CSE 332

JULY 28TH – ALPHA BETA

EXAMPLE PROBLEMS

- **Coding can be difficult**

EXAMPLE PROBLEMS

- **Coding can be difficult**
  - Let's look through some code which implements the java interfaces and get you some practice
- **Find the second smallest element in an array?**

EXAMPLE PROBLEMS

- **Coding can be difficult**
  - Let's look through some code which implements the java interfaces and get you some practice
- **Find the second smallest element in an array?**
  - What are the immediate challenges?

EXAMPLE PROBLEMS

- **Coding can be difficult**
  - Let’s look through some code which implements the java interfaces and get you some practice
- **Find the second smallest element in an array?**
  - What are the immediate challenges?
  - What does the recursive task need to return?

EXAMPLE PROBLEMS

- **Coding can be difficult**
  - Let’s look through some code which implements the java interfaces and get you some practice
- **Find the second smallest element in an array?**
  - What are the immediate challenges?
  - What does the recursive task need to return?
  - How do we break up the data?

MINIMAX

- Let's play a game of tic-tac-toe

MINIMAX

- **Let's play a game of tic-tac-toe**
  - What happened?

MINIMAX

- **Let’s play a game of tic-tac-toe**
  - What happened?
  - This game is easy, it can be easily solved

MINIMAX

- **Let's play a game of tic-tac-toe**
  - What happened?
  - This game is easy, it can be easily solved
- **Let's write a program that plays the game for us!**

MINIMAX

- **Let’s play a game of tic-tac-toe**
  - What happened?
  - This game is easy, it can be easily solved
- **Let’s write a program that plays the game for us!**
  - If the board is empty, take the middle square

MINIMAX

- **Let's play a game of tic-tac-toe**
  - What happened?
  - This game is easy, it can be easily solved
- **Let's write a program that plays the game for us!**
  - If the board is empty, take the middle square
  - If we can win, make the winning move

MINIMAX

- **Let’s play a game of tic-tac-toe**
  - What happened?
  - This game is easy, it can be easily solved
- **Let’s write a program that plays the game for us!**
  - If the board is empty, take the middle square
  - If we can win, make the winning move
  - If our opponent can win with their next move, prevent the winning move

MINIMAX

- **Let’s play a game of tic-tac-toe**
  - What happened?
  - This game is easy, it can be easily solved
- **Let’s write a program that plays the game for us!**
  - If the board is empty, take the middle square
  - If we can win, make the winning move
  - If our opponent can win with their next move, prevent the winning move
  - …

MINIMAX

- **Let’s play a game of tic-tac-toe**
  - What happened?
  - This game is easy, it can be easily solved
- **Let’s write a program that plays the game for us!**
  - If the board is empty, take the middle square
  - If we can win, make the winning move
  - If our opponent can win with their next move, prevent the winning move
  - …
  - How long would this program be? There are many possible tic-tac-toe boards

MINIMAX

- **Let’s play a game of tic-tac-toe**
  - What happened?
  - This game is easy, it can be easily solved
- **Let’s write a program that plays the game for us!**
  - If the board is empty, take the middle square
  - If we can win, make the winning move
  - If our opponent can win with their next move, prevent the winning move
  - …
  - How long would this program be? There are many possible tic-tac-toe boards
  - What if I told you to program a computer to play checkers?

MINIMAX

- Checkers

MINIMAX

- **Checkers**
  - Still a “solved” game

MINIMAX

- **Checkers**
  - Still a “solved” game (for the standard starting positions)

MINIMAX

- **Checkers**
  - Still a “solved” game (for the standard starting positions)
  - Do we want to write an inclusive bot player?

MINIMAX

- **Checkers**
  - Still a “solved” game (for the standard starting positions)
  - Do we want to write an inclusive bot player?
    - No.

MINIMAX

- **Checkers**
  - Still a “solved” game (for the standard starting positions)
  - Do we want to write an inclusive bot player?
    - No.
  - What do checkers and tic-tac-toe have in common?

