Implementación de Risk con algoritmo de Minimax.

Prieto Larios, Estefanía,[♠]
Galicia Mendoza, Fernando Abigail,[♠]
Galván Gámez, Edwin Antonio.[♠]
[♠]Facultad de Ciencias, Universidad Nacional Autónoma de México,
Circuito Exterior, C.U., A. Postal 70-264, 04510 México D.F., México.

email: *estefaniaprieto@ciencias.unam.mx*
email: *fernandogamen@ciencias.unam.mx*
email: *g.antonio@ciencias.unam.mx*

November 17, 2014

Abstract

A diferencia de los juegos de apuesta, donde el jugador se pregunta *"¿Cuál es la mejor jugada para ganar un juego?"* y así poder ser el dueño de un premio (generalmente un incentivo monetario), es bien sabido en la teoría de juegos la motiva escenarios tales cómo el ajedrez, go, gato, etc. No existe tal pregunta, si no, ésta se replantea una expresión de la forma *"¿Existe una mejor forma de jugar en tal escenario?"*.
Por lo cuál se propone un modelo de Inteligencia Artificial para una versión acotada del juego **Risk** basado en minimax, con base en estrategias muy complicadas implementadas por un experto, hasta muy básicas diseadas por un novato en el juego.

1 Introducción.

La teoría de juegos se puede interpretar cómo: *"el estudio de las decisiones interdependientes"* [4]. Con lo cuál debemos observar que a diferencia a los juegos de azar, en teoría de juegos se plantea que estrategia resulta mejor em contra del oponente y que no sea producto de una probabilidad. Por lo cuál en general no hay un mejor o peor juego para todos los juegos.

Para este proyecto se ha optado por la estrategía de ***Minimax***, creada por el matemático *John von Neumann*. El cuál es un algoritmo para minimizar una estrategía en juegos con información perfecta. [6] en el cual se espera que el se tenga cómo resultado el *"mejor"* movimiento para cada jugador suponiendo que el contrincante realicé la jugada menos favorable.
De tal forma que se espera que el juego siempre termine con tres posibles resultados, que el $jugador_1$ gane, que haga lo propio el $jugador_2$ o que el juego termine en empate.

En el capitulo 8 se hablará a cerca de la propuesta que hemos hecho para éste juego implementando la idea del algoritmo ***Minimax***.

2 Juegos con información imperfecta.

En este tipo de juego, en el que ambos jugadores conocen durante toda la duración de la partida el tablero de juego, se maneja entonces cómo un juego de información perfecta, de tal forma que en todo momento puede modelar una estrategia que resulta ser la *"optima"* para una configuración dada en el tablero [1].
Esperemos que el lector crea y se de cuenta que buscar una implementación para nuestro caso, resulta en un juego de información imperfecta, pues el agente no podrá estar al tanto del tablero y al mismo tiempo estar generando una estrategia para vencer a su rival, por lo a continuación veremos que un juego con información imperfecta es aquel que, alguno de los jugadores desconoce la estrategia que ha seleccionado su contrincante y además que el jugador desconoce la posición en la que se encuentra dentro del conjunto de vértices que le pertenecen [2].

Por lo que de ésta forma, el lector se puede ir generando una idea de cuales van a ser las características del agente para ésta implementación.

3 Descripción del agente.

Con el contexto que hemos adquirido tempranamente a esta altura, entonces, es fácil predecir que el modelo para este agente será un agente basado en modelos.

Dado que los agentes basados en modelos son eficientes al momento de manejar visibilidad parcial [5] *"He aquí el sutil detalle de por que el juego, se ha convertido en un juego con información imperfecta"* por lo cuál es un buen agente para esa implementación.

Observemos el siguiente diagrama:

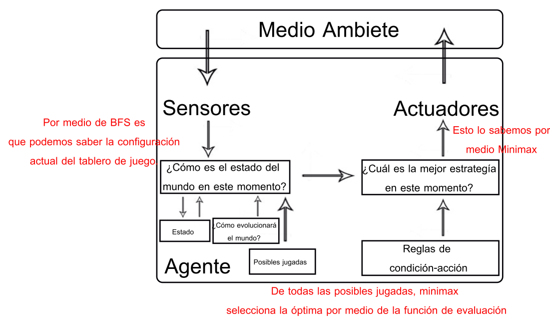

Figure 1: Diagrama en el cuál se describe al agente.

4 Risk acotado.

Tal y cómo se plantea en el juego original (***veáse [3]***) el objetivo del juego continua siendo la dominación total de un territorio dado, de tal forma que el juego queda concluido cuándo todos los territorios quedan bajo la dominación de un jugador.
En esta implementación acotaremos la cantidad de continentes, es decir, el desarrollo sera unicamente en un solo continente, también la cantidad de dados se ve acotada a unicamente dos dados y restringido a dos jugadores.

