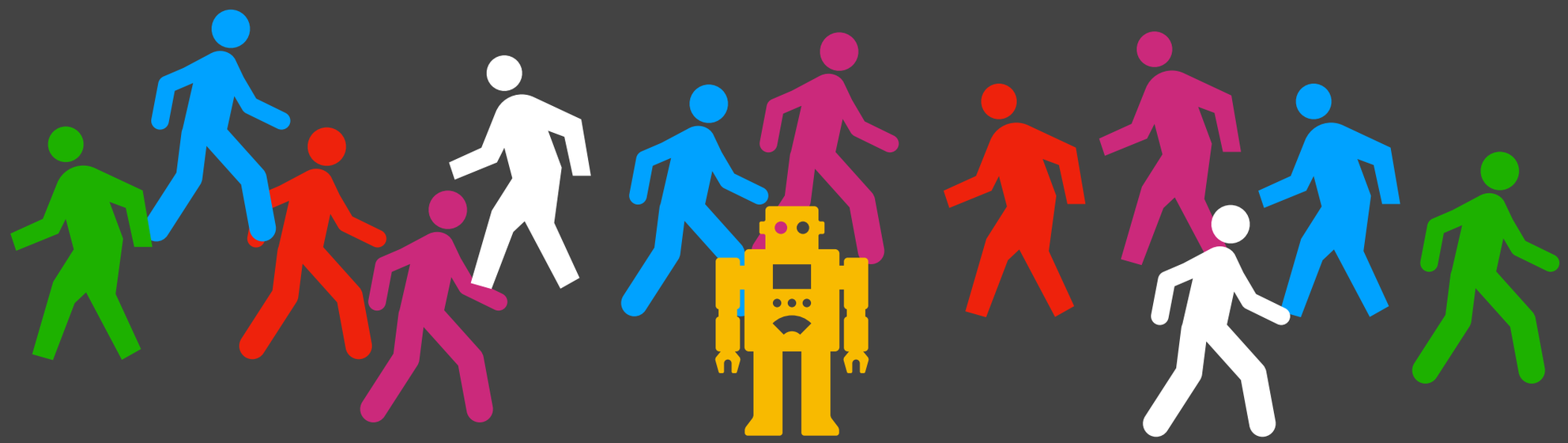
Navegación Autónoma en Simulación de Multitudes

Sergio Miguel Fernández Martínez



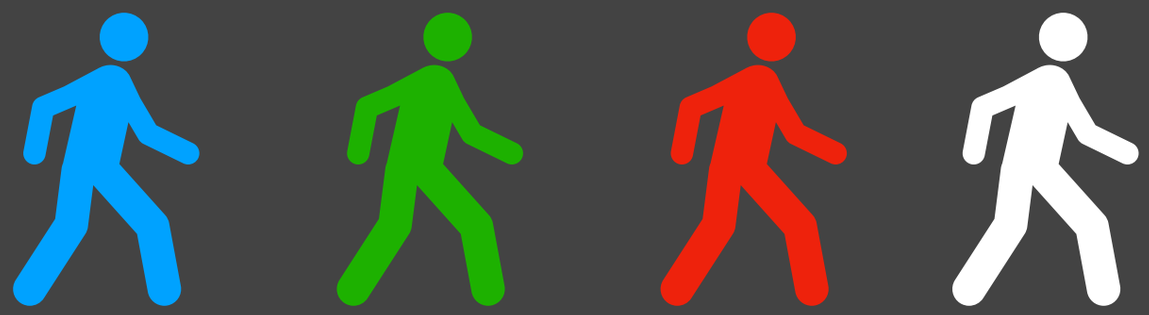
Modelación y simulación computacional basada en agentes: Proyecto Final

Objetivo

- Crear un ambiente basado en el modelo de fuerzas sociales.

- Entrenar a un agente autónomo a alcanzar un objetivo dentro del ambiente diseñando usando el algoritmo de *Deep Deterministic Policy Gradient.*

Modelo de Fuerzas Sociales



- Es un modelo que describe el movimiento de un peatón α dentro de una multitud.

