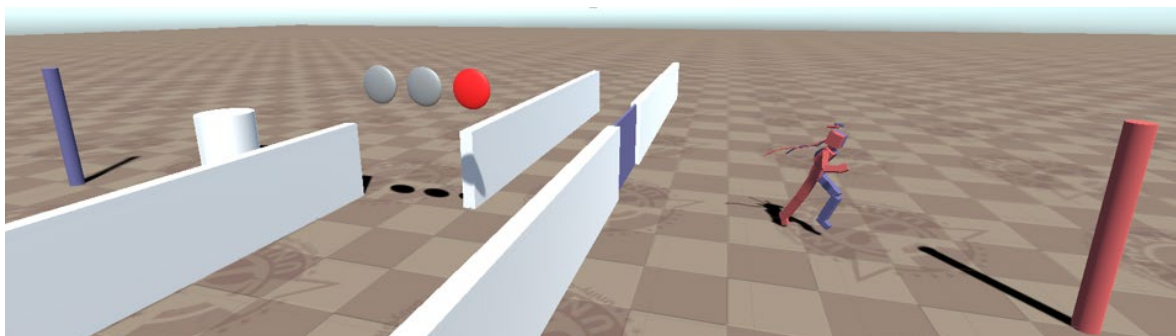


NPC（ノンプレイヤーキャラクター）

プレイヤーが操作しないキャラクター（NPC：ノンプレイヤーキャラクター）の構築に**ナビメッシュ（ナビゲーションメッシュ）**という機能を用います。Unity 独自の考え方ではなく、多くのゲームエンジンに搭載されており、ライブラリとしても一般的な呼び方なので、ゲームエンジン以外の開発現場でも一般的な用語となっています。

ある意味でAI機能であり、目的地への最短経路の計算はお手の物ですし、壁や通行人など障害物をよけて移動することもできます。途中で人と出会おうとしゃべりやバトルを始めたり、垂直のハシゴを登ったりも可能になります。



今回はそういったNPCの運営方法として、2点間の単純な往復運動に始まり、動く障害物や信号、自動ドアを回避させ、高さもジャンプで克服できる運営に発展させます。

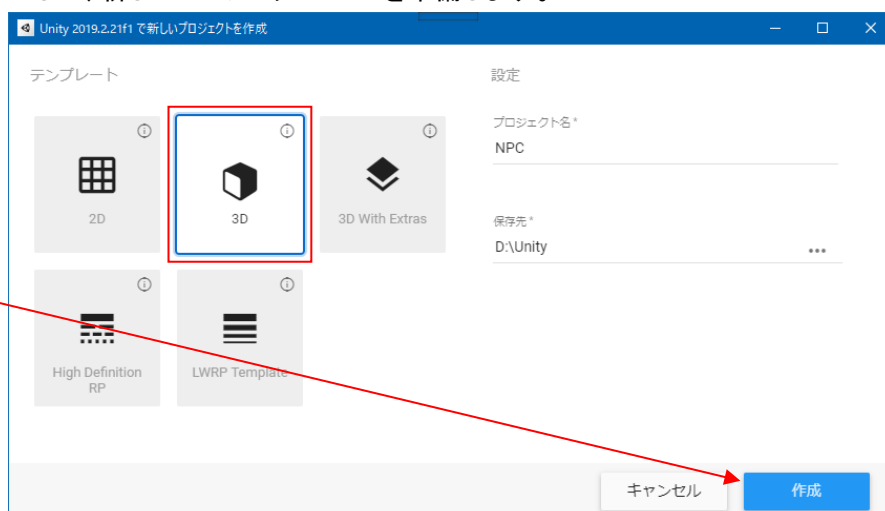
心当たりがありませんか？ ゲームの空間で村人や敵がウロウロしていたり、近寄ったら反応して攻撃してきたり、タワーディフェンスゲームで、自身の陣地に向かって攻めて来るモンスターなど、様々なシチュエーションでナビメッシュ技術は使われています。これからは心当たりのある場面で、これが動いているのかも？などと想像しながらゲームをするようになると思います。

シーンを構築する

【STEP1】プロジェクトの作成

- Unity を起動し、テンプレートを3Dにして、新しいプロジェクト **NPC** を準備します。

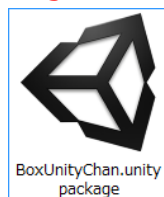
ボタン**作成**を押下します。



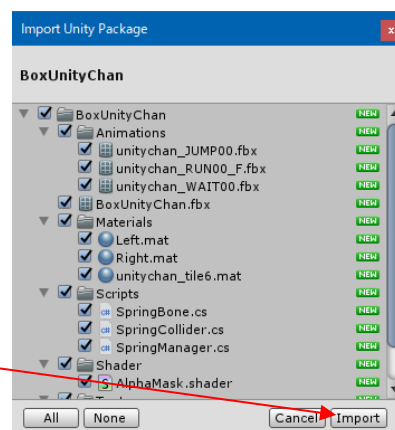
【STEP2】 素材の読み込み

配布された素材を読み込みます。

- メニューAssets から Import Package > **Custom Package** と選択し、**BoxUnityChan.unitypackage** を指定します。

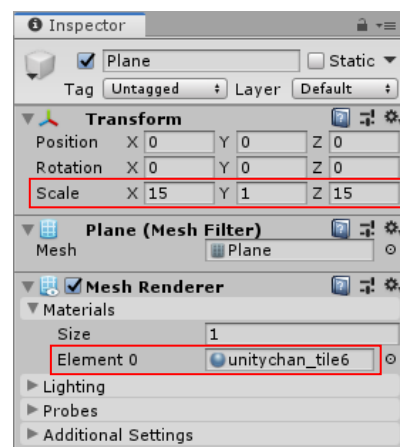
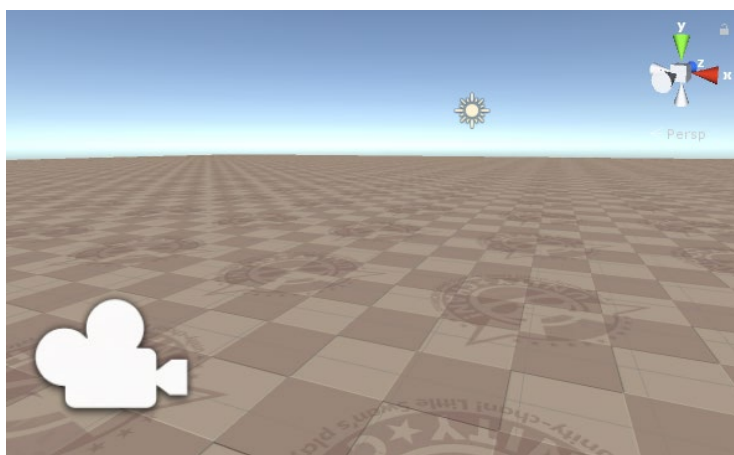


- 内容物が表示されたら、**Import** を押下します。

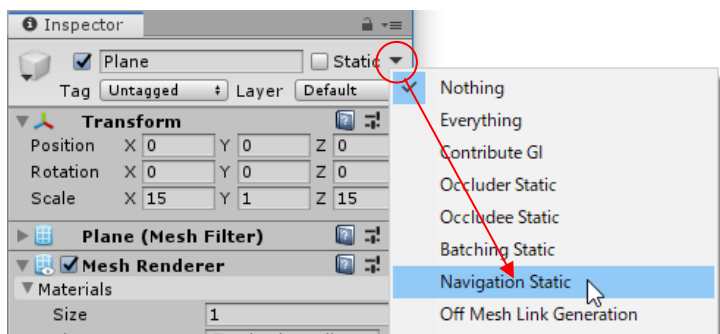


【STEP3】 ステージの設定

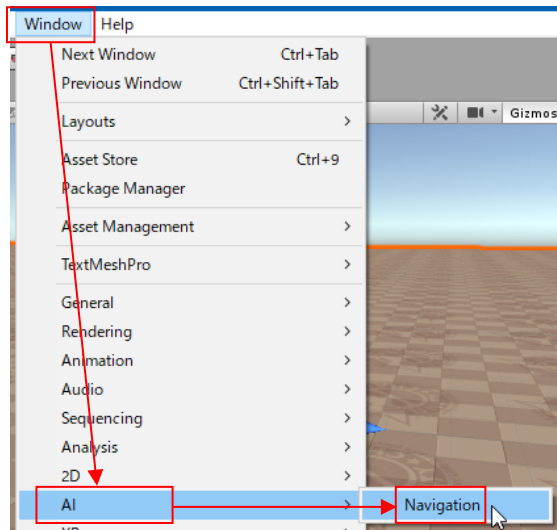
- ヒエラルキー欄の Create(クリエイト)から 3D Object > **Plane(プレーン)** を選択します。インスペクタでパラメータを設定します。



- ヒエラルキー欄の **Plane** を選択し、インスペクタ欄の **Static**▼マークを押下し、**Navigation Static** をクリックします。

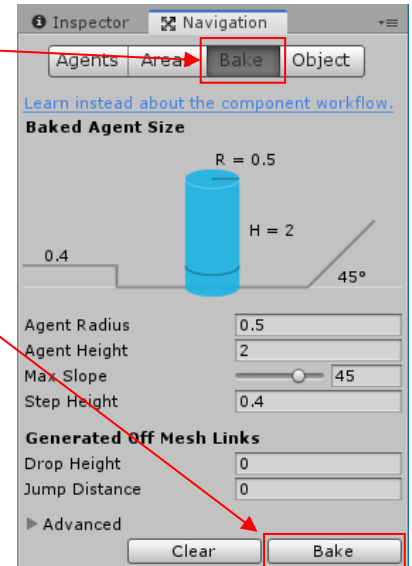
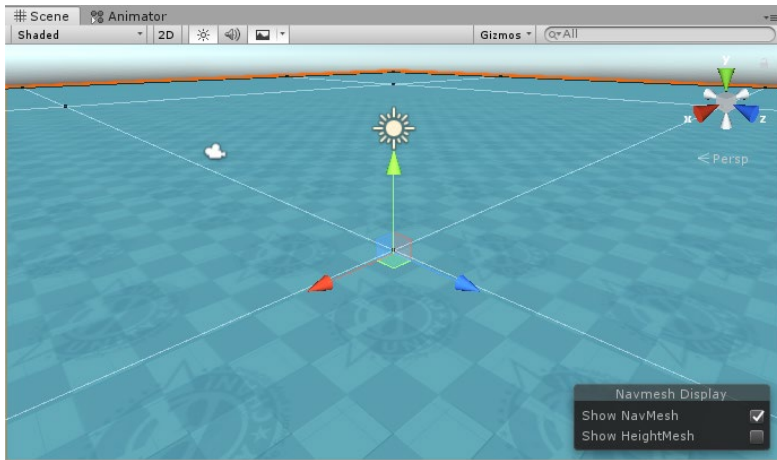


- メニューWindow(ウィンドウ)から AI > **Navigation(ナビゲーション)** を選択します。



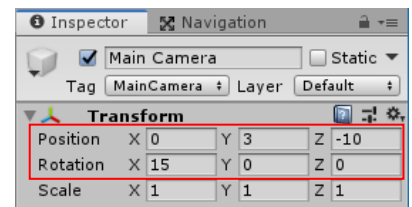
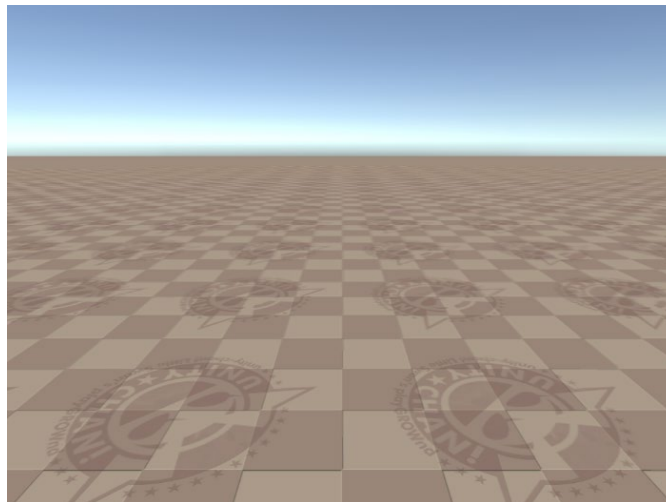
- **Bake(ベイク)**のタブを選びます。

