Inteligencia Artificial para Videojuegos

Movimiento

Comportamientos de dirección

Motivación

- Aunque no hay definición formal de "comportamiento de dirección", para nosotros equivale a movimiento dinámico
  - Algoritmos que trabajan modificando la velocidad y la rotación del agente
  - Hay miles, pero veremos unos básicos y *cómo combinarlos*
- Muy utilizado sobre todo en juegos de conducción

Motivación

- Craig Reynolds fue el primero en proponer los comportamientos de dirección, en un artículo de la GDC en 1999

Steering Behaviors For Autonomous Characters

Craig W. Reynolds

Sony Computer Entertainment America
919 East Hillsdale Boulevard
Foster City, California 94404

craig_reynolds@playstation.sony.com
http://www.red.com/cwr/
cwr@red.com

Keywords: Animation Techniques, Virtual/Interactive Environments, Games, Simulation, behavioral animation, autonomous agent, situated, embodied, reactive, vehicle, steering, path planning, path following, pursuit, evasion, obstacle avoidance, collision avoidance, flocking, group behavior, navigation, artificial life, improvisation.

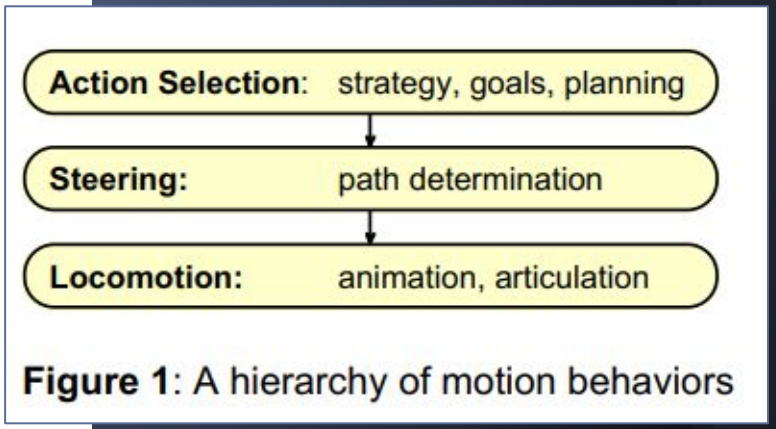


Figure 1: A hierarchy of motion behaviors

Abstract

This paper presents solutions for one requirement of autonomous characters in animation and games: the ability to navigate around their world in a life-like and improvisational manner. These "steering behaviors" are largely independent of the particulars of the character's means of locomotion. Combinations of steering behaviors can be used to achieve higher level goals (For example: get from here to there while avoiding obstacles, follow this corridor, join that

