Inteligência Artificial
para Jogos - 2025/1

Trabalho Grau A

Ricardo Moreira Gomes

Estrutura geral do código

- Novo bot herdado de AIBehaviour;
- Override sobre Init() e Execute();
- Combinação de prioridades e behaviors;
- Comportamentos utilizados:
  - Seek() e SeekOrb();
  - ObstacleAvoidance();
  - Wander();
  - FlankEnemy().

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

[CreateAssetMenu(menuName = "AI/Behaviours/Ricardobot")]
0 references
public class Ricardobot : AIBehaviour
{
    7 references
    SnakeMovement movement;
    2 references
    float changeTargetInterval = 2.0f; //variável de intervalo pra resetar o randomPoint do
    4 references
    float timeToNextChange; //intervalo INTERNO pra resetar o randomPoint do Wander()

    0 references
    public override void Init(GameObject owner, SnakeMovement movement)
    {
        this.owner = owner; //referência pro dono do comportamento
        this.movement = movement; //componente de mov. da cobra
        this.randomPoint = owner.transform.position + (Vector3)(Random.insideUnitCircle * 5f
        timeToNextChange = changeTargetInterval; //timer para próxima mudança de direção
    }
```

```
0 references
public override void Execute()
{
    if (movement.isDead) return; //verifica se a cobra está morta

    Vector3 force = Vector3.zero; //inicializa força de movimento
    bool isAggressive = movement.bodyParts.Count >= 20; //define mudança de comportamento baseado em tamanho (20 partes)

    //vetores de comportamento
    Vector3? avoid = ObstacleAvoidance(); //evitar obstáculos
    Vector3? orb = SeekOrb();             //buscar orbs
    Vector3? flank = FlankEnemy();        //flanquear/cercar inimigos

    //sistema de combinação de prioridades/behaviors
```

Comportamentos e Prioridades

- Variáveis e vetores de controle (força de movimento, vetores de comportamento);
- Morto?
  - Sim → Return;
  - Não → Calcula vetores → Evitar obstáculo?
    - Sim → Combina com prioridade máx;
    - Não → Buscar orbs?
      - Sim → Movimento para orb;
      - Não → Wander padrão.

```
//sistema de combinação de prioridades/behaviors
if (avoid.HasValue && isAggressive && flank.HasValue)
{
    //combo 1 = Agressivo + Evitando Obstáculos
    force = (avoid.Value * 4.0f + flank.Value * 2.0f).normalized;
}
else if (avoid.HasValue && !isAggressive && orb.HasValue)
{
    //combo 2 = Passivo + Evitando Obstáculos + Buscando Orbs
    force = (avoid.Value * 4.0f + orb.Value * 1.0f).normalized;
}
else if (isAggressive && flank.HasValue)
{
    //combo 3 = Agressivo
    force = flank.Value.normalized;
}
else if (avoid.HasValue)
{
    //combo 4 = Evitando Obstáculos
    force = avoid.Value.normalized;
}
else if (!isAggressive && orb.HasValue)
{
    //combo 5 = Buscando Orbs
    force = orb.Value.normalized;
}
else
{
    //combo 6 (fallback) = Wander/Vagar
    Wander();
    force += Seek(randomPoint).normalized;
}
```

Behaviors 1/4 (Seek, Wander)

```
3 references
Vector3 Seek(Vector3 target) //comportamento de ir até um ponto recebido
{
    Vector3 desired = (target - movement.transform.position).normalized;
    return desired;
}
```

```
1 reference
void Wander() //comportamento de vaguear
{
    timeToNextChange -= Time.deltaTime; //decrementa tempo entre wanders
    if (timeToNextChange <= 0f)
    {
        randomPoint = owner.transform.position + (Vector3)(Random.insideUnitCircle * 20.0f); //gera novo ponto
        //Debug.DrawLine(owner.transform.position, randomPoint, Color.green, 2.0f);
        timeToNextChange = changeTargetInterval; //reseta o timer
    }
}
```

Behaviors 2/4 (SeekOrb)