パラメータはそのままボタン **Bake** を押下します。



青いセロファンのような領域が定義されます。
この面積の中で最短距離を計算するようになります。

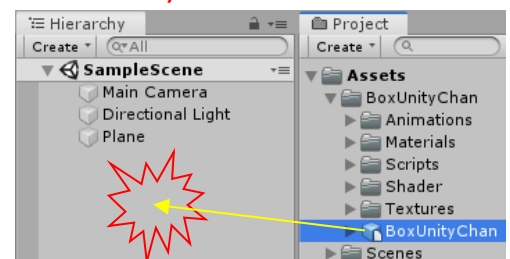
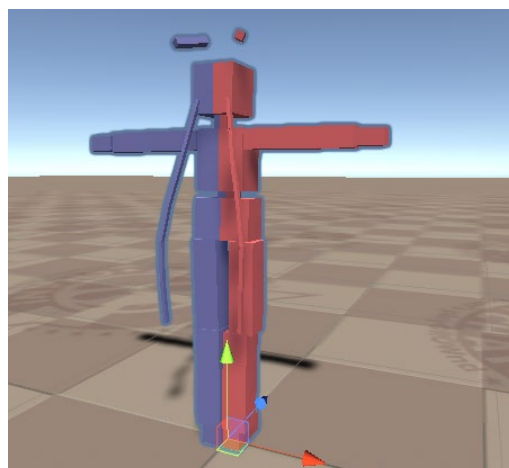
- ヒエラルキー欄の **Main Camera** を選択し、インスペクタでパラメータを設定します。



キャラクターを設定する

【STEP4】キャラクターの設置

- プロジェクト欄のフォルダ BoxUnityChan(ボックスユニティちゃん) > **BoxUnityChan** をヒエラルキー欄にドラッグ&ドロップします。

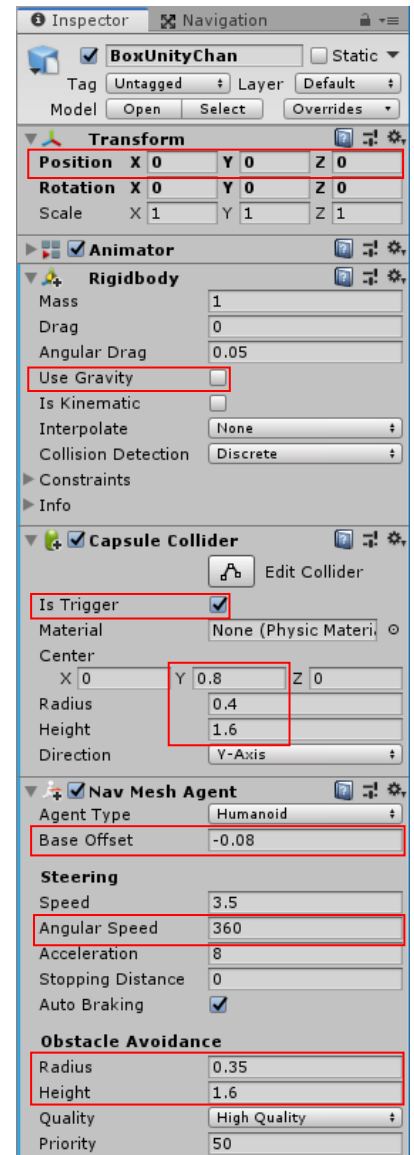


- 念の為、インスペクタで確認し、**原点(0,0,0)**にあるか？を確認します。

- インスペクタの最下段のボタン **Add Component** から Physics > **Rigidbody** を選択します。
インスペクタでパラメータを設定します。

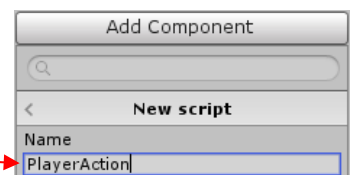
- 同様にインスペクタの **Add Component** から Physics > **Capsule Collider** を選択します。
インスペクタでパラメータを設定します。

- 同様にインスペクタの **Add Component** から Navigation > **Nav Mesh Agent** を選択します。
インスペクタでパラメータを設定します。



【STEP5】ナビゲーションメッシュ機能で目的地へ向かわせる

- ヒエラルキー欄の BoxUnityChan を選択し、インスペクタの Add Component から最下段の **New script** を選択します。
名称を **PlayerAction** と命名します。



- スクリプト **PlayerAction** を次のように編集します。

まずはナビメッシュ機能を感じてもらおうべく、仮の座標(4,0,4)に自動で向かってもらいます。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI; //ナビメッシュを使うのに必要

public class PlayerAction : MonoBehaviour {

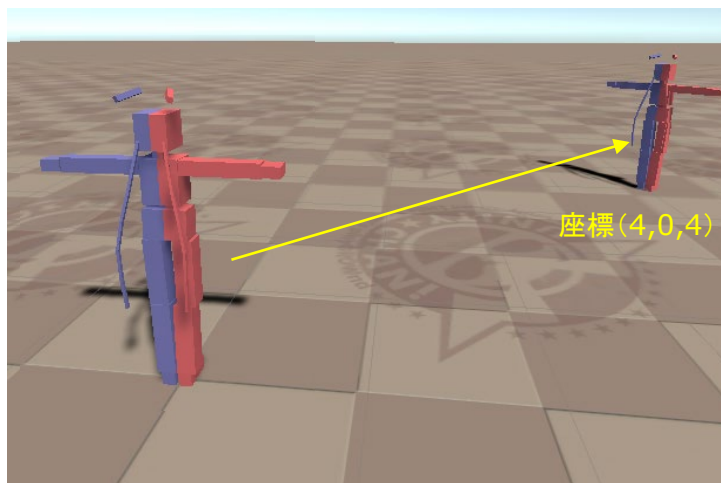
    NavMeshAgent myAgent; //自身のナビメッシュ

    void Start() {
        //ナビメッシュコンポーネントを取得
        myAgent = GetComponent<NavMeshAgent>();
        //仮の行き先座標
        myAgent.SetDestination(new Vector3(4,0,4));
    }

    //〜後略〜
}
```

- プレイボタンを押下します。 

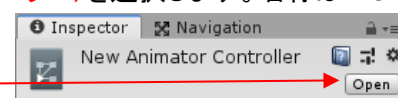
ボックスユニティちゃんが座標(4,0,4)に向かって移動することを確認します。



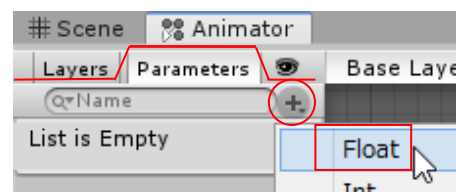
【STEP6】 ステートマシンを組む

- プロジェクト欄の Create から **Animator Controller(アニメーターコントローラー)** を選択します。名称は New Animator Controller のままでOKです。

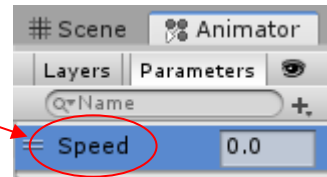
Open(オープン)ボタンを押下して編集状態に入ります。



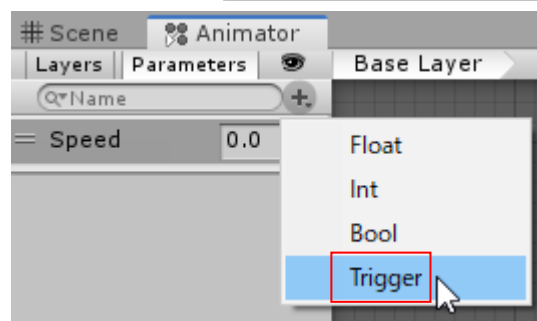
- パラメータタブを選択し、プラスボタン(+)から **Float(フロート:小数)** を選択します。



新しく出来たパラメータを **Speed(スピード)** と命名します。



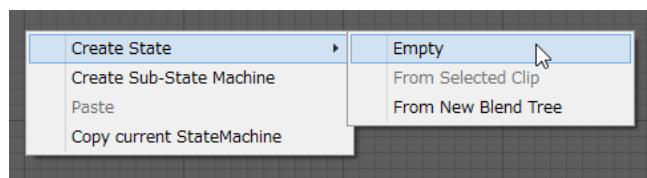
- 同様にして、プラスボタン(+)から **Trigger(トリガー)** を選択します。



- 新しく出来たパラメータを **Jump(ジャンプ)** と命名します。



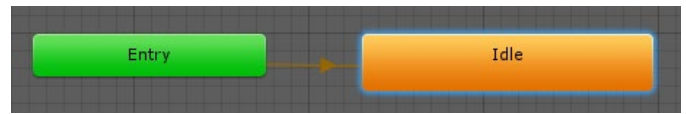
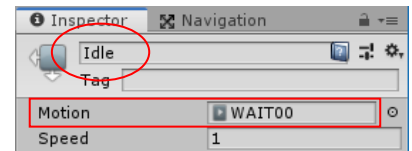
- 背景のグレーのグリッドを右クリックし、Create State(クリエイト ステート:状態作成) > **Empty(EMPTY:空っぽ)** を選択します。



- 作成された状態(オレンジ色の箱)を選択し、インスペクタでパラメータを設定します。

名称:Idle(アイドル)

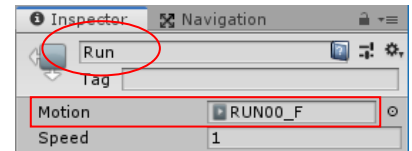
モーション:WAIT00



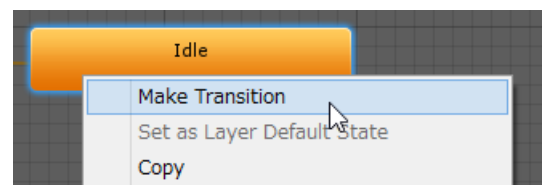
- 同様にして新しい「状態」を作成します。
インスペクタでパラメータを設定します。

名称:Run(ラン)

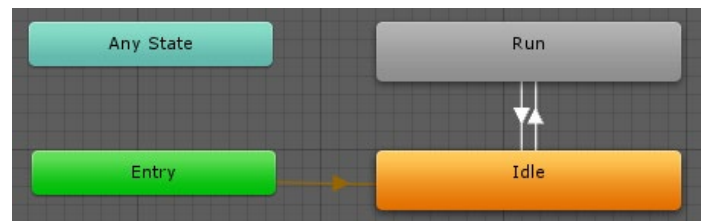
モーション:RUN00_F



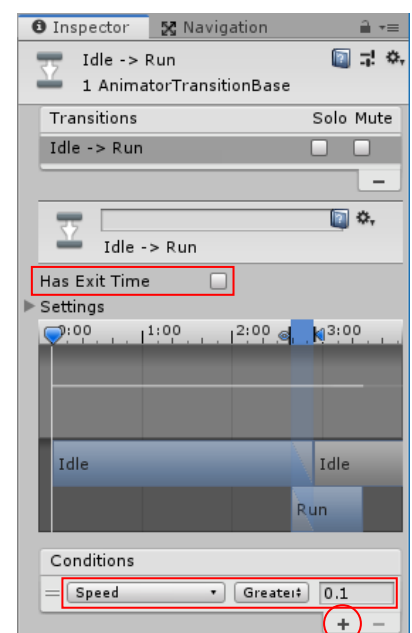
- オレンジの箱、つまり状態 Idle(アイドル)を右クリックし、**Make Transition(メイク トランジション)**を選んだら、マウスに白い矢印がくっついてくるので、状態 Run(ラン)をクリックして接続します。



- 同様にして、逆向きとなる Run(ラン)から Idle(アイドル)への白い矢印を構成して、両方の状態を行き来できるようにします。

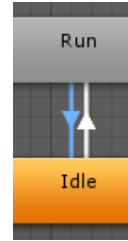


- 白い矢印 Idle -> Run をクリックします。

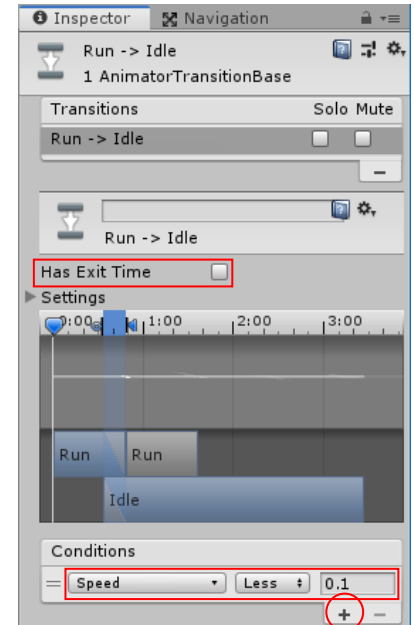


- チェックボックス **HasExitTime** をオフにします。
- Conditions(コンディションズ:条件)のプラスボタン(+)を押下します。
Speed が 0.1 を超えたら(Greater:グレーター)の条件を設定します。

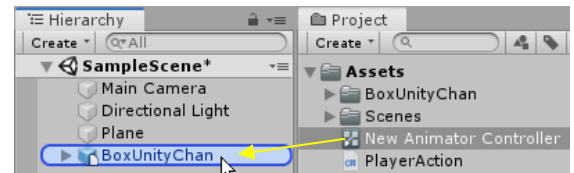
- 白い矢印 **Run -> Idle** をクリックします。



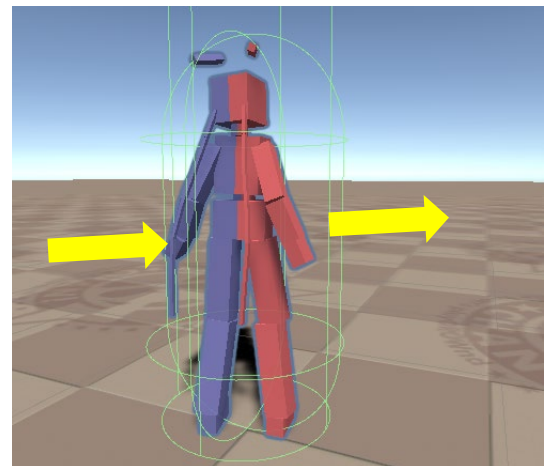
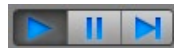
- チェックボックス **HasExitTime** をオフにします。
- Conditions(コンディションズ:条件)のプラスボタン(+)を押下します。
Speed が **0.1** を下回ったら(Less:レス)の条件を設定します。



- 先ほど作った **New Animator Controller** をヒエラルキー欄の **BoxUnityChan** にドラッグ & ドロップします。



- プレイボタンを押下します。
ボックスユニティちゃんが待機モーション(WAIT00)になったまま目的地へ移動することを確認します。



【STEP7】 ナビメッシュ移動に走行を同調させる

- スクリプト **PlayerAction** を次のように編集します。

