9.            xSpeed = speed;
10.   }
11.
12.      else if (horizontalKey < 0)
13.      {
14.         transform.localScale  =
              new Vector3(1, 1, 1);
15.         xSpeed  =  -speed;
16.      }
17.
18.      else
19.      {
20.           xSpeed = 0.0f;
21.      }
22.
23.      return xSpeed;
24.  }
```

次に縦移動のプログラムについて解説する。リスト 2 を見てほしい。

このプログラムはまず 6 行目に重力として、常に下に移動する力が働いている。それに加えて、Vertical の値を取得して、正の数ならば上限高度まで、上方向に移動する仕組みになっている。

Vertical とは Horizontal と同じような機能で、これは「w」キーを押すと正の数になるように設定されている。ただし、地面に足がついていない状態では、ジャンプができないように設定している。また、このプログラムも、上昇スピード、上限高度重力を管理できるようになっている。

リスト 2　縦移動のプログラム

```
1.  public float gravity;
2.  public float jumpSpeed;
3.  public float jumpHeight;
4.  private float GetYSpeed(){
5.  float verticalKey
    =Input.GetAxis("Vertical");
6.  float ySpeed = -gravity;
7.
8.  private float GetYSpeed(){
9.      float verticalKey =
    Input.GetAxis("Vertical");
10.     float ySpeed = -gravity;
11. }
12.
13. if(isGround){
14.     if (verticalKey > 0){
15.         ySpeed = jumpSpeed;
16.         jumpPos = transform.position.y;
17.         isJump = true;
18.         }
19.
20.     else{
21.         isJump = false;
22.         }
23. }
24.
25. else if (isJump){
26.         //ジャンプキーを押してるか
27.     bool pushUPKey = verticalKey > 0;
28.         //高さ制限
29. bool canHeight = jumpPos + jumpHeight
    > transform.position.y;
30.
31. if (verticalKey > 0 && jumpPos +
    jumpHeight > transform.position.y){
32.     if (isHead){
33.         isJump = false;
34.     }
35.     ySpeed = jumpSpeed;
36.   }
37.
38.     else{
39.         isJump = false;
40.     }
41.     }
42.
43. if (isJump){
44. ySpeed *=
    jumpCurve.Evaluate(jumpTime);
45.  }
46.
47.     return ySpeed;
48.
49. }
```

2.2　プレイヤーの当たり判定

次にプレイヤーの当たり判定の作り方について解説する。当たり判定とは、敵や地面にぶつかったことを検出する処理を指す。今回当たり判定を作るにあたって、Unity にはコライダーとタグ機能があるのでそれを活用する。

コライダー機能を設定すると、Unity 側で物に自動で当たり判定を追加できる機能である。

タグ機能は、物にタグという形でカテゴライズすることができる機能である。

私はこれらの機能を利用して、敵キャラには敵キャラ用のタグを設定し、敵キャラ用のタグを持ったコライダーに触れると、プレイヤーがやられるように設定している。

同様に、地面には地面用のタグを設定することで、地面用のタグを持ったコライダーに触れているときのみジャンプすることが可能になる。

3. 敵の作り方

次に敵の作り方について解説する。敵キャラの当たり判定も、タグで管理しているので、当たり判定の解説は省略する。そのため、ここでは敵の移動プログラムについて解説する。リスト 3 を見てほしい。

7 行目から 20 行目までで、画面に写っているとき、又はこちらで設定できるスイッチがオンになっている間に移動するプログラムが書かれている。

また 27 行目から 30 行目で壁用のタグが付いたものに衝突したとき、進む向きを逆方向にするプログラムが書かれている。敵キャラのプログラムは、以上の 2 つが組み合わさったプログラムになっている。

リスト 3　敵の移動プログラム

