Practica 2. Control de un modelo de GAMA para el aprendizaje

Agentes y Sistemas Multiagente,
Master en Inteligencia Artificial,
Departamento de Ciencia de la Computación e Inteligencia Artificial.
Universidad de Alicante. Curso 23/24

Fidel Aznar Gregori.

Control de un modelo de GAMA para el aprendizaje

Imaginemos que tenemos un modelo GAMA, un entorno que nos permite simular una cuestión particular. Sin embargo no sabemos cómo desarrollar un controlador de un agente para que se desenvuelva de manera correcta en ese entorno.

Aunque existen varias maneras de afrontar el problema mediante aprendizaje una de las más habituales y más potentes es el RL: que el agente aprenda mientras interactúa con otros agentes y el entorno.

En esta práctica aprenderemos como controlar un entorno de GAMA desde otro lenguaje de programación, concretamente python. Además se propondrá añadir el diseño de algoritmos de aprendizaje para manejar un enjambre de agentes en una tarea de *wandering* en robótica de enjambre.

Tecnología

- Gama: se utilizará GAMA para el modelado del entorno
- Gama Headless: una vez probado el modelo anterior no necesitaremos su entorno gráfico y requerimos controlarlo desde python. Para ello utilizaremos el modo *headless* de GAMA y su API de python.
- Python y librerías necesarias

Entorno

Deseamos crear un entorno que represente un tatami en el cual se mueven una serie de robots. El entorno debe permitir:

- Visualizar los agentes (usa círculos). Inicialmente serán rojos. Si hay colisión se deben pintar de negro
- Parar la simulación si todos los agentes han colisionado o han llegado a 5000 pasos de simulación
- Determinar si dos agentes han colisionado
- Posibilitar cambiar la velocidad translacional (v_t) y rotacional (v_r) a los agentes en cada paso de la simulación. Asumiremos que tienen un modelo de mov. diferencial y por tanto en cada iteración controlamos su giro y su avance. Para más información, no necesaria para desarrollar la práctica, mirar https://en.wikipedia.org/wiki/Differential_wheeled_robot.
- Para cada agente conocer la posición de su vecino más cercano

Prueba que el entorno funciona de manera correcta en GAMA disponiendo una velocidad aleatoria en cada paso de simulación para cada agente.

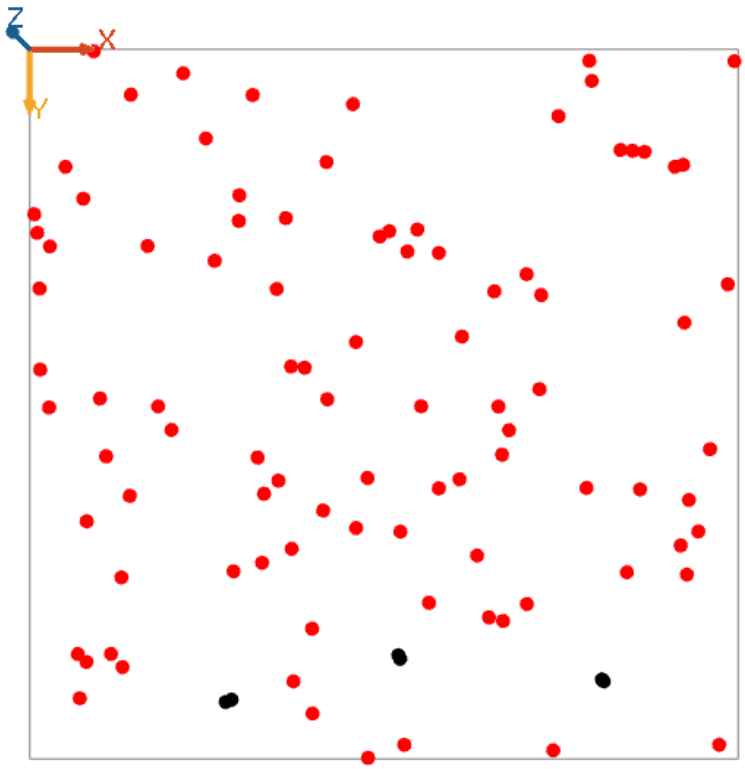

Figure 1: Ejemplo del entorno de enjambre esperado en GAMA. Los agentes que han colisionado se marcan en negro.

Optimización del entorno

Nuestro entorno se va a ejecutar cientos de veces para posibilitar su uso en el aprendizaje por refuerzo. ¿Puedes optimizar las llamadas a GAMA de la siguiente forma?

- ¿Puedes desarrollar un método global, llamado `list<point> get_agents_pos` que devuelva todas las posiciones de los agentes?
- ¿Y un método global `action set_agents_vel(matrix<float> vt, matrix<float> vr)` que reciba dos vectores de flotantes para disponer a la vez la velocidad de todos los agentes?

Definición del agente

Vamos a modelar el problema como un **Markov Decision Process** (MDP) y por tanto debemos definir:

- El espacio de observación $s_t \in S$
- El espacio de acción $a_t \in A$
- $P(a|s)$: Definición de un modelo neuronal compatible
- Reward $r_t = r(s_t, a_t, s_{t+1})$

Nuestro espacio de observaciones será un vector hacia el vecino más cercano del agente. Asumimos que solo puede percibir a su vecino más cercano independientemente de su distancia.