```
//～前略～
NavMeshAgent myAgent; //自身のナビメッシュ
Animator myAnim; //自身のアニメーターコントローラー

void Start() {
    //アニメーターコンポーネントを取得
    myAnim = GetComponent<Animator>();
    //ナビメッシュコンポーネントを取得
    myAgent = GetComponent<NavMeshAgent>();
    //仮の行き先座標
    myAgent.SetDestination(new Vector3(4,0,4));
}

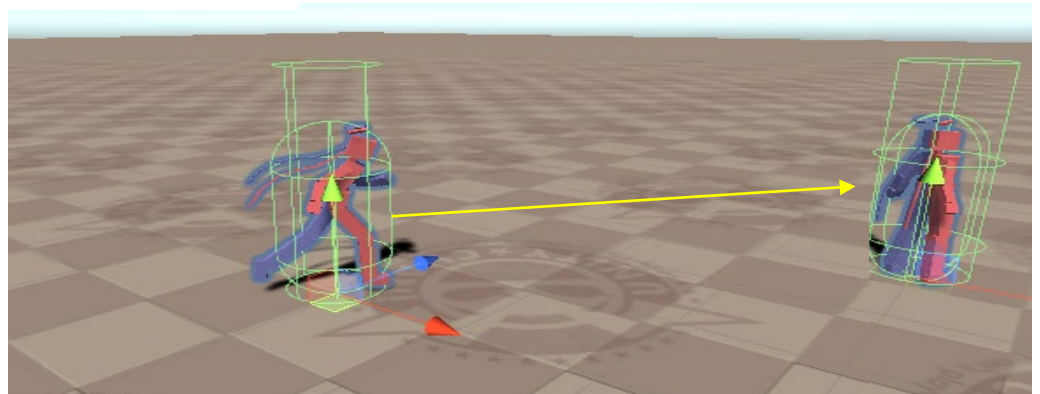
void Update() {

    //ナビメッシュの移動速度をアニメーターのSpeedへ送る
    if (myAgent.enabled) {
        myAnim.SetFloat("Speed", myAgent.velocity.magnitude);
    } else {
        myAnim.SetFloat("Speed", 0);
    }
}
```

- プレイボタンを押下します。



ボックスユニティちゃんが座標(4,0,4)に向かって走り、到着したら呼吸待機となることを確認します。



【STEP8】 クリック地点への移動指示

- 決められた座標ではなく、ユーザーがクリックした位置へ向かわせる仕様に変更します。ジョイスティックの無いスマートデバイスでは、非常に多用されるキャラクター移動指示となりました。
スクリプト **PlayerAction** を次のように編集します。

```
//～前略～
NavMeshAgent myAgent; //自身のナビメッシュ
Animator myAnim; //自身のアニメーターコントローラー
Camera Cam; //カメラ

void Start() {
    //カメラオブジェクトを探してくる
    GameObject CamObj = GameObject.FindGameObjectWithTag( "MainCamera" );
    //カメラコンポーネントを取得
    Cam = CamObj.GetComponent<Camera>();
    //アニメーターコンポーネントを取得
    myAnim = GetComponent<Animator>();
    //ナビメッシュコンポーネントを取得
    myAgent = GetComponent<NavMeshAgent>();
    //仮の行き先座標
    myAgent.SetDestination( new Vector3( 4, 0, 4 ) );
}

void Update() {

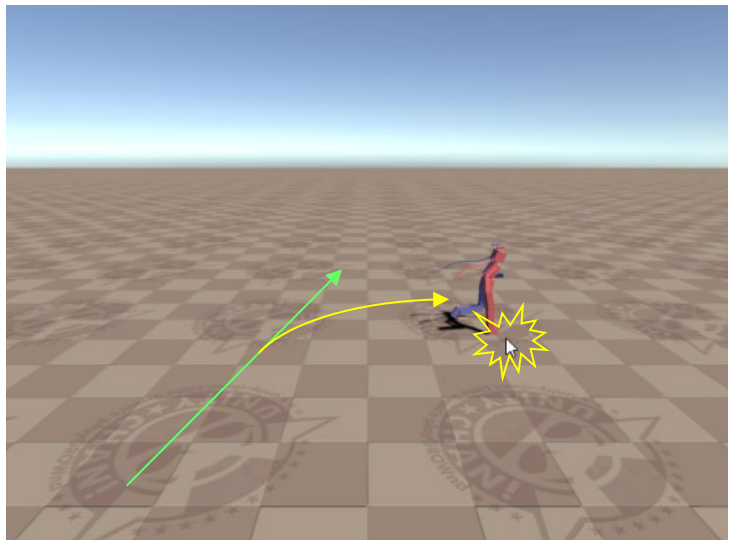
    //ナビメッシュが有効なら左クリック位置へ向かわせる
    if (Input.GetMouseButtonDown( 0 ) && myAgent.enabled) {
        RaycastHit hitInfo;
        Ray screenRay = Cam.ScreenPointToRay( Input.mousePosition );
        if (Physics.Raycast( screenRay, out hitInfo )) {
            myAgent.SetDestination( hitInfo.point );
        }
    }
}

//～後略～
```

- プレイボタンを押下します。

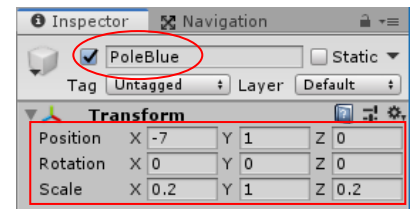


指示通りに座標(4,0,4)へ向かい始めるのですが、Game 画面で床をクリックすると、そちらへ向かい始めます。
到着すれば呼吸待機となるので、また別の場所を指示してやると移動を繰り返します。

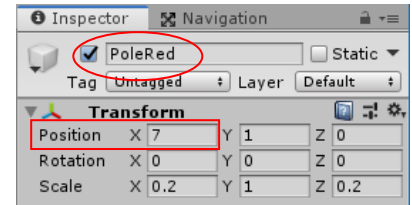


【STEP9】2地点間の自動往復

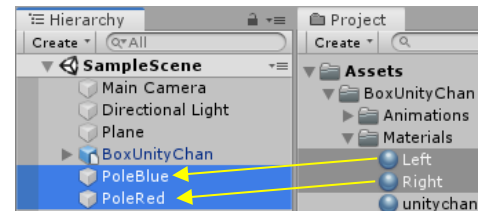
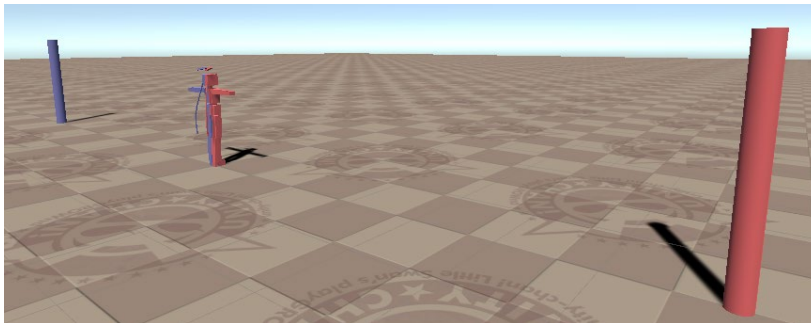
- ヒエラルキー欄の Create から 3D Object > **Cylinder**(シリンダー)を選択し、名称を **PoleBlue**(ポールブルー)とします。
インスペクタでパラメータを設定します。



- この **PoleBlue** を複製 (Ctrl + D) し、名称を **PoleRed**(ポールレッド)とします。
インスペクタで位置を変更します。



- プロジェクト欄の BoxUnityChan > Materials(マテリアルス:質感)の中にマテリアル **Left** と **Right** があり、それぞれヒエラルキー欄の **PoleBlue**(ポールブルー) と **PoleRed**(ポールレッド)にドラッグ&ドロップして割り当てます。



- この2地点間を自動的に往復するように、目的地を一つではなく2つ与え、到着すれば反対側が目的地になるような仕組みを実装します。スクリプト **PlayerAction** を次のように編集します。

```
//～前略～
int TargetID = 0;
public GameObject PoleBlue;
public GameObject PoleRed;

void Start() {
    //カメラオブジェクトを探してくる
    GameObject CamObj = GameObject.FindGameObjectWithTag( "MainCamera" );
    //カメラコンポーネントを取得
    Cam = CamObj.GetComponent<Camera>();
    //アニメーターコンポーネントを取得
    myAnim = GetComponent<Animator>();
    //ナビメッシュコンポーネントを取得
    myAgent = GetComponent<NavMeshAgent>();
    //仮の行き先座標
    myAgent.SetDestination( new Vector3( 4, 0, 4 ) );
}

void OnTriggerEnter(Collider other) {
    if (other.gameObject.name.Substring(0, 4) == "Pole") {
        TargetID ++; //目的地IDを変更
        TargetID %= 2;
    }
}

//続きます
```

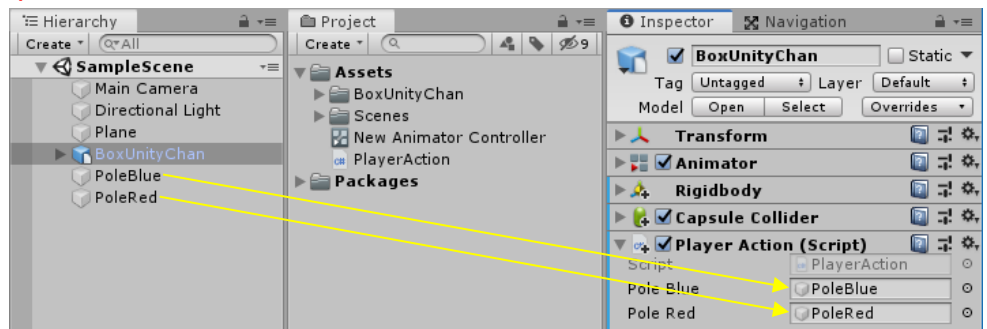
```
//続きです
```

```
void Update() {
```

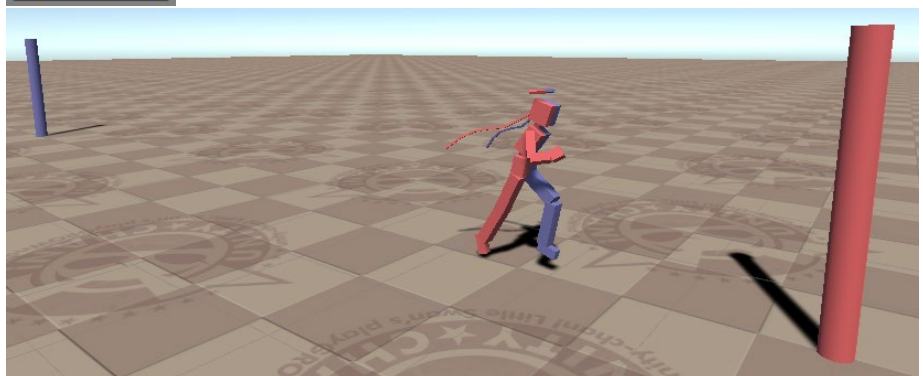
```
    //ナビメッシュが有効なら左クリック位置へ向かわせる
    if (Input.GetMouseButtonDown( 0 ) && myAgent.enabled) {
        RaycastHit hitInfo;
        Ray screenRay = Cam.ScreenPointToRay( Input.mousePosition );
        if (Physics.Raycast( screenRay, out hitInfo )) {
            myAgent.SetDestination( hitInfo.point );
        }
    }
}
```

