Inteligencia Artificial para Videojuegos

Movimiento

Percepción

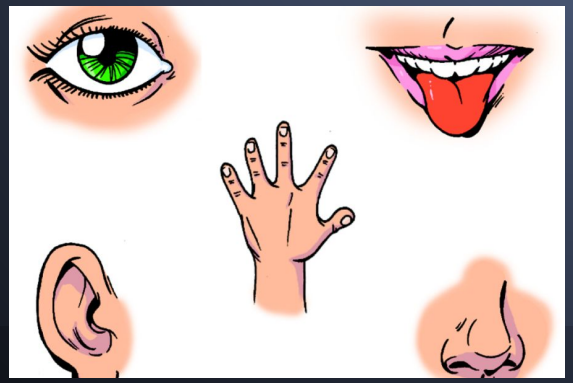
Motivación

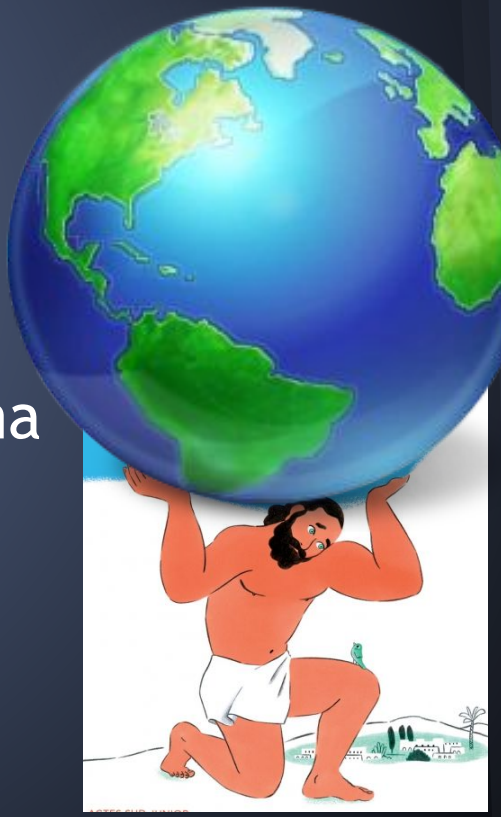
- Lo primero que hace un agente inteligente es abrirse al mundo y percibir la realidad

"Poca observación y mucho razonamiento llevan al error. Mucha observación y poco razonamiento llevan a la verdad" - Alexis Carrel

Motivación

- Según Millington, la percepción (conexión entre el agente y el mundo de juego) es lo que *más esfuerzo* cuesta desarrollar y depurar en *IA para videojuegos*
  - Sobre todo si se construye un sistema de simulación de sentidos físicos





Puntos clave

- Percepción del jugador
  - Ventana de percepción
- Hitos históricos
- Gestión sensorial
  - Vista
  - Oído
  - Tacto, olfato y otros
- Modelo de gestión sensorial regional

Percepción del jugador

- En este tema hablamos de cómo percibe una IA *lo que ocurre* en el juego
- Pero puede ser útil recordar aquí cómo funciona la percepción en el jugador
  - El concepto de ventana de percepción

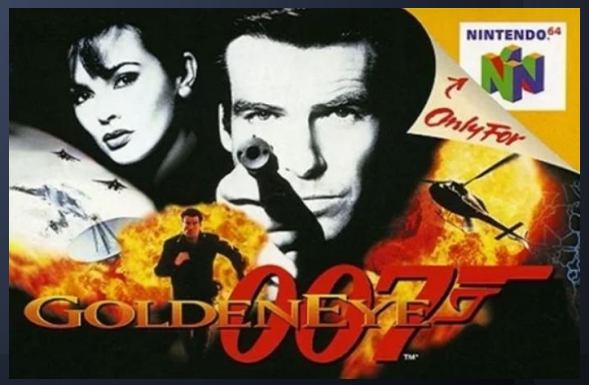
Ventana de percepción

- Cuanto más tiempo vayamos a estar viendo un personaje en el juego, mejor tendrá que ser su IA para que no defraude
  - No sólo cuenta el tiempo, sino la cantidad de comportamientos diferentes y sobre todo la cantidad de cambios de comportamiento (lo ideal es cambiar sólo cuando interactuamos con él)
- El comportamiento del personaje debe alinearse con lo que se quieren expresar y debe percibir el jugador promedio
  - Ni más ni menos comportamientos que en el diseño, ni más ni menos *sofisticados*

Hitos históricos

"Antes de simular… intenta falsear"

- En los videojuegos más primitivos apenas había "percepción" como tal
  - Ej. En los clones de Pong con oponentes controlados por ordenador, la raqueta "ve" donde está la pelota y se mueve en consecuencia
- En los 90 aparecen los primeros sistemas de simulación de sentidos físicos
  - En Goldeneye 007, los enemigos podían "ver" a sus compañeros muertos y se alarmaban de ello



