Agentes

Autores: Ismael Sagredo Olivenza

Enunciado

Hay multitud de entornos donde se usa el concepto de Agentes para desarrollar software, pero sus principales aplicaciones tradicionalmente han sido la robótica y los entornos virtuales. Debido a que disponer de robots es algo más complicado a nivel logístico, vamos a utilizar este segundo paradigma como base para realizar nuestra primera práctica de agentes.

En concreto vamos a desarrollar un agente que controle un NPC (Non Plaeyable Character) en un videojuego. Como todo en todos los demás grados usan el Pacman y debéis estar un poco hartos del mismo entorno, en esta práctica hemos innovado (aunque tampoco mucho) y vamos a utilizar otro videojuego. Eso sí, casi igual de viejuno que Pacman y la que probablemente no habréis jugado nunca :)

El entorno donde va a interactuar vuestro agente es el videojuego **Battle City** que es un videojuego de tanques producido y publicado por Namco como una adaptación del clásico arcade Tank Battalion que es el videojuego original.

En este juego, el jugador debe defender una base (simbolizada por un águila) mientras soporta diferentes oleadas de tanques enemigos. El jugo va avanzando por diferentes fases donde hay también diferentes tipos de elementos en el terreno como paredes de ladrillo, paredes metálicas, agua, hierba, etc. Cada tipo de suelo tiene ciertas implicaciones, por ejemplo, el agua no permite pasar a los tanques, pero si los disparos, los bloques de ladrillo son destruibles por los disparos de los tanques, mientras que los metálicos son invulnerables, etc.

Podéis jugar al juego en el siguiente enlace

https://www.retrogames.cz/play_014-NES.php

O en vuestro emulador de referencia de Nes.

En esta ocasión disponemos de una remasterización 3D del juego, donde se moverá nuestro agente. El agente que vamos a programar es un agente de uno de los tanques enemigos y el entorno de pruebas es un subconjunto de las reglas de Battle City muy concretas que definimos a continuación:

- Existen bloques de ladrillos que son rompibles. Los bloques de ladrillos suelen estar agrupados de 4 en 4 o de 2 en 2 pero son rompibles en bloques de 1x1. Por lo que podemos pensar que, aunque parece que el mundo tiene 13x13 casillas, en verdad a nivel lógico tiene 26X26 casillas, es decir 676 casillas en total.
- Los agentes y el jugador pueden estar en cualquier punto del mapa (esto no era así en el juego original debido a las limitaciones de cálculo en coma flotante de la Nes y de las placas arcade primitivas) y no se mueven

Figure 1: Battle City Portada

estrictamente por esas 676 casillas pudiendo estar entre dos de ellas. Esto puede dificultar el movimiento de los agentes en pasillos estrechos.
- Los agentes ocupan algo menos de 2x2 casillas para facilitar los giros y el movimiento entre las casillas (algo que nuevamente no ocurría en el juego original donde estrictamente ocupaba 2x2 casillas)
- Los agentes enemigos tienen 2 puntos de vida y el agente jugador 1 punto de vida.
- El águila está representada por una antena parabólica que simboliza la CommandCenter que debemos defender. En el juego original estaba defendida por una pared más fina, que sólo soportaba 2 toques (los mismos que soporta ahora, pero con una pared visualmente más gruesa)
- Las balas que dispara el tanque si chocan entre si se destruyen.
- Los bloques metálicos son indestructibles y pueden servirte para parapetarte tras ellos de forma segura.
- En el juego original había potenciadores (power ups) pero en esta versión del juego no están introducidos.
- En el juego original, los agentes enemigos aparecían en oleadas y sabias cuantos tanques enemigos iban a aparecer (había un indicativo de los tanques a la derecha del terreno de combate), en esta versión sólo existen dos agentes, un agente Random programado dentro del juego y el agente que vais a programar vosotros (a parte del jugador)
- No puedes disparar mientras no se destruya la bala que has disparado

Vuestro agente aparecerá siempre en la esquina superior izquierda (representado en el diseño del nivel con una miniatura del Agente 47 del videojuego Hitman caricaturizado como un personaje de MegaMan) y el jugador aparecerá siempre junto a la Command Center. El tanque rojo de la figura es el agente aleatorio. El Agente aleatorio es el Base Line, es decir, el agente más simple que podemos crear, por lo que os puede servir de referencia para ver lo buenos que son vuestros agentes... ¿Os vencera el Agente Random? Esperemos que no :)

Vuestro agente se conectará con el juego a traves de un programa en Python. La arquitectura de comunicación está implementada, vosotros sólo os teneis que encargar de implementar el comportamiento del agente.

