Navegación de Peatones en Bosque de Obstáculos

Badi Leonel, Buchhalter Nicolás Demián y Meola Franco Román

19 de julio de 2016

Índice

1. Introducción **3**

2. Fundamentos **3**

2.1. Contractile Particle Model 3

2.2. Escena 3

2.3. Métodos de Búsqueda Informados 4

3. Implementación **6**

3.1. Adaptación del método de búsqueda 6

3.2. Definición de heurística 7

3.3. Búsqueda limitada en profundidad 7

3.4. Detección de colisiones 7

3.5. Prevención de colisiones 7

4. Resultados **7**

5. Conclusiones **9**

6. Futuras extensiones **9**

7. Referencias **10**

1. Introducción

En el presente trabajo se detallan los fundamentos, la implementación y los resultados obtenidos de la simulación del trayecto de un peatón equipado con un mecanismo de navegación que viaja desde un punto inicial a uno final con el objetivo de evitar colisiones con los obstáculos fijos presentes en la escena. Para la simulación se utiliza una implementación propia del *Contractile Particle Model* y no se utilizaron plataformas de simulación peatonal.

Los fundamentos están inspirados en la temática *Steering Behavior*, propuesto por Mat Buckland en su libro *Programming Game Ai by Example* [2], así como también en los conocimientos de método de búsqueda informados.

El objetivo principal del presente trabajo consiste en crear una heurística que permita ajustar dinámicamente el ángulo del versor de la velocidad deseada en función de las posiciones de los obstáculos. Para lograrlo, se recurre una versión modificada del método de búsqueda informado A* que utiliza la heurística distancia real al objetivo. Cabe destacar que la búsqueda de la ruta se realiza antes de la simulación, ya que todos los obstáculos a evitar son fijos. Por último, se modifica el *Contractile Particle Model* para evitar colisiones en el transcurso la simulación.

2. Fundamentos

En esta sección se detallan todos los fundamentos teóricos que sustentan el presente trabajo así como también los parámetros fijos que se definieron para todas las pruebas realizadas.

2.1. Contractile Particle Model

Como ya se mencionó, la implementación propuesta utiliza el modelo de dinámica peatonal en situación no competitiva *Contractile Particle Model* [1]. Este modelo se basa en simular a un peatón como una partícula de radio variable. De esta forma, este radio representa el estado de relajación del agente.

2.2. Escena

A continuación se listan los parámetros que permanecen fijos en todas las pruebas que se explicarán en el análisis de resultados. En cuanto al agente del *Contractile Particle Model* que recorrerá la escena y deberá inteligentemente esquivar los obstáculos fijos, los parámetros son los siguientes:

- $r_0 = 0{,}25m$
- $r_{min} = 0{,}15m$
- $r_{max} = 0{,}32m$
- $v_d^{max} = 1{,}55\frac{m}{s}$

donde r_0 es el radio inicial del agente y el cual puede variar en el intervalo $[r_{min}, r_{max}]$ y v_d^{max} es la velocidad deseada del agente.

Se definieron además los valores de los siguientes parámetros fijos del modelo

- $\tau = 0{,}5s$
- $\beta = 0{,}9$

donde τ se utiliza para el cálculo de la evolución temporal del radio del agente y β para el cálculo de $|v_d|$.

Se decidió utilizar una habitación cuadrada de $25m$ de lado, donde se colocan aleatoriamente 300 obstáculos fijos, todos del mismo radio $r_{obstaculo} = r_0$. Como se puede ver en la Figura 1, la posición inicial del agente está en la esquina inferior izquierda y la destino está en la esquina superior derecha. Se prevee no colocar obstáculos muy cerca de la posición inicial y final del agente. Sin embargo, no se prohíbe la superposición de obstáculos, de forma de permitir la formación de paredes de obstáculos.