- Se basa en la segunda ley de Newton y se describe de acuerdo a 4 fuerzas.

$$\mathbf{F} = m_\alpha f_\alpha$$

$$\mathbf{f}_\alpha(t) = \mathbf{f}_\alpha^0 + \sum_\beta \mathbf{f}_{\alpha\beta} + \sum_B \mathbf{f}_{\alpha B} + \sum_i \mathbf{f}_{\alpha i}$$

Modelos de Fuerzas Sociales

- *Driving effect:* logra mover al peatón en la dirección y velocidad deseada.

$$\mathbf{f}_{\alpha}^{0}(\mathbf{v}_{\alpha}, v_{\alpha}^{0}\mathbf{e}_{\alpha}) = \frac{v_{\alpha}^{0}\mathbf{e}_{\alpha}^{0} - \mathbf{v}_{\alpha}}{t_{\alpha}},$$

- *Boundary effect:* es un efecto de repulsión del peatón algún borde B.

$$\mathbf{f}_{\alpha B}(\mathbf{r}_{\alpha B}) = -\nabla_{\mathbf{r}_{\alpha B}} U_{\alpha B}(\|\mathbf{r}_{\alpha B}\|), \qquad U_{\alpha B}(\|\mathbf{r}_{\alpha B}\|) = U_{\alpha B}^{0} e^{-\|\mathbf{r}_{\alpha B}\|/R}$$

- *Interaction effect:* es un efecto de repulsión a otros peatones.

$$\mathbf{f}_{\alpha\beta}(r_{\alpha\beta}) = -\nabla_{\mathbf{r}_{\alpha\beta}} V_{\alpha\beta}[b(r_{\alpha\beta})], \qquad V_{\alpha\beta}[b(r_{\alpha\beta})] = V_{\alpha\beta}^{0} e^{-b/\sigma}$$

- *Attractive effect: es un efecto de atracción a un objeto i, decae con el tiempo.*

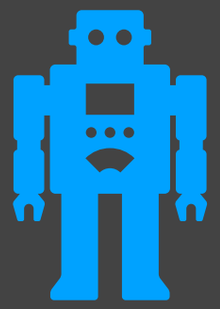
$$\mathbf{f}_{\alpha i}(\|\mathbf{r}_{\alpha\beta}\|, t) = -\nabla_{\mathbf{r}_{\alpha i}} W_{\alpha i}(\|\mathbf{r}_{\alpha\beta}\|, t)\,.$$

Actualización del movimiento

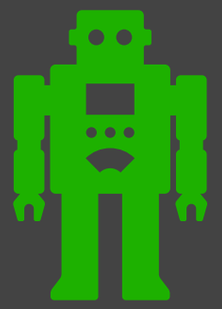
$$\sum f_\alpha = \frac{d\mathbf{v}_\alpha}{dt}$$

$$\frac{d\mathbf{r}_\alpha}{dt} = \mathbf{v}_\alpha(t)\,.$$

$$\mathbf{r}_\alpha(t + \Delta t) = \mathbf{v}(t)\Delta t + \mathbf{r}_\alpha(t)\,.$$



