II SMIO/V TLAIO, Acapulco, Guerrero, México, 14-16 de noviembre de 2013

UN SISTEMA DE AGENTES AUTÓNOMOS PARA ESTUDIOS DE MODELOS DE COMBATE/PARA PRESENTACIONES EN EL SEGUNDO CONGRESO DE LA SOCIEDAD MEXICANA DE INVESTIGACIÓN DE OPERACIONES / QUINTO TALLER LATINOIBEROAMERICANO DE INVESTIGACIÓN DE OPERACIONES

Otto Hahn Herrera
Universidad Nacional Autónoma de México,
Facultad de Ciencias, México D.F.
otto.hahn.herrera@gmail.com

Emiliano Valdés Guerrero
Universidad Nacional Autónoma de México,
Facultad de Ciencias, México D.F.
emailliano@gmail.com

Aarón Martín Castillo Medina Herrera
Universidad NacionalAutónma de México
Facultad de Ciencias, México D.F.
acuarium_329@hotmail.com

Augusto Cabrera Becerril
Universidad NacionalAutónma de México
Facultad de Ciencias, México D.F.
acb@ciencias.unam.mx

Juan Alberto Camacho Bolaños
Universidad NacionalAutónma de México
Facultad de Ciencias, México D.F.
spartan3004@gmail.com

Palabras Clave: agentes, combate, python.

RESUMEN

La modelación basada en agentes autónomos consiste en la generación de diferentes individuos con características únicas que interaccionan entre ellos y con su ambiente. Esta técnica es conveniente para el estudio de diferentes problemas en los que el factor humano debe ser tomado en cuenta.
En este trabajo presentamos un sistema de agentes que permite la simulación de combate. Nuestros agentes se situan en un campo de batalla. Tienen una cierta personalidad, motivaciones y objetivos, reglas de compartamiento y reglas para modificar su comportamiento. Su dinámica estará definida a partir

de sus interacciones (locales) con el entorno. Con nuestros agentes, sujetos a rutinas de movimiento, visión y combate podemos simular situaciones de guerra con distintos parámetros para cada agente o para grupos de agentes. Este sistema permite la investigación de diferentes conceptos de historia y táctica militar. Consiste de un programa para los cálculos y una interfaz gráfica de usuario para analizar los resultados de manera visual y generar distintos análisis

INTRODUCCIÓN

La modelación basada en agentes autónomos es una metodología que permite estudiar una gran cantidad de fenómenos complejos. Su basta aplicabilidad ha provocado que actualmente se encuentre en voga en ambientes científicos e informáticos.

Con los sistemas de agentes se construyen mundos virtuales donde se recrean los fenómenos, lo cual permite: un análisis completo, la posibilidad de predecir resultados en escenarios diversos, el estudio dinámico del objeto de estudio, hacer modificaciones en tiempo real y estudiar los efectos de distintas configuaraciones iniciales entre muchas otras cosas. Se genera un escenario artificial que reproduce con suma precisión casi cualquier fenómeno que nos interese estudiar.

Existen distintos programas que han empleado y desarrollado sistemas de agentes. Nuestro trabajo ha retomado elementos de:

-“Netlogo: un ambiente de modelación progamable de múltiples agentes” (Wilensky, U.,1999)
Con una amplia biblioteca de modelos y un interfaz que permite construir los propios”.

-Artificial War (Ilachinsky, 2004)

En este trabajo presentamos un sistema de agentes que permite la simulación de combate. Nuestros agentes se situan en un campo de batalla. Tienen una cierta personalidad, motivaciones y objetivos, reglas de compartamiento y reglas para modificar su comportamiento. Su dinámica estará definida a partir de sus interacciones (locales) en el entorno. Con nuestros agentes sujetos a rutinas de movimiento, visión y combate podemos simular situaciones de guerra con distintos parámetros para cada agente o para grupos de agentes.

.

Programa de SImulación de COmbate y Sociedad (PSICOS)

PSICOS es una biblioteca de modulos escritos en python puro para generar y manejar agentes. Decidimos utilizar Python por que es un lenguaje de alto nivel, nos permite concentrarnos en los algoritmos y, no tanto en los detalles de la implementación. Además es un lenguaje interpretado y orientado a objetos, dos características indispensables para implementar nuestros modelos. El uso de Python puro nos permitirá construir una aplicación autosuficiente y autocontenida, no requiere de la instalación de módulos externos, sólo el interprete de Python.

