Proyecto Final: Simulador de Muchedumbres

David Grandes
Instituto Tecnológico de Buenos Aires
dgrandes@alu.itba.edu.ar

Matias Santiago Pan
Instituto Tecnológico de Buenos Aires
mpan@alu.itba.edu.ar

Juan Santos
Instituto Tecnológico de Buenos Aires
jsantos@itba.edu.ar

Daniel Parisi
Instituto Tecnológico de Buenos Aires
dparisi@itba.edu.ar

Instituto Tecnológico de Buenos Aires, 28 de Diciembre de 2011

Resumen

Este proyecto tiene como objetivo la realización de un Simulador de Muchedumbres que permita la investigación de distintas políticas de movimiento. Se investigará el desempeño de técnicas de Aprendizaje Reforzado en la navegación de los agentes en distintos entornos.

La modelización y simulación de muchedumbres es un problema de interés para varias disciplinas de ingeniería y logística. Este Proyecto Final de Grado tiene como objetivo la investigación de Aprendizaje por Refuerzo como política de movimiento en la simulación de muchedumbres, con el deseo de encontrar una manera mas realista de simular la navegación de las personas en una muchedumbre

La investigación de la efectividad de políticas de Aprendizaje con Refuerzo se realizaron a través de un simulador. El simulador fue creado por el equipo y permitió la creación de entornos y distintos tipo de agentes para evaluar el aprendizaje. A su vez el simulador graba las simulaciones y las reproduce posteriormente para poder analizar los resultados.

Los resultados a favor del aprendizaje con Refuerzo son variados. Se observan mejoras en ciertos entornos simples, pero en otros mas complejos, donde el agente debe interactuar con alta densidad de obstáculos se nota una tendencia a sobrecompensar y fallar repetidamente al intentar esquivarlos.

Tabla de contenidos

Capítulo 1 - Introducción

En la actualidad existen diversos métodos que simulan el comportamiento de agentes independientes moviéndose en un entorno. Uno de los mecanismos más sencillos es el uso de fuerzas, atractivas o repulsivas, que gobiernen el movimiento de los agentes. Este mecanismo tiene serias deficiencias en entornos complejos, o con caminos no directos entre el agente y su destino. Esto se debe a que al ser dirigido meramente por fuerzas, solo puede dirigir en caminos rectos. De esta manera, si el punto de deseo del agente se encuentra detrás de una pared, este no tiene manera de saber como llegar a ella y es repelido por la fuerza normal de la pared, dejando al agente atrapado en el lugar donde la fuerza resultante es 0.

Lo que se propone es modificar este modelo por uno de aprendizaje reforzado. El aprendizaje por refuerzo es un área de estudio que contempla como un agente toma decisiones en un entorno no totalmente conocido. El agente maximiza la utilidad mediante pequeños refuerzos en su conducta, dependiendo si tomo una buena o mala decisión. A su vez se pone enfoque en el desempeño en vivo, por lo cual el agente tiene que encontrar un balance apropiado entre la exploración de mejores resultados y la explotación de conocimiento ya establecido.

Se abordará el problema utilizando el método de aprendizaje por refuerzo llamado **Q-Learning**. Este método consiste en aprender asignando diferentes utilidades a distintos estados posibles en los que puede estar un agente. El agente cambia de un estado a otro realizando una acción $a \in A$ perteneciente a ese estado. El algoritmo define la función que define la calidad, la cual el agente intentará maximizar a la hora de tomar decisiones, como:

$$Q = S \times A \rightarrow R^{1}$$

El proyecto realizará el aprendizaje por refuerzo en 3 entornos diferentes y luego se analizará el comportamiento en un entorno donde no podrá aprender sino que deberá explotar el conocimiento ya adquirido. Los agentes que no utilicen Q-Learning se regirán por el modelo de fuerza social propuesto en *Simulating dynamical features of escape panic*[2].

---

[1] Función que define la calidad en Q-Learning

[2] Simulating Dynamical Features of Escape Panic - Helbing, Farkas, Vicsek - Septiembre 2000

Capítulo 2 - Antecedentes

Modelo de fuerza social

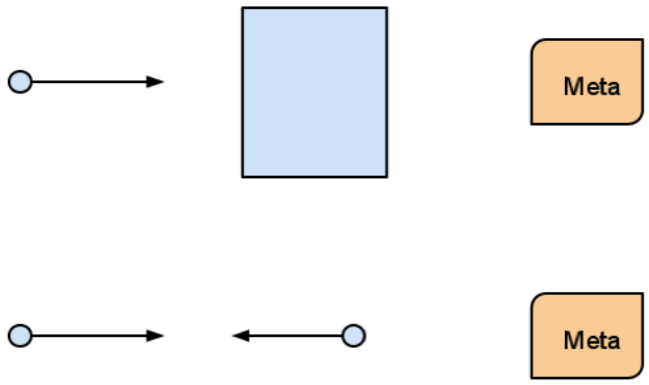
El modelo de fuerzas está inspirado en el modelo de Helbing, et al. Este consiste de 3 fuerzas:

- Fuerza del deseo
- Fuerza granular
- Fuerza de contacto

El modelo matemático se expresa de la siguiente manera:

$$m_i \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = m_i \frac{v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t)}{\tau_i} + \sum_{j(\neq i)} \mathbf{f}_{ij} + \sum_W \mathbf{f}_{iW}$$

La principal desventaja del modelo de fuerza social, y el motivo por el cual se busca una alternativa a este, es que los agentes tienden a atascarse cuando se enfrentan a un obstáculo que yace sobre la normal de éste hacia su objetivo, se pretende solucionar este problema permitiendo al agente “esquivar” obstáculos antes de llegar a la situación de contacto.



Situaciones problemáticas para el modelo de fuerza social

Fuerza de deseo

La fuerza de deseo ($\boldsymbol{v_i}$ en el modelo de fuerzas) es una fuerza que tiene como dirección el objetivo al cual el agente quiere llegar. La magnitud de está fuerza es una variable de cada simulación y basándose en el modelo propuesto por Helbing se seteó en *1.4m/s*.

Fuerza granular

Interaccion entre peatones

La fuerza granular ($\mathbf{f_{ij}}$ en el modelo de fuerzas) es una fuerza que modela la aversión natural de los agentes a entrar en contacto con otros agentes o cuerpos.

Es modelada como una función exponencial, cuyo argumento es negativo para distancias positivas entre los involucrados, es decir que cuando están lejos existe una fuerza pequeña que los obliga a repelerse. Sin embargo, cuando se tocan, esta fuerza crece exponencialmente en el eje positivo.

La fuerza granular es la que mantiene la consistencia de los cuerpos ya que los obliga a repelerse rápidamente cuando estos se acercan

social granular

$$\mathbf{f}_{ij} = \left\{A_i \exp[(r_{ij} - d_{ij})/B_i]\right\} + \left\{kg(r_{ij} - d_{ij})\right\}\mathbf{n}_{ij} + \kappa g(r_{ij} - d_{ij})\Delta v_{ji}^t \mathbf{t}_{ij}$$

Fuerza de contacto

Interaccion entre peatones y paredes

La fuerza de contacto es la que modela el rozamiento Coulombiano. Este tiene dos componentes, la normal y la tangencial.

La componente tangencial simula la fricción contra las superficies, mientras que la normal es un impulso de reflexión sobre la misma.

$$\mathbf{f}_{iW} = \left\{A_i \exp[(r_i - d_{iW})/B_i] + kg(r_i - d_{iW})\right\}\mathbf{n}_{iW} - \kappa g(r_i - d_{iW})(\mathbf{v}_i \cdot \mathbf{t}_{iW})\mathbf{t}_{iW}$$

Aprendizaje por refuerzo

El aprendizaje por refuerzo está fuertemente inspirado por la psicología conductista y el concepto, en su nivel más básico, implica que un agente en un entorno dado deberá tomar acciones que maximicen la recompensa recibida.

El modelo básico de aprendizaje por refuerzo consiste de

- Un conjunto de estados *S*
- Un conjunto de acciones *A*
- Reglas de transición entre los estados
- Reglas que determinen la recompensa para cada transición posible
- Reglas que determinen qué es lo que un agente observa en un momento dado

Los agentes que aprenden por refuerzo actúan en intervalos discretos de tiempo, en cada tiempo *t* el agente sensa una observación $o_{t,\text{ luego}}$ elige un acción a_t del conjunto de acciones permitidas y la lleva a cabo. El entorno pasa al próximo estado y se determina la recompensa por la acción elegida.