MINIMAX

- **Checkers**
  - Still a “solved” game (for the standard starting positions)
  - Do we want to write an inclusive bot player?
    - No.
  - What do checkers and tic-tac-toe have in common?
    - Turn based

MINIMAX

- **Checkers**
  - Still a “solved” game (for the standard starting positions)
  - Do we want to write an inclusive bot player?
    - No.
  - What do checkers and tic-tac-toe have in common?
    - Turn based
    - Zero sum (in the final board, the two players receive either (-1,1) or (1,-1). The outcomes for both players always sum to one. You cannot win unless your opponent fails

MINIMAX

- **Zero sum**

MINIMAX

- **Zero sum**
  - Important consideration:
    - Many games are zero sum games, tic-tac-toe, chess, checkers

MINIMAX

- **Zero sum**
  - Important consideration:
    - Many games are zero sum games, tic-tac-toe, chess, checkers
  - Some are not

MINIMAX

- **Zero sum**
  - Important consideration:
    - Many games are zero sum games, tic-tac-toe, chess, checkers
  - Some are not
    - Prisoner’s dilemma

MINIMAX

- **Zero sum**
  - Important consideration:
    - Many games are zero sum games, tic-tac-toe, chess, checkers
  - Some are not
    - Prisoner’s dilemma
    - Communicating strategy
  - Some are zero sum, but are more complicated than turned based

MINIMAX

- **Zero sum**
  - Important consideration:
    - Many games are zero sum games, tic-tac-toe, chess, checkers
  - Some are not
    - Prisoner’s dilemma
    - Communicating strategy
  - Some are zero sum, but are more complicated than turned based
    - Ultimatum game
    - Derivatives

MINIMAX

- **Back to our “checkers” problem**

MINIMAX

- **Back to our "checkers" problem**
  - How would an ideal checkers bot act?

MINIMAX

- **Back to our “checkers” problem**
  - How would an ideal checkers bot act?
    - Similar to our *naïve* tic-tac-toe bot, it should make the optimal move at any given time

MINIMAX

- **Back to our “checkers” problem**
  - How would an ideal checkers bot act?
    - Similar to our *naïve* tic-tac-toe bot, it should make the optimal move at any given time
    - How do we avoid hard coding this?

MINIMAX

- **Back to our “checkers” problem**
  - How would an ideal checkers bot act?
    - Similar to our *naïve* tic-tac-toe bot, it should make the optimal move at any given time
    - How do we avoid hard coding this?
    - Mini-max algorithm

MINIMAX

- **Back to our "checkers" problem**
  - How would an ideal checkers bot act?
    - Similar to our *naïve* tic-tac-toe bot, it should make the optimal move at any given time
    - How do we avoid hard coding this?
    - Mini-max algorithm

MINIMAX

- **Computers are good at calculating, we don’t want to hard code the decisions in advance**

MINIMAX

- **Computers are good at calculating, we don't want to hard code the decisions in advance**
  - Recognize that there is a give and take and that there is an advantage in seeing how moves will play out

MINIMAX

- **Computers are good at calculating, we don't want to hard code the decisions in advance**
  - Recognize that there is a give and take and that there is an advantage in seeing how moves will play out
  - How can we know how our moves are going to play out?

MINIMAX

- **Computers are good at calculating, we don't want to hard code the decisions in advance**
  - Recognize that there is a give and take and that there is an advantage in seeing how moves will play out
  - How can we know how our moves are going to play out?
  - We also need to know how are opponent is going to react

MINIMAX

- **Computers are good at calculating, we don't want to hard code the decisions in advance**
  - Recognize that there is a give and take and that there is an advantage in seeing how moves will play out
  - How can we know how our moves are going to play out?
  - We also need to know how are opponent is going to react
  - How can we predict this?

MINIMAX

- **Computers are good at calculating, we don't want to hard code the decisions in advance**
  - Recognize that there is a give and take and that there is an advantage in seeing how moves will play out
  - How can we know how our moves are going to play out?
  - We also need to know how are opponent is going to react
  - How can we predict this?
  - Assuming the game is symmetrical (I am playing the same game as my opponent, which is not always true) I calculate what would be the best possible move for my opponent in that scenario (which is the worst move for me)

MINIMAX

- **Consider a decision tree around tic-tac-toe**

MINIMAX

- **Consider a decision tree around tic-tac-toe**
  - Notice that it expands very quickly, at turn *k,* we have 9-k possible choices.