Hitos históricos

- En títulos como Splinter Cell, Thief: The Dark Project, y Metal Gear Solid, todo se basa en el sigilo
  - La capacidad de los agentes inteligentes de ver únicamente *objetos iluminados* es la base de esto
- Se trata de una tendencia clara de futuro
  - Pronto veremos percepción sensorial *integrada* a la hora de diseñar videojuegos en géneros como estrategia, rol y plataformas (acción)

Hitos históricos

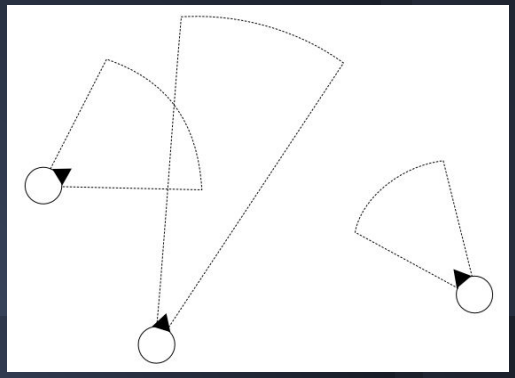
- Muchos personajes de videojuego en realidad son omniscientes
  - Ej. Siempre conocen donde está el jugador
  - El comportamiento, eso sí, se diseña para *simular la ilusión* de que no saben nada
  - Aunque el personaje te haya mirado antes, si te acercas suficientemente, empieza a perseguirte… fingiendo que realmente no ha "reparado en ti" hasta no tenerte cerca
  - Esto es más *Diseño* que *IA*, porque *los personajes no tienen sentidos, sino que fingen tenerlos*

Gestión sensorial

SENSE MANAGEMENT

- Para *simular* los sentidos de un NPC extendemos el mecanismo de paso de mensajes
  - Ej. Un gestor de eventos simple puede notificar que hay ruido a los NPCs que estén en una habitación
  - Para que esto *escale*, surge el gestor sensorial
- Revisaremos técnicas *para cada sentido*
  - Vista
  - Oído
  - Tacto, olfato y otros

Vista



RAY CASTING

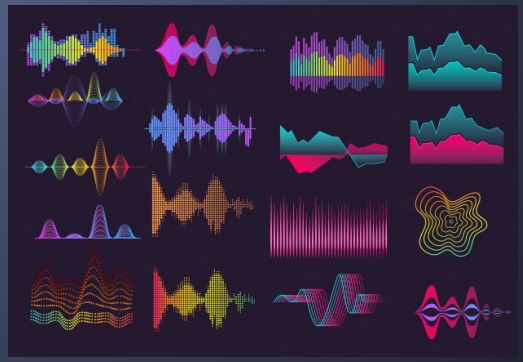
- La solución obvia es la proyección de rayos de toda la ida: ver si hay línea de visión
  - La vista no dobla esquinas, funciona recibiendo fotones en *línea recta* y es muy característico
  - Habitualmente se ignora el hecho de que la luz puede rebotar, que existen los espejos, etc.
- El problema de los rayos es que no escalan muy bien y se vuelven casi intratables
- Además, si nos preocupa la precisión, podemos programar nosotros nuestro propio gestor sensorial

Vista

- Es el más sofisticado: el jugador se da cuenta si la vista del NPC es “fingida”
  - Se utilizan conos de visión (220º en horizontal, 60º para simular la atención; y 120º en vertical), así como líneas de visión y distancias máximas de visión (en relación al tamaño del objeto a detectar)
  - Aunque instantánea, esto se hace para que los NPCs no tenga “demasiada buena vista”; al igual que ciertos juegos nos permiten camuflarnos o escondernos en zonas oscuras del entorno
    - Ej. Ghost Recon

Oído

- No hace falta *simular* cómo rebotan las ondas sonoras, se atenúan, se desplazan a distinta velocidad según el medio,
  y llegan a nuestros oídos
  - Ej. Barrito de elefante vs. Chillido de murciélago
- A menudo simplemente se avisa a todos los personajes que comparten localización con aquello que produce el ruido



Oído

- El sonido real va a 345 m/s en el aire, pero suele simularse *instantáneo*, como la luz
  - Aunque hay juegos que sí permiten apreciar la velocidad del sonido, como *Conflict: Desert Storm*
  - Hoy día hay plugins como *Steam Audio** que simulan el sonido de forma aún más realista

* Sonido 3D con HRTFs, oclusión total y parcial, más efectos del entorno y automatizados, reverberación acorde a tu escena y por convolución, seguimiento de la cabeza en VR

Tacto, olfato y otros

- El tacto suele interpretarse como colisión
  - El sistema de colisiones avisa al agente
- El olfato se ha explorado poco (Ej. en la caza)