```
    //ナビメッシュの移動速度をアニメーターのSpeedへ送る
    if (myAgent.enabled) {
        //目的地を切り替える
        if (TargetID == 0) {
            myAgent.SetDestination( PoleRed.transform.position );
        }
        else if (TargetID == 1) {
            myAgent.SetDestination( PoleBlue.transform.position );
        }
        myAnim.SetFloat("Speed", myAgent.velocity.magnitude);
    } else {
        myAnim.SetFloat("Speed", 0);
    }
}
```

- ヒエラルキー欄の **BoxUnityChan** を選択し、インスペクタに登場した項目に、2つの円柱オブジェクトをドラッグ & ドロップします。



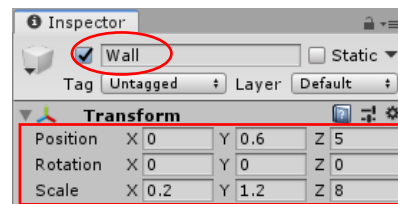
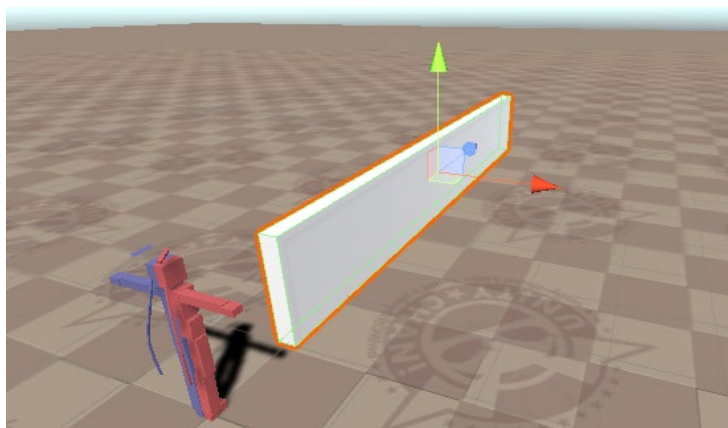
- プレイボタンを押下します。
赤青2つの円柱の間を自動的に
に往復する動作になることを
確認します。



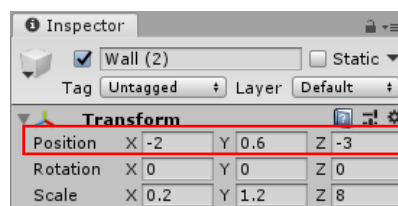
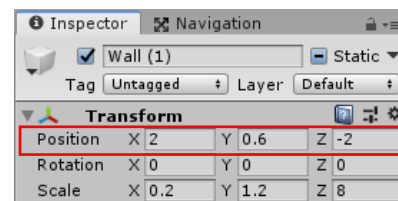
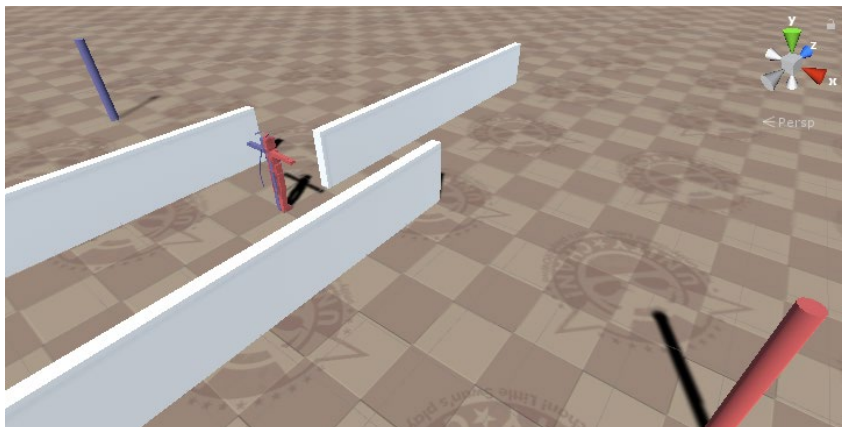
最初の仮位置やマウスでクリックした位置へ向かわせる処理は残っていますが、毎フレームでどちらかのポールに向かうように指示していますので、行き先が上書きされている状況となります。

【STEP10】 障害物を避けた経路の探索

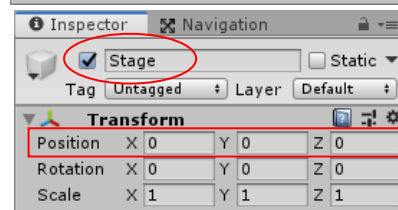
- ヒエラルキー欄の Create(クリエイト)から 3D Object > **Cube(キューブ:立方体)**を選択し、名称を **Wall(ウォール:壁)**に変更します。インスペクタでパラメータを設定します。



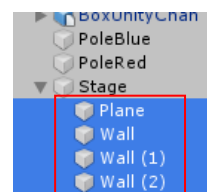
- ヒエラルキー欄の **Wall(ウォール)**を2回複製(Ctrl + D)し、**計3個**にします。名前はそのまま構いません。複製されたオブジェクトについて、インスペクタでそれぞれ位置を修正します。



- ヒエラルキー欄の Create から Create Empty を選択し、名称を **Stage** とします。必ず、原点にあるか？を確認します。

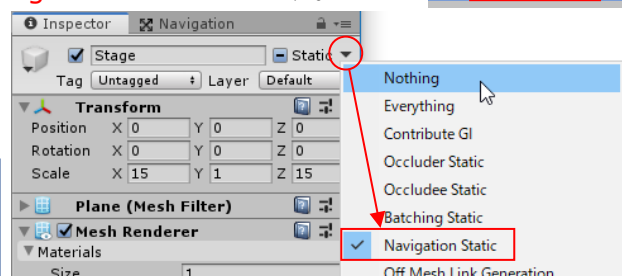
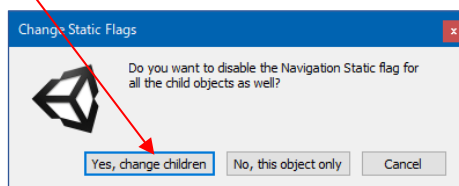


- 今までに制作したステージに関するオブジェクトについて、ヒエラルキー欄の **Stage の配下** になるようにドラッグ & ドロップします。2つのポールは含めません。



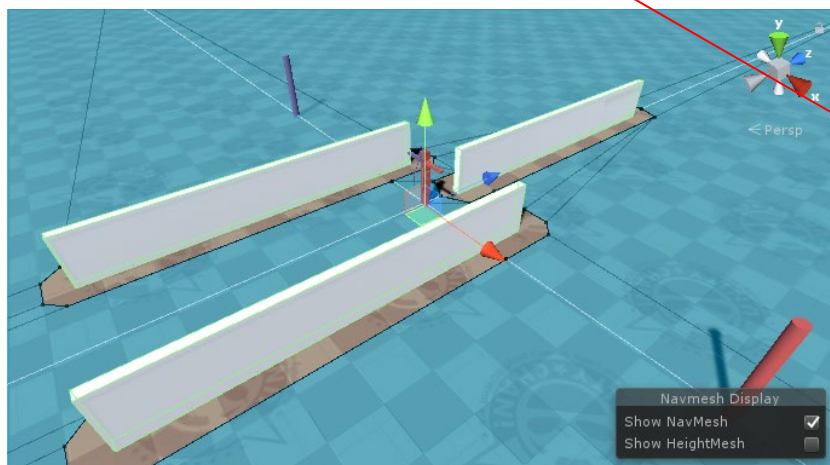
- この Stage について、インスペクタの Static 項目を **Navigation Static** にします。

作業の都度、子構造も同様に処理するか？を訪ねて来るので、**Yes** を選択します。

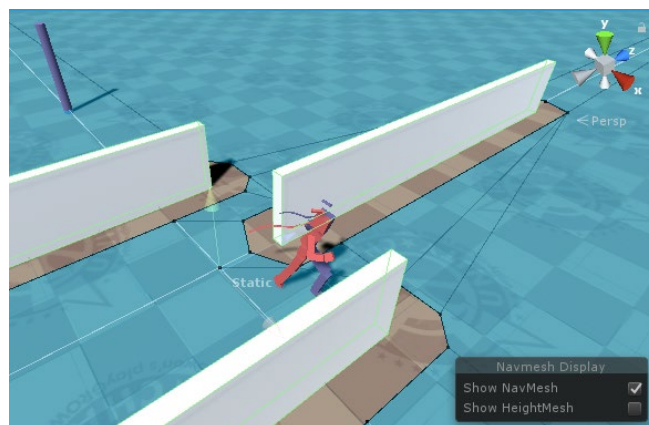
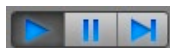


- メニューWindow(ウィンドウ)から AI > **Navigation(ナビゲーション)** を選択します。
- **Bake(ベイク)** のタブを選びます。

パラメータはそのままボタン **Bake** を押下します。

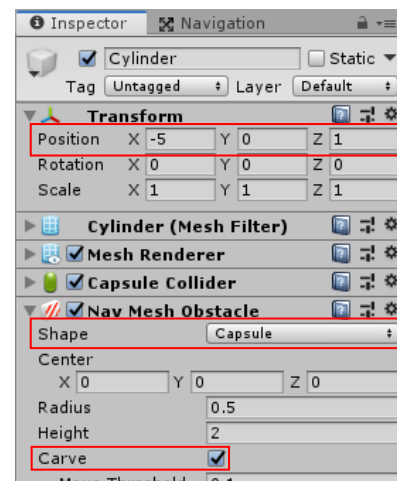
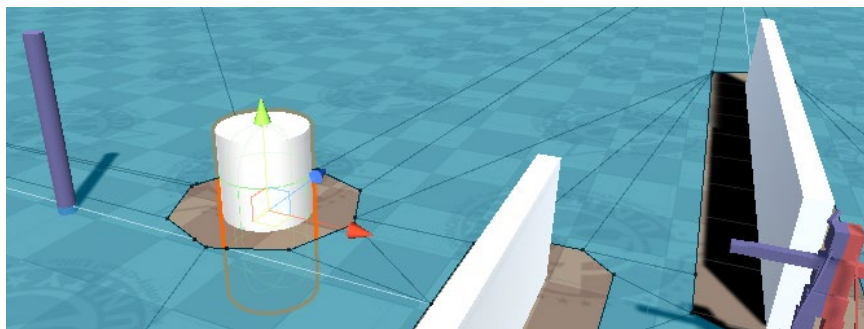


- プレイボタンを押下します。
キャラクターがナビメッシュの面積を用いて経路を計算し、結果的に障害物を避けるように往復運動することが判ります。

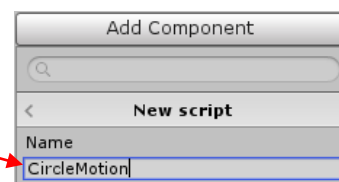


【STEP11】 動く障害物を避ける

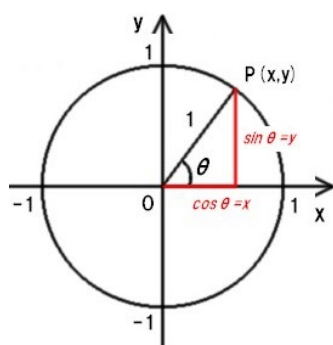
- ヒエラルキー欄の Create から 3D Object > **Cylinder** を選択し、インスペクタでパラメータを設定します。
- インスペクタの Add Component から Navigation > **Nav Mesh Obstacle** を選択します。
インスペクタでパラメータを設定します。



- この Cylinder を選択している状態でインスペクタの Add Component から **New script** を選択し、名称を **CircleMotion** とします。
- スクリプト **CircleMotion** を次のように編集します。



このオブジェクトを配置した座標Oを中心として、水平移動するPの円運動座標を加えるスクリプトとなります。



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CircleMotion : MonoBehaviour {

    Vector3 InitPos;

    void Start() {
        InitPos = transform.position;
    }

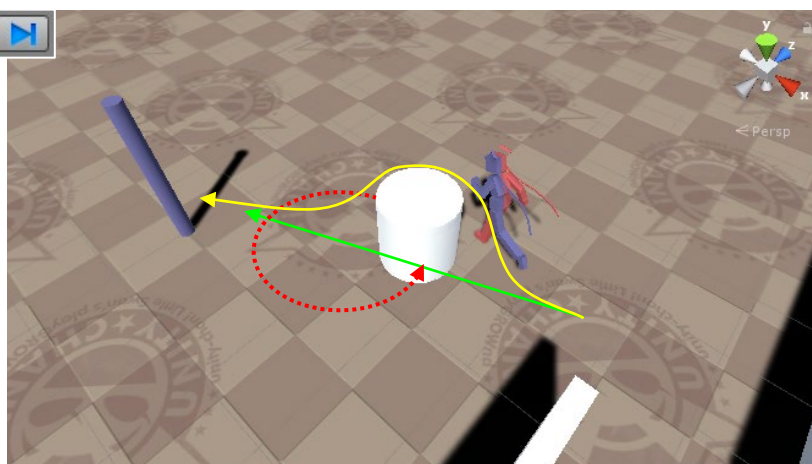
    void Update() {
        transform.position = new Vector3(
            InitPos.x + Mathf.Cos( Time.time ), 0,
            InitPos.z + Mathf.Sin( Time.time ) );
    }

}
```

