

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 como 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 diseñadas 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 estrategia de **Minimax**, creada por el matemático *John von Neumann*. El cuál es un algoritmo para minimizar una estrategia 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 *jugador₁* gane, que haga lo propio el *jugador₂* o que el juego termine en empate.

En el capítulo 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:

4 Risk acotado.

Tal y cómo se plantea en el juego original (*véase [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 países en el menor de movimientos, para esto necesitamos que el agente obtenga los países con mayor número de vecinos, es decir, los vértices con 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*.

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 países pertenezcan a cada jugador; para esto utilizaremos *BFS*.

6.3 Ataques

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

Pseudocódigo 1 Definición de la función *invasiones*

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

Salida: Lista con todos los posibles invasiones

```
1: Queue  $q$ 
2: List invasiones
3:  $G' = G$ 
4: para todo  $v \in G$  hacer
5:    $v.visitado = FALSE$ 
6: termina para todo
7:  $v_1.visitado = FALSE$  { $v_1$  es el país con identificador 1}
8: si  $v_1.jugador = NULL$  entonces
9:    $G'.v_1.jugador = actual$ 
10:   $G'.v_1.tropas = 3$ 
11:  invasiones.add( $G'$ )
12: termina si
13:  $q.enqueue(v_1)$ 
14: mientras  $q.isNotEmpty()$  hacer
15:    $v = q.remove()$ 
16:   para todo  $u \in Vecinos(v)$  hacer
17:     si  $u.visitado = FALSE$  entonces
18:        $G' = G$ 
19:       si  $v_1.jugador = NULL$  entonces
20:          $G'.u.jugador = actual$ 
21:          $G'.u.tropas = 3$ 
22:         invasiones.add( $G'$ )
23:       termina si
24:        $u.visitado = TRUE$ 
25:        $q.enqueue(u)$ 
26:     termina si
27:   termina para todo
28: termina mientras
29: devolver invasiones
```

Pseudocódigo 2 Definición de la función *reforzamientos*

Entrada: Gráfica G que representa el tablero actual

Salida: Lista con todos los posibles reforzamientos

```
1: Queue  $q$ 
2: List  $reforzamientos$ 
3:  $G' = G$ 
4: para todo  $v \in G$  hacer
5:    $v.visitado = FALSE$ 
6: termina para todo
7:  $v_1.visitado = FALSE$  { $v_1$  es el país con identificador 1}
8: si  $G'.v_1.jugador \neq NULL$  entonces
9:    $G'.v_1.tropas = v_1.tropas + 2$ 
10: termina si
11:  $reforzamientos.add(G')$ 
12:  $q.enqueue(v_1)$ 
13: mientras  $q.isNotEmpty()$  hacer
14:    $v = q.remove()$ 
15:   para todo  $u \in Vecinos(v)$  hacer
16:     si  $u.visitado = FALSE$  entonces
17:        $G' = G$ 
18:       si  $G'.u.jugador \neq NULL$  entonces
19:          $G'.u.tropas = u.tropas + 2$ 
20:          $reforzamientos.add(G')$ 
21:       termina si
22:        $u.visitado = TRUE$ 
23:        $q.enqueue(u)$ 
24:     termina si
25:   termina para todo
26: termina mientras
27: 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

```
1: Queue  $q$ 
2: Listataques
3: para todo  $v \in G$  hacer
4:    $v.visitado = FALSE$ 
5: termina para todo
6:  $v_1.visitado = FALSE$  { $v_1$  es el país con identificador 1}
7:  $q.enqueue(v_1)$ 
8: mientras  $q.isNotEmpty()$  hacer
9:    $v = q.remove()$ 
10:  para todo  $u \in Vecinos(v)$  hacer
11:    si  $u.visitado = FALSE$  entonces
12:       $G' = G$ 
13:      si  $G'.v.jugador = actual$  and  $G'.u.jugador \neq$   