El aprendizaje por refuerzo consta de dos etapas diferentes

- Exploración: donde el agente prioriza el aprendizaje. Durante esta etapa las decisiones no siempre van a ser las óptimas sino que el agente se dedica a descubrir nuevos estados.
- Explotación: donde el agente prioriza la maximización de la recompensa. Durante esta etapa el agente tomará decisiones que le reporten la mejor utilidad.

Q-Learning

El método elegido para el aprendizaje por refuerzo es Q-Learning. Q-Learning es una técnica de aprendizaje por refuerzos que se ajusta perfectamente a los requerimientos del simulador ya que no requiere que se modele el sistema completo para que esta sea efectiva, en todo momento el agente solo está consciente de lo que lo rodea y nada mas.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \times \left[\overbrace{\underbrace{R(s_t)}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{max future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right]$$

es constante por lo que debería ser alpha_t

En el caso de Q-Learning se agregan 2 variables al modelo básico de aprendizaje por refuerzo, estas son:

- Learning Rate: es un factor que determina en que medida la información nueva reemplaza a la vieja en cuanto al valor de la utilidad de una acción *a* en un estado *s*.
- Discount factor: es un factor que determina que tan importantes son las recompensas, un valor de descuento mayor o igual a 0 hará que el aprendizaje no converja.

mayor o igual a 1

En nuestra situación los valores que toman estos factores son los siguientes:

- Learning Rate: *0.1*
- Discount Factor: *0.9*

Capítulo 3 - Trabajo Realizado

Simulador

El simulador desarrollado cuenta con las siguientes características

- Configuración de agentes por medio de archivo
- Configuración de entorno por medio de archivo
- Configuración de frame skip desde GUI. El usuario, al realizar una simulación puede cambiar, si así lo desea, el frame skip del simulador, esto causa que la actualización visual del estado del simulador se realice cada una cantidad mayor de pasos pero provee la ventaja de liberar el procesador para realizar iteraciones de simulación mas rápidamente.
- Permite guardar simulaciones en formato binario.
- Permite reproducir simulaciones guardadas en formato binario.
- Durante la simulación el usuario puede hacer click derecho sobre un agente para ver un panel que se actualiza en tiempo real con la posición y la descomposición de las fuerzas que actúan sobre él.
- Durante la simulación el usuario puede hacer click para agregar un overlay sobre los agentes que indica la dirección y magnitud de las fuerzas que actúan sobre él.
- Posibilidad de pausar una simulación
- Posibilidad de reiniciar una simulación
- El simulador guarda el estado de los agentes que utilizan Q-Learning a archivos de log para su posterior análisis.

Se presta soporte para la declaración de agentes de variados diámetros y masas, permitiendo también al usuario que declare que tipo de movimiento[3] deberá seguir el agente cuando este sea parte de una simulación y cual será el punto hacia el cual el agente deberá ir.
Se pueden declarar también obstáculos circulares o de formas geométricas arbitrarias.
Finalmente se permite la declaración de sumideros (puntos de llegada o metas) y de generadores (puntos de donde se generan agentes).

El simulador fue desarrollado utilizando JAVA y Swing. Consta de una implementación de interfaz que utiliza extensivamente *Worker Threads* para garantizar la responsividad de la misma al mismo tiempo que previene problemas de concurrencia.

Además la simulación de agentes se realiza utilizando *Thread Pools*, esto permite paralelizar los cálculos dentro de una misma iteración y mejorar el tiempo que toman las mismas. Se puede asegurar que al paralelizar los cálculos no se producen problemas en cuanto al modelo de fuerzas sociales ni al de Q-Learning ya que se asegura que los *threads* están sincronizados y no se empieza nunca con el próximo paso de simulación hasta haber completado el actual.

---

[3] Ver apéndice técnico para distintos tipos de movimiento

Teniendo en cuenta ciertas dificultades que se encontraron durante el desarrollo, principalmente en el área de GUI, consideramos que JAVA, si bien es un lenguaje muy flexible, puede no haber sido la mejor elección sobre la cual construir el proyecto.

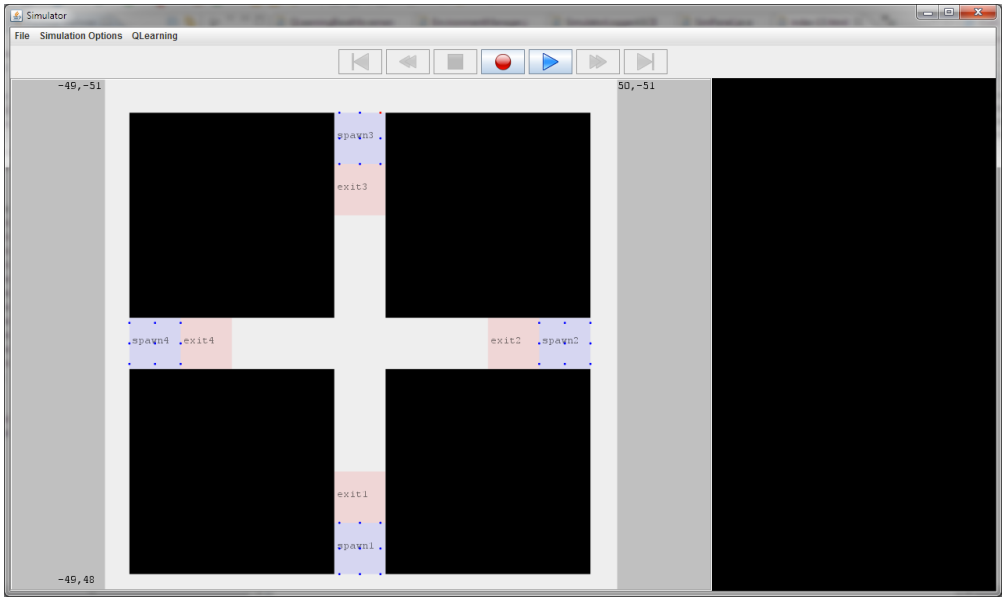
Una mejor elección podría ser la utilización de un motor de video juegos para poder aprovechar la sincronización de threads y las vastas librerías de vectores y física que estos motores poseen. Entre ellos los que más llaman la atención son:

- Cocos2d: (http://cocos2d.org/) es una librería de Python opensource que existe desde febrero de 2008. Su principal ventaja sobre la utilización de JAVA es como ya se mencionó, que este motor permite un manejo mas cómodo de la graficación. Su otra ventaja es que al estar basada sobre OpenGL provee una interfaz mas fluida al usuario.
- Unity3D: (http://unity3d.com/) es un motor de vídeo juegos con soporte nativo multiplataforma. Si bien no es opensource como Cocos2d, provee con un entorno de edición que facilita tareas típicas que acelera mucho el desarrollo. En nuestra opinión esta es la mejor de las 2 opciones.

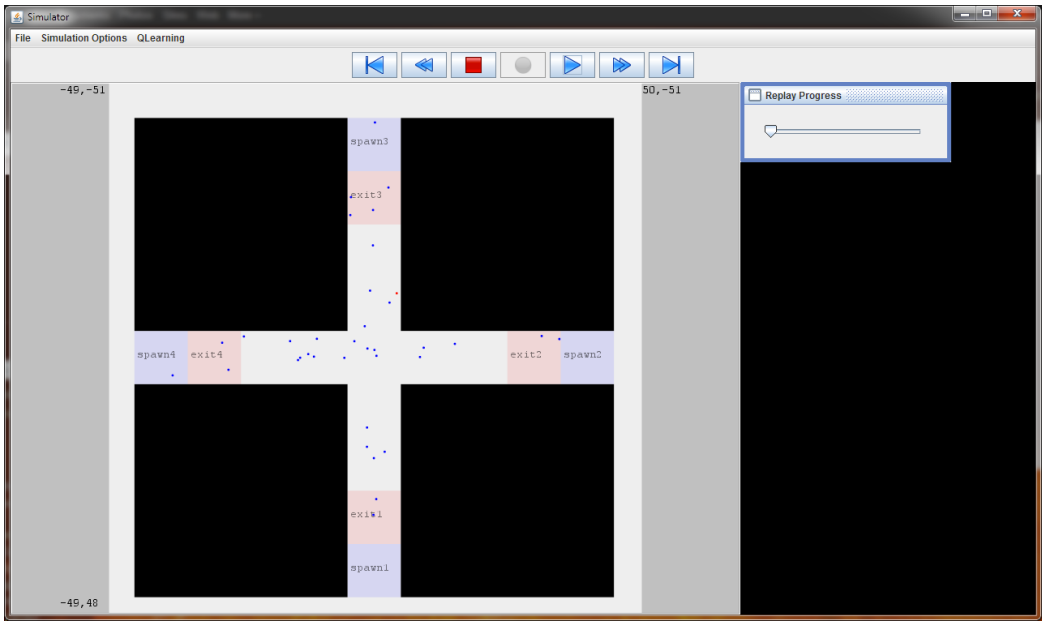
Modos de uso

El simulador cuenta con dos modos principales:

- Simulación: donde el usuario configura el simulador con archivos de entorno y agentes y realiza las simulaciones.
- Reproducción o replay: donde el usuario configura el simulador con un archivo previamente generado que guarda el estado de los agentes en cada intervalo de tiempo.