- プレイボタンを押下します。



円柱が円運動をし、それを避けるようにキャラクターがリアルタイムに迂回することを確認します。

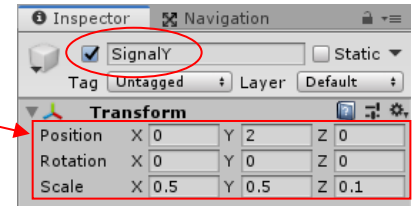


信号を運営して従わせる

キャラクターが往復運動をしている途中に信号機を設けます。この信号機が赤の時にはキャラクターが待機をし、青になったら移動を再開する仕組みとします。

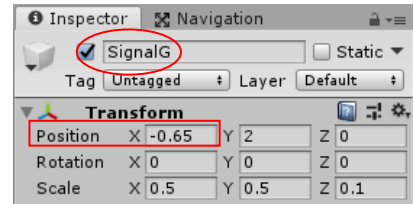
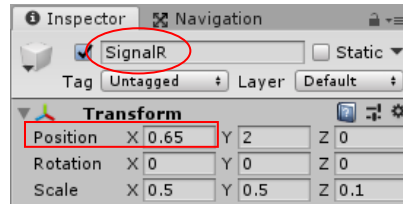
【STEP12】 信号の設置

- ヒエラルキー欄の Create から 3D Object > **Sphere** を選択し、名称を **SignalY** とします。
インスペクタでパラメータを設定します。

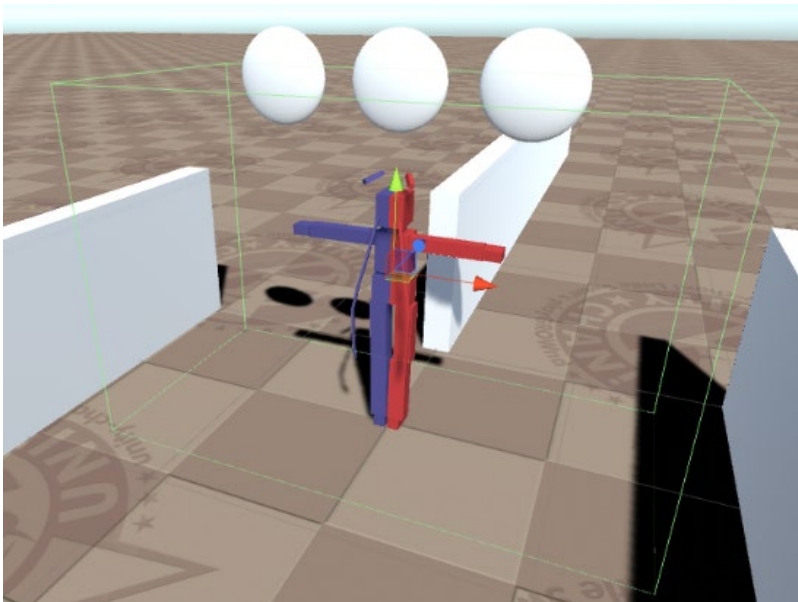
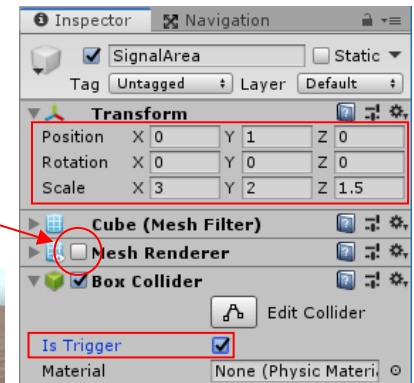


- この **SignalY** を2回複製 (Ctrl + D) し、計3個にします。

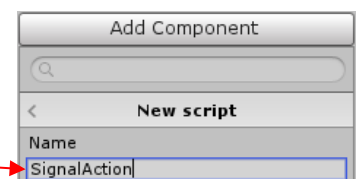
それぞれ名称を **SignalR**、**SignalG** と命名し、インスペクタで位置を変更します。



- ヒエラルキー欄の Create から 3D Object > **Cube** を選択し、名称を **SignalArea** とします。
インスペクタでパラメータを設定します。
見えないオブジェクトとなります。



- この **SignalArea** を選択し、インスペクタの Add Component から最下段の **New script** を選択します。
名称を **SignalAction** と命名します。



- スクリプト **SignalAction** を次のように編集します。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI; //ナビメッシュを使うのに必要

public class SignalAction : MonoBehaviour {

    public GameObject signalY;
    public GameObject signalR;
    public GameObject signalG;
    int signalStatus;
    float Elapsed = 0.0f;
    Color ColorR = new Color(1, 0, 0, 1.0f);
    Color ColorG = new Color(0, 1, 0, 1.0f);
    Color ColorY = new Color(0.8f, 0.8f, 0, 1.0f);
    Color ColorN = new Color(0.5f, 0.5f, 0.5f, 1.0f);

    void Start () {

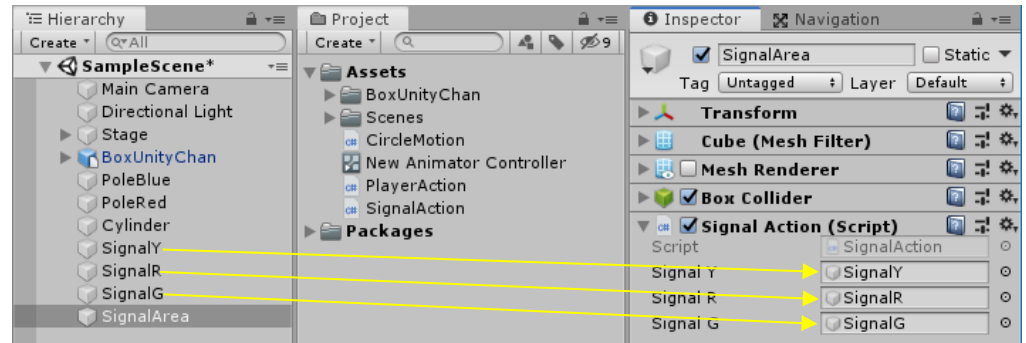
    }

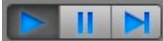
    void OnTriggerEnter(Collider other) {
        if (other.GetComponent<PlayerAction>()) {
            if (signalStatus == 2) {
                other.GetComponent<NavMeshAgent>().enabled = false;
            }
        }
    }

    void OnTriggerStay(Collider other) {
        if (other.GetComponent<PlayerAction>()) {
            if (signalStatus != 2) {
                other.GetComponent<NavMeshAgent>().enabled = true;
            }
        }
    }

