Mario AI

Tutorial 1

Grado en Ingeniería Informática

Aprendizaje Automático

Aitor Alonso Núñez NIA 100346169 Gr. 83
Daneil Gak Anagrov NIA 100318133 Gr. 83

Aitor Alonso Núñez NIA 100346169 Gr. 83
Daneil Gak Anagrov NIA 100318133 Gr. 83

uc3m | Universidad Carlos III de Madrid

Índice

Puesta en marcha

¿Qué elementos se distinguen en interfaz gráfica del juego?

En la esquina superior izquierda podemos visualizar la siguiente información:

- **Difficulty:** la dificultad del nivel o escenario.
- **Seed:** la semilla utilizada para generar el nivel o escenario
- **Type:** tipo (actualmente desconocemos a qué se refiere)
- **Lenght:** posición en el eje X de Mario
- **Height:** posición en el eje Y de Mario
- **Icono moneda:** número de monedas recogidas por Mario
- **Icono champiñón:** número de champiñones recogidos por Mario
- **Icono flor:** número de flores recogidas por Mario
- **Pressed keys:** teclas que están siendo pulsadas en teclado (se indican a la derecha del texto)

En la esquina superior derecha encontramos:

- **All kills:** número de muertes de enemigos
- **By fire:** número de muertes de enemigos causadas por fuego
- **By shell:** número de muertes de enemigos causadas por caparazones
- **By stomp:** número de muertes de enemigos causadas por pisotones
- **Time:** tiempo restante (en segundos) para completar el nivel
- **FPS:** fotogramas por segundo de la interfaz

Por último, en la parte inferior de la pantalla encontramos **intermediate reward**, es decir, la recompensa media que actualmente posee el agente. también podemos ver una escala que indica la posición de Mario con una M respecto al inicio y el final del nivel.

¿Qué cambios se producen al usar los distintos argumentos y atajos de teclado del Anexo 1?

Tras probar todos los argumentos que aparecen en el Anexo 1:

- **nombre_agente:** indica qué aregnte se debe cargar en la ejecución. Hemos probado los dos que se mencionan en el Anexo 1:
  - **human.HumanAgent:** es un agente controlado por teclado por un humano

- **BaselineAgent:** es una agente de IA muy básico. Ejecuta acciones aleatorias con tendencia a moverse hacia la derecha.

- **-ls [número]:** permite introducir manualmente las semilla (seed) que se utiliza para generar el nivel
- **-ld [número]:** permite introducir manualmente la dificultad (difficulty) el nivel
- **-fps [número]:** permite introducir manualmente el número de fotogramas por segundo
- **-vis off:** ejecuta el agente sin interfaz gráfica, lo que es mucho más rápido
- **-i on:** convierte a Mario en invencible
- **-lg off:** evita que se generen niveles con pozos
- **-le [tipo_enemigo]:** el nivel generado solo tendrá enemigos del tipo indicado en tipo_enemigo

En cuanto a los atajos de teclado, con las flechas, la tecla A y la tecla S podemos controlar a Mario. Al pulsarlas aparecerán indicadas junto a la sección *Pressed Keys* que indicamos en la pregunta anterior. El resto de teclas modifican la información que se muestra en la interfaz o la forma en que se muestra de la siguiente manera:

- **Tecla Z:** aumenta el tamaño de la ventana para facilitar la visualzación de la interfaz.
- **Tecla 8:** elimina el límite de fotogramas por segundo.
- **Tecla G:** muestra una matriz alrededor de Mario que a su vez muestra valores para los enemigos y elementos que caen dentro, así como para Mario.
- **Tecla L:** muestra etiquetas sobre todos los elementos (y en la esquina inferior izquierda de la pantalla) con información más exacta de las coordenadas X e Y de cada elemento visible en la interfaz. Estas coordenadas no se corresponden con las mostradas en el apartado *Height* y *Lenght* de la esquina superior izquierda de la interfaz.
- **Tecla C:** mantiene el foco sobre Mario en el centro de la interfaz, independientemente de su posición y sus acciones.
- **Tecla F:** permite a Mario ignorar la gravedad y “volar” por el mapa. Se controla arriba, abajo, izquierda, derecha con las flechas del teclado.

Explorando la información disponible

¿Qué es un tick de juego?

La variable `tick` es un atributo del objeto agente representado mediante una variable entera que inicialmente vale cero y se actualiza en +1 cada vez que se ejecuta el método `integrateObservation()`. Puesto que este método se ejecuta en cada instante de tiempo para obtener información del entorno, podemos concluir que un *tick* de juego representa el mínimo instante de tiempo en el juego.

¿Para qué le sirven al agente los métodos `getAction()`, `integrateObservation()` y el parámetro `environment`?

El método `getAction()` devuelve un array de seis posiciones booleanas que indica las acciones (teclas pulsadas) llevadas a cabo por el agente.

El método `integrateObservation()` permite al agente, como ya indicamos en la pergunta anterior, obtener información del entorno en cada tick.

El parámetro `environment` es un objeto de tipo `Environment` que es recibido por el método `integrateObservation()`. Este objeto posee atributos y métodos que permiten extraer información sobre el entorno (estoy icnluye Mario, enemigos, estadísticas, etc), por lo que es necesario para el correcto funcionamiento de este método.