El sistema consiste de un programa para los cálculos y una interfaz gráfica de usuario para analizar los resultados de manera visual. Actualmente se tiene implementado un sistema de combate, diferentes algoritmos de movimiento y un algoritmo para permitir la visión de los agentes.

En el desarrollo del combate intervienen 3 acciones principales: observación, disparo y movimento. Cada uno de éstos elementos necesitará de parámetros que, por convención y, a reserva de que se indique lo contrario, tendrán una escala de 0-10, siendo 0 lo peor y 10 lo mejor, dependiendo en cada parámetro.

La batalla sucede cuando todos los agentes **(vivos)** ejecutan dichas acciones al menos una vez, En términos de los algoritmos una batalla será un ejecución , de modo que para entender mejor el funcionamiento del sistema, optamos por definir todo el esquema en función de “batallas”.

ACCIONES

La observación es la forma en que el agente de averigua qué es lo que tiene a su alrededor, es decir, mediante la observación el agente es capaz de saber si en una región del terreno hay aliados, enemigos o espacios vacios. Sólo los agentes vivos pueden realizar esta acción.

El pseudocódigo es:

Parámetros:

- **camuflaje_deseado -> valor numérico que indica qué tan bueno es un agente de "esconderse" ante los enemigos (escala de 0-10).**
- **lista_vision -> una lista que contiene las celdas que "verá" el agente. Esta lista es única para cada agente.**

Rutina:

```
for each agente in ejercito:
    for each espacio in espacios_adyacentes(agente):
        if  espacio.camuflaje <= camuflaje_deseado:
            agente.lista_vision.agrega(espacio)
```

La cantidad de espacios adyacentes depende del atributo "rango de visión" propio de cada agente, este pertenece a un radio de visión mayor o menor y, en consecuencia , más o menos elementos agregados a la lista_vision de cada agente.

Para simular el hecho de que un agente enemigo puede escapar a la visión del agente que mira, hemos añadido el atributo "camuflaje"; si el agente enemigo se ha camuflajeado bien en el escenario **(esto se indica por el atributo camuflaje_deseado)** entonces no será visto y por ende no será agregado a la lista_visión del enemigo.

La siguiente acción a ejecutar es el disparo, con esta acción se pretende eliminar al enemigo. Tendrán oportunidad de disparar todos los agentes vivos, no todos podrán acertar en el blanco. Esto depende de los parámetros de el terreno, el arma y la puntería de los agentes. En caso de acertar sólamente se podrá eliminar a un enemigo por turno. Para eliminar a otro se tendrá que esperar a que se vuelva a ejecutar la acción de disparo.

Para simplificar el modelo tomamos las siguientes restricciones:

- Cada agente tiene por default un arma con cargador lleno y municiones extra **(esta condición puede variar si así se desea)**.
- No existe el fuego amigo.
- Cada agente tendrá sólamente un número determinado de oportunidades para disparar **(éstas se pasan como parámetro).** Podrá realizar solamente un disparo por oportunidad.
- La recarga del arma cuenta como una oportunidad.
- No se pueden recoger armas y/o munición de los enemigos.
- Las oportunidades del agente se acabarán automáticamente si el agente ya no tiene balas **(tanto en el arma como de reserva)**.

La rutina a grandes rasgos es: un agente mira en su lista de visión hasta encontrar un enemigo, si no encuentra nada en su lista de visión significa que no había enemigos a su alrededor o que éstos se camuflajearon muy bien; en este caso termina la acción de disparar. Cuando observa al menos un enemigo en su lista de visión dispara; primero es revisar si aún tiene balas, en el caso de que no tenga termina su oportunidad de disparar. Si aún tiene balas entonces comienzan sus oportunidades de disparar.

Para poder emular mejor un disparo al enemigo se toman en cuenta los siguientes elementos:

- Ataque del que hace el disparo **(escala de 0-10)**.
- Efectividad **(que el arma esté en buenas condiciones, no se haya sobrecalentado; escala de 0-10)** y alcance del arma **(escala real, ejemplo 500m o 1000m, dependiendo del arma)**.
- Defensa del que recibirá el disparo **(escala de 0-10)**.
- Blindaje del que recibirá el disparo **(quien el disparo cuenta con protección para ser inmune al proyectil; escala de 0-10).**
- Blindaje del espacio donde se encuentra el que recibirá el disparo **(no confundir con camuflaje, el camuflaje es sólo al momento de observar; escala de 0-10)**.

El pseudocódigo explica mejor cómo se emplean estos factores. Si el agente logra dar en el blanco, el enemigo "muere" y automáticamente el agente que dispara sale de su rutina, termiandoasí la ejecución de la acción.