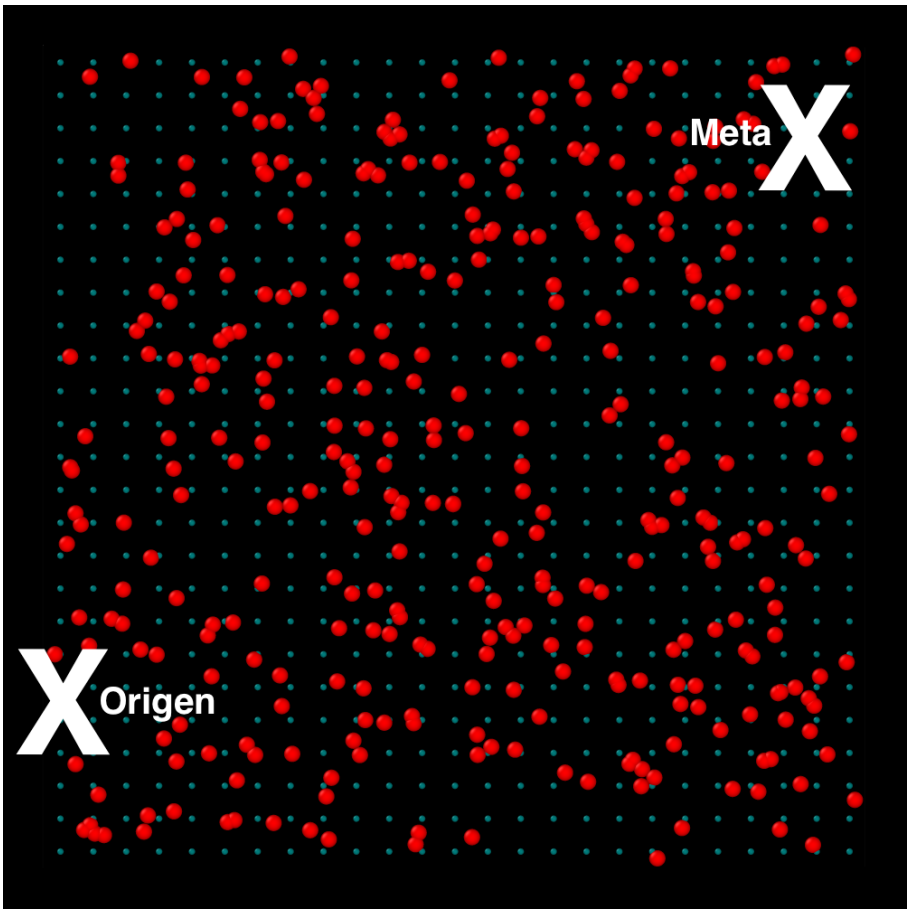

Figura 1: Ejemplo de escena con 300 obstáculos fijos distribuidos aleatoriamente. La posición inicial del agente (origen) se encuentra en la esquina inferior izquierda y la posición final del agente (meta) se encuentra en la esquina superior derecha.

2.3. Métodos de Búsqueda Informados

El objetivo de un método de búsqueda es encontrar un camino entre el estado inicial y el estado final. Los métodos de búsqueda desinformados sólo utilizan el costo de búsqueda $g(n)$, de forma que se obtiene

una búsqueda a ciegas y resulta muy ineficiente. Por lo tanto, es necesario agregar valores que estimen cuánto falta para alcanzar el objetivo.

A estos métodos de búsqueda se los denomina informados y forman parte de la familia de algoritmos *Best First* (el primer nodo a explorar es el de menor costo). Se define así una heurística $h(n)$ como el costo estimado de la ruta más barata que une el estado del nodo n con un estado meta.