NOTA: Esta es la primera versión del juego y del a práctica por lo que os pido cierta comprensión por posibles bugs que haya. La iré mejorando poco a poco y esperemos que en el futuro tengamos un mayor control de los errores, por depronto, si pasa cualquier casa rara, reiniciar el juego es la mejor opción.

El juego espera a que se conecte el agente externo antes de empezar. Una vez conectado aparece la palabra Ready y los agentes y el jugador comienzan a moverse por el entorno.

En python teneis la clase BaseAgent que implementa un agente Random. Podeis partir de ella para desarrollar vuestro agente o bien heredando y redefiniendo los métodos que os interese o programando el agente en ella.

```
class BaseAgent:
```

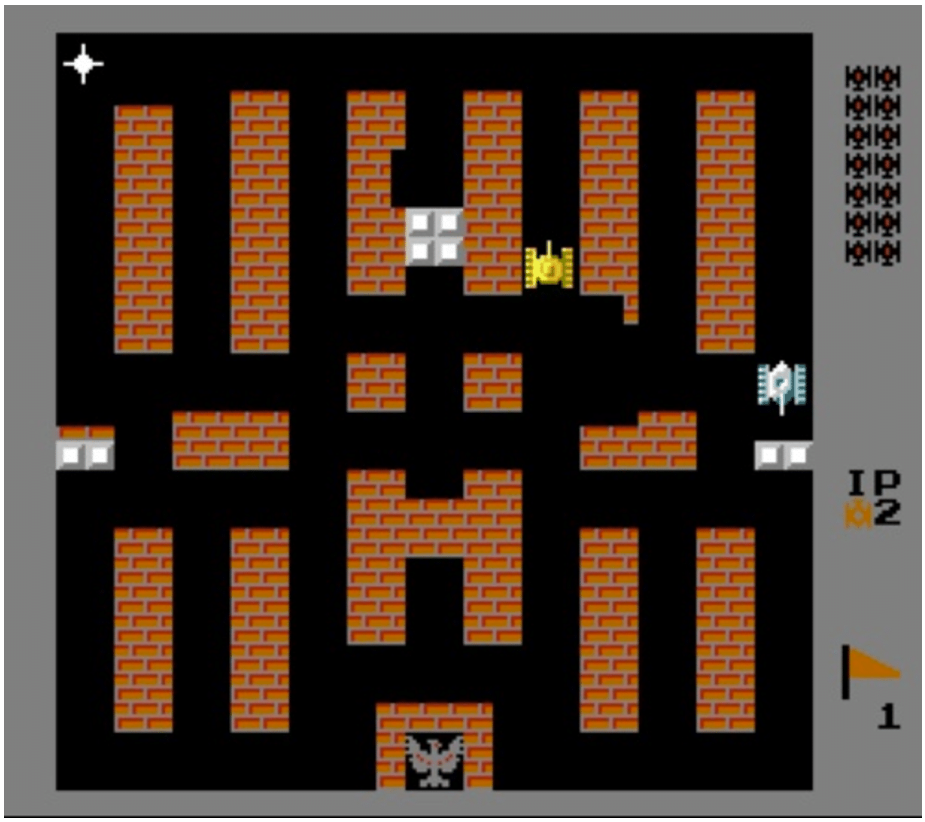

Figure 2: Battle City Captura de pantalla

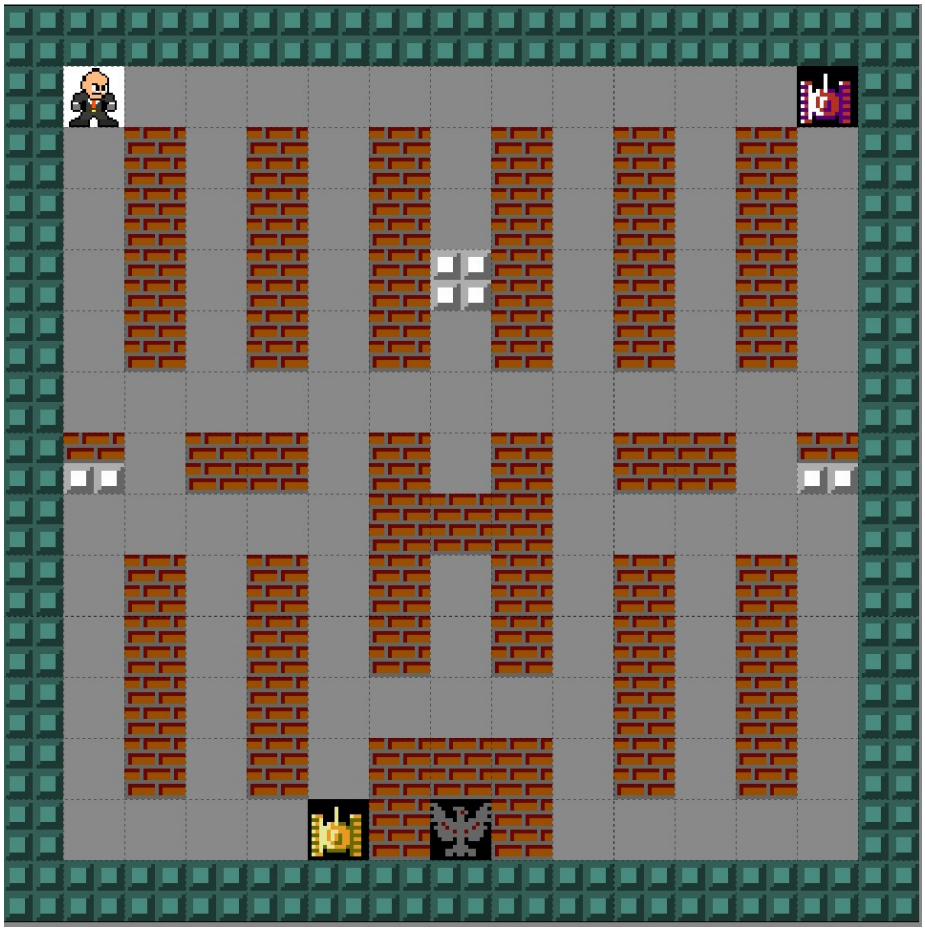

Figure 3: Battle City diseño de la fase

```
    def __init__(self, id, name):
        self.id = id
        self.name = name

    #Devuelve el nombre del agente
    def Name(self):
        return self.name
    #Devuelve el id del agente
    def Id(self):
        return self.id
    #Metodo que se llama al iniciar el agente. No devuelve nada y sirve para contruir el ag
    def Start(self):
        print("Inicio del agente ")

    #Metodo que se llama en cada actualización del agente, y se proporciona le vector de pe
    #Devuelve la acción u el disparo si o no
    def Update(self, perception):
        print("Toma de decisiones del agente")
        print(perception)
        action = random.randint(0,4)
        return action, True

    #Metodo que se llama al finalizar el agente, se pasa el estado de terminacion
