Unity ゲーム開発演習 3D迷路シューティング編 レジェメ

※この講座で作成するゲームはゲームカレッジの参考書として用いた書籍「Unity5 3D ゲーム開発講座 ユニティちゃんで作る本格アクションゲーム」掲載のサンプルを元にしてアレンジしたものです。再配布禁止。

パート 1 迷路ゲーム

1. 準備

① Unity を起動し「新しいプロジェクト」をクリック
② 「3D コア」をクリックし、プロジェクト名「maze3d〇〇(名前)」保存場所
「c:\Users\human\Documents」と入力し、プロジェクトを作成をクリック
③ 基本画面が起動
④ メニューの「Window」「Layouts ▸」「Default」を実行し、画面レイアウトを初期化
⑤ メニューの「Assets」「Import Package ▸」「Custom Package...」を実行
⑥ Import Package ウィンドウが開くので「shooting3dmaze202302.unitypackage」を選択して
開くをクリック
⑦ Import Unity Package ウィンドウが開くので Import をクリック（エラーは無視して良い）
⑧ The open scene(s) have been modified externally」ウィンドウが開くので Ignore をクリック

2. 実行までの操作方法と迷路ゲームのスケルトンの実行確認

① Project ビューの「Assets」にある「01_Goal」シーンをダブルクリック
② Hierarchy ビューに「01_Goal」が表示されるので、上のツールバーにある▸をクリック
数秒後に Game ビューに迷路とキャラが表示されることを確認する
※画面をクリックしてしまうとカーソルが見えなくなるが、Esc キーを押すと戻る
③ 実行画面の右上にある Play Focused▼をクリックし Play Maximized をクリック（これで次回からは全画面表示になる）
④ 上のツールバーにある▸をクリックすると実行が終了し基本画面に戻る

3. プログラミング：キー操作による回転

① Project ビューの「Assets」「Scripts」をクリック
② 表示された「Player」をダブルクリック
③ Visual Studio が起動するので「// 移動処理」の中の「//回転の実行」の下に下記の通り追記する

```
//回転の実行
transform.rotation = geteuler();
```

※「l」は小文字のエルです。

④ また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private Quaternion geteuler() { //回転の実行
     return Quaternion.Euler(0, m_rotationY, 0); //第 1、3 引数はゼロ
}
```

⑤ 「ファイル(F)」「すべて保存(L)」をクリック
⑥ Unity に戻り、▶をクリックすると数秒後に実行開始
⑦ ←キー、→キーで向きが左右に変わることを確認
※右上の全体マップでアイコンが回転してどちらを向いているか示していることも確認
※画面をクリックしてしまうとカーソルが見えなくなるが、Esc キーを押すと戻る
⑧ ▶を再度クリックすると終了

4. プログラミング：マウスの移動量による回転

① Visual Studio に戻り「Player.cs」の「// 移動処理」の中の「//マウスの移動量による回転」の下に下記の通り追記する

```
//マウスの移動量による回転
ay += getmouse();
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private float getmouse() { //マウスの移動量による回転
    return Input.GetAxis("Mouse X") * ROTATION_Y_MOUSE; //X の前に半角空白
}
```

② 「ファイル(F)」「すべて保存(L)」をクリック、Unity に戻り、▶をクリック
③ マウスを左右に動かしても向きが左右に変わることを確認
④ ▶を再度クリックすると終了

5. プログラミング：キー操作による前後移動

① Visual Studio に戻り「Player.cs」の「// 移動処理」の中の「//前進後退の実行」に下記の通り追記する

```
//前進後退の実行
transform.position += getap(ap);
```

② また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private Vector3 getap(Vector3 ap) { //前進後退の実行
    return transform.rotation * ap * Time.deltaTime;
}
```

③ 「ファイル(F)」「すべて保存(L)」をクリック、Unity に戻り、▶をクリック
④ ↑キーで前進、↓キーで後退することを確認
⑤ ▶を再度クリックすると終了

6. プログラミング：前後移動のモーション

① Visual Studio に戻り「Player.cs」の「// メカニム(モーション) 」の「//前後移動のモーション」の下に下記の通り追記する

```
//前後移動のモーション
domotion(an, ap);
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private void domotion(Animator an, Vector3 ap) { //前後移動のモーション
    an.SetFloat("SpeedZ", ap.z); //SpeedZ は間に空白を入れないこと
}
```

② 「ファイル(F)」「すべて保存(L)」をクリック、Unity に戻り、▶をクリック
③ ↑キーで前進すると走るモーション、↓キーで後退すると歩くモーションになることを確認
※ゴールまで行けることを確認してください（ゴールに着いても突き抜けてしまいます）
④ ▶を再度クリックすると終了