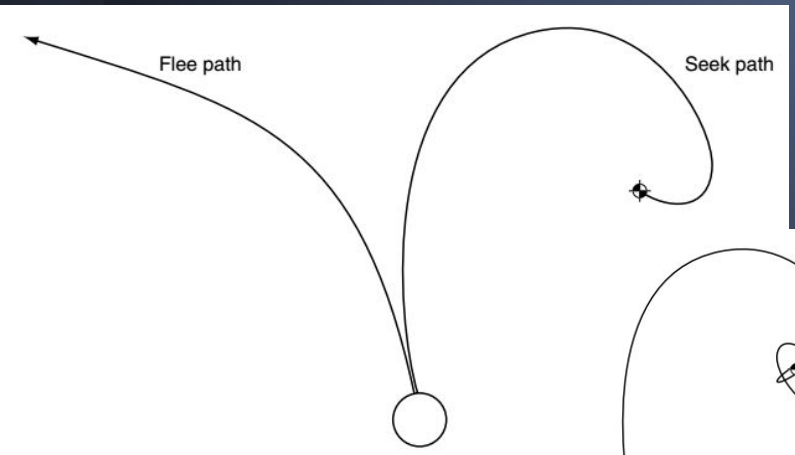
Puntos clave

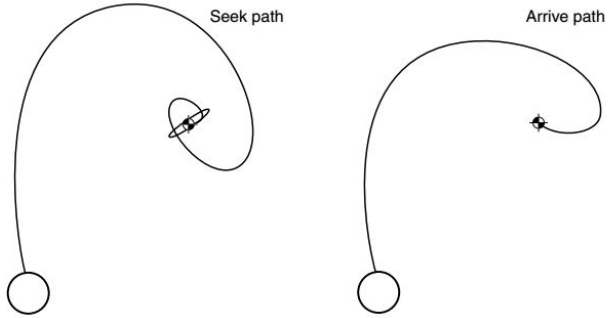
- Comportamientos de dirección
  - Seguimiento y huida
  - Llegada
  - Alineamiento
  - Equiparación de velocidad
- Más comportamientos de dirección
  - Persecución y evasión
  - Encaramiento
  - Orientación según velocidad
  - Merodeo

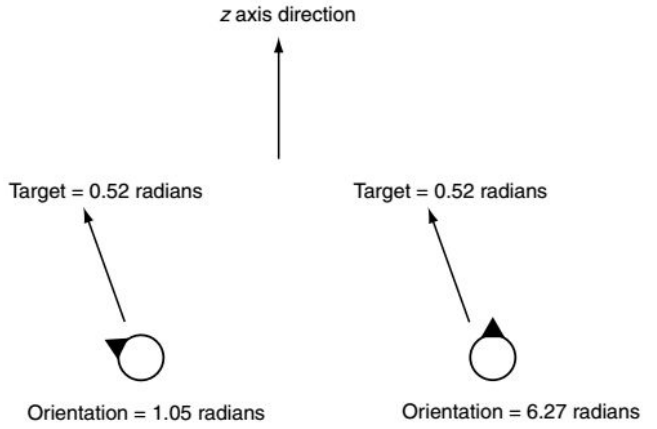
Comportamientos de dirección

STEERING BEHAVIORS

- Extensiones a los movimientos cinemáticos, que dan lugar a movimientos dinámicos
  - Reciben velocidad y rotación del agente (entrada) y aplican *aceleración* en la dirección (salida)
  - Hay algunos *básicos*, como seguimiento y huida, llegada, alineamiento o equiparación de velocidad







Comportamientos de dirección

- Estructura básica
  - Leen la información cinemática del agente y, según sea necesario, pueden leer algo de información (estática o cinemática) del objetivo
  - El objetivo puede ser uno o varios agentes, caminos, colisiones (o la geometría del nivel)
- Suele consistir en equiparar (o diferenciar, en el caso opuesto) los valores de las variables del agente y el objetivo
  - Ej. Ocupar la misma posición que el objetivo
  - Para varias variables, es mejor hacer *algoritmos separados* para cada variable… y luego combinarlos

Comportamientos de dirección

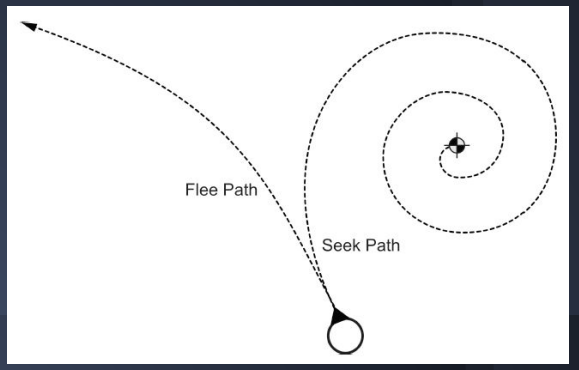
- Estos algoritmos tratan siempre de acelerar al máximo, ir lo más rápido posible, etc. para realizar la equiparación/diferenciación
  - Por eso restringimos los valores, al final de cada tick, para no exceder unos máximos (de seguridad)
  - A veces también aplicamos rozamiento, en la parte física del tick, para *contener el crecimiento de los valores* y *agotar de forma realista* el movimiento del agente

