CS 401 Artificial Intelligence

Lecture 6
Game Playing

Department of Computer Science
National University of Computers & Emerging Sciences
Lahore.

Game Playing

- Playing games have always been an attraction of computers, and will probably always be so.
- Action games where you charge around blowing everything up are one type of game, but we are more interested in this course in examining the more thoughtful games, such as chess, noughts and crosses and the game of Nim.
- How do you get a computer to make sensible moves when for Nim there are 48 possible positions, for noughts and crosses around 5400 and for chess, well for chess there's a horribly big number of board positions possible.

Surely we can use the search techniques that we have just covered?

Game Playing

- The problems we have thus far considered have been of the type where the problem is well defined at the beginning and nothing changes during the search.
- Except for one player games that have no chance involved, games involve either chance, or another player's moves or both.
- In this case we cannot simply search for a solution as the other player will try to block you and games of chance simply will not cooperate!
- So, do we throw away all that we have just looked at in Search?
- No, but we do need to modify the techniques to allow for opponents.

Generate and Test

- One approach to take is to look at the problem as one of generating then testing possible sets of moves.
- One extreme would be to generate a number of complete sets of moves and then test each of them, taking the best one.
- The alternate extreme is to generate one move at a time, evaluate each move, take the best, then generate the next move.
- Obviously, these do not make particularly ideal solutions.
- The first method might take months or years to run, the other technique is incapable of sacrificing at early stages to gain later in the game.
- We obviously need a bit of both, but the two extremes do show us where we need to concentrate.

Generate and Test

- We need to produce a *move generator* that only produces good moves, suicidal moves and plain silly moves will get us nowhere.
- We need to develop a technique for searching through the resulting possible moves that picks out the most likely helpful moves first and tests these before going onto less beneficial moves - *it needs to be able to recognize a good move when it sees one!*

Generate and Test

- If we can develop a good move generator then that will help us generate a better evaluator.
- If the move generator produces 1000 possible moves and we have a limited time to check these, we don't leave ourselves as much scope for an intense evaluation as if we only had to look through 100.
- So we want first of all to create a very good plausible move generator that will not bother to forward unlikely solutions to the main evaluator.
- By doing this we then allow ourselves to apply a far more powerful evaluator as it would have, in this case, ten times as long to evaluate each possibility.
- We would be able to whittle down the sensible solutions much more effectively.

Game State Evaluation

- Now that we have our efficient generator and evaluator, all we need to do now, surely, is find the ideal path to a winning state and we've won the game.
- *Again, it's not as easy as this!*
- Firstly, we have to remember that our opponent will seek to slow us down and even beat us, also, with games such as chess, with an average branching factor of about 35, there are still more plausible paths than we could ever generate and evaluate.
- The way that we get round this problem is to generate a value for each state considered that gives an estimate of how good this particular position is, i.e.. how near to or far from a win this position is likely to be.

Game State Evaluation

Lets demonstrate this by looking at the game of noughts and crosses.

- We have two players and for each state we want to try to estimate how good a state is for me.
- If I am close to winning, then I want a high value, if the state is to my opponent's benefit then I want to generate a low value.