MINIMAX

- **Consider a decision tree around tic-tac-toe**
  - Notice that it expands very quickly, at turn *k,* we have 9-k possible choices.
  - If we take all of our “decisions” to their conclusion, how much have we actually calculated?

MINIMAX

- **Consider a decision tree around tic-tac-toe**
  - Notice that it expands very quickly, at turn *k,* we have 9-k possible choices.
  - If we take all of our "decisions" to their conclusion, how much have we actually calculated?
  - First move has 9 options, second move has 8…

MINIMAX

- **Consider a decision tree around tic-tac-toe**
  - Notice that it expands very quickly, at turn *k,* we have 9-k possible choices.
  - If we take all of our “decisions” to their conclusion, how much have we actually calculated?
  - First move has 9 options, second move has 8…
  - This is n! options! (This assumes we can calculate a board in constant time)

MINIMAX

- What about a decision tree around checkers?

MINIMAX

- **What about a decision tree around checkers?**
  - At the beginning, there are the same “set” of moves available, but these sets are constantly changing and always dependent on future moves

MINIMAX

- **What about a decision tree around checkers?**
  - At the beginning, there are the same “set” of moves available, but these sets are constantly changing and always dependent on future moves
  - Branching factor for checkers is about 10, how many calculations do we need?

MINIMAX

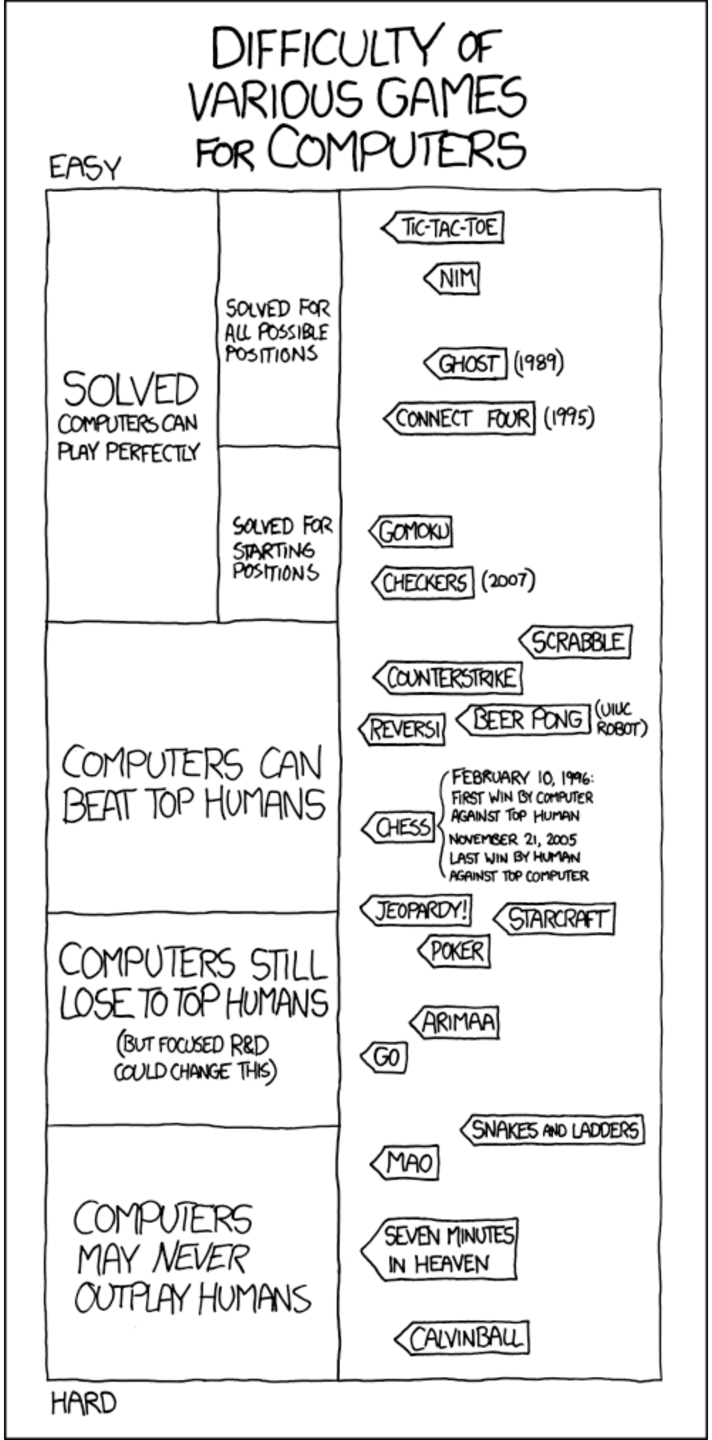
- **What about a decision tree around checkers?**
  - At the beginning, there are the same “set” of moves available, but these sets are constantly changing and always dependent on future moves
  - Branching factor for checkers is about 10, how many calculations do we need?
  - Depends on how many moves it takes for us to complete the game, we’re computing 10^t situations, where t is the number of turns