Nuestro espacio de acciones por defecto será la velocidad rotacional (v_r) y translacional (v_t) del agente. De todas maneras asumiremos que los agentes se mueven a una velocidad translacional constante y por tanto solo tendremos una acción, la velocidad rotacional.

- Propón de manera teórica un modelo neuronal compatible con s y a para ser usado en el aprendizaje por refuerzo. NOTA: este modelo solo será necesario usarlo para los que elijan la tarea C) pero todos debéis proponerlo de manera teórica.
- Propón una función de reward $r(s_t, a_t, s_{t+1})$ que guíe al algoritmo a aprendizaje por refuerzo a obtener un controlador coherente.

Modo headless de GAMA

El funcionamiento es sencillo, desde python se controla gama y se pueden ejecutar expresiones en la misma y obtener datos. Si hacemos pocas llamadas es bastante eficiente. La documentación oficial está aquí: https://gama-platform.org/wiki/HeadlessServer

1. Revisa tu modelo de gama, y verifica que tienes acciones *getters* para los sensores y *setters* para los actuadores.
2. `cd /Applications/Gama.app/Contents/headless` (o ruta equivalente en el OS que se desee). Asegurarse en entornos unix que el

script `gama-headless.sh` tiene permisos de ejecución con `chmod u+x gama-headless.sh`
3. Ejecutar gama headless en el puerto deseado : `./gama-headless.sh -socket 6868`
4. Instalar gama-client. **Recomendable hacerlo desde un entorno virtual de conda** (requiere tener instalado conda: `conda create --name gama python=3.9` y `conda activate gama`). Después `pip install gama-client`
5. Desde el cliente se controla gama de manera sencilla.
6. Gasta la consola de gamaML para probar la validez del comando introducido desde python.

Ten en cuenta que la comunicación entre python y GAMA manda paquetes de respuesta con contenido `string` por defecto. Si gastas el API de python y por ejemplo comunicas un objeto `matrix` de gama el string tiene ya el formato esperado por python para que puedas hacer algo como esto:

```
cmd = "ask world { do get_agents_pos;}"
res = self.event_loop.run_until_complete(self.ghl.exec_remote(cmd))
gama_obs = eval(res['content'])
```

Fíjate que aunque el contenido de la respuesta del comando de la segunda línea es un string (`res['content']`) podemos obtener la lista de posiciones de objetos como una lista de python interpretando el string mediante el comando de python `eval`. De esta manera hora `gama_obs` es una lista.

Uso del modelo de GAMA, control externo y aprendizaje

Ahora puedes usar el modelo de gama para alguna de las siguientes tareas. Cada una tiene un nivel de dificultad asociado y una nota máxima que se puede alcanzar desarrollándolo. Todos los modelos deben cumplir la definición de agente del punto anterior, en cuanto a sus entradas, salidas y función de reward. Elige sólo una opción.

- a) **Control remoto** [**sencillo - máx. 6 puntos**].
  Crea un script en Python que controle los agentes para que eviten colisionar entre sí.
- b) **Optimización de un modelo en Gama** [**medio - máx. 8 puntos**].
  Diseña una estrategia en gama para que los agentes eviten las colisionar entre sí. Esa estrategia debe tener algún parámetro a ajustar (por ejemplo la distancia a partir de la cual debe girar el agente) que debes optimizar. Usa métodos de optimización, análisis y calibración en gama para saber que valores de tus parámetros son los óptimos y qué variaciones producen resultados interesantes (https://gama-platform.org/wiki/ExplorationMethods)

- c) **Aprendizaje de un comportamiento [avanzado - máx. 10 puntos]**.
  Diseña un entorno para aprendizaje por refuerzo con el API de gymnasium (https://github.com/Farama-Foundation/Gymnasium) o Gym (https://github.com/openai/gym) integrado con Gama. Utiliza el entorno con aprendizaje por refuerzo para aprender un modelo neuronal que controle los agentes. Puedes utilizar aproximaciones evolutivas como la librería `estools` (https://blog.otoro.net/2017/11/12/evolving-stable-strategies/), CMAES (https://pypi.org/project/cmaes/), librerías de aprendizaje por refuerzo como (https://www.ray.io/rllib) o implementar el algoritmo de RL que consideres.

Para todas las tareas anteriores:

- ¿Cómo funciona función de reward que diseñaste?, ¿Hay casos degenerados? Documenta convenientemente este apartado
- Obtén las métricas del funcionamiento y muestra una representación gráfica de cómo se desplazan los agentes durante un tiempo determinado.
- ¿Es posible obtener emergencia de alguna conducta macroscópica ajustando parámetros de tu modelo?
- ¿Cómo afecta v_t a la conducta? Si modificas esta velocidad fija, ¿Es robusto el comportamiento? Presenta los resultados convenientemente.

Entrega

Todo el código desarrollado y la memoria de la práctica se entregarán en el portfolio de prácticas de la asignatura.