Comportamientos de dirección

```
class Kinematic:
    # ... Member data as before ...

    function update(steering: SteeringOutput,
                    maxSpeed: float,
                    time: float):
        # Update the position and orientation.
        position += velocity * time
        orientation += rotation * time

        # and the velocity and rotation.
        velocity += steering.linear * time
        rotation += steering.angular * time

        # Check for speeding and clip.
        if velocity.length() > maxSpeed:
            velocity.normalize()
            velocity *= maxSpeed
```

Seguimiento y huida

SEEK & FLEE



- Trata de equiparar la posición del agente con la del objetivo

```
class Seek:
    character: Kinematic
    target: Kinematic

    maxAcceleration: float

    function getSteering( ) -> SteeringOutput:
        result = new SteeringOutput( )

        # Get the direction to the target.
        result.linear = target.position - character.position

        # Give full acceleration along this direction.
        result.linear.normalize( )
        result.linear *= maxAcceleration

        result.angular = 0
        return result
```

* Si quieres cambiar también la orientación del agente , lo suyo es combinar este comportamiento con otro de **alinear**

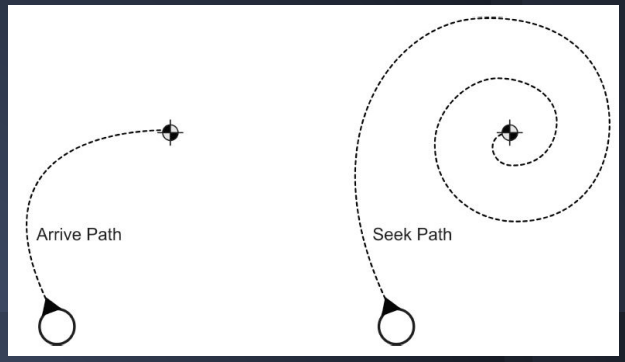
Tiempo = O(1)
Espacio = O(1)

* Modificación para Huida

```
# Get the direction to the target.
steering.linear = character.position - target.position
```

Llegada

ARRIVAL



- Igual pero deteniéndose, usando *dos radios*
  - Uno para ir *reduciendo la velocidad* y otro para *ponerla a cero* por ya estar suficientemente cerca (margen de error)
  - Más *un tiempo* para conseguir la velocidad deseada

```
1  class Arrive:
2      character: Kinematic
3      target: Kinematic
4
5      maxAcceleration: float
6      maxSpeed: float
7
8      # The radius for arriving at the target.
9      targetRadius: float
10
11     # The radius for beginning to slow down.
12     slowRadius: float
13
14     # The time over which to achieve target speed.
15     timeToTarget: float = 0.1
16
```

Llegada

```
17      function getSteering( ) -> SteeringOutput:
18          result = new SteeringOutput( )
19
20          # Get the direction to the target.
21          direction = target.position - character.position
22          distance = direction.length( )
23
24          # Check if we are there, return no steering.
25          if distance < targetRadius:
26              return null
27
28          # If we are outside the slowRadius, then move at max speed.
29          if distance > slowRadius:
30              targetSpeed = maxSpeed
31          # Otherwise calculate a scaled speed.
32          else:
33              targetSpeed = maxSpeed * distance / slowRadius
34
35          # The target velocity combines speed and direction.
```

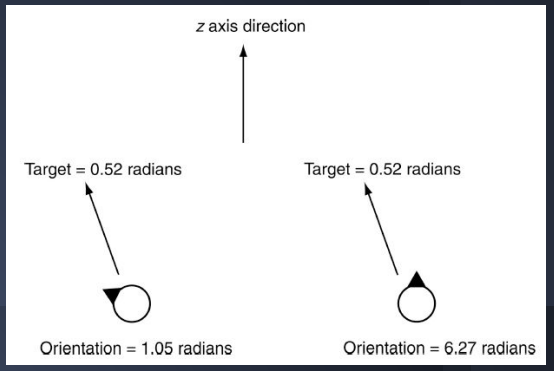
Llegada

```
36          targetVelocity = direction
37          targetVelocity.normalize()
38          targetVelocity *= targetSpeed
39
40          # Acceleration tries to get to the target velocity.
41          result.linear = targetVelocity - character.velocity
42          result.linear /= timeToTarget
43
44          # Check if the acceleration is too fast.
45          if result.linear.length() > maxAcceleration:
46              result.linear.normalize()
47              result.linear *= maxAcceleration
48
49          result.angular = 0
50          return result
```