```
1.  public float speed;
2.  public float gravity;
3.  public float returnTime;
4.  public bool nonVisibleAct;
5.  public collisionCheck checkCollision;
6.
7.  if(sr.isVisible||nonVisibleAct){
8.        spentTime += Time.deltaTime;
9.
10.       if (checkCollision.isOn){
11.             direction = !direction;
12.             spentTime = 0.0f;
13.       }
14.
15.       int xVector = -1;
16.       if (direction){
17.       xVector = 1;
18.       transform.localScale =
    new Vector3(-xScale,yScale,1);
19.
20. }
21.
22. else{
23.       xVector = -1;
24.       transform.localScale = new
    Vector3(xScale,yScale,1);
25. }
26.
27.  if (spentTime >= returnTime){
28.           direction = !direction;
29.           spentTime = 0.0f;
30.   }
31.
32. rb.velocity=
    newVector2(xVector*speed,-gravity);
33.             }
34.
35. else{
36.       rb.Sleep();
37. }
```

4. ステージの作り方

次にステージの作り方について解説する。Unity にはタイルマップという機能があるので、今回はそれを活用する。ステージを作ったら、コライダーを設定する。最後に、地面用のタグを設定して、ステージの完成である。

5. ゴール、ゲームオーバーの作り方

次にゴール、ゲームオーバーの作り方について説明する。まずゲームオーバーの画面、クリアの画面を作る。次にゲームオーバーやクリアの処理を行うプログラムを解説する。リスト 4 を見てほしい。このプログラムには 3 つの要素がある。

1 つ目は、38 行目から 41 行目の、isGameOver という敵用のタグが付いた物にぶつかった際に立つフラグがオンになっているとき、ゲームオーバーの画面を表示させるプログラムである。

2 つ目は、42 行目から 47 行目の、isGoal というゴール用のタグが付いた物にぶつかったとき、クリア画面を表示するプログラムである。

3 つ目は、14 行目から 32 行目までのゲームスタート時にプレイヤーをスタート地点に設置するプログラムである。

リスト 4　ゲームオーバー、クリアの処理

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using UnityEngine.SceneManagement;
5.
6.  public class StageController :
    MonoBehaviour
7.  public GameObject playerObj;
8.  public GameObject GameEndObj;
9.  public GameObject GameClearObj;
10.
11.       private Player p;
12.
13.       // Start is called before the first
    frame update
14. void Start()
15. {
16.     if(playerObj != null){
17.         GameEndObj.SetActive(false);
18.         GameClearObj.SetActive(false);
19.         playerObj.transform.position =
    transform.position;
20.
21.         Debug.Log("移動完了");
22.   p = playerObj.GetComponent<Player>();
23.
24.   if (p == null){
25.       Debug.Log("プレイヤー入れて");
26.                   }
27.             }
28.
29. else{
30.       Debug.Log("物入れて");
31.         }
32.       }
33. // Update is called once per frame
34.
35.
36. void Update(){
37. //ゲームオーバー処理
38. if(GManager.instance.isGameOver){
39.       GameEndObj.SetActive(true);
40.       Debug.Log("ゲームオーバー処理通
    過");
41. }
42. //クリア処理
43. if(GManager.instance.isGoal){
44.       GameClearObj.SetActive(true);
45.       Debug.Log("クリア処理通過");
46.             }
47.       }
48. }
```

6.　おわりに

今回私はこのような形でゲームを制作した。読んでいると小難しく思うだろうが、Unity はあなたが思うより、簡単で直感的な操作が可能なので、ゲーム開発をしたことがない人にもおすすめできるゲームエンジンである。そのため、ゲーム開発に興味を持った方は是非 Unity を利用してゲーム開発に取り組んでほしい。

参考文献

Unity 2D アクションゲームの作り方
https://dkrevel.com/makegame-beginner/before-2d-action/

Unity
https://unity.com/ja

Unreal Engine 技法まとめ

ロボティクスコース　4年　鈴木　晴斗

1.　はじめに

今回は Epic Games 社が開発している Unreal Engine(以下 UE)で使える技法を 2 つ紹介します。使用した UE のバージョンは 4.27.2 です。

2.　BP からマテリアルを変更する

ゲームなどにおいて、キャラクターや配置されているオブジェクトのマテリアルが変更される場面は多いと思います。例えば、あらかじめマテリアルを複数用意してプログラムから変更すれば、1 つのモデルで様々な見た目のアイテムを作ることができます。プレイヤーがダメージを受けた時にマテリアルを変更すれば、視覚的効果を生み出すことができます。

この項では、UE に内包されている Blueprint システム(以下 BP)を使い、プレイヤーキャラクターと持っている銃、そこから発射される弾のマテリアルを変更する方法を紹介します。

まずは、マテリアルを変更するための「きっかけ」を作る必要があります。今回は、「色のついた床に乗る」というのをアクションにします。

初めに BP クラスを作成し、そこにコンポーネントを追加していきます。追加するのは床の形状である Cylinder と、乗ったことを検知するための Box コリジョンです(図 1)。

