AI-m of the Game

Workshop #4: Professional Game AI

Project Ignite Instructors:
Brandon Wang, Brandon Wei

Recap

What happened last time?

Optimizations

- optimization = any modification to give code better ...
    - quality (ie. clearer, more concise code)
    - time efficiency (ie. computes in less time)
    - space efficiency (ie. uses up less memory space)
- related to the "Big-O Notation" (ie. the time and space usage of a function in the **worst case**)
    - the more optimal the code, the smaller the Big-O!

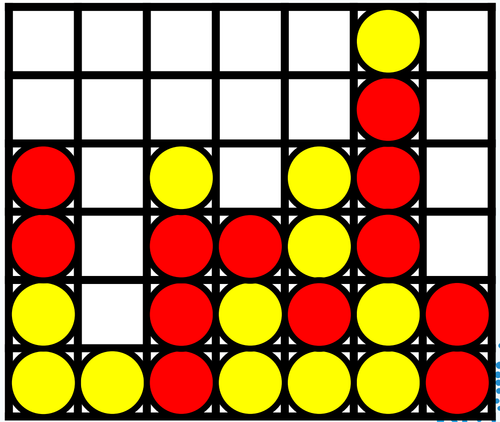
Game Optimizations

1. OOP makes the code clearer and more organized
2. Heuristics will better estimate a state than just 0
3. Better move ordering lets us prune more worse moves
4. Transposition tables store and load states and their values that we came across earlier
5. Zero windows quickly find the upper and lower bounds of the true minimax value
6. Iterative deepening lets us explore shallow depths first before exploring further depths
7. Bitboards are time and space efficient

Connect Four

Connect Four is much more complicated, so we combined multiple optimizations into an algorithm called MTD(f)

- connectfour.py isn't perfect,e but efficient enough to make very good moves in a reasonable amount of time



Student Demonstrations

Your game optimizations!

Demonstration Order

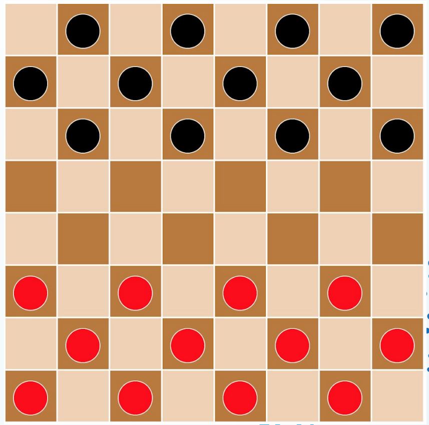
1. Vishaal Komaragir
2. Philip Papurt
3. Victor Cheng
4. Varun Damarla
5. Abhay Syamala
6. Rushil Kakkad
7. Vikram Nagarajan
8. Anirudh Govinday
9. Subham Sahoo

Complicated Combinatorial Games

Connect Four was the tip of the iceberg...

Checkers

- very simple rules and board positions
- ≈3 legal moves per checkers position, and lasts ≈70 moves
- 5×10^{20} game states
- a solved game that has been proved to be a draw under best play



Chess

- many rules and states to remember
- ≈30 legal moves per chess position, and lasts ≈80 moves
- upwards of 10^{45} game states, and at least 10^{120} unique chess games ("Shannon number")
    - there's 10^{80} atoms in the universe!
- estimated to take 10^{90} years for a supercomputer to fully solve
    - one of the first major goals of AI

Go

- 19 × 19 board (in tournaments)
- ≈250 legal moves per position, and lasts ≈150 moves (pros play for 1-6 hours!!!)
- simpler rules and tactics, and no piece hierarchies
    - though requires a *lot* of counting and knowing patterns
- upwards of 10^{170} game states, and $10\char`^(10^{48})$ - $10\char`^(10^{171})$ unique Go games

Famous Game AI

The truest champions of gaming

Chinook (1990)