El método de búsqueda informado utilizará la función de mínimo costo de ruta $f(n) = g(n) + h(n)$ donde para la Figura 4

- $g(n)$: es el costo de la ruta que va del nodo inicial hasta el nodo n
- $h(n)$: estimación del mínimo costo de ruta que va de n al nodo objetivo
- $f(n)$: costo estimado de la solución más barata que pasa por n

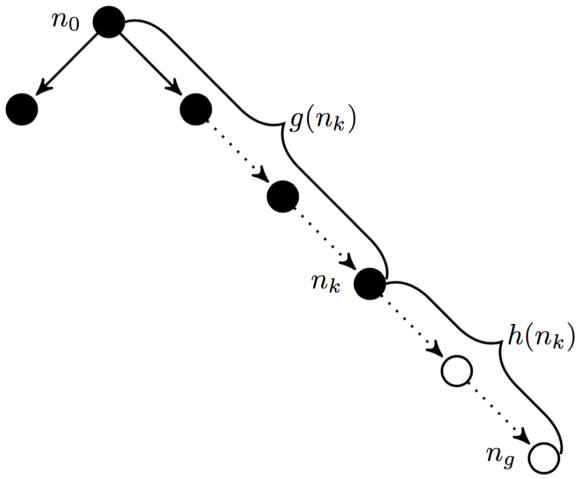

Figura 2: Diagrama de aplicación del mínimo costo de ruta para un árbol de búsqueda.

Habiendo definido la función f, se explica en una sucesión ordenada de pasos el algoritmo de búsqueda informado A*:

1. Crear un árbol de búsqueda, T_r, que sólo esté formado por el nodo inicial n_0. Insertar $n0$ en la estructura FRONTERA.
2. Crear el conjunto EXPLORADOS, inicialmente vacío.
3. Si FRONTERA está vacía, el algoritmo termina con falla.
4. Extraer el primer nodo de FRONTERA e insertarlo en EXPLORADOS. Llamar a este nodo n.
5. Si n es el nodo objetivo, salir del algoritmo y devolver la solución, que está formada por el camino definido por los arcos que llevan de n a n_0 en el árbol T_r.
6. Expandir el nodo n y generar el conjunto de nodos sucesores M. Incorporar los elementos de M como sucesores de n en el árbol Tr, creando los arcos correspondientes entre n y cada uno de los nodos de M.

7. **Reordenar FRONTERA de forma que primero estén los nodos de menor f.**

8. Ir al paso 3.

El comportamiento de A* está directamente relacionado con la heurística h utilizada. Es necesario mencionar que una heurística es admisible si nunca sobreestima el costo real. De esta forma, si h es admisible entonces $f(n)$ nunca sobreestima el costo real de la mejor solución que pasa por n. Sólo con h admisible, el algoritmo A* es completo y admisible, de forma que se garantiza siempre encontrar el camino óptimo al objetivo, si es que existe uno.

En resumen, A* actúa sobre un grafo donde el objetivo es encontrar el camino de costo mínimo, eligiendo para expandir siempre el nodo de menor f. El problema es que al actuar sobre un grafo, el espacio de búsqueda debe ser discreto. Es por eso que se requiere hacer una adaptación de la escena para poder discretizar posiciones de la habitación.

3. Implementación

En esta sección se detallan las modificaciones que se realizaron al método de búsqueda informado y al modelo de dinámica peatonal para obtener mejores resultados en la simulación propuesta.

3.1. Adaptación del método de búsqueda

El primer problema encontrado consiste en implementar A* en un sistema continuo: es necesaria la discretización del sistema. Para resolver este problema se definió el concepto de *waypoint*.

Un *waypoint* es un punto (x, y) en la escena por donde el agente podrá desplazarse. Estos serán los **nodos** por donde iterará el método de búsqueda. En el caso de este trabajo, el espacio a simular cubre toda la escena, pero en casos más complejos este sistema es flexible para tomar la forma que se desea. Por ejemplo se pueden definir *waypoints* sólo por ciertas áreas de un plano que son transitables. Cabe aclarar que en este modelo, los *waypoints* pueden estar colocados en cualquier posición de la escena. De esta manera se pueden colocar a diferentes distancias entre sí, siendo más potente que una simple grilla de casilleros.

