1. 简单场景的自动寻路。
   1. 搭建简单的场景，包括地面和高低不平的box。并设置一个目标点物体。
   2. 设置Camera，使得能够俯瞰全局。
   3. 设定一个Capsule作为待控制的角色，为其添加NavAgent组件，使其能够作为自动寻路的角色。
   4. 添加一个脚本，能够让这个角色获得目标点，进而使用Unity自带的寻路方法找到目标点。

public Transform goal;

private UnityEngine.AI.NavMeshAgent agent;

void Start()

{

agent = GetComponent<UnityEngine.AI.NavMeshAgent>();

agent.destination = goal.position;

}

* 1. 通过Bake路径，在给定一个目标点的情况下，角色可以自由移动。
  2. 将鼠标点击位置作为目标点，这个时候，需要在脚本中添加以下代码：

void Update () {

if (Input.GetMouseButtonDown(0))

{

RaycastHit hit;

if (Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hit, 100))

{

agent.destination = hit.point;

}

}

}

* 1. 这时候测试，当点击到高台上面的时候，发现无法通过。需要介绍OffMeshLink将其相连，才可以通过。
  2. 介绍NavMesh Obstacle，当不想让agent触碰某些特殊物品的时候，可以添加这个组件。比如rigidbody物体，当不设置这个组件的时候，物体可能被agent推开。默认情况下，不设置obstacle中的carve选项，这样agent将尽量避开这个obstacle，适用于运动的obstacle。当这个obstacle是静止的物体时，可以设置carve选项，寻路系统将bake出来一条绕路，容易绕过这个物体。

1. 实现一个简单的敌人智能
   1. 场景另存为。扩展地形，使得其更适合对战游戏。上次课使用的agent进行复制，其中一个作为敌人，一个作为玩家。
   2. 使用不同的材质来区别两个实体，并设置box子物体作为实体的朝向。我们实现的思路是：敌人沿着某个特定的路径巡逻；如果发现玩家，就冲刺，并追踪玩家，直到近到能够攻击玩家；如果玩家出了视野，则停止攻击，回到路径点，继续巡逻。
   3. 实现敌人寻路。将玩家实体的tag设置为Player，敌人的设置为Enemy。编写Enemy控制脚本，使其沿着特定的路径寻路。让enemy沿着给定关键点的路径巡逻，当到达关键点足够近的时候，停留一小会儿，然后转向下一个关键点。

void Awake()

{

nav = GetComponent<UnityEngine.AI.NavMeshAgent>();

}

void Update()

{

Patrolling();

}

void Patrolling()

{

nav.isStopped = false;

// Set an appropriate speed for the NavMeshAgent.

nav.speed = patrolSpeed;

// If near the next waypoint or there is no destination...

if (nav.remainingDistance < nav.stoppingDistance)

{

// ... increment the timer.

patrolTimer += Time.deltaTime;

// If the timer exceeds the wait time...

if (patrolTimer >= patrolWaitTime)

{

// ... increment the wayPointIndex.

if (wayPointIndex == patrolWayPoints.childCount - 1)

wayPointIndex = 0;

else

wayPointIndex++;

// Reset the timer.

patrolTimer = 0;

}

}

else

// If not near a destination, reset the timer.

patrolTimer = 0;

// Set the destination to the patrolWayPoint.

nav.destination = patrolWayPoints.GetChild(wayPointIndex).position;

}

* 1. 在场景中添加多个empty object用来组成多个关键点，并将其作为另外一个empty object的子物体，重命名。将这个父物体作为敌人的寻路关键点物体。
  2. 实现敌人的“视野功能”。即当玩家到达敌人的视野范围内时，认为发现玩家，并记录玩家的位置，便于敌人作为反应。我们使用碰撞体来实现。在敌人物体下面挂接一个子物体，子物体是空的，只有一个collider用来表示敌人的警戒范围。勾选trigger属性，当有物体进入这个trigger的时候，出发响应函数。为这个视野物体编写脚本。需要处理的情况，只有当玩家在敌人前方一定角度范围内才算是看到，而且需要保证中间没有障碍物。

public float fieldOfViewAngle = 110f;

public bool playerInSight; // Whether or not the player is currently sighted.

public Vector3 personalLastSighting; // Last place this enemy spotted the player.

public static Vector3 resetPos = Vector3.back;

private BoxCollider col; // Reference to the sphere collider trigger component.

private GameObject player; // Reference to the player.

void Awake()

{

col = GetComponent<BoxCollider>();

player = GameObject.FindGameObjectWithTag("Player"/\*Tags.player\*/);

// Set the personal sighting and the previous sighting to the reset position.

personalLastSighting = resetPos;

}

void OnTriggerStay(Collider other)

{

// If the player has entered the trigger sphere...

if (other.gameObject == player)

{

// By default the player is not in sight.

playerInSight = false;

// Create a vector from the enemy to the player and store the angle between it and forward.

