Projeto final

Explicando o código

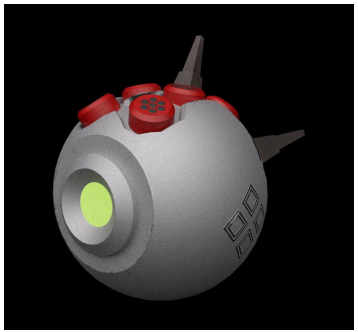
Acadêmicos: Diego Hartmann e Philipp Altendorf

Disciplina: Inteligência Artificial

Professor: Ricardo Cherobin e Marcelo Dornbusch

Três IA’s principais

Vespa

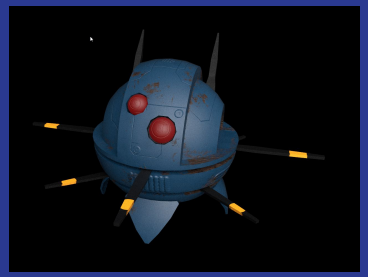


- **Flocking** → movimentação em bando para atacar Inimigos em um enxame.

Inimigo

- **M.E** → altera e Aplica estados ;
- **A*** → voltar para área de patrulha ;
- **Waypoints** → patrulha dentro da área.

Minion



- **M.E** → aplica estados (alterados em Eventos);
- **A*** → seguir *player* ;

Máquina de Estados (inimigo)

Tutorial: nenhum

Drone inimigo

```
public enum DroneStates{
    Patrol,
    Chase,
    Search,
    BackToPatrol,
    Attack,
}
```

```
private void Update(){
    LookForTargets();
    SetState();
    ExecuteState();
}
```

Drone inimigo

```
public enum DroneStates{
    Patrol,
    Chase,
    Search,
    BackToPatrol,
    Attack,
}
```

```
private void Update(){
    LookForTargets();
    SetState();
    ExecuteState();
}
```

```
private void SetState(){
    if (HasATarget())
    {
        RefillSearchTimer();
        SeesTarget(true);
        if (ReachedAnyPatrolPoint()){
            ReachedAnyPatrolPoint(false);
        }
        if(!DroneCanRequestAPathAStar()){
            DroneCanRequestAPathAStar(true);
        }
        if (DistFromTarget_GraterThan(comp.status.distanceToAttack)){
            State = DroneStates.Chase;
            return;
        }
        State = DroneStates.Attack;
        return;
    }
```

```
        SeesTarget(false);
        if (AnotherDroneHasATarget()){
            RefillSearchTimer();
            State = DroneStates.Search;
            return;
        }
        if (IsTimeToSearch()){
            State = DroneStates.Search;
            comp.status.searchTimer -= Time.deltaTime;
            return;
        }
        if (!ReachedAnyPatrolPoint()){
            State = DroneStates.BackToPatrol;
            return;
        }
        if(DroneCanRequestAPathAStar()){
            DroneCanRequestAPathAStar(false);
        }
        State = DroneStates.Patrol;
    }
```

Drone inimigo

```
public enum DroneStates{
    Patrol,
    Chase,
    Search,
    BackToPatrol,
    Attack,
}
```

```
private void Update(){
    LookForTargets();
    SetState();
    ExecuteState();
}
```

```
private void ExecuteState(){
    switch (State)
    {
        case DroneStates.Attack:
            comp.actions.Attack();
        break;

        case DroneStates.Chase:
            comp.actions.Chase();
        break;

        case DroneStates.Search:
            comp.actions.Search();
        break;

        case DroneStates.BackToPatrol:
            comp.actions.GoingBackToPatrol();
        break;

        case DroneStates.Patrol:
            comp.actions.Patrol();
        break;

        default:
        break;
    }
```

A* Path (inimigo)

Tutorial: youtube.com/watch?v=dn1XRIaROM4&ab_channel=SebastianLague

Quando o agente entra no estado **BackToPatrol**...

Ele volta a seus waypoints de patrulha através de um caminho criado pelo algoritmo **A***.

```
private void AStartTo(Vector3 finalPos){
    if(comp.aStar.canRequestAPath){
        comp.aStar.RequestAPath(finalPos);
    }
    comp.aStar.UpdatePathfindingWay();
    RotateTo(comp.aStar.currentTargetWaypoint,
    MoveForward(comp.status.aStarSpeed);
}
```

Quando o agente entra no estado **BackToPatrol**...

Para isso, são criados Waypoints nas vértices (curvas) do caminho criado pelo A*.

```
private void AStartTo(Vector3 finalPos){
    if(comp.aStar.canRequestAPath){
        comp.aStar.RequestAPath(finalPos);
    }
    comp.aStar.UpdatePathfindingWay();
    RotateTo(comp.aStar.currentTargetWaypoint,
    MoveForward(comp.status.aStarSpeed);
}
```

Classe / métodos criados com o tutorial de Sebastian

```
public static void RequestPath(Vector3 pathStart, Vector3 pathEnd, Action<Vector3[], bool> callback){
    PathRequest newRequest = new PathRequest (pathStart, pathEnd, callback);
    instante.pathRequestQueue.Enqueue(newRequest);
    instante.TryProcessNext();
}

private void TryProcessNext() {
    if (!isProcessingPath && pathRequestQueue.Count > 0) {
        currentPathRequest = pathRequestQueue.Dequeue();
        isProcessingPath = true;
        pathfinding.StartFindingPath(currentPathRequest.pathStart, currentPathRequest.pathEnd);
    }
}

public void FinishedProcessingPath(Vector3[] path, bool success) {
    currentPathRequest.callback(path,success);
    isProcessingPath = false;
    TryProcessNext();
}
```

Quando o agente entra no estado **BackToPatrol**...