Se decidió armar una grilla de *waypoints* sobre la habitación donde los *waypoints* están separados unos de otros por una distancia W_d. Otro de los parámetros que se pueden variar en el modelo es el grado de conectividad entre los *waypoints*. Este parámetro se denomina ventana y representa el número de vecinos aledaños del *waypoint*. Para un *waypoint*, la ventana es siempre cuadrada y se llama W_v a la longitud del lado de la ventana. De esta forma, si $W_v = 3$ la ventana contiene a los 8 vecinos aledaños. Si $W_v = 5$, la ventana contiene a los 24 vecinos.

El algoritmo de *pathfinding* de la implementación consiste en una modificación de A* donde el grafo a recorrer es el formado por los *waypoints* y sus conexiones. El conjunto de vecinos de la ventana del nodo n constituye a los nodos sucesores del mismo. El objetivo es obtener una lista de *waypoints* por donde el agente tiene que transitar desde el nodo inicial hasta alcanzar el nodo meta. La lista de *waypoints* obtenida se define como un conjunto de sub-objetivos que tiene que ir recorriendo el agente. Así, para

cierto momento dado, el versor velocidad del agente tiene como dirección a la línea que se forma entre la posición actual del agente y el primer sub-objetivo.

3.2. Definición de heurística

La heurística $h(n)$ es la distancia real que se tiene al objetivo (distancia en línea recta al nodo meta). Esta heurística es admisible ya que no sobreestima la solución y el costo es la distancia que ya se recorrió. De esta forma, A* será completo y admisible, y siempre encontrará el camino óptimo al objetivo si es que existe uno.

3.3. Búsqueda limitada en profundidad

Una modificación implementada fue la limitación en profundidad de la búsqueda de la solución óptima. El árbol de búsqueda tiene un máximo de profundidad L y en el borde del límite se encuentra un nodo meta sub-óptimo. Una vez alcanzado el sub-óptimo, se reinicia la búsqueda desde allí.

3.4. Detección de colisiones

Otra de las modificaciones al algoritmo original es la detección de posibles colisiones en caminos. Para detectarlo se genera un *raycast*, es decir un rayo en línea recta desde un *waypoint* origen hacia otro waypoint *destino*. Si el trayecto del *raycast* impacta sobre un obstáculo, entonces ese camino es descartado del árbol de búsqueda. Así se evitan muchas colisiones innecesarias y caminos sin salida.

3.5. Prevención de colisiones

Los *steering behaviors* son comportamientos que se le pueden aplicar al agente para que se comporte de una manera determinada. Con cada *steering behavior* se obtiene un vector velocidad normalizado, que luego sumado, resulta en un vector velocidad que representa el comportamiento del agente en ese instante.

El objetivo entonces es evitar los choques del agente con obstáculos que tiene en sus inmediaciones. Así, utilizando una partícula de radio $r_p = 1{,}1r$ se calculan todos los posibles solapamientos con obstáculos. Para cada solapamiento detectado se genera un versor director en dirección contraria a la del agente, para así evitar este obstáculo. Al final, todos los versores se suman y se obtiene un versor sumarizado que representa la dirección a transitar para no colisionar con los obstáculos.

4. Resultados

En la Figura 3 se puede ver cómo, a medida que se aumenta el número de obstáculos fijos en la escena, la distancia recorrida promedio por el agente es mayor. Del mismo modo, el error (desvío estándar) de la distancia también aumenta.