Reinforcement Learning



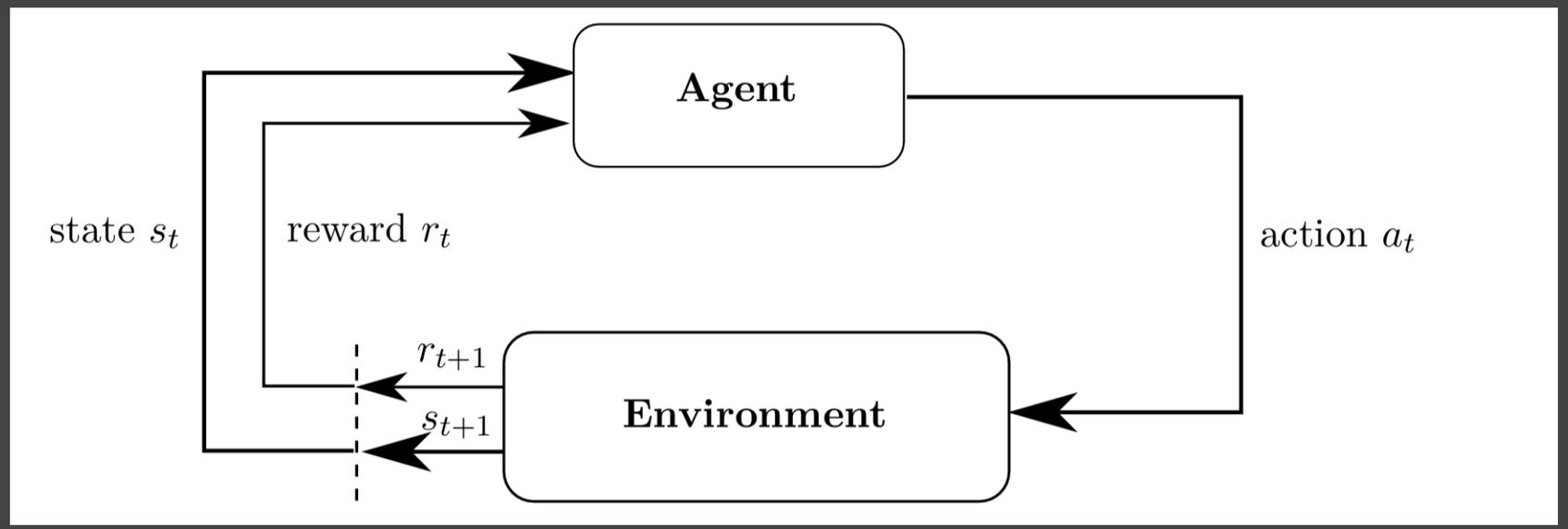
- Modelo para desarrollar agentes autónomos para desarrollar una tarea especifica teniendo como estimulo una recompensa. r_t
- Los agentes toman decisiones de acuerdo a una política

$$\pi(s_t) \rightarrow a_t$$

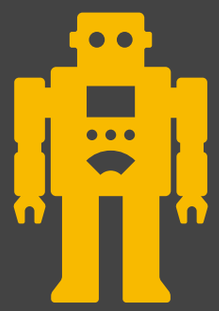
- El objetivo es maximizar la ganancia a largo plazo. Esto se puede hacer buscando la política optima

$$Q_\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$$

Reinforcement Learning con Redes Neuronales

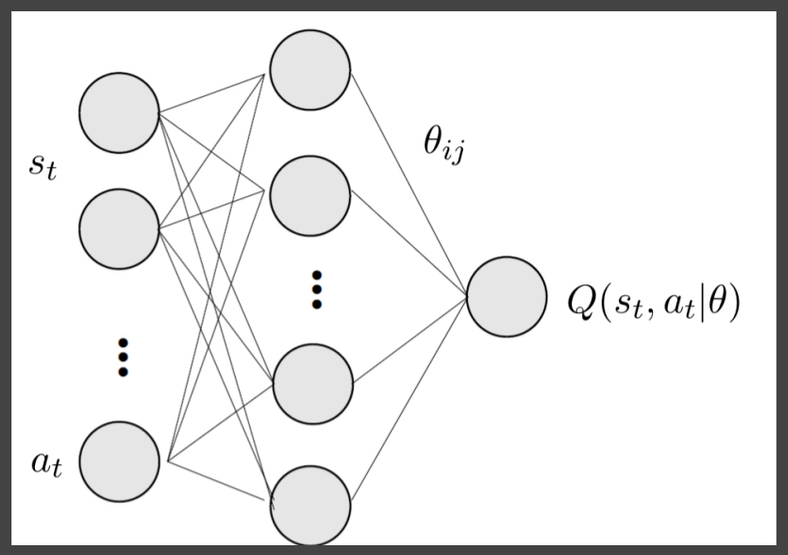


Esquema de interacción en RL.