```
1 reference
Vector3? SeekOrb() //comportamento de buscar orbs
{
    Collider2D[] hits = Physics2D.OverlapCircleAll(owner.transform.position, 5.0f); //detecta todos colliders num raio de 5u
    //Debug.DrawLine(owner.transform.position, owner.transform.position + Vector3.right * 3.0f, Color.green);

    Collider2D nearestOrb = null; //variável para encontrar orb mais próximo
    float minDistance = Mathf.Infinity;

    foreach (var hit in hits) //iteração sobre todos colliders detectados
    {
        if (hit.CompareTag("Orb"))
        {
            float distance = Vector3.Distance(owner.transform.position, hit.transform.position);

            if (distance < minDistance) //atualização de orb mais próximo
            {
                minDistance = distance;
                nearestOrb = hit;
            }
        }
    }
    if (nearestOrb != null) //se for um orb válido
    {
        //Debug.DrawLine(owner.transform.position, nearestOrb.transform.position, Color.red);
        return Seek(nearestOrb.transform.position); //puxa o seek pra posição dele
    }
    return null; //se não encontrar...
}
```

Behaviors 3/4 (ObstacleAvoidance)

```
1 reference
Vector3? ObstacleAvoidance() //comportamento de evitar outros bots
{
    float detectionRadius = 2.0f; //raio de detecção
    Collider2D[] hits = Physics2D.OverlapCircleAll(owner.transform.position, detectionRadius); //detecta todos colliders

    Collider2D nearestObstacle = null; //rastreamento do collider mais próximo
    float closestDistance = Mathf.Infinity;

    foreach (var hit in hits) //iteração sobre todos colliders
    {
        if (hit == null || hit.transform.root == owner.transform.root) //ignora nulos ou que pertençam ao dono do comportamento
            continue;

        if (hit.CompareTag("Body") || hit.CompareTag("Bot")) //se tiver tag body ou bot (cobras)
        {
            float dist = Vector3.Distance(owner.transform.position, hit.transform.position); //calcula distância
            if (dist < closestDistance) //atualiza próximo mais perto
            {
                closestDistance = dist;
                nearestObstacle = hit;
            }
        }
    }

    if (nearestObstacle != null) //se for válido...
    {
        Vector3 away = (owner.transform.position - nearestObstacle.transform.position).normalized; //calcula vetor de fuga
        Vector3 side = Vector3.Cross(away, Vector3.forward).normalized; //calcula direção lateral para desvio
        Vector3 steer = (away + side * 3.0f).normalized; //combina vetores com peso *3 lateral

        //Debug.DrawLine(owner.transform.position, nearestObstacle.transform.position, Color.yellow);
        //Debug.DrawRay(owner.transform.position, steer * detectionRadius, Color.blue);
        //Debug.Log("Steering away from: " + nearestObstacle.name + " (root: " + nearestObstacle.transform.root.name + ")");