MINIMAX

- **How long would it take to "solve" chess using this method?**
  - The average branching factor is 35
  - The average number of moves in a game is 40
  - The number of end moves we need to calculate is 35^{40}
  - Even if we could calculate the win-loss outcome of one trillion boards in a second, it would take 10^{40} years

MINIMAX

- **How long would it take to "solve" chess using this method?**
  - The average branching factor is 35
  - The average number of moves in a game is 40
  - The number of end moves we need to calculate is 35^{40}
  - Even if we could calculate the win-loss outcome of one trillion boards in a second, it would take 10^{40} years
  - The universe is ~10^{10} old


DIFFICULTY OF VARIOUS GAMES FOR COMPUTERS
EASY
SOLVED
COMPUTERS CAN PLAY PERFECTLY
SOLVED FOR ALL POSSIBLE POSITIONS
SOLVED FOR STARTING POSITIONS
COMPUTERS CAN BEAT TOP HUMANS
COMPUTERS STILL LOSE TO TOP HUMANS
(BUT FOCUSED R&D COULD CHANGE THIS)
COMPUTERS MAY NEVER OUTPLAY HUMANS
HARD
TIC-TAC-TOE
NIM
GHOST (1989)
CONNECT FOUR (1995)
GOMOKU
CHECKERS (2007)
SCRABBLE
COUNTERSTRIKE
REVERSI
BEER PONG (UIUC ROBOT)
CHESS
FEBRUARY 10, 1996: FIRST WIN BY COMPUTER AGAINST TOP HUMAN
NOVEMBER 21, 2005 LAST WIN BY HUMAN AGAINST TOP COMPUTER
JEOPARDY!
STARCRAFT
POKER
ARIMAA
GO
SNAKES AND LADDERS
MAO
SEVEN MINUTES IN HEAVEN
CALVINBALL

MINIMAX

- **With games with high branching factor (Chess' is ~35), we cannot reasonably calculate all factors and all moves (if we did, the game would be "solved")**

MINIMAX

- **With games with high branching factor (Chess' is ~35), we cannot reasonably calculate all factors and all moves (if we did, the game would be "solved")**
  - At some point, we require an estimator (for P3) we will provide it.

MINIMAX

- **With games with high branching factor (Chess' is ~35), we cannot reasonably calculate all factors and all moves (if we did, the game would be "solved")**
  - At some point, we require an estimator (for P3) we will provide it.
  - We want to look as far into the future as we can, but at a certain point, we need to look at the board and make a reasonable assessment of the pieces

MINIMAX

- **With games with high branching factor (Chess' is ~35), we cannot reasonably calculate all factors and all moves (if we did, the game would be "solved")**
  - At some point, we require an estimator (for P3) we will provide it.
  - We want to look as far into the future as we can, but at a certain point, we need to look at the board and make a reasonable assessment of the pieces
  - Remember, since this is a zero sum game, our accessment can (and should be a partial assignment)
  - For example, we can access a board to be (-.85,.85) to indicate that we estimate player 2 has an 85% chance of winning

MINIMAX

- **So, our strategy now is**

MINIMAX

- **So, our strategy now is**
  - Look as far down the decision tree as we can

MINIMAX

- **So, our strategy now is**
  - Look as far down the decision tree as we can
  - Assume that our opponent is playing as well as they can