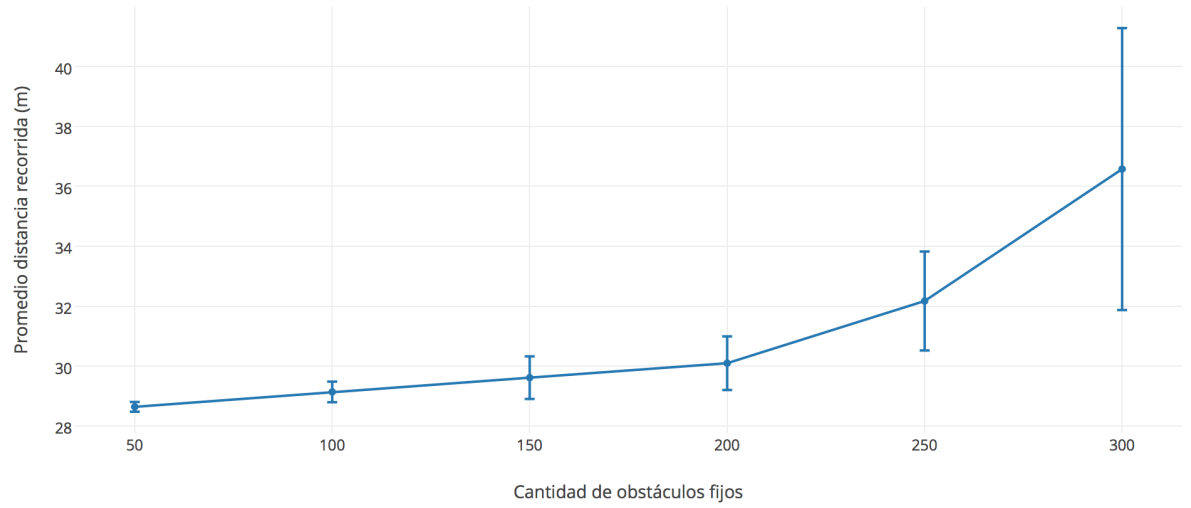

Figura 3: Gráfico de la distancia recorrida promedio para distintos valores de obstáculos fijos.

Como se puede ver en el la Figura 4 con el valor $W_s = 1{,}5$ se obtuvieron los mejores resultados ya que el tiempo promedio de la simulación resultó ser el menor de todas las pruebas, sin importar el valor del lado de la ventana W_v. Allí también el error (desvío estándar) resultó bajo.

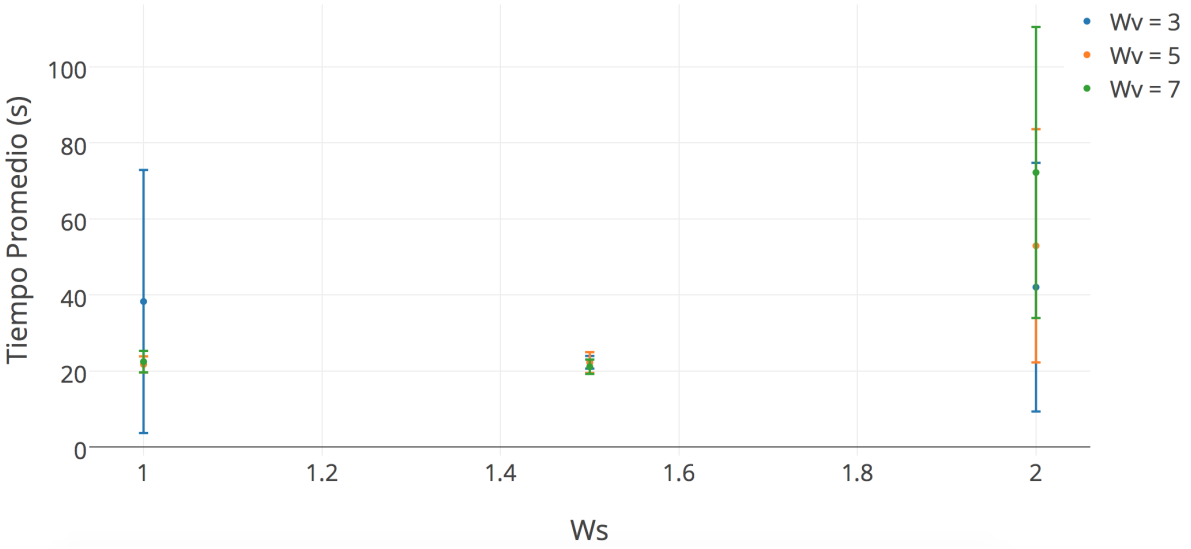

Figura 4: Gráfico del tiempo promedio de simulación para distintos valores de W_s y W_v.