Vector3 direction = other.transform.position - transform.position;

float angle = Vector3.Angle(direction, transform.forward);

// If the angle between forward and where the player is, is less than half the angle of view...

if (angle < fieldOfViewAngle \* 0.5f)

{

RaycastHit hit;

// ... and if a raycast towards the player hits something...

if (Physics.Raycast(transform.position + transform.up, direction.normalized, out hit, col.size.z))

{

// ... and if the raycast hits the player...

if (hit.collider.gameObject == player)

{

// ... the player is in sight.

playerInSight = true;

// Set the last global sighting is the players current position.

//lastPlayerSighting.position = player.transform.position;

personalLastSighting = player.transform.position;

}

}

}

}

}

void OnTriggerExit(Collider other)

{

// If the player leaves the trigger zone...

if (other.gameObject == player)

// ... the player is not in sight.

playerInSight = false;

}

* 1. 使用视野来驱动敌人行为。运行游戏，添加debug输出的方法，测试当前方法是否有效。将视野脚本放置到敌人行为当中，修改敌人脚本如下：

public float patrolSpeed = 2f; // The nav mesh agent's speed when patrolling.

public float chaseSpeed = 5f; // The nav mesh agent's speed when chasing.

public float chaseWaitTime = 5f; // The amount of time to wait when the last sighting is reached.

public float patrolWaitTime = 1f; // The amount of time to wait when the patrol way point is reached.

public Transform patrolWayPoints; // An array of transforms for the patrol route.

public float shootRotSpeed = 5f; // 瞄准时候旋转朝向的速度

public float sqrPlayerDist = 4f;

private EnemySight enemySight; // Reference to the EnemySight script.

private UnityEngine.AI.NavMeshAgent nav; // Reference to the nav mesh agent.

private float chaseTimer; // A timer for the chaseWaitTime.

private float patrolTimer; // A timer for the patrolWaitTime.

private int wayPointIndex; // A counter for the way point array.

private bool chase = false; // 当遇到攻击或者在射击的时候玩家跑开的话

private Transform player; // Reference to the player's transform.

void Awake()

{

// Setting up the references.

enemySight = transform.Find("EnemySight").GetComponent<EnemySight>();

Debug.Assert(enemySight);

nav = GetComponent<UnityEngine.AI.NavMeshAgent>();

player = GameObject.FindGameObjectWithTag("Player").transform;

}

void Update()

{

//Debug.Log("Update!");

// If the player is in sight and is alive...

if (enemySight.playerInSight)

// ... shoot.

{

Shooting();

chase = true;

}

// If the player has been sighted and isn't dead...

else if (chase)

// ... chase.

Chasing();

// Otherwise...

else

// ... patrol.

{

Patrolling();

//Debug.Log("Patrol!");

}

}

void Shooting()

{

Vector3 lookPos = player.position;

lookPos.y = transform.position.y;

Vector3 targetDir = lookPos - transform.position;

float step = shootRotSpeed \* Time.deltaTime;

Vector3 newDir = Vector3.RotateTowards(transform.forward, targetDir, step, 0.0F);

transform.rotation = Quaternion.LookRotation(newDir);

// Stop the enemy where it is.

nav.isStopped = true;

Debug.Log("Shoot player!");

}

void Chasing()

{

//Debug.Log("Chasing");

nav.isStopped = false;

// Create a vector from the enemy to the last sighting of the player.

Vector3 sightingDeltaPos = enemySight.personalLastSighting - transform.position;

// If the the last personal sighting of the player is not close...

if (sightingDeltaPos.sqrMagnitude > sqrPlayerDist)

// ... set the destination for the NavMeshAgent to the last personal sighting of the player.

nav.destination = enemySight.personalLastSighting;

// Set the appropriate speed for the NavMeshAgent.

nav.speed = chaseSpeed;

// If near the last personal sighting...

if (nav.remainingDistance < nav.stoppingDistance)

{

// ... increment the timer.

chaseTimer += Time.deltaTime;

// If the timer exceeds the wait time...

if (chaseTimer >= chaseWaitTime)

{

// ... reset last global sighting, the last personal sighting and the timer.

//lastPlayerSighting.position = lastPlayerSighting.resetPosition;

chase = false;

chaseTimer = 0f;

}

}

else

// If not near the last sighting personal sighting of the player, reset the timer.

chaseTimer = 0f;

}