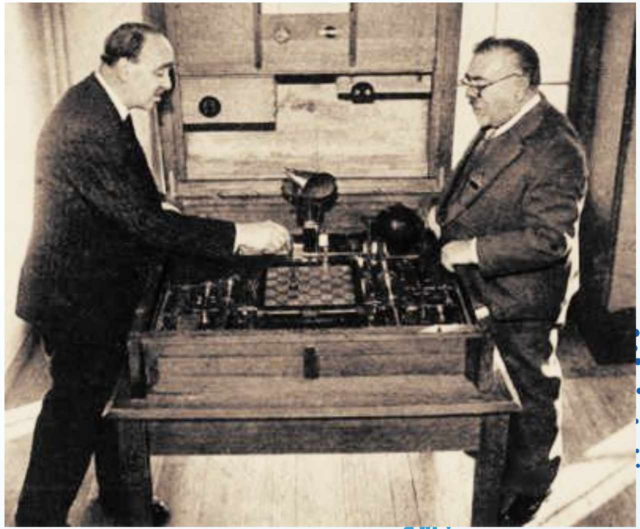
- created by a small team from University of Alberta
- 1994: “defeated” Marion Tinsley (world champion for 40 straight years, GOAT)
    - 6 draws, but Tinsley withdrew b/c of pancreatic cancer
    - the first time a computer program won a human world championship
- 1995: defeated another pro Don Laferty
    - 1 win, 0 losses, 31 draws

How Chinook Works

- Chinook used:
  - a deep search algorithm
  - a good move evaluation function
    - including piece count, kings count, trapped kings, turn, and runaway checkers
  - an end-game database for all positions w/ ≤8 pieces
    - hardcoded instead of learned
    - database was eventually increased until checkers was completely solved

Turochamp (1951)

- an algorithm created by Alan Turing ("father of computer science") and his colleague
- predated even before AI was a term
- the first chess program, even though it never ran on a computer
    - Turing would spend 30 min per move calculating by hand!



How Turochamp Works

- very crude calculations w/o even using a computer!
    - could only think 2 moves in advance, whereas world champion Garry Kasparov could see 3-5 on average
- at a very low level, calculates and scores all possible moves and opponent's responses
    - more of a tedious math algorithm than a program
- we don't know much else b/c the original algorithm is lost

Deep Blue II (1997)

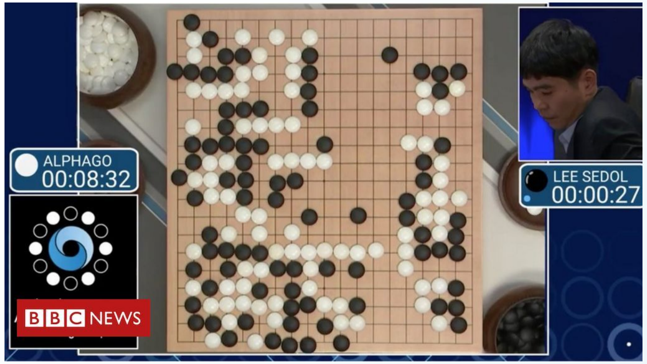
- created by IBM, and began as a CMU graduate student's dissertation project
- successor to Deep Thought (1991) and Deep Blue I (1996)
- defeated Kasparov
  - 2 wins, 1 loss, 3 draws
  - the first time AI exceeded human play in chess

How Deep Blue II Works

- parallel computer w/ 30 processor nodes, each w/ a CPU and 16 chess chips
    - up to 200 million positions per second
    - searches up to 40 moves deep
- first, plays a standard opening
- then, follows 3 steps:
    - search all possible moves and positions w/ brute force and optimizations
    - evaluate and rank the positions
        - uses an evaluation function w/ 8000 features
    - decides best moves based on ranks and playbooks

AlphaGo (2016)

- created by DeepMind Technologies, which was acquired by Google
- defeated top 3 Go player Lee Sedol
  - 4 wins, 1 loss
- made headlines around the world



How AlphaGo Works

- single machine with 48 CPUs, 8 GPUs, and 4 TPUs (= an AI circuit made by Google specifically for machine learning)
- how AlphaGo finds the best moves:
    - 2 neural networks trained to find the best moves from from 160,000 grandmaster games
    - Monte Carlo tree search (MCTS) to solve the game tree by exploring random moves and weighting them
- plays very conservatively (ie. safe moves that barely give a higher score)
- surprisingly, AlphaGo didn’t use any ground-breaking new algorithms, just basic neural nets!