        $NULL$  and  $G'.u.jugador \neq actual$  entonces
14:        si  $G'.v.tropas \geq G'.u.tropas$  entonces
15:           $p = generaDatos()$  {generaDatos() es una función que genera  

           dos números aleatorios y realiza su resta, representa los dados}
16:          si  $p > 0$  entonces
17:             $G'.u.jugador = actual$ 
18:             $G'.u.tropas = 1$ 
19:             $ataques.add(G')$ 
20:          si no
21:             $G'.v.tropas = G'.v.tropas - G'.u.tropas$ 
22:             $ataques.add(G')$ 
23:          termina si
24:        termina si
25:      termina si
26:       $u.visitado = TRUE$ 
27:       $q.enqueue(u)$ 
28:    termina si
29:  termina para todo
30: termina mientras
31: devolver ataques
```

7 Función de evaluación.

La estrategia 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 comparación de que países son del oponente y cuanto se diferencian las tropas, para as tomar su mejor desición de ataque o reforzamiento.

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

Donde:

`invadeReforzaAtaca` es un algoritmo que por medio de una modifcaición a *BFS* cuenta las tropas y países 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.

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 estrategia 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 asignarles su valor y por último

Pseudocódigo 4 Definición de la función *invadeReforzaAtaca*

Entrada: La gráfica que representa el tablero

Salida: Es el entero descrito anteriormente

```
1: puntuacion = 0
2: Queue q
3: para todo v ∈ G hacer
4:   v.visitado = FALSE
5: termina para todo{v1 es el país con identificador 1}
6: v1.visitado = FALSE
7: q.enqueue(v1)
8: mientras q.isNotEmpty() hacer
9:   v = q.remove()
10:  para todo u ∈ Vecinos(v) hacer
11:    si u.visitado = FALSE entonces
12:      si u.jugador = NULL and puntuacion < u.grado entonces
13:        puntuacion = puntuacion + u.grado
14:      termina si
15:      si v.Jugador = MAX and Jugador = MIN entonces
16:        puntuacion = puntuacion − 1
17:      termina si
18:      si u.jugador = MAX entonces
19:        puntuacion = puntuacion + u.daTropas() + 1
20:      termina si
21:      si u.jugador = MIN entonces
22:        puntuacion = puntuacion − u.daTropas() − 1
23:      termina si
24:      si v.Jugador = MAX and u.Jugador = MIN entonces
25:        puntuacion = puntuacion − 1
26:      termina si
27:      si v.tropas > u.tropas entonces
28:        puntuacion = puntuacion + 1
29:      termina si
30:      si v.tropas < u.tropas entonces
31:        puntuacion = puntuacion − 1
32:      termina si
33:      u.visitado = TRUE
34:      q.enqueue(u)
35:    termina si
36:  termina para todo
37: termina mientras
38: devolver puntuacion
```

obtener la mejor estrategia.

9 Complejidad.

Procederemos a hacer el análisis de complejidad para las funciones antes definidas, empecemos con la función **invaciones**.¹

Invaciones:

En las primeras 3 líneas 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 $O(\nu + \epsilon)$.

Continuemos 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 $O(\nu + \epsilon)$.

10 Conclusiones.

Informar las conclusiones que hemos encontrado en nuestra implementación.

Pseudocódigo 5 Definición de *minimax*

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

Salida: Gráfica que representa la mejor jugada

```
1: List movimientos
2: si graficaLlena(G) entonces
3:   movimientos = concatena(ataques(G, actual), reforzamientos(G))
4: si no
5:   movimientos = invasiones(G, actual)
6: termina si
7: mejorPuntuacion = 0
8: si actual = 1 entonces
9:   mejorPuntuacion =  $-\infty$ 
10: si no
11:   mejorPuntuacion =  $\infty$ 
12: termina si
13: puntuacionActual = 0
14: mejorMovimiento = null
15: si movimientos.isEmpty() or profundidad = 0 entonces
16:   mejorPuntuacion = funcionEval(G)
17:   mejor = G
18: si no
19:   para todo movimiento  $\in$  movimientos hacer
20:     si actual = 1 entonces
21:       puntuacionActual = funcionEval(minimax( $p - 1$ , min, G, f))
22:       si puntuacionActual > mejorPuntuacion entonces
23:         mejorPuntuacion = puntuacionActual
24:         mejor = movimiento
25:       termina si
26:     si no
27:       puntuacionActual = funcionEval(minimax( $p - 1$ , max, G, f))
28:       si puntuacionActual < mejorPuntuacion entonces
29:         mejorPuntuacion = puntuacionActual
30:         mejor = movimiento
31:       termina si
32:     termina si
33:   termina para todo
34: termina si
35: devolver mejor
```

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.