Università Degli Studi Di Milano

Artificial Intelligence For Video Games

Drunk Stride

Author:
Giovanni Cocco

Academic year:
2023-2024

Wednesday 3rd January, 2024

Contents

List of Figures

Listings

Abstract

The project goal was to create an agent roaming on a platform while never moving along a straight line. More details were provided by the professors as follows:

* The platform can be of any size; square or rectangular.

* The agent will change trajectory at random intervals. Each time the agent will travel over a circumference leading first to right then to the left, then to the right again ... and so on.

* The agent is moving at a constant speed of 1 meter per second.

* At each interval, the agent will pick a random value for the time to the next trajectory change in the range (0, 10] seconds. Note that 0 is excluded.

* Then, the agent selects a random circumference leading right or left with a radius between 0 (excluded) and the maximum radius that is not making the agent fall off the platform.

* The selection of the radius is independent of the time of the next trajectory change.

* The agent never stops moving.

* The system must work independently on the platform shape or size.

1   Scene setup

The project main scene is **Assets/Scenes/Main.unity**.

1.1   Agent

As the agent movement is quite simple and does not require any complex physical simulation **kinematic movement** was chosen. This means that the agent does not have any rigid body component and its movement is not managed in any way by the Unity physics engine. It should be noted that it is not affected by gravity, but given the constraint that the agent should never fall off the platform it makes no difference in the resulting behavior.

The agent still has a **capsule collider** that is used to determine its own radius to avoid having half of the agent off the platform.

1.2   Platform

The platform is a simple plane with a **mesh collider**, as the default plane uses this. The agent code will hold a reference to the platform mesh collider.

The platform size is quite small to better test the interaction at the edges and to allow to see the whole platform in the camera. Although bigger platform size lead to bigger radius giving more believable results.

2   Agent script

2.1   Public attributes

The Agent class has few public attributes to allow some configuration from the Unity inspector.

```
[Range(0.5f, 20.0f)]
public float maxDecisionInterval = 10.0f;
[Range(0.1f, 10.0f)]
public float speed = 1.0f;
[Range(10.0f, 360*4.0f)]
public float maxRotationSpeed = 360.0f;
public bool drawDebugGizmos = false;
public MeshCollider platform;
```

Listing 1: Public attributes

The **maxDecisionInterval** and **speed** are set to a default value as required by the specifications.

The value of **maxRotationSpeed** controls the maximum rotation speed, this is used to avoid unrealistic fast spinning when a minuscule radius is randomly selected.

The flag **drawDebugGizmos** enables the rendering of the maximum radius that is not making the agent fall off the platform and the current selected circular trajectory.

A reference to the platform **MeshCollider** allows querying for its size and rotation along the Y axis. It would be possible to use a **Collider** reference instead making it working also with a **BoxCollider**, this choice however would make the support of rotations on the Y axis troublesome.

From a **Collider** it is only possible to query the **world aligned bounds**

making it impossible to query the actual height and width of the platform. As the platform is a plane and, by defaults, Unity uses a **MeshCollider** for planes it was preferred to maintain the support for rotations rather than supporting **BoxCollider** too.

2.2   Private attributes

The Agent class has 3 private attributes.

```
private float _direction = 1.0f;
private Vector3 _circleCenter;
private float _circleRadius;
```

Listing 2: Private attributes

The attributes **_circleCenter** and **_circleRadius** keep track of the current selected trajectory to follow while **_direction** alternates the values 1.0f and -1.0f to keep track if the agent is moving right or left.

2.3   Decision procedure

To choose the next circular trajectory at random intervals a **coroutine** was used.

The coroutine **ChooseNextCircle()** is an infinite loop that waits for a random amount of time, between **float.Epsilon** and **maxDecisionInterval**, at every iteration.

The coroutine is in charge of updating the values of **_circleCenter** and **_circleRadius**. The radius is random between the maximum radius minus the agent size and **float.Epsilon**.

The circle center is simply calculated displacing the current agent position left or right by the radius.