- Se pueden diseñar sentidos *ficticios* o probar a usar el *gusto* 😅
  - Ej. “Sentido arácnido” de Spider-Man

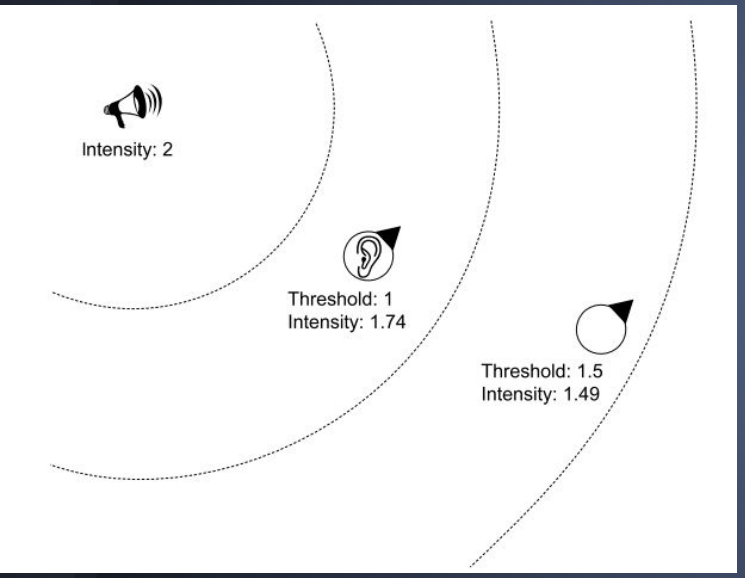
Participación

tiny.cc/IAV

- ¿Cómo funciona la vista en Splinter Cell?
  A. Los agentes *ven* lo que esté en línea de visión
  B. Los agentes sólo *ven* cosas que estén iluminadas
  C. Los agentes *ven* gracias a un sistema de colisiones
  D. Los agentes sólo ven los fotones simulados
- Desarrolla tu respuesta (en texto libre)

Modelo de gestión sensorial regional

- Ejemplo



* Un gestor sensorial basado en regiones esféricas de influencia

```
class RegionalSenseManager:
    # A record in the notification queue, ready to notify the sensor
    # at the correct time.
    class Notification:
        time: int
        sensor: Sensor
        signal: Signal

    # The list of sensors.
    sensors: Sensor[]

    # A queue of notifications waiting to be honored.
    notificationQueue: Notification[]

    # Introduces a signal into the game. This also calculates the
    # notifications that this signal will be needed.
    function addSignal(signal: Signal):
        # Aggregation phase.
        validSensors: Sensor[] = []

        for sensor in sensors:
            # Testing phase.

            # Check the modality first.
            if not sensor.detectsModality(signal.modality):
                continue
```

Ejemplo

* La **intensidad** es la fuerza de la señal por la atenuación elevada a la distancia con el sensor

```
28          # Find the distance of the signal and check range.
29          distance = distance(signal.position, sensor.position)
30          if signal.modality.maximumRange < distance:
31              continue
32
33          # Find the intensity of the signal and check
34          # the threshold.
35          intensity = signal.strength *
36                      pow(signal.modality.attenuation, distance)
37          if intensity < sensor.threshold:
38              continue
39
40          # Perform additional modality specific checks.
41          if not signal.modality.extraChecks(signal, sensor):
42              continue
43
44          # Notification phase.
```

Ejemplo

```
46              # We're going to notify the sensor, work out when.
47              time = getCurrentTime( ) +
48                     distance * signal.modality.inverseTransmissionSpeed
49
50              # Create a notification record and add it to the queue.
51              notification = new Notification( )
52              notification.time = time
53              notification.sensor = sensor
54              notification.signal = signal
55              notificationQueue.add(notification)
56
57          # Send signals, in case the current signal is ready to
58          # notify immediately.
59          sendSignals( )
60
61      # Flush notifications from the queue, up to the current time.
62      function sendSignals( ):
63          # Notification Phase.
64          currentTime: int = getCurrentTime( )
65
66          while notificationQueue:
67              notification: Notification = notificationQueue.peek( )
68
69              # Check if the notification is due.
70              if notification.time < currentTime:
71                  notification.sensor.notify(notification.signal)
72                  notificationQueue.pop( )
73
74              # If we are beyond the current time, then stop
75              # (assuming the queue is sorted).
76              else:
77                  break
```

Ejemplo

- Interfaz para las modalidades
  - Ej. La vista

```
class Modality:
    maximumRange: float
    attenuation: float
    inverseTransmissionSpeed: float

    function extraChecks(signal: Signal, sensor: Sensor) -> bool
```

```
class SightModality:
    function extraChecks(signal: Signal, sensor: Sensor) -> bool
        if not checkSightCone(signal.position,
                              sensor.position,
                              sensor.orientation):
            continue
        if not checkLineOfSight(signal.position,
                                sensor.position):
            continue
```