MINIMAX

- **So, our strategy now is**
  - Look as far down the decision tree as we can
  - Assume that our opponent is playing as well as they can
  - Choose the decision at our current depth that gives us the highest expected win-percentage

MINIMAX

- **So, our strategy now is**
  - Look as far down the decision tree as we can
  - Assume that our opponent is playing as well as they can
  - Choose the decision at our current depth that gives us the highest expected win-percentage
  - Repeat until the game is over

MINIMAX

- **So, our strategy now is**
  - Look as far down the decision tree as we can
  - Assume that our opponent is playing as well as they can
  - Choose the decision at our current depth that gives us the highest expected win-percentage
  - Repeat until the game is over
- **Do we need to consider all of the boards?**

MINIMAX

- **So, our strategy now is**
  - Look as far down the decision tree as we can
  - Assume that our opponent is playing as well as they can
  - Choose the decision at our current depth that gives us the highest expected win-percentage
  - Repeat until the game is over
- **Do we need to consider all of the boards?**
  - In tic-tac-toe, once we recognize that our move lets the opponent win, do we need to consider any other opposing moves?

ALPHA-BETA PRUNING

- **We can prune the number of boards that we actually need to analyze**

ALPHA-BETA PRUNING

- **We can prune the number of boards that we actually need to analyze**
  - Let’s look at tic-tac-toe again as an example

ALPHA-BETA PRUNING

- **We can prune the number of boards that we actually need to analyze**
  - Let's look at tic-tac-toe again as an example
- **We use alpha and beta to recognize the current boundaries for our best performance**

ALPHA-BETA PRUNING

- **We can prune the number of boards that we actually need to analyze**
  - Let’s look at tic-tac-toe again as an example
- **We use alpha and beta to recognize the current boundaries for our best performance**
  - If our opponent can do better, we know that they will and that that path in the decision tree won’t happen

ALPHA-BETA PRUNING

- **We can prune the number of boards that we actually need to analyze**
    - Let’s look at tic-tac-toe again as an example
- **We use alpha and beta to recognize the current boundaries for our best performance**
    - If our opponent can do better, we know that they will and that that path in the decision tree won’t happen
    - If we could do better in an earlier move, we don’t want to consider poor moves

ALPHA-BETA PRUNING

- **We can prune the number of boards that we actually need to analyze**
  - Let's look at tic-tac-toe again as an example
- **We use alpha and beta to recognize the current boundaries for our best performance**
  - If our opponent can do better, we know that they will and that that path in the decision tree won't happen
  - If we could do better in an earlier move, we don't want to consider poor moves

ALPHA-BETA PRUNING

- Let’s consider the tic-tac-toe decision tree again

ALPHA-BETA PRUNING

- **Let's consider the tic-tac-toe decision tree again**
  - While the first move is irrelevant, early on, there are other moves that we can eliminate, although not necessarily all

ALPHA-BETA PRUNING

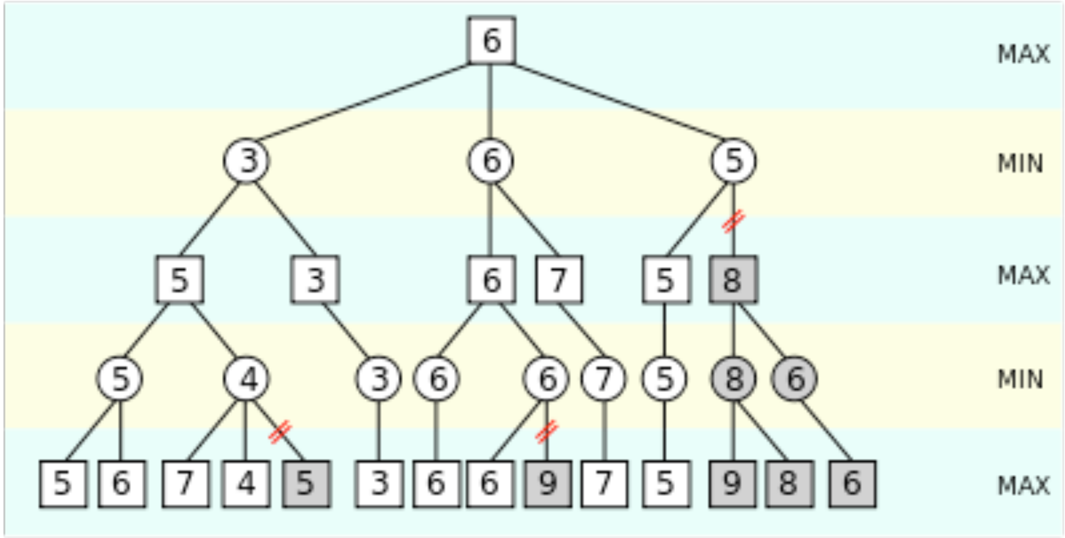
- **Let's consider the tic-tac-toe decision tree again**
  - While the first move is irrelevant, early on, there are other moves that we can eliminate, although not necessarily all
  - If we consider moves in order (and what that order is actually ends up being important), if our best move is the last one we consider, and our opponents best move is the last one we consider from that, we could actually avoid pruning all together