At every iteration **_direction** is inverted.

```
void Start() {
    StartCoroutine(ChooseNextCircle());
}

private IEnumerator ChooseNextCircle() {
    while (true) {
        // Subtract the Agent radius to avoid having half
            the agent off the platform
```

```
_circleRadius = Random.Range(float.Epsilon,
    MaxRadius(_direction)-GetComponent<Collider>().
    bounds.extents.x*0.5f);
_circleCenter = transform.position + _direction*
    transform.right * _circleRadius;

yield return new WaitForSeconds(Random.Range(float.
    Epsilon, maxDecisionInterval));
_direction *= -1;
        }
    }
```

<div align="center">

Listing 3: Decision procedure

</div>

2.4   Motion

The agent motion is implemented in the **FixedUpdate** method simply by rotating the agent around the circle center.

The angle of the rotation (per second) is calculated based on the agent speed and the circle perimeter, capped to the **maxRotationSpeed**.

```
void FixedUpdate() {
    // Cap the maximun rotation speed to avoid ultra fast
        spinning on small circles
    float degreePerSecond = Mathf.Min(360.0f * speed/(2.0f*
        _circleRadius*Mathf.PI), maxRotationSpeed);
    Quaternion rotation = Quaternion.AngleAxis(_direction *
        degreePerSecond * Time.fixedDeltaTime, Vector3.up);

    transform.position = rotation*(transform.position -
        _circleCenter) + _circleCenter;
    transform.rotation *= rotation;
}
```

<div align="center">

Listing 4: FixedUpdate

</div>

2.5   Calculating the maximum safe radius

The calculation of the maximum safe radius is divided in 4 steps:

1. Rotate the coordinate system to have the platform axis aligned.

2. Calculate the platform extents.

3. Calculate the maximum safe radius for every edge of the platform.

4. Return the minimum of the 4 safe radius calculated.

```
private float MaxRadius(float direction) {
    // Apply the inverse of the platform rotation to get
        back to an axis aligned coordinate system
    Quaternion invRotation = Quaternion.Inverse(platform.
        transform.rotation);
    Vector3 position = invRotation*transform.position;
    Vector3 right = invRotation*transform.right;

    // Get the extends of the platform from the mesh as the
        collider bounds are AA in world space
    Vector3 extents = Vector3.Scale(platform.sharedMesh.
        bounds.extents, platform.transform.lossyScale);
    Vector3 min = invRotation*platform.transform.position -
        extents;
    Vector3 max = invRotation*platform.transform.position +
        extents;

    // Get the max radius for each edge, then return the
        minimun one
    float x1 = (position.x-min.x)/(1.0f-right.x*direction);
    float x2 = (max.x-position.x)/(1.0f+right.x*direction);
    float z1 = (position.z-min.z)/(1.0f-right.z*direction);
    float z2 = (max.z-position.z)/(1.0f+right.z*direction);

    return Mathf.Min(Mathf.Min(x1, x2), Mathf.Min(z1, z2));
}
```

Listing 5: MaxRadius

To get the platform extends we can query the **sharedMesh** bounds of the **MeshCollider** and scale them based on the platform **lossyScale**. Note that unless the platform is no longer a rectangle the **lossyScale** is exact.

All the code in the project query informations when needed avoiding caching them to better allow for real time changes, while this can be changed to improve performances, for the scope of this project, the ability to see changes in real time was preferred.