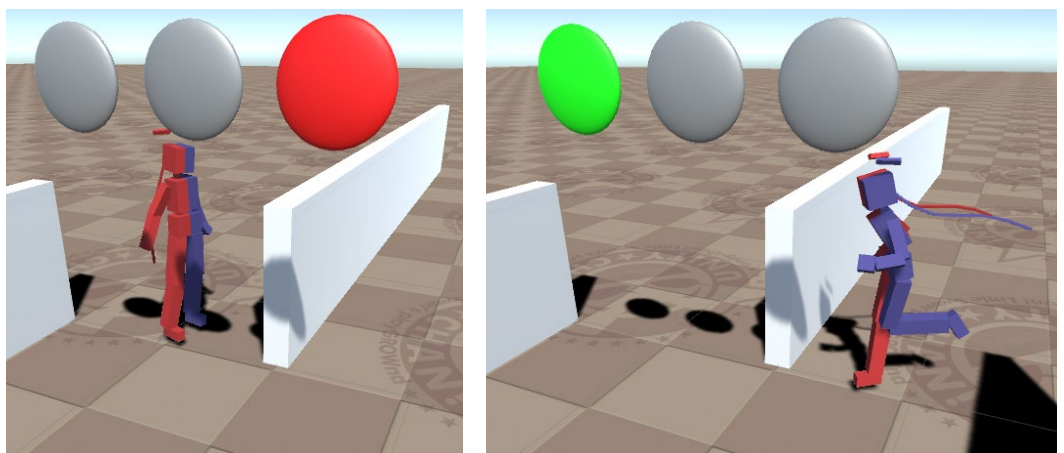
    void Update() {
        Elapsed += Time.deltaTime;
        Elapsed %= 18.0f;
        if (Elapsed < 8.0f) {
            signalStatus = 0;
            signalR.GetComponent<Renderer>().material.color = ColorN;
            signalG.GetComponent<Renderer>().material.color = ColorG;
            signalY.GetComponent<Renderer>().material.color = ColorN;
        } else if (Elapsed < 10.0f) {
            signalStatus = 1;
            signalR.GetComponent<Renderer>().material.color = ColorN;
            signalG.GetComponent<Renderer>().material.color = ColorN;
            signalY.GetComponent<Renderer>().material.color = ColorY;
        } else {
            signalStatus = 2;
            signalR.GetComponent<Renderer>().material.color = ColorR;
            signalG.GetComponent<Renderer>().material.color = ColorN;
            signalY.GetComponent<Renderer>().material.color = ColorN;
        }
    }
}
```

- ヒエラルキー欄の **SignalArea** を選択し、インスペクタに登場した3つの項目について、同名のオブジェクトをヒエラルキー欄からドラッグ & ドロップして設定します。



- プレイボタンを押下します。 

今まで通りキャラクターは青と赤の円柱を往復運動するのですが、信号部分が赤の時だけ、ちゃんと待機モードになり、青信号になれば再び走行を開始します。

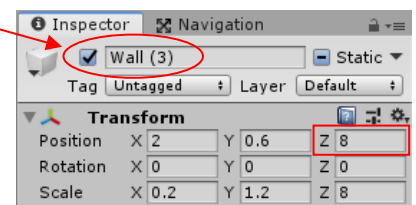
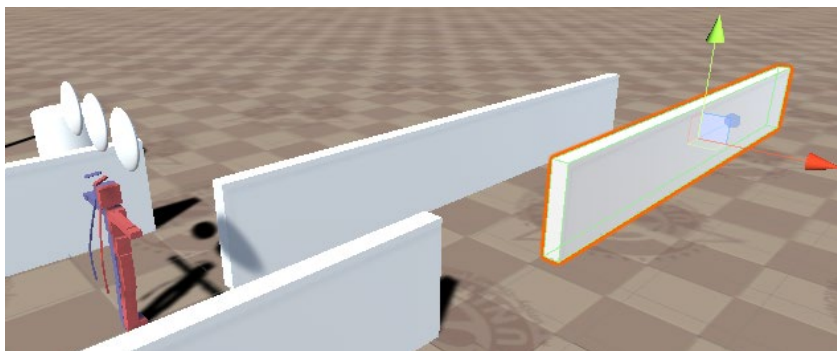


自動ドアと同調する

キャラクターの移動経路の途中に自動ドアを設けます。普段は閉まっていて、前に立つと自動的に開きます。キャラクターは開くまで待ってから通り抜け、ドアはキャラクターが去った後に自動的に閉まります。

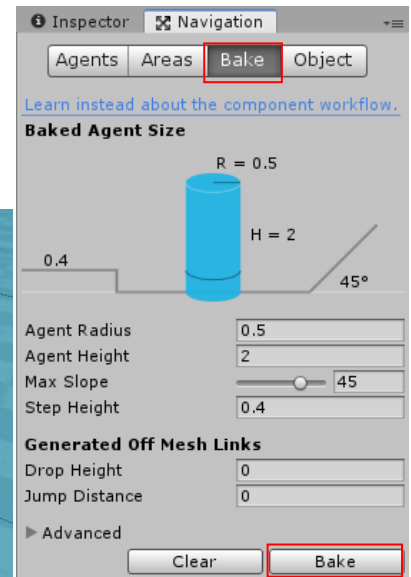
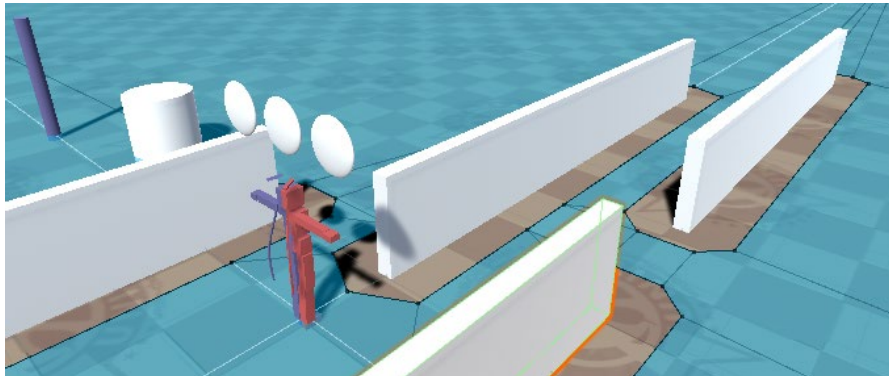
【STEP13】 自動ドアの設置

- ヒエラルキー欄の **Wall(1)** を複製 (Ctrl + D) します。名称は **Wall (3)** のはずです。インスペクタでパラメータを設定します。



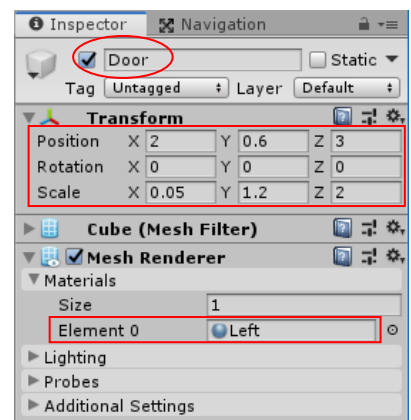
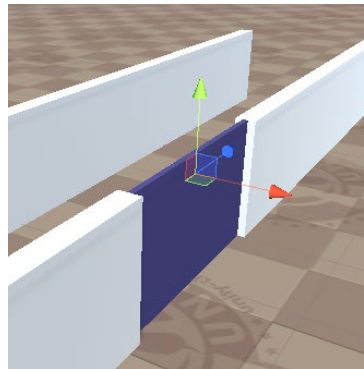
- メニューWindow(ウィンドウ)から AI > **Navigation(ナビゲーション)**を選択します。

- タブ Bake(ベイク)を選択し、ボタン **Bake(ベイク)**を押下します。



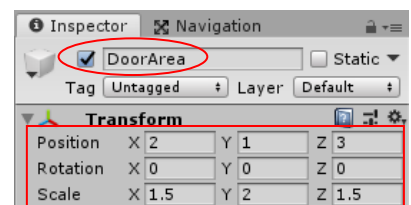
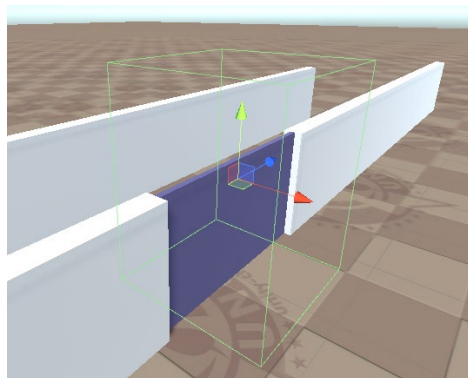
- ヒエラルキー欄の Create から 3D Object > **Cube(キューブ)**を選択し、名称を **Door(ドア)**に変更します。

インスペクタでパラメータを設定します。



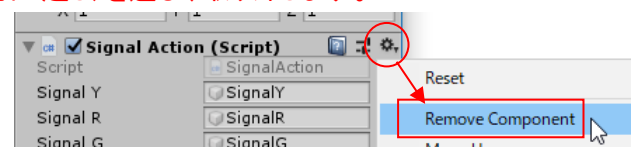
- ヒエラルキー欄の **SignalArea(シグナルエリア)**を複製(Ctrl+D)し、名称を **DoorArea(ドアエリア)**とします。

インスペクタでパラメータを設定します。

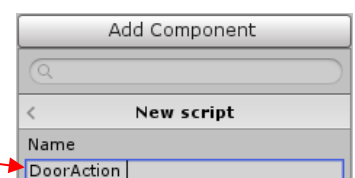


【STEP14】 自動ドアの運営

- この DoorArea(ドアエリア)を選択し、インスペクタで SignalAction(シグナルアクション)の歯車アイコンを押下し、**RemoveComponent(リムーヴ コンポーネント: 引っ越し)**を選び、取り外します。



- インスペクタの Add Component から **New script** を選択し、スクリプトの名称を **DoorAction** とします。



- スクリプト **DoorAction** を次のように編集します。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI; //ナビメッシュを使うのに必要

public class DoorAction : MonoBehaviour {

    public GameObject Door;
    bool isOpen = false;
    public float PosClose = 3.0f;
    public float PosOpen = 4.6f;

    void Start() {

    }

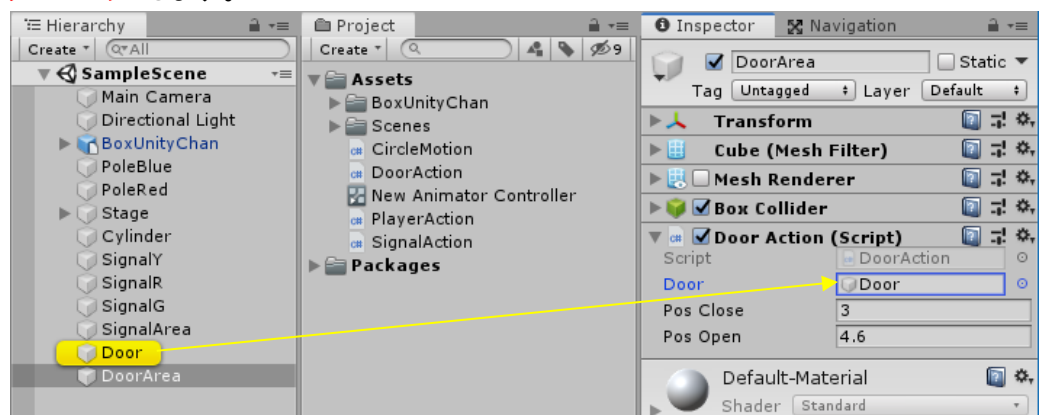
    void OnTriggerEnter(Collider other) {
        if (other.GetComponent<PlayerAction>()) {
            isOpen = true;
            other.gameObject.GetComponent<NavMeshAgent>().enabled = false;
        }
    }

    void OnTriggerStay(Collider other) {
        if (Door.transform.position.z >= PosOpen) {
            other.gameObject.GetComponent<NavMeshAgent>().enabled = true;
        }
    }

    void OnTriggerExit(Collider other) {
        if (other.GetComponent<PlayerAction>()) {
            isOpen = false;
        }
    }

    void Update () {
        if (isOpen && Door.transform.position.z <= PosOpen) {
            Door.transform.position += new Vector3(0,0,3 * Time.deltaTime);
        }
        if (!isOpen && Door.transform.position.z >= PosClose) {
            Door.transform.position -= new Vector3(0, 0, 3 * Time.deltaTime);
        }
    }
}
```

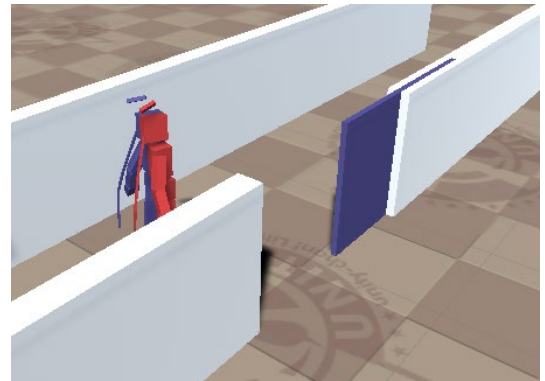
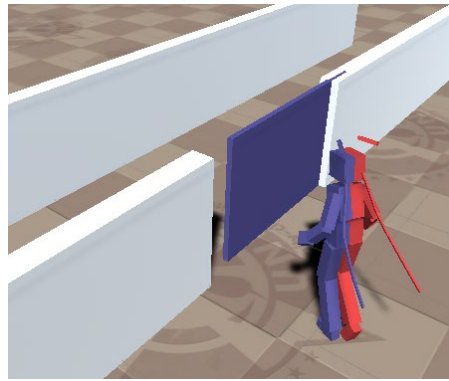
- ヒエラルキー欄の **DoorArea(ドアエリア)** を選択し、インスペクタに登場した項目 **Door(ドア)** にヒエラルキー欄の **Door(ドア)** をドラッグ & ドロップします。



- プレイボタンを押下します。



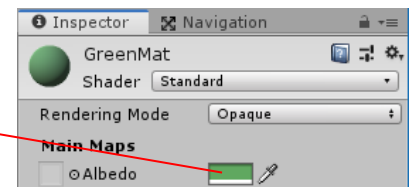
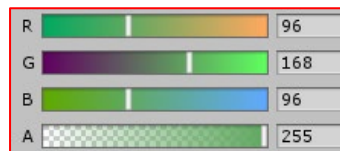
自動ドアの手前で立って待つと、ドアが自動で開きます。通り抜けた後、ドアは自動的に閉まります。



高低差のある目的地

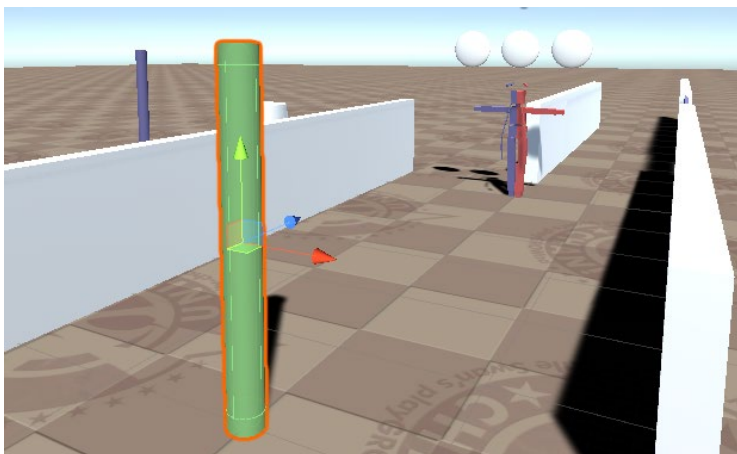
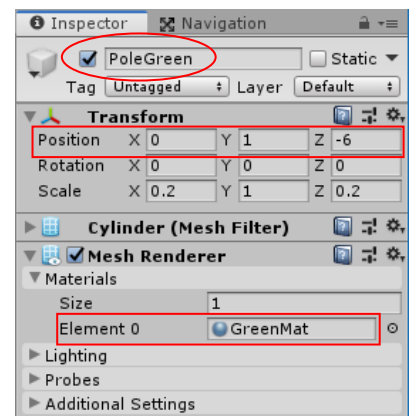
【STEP15】 目的地の追加