        return steer;
    }
    return null; //se não encontrar...
}
```

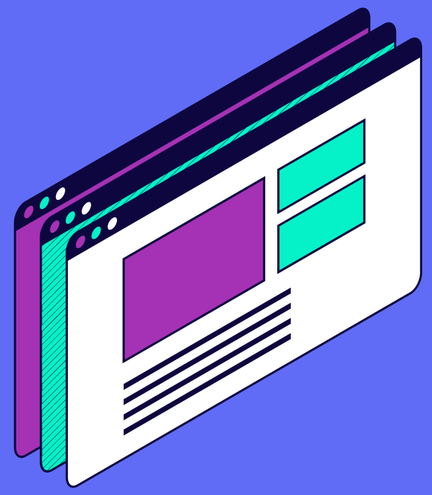
Behaviors 4/4 (FlankEnemy)

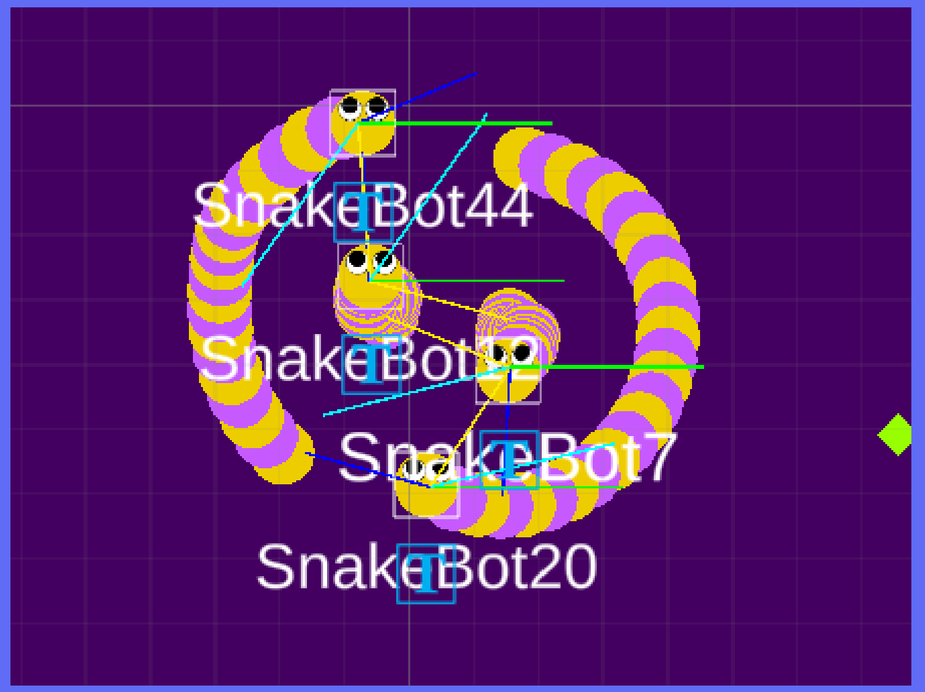
```
1 reference
Vector3? FlankEnemy()
{
    float raioOfensivo = 6.0f; //raio de detecção
    if (movement.bodyParts.Count >= 20) //mudança baseada em tamanho
    {
        raioOfensivo = 6.0f * 2.0f; //mudança para behavior agressivo (expande raio de visão)
    }

    Collider2D[] hits = Physics2D.OverlapCircleAll(owner.transform.position, raioOfensivo); //detecta todos colliders
    Collider2D nearestEnemyHead = null; //collider mais próximo
    float minDist = Mathf.Infinity;

    foreach (var hit in hits)
    {
        if (hit.CompareTag("Bot") && hit.transform.root != owner.transform.root) //filtra por cabeça e não pertencente ao dono do behavior
        {
            float dist = Vector3.Distance(owner.transform.position, hit.transform.position); //calcula distância
            if (dist < minDist) //substitui mais próximo
            {
                minDist = dist;
                nearestEnemyHead = hit;
            }
        }
    }
```

```
    if (nearestEnemyHead != null) //se encontrou inimigo...
    {
        Vector3 toEnemy = (nearestEnemyHead.transform.position - owner.transform.position).normalized; //vetor do dono até a head inimiga

        //calculo de ponto de flanqueamento lateral (direita e esquerda)
        Vector3 flankOffset = Vector3.Cross(toEnemy, Vector3.forward) * 2.0f;
        Vector3 flankTarget = nearestEnemyHead.transform.position + flankOffset;

        //Debug.DrawLine(owner.transform.position, flankTarget, Color.cyan);
        return Seek(flankTarget); //retorna vetor de perseguição ao ponto de flanqueamento
    }
    return null;
}
```

Problemas Enfrentados

- Vários problemas de lógica nas verificações de if/else dos comportamentos combinados e dentro de métodos (foi um inferno acertar o ObstacleAvoidance() pois demorei pra notar que não existe Collider2D no projeto, aí precisei deixar raycast de lado);
- ObstacleAvoidance foi minha maior dificuldade mas quando consegui implementá-lo, abriu caminho pra simplificar o resto;
- Precisei criar vários debugs pra ter certeza que estava tudo correto pois não conseguia visualizar o comportamento direito;
- Gostaria de ter implementado mais coisas, ainda acho que ficou muito simples, mas gostei do projeto.





https://github.com/rikmgomes/slitherio-simple-bot

Obrigado pela atenção!

Ricardo Moreira Gomes

{ Trabalho Grau A }

Inteligência Artificial para Jogos 2025/1