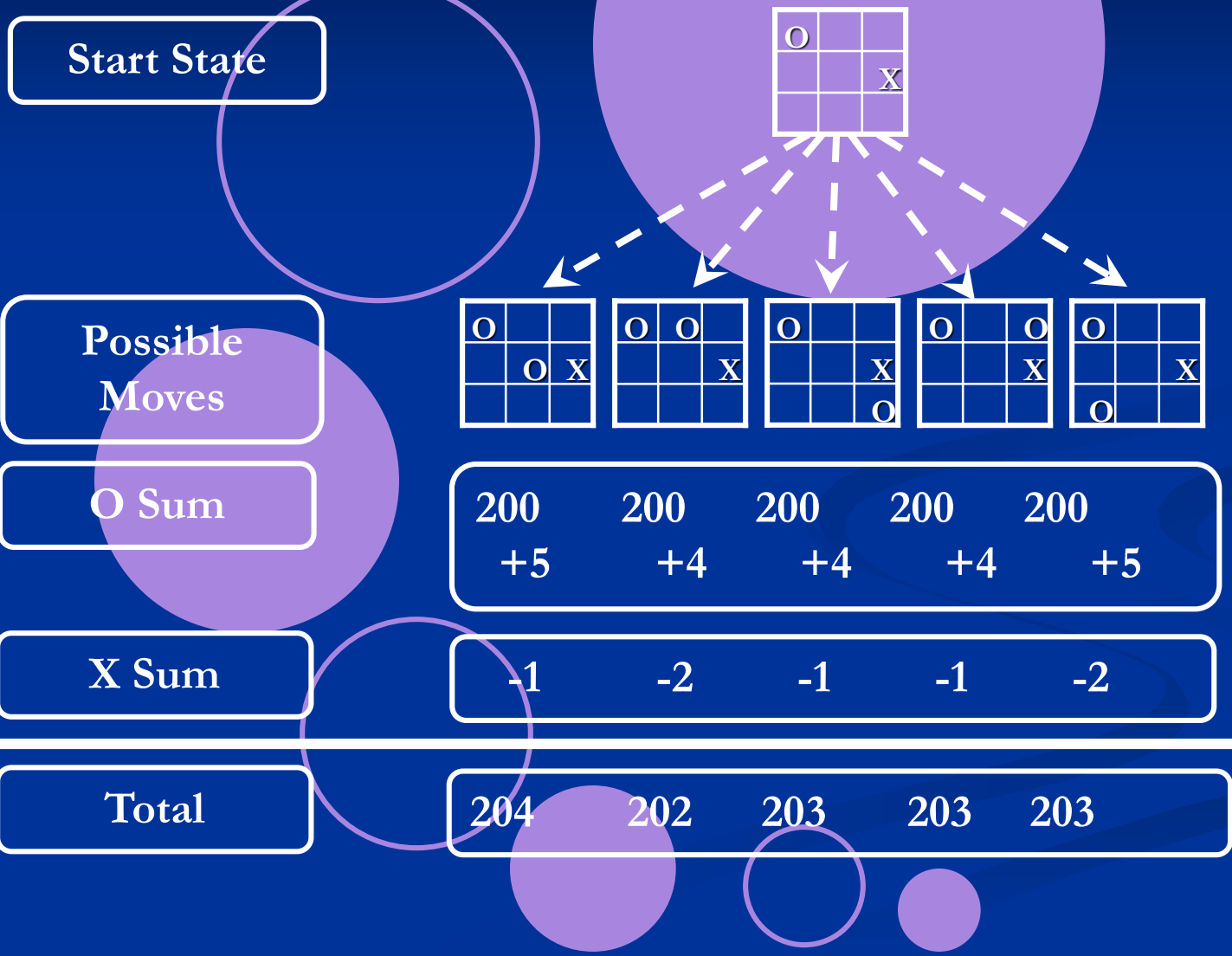
Let's assign these values to possible board states:

1. If a player has two collinear board places and the third space in the line is empty, award that player 200 points. (In other words, if you almost have one complete line that is not blocked have 200 points).
2. If a player has two nearly complete lines then score 300.
3. If a player has a complete line score 600.
4. The number of possible lines that can be completed by this player from this state.
5. Having found the value for each player, we subtract the opposition's score from our own and this is the overall score for this state.

Obviously, our desire is to maximize this value, our opposition will likewise seek to minimize it.

Game State Evaluation

The diagram shows the current state and a number of possible new states.



Game State Evaluation

- Taking the highest value we see that with this information the best move to take is the center position in the board.
- How can we be sure that this is going to produce the best possible path though?
- We need to introduce a new searching technique that will allow us to go several moves deep.