- プロジェクト欄の BoxUnityChan > Materials > **Left** を複製 (Ctrl + D) し、名称を **GreenMat** とします。
Albedo の色を緑にします。



- ヒエラルキー欄の **PoleRed** を複製 (Ctrl + D) し、名称を **PoleGreen** とします。

インスペクタでパラメータを設定します。



- スクリプト **PlayerAction** を次のように編集します。

3にします。

```
//～前略～

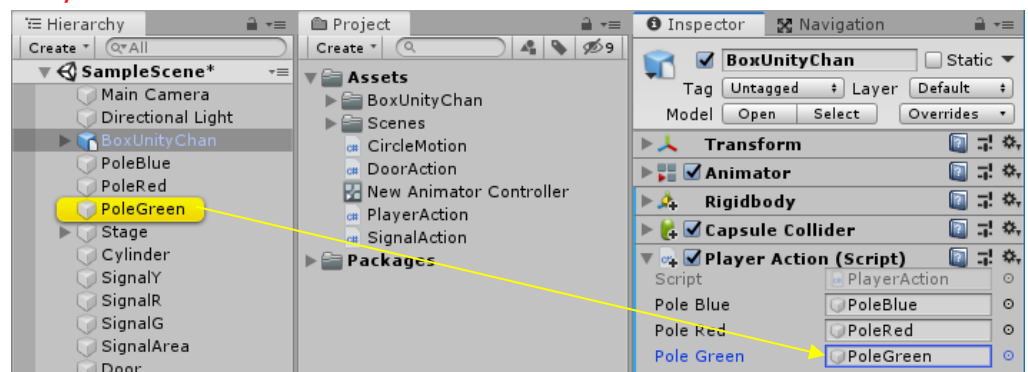
public GameObject PoleGreen;

void OnTriggerEnter(Collider other) {
    if (other.gameObject.name.Substring( 0, 4 ) == "Pole") {
        TargetID++;
        TargetID %= 3;
    }
}

void Update() {
    //ナビメッシュが有効なら左クリック位置へ向かわせる
    if (Input.GetMouseButtonDown( 0 ) && myAgent.enabled) {
        RaycastHit hitInfo;
        Ray screenRay = Cam.ScreenPointToRay( Input.mousePosition );
        if (Physics.Raycast( screenRay, out hitInfo )) {
            myAgent.SetDestination( hitInfo.point );
        }
    }

    //ナビメッシュの移動速度をアニメーターのSpeedへ送る
    if (myAgent.enabled) {
        //目的地を切り替える
        if (TargetID == 0) {
            myAgent.SetDestination( PoleRed.transform.position );
        }
        else if (TargetID == 1) {
            myAgent.SetDestination( PoleBlue.transform.position );
        }
        else if (TargetID == 2) {
            myAgent.SetDestination( PoleGreen.transform.position );
        }
        myAnim.SetFloat( "Speed", myAgent.velocity.magnitude );
    } else {
        myAnim.SetFloat( "Speed", 0 );
    }
}
}
```

- ヒエラルキー欄の **BoxUnityChan** を選択し、ヒエラルキー欄の項目に **PoleGreen** をドラッグ & ドロップします。

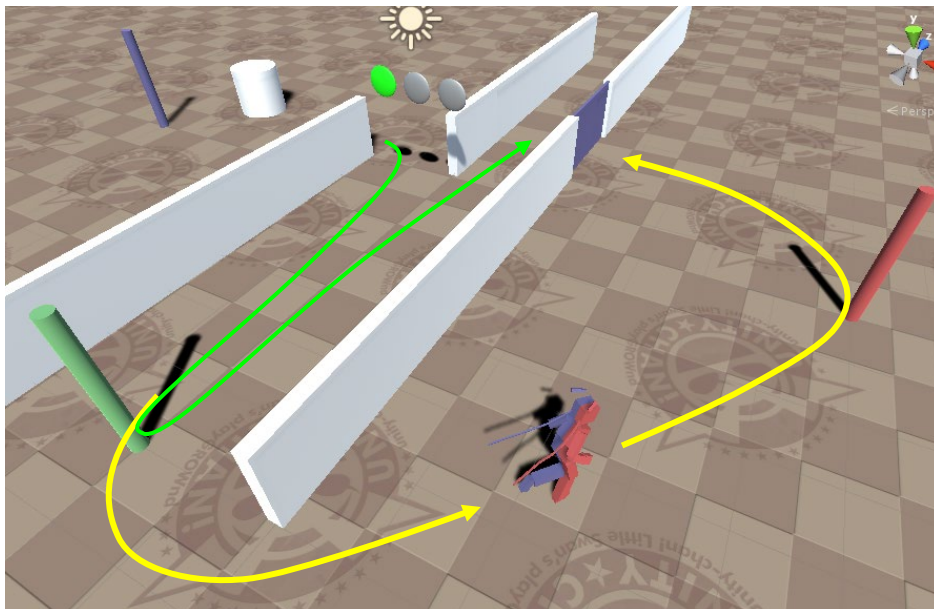


- プレイボタンを押下します。



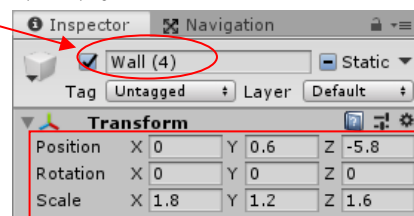
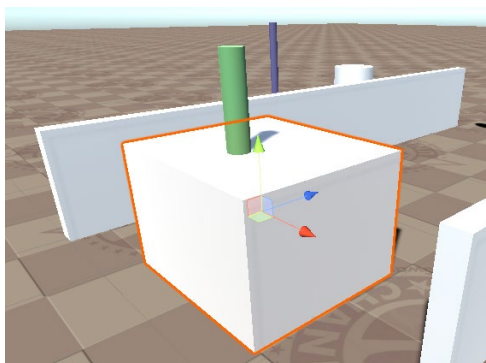
赤→青→緑、再び赤という
順回を開始しましたが、緑
の通過後に最短距離が外
側である判定をしています。

緑も赤も通過地点になって
しまって目的地らしくないので、
緑を高さのある行き止
まりに設置します。

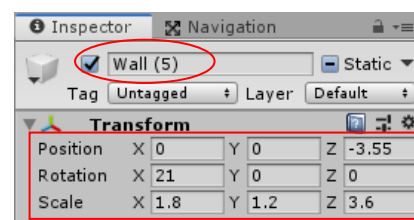
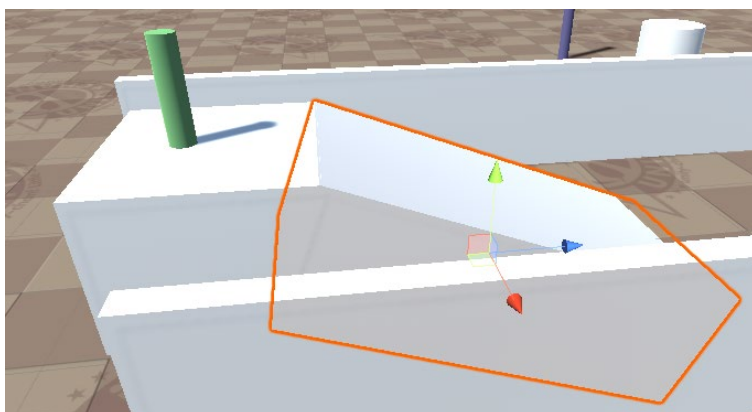


【STEP16】 高台の目的地

- ヒエラルキー欄の Wall(1)を複製(Ctrl + D)します。名称は Wall (4)のはずです。
インスペクタでパラメータを設定します。

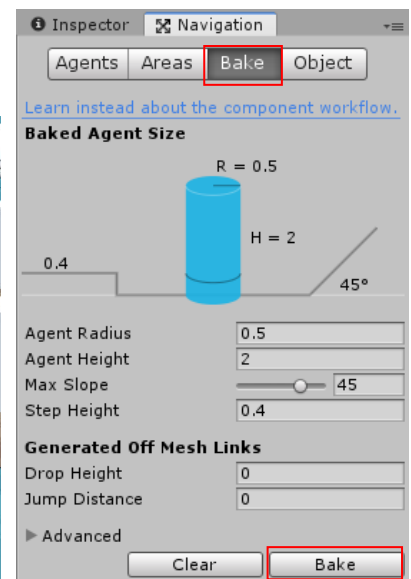
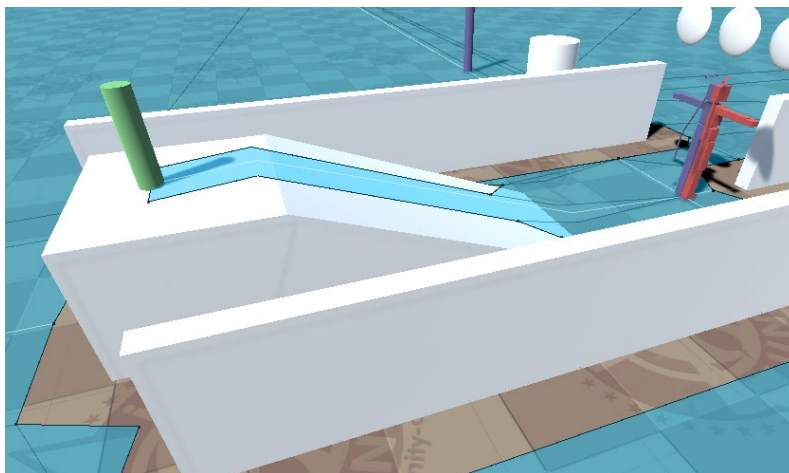


- この Wall(4)を複製(Ctrl + D)します。名称は Wall (5)のはずです。
インスペクタでパラメータを設定します。



- メニューWindow(ウィンドウ)から AI > **Navigation(ナビゲーション)**を選択します。

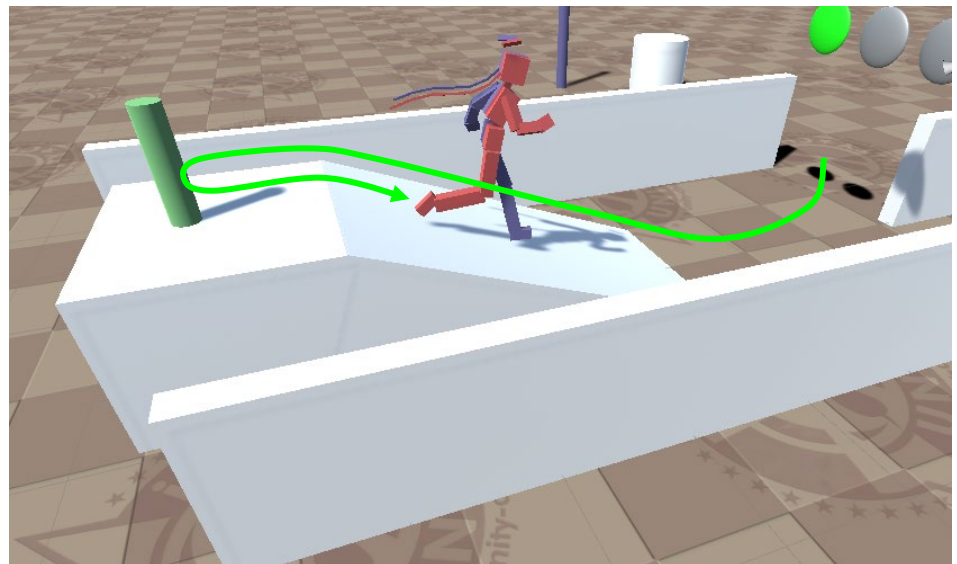
- タブ Bake(ベイク)を選択し、ボタン **Bake(ベイク)**を押下します。



- プレイボタンを押下します。



高台でターンすることで、目的地を経由している雰囲気になりました。

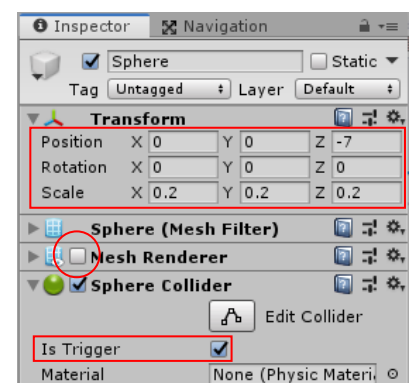
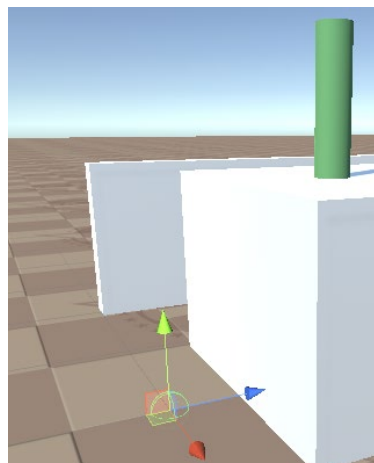