Sin embargo mantendremos las demás condiciones iniciales con respecto a las tropas y al equivalente de tropas en cada territorio, es decir:

* Cada unidad representa una *Armada*.

* Cada *Caballería* representa 5 unidades.

* Cada *Artillería* representa 10 unidades.

Teniendo ya esto definido, entonces, cada jugador tendrá un ejercito inicial de 3 tropas, por cada invasión se le asigna 3 tropas y por cada reforzamiento se le asigna 2 tropas.

5 Marco teórico

Para una posible implementación del juego de **RISK** hemos ideado una gráfica donde cada nodo contiene como información:

- Nombre e identificador del país que se representa.
- Nombre e identificador del jugador que tiene invadido a este país, en caso de no estar invadido es nulo.
- Número de tropas que tienen ocupado ese país, en caso de no estar invadido es cero.

Y cada arista representa la frontera de cada país.
Recomendamos utilizar la representación por lista de adyacencias, ya que la generación de movimientos y la función de evaluación están basados en el algoritmo *BFS*, que es una búsqueda por amplitud.
Buscamos por amplitud ya que nuestro objetivo es lograr obtener todos los paises en el menor de movimientos, para esto necesitamos que el agente obtenga los paises con mayor número de vecinos, es decir, los vértices com mayor grado, para así poder encapsular al enemigo.

6 Generación de movimientos

6.1 Invasiones

Por cada nodo existirá una nueva gráfica con el nuevo jugador y número de tropas asignado (3), lo cual significa que habrá tantas nuevas gráficas como número de vértices que tenga el tablero; para esto utilizaremos *BFS*.

Pseudocódigo 1 Definición de la función $invasiones$

Entrada: Gráfica G que representa el tablero actual,Jugador $actual$

Salida: Lista con todos los posibles invasiones

```
Queue q
List invasiones
G' = G
para todo v ∈ G hacer
   v.visitado = FALSE
termina para todo
v_1.visitado = FALSE{v_1 es el país con identificador 1}
si v_1.jugador = NULL entonces
   G'.v_1.jugador = actual
   G'.v_1.tropas = 3
   invasiones.add(G')
termina si
q.enqeue(v_1)
mientras q.isNotEmpty() hacer
   v = q.remove()
   para todo u ∈ Vecinos(v) hacer
      si u.visitado = FALSE entonces
         G' = G
         si v_1.jugador = NULL entonces
            G'.u.jugador = actual
            G'.u.tropas = 3
            invasiones.add(G')
         termina si
         u.visitado = TRUE
         q.enqeue(u)
      termina si
   termina para todo
termina mientras
devolver invasiones
```

6.2 Reforzamientos

Por cada nodo existirá una nueva gráfica con el nuevo número de tropas (2), lo cual significa que habrá tantas nuevas gráficas como paises pertenezcan a cada jugador; para esto utilizaremos *BFS*.

6.3 Ataques

Para cada nodo exisitirá nuevas gráficas, una donde el ataque fue exitoso y se le asginará una tropa donde provino el ataque, y otra donde el ataque no fue existoso y se elimina el número de tropas con respecto al número de tropas del atacado.

7 Función de evaluación.

La estrategía consiste en tomar cada país ocupado por **MAX** y ver a todos sus vecinos, se sumará el grado de cada vértice y se resta los que pertenecen al oponente (esto para la parte de invasión), acto seguido se realiza una comparacíon de que paises son del oponente y cuanto se diferencian las tropas, para as tomar su mejor desición de ataque o reforzamiento.

$$funcionEval(G) = \begin{cases} \infty \textit{ Max resulta ser ganador.} \\ -\infty \textit{ Min resulta ser ganador.} \\ \texttt{invadeReforzaAtaca(G)} \textit{ e.o.c.} \end{cases}$$

Donde:

`invadeReforzaAtaca` es un algoritmo que por medio de una modificaición a *BFS* cuenta las tropas y paises de cada jugador, suma los grados de cada vértice, suma los pertenecientes al jugador **MAX** y resta los del oponente **min**, también por cada país de **MAX** resta los vecinos que pertenezcan a **min**, por otra parte bajo esa misma idea compara el número de tropas.

8 Minimax

Es un algoritmo para *minimizar* la pérdida *máxima* esperada en juegos de adversarios con información perfecta.