Simulador en modo Simulación



Simulador en modo Reproducción o replay

Cuando el simulador se encuentra en modo reproducción el usuario tiene la posibilidad de avanzar cuadro por cuadro, reproducir normalmente e ir al principio o final. Así mismo el simulador provee un slider para que el usuario pueda situar la simulación en cualquier punto arbitrario de la misma

Fuerza Social

El modelo de fuerza social se implementó sin modificaciones sobre el que propone Helbing et. El simulador soporta agentes sujetos a fuerza social de variados diámetros y masas así como también obstáculos de formas geométricas arbitrarias.

Implementación de Q-Learning

Modelización de los estados *S*

Los estados han sido modelados como un vector de estado de sensores dentro del agente. Este vector representa el campo visual del agente. Se decidió aproximar el campo visual del ser humano a 180 grados hacia el frente.

Los sensores se sitúan cubriendo todo el espectro visual, con mayor concentración en la zona frontal (Ver figura de distribución de sensores). Inicialmente se evaluó la posibilidad de utilizar 6 sensores distribuidos uniformemente a lo largo del arco visual del agente pero luego de realizar experimentos con esta configuración se optó por un modelo de 8 sensores con mayor concentración en la zona frontal.

Se probaron ambas alternativas en un escenario similar al del experimento 3 y se llegó a la conclusión de que el modelo de 8 sensores representa una mejora del 40% (en performance de llegadas al objetivo vs. colisiones) a cambio de un impacto en el tiempo de sensado de un 25% (el sensado toma 25% más que cuando se utilizan solo 6 sensores).

El agente tiene la capacidad de observar a una distancia limitada de su posición en un momento dado, esta distancia fue determinada en conjunto con los tutores en 10m.

La forma en la que el agente releva los datos se refleja en el estado con una escala de 3 valores: *0* significa que no se observa nada en ese sensor; *1* y *2* significan que el agente detecta riesgo en ese sensor. El valor del sensor depende directamente de la velocidad relativa observada por el agente en sentido normal para un sensor dado, esto resulta en un total de 6561 estados diferentes posibles.

La formula que calcula el riesgo entre el agente i y el agente k es la siguiente:

$$\mathbf{R_{ik}} = ((\mathbf{V_i} - \mathbf{V_k}) \cdot (\mathbf{P_i} - \mathbf{P_k}))$$

$$\mathbf{Riesgo} = \begin{cases} \mathbf{0} \text{ si } \mathbf{R_{ik}} > 0 \\ \mathbf{1} \text{ si } \mathbf{R_{ik}} <= 0 \wedge \mathbf{R_{ik}} > -0.5 \\ \mathbf{2} \text{ si } \mathbf{R_{ik}} <= -0.5 \end{cases}$$

Donde $\mathbf{V_i}$ es la velocidad instantanea del agente **i** y $\mathbf{P_i}$ es la posición del agente **i**.

Configuración de 6 sensores (izquierda) contra 8 sensores. La configuración de 8 sensores aprovecha huecos en el arco frontal de visión que la configuración de 6 sensores es incapaz de reconocer

Modelización de las acciones *A*

Las acciones que el agente puede tomar son las siguientes:

Nombre	Descripción
ACTION_NONE	El agente no hace nada para cambiar su trayectoria. En la última versión del simulador existe un reflejo que fuerza al agente a elegir esta acción cuando no detecta amenazas.
ACTION_BACK	El agente activa un actuador que le imprime una fuerza de $0.5 * v_d$ en dirección opuesta a su velocidad actual
ACTION_LEFT	El agente activa un actuador que le imprime una fuerza de v_d hacia su izquierda
ACTION_RIGHT	El agente activa un actuador que le imprime una fuerza de v_d hacia su derecha

Donde v_d es la fuerza del deseo que ya se mencionó y cuyo valor es

$$\mathbf{v_d = 1.4\ m/s}$$

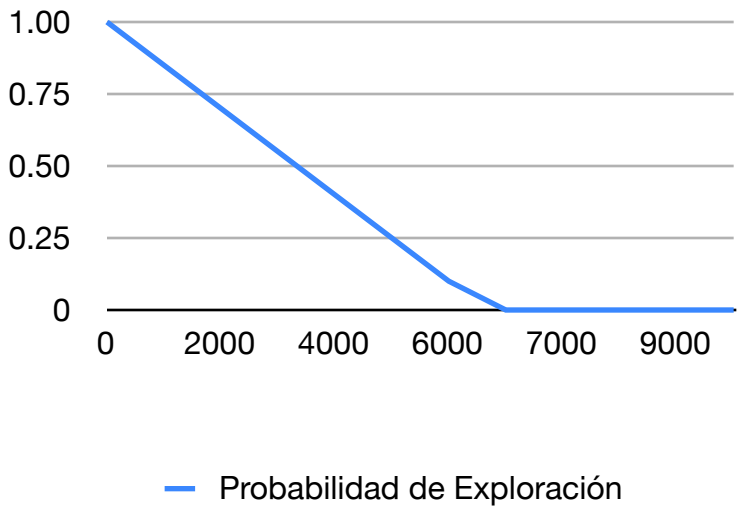
Esquema de refuerzos

Se decidió por un esquema de esfuerzos puramente negativos, cada vez que el agente choca con otro agente se refuerza con un valor de -1. La decisión se basó en que al seguir presente la fuerza del deseo mientras el agente aprende utilizando Q-Learning no es necesario recompensarlo por avanzar correctamente.

.

Exploración vs. Explotación

El agente que utiliza Q-Learning como su método de aprendizaje tiene inicialmente una probabilidad de explorar de *1.0*. Cada vez que se toma una decisión esta probabilidad se reduce en *0.0015*. De esta manera a partir del paso *6666* el agente ya no explora mas sino que siempre elige la acción que le reporta el mayor beneficio



Probabilidad de exploración vs. Iteraciones de Q-Learning

Capítulo 4 - Experimentos y Resultados

A continuación se presentan los resultados de los 4 experimentos propuestos. Los supuestos para las simulaciones son los siguientes:

- Se entrenó el agente durante *1000* pasos de decisión de Q-Learning.
- Se evalúa Q-Learning cada el equivalente a *200ms* en tiempo de simulación.
- El decay rate (el factor que decide cuando un agente debe explorar y cuando explotar) vale 0.15. Esto quiere decir que en *2/3* de la simulación el agente deja de explorar por completo
- Solo se otorgan refuerzos negativos cuando el agente colisiona con otro o con un obstáculo.
- No se otorgan refuerzos positivos.
- Cuando un agente que utiliza Q-Learning colisiona se reinicia su posición y la simulación continúa.
- La performance de una simulación se mide por el ratio *llegadas/colisiones*.
- Cuando debe decidir, el agente elige la acción con utilidad menos negativa.

Formatos de salida

Las simulaciones arrojan resultados en formato de log, los cuales se analizan para construir gráficos para dos métricas distintas:

- Performance: representa la cantidad de llegadas con exito contra la cantidad de colisiones que sufrio un agente
- Velocidad: representa la velocidad a la que se trasladó el agente durante el intervalo de tiempo entre decisiones de Q-Learning.

