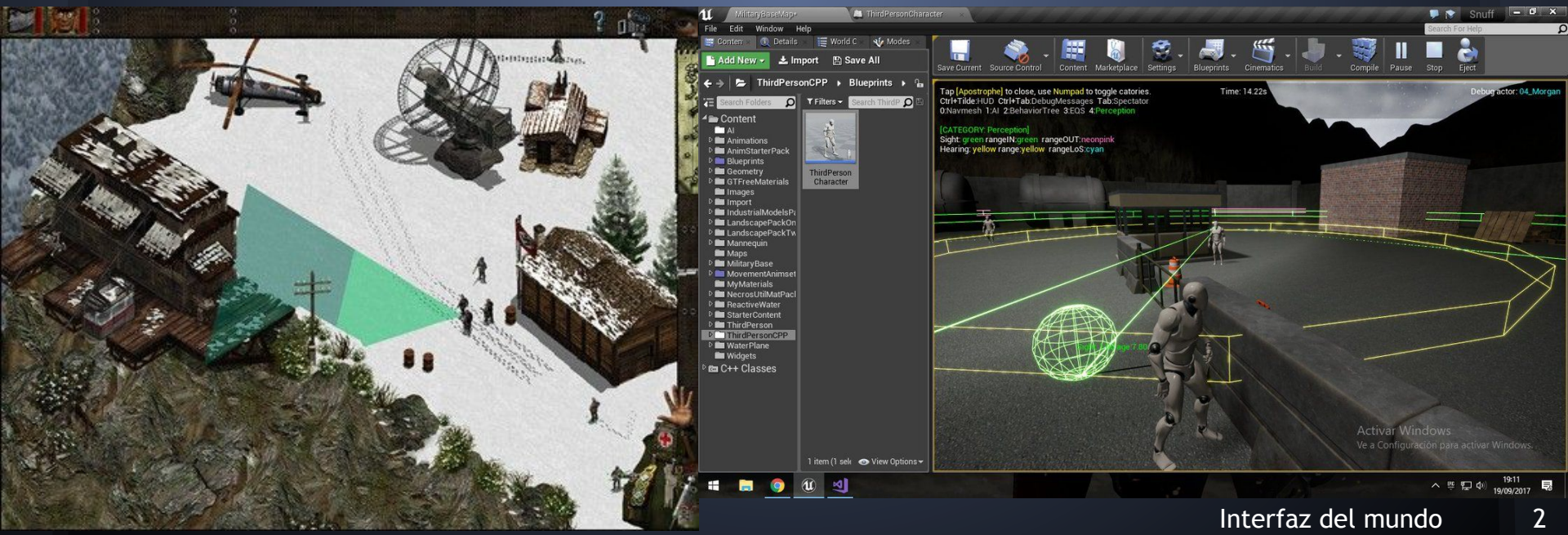
Inteligencia Artificial para Videojuegos

Cuestiones avanzadas

Interfaz del mundo

Motivación

- El uso más evidente de la *interfaz del mundo* es la percepción de los agentes
  - Todo eso que vimos sobre gestión sensorial (poder ver, escuchar, etc.)… *¿cómo se implementa?*



Motivación

- En realidad un buen motor de IA representará el entorno de forma estable y robusta para poder usar todos sus algoritmos de comportamiento
  - No sólamente la *percepción* de los personajes

Puntos clave

- Interfaz del mundo
- Sondeos
  - Estación de sondeo
  - Ejemplos
- Paso de mensajes
  - Gestor de eventos
  - Ejemplos

Interfaz del mundo

WORLD INTERFACING

- Es la segunda de las características del motor de IA que da soporte a los agentes
- Debemos tener en cuenta la ventana de percepción del jugador para ajustar la IA
  - Según la *duración* de los comportamientos del NPC y los *cambios* que se produzcan en ellos
    - Ej. Un NPC compañero no es sólo un 'figurante'
- En general la IA debe ser *eficaz* obteniendo información sobre el entorno
  - Para ello se usan dos técnicas básicas, los sondeos y el paso de mensajes (la *tendencia* actual)

Sondeos

POLLING

- Así se llama al hecho de *buscar activamente* información en el entorno
- Lo fácil es el sondeo directo de cada NPC
  - Pero si hay 100, sólo para ver si la trayectoria de uno colisiona con otro hacen falta 10.000 chequeos (*¡uf!*)

* ¡Esto de comprobar combinaciones de 2 no escala *nada bien*! 😧

Estación de sondeo

POLLING STATION

- Una estación de sondeo permiten *depurar* de forma *centralizada* y *cachear* chequeos de manera que esto se vuelva más escalable
  - Muchos NPCs podrán usar la misma estación

Ejemplos

* En un juego de **Captura la Bandera** por *equipos*, sólo hace falta comprobar si la bandera de un equipo está segura (en su base) un *máximo* de una vez por fotograma