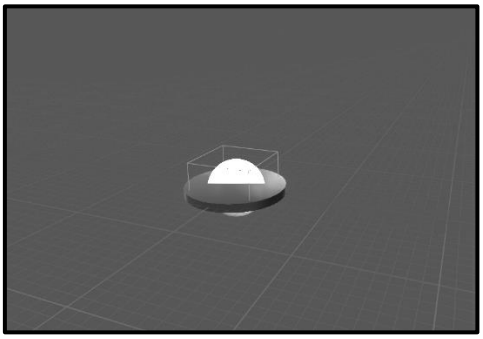

図 1　作成した床

次に処理を作っていきます。作成した BP が図 2 です。プレイヤーが床に乗ったとき、On Component Begin Overlap イベントが呼び出されます。次に Cast To FirstPersonCharacter でキャラクターと、持っている銃を参照できるようにします。最後に Set Material で指定したマテリアルに変更することで、床に乗った時キャラクターと銃のマテリアルを変更することができるようになります。

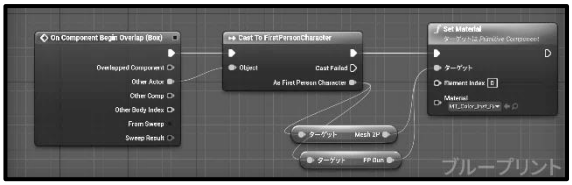

図 2　マテリアルを変更する処理

発射される弾のマテリアルを変更するには、まずプレイヤーキャラクターの BP クラスで、現在のマテリアルがどの色なのかを保持する変数を定義します。床に乗った時に、この変数に変更後のマテリアルを判別するための値を代入します。弾を発射する処理はこの BP クラス内にあるので、発射される直前にその変数を参照してマテリアルをセットします。図 3 が実際の BP です。

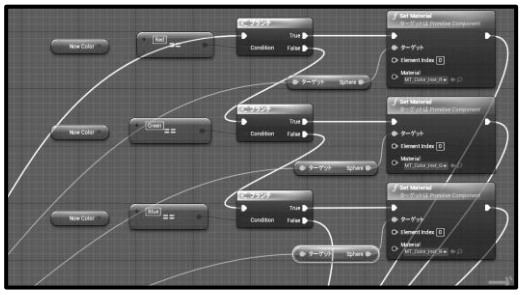

図 3　弾のマテリアルを変更する処理

最後に BP クラスをレベルに配置し、その上に乗ってみると、キャラクターと銃、弾のマテリアルが切り替わります。

3. テクスチャの繰り返しを防ぐ

UE にはランドスケープモードというものがあります。これを使うと、オープンワールドゲームのような広大な自然風景などを、簡単に作成することができます。このモードでは主に以下の 3 つのことを行えます。

- 管理(ランドスケープの追加、削除、サイズ変更)
- スカルプト(地形を作る)
- ペイント(地形にマテリアルを塗る)

この項では、3 つ目のペイントについての技法を紹介します。

ペイントでマテリアルを塗ると図 4 のようにテクスチャの繰り返しが起こります。

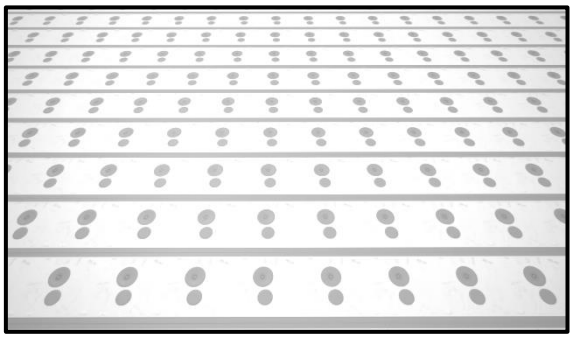

図 4 テクスチャの繰り返し

テクスチャのサイズを大きくすることで、カメラが近くに寄った時は目立たなくすることはできますが、引いた時にはやはり繰り返しが見えてしまい、自然な複雑さが失われてしまいます。

これはマテリアルノードに Texture_Bombing ノードを追加し、テクスチャをランダムに回転・反転することで繰り返しを防ぎ、自然な複雑さを生み出すことができます(図 5、図 6)。