    def End(self, win):
        print("Agente finalizado")
        print("Victoria ",win)
```

Los métodos estan ocmentados y son auto explicativos pero por si acaso os los describo brevemente.

- Name(): devuelve el nombre del agente que le hayais puesto.
- Id(): Devuelve el identificador, este identificador es el que identifica al agente en el juego. Por defecto como solo tenemos un agente con "1" es suficiente.
- Start(): este método es invocado una sola vez cuando se inicia el mundo (El agente ya está conectado y ha sido aceptado por el entorno)
- Update(perception): Este método es invocado cada 1 segundo por el juego para actualizar el estdo de la percepción del mismo. Tenéis 1 segundo de margen para contestar o si no el agente volverá a mandar otro mensaje y se producirá un error. En la percepción teneis una lista con las cosas que ve y sabe el agente en formato float, debeis tratarla como más os interese dentro del método update. Está en formato float para que puedea ser proecesada por algoritmos de Machine Learning aunque en vuestro caso no sea estrictamente necesario. Este método devuelve dos valores, un entero con la acción que quieres realizar y un booleano que indica si debemos disparar o no.
- End(): se os invoca cuando el juego acaba que es por dos motivos, o bien

que vuestro agente ha sido destruido (en ese caso win == False) o si la command center o el jugador han sido destruidos (en ese caso win == True). No tiene porqué ser el agente el que acabe con la Command Center o con el jugador, podría ser el propio jugador o el otro agente, es indiferente.

