CT3536 Games Programming Unity3D

Raycasting

Raycasting

- Raycasting is an important and useful concept in games
- 'Cast a ray' (theoretically) from a source position in a specified direction, and see what it hits (in the Physics world)
- In Unity, raycasts are implemented as static methods of the Physics class
- Raycasts can be performed against Colliders and/or Triggers
- As well as rays, Unity lets you cast larger volumes such as spheres or boxes in a direction
- You can also filter by "Layer"
- There are also other, related methods provided by the Physics class for finding Colliders that coincide with volumes of space (without using an actual raycast)

Physics.Raycast

A simple Raycast that returns true/false identifying whether there is a collision:

Only the 1st 2 params are mandatory

public static bool Raycast(Vector3 origin, Vector3 direction,
float maxDistance = Mathf.Infinity, int layerMask = DefaultRaycastLayers,
QueryTriggerInteraction queryTriggerInteraction = QueryTriggerInteraction.UseGlobal);

Parameters

origin	The starting point of the ray in world coordinates.
direction	The direction of the ray (world coords).
maxDistance	The max distance the ray should check for collisions.
layerMask	A *Layer mask* that is used to selectively ignore Colliders when casting a ray. (See: https://docs.unity3d.com/Manual/Layers.html)
queryTriggerInteraction	Specifies whether this query should hit Triggers.

(Raycasts will not detect Colliders for which the Raycast origin is inside the Collider.)

Physics.Raycast

Or if you need detailed information on the raycast-hit:

```
public static bool Raycast(Vector3 origin, Vector3 direction, out RaycastHit hitInfo
    float maxDistance = Mathf.Infinity, int layerMask = DefaultRaycastLayers,
    QueryTriggerInteraction queryTriggerInteraction = QueryTriggerInteraction.UseGlobal );
```

If true is returned, **hitInfo** will contain more information about the (first) collider that was hit:

.collider - the Collider that was hit
.distance - distance from ray's origin to impact point
.normal - the surface normal (Vector3) of the hit surface
.point - the impact point (Vector3) in world space
.rigidbody - the rigidbody of the collider that was hit
(could be null)

In C#, "out" is a way of passing by reference

Physics.RaycastAll

Cast a ray through the scene and return *all* hits (as an array of RaycastHit objects). Note that order is not guaranteed.

public static RaycastHit[] RaycastAll(Vector3 origin, Vector3 direction, float maxDistance = Mathf.Infinity, int layermask = DefaultRaycastLayers, QueryTriggerInteraction queryTriggerInteraction = QueryTriggerInteraction.UseGlobal);

Unity also allows you to "cast shapes" (that are wider than rays):