Ademas se loguea la matriz-Q que resultó del experimento para poder utilizarla en futuras simulaciones.
El formato de la matriz crudo es algo dificil de analizar por lo que los resultados que aqui se presentan se acompañan de las matrices en un formato mas ordenado.

El estado de la matriz-Q debe interpretarse de izquierda a derecha, es decir que el primer valor del vector corresponde al sensor de la derecha y el ultimo al ultimo sensor de la izquierda.

Los archivos de log tienen el siguiente formato:

```
-----SIMULATION STEPS------
Steps | Exploration Rate | QMatrix Size | Crashes | Victories | Distance Moved | Speed

[...]

3 9.955 0 0 0 7.8725033 0.03166943
4 9.94 0 0 0 11.027603 0.031075181
5 9.925 0 0 0 14.123461 0.03041324
6 9.91 0 0 0 17.150507 0.029613486
7 9.895 0 0 0 20.085972 0.018325046
8 9.88 0 0 0 21.883408 0.029660393
9 9.865 0 0 0 24.809654 0.03112627
10 9.85 0 0 2 27.918789 0.015563965
11 9.835 0 0 2 29.483282 0.04667282
12 9.82 0 0 2 34.14764 0.015563965
13 9.805 1 1 2 40.371246 0.04667282
14 9.79 1 1 2 45.025497 0.03206743
15 9.775 1 1 2 48.21961 0.04661985
16 9.76 1 1 2 52.873848 0.031128146
17 9.745 1 1 2 55.96422 0.046621844

[...]

----QMATRIX: [Sensed Threat] > ACTION == UTILITY------
[1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0] > ACTION_BACK == -0.0090
[1.0,1.0,0.0,0.0,0.0,1.0,0.0,1.0] > ACTION_BACK == -1.2926415E-5
[1.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,1.0] > ACTION_BACK == -4.7752223E-4
[1.0,1.0,1.0,1.0,1.0,1.0,0.0,1.0] > ACTION_RIGHT == -8.531005E-4
[1.0,1.0,1.0,0.0,1.0,1.0,0.0,1.0] > ACTION_BACK == -4.782969E-9
[1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0] > ACTION_BACK == -0.010286796
[1.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,1.0] > ACTION_BACK == -6.9513E-6
[1.0,1.0,0.0,0.0,1.0,1.0,0.0,1.0] > ACTION_RIGHT == -0.0090
[1.0,1.0,1.0,1.0,0.0,1.0,1.0,1.0] > ACTION_NONE == -0.008105819
[1.0,1.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0] > ACTION_BACK == -6.1514186E-7
[0.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,1.0] > ACTION_BACK == -1.3234372E-4
[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0] > ACTION_BACK == -0.0076863146
[1.0,1.0,0.0,1.0,1.0,1.0,1.0,1.0] > ACTION_BACK == -1.2707816E-4
[0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0] > ACTION_BACK == -6.536191E-4
[1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0] > ACTION_BACK == -0.015301644
[1.0,1.0,1.0,0.0,1.0,1.0,1.0,1.0] > ACTION_BACK == -5.7926154E-6
[1.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0] > ACTION_BACK == -0.006561402
```

Formato de archivo de Log analizado

Experimento 1

Dos agentes alineados que se enfrentan y deben eludirse con la política aprendida.

Para aprender se usarán 3 esquemas independientes, la política hallada en cada uno debe ser usada para llevar a cabo el Experimento Final 1 y comparar cual esquema de aprendizaje funciona mejor:

1. **Un sistema de un agente con un obstáculo (otro agente quieto) en la linea del target. la posición inicial del agente debe ser random alrededor de la linea que pasa por el obstáculo , de modo que lo eluda a veces por derecha y a veces por izquierda.**
2. **Ídem pero con la velocidad de deseo del agente aprende al doble.**
3. **Ídem pero con obstáculo móvil, es decir reemplazar al obstáculo por un agente que solamente tiene fuerza de deseo (y de contacto) y cuyo target esta en el origen del agente que aprende.**

Experimento 1: Dos agentes enfrentados

El experimento 1 es una prueba simple de la capacidad del simulador. En los 3 escenarios propuestos el agente que aprende se comporta como se esperaba y tiene una performance positiva. Lamentablemente esto no se debe tanto a la capacidad del simulador para entrenar agentes sino a la poca densidad de obstáculos que encuentra el agente entrenado

Observando la matriz Q se puede ver que los valores de utilidad son los esperados en casi todos los casos. Se adjunta la matriz Q del experimento 1.1 donde se ve que el agente favorece la acción ACTION_BACK cuando no siente amenazas inminentes

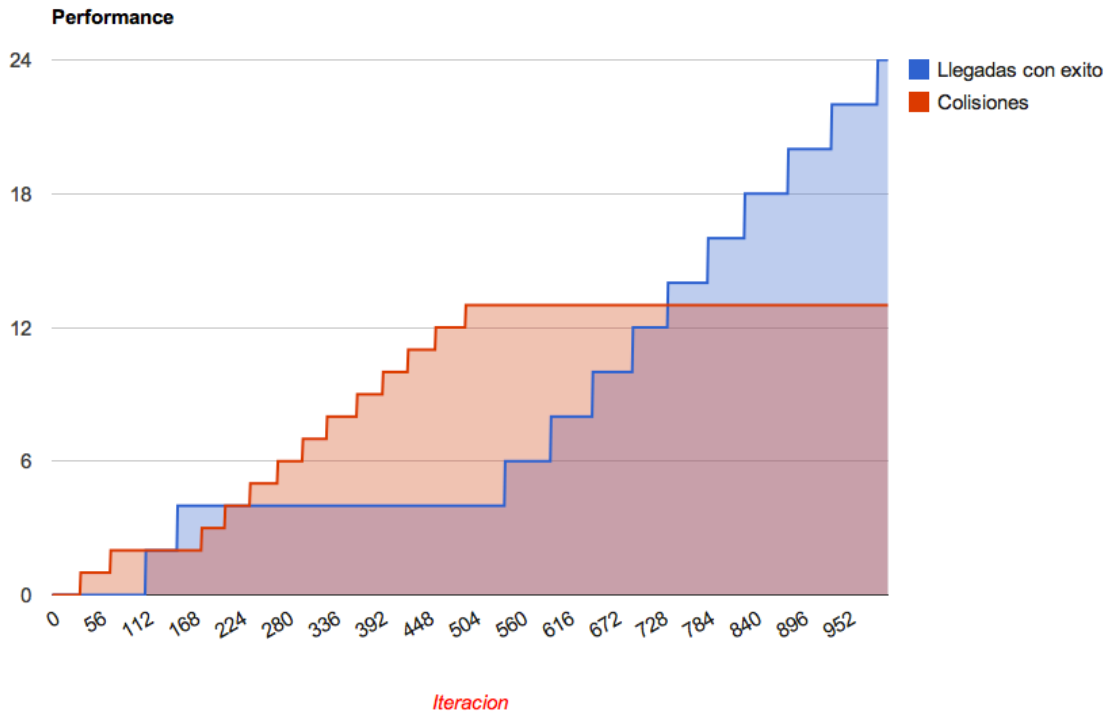
Cabe destacar también que viendo los gráficos de utilidad se nota claramente que alrededor del paso 400 la performance experimenta una mejoría notable causada por la menor posibilidad de movimientos random y el aprovechamiento de la estrategia aprendida.

En el experimento 1.3 sin embargo la performance no es igual de buena, debido a que al reiniciarse la posición del agente luego de llegar a la meta o de colisionar, el agente con movimiento trivial lo choca.

Se detecta un artefacto, especialmente en el experimento 1.1 donde el agente aprendió que en caso de no detectar nada en sus sensores la mejor opción es ir hacia la derecha, para solucionarlo y conseguir una solución al problema de esquivar obstáculos más optima se decidió implementar un reflejo que hace que en caso de que los sensores no detecten amenazas el agente realice siempre la acción ACTION_NONE.[4]

GRUPO	ACTION_LEFT	ACTION_RIGHT	ACTION_NONE	ACTION_BACK
[0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0]	+0.000000	-0.081729	-0.008019	+0.000000
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0]	+0.000000	-0.090000	+0.000000	+0.000000
[0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0]	+0.000000	-0.008355	-0.047830	-0.007290
[0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0]	+0.000000	-0.050115	-0.080680	-0.081000
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]	-0.000349	-0.000001	-0.005827	-0.000036
[0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]	+0.000000	+0.000000	-0.099000	-0.041395
[1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]	+0.000000	+0.000000	+0.000000	-0.044429
[0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0]	+0.000000	-0.002808	-0.003360	-0.009526
[0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0]	+0.000000	-0.078531	-0.080767	-0.062748

Matriz-Q resultante del experimento 1.1



Performance del experimento 1.1. Se puede observar que raramente colisiona el agente.