Esses waypoints são percorridos através de um algoritmo de waypoints comum.

```
private void AStartTo(Vector3 finalPos){
    if(comp.aStar.canRequestAPath){
        comp.aStar.RequestAPath(finalPos);
    }
    comp.aStar.UpdatePathfindingWay();
    RotateTo(comp.aStar.currentTargetWaypoint,
    MoveForward(comp.status.aStarSpeed);
}
```

```
public void UpdatePathfindingWay() {
    if(startFollow && (targetIndex < path.Length)){
        if (DistanceFrom(targetIndex) < 1f)
        {
            targetIndex ++;
            if (targetIndex == path.Length){
                startFollow = false;
                canRequestAPath = true;
                targetIndex = 0;
                return;
            }
            currentTargetWaypoint = path[targetIndex];
        }
        return;
    }
    canRequestAPath = true;
    targetIndex = 0;
    startFollow = false;
}
```

Waypoints (inimigo)

Tutorial: nenhum

Quando o agente entra no estado **Patrol**...

Ele entra em um caminho de waypoints pré-determinados.

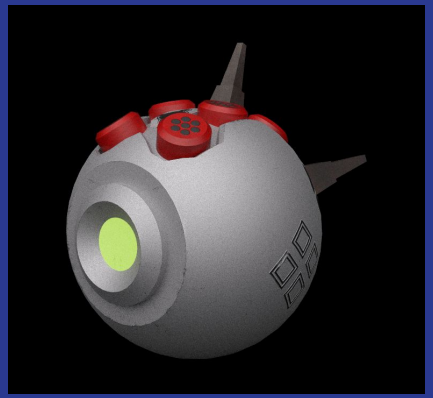
```
public void Patrol(){
    comp.patrol.SimpleWaypoints();
    RotateTo(comp.patrol.targetWaypoint.position,
    MoveForward(comp.status.patrolSpeed);
}
```

O *waypoint-alvo* é atualizado para o próximo...

...apenas quando o drone chega a uma distância X do *waypoint-alvo* atual.

```
public void SimpleWaypoints(){
    SetNextWaypointBasedOn(targetIndex);
}

private void SetNextWaypointBasedOn(int _index){
    if (DistanceFrom(_index) < minDistToWaypoint){
        SetBoolFlag();
        TargetIndexEqualsTo(_index + direction);
    }
}
```



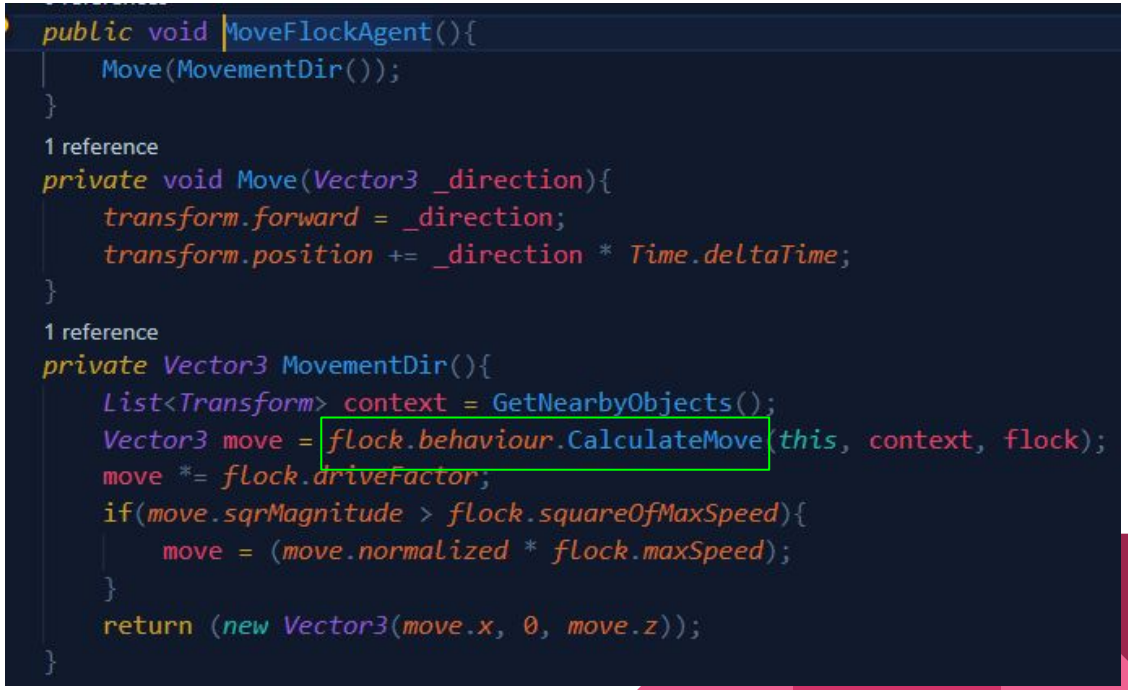
Flocking (vespa)