To get the maximum radius for a single axis aligned edge we should consider the following situation:
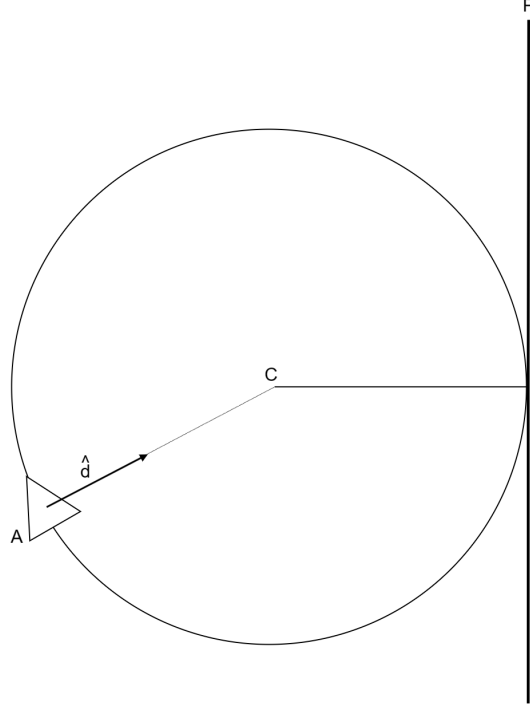
Figure 1: Geometrical situation

The center of the circle C can be determinated from the agent position A, the radius r and the right direction $\hat{d}$:

$$C = A + r\hat{d}, \text{ with } \|\hat{d}\| = 1$$

The radius r is also the distance of the circle center from the platform border P, as we are axis aligned:

$$r = P_x - C_x$$

Putting all together:

$$r = P_x - (A_x + r\hat{d}_x)$$

Solving for r we get:

$$r = \frac{P_x - A_x}{1 + \hat{d}_x}$$

Note that in the project the agent center never reaches the edge of the platform, so the numerator can never be zero, and even if the denominator is

zero we get, by floating point IEEE 754 arithmetic, an infinite value that will not get past the pick of the minimum of the 4 edges.

Mutatis mutandis for the other 3 edges.

2.6   Gizmos

To better debug the project Gizmos showing the maximum radius and the current trajectory were added with the following code:

```
void OnDrawGizmos() {
    if (drawDebugGizmos) {
        Vector3 offset = (platform.transform.position.y-
            transform.position.y+0.01f)*Vector3.up;
        Gizmos.color = Color.blue;
        DrawCircle(_circleCenter+offset, _circleRadius);
        Gizmos.color = Color.red;
        DrawCircle(transform.position+transform.right*
            MaxRadius(1.0f)+offset, MaxRadius(1.0f));
        DrawCircle(transform.position-transform.right*
            MaxRadius(-1.0f)+offset, MaxRadius(-1.0f));
    }
}
```

Listing 6: Gizmos drawing

The method **OnDrawGizmos()** get called even when the agent is not selected, this allows to see the maximum radius when editing the platform size, orientation and position from the editor.
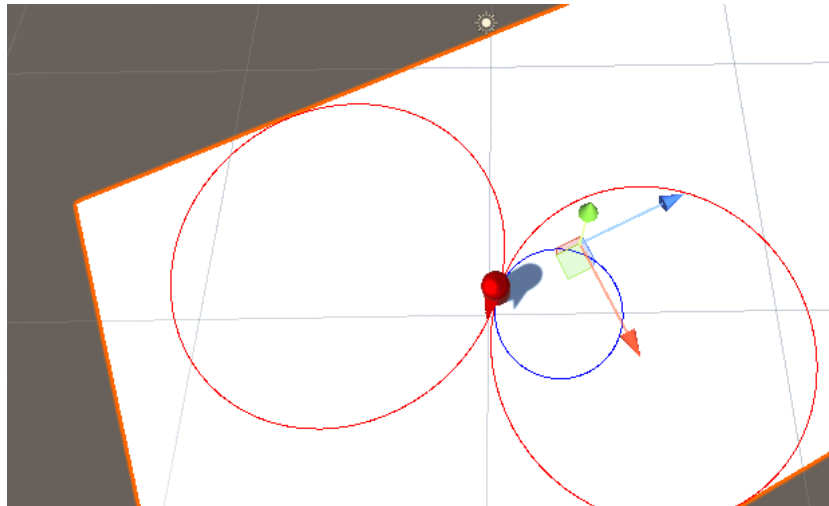
Figure 2: Gizmos as seen in the editor