7. プログラミング：アイテムの回転

① Unity に戻り、下の Project ビューの「Assets」「Scripts」の「Round」をダブルクリック
② Visual Studio が起動するので「// 毎フレーム呼び出されるメソッド」の「//アイテムを回転」の下に下記の通り追記する

```
//アイテムを回転
transform.rotation = geteuler();
```

※「l」は小文字のエルです。「()」は「(」と「)」を並べたもので「付加情報なし」を意味します

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private Quaternion geteuler() { //アイテムを回転
    return Quaternion.Euler(0, rotationNow, 0); //第 1、3 引数はゼロ
}
```

③ 「ファイル(F)」「すべて保存(L)」をクリック、Unity に戻り、▶をクリック
④ ゴールが回転していることを確認（ゴール以外は変わりません）
⑤ ▶を再度クリックすると終了

8. プログラミング：ゴールの判定

① Unity に戻り、下の Project ビューの「Assets」「Scripts」の「Goal」をダブルクリック
② Visual Studio が起動するので「// 何かに接触したら呼び出されるメソッド」の中の「//ステージクリア」の下に下記の通り追記する

```
//ステージクリア
stageclear();
```

※「l」は小文字のエルです。

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private void stageclear() { //ステージクリア
    Game.SetStageClear();
}
```

③ 「ファイル(F)」「すべて保存(L)」をクリック、Unity に戻り、▶をクリック
④ ゴールに到着すると「STAGE CLEAR」と表示され、移動や回転ができなくなることを確認
⑤ ▶を再度クリックすると終了

パート 2 迷路シューティングゲーム

9. 迷路シューティングゲームのスケルトンの実行確認

① Unity に戻り、下の Project ビューの「Assets」にある「**02_Target**」シーンをダブルクリック
② 上の「Hierarchy」に「02_Target」が表示されるので、▶をクリック
③ 今度はゴールの代わりに複数のターゲットが回転表示されるが、キャラをターゲットまで移動させて接触しても、反応しないことを確認
④ ▶を再度クリックすると終了

10. プログラミング：弾の発射

① Visual Studio に戻り、タブで「Player.cs」を再度開く
② 「// マウスクリックによる射撃」の中の「//弾を撃つ」の下に下記の通り追記する

```
//弾を撃つ
shoot(vbp);
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private void shoot(Vector3 vbp) { //弾を撃つ
    Instantiate(bulletObject, vbp, transform.rotation);
}
```

③ 「ファイル(F)」「すべて保存(L)」をクリック、Unity に戻り、▶をクリック
④ マウスを左クリックすると弾円が表示され発射音が鳴ることを確認（まだ弾は飛ばない）
⑤ Esc キーを押してから、▶を再度クリックすると終了

11. プログラミング：弾の発射モーション

① Visual Studio に戻り、「Player.cs」の「//メカニム(モーション) 」の中の「//発射モーション」の下に下記を追記する

```
//発射モーション
shootmotion(an, sf);
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private void shootmotion(Animator an, bool sf) { //発射モーション
    an.SetBool("Shoot", sf);
}
```

② 「ファイル(F)」「すべて保存(L)」をクリック、Unity に戻り、▶をクリック
③ マウスを左クリックすると発射モーションをすることを確認（まだ弾は飛ばない）
④ Esc キーを押してから、▶を再度クリックすると終了

12. プログラミング：弾の移動

① Unity に戻り、下の Project ビューの「Assets」「Scripts」の「Bullet」をダブルクリック
② Visual Studio が起動するので「// 毎フレーム呼び出されるメソッド」の中の「//弾の移動」の下に下記の通り追記する

```
//弾の移動
transform.position += getap(vap);
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private Vector3 getap(Vector3 vap) { //弾の移動
    return transform.rotation * vap * Time.deltaTime;
}
```

③ 「ファイル(F)」「すべて保存(L)」をクリック、Unity に戻り、▶をクリック
④ マウスを左クリックすると弾が飛ぶことを確認（まだ衝突判定はないので果てまで飛ぶ）
⑤ Esc キーを押してから、▶を再度クリックすると終了

13. プログラミング：衝突したら弾を消す

① Visual Studio に戻り、「Bullet.cs」の「// 何かに接触したら呼び出されるメソッド」の中の「//弾を消す」の下に下記を追記する

```
//弾を消す
destroy();
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private void destroy() { //弾を消す
    Destroy(gameObject);
}
```

② 「ファイル」「すべて保存」をクリック、Unity に戻り、▶をクリック
③ マウスを左クリックすると弾が飛び、壁やターゲットにぶつかると**消える**ことを確認
④ Esc キーを押してから、▶を再度クリックすると終了

