Smart AI for 4 in a Row

Martin Mao

Yang Jiao

Zeyu Su

Introduction

Describe the game

Describe why we are doing this

–Define problem you are solving

–Describe your approach

Performance

* Win the game
* Too slow, infeasible time

Environment

* Game board 8x8 grid

Actuator

* Placing tiles on the board

Sensor

* Board stored as matrix for evaluation

Describe GUI and how we capture this. Describe parts of GUI.

Describe Problem/Constraints

Strategic game

* Adversary in Game
* Ideal moves may be blocked

Very large branching factor of moves

* Board size of 64
* Not all moves will contribute anything

Good moves are deep, need to plan ahead

* Because of nature of the game, obvious winning moves are easy to block. AI needs to plan ahead

–Define algorithm(s), define heuristic

Describe Utility Function

Utility function

* Tiles next to existing moves are more ideal
* A tile’s value increases the more tiles are in the vicinity
  + Need to chain tiles together to win

However

* Straight utility function not completely ideal. Winning moves need to have a higher value so that these moves are committed to first.

Descibe improvements, like blocking moves

Describe Algorithm

Implement an Iterative Deepening Search Tree with Alpha Beta Pruning

* Maintain a variable called depth
* and increase/decrease depth based on timing requirements
  + If there is extra time in the current search, then increase depth for next iteration, and run depth first search again.
  + If this search is taking too long, then decrease depth. The next move will use a search with a lower depth
* Alpha beta pruning to speed up searches
  + Ignore searching the rest of the children in a branch if this branch is bad
  + Maintain a beta and alpha value in searching. If beta <= alpha, we know the adversary will attempt a very good move. Do not bother to expand the rest
  + Gives algorithm more time, allowing for a larger depth

Describe speed enhancements to algorithm

Even with Alpha Beta Pruning, the algorithm is too slow when we want Iterative Deepening to increase the depth more

Not all tiles are good to search

* Avoid branching on all of 64 tiles
* Ignore illegal moves
* Restrict branching to tiles near existing tiles. A distance of 4 away (since 4 moves needed to win)

Even if some tiles are valid for expansion. Do not bother to expand if the tile’s value is too low.

* Algorithm will look all values of available tiles and determine a threshold value.
* Tiles with values below the threshold will not be expanded.

Reduce branching factor and speed up algorithm. This gives it more time to evaluate better branches to a larger depth

Improve utility function

* We were able to improve the cost function after running several instances of our initial AI

Tile location of the first move doesn’t matter too much

* + We can save time by starting the algorithm with a low depth value
  + Depth will increase on successive moves through

Tile values needed to be improved

* + Original algorithm would increase tile values for nearby tiles of same color and decrease tile values for tiles of adversary’s color
  + This led to occasional bad moves of the AI where it would place tiles very far from its own tiles and adversary’s tiles
  + Algorithm was adjusted to base tile values only it’s own color. Each tile thus had two values, it’s worth to the AI and it’s worth to the adversary
  + Had to alpha-beta pruning to take the min and max of the adversary’s tiles values and not it’s own tile’s values when pruning.

Describe issues of winning

Compared to Bad AI

* Algorithm will always win against very bad AI, such as Random AI
* It will also usually win against AI using only depth of 0
  + Depth 0 AI will not expand moves, but just find max of tile values
  + Algorithm sometimes lost if the first move was bad. We gave algorithm a low starting depth value and it would occasionally pick a bad first move

Compared to Decent AI

* Against our original AI that compared tile values of both itself and the adversary, the algorithm was very good
  + This was because the original AI had a higher chance of wasting bad moves

Compared to Players

* Compared to players, the algorithm would win sometimes
* It was very good at making plans that led to it’s own victory
  + Moves that allowed multiple possible winning moves
* However it was bad at stopping Player’s strategy of making good plans
  + It can be easy to beat if the player went first
  + AI did not make good blocking moves that could stop players early on

Describe timing issues

Timing

* Very fast if depth does not increase too much
* Speeds up of reducing branch size allowed more depth within reasonable time
* Removing speed ups would have made a high depth infeasible
  + 64 possible tiles was a huge branching factor

Space

* Storing the data was compact
* Did not search into infinity and create memory issues

–Describe/analyze properties

–Describe/analyze experimental results

–Make observations

Problems/Improvements with the Algorithm?

Conclusion

Going first has a huge advantage

Early in the game, large depth has little deciding factor

* Tiles mostly have relatively same values

Large depth matters more later in the game

* Tile values will differ more as the game progresses
* The best moves are not as obvious

Alpha beta pruning and branch reduction are necessary

* Branching factor of game is huge
* AI’s moves will take too long when depth increases
* O(bd)