Ejemplo

- Interfaz para sensores y señales
  - Y estructura auxiliar para la cola de notificaciones

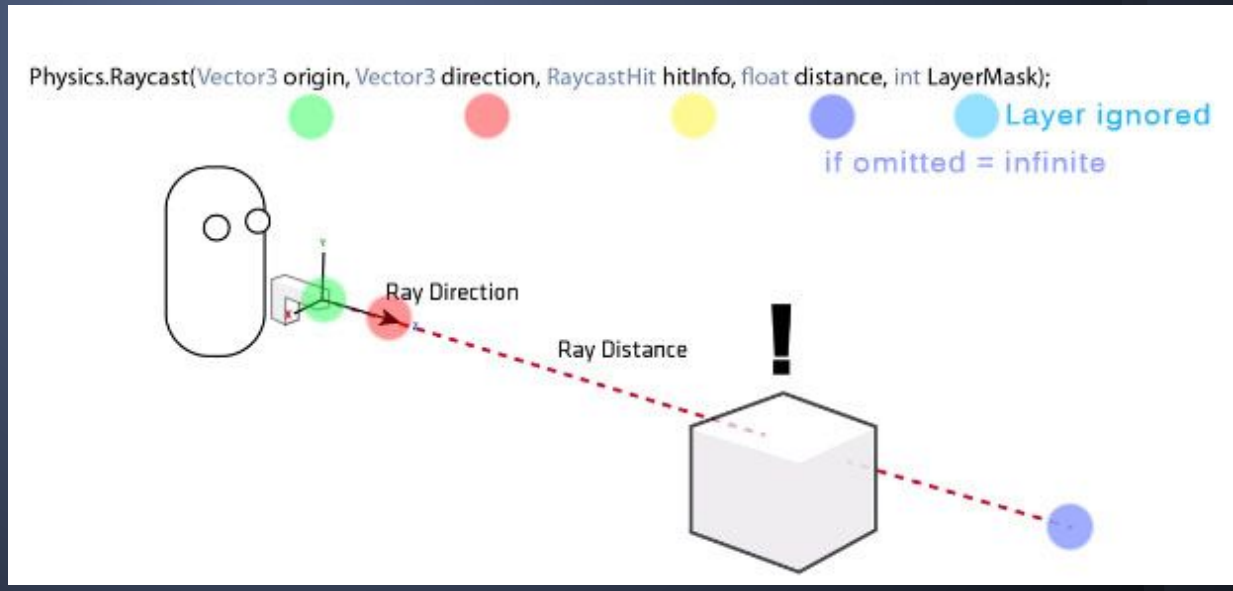
```
1  class Sensor:
2      position: Vector
3      orientation: Quaternion
4
5      function detectsModality(modality: Modality) -> bool
6      function notify(signal: Signal)
```

```
1  class Signal:
2      strength: float
3      position: Vector
4      modality: Modality
```

```
1  class NotificationQueue:
2      function add(notification: Notification)
3      function peek( ) -> Notification
4      function pop( ) -> Notification
```

Ejemplo en Unity

- Es necesario trabajar con API de bajo nivel
- Raycast, que usa RaycastHit
  - Proyección de rayos de la simulación física
- Collider, que usa Collision
  - Detección de colisiones entre cuerpos rígidos (o si sólo es un *trigger* solapamiento entre objetos)



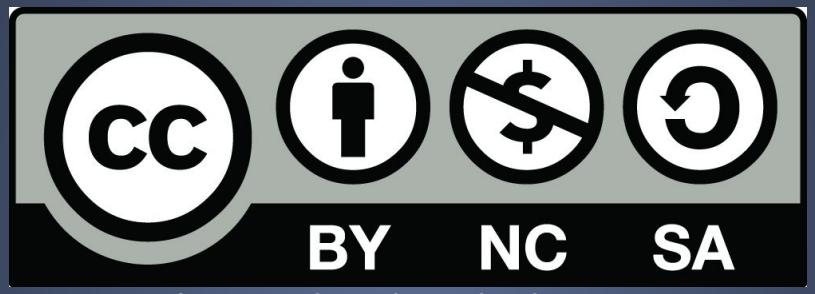
Resumen

- La percepción del jugador es otra cosa, pero conviene conocerse para no defraudar
- La gestión sensorial es un tipo de paso de mensajes que simula los sentidos
  - Vista, con la típica línea o cono de visión
  - Oído, llegando a explicar la velocidad del sonido
  - Tacto, olfato y otros
- El modelo de gestión sensorial regional es una versión algorítmica sencilla que integra lo más importante de cada sentido

Más información

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

Críticas, dudas, sugerencias...



Federico Peinado (2019-2020)

www.federicopeinado.es