【STEP17】 高低差のある最短経路

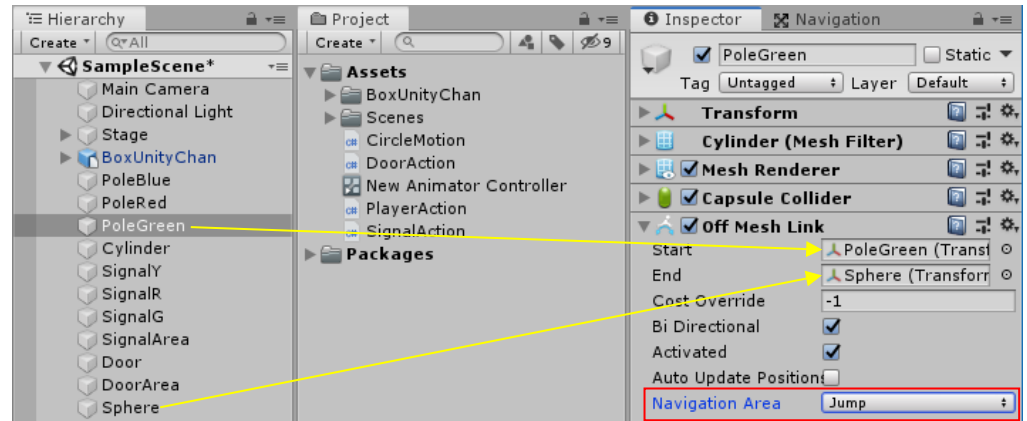
目的地が高さを伴った状態で、もう一度、GreenPole(緑)を先の経由(通過)地点と考えます。

最短経路は外側ですので、この高低差を飛び降りなければなりません。経路やアクションの設定次第では、高低差を飛び上がったり、垂直の非常階段を上ったりが可能になります。ここでは、飛び降りるアクションを追加します。

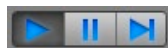
- ヒエラルキー欄の Create から 3D Object > **Sphere** を選択します。
インスペクタでパラメータを設定します。



- ヒエラルキー欄の **PoleGreen** を選択し、インスペクタの Add Component から Navigation > **Off Mesh Link** を選択します。Start と End の位置情報を持つオブジェクトをドラッグ & ドロップします。

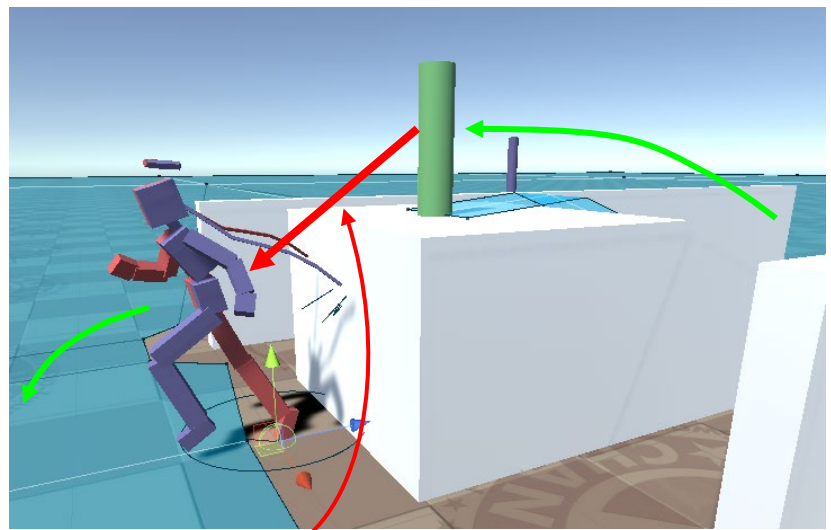


- プレイボタンを押下します。



ナビメッシュが途切れていますが、オフメッシュリンクが設定されて最短経路と判定されれば、その区間を移動することが実現できます。

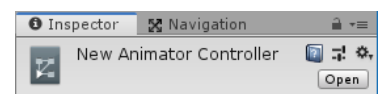
高低差でなく、水路など単に幅のある区間だけでも移動できます。



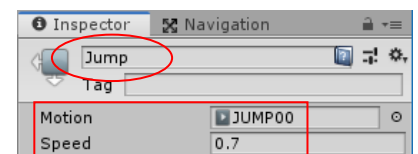
ところが、このオフメッシュリンク移動の時だけ、**異様に速く動いて**しまいます。見映えの問題として、速度だけでなく、キャラクターのモーションも同調することが望まれます。

【STEP18】 ジャンプモーション

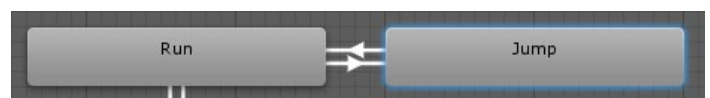
- プロジェクト欄の **New Animator Controller** を選択し、Open ボタンを押下してアニメーターを編集状態にします。



- 背景のグリッドを右クリックし、Create State > **Empty** を選択し、新しい状態を作成します。インスペクタでパラメータを設定します。



- 状態 Run と Jump を行き来するトランジションを作成します。

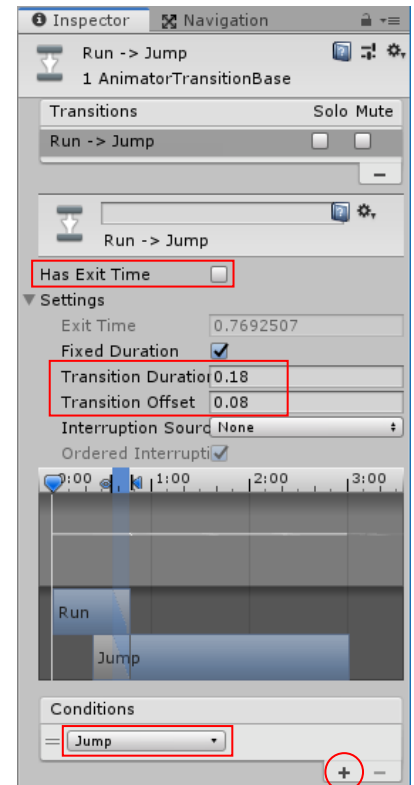


- 白い矢印 **Run -> Jump** をクリックします。

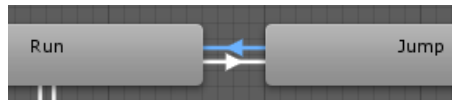


チェックボックス **HasExitTime** を**オフ**にします。
Transition 時間を設定します。

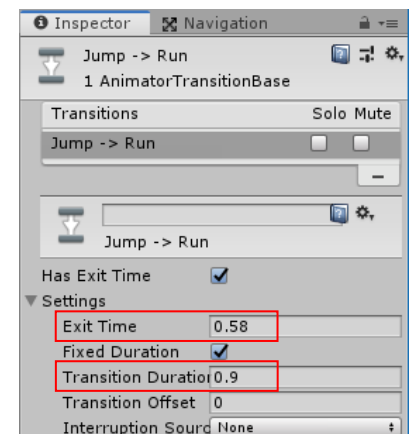
Conditions(コンディションズ:条件)のプラスボタン(+)を押下します。
Jump が宣せられたら、を設定します。



- 白い矢印 **Jump -> Run** をクリックします



Transition 時間を設定します。



- スクリプト **PlayerAction** を次のように編集します。

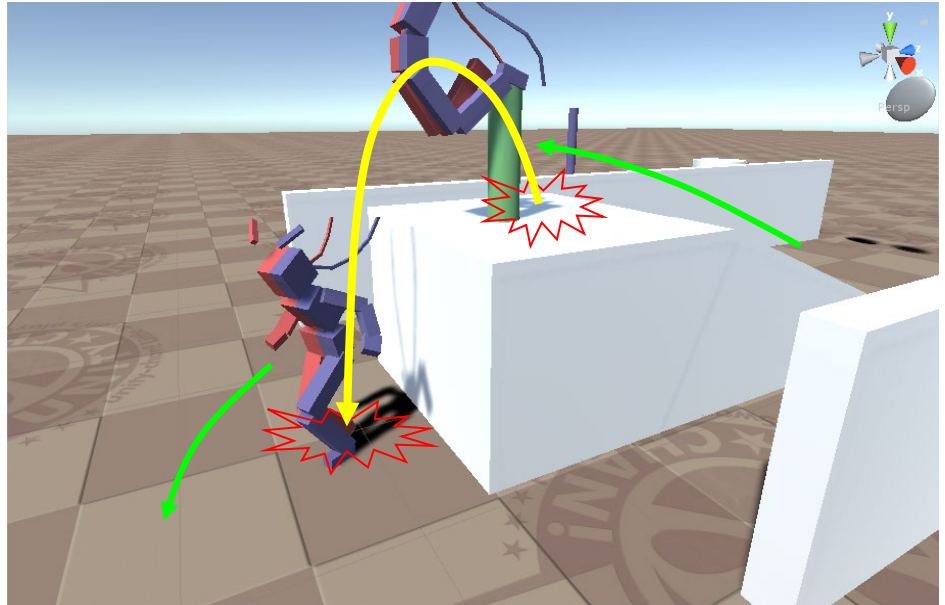
```
//～前略～

void OnTriggerEnter(Collider other) {
    if (other.gameObject.name.Substring( 0, 4 ) == "Pole") {
        TargetID++;
        TargetID %= 3;
        if (other.gameObject.name == "PoleGreen") {
            myAnim.SetTrigger( "Jump" );
            myAgent.speed = 0.7f;
            Invoke( "SpeedUp", 1.6f );
        }
    }
}

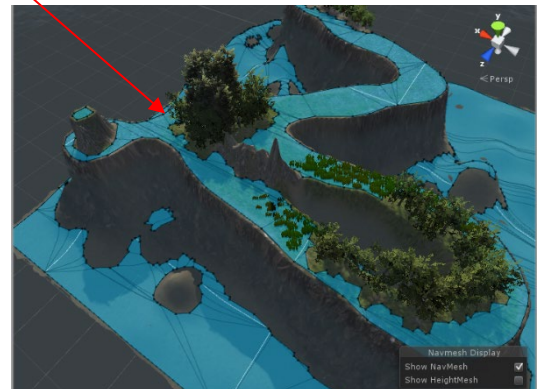
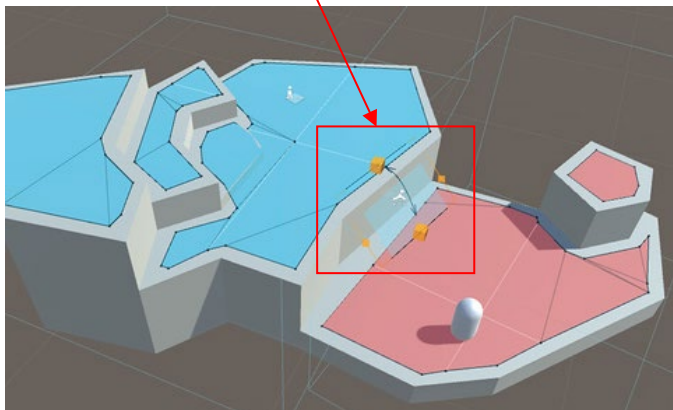
void SpeedUp() {
    myAgent.speed = 3.5f;
}

//～後略～
```


- プレイボタンを押下します。



- ナビメッシュは様々な場面で使われており、水平を基本とした**自然地形**も定義できます。(垂直面は無理です。)今回は複雑になり過ぎるので割愛しましたが、新たにステージが登場するなどして、次のステージが持つ別のナビメッシュとリンクを行う **NavMesh Link(ナビメッシュリンク)** といった機能も実現可能です。



- タワーディフェンスというゲームのジャンルがあります。

Unityの公式サイトから、タワーディフェンスは
このようにして作る！といったチュートリアルが
公開されていて、この中でも敵が攻めて来るコ
ースがナビメッシュで定義され活用されていま
す。

興味がある人は挑戦してみてください。



<https://unity3d.com/jp/learn/tutorials/projects/tower-defense-template/setting-stage-tower-defense-template>