[4] Los resultados de aplicar este refuerzo se ven el “Experimento 1 - Modificación de reflejo automático”

Experimento 1 - Modificación de reflejo automático

Dos agentes alineados que se enfrentan y deben eludirse con la política aprendida.

Para aprender se usarán 3 esquemas independientes, la política hallada en cada uno debe ser usada para realizar el Experimento Final 1 y comparar cual esquema de aprendizaje funciona mejor:

1. **Un sistema de un agente con un obstáculo (otro agente quieto) en la linea del target. la posición inicial del agente debe ser random alrededor de la linea que pasa por el obstáculo, de modo que lo eluda a veces por derecha y a veces por izquierda.**
2. **Un sistema de un agente con un obstáculo (otro agente quieto) corrido hacia la derecha de la linea del target. la posición inicial del agente debe ser random alrededor de la linea que pasa por el target, de modo que lo eluda por izquierda.**
3. **Un sistema de un agente con un obstáculo (otro agente quieto) corrido hacia la izquierda de la linea del target. la posición inicial del agente debe ser random alrededor de la linea que pasa por el target, de modo que lo eluda por derecha.**

Se implementa un reflejo automático que obliga al agente a realizar ACTION_NONE cuando no detecta amenazas.

Se observa en esta nueva iteración del experimento 1 que el artefacto visto en el la iteración anterior ha desaparecido, con lo que se puede afirmar que la corrección propuesta es exitosa.

El comportamiento del agente es esquivar el obstáculo por su derecha, de la misma manera que había aprendido en el experimento anterior.

Cuando se evalúa la política aprendida corriendo el objetivo para la derecha el agente esquiva con éxito el obstáculo en todos los episodios. No es ese el caso cuando se lo corre a la izquierda, ya que el agente intenta esquivar el obstáculo para el mismo lado y por lo general no es exitoso.

Cuando se reentrena el primer punto utilizando un target que toma una posición random dentro del sumidero el experimento 2 el agente sigue intentando esquivar siempre por la derecha, pero, a diferencia de cuando se lo entrena con el objetivo fijo, esta vez triunfa en cada episodio y finaliza la simulación con 0 colisiones. En el experimento 3 se repite esta tendencia.[5]

[5] Se incluye la matriz-Q aprendida en este experimento en los archivos con resultados

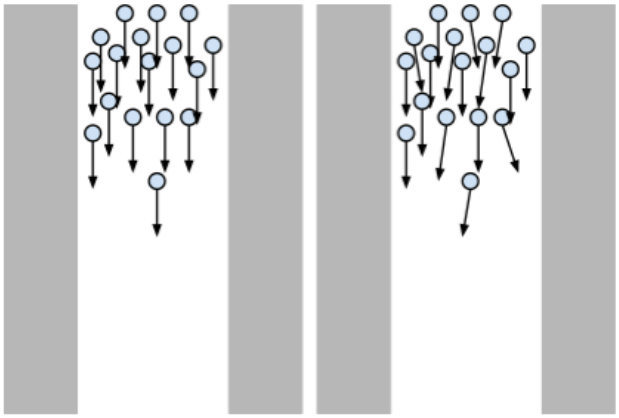
Experimento 2

Pasillo de 3 m de ancho y 10-15 m de largo. Originalmente hay entre 20 y 50 peatones en un extremo del pasillo con una densidad de aproximada de 1 agente/m^2. Todos tienen velocidades deseadas distintas entre 0.7 y 1.5 m/s. Todos tienen un target asignado en el extremo opuesto del pasillo con una distribución random y fija (elegida al inicio y queda el target fijo después) lo cual producirá ciertos cruces suaves.

En este caso el esquema para aprender seria, un sistema igual al Experimento (2) pero un solo agente aprende y los demás tienen solo fuerza del deseo, primero con targets alineados (no se deberían entrecruzar), luego targets random (puede haber entrecruzamiento entre los agentes que no aprenden). A su vez se debe entrenar con las dos variantes de densidad baja (1 p /m^2) y alta (2 p/m^2).

Se entrena de 4 formas diferentes

1. **Densidad baja y targes alineados**
2. **Densidad baja y targets cruzados**
3. **Densidad alta y targets alineados**
4. **Densidad alta y targets cruzados**



Experimento 2: Un corredor lleno con 20 a 50 agentes.

Los resultados del experimento 2 revelan que el modelo de estados de Q-Learning que se emplea no se comporta bien cuando los sensores están sobrecargados. Esto se nota especialmente en el experimento de alta densidad, donde el agente tiene aún menos espacio para moverse. Incluso en el los experimentos de baja densidad el resultado no es bueno, aunque presenta cierta mejoría en el caso 2.1.1, único experimento donde las llegadas a la meta fueron superiores a las colisiones.

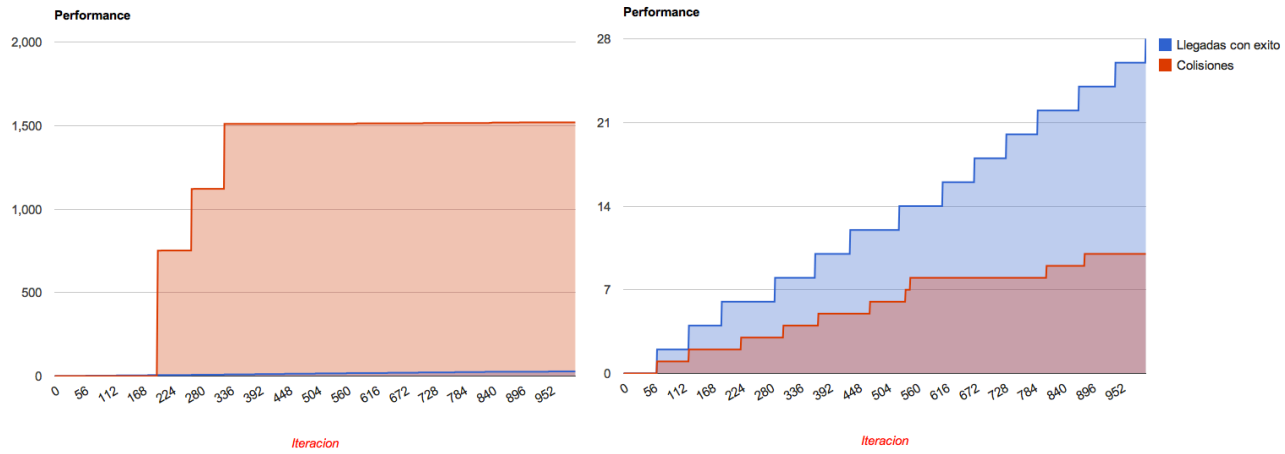
En la matriz-Q se puede observar que en muchos casos todas las acciones asociadas a un estado tienen utilidades muy similares o los valores de utilidad mas altos no están asociados a la acción que uno esperaría, esto sucede por 2 razones principales:

- El agente se encuentra atrapado entre otros agentes y no existe una acción que lo haga salir de esa posición, tome la acción que tome choca con otro agente en los costados y por eso se lo recompensa de manera negativa.

- El agente se encuentra atrapado entre otros agentes, si decide frenar sucede muy seguido que un agente mas retrasado lo choca y por eso se lo recompensa de manera negativa también.

La mas preocupante de las razones es la segunda, ya que una decisión perfectamente viable dada la situación sensada (todo el arco visual presenta amenazas) sería frenar, pero al haber agentes que avanzan desde atrás no se tiene la imagen del entorno completo y por lo tanto las decisiones que toma el agente de Q-Learning no son lo necesariamente informadas.