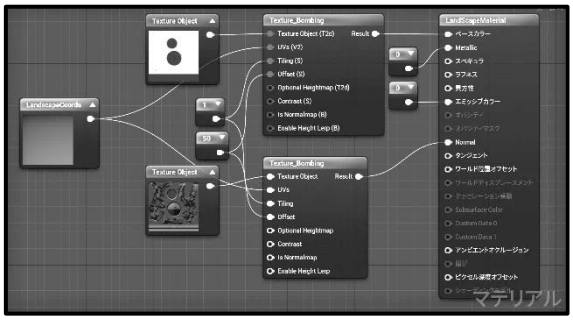

図 5 繰り返しを防ぐノードの組み方

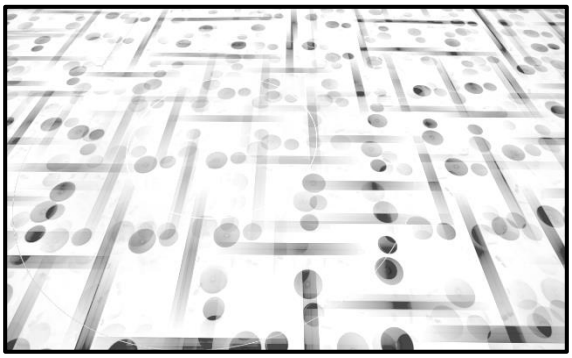

図 6 Texture_Bombing を使った場合

4. おわりに

今回は自分が UE を使っていて参考になったこと、役立ったことを備忘録的にまとめてみました。マテリアルの変更は、現在開発しているゲームで学んだことです。変数を定義して、今どのマテリアルがセットされているのかを保存できるため、例えばマテリアルによって処理を変えるなど、応用が利きそうです。テクスチャの繰り返しを防ぐ方法は、スカルプトや高品質なアセットを組み合わせることで、ランドスケープをより自然な風景に近づけることができます。

まだ UE を使っていてわからないことが多いと感じます。今後もゲーム開発などを通して、UE の技法を学んでいきたいです。ゲーム開発において UE のほかに Unity がよく使われますが、それに比べると UE は日本語ドキュメントや参考書などが少ないように感じます。この記事が UE を使ってゲーム開発をする人の役に、少しでも立てれば幸いです。

参考文献

Unreal Engine 4.27 ドキュメント
https://docs.unrealengine.com/4.27/ja/

株式会社ヒストリア
https://historia.co.jp/

Qiita
https://qiita.com/

弾幕の胎動

ロボティクスコース　4年　冨山　結都

1.　はじめに

今回私は Unity を用いて弾幕 STG を作成した。この記事では、その基礎となる作り方を解説していく。

2.　画面とプレイヤー

2.1.　初期画面設定

シーンは 3D で作るが、画面は 2D のようにしたい。そのためにシーンギズモの Y をクリックすることで画面を垂直に見る。

2.2.　プレイヤーの作成と移動と範囲制限

プレイヤーの Prefab をつくり、移動するスクリプトを書く。スクリプトの中身はリスト 1 を見てほしい。ここにはプレイヤーの移動、移動速度、移動範囲の制限が書いてある。1 行目で移動速度を指定し、4～9 行目で移動を管理している。移動するにはキーボードの矢印キーで操作する。さらに 11～17 行目でプレイヤーの移動範囲の制限を設けている。

リスト 1　移動と範囲制限のスクリプト

```
1.  public float moveSpeed = 0.2f;
2.  private Vector3 pos;
3.  void Update()
4.  {;
5.    float moveH
       =Input.GetAxis("Horizontal")*
       moveSpeed;
6.    float moveV
      =Input.GetAxis("Vertical")*
      moveSpeed;
7.    transform.Translate  (moveH,0.0f,moveV);
8.    Clamp();
9.  }
10.
11. void Clamp()
12. {
13.   pos = transform.position;
14.   pos.x = Mathf.Clamp(pos.x, -20, 20);
15.   pos.z = Mathf.Clamp(pos.z, -15, 15);
16.   transform.position = pos;
17. }
```

