SEGUNDA TAREA PROGRAMADA, INTELIGENCIA ARTIFICIAL

JOSÉ CASTRO

Contents

1. Introducción

La programación de juegos es una de las áreas en las cuales la IA ha tenido mas éxito. En especial el algoritmo Minimax y sus variantes se han utilizado para la programación de agentes autonomos para ajedrez, damas, y un sin fin de juegos de dos contrincantes. En esta tarea se pretende programar el algoritmo minimax en el contexto de jugar una partida de Othello. El objetivo de este ejercicio es entender bien el algoritmo minimax y las consideraciones prácticas que se deben tener cuando se efectua su implementación.

2. Especificación de la tarea a programar

En esta tarea usted debe programar un agente de Othello. Othello es un juego de dos jugadores en los cuales los contrincantes se turnan para poner fichas en un tablero 8x8. El juego siempre empieza con las blancas y cada jugador solo puede poner fichas al lado de una ficha del contrincante, de tal forma que se encuentren rodeadas por lo menos una fila de fichas del contrincante por un par de fichas del jugador.

En vez de explicar más a fondo como funciona el juego de Othello, se provee el códico en Erlang de un servidor de Othello el cual también sirve para jugar interactivamente. El programa se encuentra en la página del tec-digital del curso y se llama `server.erl`. Para correr el programa se ejecuta dentro del ambiente de erlang el comando `server:start().`

El programa `server.erl` comparte la siguiente estructura de datos con sus clientes (jugadores) que se encuentra en el archivo `othello.hrl` (*erland header file.*)

```
-record(game,
    {id      = othello, % constante
     black   = none,    % proceso que juega con negras (ie: <0.33.0>)
     white   = none,    % proceso que juega con blancas
     current = white,   % identificador de turno (white|black)
     seconds = 30,      % segundos que dura el turno
     board   = none,    % tupla con el tablero actual
     border  = none     % lista con indices de posicion del borde
    }).
```

El programa `server.erl` incorpora las siguientes funciones:

- `server:start()`: inicializa y ejecuta el server con su interfaz gráfico.
- `server:connect(Color)`: indica al servidor cual es el proceso que va a estar jugando con el color `Color`, lo que provoca que el campo `Color`, el cual puede ser `white` o `black` del registro `game` sea actualizado con la identidad del proceso. Dicho proceso es notificado por el servidor con un mensaje que contiene el record `game` indicado anteriormente cada vez que el estado del juego cambia. La estructura del mensaje que envia el servidor se expone más adelante.
- `server:disconnect()`: desconexión del servidor, limpia el campo de `Color` del registro `game` que fue actualizado con el comando `server:connect()`.
- `server:get_status()`: retorna el estado del servidor, el cual es una instancia del record `game`.
- `server:make_move(Color, Pos)`: indica al servidor que hay que poner una ficha de color `Color`, la cual puede ser `white` o `black`, en la posición `Pos` del tablero.

Usted debe hacer un modulo llamado `tcarnet.erl` donde `carnet` lo debe sustituir por su carnet, por ejemplo si su carnet es 20113124 entonces el módulo debe llamarse `t20113124.erl`. El módulo debe proveer la siguiente interface.

- `t20113124:start()`: inicia su tarea como un proceso independiente con interfaz gráfico (si desea). Se sugiere que el su tarea registre un proceso con el nombre del m'odulo que implementa su tarea:
  `register(t20113124, spawn(fun() -> agent() end)).`
- `t20113124:stop()`: termina el proceso.
- `t20113124:connect(Color)`: se conecta con el servidor en el color indicado.

Además su tarea debe responder a los siguientes mensajes que envia el servidor:

- {`ok, State`}: indica al proceso cual es el estado actual del juego en el servidor, no requiere de ninguna acción por parte del proceso.
- {`your_turn, State`}: indica al proceso que es su turno de jugar, en cuyo caso el proceso debe responder al servidor con `server:make_move(Color, Pos)` en tiempo menor o igual a los segundos indicados en el registro `game` que contiene la variable `State#game.seconds`.

3. Archivos y Ejemplos

En la página web del curso se encuentran los siguientes archivos:

- `server.erl`: código fuente del servidor de othello.
- `display.erl`: código fuente de funciones utilitarias de desplieque del tablero de othello.
- `othello.erl`: código fuente de funciones utilitarias del juego othello.
- `othello.hrl`: archivo *header* que contiene registro de estado del servidor de othello.
- `agent.beam`: archivo binario de código de agente de erlang que interactúa con el servidor.
- `minimax.beam`: archivo binario de código de funciones de minimax.

A continuaciíon, una posible interacción con erlang una vez cargados los archivos anteriores:

```
erl> c(server).
{ok,server}
erl> server:start().
true
erl> agent:start(blancas).
true
erl> agent:connect(blancas, white).
erl> agent:connect(negras, black).
```

Una vez hecho esto se oprime el botón de *New Game* del servido de Othello.

4. Evaluación

- 15% Documentación
- 65% Funcionalidad
- 10% Correctitud del programa
- 10% Análisis de resultados

5. Fecha de Entrega: 11 de Setiembre 2013