Tiempo = O(1)
Espacio = O(1)

Alineamiento

ALIGN



- Como Llegada, pero tratando de equiparar la orientación del agente y la del objetivo
  - Rotando *por el camino más corto* (-π, π)
    - ¡Cuidado con rotar más rápido que π por tick!
  - Y *dos ángulos*, por lo mismo: uno para empezar a *reducir la velocidad de rotación* y otro como *margen de error*

```
class Align:
    character: Kinematic
    target: Kinematic

    maxAngularAcceleration: float
    maxRotation: float

    # The radius for arriving at the target.
    targetRadius: float

    # The radius for beginning to slow down.
    slowRadius: float

    # The time over which to achieve target speed.
    timeToTarget: float = 0.1
```

Alineamiento

```
17      function getSteering() -> SteeringOutput:
18          result = new SteeringOutput()
19
20          # Get the naive direction to the target.
21          rotation = target.orientation - character.orientation
22
23          # Map the result to the (-pi, pi) interval.
24          rotation = mapToRange(rotation)
25          rotationSize = abs(rotation)
26
27          # Check if we are there, return no steering.
28          if rotationSize < targetRadius:
29              return null
30
31          # If we are outside the slowRadius, then use maximum rotation.
32          if rotationSize > slowRadius:
33              targetRotation = maxRotation
34          # Otherwise calculate a scaled rotation.
35          else:
36              targetRotation =
37                  maxRotation * rotationSize / slowRadius
```

*** mapToRange** también se conoce como "hallar el ángulo delta"

Alineamiento

```
38
39          # The final target rotation combines speed (already in the
40          # variable) and direction.
41          targetRotation *= rotation / rotationSize
42
43          # Acceleration tries to get to the target rotation.
44          result.angular = targetRotation - character.rotation
45          result.angular /= timeToTarget
46
47          # Check if the acceleration is too great.
48          angularAcceleration = abs(result.angular)
49          if angularAcceleration > maxAngularAcceleration:
50              result.angular /= angularAcceleration
51              result.angular *= maxAngularAcceleration
52
53          result.linear = 0
54          return result
```

* Lo contrario es el Alineamiento en sentido opuesto
(sumar π a la orientación del objetivo y alinearse a eso)

Tiempo = O(1)
Espacio = O(1)

Equiparación de velocidad

- Se reutiliza el trozo que interesa de Llegada
  - No suele usarse sólo, aunque sí combina con otros

```
class VelocityMatch:
    character: Kinematic
    target: Kinematic

    maxAcceleration: float

    # The time over which to achieve target speed.
    timeToTarget = 0.1

    function getSteering( ) -> SteeringOutput:
        result = new SteeringOutput( )

        # Acceleration tries to get to the target velocity.
        result.linear = target.velocity - character.velocity
        result.linear /= timeToTarget

        # Check if the acceleration is too fast.
        if result.linear.length( ) > maxAcceleration:
            result.linear.normalize( )
            result.linear *= maxAcceleration

        result.angular = 0
        return result
```

Tiempo = O(1)
Espacio = O(1)

