KR-IST - Lecture 5b
Game play in Java

Chris Thornton

October 30, 2014

Introduction

This lecture will look at a program for playing noughts and crosses (tic-tac-toe).

The implementation makes use of ‘negmax’ evaluation but does not use alpha-beta pruning.

Step-by-step guide for writing a search program

Step-by-step guide for writing a search program

(1) Informally characterise states, goals and transitions.

Step-by-step guide for writing a search program

(1) Informally characterise states, goals and transitions.

(2) Choose a representation for states.

Step-by-step guide for writing a search program

(1) Informally characterise states, goals and transitions.

(2) Choose a representation for states.

(3) Sketch out a method for generating successors. (Go back to step 2 if this is difficult.)

Step-by-step guide for writing a search program

(1) Informally characterise states, goals and transitions.

(2) Choose a representation for states.

(3) Sketch out a method for generating successors. (Go back to step 2 if this is difficult.)

(4) Write Java code to implement the successor method. (Go back to step 2 if this is difficult.)

Step-by-step guide for writing a search program

(1) Informally characterise states, goals and transitions.

(2) Choose a representation for states.

(3) Sketch out a method for generating successors. (Go back to step 2 if this is difficult.)

(4) Write Java code to implement the successor method. (Go back to step 2 if this is difficult.)

(5) Write code for the main search loop (i.e., procesing of nodes on OPEN).

Step-by-step guide for writing a search program

(1) Informally characterise states, goals and transitions.

(2) Choose a representation for states.

(3) Sketch out a method for generating successors. (Go back to step 2 if this is difficult.)

(4) Write Java code to implement the successor method. (Go back to step 2 if this is difficult.)

(5) Write code for the main search loop (i.e., procesing of nodes on OPEN).

(6) Add everything else.

Step-by-step guide for writing a search program

(1) Informally characterise states, goals and transitions.

(2) Choose a representation for states.

(3) Sketch out a method for generating successors. (Go back to step 2 if this is difficult.)

(4) Write Java code to implement the successor method. (Go back to step 2 if this is difficult.)

(5) Write code for the main search loop (i.e., procesing of nodes on OPEN).

(6) Add everything else.

Node class

```
import java.util.*;

class Node {
   int state[] = new int[9];
   int evaluation = -1;
   Node parent = null;

   Node(int s[], Node parent) {
      for (int i = 0; i < 9; i++) state[i] = s[i];
      this.parent = parent;
   }

   public String toString() {
      String s = "";
      for (int i = 0; i < 9; i++) s = s + state[i] + " ";
      return s;
   }
```

Node class cont.

```
    int[] getStateCopy() {
        int[] s = state, copy = {s[0], s[1], s[2], s[3],
                  s[4], s[5], s[6], s[7], s[8]};
        return copy;
    }

    Vector<Node> getPath(Vector<Node> v) {
        v.insertElementAt(this, 0);
        if (parent != null) v = parent.getPath(v);
        return v;
    }

    Vector<Node> getPath() { return(getPath(new Vector<Node>())); }
}
```

NoughtsAndCrossesSpace class

```
class NoughtsAndCrossesSpace {

   Node getRoot() {
      int state[] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
      return new Node(state, null);
   }

   Vector<Node> getSuccessors(Node parent, int player) {
      Vector<Node> successors = new Vector<Node>();
      for (int r = 0; r < 3; r++) {
         for (int c = 0; c < 3; c++) {
            if (parent.state[(r * 3) + c] == 0) { /* empty cell here
               int state[] = parent.getStateCopy();
               state[(r * 3) + c] = player;
               successors.add(new Node(state, parent)); }
         }
      }
      return successors;
   }
}
```

Search space

```
public class NoughtsAndCrossesSearch {
   NoughtsAndCrossesSpace space = new NoughtsAndCrossesSpace();

   String name(int player) {
      String s = "_";
      if (player == 1) {
         s = "X"; }
      else if (player == -1) {
         s = "O"; }
      return s;
   }

   void pr(String s) {
      System.out.println(s);
   }

   void printState(Node node) { /* print board state */
      System.out.println("\n");
      for (int r = 0; r < 3; r++) {
         for (int c = 0; c < 3; c++) {
            int player = node.state[(r * 3) + c];
```