3.　ミサイルの作成と発射

プレイヤーが発射するミサイルを作る。3D オブジェクトからミサイルを作成し、Rigidbody を付ける。そして Rigidbody の”重力を使用”を切る。これらの操作をしたら、このミサイルをプレハブ化する。次に、このミサイルのスクリプト（リスト 2）を書く。このスクリプトの 1～3 行目は変数の定義をし、11～13 行目にミサイルオブジェクトを作成し発射するように書いた。そして 13 行目にミサイルを、発射された 2 秒後に削除するようにした。ミサイルの発射はコントロールキーで発射できる。弾幕 STG といえばミサイルは長押しで連射したいので、6、7 行目で長押し中に連射するようにし、9 行目で発射間隔を設定した。

4.　敵と HP と弾幕の作成

4.1.　敵とそのミサイルとHP

敵の体も 3D オブジェクトで作る。そして敵の HP をつける為のスクリプト（リスト 3）を書く。これは Missile タグの物が当たったら HP を 1 減らし、HP が 0 になったら敵を削除する。そのためプレイヤーのミサイルプレハブに Missile のタグをつける必要がある。

リスト 2 ミサイルのスクリプト

```
1.  public GameObject missilePrefab;
2.  public float missileSpeed;
3.  private int timeCount;
4.  void Update()
5.  {
6.   timeCount += 1;
7.   if (Input.GetButton("Fire1"))
8.   {
9.    if (timeCount % 20 == 0)
10.   {
11.    GameObject missile =
       Instantiate(missilePrefab,
       transform.position,
       Quaternion.identity);
12.    Rigidbody missileRb =
       missile.GetComponent<Rigidbody>();
13.    missileRb.AddForce
       (transform.forward * missileSpeed);
       Destroy(missile, 2.0f);
14.   }
15.  }
16. }
```

リスト 3 敵の HP スクリプト

```
1.  public int enemyHP;
2.  private void
3.  OnTriggerEnter(Collider other)
4.  {
5.   If
6.   (other.gameObject.Compare
      Tag("Missile"))
7.   {
8.    enemyHP -= 1;
9.    Destroy(other.gameObject);
10.   if (enemyHP == 0)
11.   {
12.    Destroy(transform.root.gameObject);
13.   }
14.  }
15. }
```

そして、今度は敵のミサイルを作る。リスト 2 を参考にしてほしい。7、8、15 行目を消し、新たに敵のミサイルプレハブを別に作り、このスクリプトのミサイルプレハブに置き換えればよい。このミサイルプレハブのタグには EnemyMissile をつけよう。これで、敵はミサイルを放つようになる。

最後にプレイヤーの HP を作る。これはリスト 3 を参考にしてほしい。スクリプト内の enemy の文字を消し 6 行目の Tag 内を EnemyMissile にすればできる。これで、プレイヤーは敵のミサイルを避けねばならない。

4.2. 最初の弾幕

敵のミサイルを作り、攻撃されるようになったが、まだ弾幕と言える程では無い。ここでは簡単な弾幕の作り方を書く。まず画面内に多くのミサイルが必要なので、敵のミサイルの発射速度を上げる。そして、transform.Rotate(0, 1, 0);を、先ほど作った敵のミサイルのスクリプトの void Update()内に入れると、螺旋状に飛んでくる。1 の値を大きくしていくと回転速度が上り、いかにも弾幕らしい光景が見ることが出来る。

5. おわりに

今回ここに書いたことはほんの最初に過ぎないが、ここまでできれば理想の弾幕ができるだろう。Unity の最も難しいことは、Unity をインストールすることだと思う。それさえできれば、ゲームを作る事など簡単なので頑張って作ってみてほしい。

参考文献

CodeGenius

https://codegenius.org/

Unity

https://unity.com

編集後記

猛暑が過ぎ去り、次第に寒くなってきました。

さて今回の高専祭のテーマは「煌閃」ということで、今回の「Rational」の記事もテーマの名前に恥じない、部員一人一人の個性が輝くような記事になっております。また部員が実際に制作したゲームなども展示しているので、興味がある人はプレイしていただけると幸いです。

最後になりますが、記事を執筆してくださった部員の皆様、記事の構成をしてくださった先輩方、そして何より最後まで読んでくれた皆様方にこの場を借りてお礼申し上げます。

編集長　佐藤 至

Rational Volume36

発行日	10 月 25 日
発行団体	仙台高専(名取)　ソフトウェア研究部会
印刷所	仙台高専(名取)

STAFF

代表	佐藤至
校正	鈴木　雄裕　　鈴木晴斗
	冨山　結都　　佐藤　至
執筆	ソフトウェア研究部員
製本	ソフトウェア研究部員