ALPHA-BETA PRUNING

- Consider an arbitrary min/max game

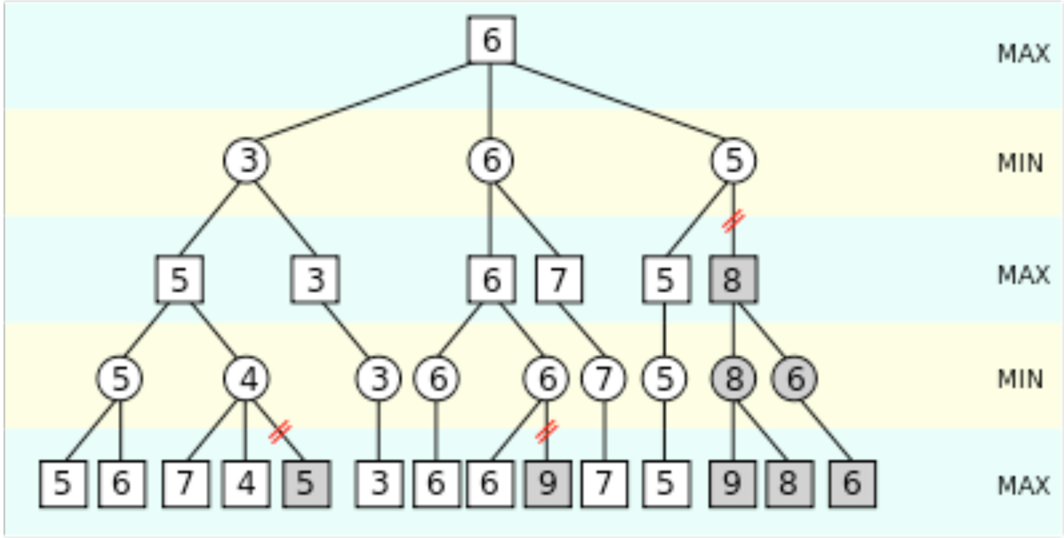
ALPHA-BETA PRUNING

- **Consider an arbitrary min/max game**



ALPHA-BETA PRUNING

- Consider an arbitrary min/max game



- The “max” player wants to maximize the score, whereas the “min” player wants to minimize the score. This arrangement can be reduced to zero sum format (there is no “all-winners” scenario.

MINIMAX

- **What's the big assumption here?**

MINIMAX

- **What's the big assumption here?**
  - That our opponent is also good at chess

MINIMAX

- **What's the big assumption here?**
  - That our opponent is also good at chess
- **What could go wrong?**

MINIMAX

- **What's the big assumption here?**
  - That our opponent is also good at chess
- **What could go wrong?**
  - They have a better board predictor

MINIMAX

- **What's the big assumption here?**
  - That our opponent is also good at chess
- **What could go wrong?**
  - They have a better board predictor
  - Our opponent can predict exactly how we will behave
- **Consider a poker game**

MINIMAX

- **What's the big assumption here?**
  - That our opponent is also good at chess
- **What could go wrong?**
  - They have a better board predictor
  - Our opponent can predict exactly how we will behave
- **Consider a poker game**
  - Suppose no communication is allowed at the table, you cannot see or hear anything your opponent says

MINIMAX

- **What's the big assumption here?**
  - That our opponent is also good at chess
- **What could go wrong?**
  - They have a better board predictor
  - Our opponent can predict exactly how we will behave
- **Consider a poker game**
  - Suppose no communication is allowed at the table, you cannot see or hear anything your opponent says
  - Can you still predict their behavior?