Parámetros:

ATACANTE:

- **lista_vision -> lista del agente que va a disparar (se "llena" en el método de observación).**
- **ataque -> el ataque del atacante (es un atributo del agente).**
- **oportunidades: el número de oportunidades que tiene para disparar cada agente (es un atributo del agente).**

ARMA:

- **efectividad -> efectividad del arma (es un atributo variable del arma).**
- **alcance -> el alcance del arma (atributo no variable del arma).**

ATACADO:

- **defensa -> la defensa del atacado (atributo del agente).**
- **blindaje -> el blindaje que tiene el atacado (atributo del agente).**
- **blindaje_espacio -> el blindaje que tiene el espacio donde se encuentra el atacado.**

Rutina:

```
for each agente in ejercito:
    objetivo = selecciona_enemigo(agente.lista_vision)
    if objetivo == null:
      terminaEjecucion()
    else:
        //Si el agente tiene balas entonces procederá a dispararle al enemigo seleccionado.
        if agente.tieneBalas():
          if oportunidadesAgotadas():
             terminaEjecucion()
          else while  tengaOportunidades():
              //Cada vez que se necesite recargar el arma se perderá una oportunidad.
              if necesarioRecargar():
                recargaArma()
                oportunidades --
               else:
                  // Aquí es donde se utilizan todos los elementos que se encuentran en los parámetros; se define el ataque del que dispara y la defensa del que va a recibir el disparo, como se puede ver es un enfrentamiento de fuerzas donde al final se decide quién gana mediante un número aleatorio.
                  ataque_total = ataque + efectividad + alcance
                  defensa_total = defensa + blindaje + blindaje_espacio
                  fuerza_total = ataque – defensa
                  evento_azaroso = aleatorio (0, fuerza_total)
                 // Aquí es donde entra la "incertidumbre del disparo"
                  if evento_azaroso >= (fuerza_total / 2):
                       muereObjetivo()
                  else:
                      oportunidades --
        else:
            terminaEjecucion()
```

La ultima acción que se ejecuta es el movimiento; ésta se realiza independientemente del disparos. Sólo los agentes vivos se puedan mover. En el movimiento se toman en cuenta una serie de reglas; cada regla considera lograr un movimiento diferente del agente **(de modo que todas las reglas regresan coordenadas)**, indicando cada una de éstas hacia dónde se debería mover el agente; al final se toman todas las coordenadas **(producto de las reglas)** y se realiza un promedio. Las coordenadas resultantes generan el desplazamientos del agente. El movimiento es válido, si y sólo si, la celda destino del agente no está ocupada por algún otro **(ya sea aliado o enemigo)**, en este caso el agente conserva su posición previa. Un agente puede ocupar unaa celda donde haya muerto un otro.

Las reglas de movimiento reproducen el compartamiento de boids. Son:

1. Mantener cohesión **(acerca a agentes del mismo bando)**.
2. Mantener separación mínima **(alejar a agentes del mismo bando)**.
3. Emparejar velocidades **(empareja la velocidad del agente con las de los demás del mismo bando, entiéndase en este caso por velocidad el número de espacios que puede moverse un agente en cada batalla)**.
4. Ir hacia un objetivo **(llevar a los agentes hacia un objetivo en común, donde el objetivo no es otra cosa que unas coordenadas que se pasan como parámetro)**.
5. Alejarse de fronteras **(aleja a cualquier agente de las orillas del campo de batalla)**.
6. Mover blindaje **(mueve a los agentes de cualquier bando a aquellos lugares donde exista más blindaje, es decir, aquellos espacios en los cuales su atributo blindaje_espacio tenga valor más alto)**.

Parámetros:

- **lista_vision -> visión que se necesita del agente que se desea mover; esta lista sirve para las reglas 1, 2, 3 y 6.**
- **coords_origen -> las coordenadas (de la forma (x,y)) donde se encuentra el agente.**
- **coord_obj -> las corrdenadas del objetivo para la regla 4.**
- **dimensiones_mapa -> las dimensiones (coordenadas) del campo de batalla para la regla 5**

Rutina:

```
for each agente in ejercito:
    coords1 =  regla1 (lista_vision)
    coords2 = regla2 (lista_vision)
    coords3 = regla3 (lista_vision)
    coords4 = regla4 (coord_obj)
    coords5 = regla5 (dimensiones_mapa)
    coords6 = regla6 (lista_vision)
   coords_destino = (coors1 + coords2 + coords3 + coords4 + coords5 + coords6) / 6
    if ! coords_destino.Ocupada():
       mueveAgente(coords_origen,coords_destino)
    else:
        terminaEjecucion()
```

Cada espacio en el campo de batalla contiene una lista, tal que, cada vez que "muera" un agente en determinado espacio sea agregado en esta lista. Al finalizar cada batalla se muestran las casillas en donde se ha eliminado agentes; con el objetivo de realizar un análisis posterior a la batalla tanto más a profundidad tanto del terreno, como de las estrategias implementadas.