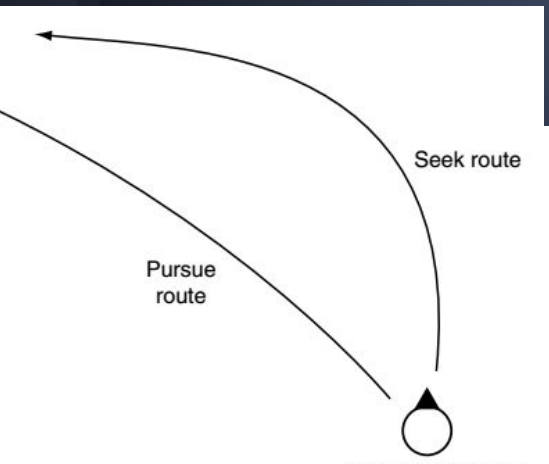
Participación

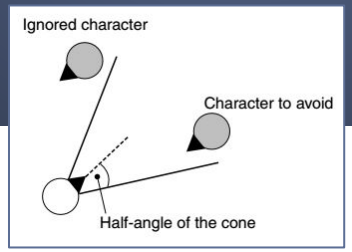
tiny.cc/IAV

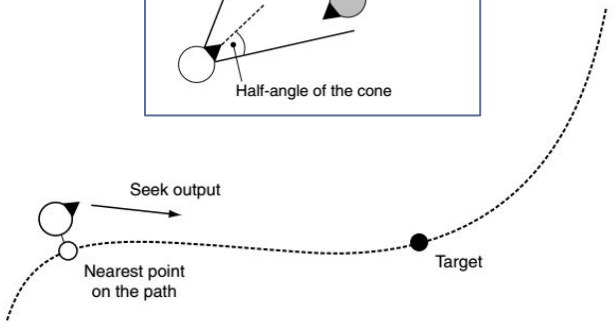
- Entre los dos radios de la Llegada...
  A. Aumenta la velocidad desde cero hasta el máximo
  B. Se reduce la aceleración hasta el máximo
  C. Se reduce la velocidad desde máximo hasta cero
  D. Aumenta la aceleración de cero hasta el máximo
- Desarrolla tu respuesta (en texto libre)

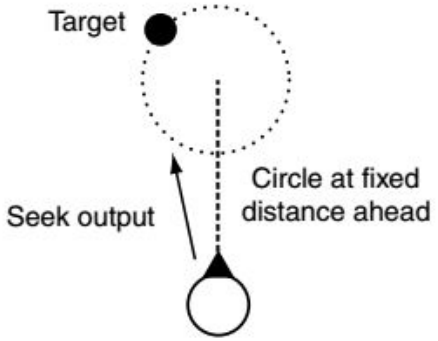
Más comportamientos de dirección

- Los básicos se pueden *combinar* (mediante herencia o mediante composición) en otros: persecución y evasión, encaramiento, orientación según velocidad, merodeo...
  - A veces es necesario *predecir* la posición futura de un objetivo, incluso proyectando volúmenes