* Sin embargo esa misma estación puede ofrecer *sondeos directos* de visibilidad, por ejemplo

```
class PollingStation:
    # The cache for a boolean property of the game.
    class BoolCache:
        value: bool
        lastUpdated: int

    # The cache value for one topic.
    isFlagSafe: BoolCache[MAX_TEAMS]

    # Updates the cache, when required.
    function updateIsFlagSafe(team: int)
        isFlagSafe[team].value = # ... query game state ...
        isFlagSafe[team].lastUpdated = getFrameNumber()

    # Query the cached topic.
    function getIsFlagAtBase(team: int) -> bool:
        # Check if the topic needs updating.
        if isFlagSafe[team].lastUpdated < getFrameNumber():
            # Only update if the cache is stale.
            updateIsFlagSafe(team)

        # Either way, return its value.
        return isFlagSafe[team].value

    # ... add other polling topics ...

    # A polling topic without a cache.
    function canSee(fromPos: Vector, toPos: Vector) -> bool:
        # ... always query game state ...
        return not raycast(from, to)
```

Ejemplos

ABSTRACT
POLLING STATION

- Estación de sondeo abstracta
  - Una idea algo más *sofisticada* que se acerca a la de los sondeos “plug-and-play” (tareas de sondeo)

```
# Abstract class; the base for any pollable topic.
class PollingTask:
    taskCode: id
    value: any
    lastUpdated: int

    # Checks if the cache is out of date.
    function isStale( ) -> bool:
        return lastUpdated < getFrameNumber( )

    # Update the value in the cache - implement in subclasses.
    function update( )

    # Get the correct value for the polling task.
    function getValue( ) -> any:
        # Update the internal value, if required.
        if isStale( ):
            update( )

        # Return it.
        return value

class AbstractPollingStation:
    # The tasks registered as a hash-table indexed by code.
    tasks: HashTable[id -> PollingTask]

    function registerTask(task: PollingTask):
        tasks[task.code] = task

    function poll(code: id) -> value:
        return tasks[code].getValue( )
```

Participación

tiny.cc/IAV

- ¿Qué infraestructura habéis usado en las prácticas que consideras más reutilizable? ¿Cómo soléis *diseñar e implementar* la conexión entre IA y mundo?
  - Responde directamente en texto libre

Paso de mensajes

MESSAGE PASSING

- Sigue el modelo de eventos, el comportamiento se suscribe a un gestor de eventos que le *notifica* un mensaje cuando se produce determinado evento
  - Útil para cuando muchos se interesan en un evento, si no... sería incluso más lento que los sondeos

Gestor de eventos

EVENT MANAGER

- El *gestor de eventos* facilita la modularidad y da información de depuración muy valiosa
  - *Motor de chequeos* (opcional)
  - *Cola de mensajes*
  - *Registro de observadores*
  - *Despachador de mensajes*

Ejemplos

```
1  class Listener:
2      function notify(event: Event)
```

```
1  class Event:
2      code: id
```

* Luego se pueden especializar los eventos… por ejemplo, todos los guardias de una misma zona se registran en la *alarma* de esa zona

```
1  class CollisionEvent:
2      code: id = 0x4f2ff1438f4a4c99
3      character1: Character
4      character2: Character
5      collisionTime: int
6
7  class SirenEvent:
8      code: id = 0x9c5d7679802e49ae
9      sirenId: id
```

Ejemplos

```
class EventManager:
    # Holds data on one registered listener. The same listener may be
    # registered multiple times.
    class ListenerRegistration:
        interestCode: id
        listener: Listener

    # The list of registered listeners.
    listeners: Listener[]

    # The queue of pending events.
    events: Event[]

    # Check for new events, and add them to the queue.
    function checkForEvents( )

    # Schedule an event to be dispatched as soon as possible.
    function scheduleEvent(event: Event):
        events.push(e)

    # Add a listener to the registry.
    function registerListener(listener: Listener, code: id):
        # Create the registration structure.
        lr = new ListenerRegistration( )
        lr.listener = listener
        lr.code = code

        # And store it.
        listeners.push(lr)

    # Dispatch all pending events.
    function dispatchEvents( ):
        # Loop through all pending events.
```

Ejemplos

```
34          while events:
35              # Get the next event, and pop it from the queue.
36              event = events.pop( )
37
38              # Go through each listener.
39              for listener in listeners:
40                  # Notify if they are interested.
41                  if listener.interestCode == event.code:
42                      listener.notify(event)
43
44      # Call this function to run the manager (from a scheduler for
45      # example).
46      function run( ):
47          checkForEvents( )
48          dispatchEvents( )
```

Difusión de mensajes

- Hay dos *filosofías de diseño* a la hora de implementar el paso de mensajes
  - Difusión masiva, opción flexible con *pocos gestores de eventos con muchos observadores* que deben filtrar lo que les interesa a cada uno

BROADCASTING

  - Difusión selectiva, opción eficiente con *muchos gestores de eventos especializados*

NARROWCASTING

    - Ej. Comunicación interna en un equipo de NPCs
  - Se suele llegar a *un compromiso*, primero difusión masiva y al cierre del juego optimizar con selectiva

Resumen

- La interfaz del mundo debe ser una manera eficaz de obtener información (y de actuar sobre el entorno también)
- Se utilizan sondeos, pudiendo usar una estación de sondeo centralizada, incluso una estación de sondeo abstracta
- Se utiliza paso de mensajes, pudiendo usar muchos o pocos gestores de eventos, usando difusión masiva o selectiva de dichos mensajes

Más información

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

Críticas, dudas, sugerencias...

Federico Peinado (2019-2021)

www.federicopeinado.es