MINIMAX

- **What’s the big assumption here?**
  - That our opponent is also good at chess
- **What could go wrong?**
  - They have a better board predictor
  - Our opponent can predict exactly how we will behave
- **Consider a poker game**
  - Suppose no communication is allowed at the table, you cannot see or hear anything your opponent says
  - Can you still predict their behavior?
  - Poker is a turn-based, zero-sum game, do we advise minimax?

MINIMAX

- **Chess, checkers and tic-tac-toe also have "perfect information"**
  - There is no way to deceive your opponent about the state of the game

MINIMAX

- **Chess, checkers and tic-tac-toe also have "perfect information"**
  - There is no way to deceive your opponent about the state of the game
  - All players have equal understanding of the rules and of the current game state

MINIMAX

- **Chess, checkers and tic-tac-toe also have "perfect information"**
  - There is no way to deceive your opponent about the state of the game
  - All players have equal understanding of the rules and of the current game state
- **Minimax works on all these types of games, the interchangeable part comes from the board comparator, we need to know something about what's good/bad**

MINIMAX

- **Also, we can see that it's most important to make good decisions early in the game**

MINIMAX

- **Also, we can see that it's most important to make good decisions early in the game**
  - The first turn is when it costs the most to determine which move is best, but it also has the biggest impact on how the game will play out

MINIMAX

- **Also, we can see that it's most important to make good decisions early in the game**
  - The first turn is when it costs the most to determine which move is best, but it also has the biggest impact on how the game will play out
  - Select a restricted starting sequence to simplify (all chess boards start the same way)

MINIMAX

- **Also, we can see that it’s most important to make good decisions early in the game**
  - The first turn is when it costs the most to determine which move is best, but it also has the biggest impact on how the game will play out
  - Select a restricted starting sequence to simplify (all chess boards start the same way)
  - Not a coincidence that chess strategy books start with classic openings, humans don’t think that differently from computers.

NEXT CLASS

- Parallelizing this process

NEXT CLASS

- Parallelizing this process
- More examples of alpha-beta pruning

NEXT CLASS

- Parallelizing this process
- More examples of alpha-beta pruning
- Alternatives to map and reduce

NEXT CLASS

- **Parallelizing this process**
- **More examples of alpha-beta pruning**
- **Alternatives to map and reduce**
  - Is sorting really a map or a reduction?

PROJECT 3

- Project 3 goes out tonight

PROJECT 3

- **Project 3 goes out tonight**
  - Checkpoint 1 will be due next Friday

PROJECT 3

- **Project 3 goes out tonight**
  - Checkpoint 1 will be due next Friday
  - You will need to have minimax completed by then (but we’ve provided a lot of helper code)

PROJECT 3

- **Project 3 goes out tonight**
  - Checkpoint 1 will be due next Friday
  - You will need to have minimax completed by then (but we've provided a lot of helper code)
- **Multiple parallelism assignments will also go out tonight**

PROJECT 3

- **Project 3 goes out tonight**
  - Checkpoint 1 will be due next Friday
  - You will need to have minimax completed by then (but we've provided a lot of helper code)
- **Multiple parallelism assignments will also go out tonight**
  - They will work like projects, you will clone a gitlab repo and push code to the server, however these assignments will be individual work

PROJECT 3

- **Project 3 goes out tonight**
  - Checkpoint 1 will be due next Friday
  - You will need to have minimax completed by then (but we've provided a lot of helper code)
- **Multiple parallelism assignments will also go out tonight**
  - They will work like projects, you will clone a gitlab repo and push code to the server, however these assignments will be individual work
  - 2 simple ones will go out tonight and 2 more complicated ones will go out next week