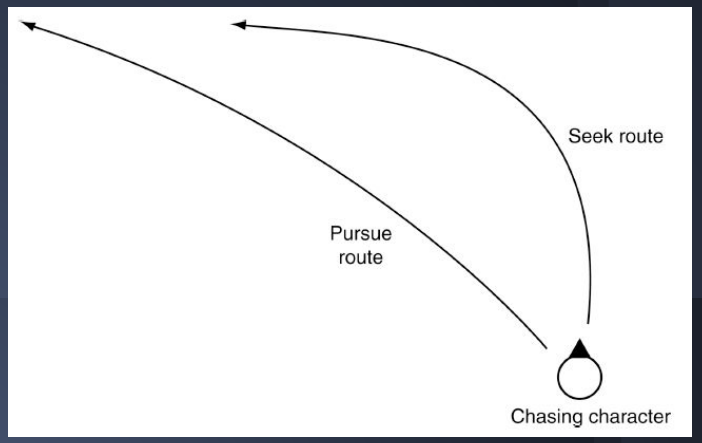




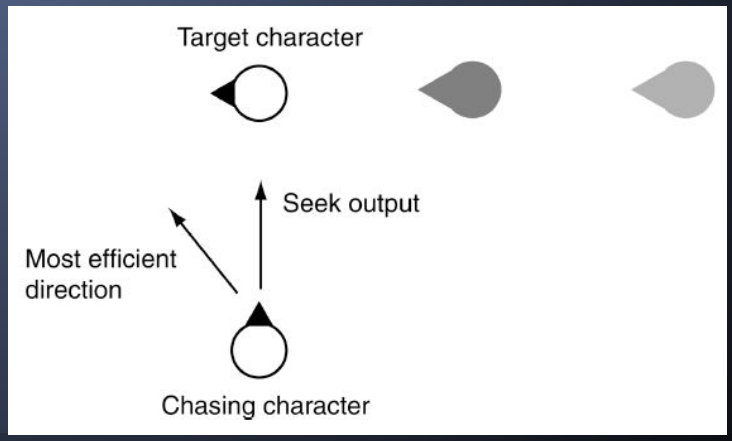


Persecución y evasión

PURSUE & EVADE



- Heredando de Seguimiento y Huida añadiendo una predicción básica (tener en cuenta *la velocidad del objetivo*)
  - Se calcula lo que tardaría en llegar hasta el objetivo y ese tiempo (hasta un máximo) es lo que sirve para para predecir *donde estará el objetivo*
  - También se puede implementar mediante composición, claro… pero hay que copiar más valores



Persecución y evasión

```
class Pursue extends Seek:
    # The maximum prediction time.
    maxPrediction: float

    # OVERRIDES the target data in seek (in other words this class has
    # two bits of data called target: Seek.target is the superclass
    # target which will be automatically calculated and shouldn't be
    # set, and Pursue.target is the target we're pursuing).
    target: Kinematic

    # ... Other data is derived from the superclass ...

    function getSteering( ) -> SteeringOutput:
        # 1. Calculate the target to delegate to seek
        # Work out the distance to target.
        direction = target.position - character.position
        distance = direction.length( )

        # Work out our current speed.
        speed = character.velocity.length( )

        # Check if speed gives a reasonable prediction time.
        if speed <= distance / maxPrediction:
            prediction = maxPrediction
```

Persecución y evasión

```
26          # Otherwise calculate the prediction time.
27          else:
28              prediction = distance / speed
29
30          # Put the target together.
31          Seek.target = explicitTarget
32          Seek.target.position += target.velocity * prediction
33
34          # 2. Delegate to seek.
35          return Seek.getSteering( )
```

* Lo contrario es Evasión
(cambiar *Seek* por *Flee*, básicamente...)

Tiempo = O(1)
Espacio = O(1)

Encaramiento

FACE

- Trata de modificar la orientación según la posición relativa del agente con el objetivo
  - Reutiliza Alineamiento

```
class Face extends Align:
    # Overrides the Align.target member.
    target: Kinematic

    # ... Other data is derived from the superclass ...

    # Implemented as it was in Pursue.
    function getSteering( ) -> SteeringOutput:
        # 1. Calculate the target to delegate to align
        # Work out the direction to target.
        direction = target.position - character.position

        # Check for a zero direction, and make no change if so.
        if direction.length( ) == 0:
            return target

        # 2. Delegate to align.
        Align.target = explicitTarget
        Align.target.orientation = atan2(-direction.x, direction.z)
        return Align.getSteering( )
```

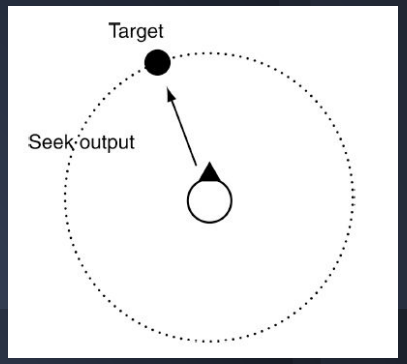
Orientación según velocidad

- Trata de orientar al agente hacia donde este se está moviendo
  - Reutiliza Alineamiento, también

```
class LookWhereYoureGoing extends Align:
    # No need for an overridden target member, we have
    # no explicit target to set.

    # ... Other data is derived from the superclass ...

    function getSteering() -> SteeringOutput:
        # 1. Calculate the target to delegate to align
        # Check for a zero direction, and make no change if so.
        velocity: Vector = character.velocity
        if velocity.length() == 0:
            return null

        # Otherwise set the target based on the velocity.
        target.orientation = atan2(-velocity.x, velocity.z)

        # 2. Delegate to align.
        return Align.getSteering()
```