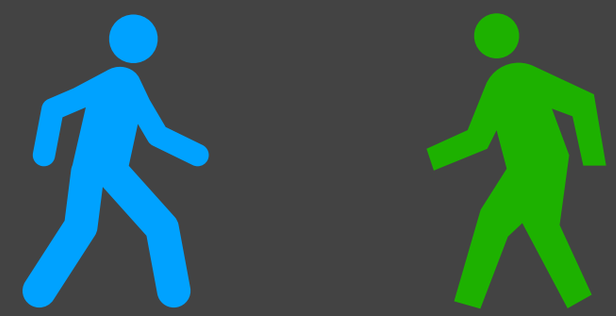


$$\pi(s_t) \rightarrow a_t$$

+



Red que aproxima Q



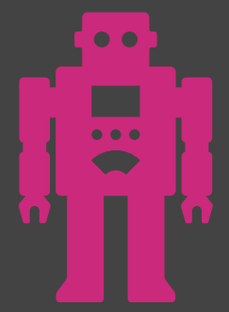
Deep Deterministic Policy Gradient

- Es un algoritmo tipo free-model actor-critic.
- Aproximar la política y la función Q por medio de redes neuronales.

$$Q(s, a \mid \theta^Q) \quad \mu(s \mid \theta^\mu)$$

- Sigue el principio de exploración vs explotación al agregar un proceso de Ornstein-Ulhenbeck.

$$\mu'(s_t) = \mu(s_t \mid \theta_t^\mu) + \mathcal{O}$$
$$\mathcal{O}(k) = \mathcal{O}(k-1) - \theta_{OU}\mathcal{O}(k-1) + \sigma_{OU}\mathcal{N} \, .$$



Deep Deterministic Policy Gradient

Algoritmo 1: *Deep deterministic policy gradient* (Lillicrap et at., [5])

Inicializar aleatoriamente $Q(s, a|\theta^Q)$ y $\mu(s|\theta^\mu)$ con θ^Q y θ^μ

Inicializar Q' μ' con $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \rightarrow \theta^\mu$

Inicializar el replay buffer $\mathcal{D}$

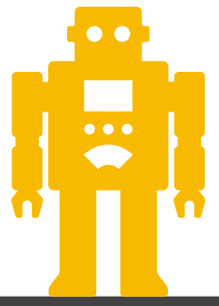
para *cada episodio* **hacer**

- Inicializar el proceso $\mathcal{O}$ para la exploración
- Recibir una observación inicial s_1
- **para** *cada paso en el episodio* **hacer**
  - Seleccionar $a_t = \mu(s_t|\theta^\mu) + \mathcal{O}$
  - Ejecutar acción a_t, observar recompensa r_t y el nuevo estado s_{t+1}
  - Almacenar (s_t, a_t, r_t, s_{t+1}) en $\mathcal{D}$
  - Tomar una muestra con un tamaño N de la forma (s_i, a_i, r_i, s_{i+1}) de $\mathcal{D}$
  - Establecer $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
  - Actualizar critic minimizando $L = \frac{1}{N}\sum(y_i - Q(s_i, a_i|\theta^Q))^2$
  - Actualizar actor usando *sampled policy gradient*:

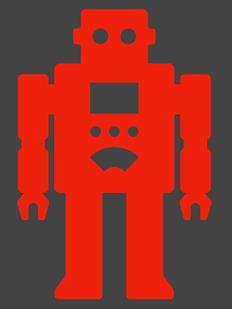
$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)\Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)\Big|_{s=s_i}$$

  - Actualizar las redes target:

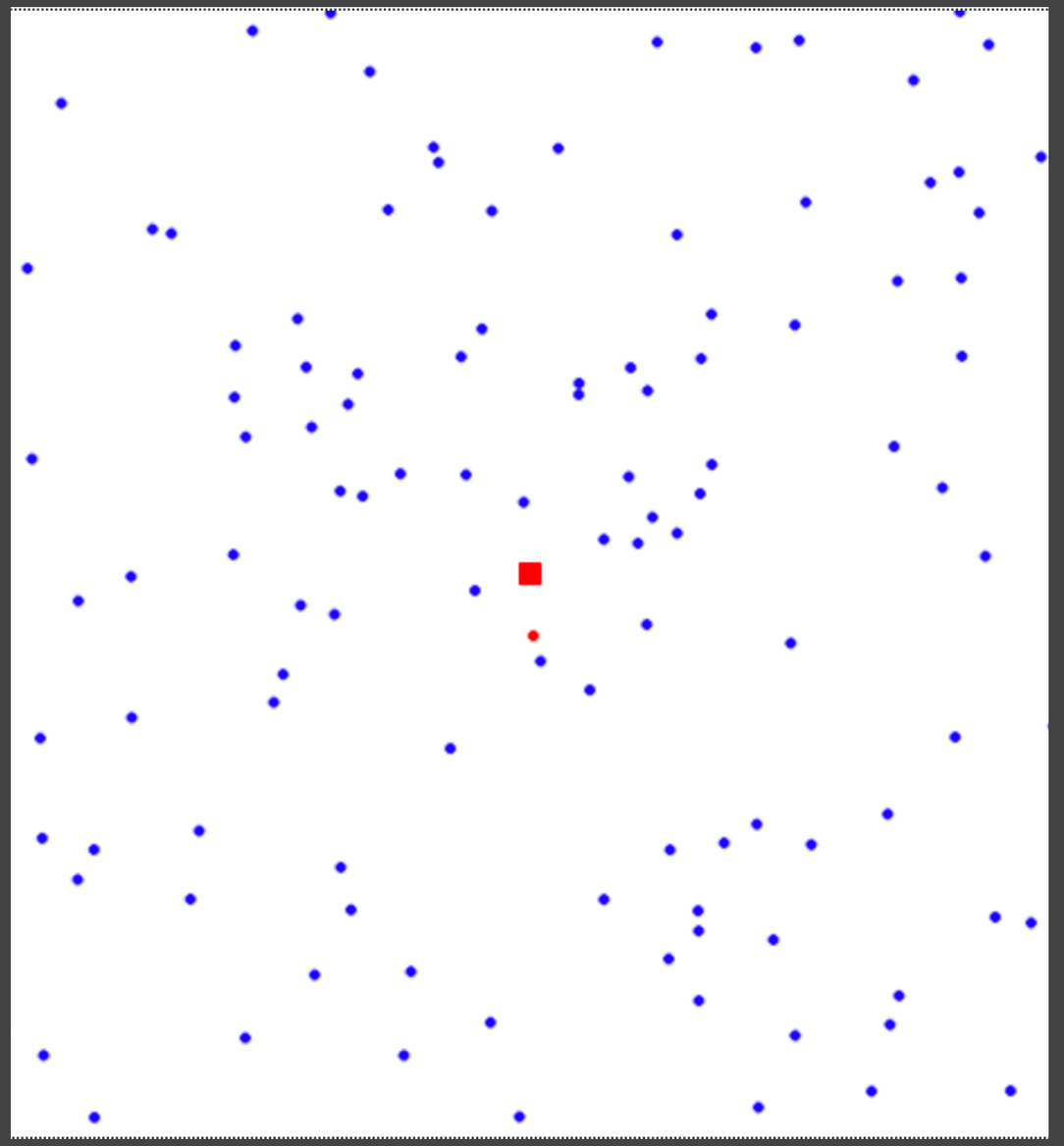
$$\theta^{Q'} \rightarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \rightarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'},$$



Ambiente

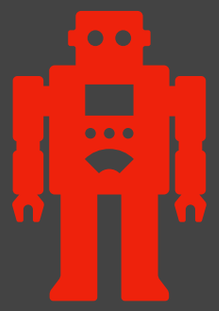


- Basado en el modelo de fuerzas sociales
- Diseñado en Mesa, Python.
- Objetivo: (x_g, y_g)

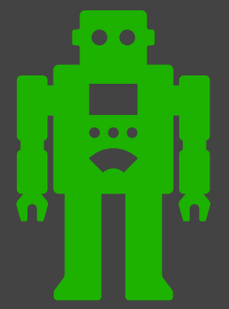


El agente (circulo rojo) debe de llegar al parche rojo.