Pseudocódigo 2 Definición de la función $reforzamientos$

Entrada: Gráfica G que representa el tablero actual
Salida: Lista con todos los posibles reforzamientos

$Queue\ q$
$List reforzamientos$
$G' = G$
para todo $v \in G$ **hacer**
  $v.visitado = FALSE$
termina para todo
$v_1.visitado = FALSE$ {v_1 es el país con identificador 1}
si $G'.v_1.jugador \neq NULL$ **entonces**
  $G'.v_1.tropas = v_1.tropas + 2$
termina si
$reforzamientos.add(G')$
$q.enqeue(v_1)$
mientras $q.isNotEmpty()$ **hacer**
  $v = q.remove()$
  para todo $u \in Vecinos(v)$ **hacer**
    si $u.visitado = FALSE$ **entonces**
      $G' = G$
      si $G'.u.jugador \neq NULL$ **entonces**
        $G'.u.tropas = u.tropas + 2$
        $reforzamientos.add(G')$
      termina si
      $u.visitado = TRUE$
      $q.enqeue(u)$
    termina si
  termina para todo
termina mientras
devolver $reforzamientos$

Pseudocódigo 3 Definición de la función $ataques$

Entrada: Gráfica G que representa el tablero actual, Jugador $actual$
Salida: Lista con todos los posibles ataques

$Queue\ q$
$List ataques$
para todo $v \in G$ **hacer**
  $v.visitado = FALSE$
termina para todo
$v_1.visitado = FALSE$ {v_1 es el país con identificador 1}
$q.enqeue(v_1)$
mientras $q.isNotEmpty()$ **hacer**
  $v = q.remove()$
  para todo $u \in Vecinos(v)$ **hacer**
    si $u.visitado = FALSE$ **entonces**
      $G' = G$
      si $G'.v.jugador = actual$ and $G'.u.jugador \neq NULL$ and $G'.u.jugador \neq actual$ **entonces**
        si $G'.v.tropas >= G'.u.tropas$ **entonces**
          $p = generaDados()$ {$generaDados()$ es una función que genera dos números aleatorios y realiza su resta, representa los dados}
          si $p > 0$ **entonces**
            $G'.u.jugador = actual$
            $G'.u.tropas = 1$
            $ataques.add(G')$
          si no
            $G'.v.tropas = G'.v.tropas - G'.u.tropas$
            $ataques.add(G')$
          termina si
        termina si
      termina si
      $u.visitado = TRUE$
      $q.enqeue(u)$
    termina si
  termina para todo
termina mientras
devolver $ataques$

Pseudocódigo 4 Definición de la función $invadeReforzaAtaca$

Entrada: La gráfica que representa el tablero

Salida: Es el entero descrito anteriormente

```
puntuacion = 0
Queue q
para todo v ∈ G hacer
    v.visitado = FALSE
termina para todo{v_1 es el país con identificador 1}
v_1.visitado = FALSE
q.enqeue(v_1)
mientras q.isNotEmpty() hacer
    v = q.remove()
    para todo u ∈ Vecinos(v) hacer
        si u.visitado = FALSE entonces
            si u.jugador = NULL and puntuacion < u.grado entonces
                puntuacion = puntuacion + u.grado
            termina si
            si v.Jugador = MAX and Jugador = MIN entonces
                puntuacion = puntuacion − 1
            termina si
            si u.jugador = MAX entonces
                puntuacion = puntuacion + u.daTropas() + 1
            termina si
            si u.jugador = MIN entonces
                puntuacion = puntuacion − u.daTropas() − 1
            termina si
            si v.Jugador = MAX and u.Jugador = MIN entonces
                puntuacion = puntuacion − 1
            termina si
            si v.tropas > u.tropas entonces
                puntuacion = puntuacion + 1
            termina si
            si v.tropas < u.tropas entonces
                puntuacion = puntuacion − 1
            termina si
            u.visitado = TRUE
            q.enqeue(u)
        termina si
    termina para todo
termina mientras
devolver puntuacion
```

Como se mencionó anteriormente dado que el juego de **Risk** es prácticamente intratable, se pierde esta propiedad de información perfecta, ya que el factor de ramificación es demasiado grande para poder ser implementada.

La idea teórica del algoritmo minimax es generar todo el árbol del juego, asignarles valor a cada nodo del árbol y hacer un recorrido *DFS* para obtener la mejor estrategía para **MAX**.

Dado que esto requiere una gran cantidad de espacio y tiempo, entonces la práctica usual es realizar el algoritmo *minimax* de forma recursiva, tal que, vaya simulando la creación de las ramas y después asginarles su valor y por último obtener la mejor estrategía.

9 Complejidad.

Notacion
ν denota a la cardinalidad del conjunto de vértices [**?**].
ϵ denota a la cardinalidad del conjunto de aristas [**?**].