Tutorial: youtube.com/watch?v=i_XinoVBqt8&ab_channel=BoardToBitsGames

Criados com tutorial de BoardToBitsGames

O Minion é movido pela função MoveFlockAgent();

Essa função leva em consideração a direção calculada pelo Flock;



```
public void MoveFlockAgent(){
    Move(MovementDir());
}

1 reference
private void Move(Vector3 _direction){
    transform.forward = _direction;
    transform.position += _direction * Time.deltaTime;
}

1 reference
private Vector3 MovementDir(){
    List<Transform> context = GetNearbyObjects();
    Vector3 move = flock.behaviour.CalculateMove(this, context, flock);
    move *= flock.driveFactor;
    if(move.sqrMagnitude > flock.squareOfMaxSpeed){
        move = (move.normalized * flock.maxSpeed);
    }
    return (new Vector3(move.x, 0, move.z));
}
```

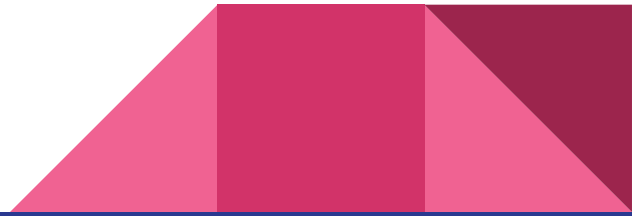
Essa direção (Vector3) retorna a soma de...

...Align, Cohesion, Avoidance.

```
public override Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock)
{
    if (weights.Length != behaviors.Length){
        Debug.LogError("numero de behaviours é diferente do numero de pesos, em " + name, this);
        return Vector3.zero;
    }
    //seta movimento
    Vector3 move = Vector3.zero;
    //passa pelos behaviours (align, cohesion, avoidance)
    for (int i = 0; i < behaviors.Length; i++){
        Vector3 partialMove = behaviors[i].CalculateMove(agent, context, flock) * weights[i];
        if (partialMove != Vector3.zero){
            if (partialMove.sqrMagnitude > weights[i] * weights[i]){
                partialMove.Normalize();
                partialMove *= weights[i];
            }
            move += partialMove;
        }
    }
    return move;
}
```

Ou seja, cada regra tem seu próprio CalculateMove()...

...para que possam ser somados no *behaviour-final* mencionado anteriormente.



Cohesion

```
public class FlockCohesion : FilteredFlockBehaviour
{
    0 references
    public override Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock){
        if(context.Count == 0){
            return Vector3.zero;
        }
        Vector3 cohesionMove = Vector3.zero;
        List<Transform> filteredContext = (filter == null) ? context : filter.Filter(agent, context);
        foreach (Transform item in filteredContext){
            cohesionMove += (item.position);
        }
        cohesionMove /= context.Count;
        //diferença
        cohesionMove -= (agent.transform.position);
        return cohesionMove;
    }
}
```

Alignment

```
[CreateAssetMenu(menuName = "Flock/Behaviour/Alignment")]
public class FlockAlignment : FilteredFlockBehaviour
{
    public override Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock){
        if(context.Count == 0){
            return agent.transform.forward;
        }
        Vector3 alignmentMove = Vector3.zero;
        List<Transform> filteredContext = (filter == null) ? context : filter.Filter(agent, context);
        foreach (Transform item in filteredContext){
            alignmentMove += (item.transform.forward);
        }
        alignmentMove /= context.Count;
        return alignmentMove;
    }
}
```

Avoidance

```
public class FlockAvoidance : FilteredFlockBehaviour
{
    public override Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock){
        if(context.Count == 0){
            return Vector3.zero;
        }
        Vector3 avoidanceMove = Vector3.zero;
        int nAvoid = 0;
        List<Transform> filteredContext = (filter == null) ? context : filter.Filter(agent, context);
        foreach (Transform item in filteredContext){
            float magnitude = Vector3.SqrMagnitude(item.position - agent.transform.position);
            if(magnitude < flock.SquareOfAvoidanceRadious){
                nAvoid ++;
                avoidanceMove += (agent.transform.position - item.position);
            }
        }
        if(nAvoid > 0){
            avoidanceMove /= nAvoid;
        }
        return avoidanceMove;
    }
```

Conclusão

- Embora Inimigo e Minion compartilhem M.E e A*, o Inimigo foi usado como exemplo por se tratar de implementações mais complexas.

- Ainda há dúvida em onde, exatamente, utilizar o ***Flocking*** com os minions. Há algumas ideias como:
  - Seguir o minion líder em direção ao player (médio) ou;
  - **Criar um enxame para atacar/distrair inimigos(fácil);**
  - Criar um estado de “susto” onde os minions se dispersam (difícil).

Obrigado pela atenção!