14. プログラミング：弾の衝突エフェクト

① Visual Studio に戻り、「Bullet.cs」の「// 何かに接触したら呼び出されるメソッド」の中の「//弾の衝突エフェクト」の下に下記を追記する

```
//弾の衝突エフェクト
effect();
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private void effect() { //弾の衝突エフェクト
    Instantiate(hitEffectPrefab, transform.position, transform.rotation);
}
```

② 「ファイル」「すべて保存」をクリック、Unity に戻り、▶をクリックすると数秒後に実行開始
③ マウスを左クリックすると弾が飛び、壁やターゲットにぶつかると**四散する**ことを確認
④ Esc キーを押してから、▶を再度クリックすると終了

15. プログラミング：ターゲットの衝突判定

① Unity に戻り、下の Project ビューの「Assets」「Scripts」の「Target」をダブルクリック
② Visual Studio が起動するので「// 何かに接触したら呼び出されるメソッド」の中の「//ターゲットを削除する」に下記の通り追記する

```
//ターゲットを削除する
destroy();
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private void destroy() { //ターゲットを削除する
    Destroy(gameObject);
}
```

③ 「ファイル(F)」「すべて保存(L)」をクリック、Unity に戻り、▶をクリック
④ 弾をターゲットに当てると消えることを確認
⑤ Esc キーを押してから、▶を再度クリックすると終了

16. プログラミング：ターゲットの破壊エフェクト

① Visual Studio に戻り、「Target.cs」の「// 何かに接触したら呼び出されるメソッド」の中の「//ターゲットの破壊エフェクトを出す」に下記を追記する

```
//ターゲットの破壊エフェクトを出す
effect();
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private void effect() { //ターゲットの破壊エフェクトを出す
    Instantiate(hitEffectPrefab, transform.position, transform.rotation);
}
```

② 「ファイル」「すべて保存」をクリック、Unity に戻り、▶をクリック
③ 弾をターゲットに当てると破壊音が流れエフェクトが表示されることを確認
④ Esc キーを押してから、▶を再度クリックすると終了

17. プログラミング：ステージクリアのチェック

① Visual Studio に戻り、「Target.cs」の「// 何かに接触したら呼び出されるメソッド」の中の「//ステージクリアにする」の下に下記１行を追記する

```
//ステージクリアにする
stageclear();
```

また「//サポートメソッド」の下に下記の通り追記する

```
//サポートメソッド
private void stageclear() { //ステージクリアにする
    Game.SetStageClear();
}
```

② 「ファイル」「すべて保存」をクリック、Unity に戻り、▶をクリック
③ 弾をターゲットに当て続けると最後の 1 個の破壊後「STAGE CLEAR」が表示され、移動や回転ができなくなることを確認
④ Esc キーを押してから、▶を再度クリックすると終了

パート 3 ちょっと上級編：もっと自由に

18. 操作の変更・追加

Player.cs の「マウスクリックによる射撃」を書き換えると、発射ボタンを右ボタンにしたり、ホイールボタンにしたり、どちらでも OK にできます。試してみてください。
ヒント：

```
Input.GetButtonDown("Fire2")   //右ボタン
Input.GetButtonDown("Fire3")   //ホイールボタン
Input.GetButtonDown("Fire1") || Input.GetButtonDown("Fire2")   //左右ボタンどちらでも
```

19. ゲーム感の調整

各ソースにゲーム感を調整する数値が入っています。下記を探して１つずつ変更と実行を繰り返して、皆さんにとってベストなゲーム感にしてみてください。
なお、ほとんど変化しないものもありますし、大幅に影響するものもあります(３D 酔いにはご注意を)。

```
private static readonly float MOVE_Z_FRONT = 5.0f; // 前進の速度
private static readonly float MOVE_Z_BACK = -2.0f; // 後退の速度
private static readonly float ROTATION_Y_KEY = 360.0f; // 回転の速度(キーボード)
```

private static readonly float ROTATION_Y_MOUSE = 360.0f; // 回転の速度(マウス)

private static readonly float bulletMoveSpeed = 10.0f; // 弾が 1 秒間に進む距離

public float fTimeLimit = 10f; //各プレハブの生存時間

public float rotationAdd = 45f; //ターゲットが 1 秒間に回転する量
private float rotationNow = 180f; //回転量

private static readonly int TARGET_NUM = 5;　//破壊するターゲットの数
private static readonly int MAZE_LINE_X = 8;　//迷路の X 通路数
private static readonly int MAZE_LINE_Y = 8;　//迷路の Y 通路数
private static readonly float MAZE_BLOCK_SCALE = 2.0f;　//迷路のスケール(ブロック 1 つ分のサイズ)

以上