Procederemos a hacer el análisis de complejidad para las funciones antes definidas, empecemos con la función **invaciones.**[1]
Invaciones:
En las primeras 3 lineas de código es fácil observar que la complejidad es constante, por lo que obtenemos un $O(1)$.
Continuando con el análisis observamos que asegurar el estado de *"no visitado"* de cada nodo es de orden $O(n)$ pues eso se consigue visitando cada nodo en la gráfica. Continuando así en el ciclo *while* y dado que éste tiene contenido un ciclo *for*, se observa que entonces la complejidad es del orden $O(\nu + \epsilon)$.

Con el algoritmo **Reforzamiento.**[2]
Reforzamiento:
El algoritmo de *reforzamiento* tenemos que la sección para visitar todos los nodos de la gráfica es de orden $O(n)$. Continuamos con la sección del ciclo *while* y dado que dentro del éste tiene un ciclo *for* vemos nuevamente que tenemos una complejidad de $O(\nu + \epsilon)$.

Con el algoritmo **Ataques**[3]
Al ser un algoritmo basado únicamente en *BFS*, de hecho, son sólo unas líneas modificadas de este, su complejidad corresponde al de dicho algoritmo, el método $generaDados$ es de orden constante ya que generamos dos números aleatorios y se restan, todo esto es constante, por lo que el metodo $ataques$ tendrá una complejidad de $O(\nu + \epsilon)$.

Para el algoritmo **minimax**[3]
Sabemos que la complejidad del algoritmo *minimax* es $O(b^n)$ donde b es el factor de ramificación y n la profundidad del árbol [5], en nuestro caso habrá tantas ramas como la gráfica tiene vértices y aristas debido a todos los posibles reforzamientos, invasiones y ataques, por lo que la complejidad de *minimax* en este caso es de $O((\nu + \epsilon)^n)$ en el peor de los casos.

Pseudocódigo 5 Definición de $minimax$

Entrada: Entero p que representa la profundiad del árbol, Jugador $actual$, gráfica G que representa el tablero actual

Salida: Gráfica que representa la mejor jugada

$List\ movimientos$

si $graficaLlena(G)$ **entonces**

$movimientos = concatena(ataques(G, actual), reforzamientos(G))$

si no

$movimientos = invasiones(G, actual)$

termina si

$mejorPuntuacion = 0$

si $actual = 1$ **entonces**

$mejorPuntuacion = -\infty$

si no

$mejorPuntuacion = \infty$

termina si

$puntuacionActual = 0$

$mejorMovimiento = null$

si $movimientos.isEmpty()\ or\ profundidad = 0$ **entonces**

$mejorPuntuacion = funcionEval(G)$

$mejor = G$

si no

para todo $movimiento \in movimientos$ **hacer**

si $actual = 1$ **entonces**

$puntuacionActual = funcionEval(minimax(p - 1, min, G, f))$

si $puntuacionActual > mejorPuntuacion$ **entonces**

$mejorPuntuacion = puntuacionActual$

$mejor = movimiento$

termina si

si no

$puntuacionActual = funcionEval(minimax(p - 1, max, G, f))$

si $puntuacionActual < mejorPuntuacion$ **entonces**

$mejorPuntuacion = puntuacionActual$

$mejor = movimiento$

termina si

termina si

termina para todo

termina si

devolver $mejor$

10 Conclusiones.

Se comenzó por una implementación en el lenguaje de programación *JAVA* pero al momento de querer realizar la lista de gráficas de todos los posibles movimientos, se volvió intratable en el sentido de que generar tales gráficas requiere de un gran espacio en memoria.
Acto seguido se optó por pasar al lenguaje de programación *Python* por su sencillez de la construcción de la gráfica que representa al tablero y sus funciones predefinidas sobre tal gráfica, pero se tuvo el mismo problema que *JAVA*.
Implementar el juego de risk acotado, es posible, sin embargo aún con estas restricciones, es muy ineficiente para la computadora, por la verificación de cada gráfica que representa a los movimientos.

References

[1] Información perfecta, la teora de juegos, microeconoma. http://centrodeartigo.com/articulos-utiles/article_113471.html. 17/Noviembre/2014.

[2] Juegos dinmicos. http://www.eco.uc3m.es/docencia/new_juegos/doc/2.2Dinamicosinfoimperfecta.pdf. 17/Noviembre/2014.

[3] Parker Brothers. Risk the world conquerior game. http://www.hasbro.com/common/instruct/risk.pdf. 27/Octubre/2014.

[4] Alexander Galetovic. Microeconomia ii. http://www.microeconomia.org/documentos_new/apun702.pdf. 17/Noviembre/2014.

[5] S.J. Russell and P. Norvig. *Inteligencia artificial: un enfoque moderno*. Colección de Inteligencia Artificial de Prentice Hall. Pearson Educación, 2004.

[6] Bruno López Takeyas. Algoritmo minimax. http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/InteligenciaArtificial/Apuntes/IA/Minimax.pdf. 17/Noviembre/2014.