Para $W_s = 1$, el desvío estándar resultó más alto para un tamaño de ventana menor $W_v = 3$. Esto se explica por el hecho de que a una menor cantidad de vecinos, los movimientos que puede realizar el agente son acotados en comparación con tamaños de ventana mayores, donde los errores resultaron bajos. Es por eso que al tener menos movimientos se pierde la posibilidad de tomar caminos más cortos.

De las pruebas realizadas, $W_s = 2$ produjo los peores resultados debido a que, al ser los caminos más largos, aumentaba la posibilidad de ser descartados del método de búsqueda por el hecho de una colisión con un obstáculos. Y este hecho se repite para todos los valores de W_v probados. Además, con este valor de W_s muchas de las pruebas realizadas no cumplieron con el objetivo de llegar al goal.

5. Conclusiones

En primer lugar, se logró obtener un modelo capaz de navegar por un tablero con gran densidad de obstáculos obteniendo más de un 80 % de efectividad, es decir, la cantidad de veces que alcanza el agente alcanza el nodo meta.

Además este modelo de navegación resultó altamente personalizable, permitiendo modificar parámetros como la posición y cantidad de los *waypoints* así como también la utilización de *steering behaviors* distintos.

Concluimos además que a valores mayores de W_v, se pueden encontrar caminos diagonales que, claramente, tendrán una distancia menor y por lo tanto serán considerados con mayor prioridad en el método de búsqueda informado.

6. Futuras extensiones

Uno de los principales componentes del algoritmo al que se le pueden introducir mejoras es la definición de la heurística que utiliza la modificación del método de búsqueda informado A*. Actualmente la heurística utiliza la distancia al nodo meta y descarta los posibles caminos obstruidos por obstáculos que, de transitarlos, generarán colisiones. Este punto podría mejorarse haciendo que los posibles choques con obstáculos fijos no sean causa de descarte del camino sino que le solamente se pondere a ese trayecto con un costo mayor en el árbol de búsqueda.

Una extensión relacionada a lo mencionado anteriormente es la forma en que se realiza el cálculo de colisiones con la utilización del *raycast*. Esto se podría implementar proyectando varios *raycast* con distintos radios. Cabe destacar que en el modelo de simulación peatonal utilizado (*Contractile Particle Model*) el radio de una partícula es variable y proporcional a su velocidad. De esta manera, el rayo con mayor apertura puede detectar una colisión, mientras que el rayo más pequeño no lo detecta. Si éste es el caso, se puede, en vez de descartar el camino, hacer que ese camino tenga un costo mayor. Así se aprovecharía mejor la variación del radio del modelo.

Por último, también se le pueden agregar fácilmente otros comportamientos al modelo para que el agente tenga un movimiento más complejo. El comportamiento a implementar va a depender de cómo se desea que se comporte el agente. Por ejemplo, si se quiere que un grupo de agentes siga a un líder, se puede implementar el comportamiento *Follow the leader*. Este comportamiento puede ser útil, por ejemplo, cuando se quiere simular un grupo numeroso de personas circulando en bicicleta por una ciudad siguiendo a otro grupo de personas (masa crítica).

7. Referencias

[1] Gabriel Baglietto and Daniel R Parisi. Continuous-space automaton model for pedestrian dynamics. *Physical Review E*, 83(5):056117, 2011.

[2] Mat Buckland. Chapter 3: How to create autonomously moving game agents. *Programming Game AI by Example, Wordware Publishing*, 2005.