- Physics.BoxCast
- Physics.CapsuleCast
- Physics.SphereCast

Physics static methods that are related to Raycasting

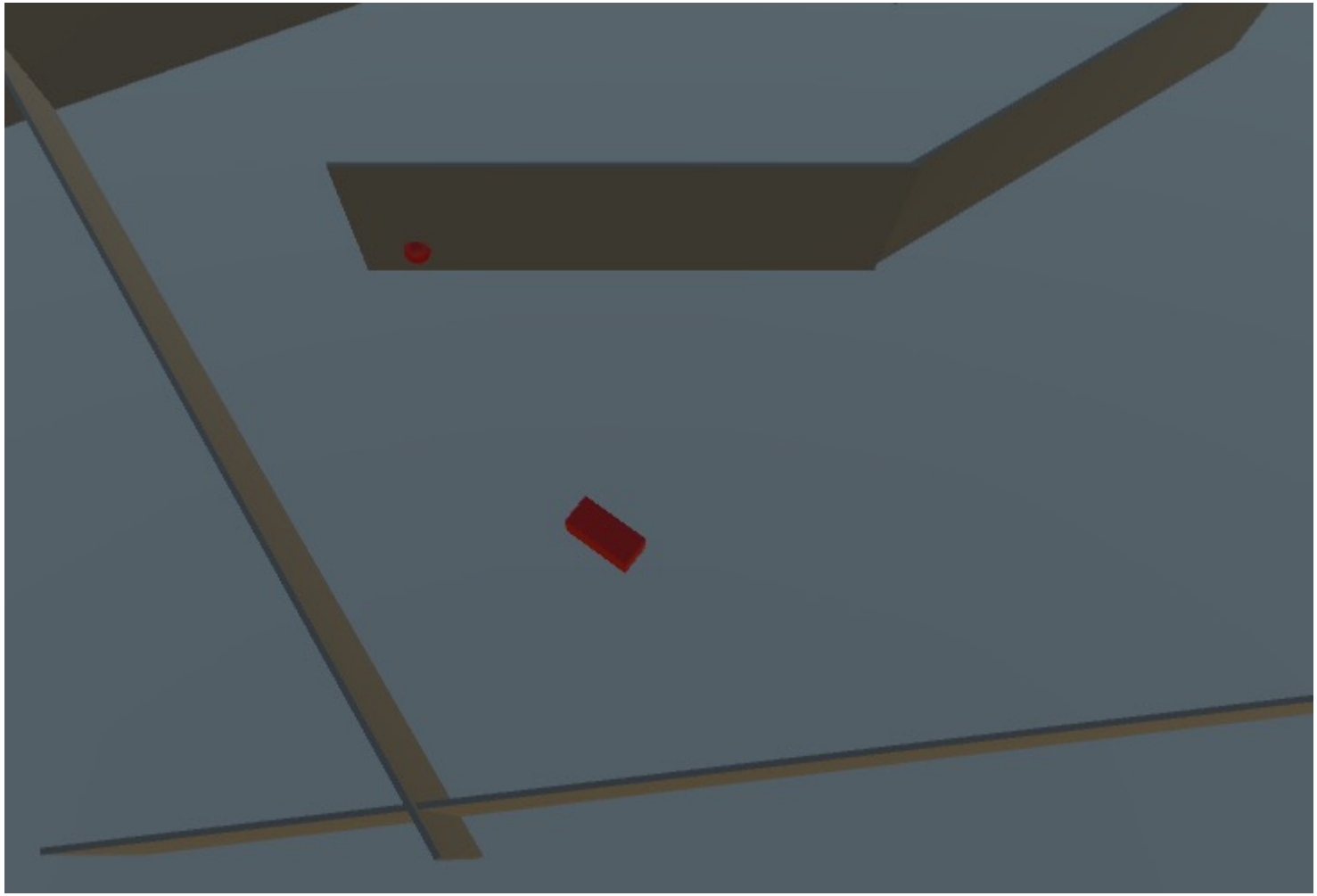
Unity offers various methods for collecting an array of Colliders that intersect with specifically-shaped volumes of space in the world:

Physics.OverlapBox
Physics.OverlapCapsule
Physics.OverlapSphere

E.g. we'll use this in a later example (in a 2D context) to check that an area of the world is empty before spawning a zombie into it

Steering-with-Raycasting demo

(this project has been made available on Canvas)



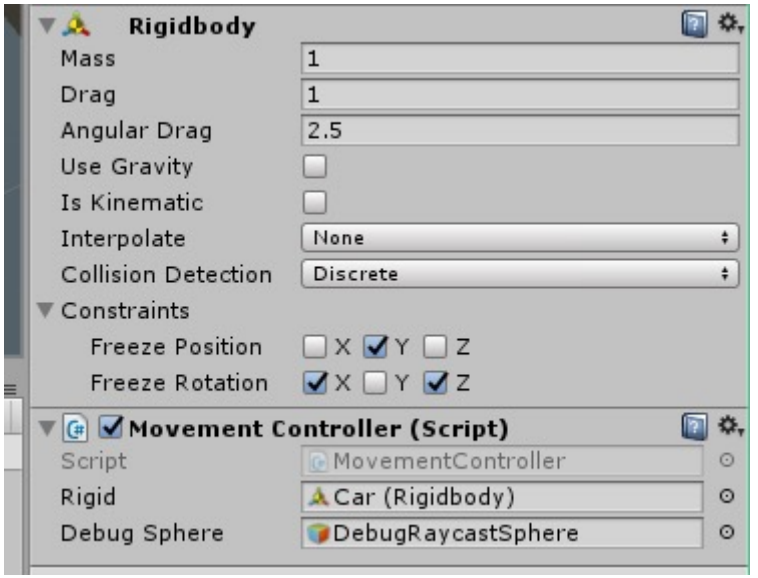
```
public class MovementController : MonoBehaviour {  // (attached to the car)
    public Rigidbody rigid;
    public GameObject debugSphere;
    //

    private Quaternion twentyDegreesYAxisRotation = Quaternion.AngleAxis(20f, Vector3.up);
    private Quaternion minusTwentyDegreesYAxisRotation = Quaternion.AngleAxis(-20f, Vector3.up);
    private RaycastHit hitInfo = new RaycastHit();


    void FixedUpdate () {
        // raycast forward-left and forward-right to determine steering (with angular force)
        Vector3 leftDirection = twentyDegreesYAxisRotation * transform.forward;
        Vector3 leftFromPos = transform.position + 0.1f * leftDirection;
        float leftDist = GetRaycastDistance (leftFromPos, leftDirection);
        Vector3 rightDirection = minusTwentyDegreesYAxisRotation * transform.forward;
        Vector3 rightFromPos = transform.position + 0.1f * rightDirection;
        float rightDist = GetRaycastDistance (rightFromPos, rightDirection);
        if (leftDist < rightDist && leftDist <= 1.5f) {
            rigid.AddTorque (new Vector3 (0f, -0.03f / leftDist, 0f));
            debugSphere.SetActive (true);
            debugSphere.transform.position = leftFromPos + leftDirection * leftDist;
        }
        else if (rightDist <= 1.5f) {
            rigid.AddTorque (new Vector3 (0f, 0.03f / rightDist, 0f));
            debugSphere.SetActive (true);
            debugSphere.transform.position = rightFromPos + rightDirection * rightDist;
        }
        else
            debugSphere.SetActive (false);
        // Add linear force forwards
        if (rigid.velocity.magnitude<0.5f)
            rigid.AddForce (transform.forward);
    }
```