Static evaluation

```
/** decide whether board state s is won for player p */
boolean wonFor(int s[], int p) {
   boolean b = (s[0] == p && s[1] == p && s[2] == p)
   || (s[3] == p && s[4] == p && s[5] == p)
   || (s[6] == p && s[7] == p && s[8] == p)
   || (s[0] == p && s[3] == p && s[6] == p)
   || (s[1] == p && s[4] == p && s[7] == p)
   || (s[2] == p && s[5] == p && s[8] == p)
   || (s[0] == p && s[4] == p && s[8] == p)
   || (s[2] == p && s[4] == p && s[6] == p);
   return b;
}

int winnerOf(int state[]) {
   int player = 0;
   if (wonFor(state, 1)) {
      player = 1; }
   else if (wonFor(state, -1)) {
      player = -1; }
   return player;
}
```

Dynamic evaluation

```
int evaluate(Node node, int player) { /* using NEGMAX */
   int value = 0;
   if (wonFor(node.state, player)) {
      value = 1; }
   else if (wonFor(node.state, -player)) {
      value = -1; }
   else {
      Vector<Node> successors = space.getSuccessors(node, -player);
      if (successors.size() == 0) { /* draw */
         value = 0; }
      else {
         for (int i = 0; i < successors.size(); i++) {
            Node successor = successors.get(i);
            successor.evaluation = evaluate(successor, -player);
            if (successor.evaluation > value) {
               value = successor.evaluation; } }
         value = -value; } }
   return(value);
}
```

Main loop

```
void run() {
   int p, player = 1;
   Node node = space.getRoot();
   Vector<Node> bestNodes = new Vector<Node>();
   printState(node);
   while ((p = winnerOf(node.state)) == 0) { /* while no winner */
      Vector<Node> successors = space.getSuccessors(node, player);
      int maxValue = -Integer.MAX_VALUE;
      bestNodes.clear();
      for (int i = 0; i < successors.size(); i++) {
         Node newNode = successors.get(i);
         int value = evaluate(newNode, player);
         if (value == maxValue || player == -1) {
            bestNodes.add(newNode); } /* ensure random opponent */
         else if (value > maxValue) {
            bestNodes.clear();
            bestNodes.add(newNode);
            maxValue = value; } }
```

Main loop cont.

```
            if (successors.size() == 0) { /* game drawn */
               break; }
            else {
               pr("State after new " + name(player) + " (" + player + ")
               int randomIndex = (int)(Math.random() * bestNodes.size())
               node = bestNodes.get(randomIndex);
               printState(node);
               player = -player; }
         }
         pr(p == 0 ? "DRAW" : "GAME WON FOR " + name(p) + "\n\n");
      }

      public static void main(String args[]) { // do the search
         new NoughtsAndCrossesSearch().run();
      }
   }
```

Simulated game

```
Initial node: (X never errs, O plays randomly)
 - - -
 - - -
 - - -

State after new X (1)
 - - -
 X - -
 - - -

State after new O (-1)
 - - -
 X - -
 O - -
```

Cont.

```
State after new X (1)
 _ X _
 X _ _
 O _ _

State after new O (-1)
 _ X _
 X _ _
 O O _

State after new X (1)
 _ X _
 X _ _
 O O X
```