Other DeepMind Game AIs

- AlphaZero: a general program and successor to AlphaGo
  - mastered shogi, chess, and Go from scratch using reinforcement learning
- Agent57: better than humans in 57 Atari2600 games
  - including Pong, Donkey Kong, Space Invaders, Frogger, Q*bert, Mario Bros, Pac-Man
- MuZero: everything AlphaZero and Agent57 can do, but even more powerful
  - uses much better TPUs than AlphaZero
- AlphaStar: better than 99.8% of players in StarCraft II
  - beat pro player Mana 5-0

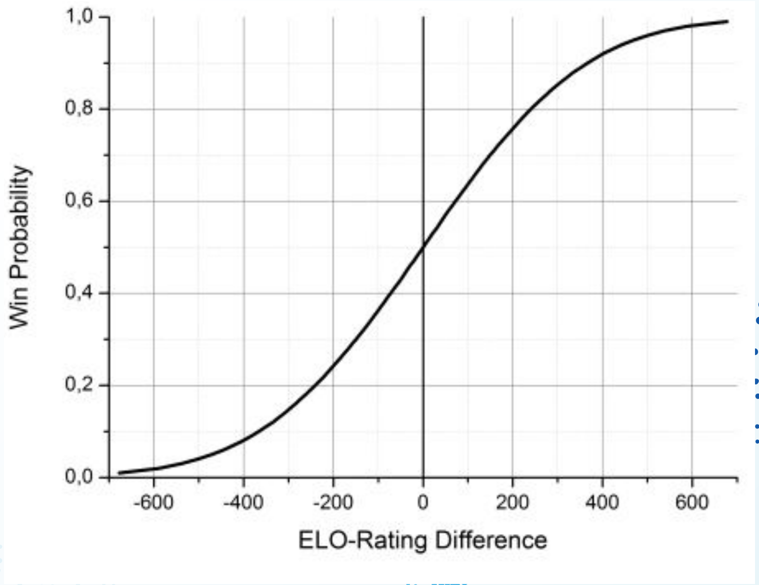
Elo Ratings

Measuring how much of a pro-gamer you are

Elo Rating System

Elo rating = way to measure skill levels of players in zero-sum games (eg. checkers, chess, Go)

- created by physics professor Arpad Elo
- it's even used in football, basketball, baseball, and esports



Checkers

Name	**Year**	**Elo Ranking**
final Chinook program	current	infinity (completely solved checkers)
first Chinook program	1995	2814
Marion Tinsley (former world champion)	1995	unknown

Chess

Name	**Year**	**Elo Ranking**
AlphaZero (best chess engine)	current	3500+
Stockfish (best free chess engine)	current	3400+
Magnus Carlsen (current world champion)	2014 (at his peak)	2882 (at his peak)
Garry Kasparov (former world champion)	1999 (at his peak)	2851 (at his peak)
Deep Blue II	1997	≈2800

Go

Name	**Year**	**Elo Ranking**
AlphaGo Zero (best Go engine)	Oct 2017	5185
AlphaZero (beat AlphaGo Zero 60:40)	Dec 2017	5018
Shin Jinseo (current world champion)	current	3825
AlphaGo Lee (beat Lee Sedol 4-1)	2016	3739
Lee Sedol (18-time world champion)	2016	3573

What the Elo rankings tell us

AI will *always* be much, much better than human players

- given enough data, engineering, hardware, and time
    - as more and more research is done, AI will continue to rapidly grow until it is unstoppable
- AI can be applied anywhere, even outside of turn-based games
    - any game you can think of can be mastered by AI

Workshops Takeaways

What you should know from all of our workshops

Games are diverse

there are many, many classifications of games

- some are random, some are turn-based, some hide some information, some aren’t even competitive
  - we covered combinatorial strategy games
- games have different winning strategies, which can be very very easy (eg. Nim) or very very hard (eg. chess)

Game strategies are methodical

First, try to mathematically solve for the best moves

- eg. takeaway game and Nim just require simple math

Next, go through every single continuation of the game

- ie. minimax algorithm
- never fails, but may take forever to give the solution

Finally, if you can't search the whole tree, predict the best move

- use heuristics, probability, and/or past games to estimate a position's score

Optimizations save time and space

- **alpha-beta pruning**
- **object-oriented programming**
- **heuristics**
- **better move ordering**
- **transposition tables**
- **zero windows**
- **iterative deepening**
- **bitboards**

- quiescence search
- selectivity
- best-first search
- random search
- parallelism/concurrency
- pondering during opponent’s turn
- finding optima using gradients
- neural net training

Complex games need complex AI

using optimized minimax is tough for very complex games

- esp. chess and Go
- and what about real-time games like PvPs, FPSs, RTSs, MOBAs??

modern approaches to game AI

- very powerful hardware (use multi-core CPUs and GPUs)
- machine learning (= train your program to learn or copy from thousands or millions of previous games)
    - neural networks use 1000s of features to analyze a position, then return a best move

Thanks for Listening!

