Dance Battle - Output Analysis

**Joseph Sackett**

**CSC 480- Artificial Intelligence I**

**Feb. 2, 2014**

While the MinMax algorithm underlying this project seemed simple on the surface, it was very illuminating to implement. I struggled with the edge cases and am still not certain it is correct as the difficult test cases are hard to validate. This will briefly present my solution and analyze the output of the dance battle test cases.

The implementation contains two main functions. The first is to recursively build out the state tree of successor moves from each state. The second is to recursively implement the MinMax algorithm to determine the winning player.

Let’s look at some results:

> java ai1.battle.DanceBattle simple\_win.txt -debug

# Moves: 3

# Turns: 2

0 0

0 1

MIN: from [2,0] no moves left for: MIN

MAX: from [1,2] move to [2,0] win for: MAX

MIN: from [1,1] no winning results for: MIN

MAX: from [0,1] move to [1,1] win for: MAX

MAX:

Win

> java ai1.battle.DanceBattle simple\_win.txt -debug

# Moves: 3

# Turns: 2

0 0

0 1

MIN: from [2,0] no moves left for: MIN

MAX: from [1,2] move to [2,0] win for: MAX

MIN: from [1,1] no winning results for: MIN

MAX: from [0,1] move to [1,1] win for: MAX

MAX:

Win

Reading from bottom to top, this shows:

* MAX moves to [1,1].
* MIN tries it’s only legal move [1,2], but
* MAX moves to [2,0]
* MIN stuck.

> java ai1.battle.DanceBattle simple\_loss.txt -debug

# Moves: 3

# Turns: 1

0 1

MIN: from [0,0] no moves left for: MIN

MAX: from [2,0] move to [0,0] win for: MAX

MAX: from [0,0] no moves left for: MAX

MIN: from [2,0] move to [0,0] win for: MIN

MAX: from [2,2] no winning results for: MAX

MIN: from [1,2] move to [2,2] win for: MIN

MAX: from [1,1] no winning results for: MAX

MIN: from [0,1] move to [1,1] win for: MIN

MAX:

Lose

Reading from bottom to top, this shows:

* MIN moves to [1,1].
* MAX must move to [1,2].
* MIN realizes it should not move to [2,0].
* MIN moves to [2,2].
* MAX must move to [2,0].
* MIN moves to [0,0].
* MAX stuck.

> java ai1.battle.DanceBattle fresh.txt -debug

# Moves: 3

# Turns: 0

MIN: from [2,0] no moves left for: MIN

MAX: from [1,2] move to [2,0] win for: MAX

MIN: from [1,1] no winning results for: MIN

MAX: from [0,1] move to [1,1] win for: MAX

MAX: from [1,0] no moves left for: MAX

MIN: from [2,1] move to [1,0] win for: MIN

MIN: from [1,0] no moves left for: MIN

MAX: from [2,1] move to [1,0] win for: MAX

MIN: from [2,2] no winning results for: MIN

MAX: from [0,2] move to [2,2] win for: MAX

MIN: from [0,0] no winning results for: MIN

MAX: from [0,-1] move to [0,0] win for: MAX

MAX:

Win

Reading from bottom to top, this shows:

* MAX starts from fresh state with no turns having happened, moves to [0,0].
* MIN tries [0,1], MAX counters [1,1], MIN forced [1,2], MAX moves [2,0]. MIN stuck.
* MIN tries [0,2], MAX counters [2,2], MIN forced [2,1], MAX moves [1,0]. MIN stuck.

In the end this was a pretty cool implementation of the MinMax algorithm. It works for the cases where I am able to visibly validate. My only regret is that I did not have time to implement alpha-beta pruning.