¿Cómo se lanza el agente? En Main.py tenéis un ejemplo, simplemente se crea una instancia del agente (tendréis que cambiarl opor el vuestro si os creais un agente que herede de BaseAgent) y pasarlo como argumento a

"'Python

agentLoop(agent,False)

"'

El booleano si está a True nos da información del estado de las comunicaciones y conexiones con el entorno. E n principio no es necesario tocar nada de la comunicación entre el entorno y el agente, pero en caso de que necesiteis mirar algo, tenéis el código disponible.

El array de percepción tiene el siguiente esquema donde los números indican la posición del vector y el nombre el campo que representa a nivel lógico:

NEIGHBORHOOD_UP = 0, NEIGHBORHOOD_DOWN = 1, NEIGHBORHOOD_RIGHT = 2, NEIGHBORHOOD_LEFT = 3, NEIGHBORHOOD_DIST_UP = 4, NEIGHBORHOOD_DIST_DOWN = 5, NEIGHBORHOOD_DIST_RIGHT = 6, NEIGHBORHOOD_DIST_LEFT = 7, PLAYER_X = 8, PLAYER_Y = 9, COMMAND_CENTER_X = 10, COMMAND_CENTER_Y = 11, AGENT_X = 12, AGENT_Y = 13, CAN_FIRE = 14, HEALTH = 15

NEIGHBORHOOD_DIST es la distancia del objeto detectado en NEIGHBORHOOD. Los objetos detectados son los siguientes:

NOTHING = 0, UNBREAKABLE = 1, BRICK = 2, COMMAND_CENTER = 3, PLAYER = 4, SHELL = 5, OTHER = 6

Agente a implementar

El comportamiento del agente que queremos implementar es el siguiente: Nuestro agente debe disparar al jugador si lo tiene a tiro. Lo mismo ocurre con la Command Center que, aunque no está inicialmente visible por el agente (la defiende una pared de ladrillos) el agente conoce en todo momento donde está por lo que puede saber si puede dispararla o no. Si ve una bala que va hacia él, intentará dispararla ya que es más rápido que intentar esquivarla. Para disparar a una bala es muy probable que se requieran varias acciones ya que, por ejemplo, una de ellas podría ser girar hacia la dirección de la bala y otra disparar, pero es posible que no puedas disparar en ese momento. En este caso y si la distancia de la bala es muy grande, lo más interesante sería huir de la bala. Si encuentra un enemigo, el agente también debería dispararle. En cuanto a los bloques, queda a

vuestro criterio si creéis que es mejor dispararlos o no, así como cualquier otra decisión que os encontréis que debéis tomar.

Acciones disponibles:

NOTHING = 0, MOVE_UP = 1, MOVE_DOWN = 2, MOVE_RIGHT = 3, MOVE_LEFT = 4

No se puede mover en diagonal, si se intenta el agente no se moverá.

Disparo si o no. Se puede disparar y moverse simultáneamente. Si se intenta disparar, pero el agente no puede disparar ignorará la orden.

Un agente será mejor que otro si es capaz de vencer en menor tiempo.

Se pide:

En un pdf, jupiter notebook o donde os venga mejor hacerlo:

1. Descripción detallada del entorno usando REAS
2. Qué tipo de agente es y explícalo
3. Diseño preliminar del agente (la próxima clase lo chequearemos)
4. Diseño final del agente

En python Implementación del agente.

Test de prueba. Vamos a realizar 3 test que podéis recrear vosotros mismos en vuestras pruebas.

1. Que el agente venza sin que el jugador haga nada. El mejor agente deberá resolver esto en el menor tiempo posible.
2. Que el agente venza sin que el jugador dispare. El mejor agente deberá resolver esto en el menor tiempo posible.
3. Que el agente tenga una inteligencia aparente compitiendo contra el jugador. Este último punto es más subjetivo, pero debería ser lo suficientemente inteligente como para que matarlo no sea algo muy simple. Obviamente no esperamos conseguir un agente que venza al humano.

Cosas a valorar

- Diseño del agente: 15% de la nota
- Que el agente cumpla con las dos test. Red line para superar la práctica y cuenta un 40% de la nota.
- Que en el tercer test su comportamiento sea parentemente inteligente: 30% de la nota.
- Código razonablemente bien desarrollado con buenas prácticas de programación, eficiente y minimamente robusto frente a fallos: 15%

NOTA: si se detecta un bug que impida el desarrollo del agente, enviarme un correo a isagredo@ucm.es