Otro motivo para este resultado negativo es que cuando el agente aplica una fuerza correctiva no puede moderar la magnitud de la misma, lo que causa que al tratar de esquivar un obstáculo termine golpeando a otro. Los resultados serían aun peores si los agentes que no aprenden no esquivaran al agente que utiliza Q-Learning, ya que este reflejo muchas veces salva malas decisiones que toma el agente.



Performance para el experimento 2.4 (izquierda) y el 2.1 (derecha)

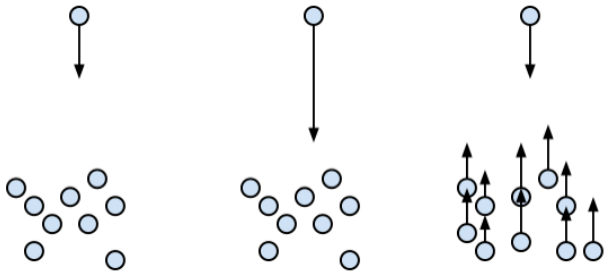
GRUPO	ACTION_LEFT	ACTION_RIGHT	ACTION_NONE	ACTION_BACK
[...]				
[0.0,0.0,1.0,1.0,1.0,1.0,0.0,1.0,1.0]	+0.000000	+0.000000	+0.000000	-0.000005
[0.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0]	-0.000537	+0.000000	+0.000000	+0.000000
[0.0,0.0,0.0,1.0,1.0,1.0,1.0,0.0,0.0]	-0.000048	-0.000008	+0.000000	-0.000063
[0.0,0.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0]	-0.000109	+0.000000	+0.000000	-0.008151
[0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0]	+0.000000	+0.000000	+0.000000	-0.000044
[0.0,0.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0]	-0.000001	+0.000000	+0.000000	-0.000007
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0]	-0.000109	+0.000000	+0.000000	-0.000120
[0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,1.0]	-0.000004	-0.003120	+0.000000	-0.000025
[...]				

Extracto de la matriz-Q del experimento 2.2

Experimento 3

Ídem Experimento 1 pero con N agentes enfrentados. Para el aprendizaje se utiliza uno que aprende con "un bosque" de obstáculos (agentes fijos) en un arreglo desordenado. En este caso usar las 3 variantes de aprendizaje análogas a las del Experimento 1

1. **Agente con velocidad de deseo normal (aproximadamente 1,2 m/s) y obstáculos fijos**
2. **Agente con velocidad de deseo al doble y obstáculos fijos**
3. **Agente con velocidad de deseo normal y los obstáculos son ahora agentes móviles que se mueven en bloque todos con la misma velocidad de deseo en sentido opuesto al del agente que aprende.**



Experimento 3: Un agente se enfrenta a un bosque de agentes.

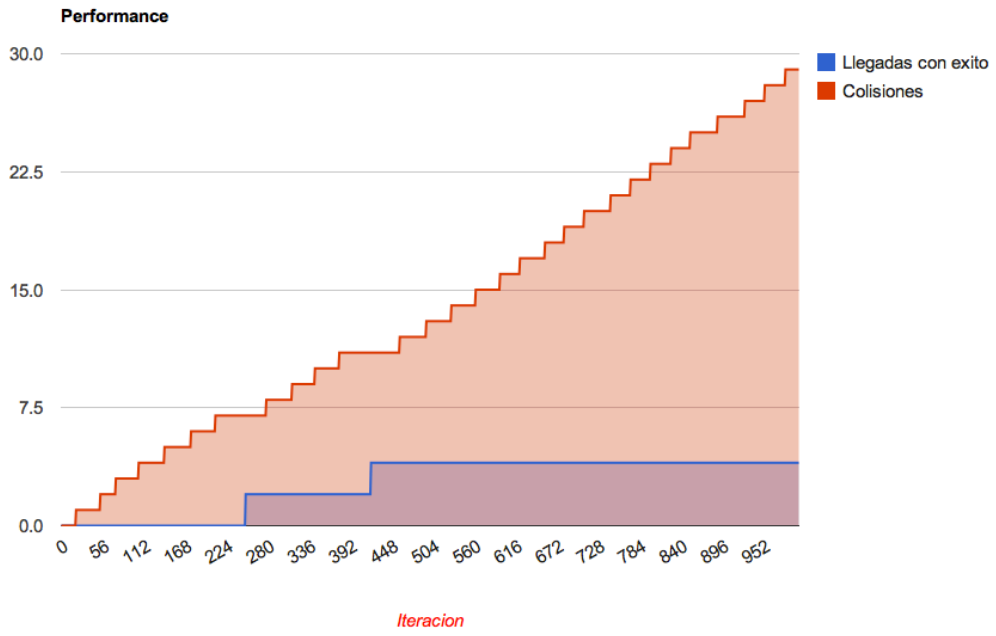
En el experimento 3 el agente se encuentra en un entorno menos agresivo que el que enfrenta en el experimento 2. si bien los resultados son mejores, queda en evidencia que el agente no es bueno manejándose en espacios cerrados.

Al revisar el replay del experimento 3.1 (el más simple de los tres escenarios) se ve que el agente supera con éxito la primer barrera de agentes pero luego no puede pasar la segunda más de 4 veces. Dos conclusiones surgen de este hecho:

- El agente podría beneficiarse de mayor maniobrabilidad, muchas veces, al estar obligado a ir siempre hacia adelante no le es posible evitar el peligro
- El agente podría beneficiarse de un mayor rango de visión, si tuviera esta posibilidad podría ver el peligro desde mayor distancia y quizás evitar el bosque de obstáculos rodeándolo en vez de intentar atravesarlo.

El experimento 3.3 ofrece otro escenario distinto donde los obstáculos se mueven en dirección contraria al agente que aprende. En este caso los resultados son positivos aunque se observa que el agente no siempre acierta cuando debe decidir en situaciones donde los sensores se encuentran en su mayoría cargados.

En general en el experimento nunca se establece una clara tendencia a superar obstáculos por lo que el resultado no es positivo

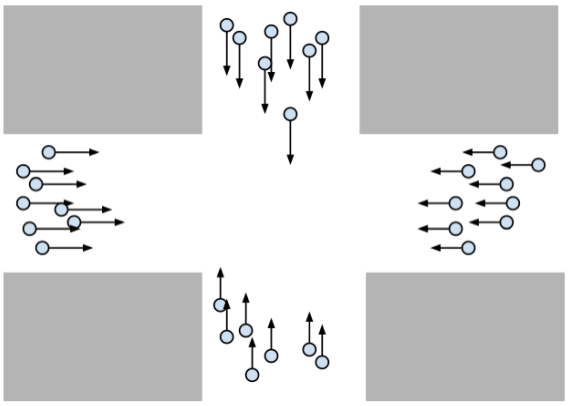


Performance para el experimento 3.1

Experimento 4

Cruce de calles perpendiculares (Lavalle y Florida) con agentes avanzando en ambos sentidos en cada calle. En este caso no habrá entrenamiento, se debe usar la política aprendida en el Experimento 3.

Se utiliza la política descripta en el punto anterior



Experimento 4: Cruce de calles transitadas

Los resultados coinciden con lo que a priori se esperaba: el experimento 4.1 no tiene resultados positivos (de hecho el agente nunca llega al objetivo), esto se debe a que la política que utiliza no maneja correctamente valores de riesgo altos (como cuando el agente tiene por delante uno o mas agentes en dirección contraria a la suya). El principal problema observado es que al registrar valores de riesgo similares en todos sus sensores los agentes tienden a irse hacia la derecha por no conocer una mejor forma de resolver el problema. Esta característica se repite en el experimento 4.2 y en el 4.3.

En el experimento 4.3 se puede ver que los agentes resuelven exitosamente colisiones entre ellos pero esto no es suficiente para solucionar el problema de la tendencia de moverse hacia la derecha. Cabe resaltar que el experimento 4.3 es el único en el cual el agente monitoreado llega al objetivo

Capítulo 5 - Conclusiones

Se modificó el modelo de fuerza social propuesto por Helbing et. al para incluir elementos del aprendizaje por refuerzo. Se mantuvieron las fuerzas de deseo y de contacto pero se reemplazó la fuerza granular por un ajuste de dirección que surge de la inclusión del aprendizaje por refuerzo.