Underlying evaluations for prev move

```
Evaluations
|-- 0 for O -1 1 0 1 0 0 -1 -1 1
|    |-- 0 for X -1 1 1 1 0 0 -1 -1 1
|    |    |-- -1 for O -1 1 1 1 -1 0 -1 -1 1
|    |    |    |-- 1 for X -1 1 1 1 -1 1 -1 -1 1
|    |    |-- 0 for O -1 1 1 1 0 -1 -1 -1 1
|    |         |-- 0 for X -1 1 1 1 1 -1 -1 -1 1
|    |-- 0 for X -1 1 0 1 1 0 -1 -1 1
|    |    |-- -1 for O -1 1 -1 1 1 0 -1 -1 1
|    |    |    |-- 1 for X -1 1 -1 1 1 1 -1 -1 1
|    |    |-- 0 for O -1 1 0 1 1 -1 -1 -1 1
|    |         |-- 0 for X -1 1 1 1 1 -1 -1 -1 1
|    |-- 0 for X -1 1 0 1 0 1 -1 -1 1
|         |-- -1 for O -1 1 -1 1 0 1 -1 -1 1
|         |    |-- 1 for X -1 1 -1 1 1 1 -1 -1 1
|         |-- -1 for O -1 1 0 1 -1 1 -1 -1 1
|              |-- 1 for X -1 1 1 1 -1 1 -1 -1 1
```

Evaluations cont.

```
|-- 0 for O 0 1 -1 1 0 0 -1 -1 1
|    |-- -1 for X 1 1 -1 1 0 0 -1 -1 1
|    |    |-- 1 for O 1 1 -1 1 -1 0 -1 -1 1
|    |    |-- -1 for O 1 1 -1 1 0 -1 -1 -1 1
|    |         |-- 1 for X 1 1 -1 1 1 -1 -1 -1 1
|    |-- 0 for X 0 1 -1 1 1 0 -1 -1 1
|    |    |-- -1 for O -1 1 -1 1 1 0 -1 -1 1
|    |    |    |-- 1 for X -1 1 -1 1 1 1 -1 -1 1
|    |    |-- -1 for O 0 1 -1 1 1 -1 -1 -1 1
|    |         |-- 1 for X 1 1 -1 1 1 -1 -1 -1 1
|    |-- -1 for X 0 1 -1 1 0 1 -1 -1 1
|         |-- -1 for O -1 1 -1 1 0 1 -1 -1 1
|         |    |-- 1 for X -1 1 -1 1 1 1 -1 -1 1
|         |-- 1 for O 0 1 -1 1 -1 1 -1 -1 1
```

Evaluations cont.

```
|-- 0 for O 0 1 0 1 -1 0 -1 -1 1
|    |-- -1 for X 1 1 0 1 -1 0 -1 -1 1
|    |    |-- 1 for O 1 1 -1 1 -1 0 -1 -1 1
|    |    |-- -1 for O 1 1 0 1 -1 -1 -1 -1 1
|    |         |-- 1 for X 1 1 1 1 -1 -1 -1 -1 1
|    |-- 0 for X 0 1 1 1 -1 0 -1 -1 1
|    |    |-- -1 for O -1 1 1 1 -1 0 -1 -1 1
|    |    |    |-- 1 for X -1 1 1 1 -1 1 -1 -1 1
|    |    |-- -1 for O 0 1 1 1 -1 -1 -1 -1 1
|    |         |-- 1 for X 1 1 1 1 -1 -1 -1 -1 1
|    |-- -1 for X 0 1 0 1 -1 1 -1 -1 1
|         |-- -1 for O -1 1 0 1 -1 1 -1 -1 1
|         |    |-- 1 for X -1 1 1 1 -1 1 -1 -1 1
|         |-- 1 for O 0 1 -1 1 -1 1 -1 -1 1
```

Evaluations cont.

```
|-- 0 for O 0 1 0 1 0 -1 -1 -1 1
    |-- 0 for X 1 1 0 1 0 -1 -1 -1 1
    |   |-- -1 for O 1 1 -1 1 0 -1 -1 -1 1
    |   |   |-- 1 for X 1 1 -1 1 1 -1 -1 -1 1
    |   |-- -1 for O 1 1 0 1 -1 -1 -1 -1 1
    |       |-- 1 for X 1 1 1 1 -1 -1 -1 -1 1
    |-- 0 for X 0 1 1 1 0 -1 -1 -1 1
    |   |-- 0 for O -1 1 1 1 0 -1 -1 -1 1
    |   |   |-- 0 for X -1 1 1 1 1 -1 -1 -1 1
    |   |-- -1 for O 0 1 1 1 -1 -1 -1 -1 1
    |       |-- 1 for X 1 1 1 1 -1 -1 -1 -1 1
    |-- 0 for X 0 1 0 1 1 -1 -1 -1 1
        |-- 0 for O -1 1 0 1 1 -1 -1 -1 1
        |   |-- 0 for X -1 1 1 1 1 -1 -1 -1 1
        |-- -1 for O 0 1 -1 1 1 -1 -1 -1 1
            |-- 1 for X 1 1 -1 1 1 -1 -1 -1 1
```

