COMP 6600: TERM PROJECT ARTIFICIAL INTELLIGENCE

Computer Science and Software Engineering, Auburn University

Samir Hasan, szh0064@auburn.edu

Steffi Mariya Gnanaprakasa, smg0033@auburn.edu

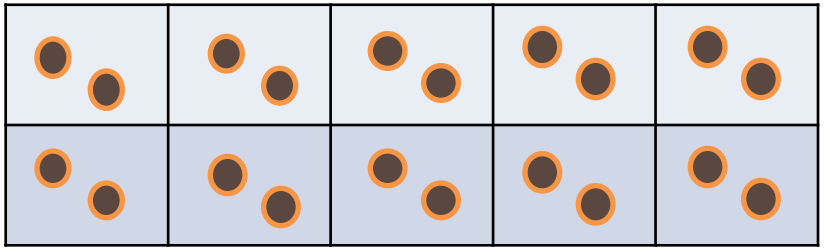
Problem Definition: Game Rules

- Two Player Board Game
- Each player has a row of squares with the same number of pebbles in each square
- A move is taking pebbles from a square and distributing them clockwise
- A player has to make a move if they have at least one non-empty square.
- If all the squares of a player are empty, the player loses the game.

Game Board

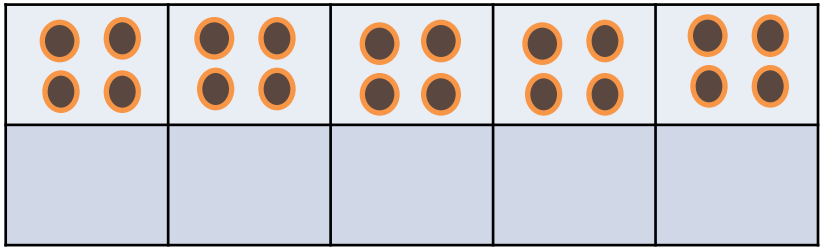
Player 1

Player 2



Player 1

Player 2



Player 1 WINS

Strategies

- And-Or
  - Applied a heuristic when a given ply (depth) is reached
- MiniMax with aB prunning
  - Implemented two different heuristics

MiniMax Heuristic – 1

- The total number of pebbles the agent has in the given the board configuration

```
function H1(board)
      //total:count of total pebbles computer has
      total = 0
      for all columns c in agent's row r in board
            total += board[r][c]

      return total
```

MiniMax Heuristic – 2

- Counting the number of ways the agent can make a move to maximize its chances of winning

```
function H2(board)
      //totalOptions:  count of non-zero cells
      totalOptions = 0
      for all column c in agent's row r in board
            if board[r][c] > 0  then totalOptions += 1

      return totalOptions
```

And-Or Heuristic

- **pseudo-goal states**
  - states in which the agent has more pebbles than opponent

- When ply is reached before an actual goal state, And-Or creates a plan to reach the pseudo-goal states instead

```
function And-Or-Heuristic(board)
        //true/false: if this is a pseudo-goal state
        countAgent = 0
        countOpponent = 0
        for all columns c in agent's row r in board
            countAgent += peeblesAt(r, c)

        for all columns c in opponent's row r
            countOpponent += peeblesAt(r, c)

        return countAgent > countOpponent
```

AND/OR Algorithm(1)

```
function getNextMove(board, player)
        gamePlan = orSearch(board, path, player, 0);
        if(gamePlan == null)
                nextMove = getRandomChoice(board, player);
        else
                if(gamePlan.isEmpty()) return null;
                nextMove = getBestMove(gamePlan);
        return nextMove;
```

AND OR Algorithm(2)

```
function orSearch(state, path, player,curPly)
        if(isGoalState(state, player))return plan;
        if(path has state)
                return null;
        for a in Actions list of Player
          plan = andSearch(Results(a),path,player, curPly);
          if(plan != null)
                return plan
        return null;
```

AND OR Algorithm(3)

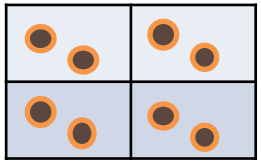
```
function andSearch(states, path, player, curPly)
        for i = 0 to states.size
            plans[i]=orSearch(states[i], path, player,curPly+1);
            if (plans[i] == null)
                    return null;
        return constructPlan(states,plan)
```

Implementation

- Command-line game
  - Human Player : based on the user input
  - And-Or strategy
  - MiniMax strategy
  - Random strategy
- UI
  - Console based Java application
  - The board state is written out to “board.txt”
  - The history of moves is written out to “moves.txt”

Tournaments

Player 1: And-Or
Player 2: aB-MiniMax



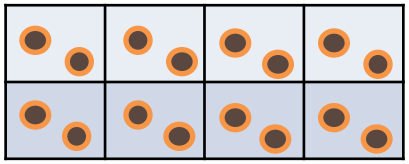
	Board size	2									
	Initial pebbles	2									
	Ply	1	2	3	4	5	6	7	8	9	10
And-Or vs aB-H1		And-Or	And-Or	And-Or	aB-H1	aB-H1	And-Or	And-Or	And-Or	And-Or	And-Or
Any repeated states?		No	No	No	No	No	No	No	No	No	No
	Ply	1	2	3	4	5	6	7	8	9	10
And-Or vs aB-H2		And-Or	And-Or	And-Or	And-Or	And-Or	And-Or	And-Or	And-Or	And-Or	And-Or
Any repeated states?		No	No	No	No	No	No	No	No	No	No

- ab-MiniMax-H1 wins only for Ply of 4 and 5. The game state space is small, so And-Or performs superior
- ab-MiniMax-H2 is worse than ab-MiniMax-H1

When state space small, And-Or is superior.

Tournaments

Player 1: And-Or
Player 2: aB-MiniMax



	Board size	4									
	Initial pebbles	2									
	Ply	1	2	3	4	5	6	7	8	9	10
And-Or vs aB-H1		aB-H1	aB-H1	aB-H1	aB-H1	aB-H1	And-Or	aB-H1	And-Or	And-Or	And-Or
Any repeated states?		No	yes	yes	yes	yes	yes	yes	yes	no	no
	Ply	1	2	3	4	5	6	7	8	9	10
And-Or vs aB-H2		aB-H2	And-Or	And-Or	And-Or	And-Or	And-Or	And-Or	And-Or	And-Or	And-Or
Any repeated states?		no	yes	yes	yes	yes	yes	yes	no	no	no

- ab-MiniMax-H1 performs better than And-Or for smaller plys. However, for larger Ply, And-Or looks ahead enough "safe" states to guarantee its victory.
- For larger Plys, And-Or wins with lesser number of turns, although each single turn takes longer than previous runs. The game ends before even seeing any repeated state.
- ab-MiniMax-H2 is still worse than ab-MiniMax-H1 or And-Or

aB-MiniMax is superior for smaller Ply, while And-Or is better at larger Plys. Since state space may grow very large, Ply can not be too high. Therefore, aB-MiniMax is a more practical strategy for the game.

Other Observations

- All strategies win against the RandomStrategy
  - The strategies work!
- The game results do actually change
  - Randomness introduced for removing repeated states
  - Some results are different if played the next hour
- However, at any given time,
  - And-Or always performs better than MiniMax for larger Plys
  - aB-MiniMax is the quickest between the two