Se implementó un simulador que tiene la capacidad para manejar distintos métodos de movimiento, entre ellos el modelo de fuerza social y el de aprendizaje por refuerzo utilizando Q-Learning.

Una vez que el simulador se completó se procedió a probar la hipótesis de que la inclusión de Q-Learning dentro del modelo de fuerza social podía mejorar el comportamiento de los agentes en ciertos entornos en los que la fuerza social no es efectiva.

Cumplimiento de las expectativas

En general el comportamiento no es el esperado ya que en situaciones similares a las de la vida real el modelo propuesto no es fiable para resolver correctamente la evasión de obstáculos.

Solo se obtuvieron resultados positivos para los casos de simulación mas simples, como lo son el experimento 1 y 3, mientras que en el experimento 2 y 4 el agente falla por verse superado por la situación.

Posibles mejoras

Modelo híbrido de fuerza social - Q-Learning

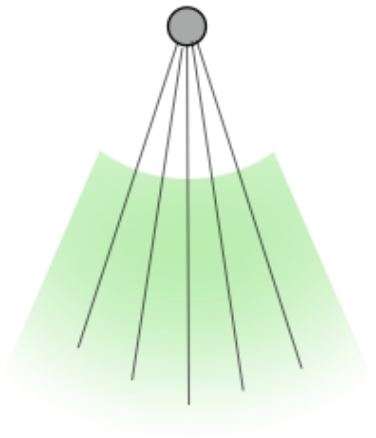
Consideramos que Q-Learning no es del todo efectivo en situaciones en las que el espacio disponible es escaso pero sí lo es cuando hay amplio lugar para maniobrar, inversamente, el modelo de fuerza social no maneja correctamente obstáculos a distancias grandes debido a que las fuerzas que lo componen disminuyen por razón del cuadrado de la distancia.

Proponemos un modelo híbrido donde el agente tenga una cantidad de sensores similar, agrupados en un ángulo de menor amplitud pero con un rango de visión considerablemente mayor, 30m por ejemplo. Los sensores incluso podrían funcionar como rayos en vez de triángulos ya que lo que se intenta evitar es chocar con obstáculos de gran tamaño y para ello no es necesario tener noción de que hay entre cada sensor sino que es suficiente con tener una visión general de los obstáculos en distancia.

La idea atrás de este modelo es que el maniobrar en situaciones de confinamiento sea manejado por el modelo de fuerza social pero que el agente tenga la suficiente inteligencia para esquivar obstáculos que no le reportan un perjuicio inmediatamente.

Para lograr esto los sensores de Q-Learning solo podrían sensar obstáculos mas allá del rango inmediato del agente, es decir a partir de los 15 metros.

El esquema de refuerzo no contemplaría refuerzos negativos sino que recompensaría al agente cuando sus sensores detecten menos obstáculos que en una iteración anterior. Se debería dar mayor importancia al modelo de fuerza social cuando hay obstáculos cerca y relegar a Q-Learning en este caso, ya que siempre será más importante la resolución inmediata de conflictos.



Posibilidad de crear entornos usando una herramienta gráfica

En este momento el simulador solo acepta entornos creados a partir de un archivo JSON. Sería de gran utilidad que éste permitiera la creación de entornos y agentes utilizando el mouse y arrastrando estos elementos al área de simulación.

Idealmente se permitiría grabar los entornos en el mismo formato JSON para luego poder abrirlos y correrlos en otro momento.

Este feature se tuvo en cuenta durante el desarrollo del simulador pero finalmente no se incluyó debido a la complejidad extra que esto hubiera traído al proyecto y a la inexperiencia de los desarrolladores en el diseño de interfaces interactivas.

Posibilidad de exportar simulaciones en formato mp4 o AVI

Las simulaciones creadas por este simulador no pueden verse en ningún otro lado, es decir, se requiere del simulador para poder reproducir los replays. Sería conveniente para el usuario poder exportar los replays a algún formato no propietario de vídeo para poder analizarlos cuando le resulto cómodo.

El problema a resolver para agregar este feature es el tamaño que tendrían los vídeos si se guardara cuadro por cuadro todos los estados de la simulación por lo que habría que encontrar un buen compromiso entre calidad y tamaño.

Mapeo de agentes en un KD-Tree[6]

Lo más costoso al realizar las simulaciones es el sensado y el cálculo de fuerza social. Ya se implementaron optimizaciones que descartan agentes y obstáculos que se encuentran mas allá de una distancia prudencial pero una forma adicional de optimizar este calculo sería agregar los agentes a un árbol KD para obtener más rápidamente los vecinos inmediatos y de esta manera acelerar la simulación

[6] En ciencias de la computación, un Árbol KD (abreviatura de árbol k-dimensional) es una estructura de datos de particionado del espacio que organiza los puntos en un Espacio euclídeo de k dimensiones. Los árboles KD son un caso especial de los árboles BSP.
Un árbol KD emplea sólo planos perpendiculares a uno de los ejes del sistema de coordenadas. Esto difiere de los árboles BSP, donde los planos pueden ser arbitrarios. Además, todos los nodos de un árbol KD, desde el nodo raíz hasta los nodos hoja, almacenan un punto. Mientras tanto, en los árboles BSP son las hojas los únicos nodos que contienen puntos (u otras primitivas geométricas). Como consecuencia, cada plano debe pasar a través de uno de los puntos del árbol KD.

Capítulo 6 - Referencias

- Simulating Dynamical Features of Escape Panic - Helbing, Farkas, Vicsek - Septiembre 2000
- Reinforcement Learning: An Introduction - Richard Sutton, Andrew Barto - The MIT Press
- Q-Learning - Christopher Watkins, Peter Dayan - 1992
- Reinforcement Learning for robot soccer - Martin Riedmiller, Thomas Gabel, Roland Hafner, Sascha Lange - Mayo 2009

Apéndice: Técnico

Formatos de archivo del simulador

Los archivos de entrada se dividen en 2, el archivo de entorno y el agentes. Ambos archivos utilizan la notación JSON[7] para definir los componentes de la simulación

Archivo de agentes

```
{
    "agents": {
        "2": {
            "mass": 90,
            "radius": 0.4,
            "movementType": "idle",
            "position": {
                "x": 30,
                "y": 40
            },
            "velocity": {
                "x": 0,
                "y": 0
            },
            "id": 2,
            "destination": {
                "x": 30,
                "y": 40
            },
             "originalSpawnPoint": "spawn1",
            "exitIdentifier": "exit1"
        }
    },
    "A": 2000,
    "B": 0.08,
    "Kn": 100000,
    "Km": 100000,
    "vDesired": 1.4,
    "tao": 0.5
}
```

Los agentes se definen dentro del diccionario “agents”. Los campos requeridos son los siguientes:

- **id**: El id del agente.
- **mass**: Define la masa en Kg del agente.
- **radius**: Define el radio en m del agente.

---

[7] JSON: Javascript Object Notation http://www.json.org

- **movementType**: Define el método que usará el agente para moverse, existen 5 métodos distintos
  - idle - *El agente permanece quieto.*
  - trivial - *El agente solo tiene fuerza de deseo, por lo que su movimiento siempre será hacia el objetivo.*
  - social - *El agente obedece al modelo de fuerza social.*
  - q-learning - *El agente utiliza Q-Learning para moverse.*
  - q-interpreter - *El agente utiliza Q-Learning para moverse pero no explora, utiliza una matriz previamente aprendida para decidir hacia donde moverse. Al momento de cargar un archivo con agentes con esta característica el simulador preguntará al usuario cual es la matriz a utilizar.*
- **position**: La posición inicial del agente dentro del mundo.
- **velocity**: La velocidad inicial del agente.
- **destination**: El punto hacia el cual el agente querrá dirigirse.
- **originalSpawnPoint**: El spawn point al cual el agente volverá al llegar con éxito a un sumidero.
- **exitIdentifier**: Identifica un sumidero hacia el cual el agente intentará llegar, si se define un identificador para este campo se ignora el campo “destination”.

Archivo de entorno

```
{
    "height": 105,
    "width": 105,
    "polygonObstacles": [
        {
            "body": {
                "npoints": 4,
                "xpoints": [
                    5,
                    45,
                    45,
                    5
                ],
                "ypoints": [
                    95,
                    95,
                    55,
                    55
                ]
            }
        }
    ],
    "circleObstacles": [
        {
            "radius": 4,
            "center": {
                "x": 10,
                "y": 10
            }
        }
    ]

        [...continúa...]
}
```

Dentro del archivo de entorno se pueden definir los obstáculos, los sumideros y los spawn points.