Algoritmo de busqueda A-estrella

En **PSICOS** tenemos dos tipos de agente de combate, el soldado, que se mueve siguiendo el algoritmo de los **boids** de Craig Reynolds, y el lider o Scout, que se mueve siguiendo una ruta trazada usando un algoritmo de busqueda de rutas. La idea principal es que el Scout tiene más información acerca del terreno que el resto de los soldados, de tal manera se vuelve el lider de un grupo.

La filosofía de la busqueda de rutas.

Decidimos utilizar un algorítmo tipo "best-first", una modificación del algorítmo de Djikstra, el A*. El algorítmo esta pensado para ser aplicado a una gráfica con pesos, requerimos definir nodos en nuestro mapa, los nodos estarán ubicados en el centro de las celdas del mapa.

Busqueda de rutas en PSICOS

Hemos implementado el algoritmo A* en python, para ello definimos dos funciones auxiliares, la métrica y la heurística. En esta etapa, tanto la métrica como la heurística son las más sencillas posibles, usamos la métrica del taxista:

$$\delta (x,y) = (x_1-x_2)^2 + (y_1-y_2)^2$$

La función de peso heurístico es la distancia entre la posición actual y el nodo objetivo, ponderada por un factor D, que representa el costo mínimo del cruce por los nodos en una vecindad de Moore.

$$D(pos)\ \delta\ (pos,obj)$$

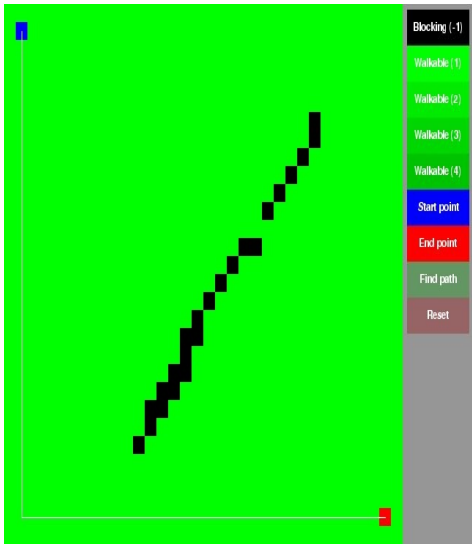
Nesesitaremos crear dos listas de nodos, la lista "abierta" donde buscaremos candidatos para nuestra ruta, y la llista "cerrada" donde pondremos aquellos que ya han sido considerados. El algorítmo produce una busqueda en una vecindad por el nodo con el mejor "marcador", en la dirección del nodo marcado como objetivo. Para determinar dicho "marcador", definimos dos funciones auxiliares. La función **G** que mide el costo del movimiento desde el nodo inicial hasta el nodo objetivo. La función **H** es una estimación heurística del costo de movimiento desde el nodo actual y el objetivo. El marcador queda dado en términos de las funciones **H** y **G:**