¿Qué información contienen las matrices devueltas por los métodos `getLevelSceneObservationZ()`, `getEnemiesObservationZ()` y `getMergedObservationZZ()`?

La matriz devuelta por el método `getLevelSceneObservationZ()` tiene tamaño 19x19 y contiene información sobre los elementos en la escena del nivel. Se puede elegir entre tres niveles distintos de detalle.

La matriz devuelta por el método `getEnemiesObservationZ()` tiene también tamaño 19x19 y contiene información sobre los enemigos en la escena del nivel. Al igual que en el caso anterior, se puede elegir entre tres niveles distintos de detalle.

La matriz devuelta por el método `getEnemiesObservationZ()` tiene asimismo tamaño 19x19 y contiene información combinada de las dos matrices devueltas por los métodos anteriores. También permite filtrar entre tres niveles distintos de detalle.

¿Cuál es la posición que ocupa Mario en estas matrices?

Mario siempre ocupa la posición 9,9 en estas matrices de domensión 19x19. Es decir, está situado en el centro de las mismas.

A las funciones indicadas en los puntos anteriores se les puede pasar un parámetro para indicar el nivel de detalle de la información devuelta (consultar Anexo 2). ¿Qué implica cambiar el nivel de detalle?

Como ya hemos indicado, permite filtrar mucho más la información contenida en las matrices devueltas por estos métodos. Existen tres posibles niveles de filtrado. 0,1,2 que solicitan mayor (0) o menor (2) detalle de la información recuperada.

¿Cuál es el número que codifica las monedas, los obstáculos y los enemigos en el nivel de detalle 1?

En el nivel de detalle 1, las monedas esán codificadas con el valor 2, los obstáculos con -60 o -62 según su tipo, y los enemigos con valor 80.

Creando el fichero de entrenamiento

Para la creación de los ejemplos de entrenamiento hemos definido la clase `TrainingFile.java`, que hemos situado en `ch.idsia.tools`. Esta clase tiene su constructor preparado para también poder leer el fichero con los ejemplos de entrenamiento,ya que es lógico que estamos guardando ejemplos de entrenamiento en un fichero para leerlos posteriormente y aprender de ellos, pero sin embargo de momento solo está implementada la lógica para obtener y escribir los ejemplos.

En los ficheros de los nuevos agentes `T1HumanAgent.java` y `T1BotAgent.java` hemos añadido un nuevo atributo objeto de tipo `TrainingFile`, y para el almacenamiento de los ejemplos en el fichero ejecutamos el método `writeExample()` sobre este objeto en cada llamada al método `integrateObservation()`.

El contenido de nuestro fichero de ejemplos de entrenamiento contiene, en este orden y para cada tick (en cada línea):

- 1. Monedas recogidas hace 5 ticks
- 2. Muertes de enemigos hace 5 ticks
- 3. Distancia (posición X celdas)
- 4. Distancia (posición X física)
- 5. Flores recogidas
- 6. Muertes de enemigos por fuego
- 7. Muertes de enemigos por caparazón
- 8. Muertes de enemigos por pisotón
- 9. Muertes de enemigos totales
- 10. Modo de Mario
- 11. Estado de Mario
- 12. Champiñones recogidos
- 13. Monedas recogidas
- 14. Tiempo restante
- 15. Tiempo gastado
- 16. Bloques ocultos encontrados
- 17-378. Valores de getMergedObservationZZ(0, 0) (19x19 valores al máximo nivel de detalle)

Programación de un agente procedural simple

El comportamiento de este agente procedural simple, definido en la clase `T1BotAgent.java`, se fundamenta en avanzar hacia la derecha y saltar cuadno tiene un enemigo u obstáculo cerca. Esto se ha conseguido modificando el contenido de los métodos `integrateObservation()` y `getAction()` de la siguiente forma.

En el método `integrateObservation()` se ha extraído, para cada *tick*, la información necesaria tanto del entorno como del propio Mario para decidir qué acciones llevar a cabo. Esto es únicamente cuándo saltar, cuándo mantener el salto y cuándo dejar de saltar; ya que la acción de moverse hacia la derecha es constante. En el proceso se utilizan variables declaradas como `protected` dentro de la clase `BasicMarioAIAgent.java`, por lo que **es necesario** la existencia de esta clase en el mismo paquete, la cual **ya venía proporcionada** y **no** hemos modificado.

Los cambios en el método `getAction()` son, aunque más numerosos, igual de fáciles de comprender puesto que el código está completamente comentado. Como hemos indicado, marcamos constantemente que Mario se mueva hacia la derecha, manteniendo siempre activada la tecla correspondiente. Ahora, en caso de que el salto esté permitido, comprobamos si tenemos un obstáculo cerca, en cuyo caso procedemos a activar la tecla de salto.

Si por el contrario el salto está bloqueado, lo más probable es que se deba a que o bien estamos en el aire saltando todavía, o bien ya estamos en tierra pero no hemos desactivado la tecla de salto desde que la pulsamos al comienzo del salto. Para diferenciar estos casos, comprobamos si hemos sobrepasado ya el obstáculo, o si estamos ya en tierra, momento en el que desactivamos la tecla de salto y así nos preparamos para el próximo salto que debamos dar.