Para crear obstáculos poligonales se deberán agregar dentro de polygonObstacles, los campos necesarios son los siguientes:

- **body**: define los lados del obstáculo
  - npoints: *define la cantidad de vértices.*
  - xpoints: *define la sucesión de puntos en x en los cuales está contenido el obstáculo. Su orden debe coincidir con los puntos de ypoints.*
  - ypoints: *define la sucesión de puntos en y en los cuales está contenido el obstáculo. Su orden debe coincidir con los puntos de xpoints.*

Para crear obstáculos circulares se deberán agregar dentro de circleObstacles, los campos necesarios son los siguientes:

- **radius**: define el radio del obstáculo circular.
- **center**: define el centro del obstáculo.
  - x: *es el punto x del centro del obstáculo.*
  - y: *es el punto y del centro del obstáculo.*

```
{
        [...continúa...]

    "exits": [
        {
            "id": "exit1",
            "preservesTargets": true
            "body": {
                "npoints": 4,
                "xpoints": [
                    45,
                    55,
                    55,
                    45
                ],
                "ypoints": [
                    75,
                    75,
                    85,
                    85
                ]
            }
        }
    ]

        [...continúa...]
}
```

Para agregar sumideros se deberá declarar las mismas dentro del array “exits”, los campos requeridos son los siguientes:

- **id**: es el id del sumidero. Los agentes y los spawn points lo utilizan, por este motivo se debe tener cuidado de no asignar ids duplicados.
- **preservesTargets**: cuando esta propiedad se encuentra en *true* los agentes que lleguen a este sumidero, cuando aparezcan nuevamente en el spawn, conservarán el punto dentro del sumidero al que desean llegar. Si esta propiedad

se encuentra en *false* entonces cada vez que se regenere el agente se determinará un nuevo punto random dentro del sumidero como nuevo destino.

- **body**: define los lados del sumidero
  - npoints: *define la cantidad de vértices.*
  - xpoints: *define la sucesión de puntos en x en los cuales está contenido el* sumidero*. Su orden debe coincidir con los puntos de ypoints.*
  - ypoints: *define la sucesión de puntos en y en los cuales está contenido el* sumidero*. Su orden debe coincidir con los puntos de xpoints.*

```
{

        [...continúa...]

    "spawnPoints": [
        {
            "generatesOnAgentDeletion": true,
            "preservesSpawns": true,
            "lapsusOne": {
                "time": 20000,
                "f": 0.00001
            },
            "lapsusTwo": {
                "time": 30000,
                "f": 0.00001
            },
            "vMin": 1.2,
            "vMax": 1.4,
            "massMin": 65,
            "massMax": 75,
            "radiusMin": 0.2,
            "radiusMax": 0.4,
            "desiredExitId": "exit1",
            "id": "spawn1",
            "body": {
                "npoints": 4,
                "xpoints": [
                    29,
                    31,
                    31,
                    29
                ],
                "ypoints": [
                    15,
                    15,
                    25,
                    25
                ]
            }
        }
    ]
}
```

Para agregar spawn points se deberá declarar los mismos dentro del array “spawnPoints”, los campos requeridos son los siguientes:

- **id**: es el id del spawn point. Los agentes lo utilizan para saber donde deben regenerarse, por este motivo se debe tener cuidado de no asignar ids duplicados.
- **generatesOnAgentDeletion**: cuando esta propiedad se encuentra en *true* los agentes que lleguen a un sumidero y que tengan a este spawn point configurado como originalSpawnPoint se regenerarán en un punto dentro del mismo.
- **preservesSpawns:** cuando esta propiedad se encuentra en *true* los agentes que tengan a este spawn point configurado como originalSpawnPoint se regenerarán siempre en el mismo punto original donde fueron creados. Si esta propiedad se encuentra en *false* entonces los agentes se regenerarán en un punto random dentro de este spawn point.
- **vMin** y **vMax**: determinan un rango de velocidades de deseo al cual deberán ajustarse los agentes generados por este spawn point. El valor se decide de manera aleatoria con distribución normal.
- **massMin** y **massMax**: determinan un rango de masas al cual deberán ajustarse los agentes generados por este spawn point. El valor se decide de manera aleatoria con distribución normal.
- **radiusMin** y **radiusMax**: determinan un rango de radios al cual deberán ajustarse los agentes generados por este spawn point. El valor se decide de manera aleatoria con distribución normal.
- **lapsusOne** y **lapsusTwo**: determinan dos momentos de generación de agentes.
  - time: *cuánto dura cada fase de generación.*
  - f: *qué probabilidad existe de que se genere un agente en cada paso de iteración.*
- **desiredExitId**: define cual es el id del sumidero al cual se dirigirán los agentes creados en este spawn point.
- **body**: define los lados del spawn point
  - npoints: *define la cantidad de vértices.*
  - xpoints: *define la sucesión de puntos en x en los cuales está contenido el* spawn point*. Su orden debe coincidir con los puntos de ypoints.*
  - ypoints: *define la sucesión de puntos en y en los cuales está contenido el* spawn point*. Su orden debe coincidir con los puntos de xpoints.*

Esquema de clases

La clase principal del simulador es Simulator.java. Esta clase se encarga de crear y mantener los distintos elementos que hacen al entorno del simulador: la GUI, el AgentManager, el EnvironmentManager y la simulación actual.

Clases de modelo

- **Agent**: clase que modela los agentes, cada agente tiene las propiedades que lo ubican en el espacio así como una instancia de su método de movimiento.
- **EnvironmentObject**: modela los distintos elementos que se pueden encontrar en el entorno
  - **Obstacle**: modelo los obstáculos
    - **CircleObstacle**: modela los obstáculos circulares.
    - **PolygonObstacle**: modelo los obstáculos poligonales.
    - **SquareObstacle**<Deprecada>: modela obstáculos cuadrados.
  - **Exit**: modela los sumideros
  - **SpawnPoint**: modela los spawn points
- **Lapsus**: modela los lapsus de generación de agentes

Managers

El simulador utiliza dos clases que se encargan de importar/exportar los agentes y entornos así como también de mantener el estado de los mismos durante la simulación

- **Manager**: clase de la que heredan los otros manager, requiere que se implementen métodos para inicializar instancias desde archivo así como también métodos para guardar el estado a un archivo.
  - **AgentManager**: controla los agentes.
  - **EnvironmentManager**: controla el entorno.

Logger

Contiene clases que se encarga de la serialización de datos sobre simulaciones o logs.

- **IBaseLogger**: clase de la que heredan los otros loggers, requiere que se implementen los métodos *openLog()*, *writeState()* y *closeLog()*.
  - **ISimulatorLogger**: clase de que heredan los loggers para guardar replays, requiere que se implementen los métodos *putNewAgent()* y *agentDied()*.
    - **SimulatorLoggerASCII**: guarda los datos que loguea en formato ASCII para leerse mas fácilmente
    - **SimulatorLoggerBinary**: guarda los datos en formato binario para que éstos ocupen menos espacio.
  - **QLearningLogger**: clase que se encarga de loguear la performance de los agentes con Q-Learning cuando termina una simulación.

Movimiento

Contiene clases que modelan las distintas estrategias de movimiento. Para agregar nuevas estrategias se deberá implementar la interfaz *IAgentMovement* y se deberá agregar el tipo nuevo de movimiento en la clase *SimEnvironment* para poder inicializarlo correctamente.

- **IAgentMovement**: Interfaz que deben implementar todos los tipos de movimiento. Define los métodos *updateMovement()* y *getDescription()*.
  - **IdleMovement**: modela el movimiento idle como se describió anteriormente.
  - **SocialForceMovement**: modela el movimiento de fuerza social como se describió anteriormente.
  - **TrivialMovement**: modela el movimiento trivial como se describió anteriormente.
  - **QLearningBaseMovement**: modela el movimiento base de Q-Learning
    - **QLearningMovement**: modela el movimiento de Q-Learning normal, donde el agente aprende y explora
    - **QInterpreterMovement**: modela el movimiento de Q-Learning de un agente que nunca explora sino que se limita a aplicar la politica.