Recompensa basada en el modelo de fuerzas sociales



$$r_t = r(g)_t + r(a)_t + \left(\sum_{h=1}^{H} r(h)_t \right)$$



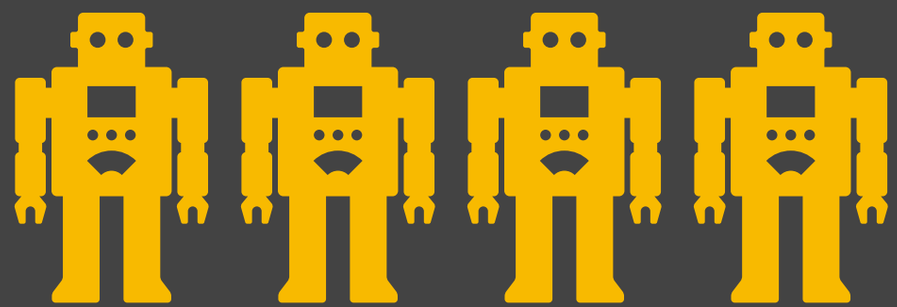
- Estado y acción del agente: $s_t = (x_t, y_t, \phi_t), \quad a_t = \omega_t$ **(vel angular)**

- Penalizar cercanía con peatones: $r(h)_t = r_h \frac{1}{\sigma_h \sqrt{2\pi}} \exp(- (d(x_t, y_t, x_h, y_h) - \mu_h)^2 / 2\sigma_h^2),$

- Recompensar acercamiento al objetivo : $r(g)_t = \begin{cases} r_g & \textbf{si } d(x_t, y_t, x_g, y_g) < D_g \\ r_p \cdot l_t & \textbf{e.o.c}, \end{cases}$

- Estimular una velocidad angular suave: $r(a)_t = r_a |\omega_t|$

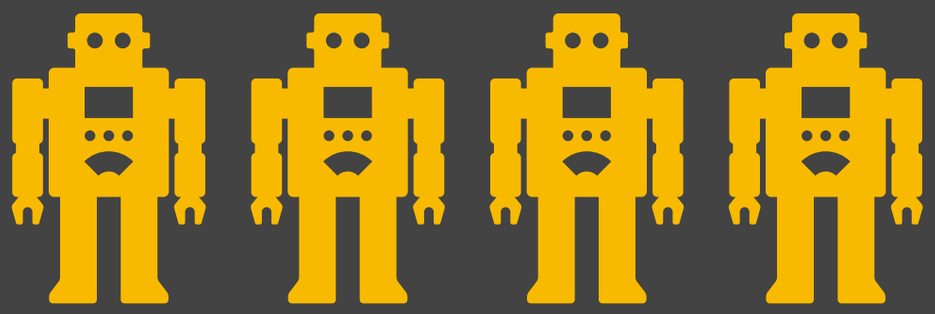
Evaluación

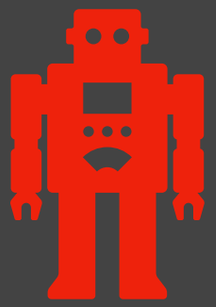


- Episodios: 8
- Pasos: 25
- Condición inicial:

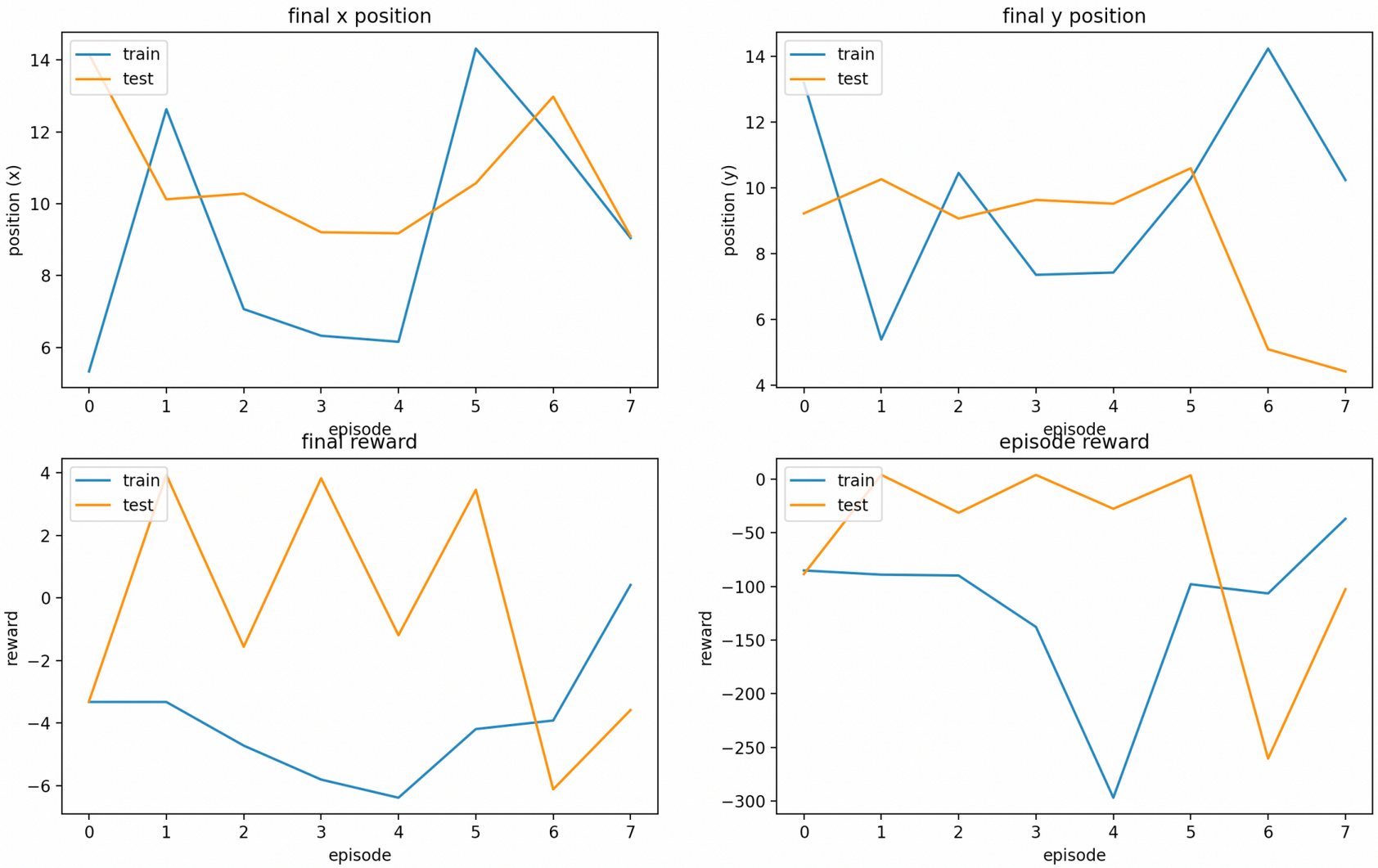
$$(x_t, y_t) \in [x_g - 2.5, y_g - 2.5] \times [x_g + 2.5, y_g + 2.5]$$

- Por cada episodio un entrenamiento y una prueba





Resultados


final x position
train
test
position (x)
episode
final y position
train
test
position (y)
episode
final reward
train
test
reward
episode
episode reward
train
test
reward
episode