Game continues

```
State after new O (-1)
 _ X _
 X _ O
 O O X

State after new X (1)
 _ X X
 X _ O
 O O X

 Evaluations
 |-- 0 for O -1 1 1 1 0 -1 -1 -1 1
 |    |-- 0 for X -1 1 1 1 1 -1 -1 -1 1
 |-- -1 for O 0 1 1 1 -1 -1 -1 -1 1
      |-- 1 for X 1 1 1 1 -1 -1 -1 -1 1
```

Game concludes

```
State after new O (-1)
 _ X X
 X O O
 O O X

 Evaluations
 |-- 1 for X 1 1 1 1 -1 -1 -1 -1 1



State after new X (1)
 X X X
 X O O
 O O X

GAME WON FOR X
```

Summary

Summary

- Node class

Summary

- Node class
- NoughtsAndCrossesSpace class

Summary

- Node class
- NoughtsAndCrossesSpace class
- Static evaluations

Summary

- Node class
- NoughtsAndCrossesSpace class
- Static evaluations
- Dynamic evaluation

Summary

- Node class
- NoughtsAndCrossesSpace class
- Static evaluations
- Dynamic evaluation
- Tree generation

Summary

- Node class
- NoughtsAndCrossesSpace class
- Static evaluations
- Dynamic evaluation
- Tree generation
- Main loop

Summary

- Node class
- NoughtsAndCrossesSpace class
- Static evaluations
- Dynamic evaluation
- Tree generation
- Main loop
- main method

Summary

- Node class
- NoughtsAndCrossesSpace class
- Static evaluations
- Dynamic evaluation
- Tree generation
- Main loop
- main method

Questions

Questions

- Why does this program use an evaluation parameter for nodes rather than a cost parameter?

Questions

- Why does this program use an evaluation parameter for nodes rather than a cost parameter?
- How easy would it be to rewrite the wonFor method so as to make use of iterative constructs.

Questions

- Why does this program use an evaluation parameter for nodes rather than a cost parameter?
- How easy would it be to rewrite the wonFor method so as to make use of iterative constructs.

Exercises

Exercises

- Modify the NoughtsAndCrossesSearch program so that it prefers wins which involve fewer moves.

Exercises

- Modify the NoughtsAndCrossesSearch program so that it prefers wins which involve fewer moves.
- Modify the NoughtsAndCrossesSearch program so that it uses ordinary minimax evaluation.

Exercises

- Modify the NoughtsAndCrossesSearch program so that it prefers wins which involve fewer moves.
- Modify the NoughtsAndCrossesSearch program so that it uses ordinary minimax evaluation.
- Modify the NoughtsAndCrossesSearch program so as to implement alpha-beta pruning.

Exercises

- Modify the NoughtsAndCrossesSearch program so that it prefers wins which involve fewer moves.
- Modify the NoughtsAndCrossesSearch program so that it uses ordinary minimax evaluation.
- Modify the NoughtsAndCrossesSearch program so as to implement alpha-beta pruning.
- Modify the NoughtsAndCrossesSearch program so that it is interactive, i.e., moves by the second player are chosen by the user.

Exercises

- Modify the NoughtsAndCrossesSearch program so that it prefers wins which involve fewer moves.
- Modify the NoughtsAndCrossesSearch program so that it uses ordinary minimax evaluation.
- Modify the NoughtsAndCrossesSearch program so as to implement alpha-beta pruning.
- Modify the NoughtsAndCrossesSearch program so that it is interactive, i.e., moves by the second player are chosen by the user.