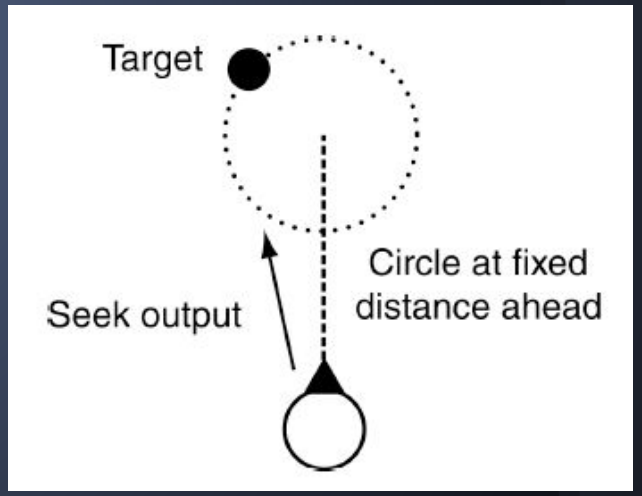
Tiempo = O(1)
Espacio = O(1)

Merodeo



- Suaviza el merodeo cinemático, fijando un objetivo aleatorio (situado unos pasos por delante) y siguiéndolo dinámicamente

```
1  class Wander extends Face:
2      # The radius and forward offset of the wander circle.
3      wanderOffset: float
4      wanderRadius: float
5
6      # The maximum rate at which the wander orientation can change.
7      wanderRate: float
8
9      # The current orientation of the wander target.
10     wanderOrientation: float
11
12     # The maximum acceleration of the character.
13     maxAcceleration: float
14
15     # Again we don't need a new target.
16     # ... Other data is derived from the superclass ...
```



Merodeo

```
18      function getSteering() -> SteeringOutput:
19          # 1. Calculate the target to delegate to face
20          # Update the wander orientation.
21          wanderOrientation += randomBinomial() * wanderRate
22
23          # Calculate the combined target orientation.
24          targetOrientation = wanderOrientation + character.orientation
25
26          # Calculate the center of the wander circle.
27          target = character.position +
28                   wanderOffset * character.orientation.asVector()
29
30          # Calculate the target location.
31          target += wanderRadius * targetOrientation.asVector()
32
33          # 2. Delegate to face.
34          result = Face.getSteering()
35
36          # 3. Now set the linear acceleration to be at full
37          # acceleration in the direction of the orientation.
38          result.linear =
39              maxAcceleration * character.orientation.asVector()
40
41          # Return it.
42          return result
```

Tiempo = O(1)
Espacio = O(1)

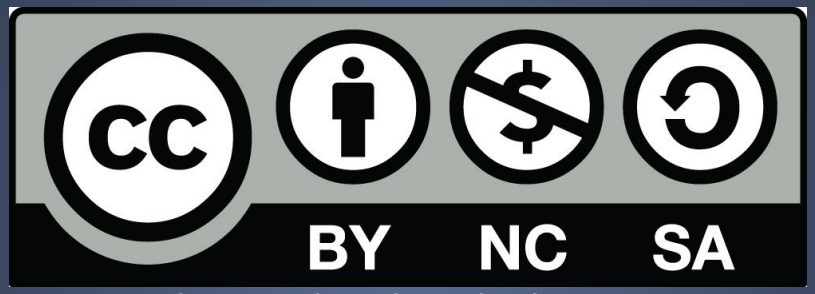
Resumen

- Los comportamientos de dirección son una forma “inteligente” de moverse
- Los más básicos son el seguimiento y la huida, la llegada a un punto, el alineamiento y la equiparación de velocidad
- Hay otros que surgen de “combinar” un poco los anteriores, como persecución y evasión, encaramiento, orientación según velocidad o el merodeo de algunos agentes

Más información

- Millington, I.: Artificial Intelligence for Games. CRC Press, 3rd Edition (2019)

Críticas, dudas, sugerencias...



Federico Peinado (2019-2020)

www.federicopeinado.es