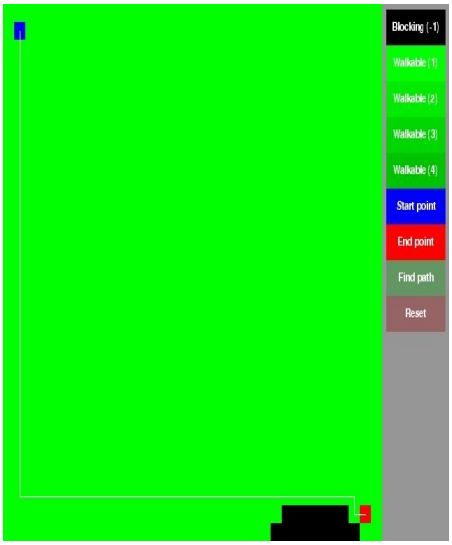
$$F = H + G$$

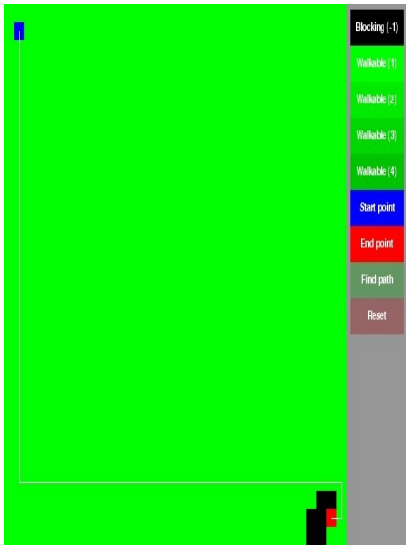
Pseudo código

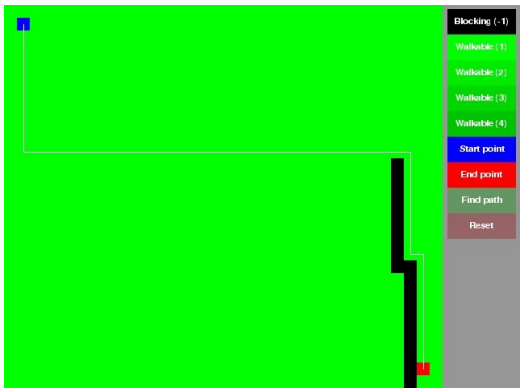
1. Mueve el nodo inicial a la lista de nodos abiertos.
2. Repetir lo siguiente.
   - (a) Busca por el nodo con el menor marcadoren la lista abierta. Renómbralo como "nodo_actual".
   - (b) Ponlo en la lista cerrada.
   - (c) Para los nodos vecinos de "nodo_actual"
     - (i) Si no es posible atravezar el nodo o ya esta en la lista cerrada, ignóralo. En otro caso
     - (ii) Si no está en la lista abierta, ponlo en la lista abierta y calcula su "marcador".
     - (iii) Si ya esta en la lista abierta, checa si el valor de **G** en este nodo es el mínimo.
   - (d) Detenerse si :
     - (i) Se ha puesto el nodo objetivo a la lista cerrada, (en este caso la ruta ha sido hallada)
     - (ii) Se ha fallado en hallar el nodo objetivo y la lista abierta está vacía. (en este caso no hay ruta)
3. Guarda la ruta. Reordena la lista de nodos abiertos inviertiendo el orden.

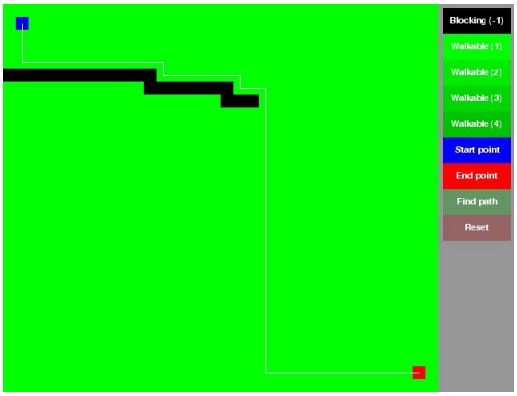
Acontinuación presentamos imagenes de la implementavción en Python del algorítmo A*, usando el módulo PyGame, para visualizar.











Ejemplos del trazado de rutas con el algorítmo A*, tratando de evadir obstáculos.

Ejemplo en el que no es posible hallar una ruta.

Las capacidades potenciales del sistema son: establecer y analizar algoritmos para simular comunicación uni y bidireccional, el uso de diferentes objetivos de manera dinámica, el uso de sistemas de clasificación para evolución de estrategias y técnicas de recorte de árboles de decisión para estudios de planeación y estrategias. Además se proyecta el desarrollo e implementación de diferentes indicadores visuales y numéricos que permitan hacer análisis detallados.

Aplicaciones.

La aplicación inmediata es el análisis de combate terrestre, usamos un conjunto de distintas técnicas de modelación tipo "*Vida Artificial*", lo cual nos permitirá estudiar distintos comportamientos emergentes en el combate terrestre. Utilizamos agentes autónomos para modelar los comportamientos y personalidades individuales, en lugar de considerar unidades de combate homogéneas como en los modelos lanchesterianos basados en ecuaciones diferenciales.

Otra aplicación es el estudio de la dinámica espacio-temporal de las epidemias. La motivación es muy parecida, se trata de un fenómeno donde las interacciones entre individuos son insoslayables, por lo que los modelos basados en ecuaciones diferenciales, que consideran las poblaciones relativamente homogéneas compartimentadas, en las que la epidemia se extiende de manera homogénea siguiendo la ley de Fick. En cambio, el enfoque de la modelación basada en agentes permite estudiar poblaciones heterogéneas, en las que los individuos se mueven de manera casi aleatoria.