(completed on next slide)

(MovementController script, continued..)

```
    private float GetRaycastDistance(Vector3 fromPos, Vector3 direction) {
        Physics.Raycast (fromPos, direction, out hitInfo); // out => pass-by-ref
        return hitInfo.distance;
    }
}
```

(RigidBody settings of "car")



(FollowCam script, attached to the camera)

```
public class FollowCam : MonoBehaviour {

    public Transform followTransform;
    public Vector3 followOffset;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        transform.position = followTransform.position + followOffset;
        transform.LookAt (followTransform);
    }
}
```

DemonPit use of Raycasts

- *We use raycasts in numerous places, e.g.:*
- Raycast guns
  - Do a raycast from the centre of the screen (where there's a crosshair) to see if the player hit something (simulates an infinitely-fast moving bullet)
- Teleport/Lasso
  - Raycast every frame from centre of screen to see whether a 'teleport node' is selected – turn a light on these on/off to give visual feedback
- Monsters chasing the player: don't fall into a hole or run into walls – see next slide
- Monsters: can they see the player at all? – see 2 slides ahead

DemonPit Walkers: don't fall into a hole or run into walls

In Monster::FixedUpdate()

```
if (rigid.useGravity) {
   // walkers have to move in the direction they're facing, so make sure it's almost the right way
   Vector3 targetDir = (targetPos-myCurrentPos).normalized;
   if (Vector3.Dot(targetDir,transform.forward)>0.85f) {

       // don't run into walls
       if (!Physics.Raycast(myCurrentPos, targetDir, 1.5f, GameManager.wallsMask)) {
            // don't run over an edge
            Vector3 posPlus3Metres = myCurrentPos + (3f*targetDir);
            if (Physics.Raycast(posPlus3Metres, Vector3.down, 2f, GameManager.floorsMask))
               // here's where we actually give the monster some velocity
               // (note that the direction of force is straight at target, not our local forward)
               rigid.AddForce(targetDir * (acceleration * Time.fixedDeltaTime * 50f * rigid.mass));

            else
               Debug.Log("Avoiding lava!");
        }

   }

}
```

DemonPit: Monster::CanSeePlayer() and CanSeePlayerFrom()

```
public bool CanSeePlayer() {
    Vector3 fromPos = transform.position;

    for (float x=-0.2f; x<=0.2f; x+=0.2f) {
       for (float z=-0.2f; z<=0.2f; z+=0.2f) {
          fromPos.x = myCurrentPos.x + x;
          fromPos.z = myCurrentPos.z + z;
          if (Physics.Raycast(fromPos, dirToPlayer, distFromPlayer-0.4f,
seePlayerTestMask))
               return false;
       }
    }

    return true;
}
```

Question:
How do we calculate dirToPlayer and distFromPlayer?

```
public static LayerMask seePlayerTestMask;
seePlayerTestMask = LayerMask.GetMask("ArenaFloor","ArenaInnerWalls");
```

```
public bool CanSeePlayerFrom(Vector3 fromPos) {
    Physics.queriesHitBackfaces = true;
    // go back a bit, just incase our gun is embedded in the wall etc.
    fromPos -= dirToPlayer.normalized;

    if (Physics.Raycast(fromPos, dirToPlayer, distFromPlayer+1f-0.4f, seePlayerTestMask))
        return false;

    return true;
}
```

Another Raycasting example

- In "Let's Break Stuff!" (2012, Shiva3D) during the level builder mode, the user selects objects (from a UI) and drags in the 3D world to position them
- This involves projecting a line from the camera in the direction they are pointing with the mouse, and casting rays downwards at intervals along it
- The *highest (y axis) result is accepted*

Yet Another Raycasting example

- In "The Necromancer's Tale" we raycast every frame from the camera to the player character.
- Objects blocking this ray have their material settings changed in order to animate their opacity down to about 20%, and back up to 100% when they stop blocking the player's view of the character