Implementación de Risk con algoritmo de Minimax.

Prieto, Estefanía[1]
Galicia, Fernando[1]
Galván, Antonio[1]
[1]Facultad de Ciencias, U.N.A.M.

estefaniaprieto@ciencias.unam.mx
fernandogamen@ciencias.unam.mx
g.antonio@ciencias.unam.mx

21-Noviembre-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.

Hacer acá la introducción de minimax.

2 Juegos con información perfecta. [MODIFICAR A INFORMARCIÓN IMPERFECTA Y ARGUMENTAR EL POR QUE PODEMOS ADAPTARLO ASÍ.]

Explicar por que catalogamos al **RISK** cómo un juego de información perfecta y por que hemos elegido esta implementación.

3 Risk acotado.

Tal y cómo se plantea en el juego original (***veáse [2]***) 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 35 tropas.

4 Descripción del agente.

Aquí es donde describimos el comportamiento del agente.

5 Función de evaluación.

Dado que el juego de risk consiste en dos partes, entonces se necesitan dos series de estrategias: la primera para la invasión de territorio y la segunda parte consiste en reforzamiento y ataque; por lo que se necesitan dos funciones de evaluación.

5.1 Invasión

La estrategía consiste en invadir los paises con mayor número de vecinos y poder encapsular al enemigo.

$$F(n) = \begin{cases} \infty \textit{ Max obtiene los paises con mayor grado.} \\ -\infty \textit{ Min obtiene los paises con mayor grado.} \\ \texttt{invasion}(\texttt{n}) \textit{ e.o.c.} \end{cases}$$

Donde:

`invasion` por medio de **BFS** suma al valor de la función de evaluación: el grado de cada vértice, el total de vecinos no ocupados por ninguno de los jugadores; y por último resta los vecinos ocupados por **min** por cada país de **MAX**.

5.2 Reforzamiento y ataque

La estrategía consiste en tomar cada país ocupado por **MAX** y ver a todos sus vecinos, 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.

Pseudocódigo 1 Definición de la función $inavsion$

Entrada: La gráfica que representa el tablero
Salida: Es el entero descrito anteriormente

$puntuacion = 0$
$Queue\ q$
para todo $v \in G$ **hacer**
    $v.visitado = FALSE$
termina para todo
$v_1.visitado = FALSE$ {v_1 es el país con identificador 1}
$puntuacion = v_1.grado$
$q.enqeue(v_1)$
mientras $q.isNotEmpty()$ **hacer**
    $v = q.remove()$
    para todo $u \in Vecinos(v)$ **hacer**
        si $u.visitado = FALSE$ **entonces**
            si $puntuacion < u.grado$ **entonces**
                $puntuacion = u.grado$
            termina si
            si $v.Jugador = MAX\ and\ Jugador = MIN$ **entonces**
                $puntuacion = puntuacion - 1$
            termina si
            $u.visitado = TRUE$
            $q.enqeue(u)$
        termina si
    termina para todo
termina mientras
devolver $puntuacion$

$$F(n) = \begin{cases} \infty \text{ \textit{Max resulta ser ganador.}} \\ -\infty \text{ \textit{Min resulta ser ganador.}} \\ \texttt{reforzaAtaca(n)} \text{ \textit{e.o.c.}} \end{cases}$$

Donde:

`reforzaAtaca` es un algoritmo que por medio de una modificaición a *BFS* cuenta las tropas y paises de cada jugador, 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.

6 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 estrategía para **MAX**.

Dado que esto requiere una gran cantidad de espacio y tiempo, entonces la práctica usual es realizar el algoritmo *minimax* de

Pseudocódigo 2 Definición de la función $reforzaAtaca$

Entrada: La gráfica que representa el tablero
Salida: Es el entero descrito anteriormente

$puntuacion = 0$
$Queue\ q$
para todo $v \in G$ **hacer**
  $v.visitado = FALSE$
termina para todo{v_1 es el país con identificador 1}
$v_1.visitado = FALSE$
si $v_1.jugador = MAX$ **entonces**
  $puntuacion = puntuacion + u.daTropas() + 1$
termina si
si $v_1.jugador = MIN$ **entonces**
  $puntuacion = puntuacion - u.daTropas() - 1$
termina si
$q.enqeue(v_1)$
mientras $q.isNotEmpty()$ **hacer**
  $v = q.remove()$
  para todo $u \in Vecinos(v)$ **hacer**
    si $u.visitado = FALSE$ **entonces**
      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

forma recursiva, tal que, vaya simulando la creación de las ramas y después asginarles su valor y por último obtener la mejor estrategía.

7 Especificaciones del programa.

La implementación concreta del proyecto se ha realizado en el lenguaje de programación ***Java*** [4] que se ha optado por que *Escribir ventajas de java y por que hemos optado por él*

Una vez aclarado esto, introduciremos la representación del territorio por medio de un archivo llamado *"Territorio.xml"* en el cual obtenemos las ventajas de que éste nos brinda una estructura la cual nos permite adaptar información de manera independiente al manejo de ésta [3]. Así el territorio del juego en el que cada país tendrá una etiqueta que lo represente y dentro de ésta estarán especificados los atributos de cada país.

De esta, hemos seccionado a los países en una pequeña base de datos. Así que usaremos la interfaz *JAXP* [1]

8 Conclusiones.

Informar las conclusiones que hemos encontrado en nuestra implementación.

Pseudocódigo 3 Definición de $minimax$

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

Salida: Gráfica que representa la mejor jugada

$List\ movimientos$
$mejorPuntuacion = 0$
si $actual = 1$ **entonces**
  $mejorPuntuacion = -\infty$
si no
  $mejorPuntuacion = \infty$
termina si
$puntuacionActual = 0$
$mejorMovimiento = null$
si $movimientos.isEmpty()\ or\ profundidad = 0$ **entonces**
  $mejorPuntuacion = f(G)$
  $mejor = G$
si no
  para todo $movimiento \in movimientos$ **hacer**
    si $actual = 1$ **entonces**
      $puntuacionActual = f(minimax(p - 1, min, G, f))$
      si $puntuacionActual > mejorPuntuacion$ **entonces**
        $mejorPuntuacion = puntuacionActual$
        $mejor = movimiento$
      termina si
    si no
      $puntuacionActual = f(minimax(p - 1, max, G, f))$
      si $puntuacionActual < mejorPuntuacion$ **entonces**
        $mejorPuntuacion = puntuacionActual$
        $mejor = movimiento$
      termina si
    termina si
  termina para todo
termina si
devolver $mejor$

References

[1] Jaxp reference implementation. `https://jaxp.java.net/`. 07/Noviembre/2014.

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

[3] Borland Software Corporation. *XML Developer's Guide*. 1997.

[4] ORACLE. Descarga gratuita de java. `https://java.com/es/download/index.jsp`. 06/Noviembre/2014.