Finally, all the lectures are completed!

Recap Preparation

Create a few slides (no more than 4) briefly describing your accomplishments during the past few workshops

- use this blank Google Slides
- summarize what you learned and wrote in code
- prepare to demonstrate your code, if you want to

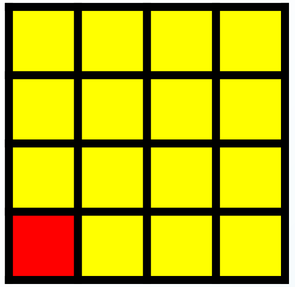
As a group, spend 1 hour to complete these slides!

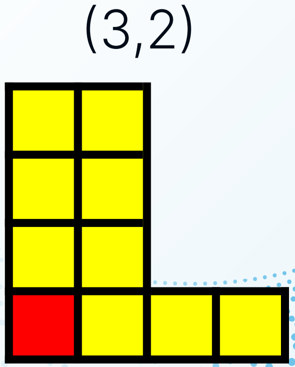
Before we go...

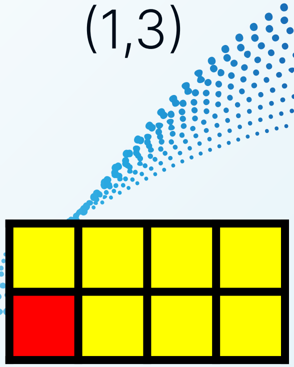
A quick “snack” before your final project

Rules of Chomp

- any grid of squares
- players take turns removing 1 square, and all squares above and to the right
    - cannot take bottom-left; assume it's "poisoned"
- the player that's left with the bottom-left square loses!

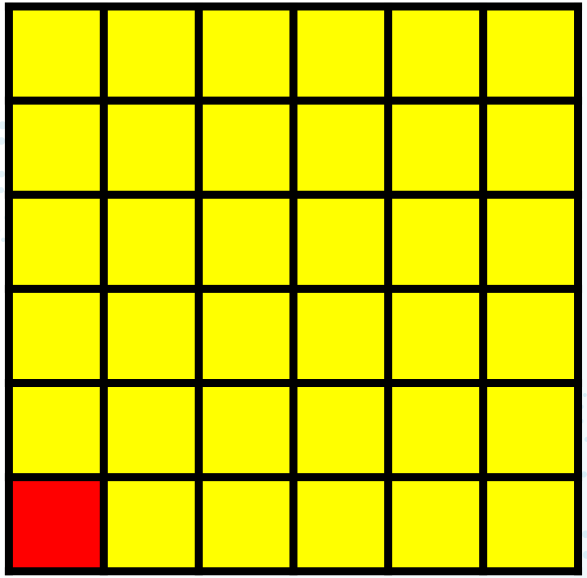






Let’s Play Chomp!

Choose a square (except red), which removes it and all square up and right. Leave your opponent w/ just red!



Chomp Team Activity

Team 1

- Philip Papurt
- Victor Cheng
- Anirudh Govinday
- Vishaal Komaragir
- Subham Sahoo

Team 2

- Vikram Nagarajan
- Varun Damarla
- Rushil Kakkad
- Abhay Syamala

Chomp Competition Rules (1)

Download chomp_util.py (Board, GUI, and Player classes) and chomp.py (Game class and main method)

- briefly scan the program's structure - don't worry about how each function is explicitly implemented
- in chomp_util.py, as a team, fill out AIStrat1 or AIStrat2 using any AI method/s you want
- in chomp.py, play around with different boards and player strategies in the main method

Chomp Competition Rules (2)

Next week, we'll hold a competition after the presentation

- briefly explain what your AI strategy is
- Teams 1 and 2 will face off on the following boards:
    - (5,4): width=5, height=4
    - (12,8): width=12, height=8
    - (20,18): width=20, height=18
    - custom boards if there are ties
- each team alternates between Player 1 and Player 2

Chomp Competition Rules (3)

Things your team's AI strategy should do:

- use the provided or your own helper functions
  - please end your helper function's names with your team number (eg. minimax_pruning1(), mtdf2())
- return a valid move within 20 seconds
  - even if the search isn't done or there's no good move

Things your team's AI strategy should NOT do:

- modify or delete any other code
- use external files, fields, or classes
- use over 10 MB of hardware space

Chomp References

- https://github.com/kadartamascsaba/chomp-game
- https://www.math.wisc.edu/wiki/images/Chomp_Sol.pdf
- http://www.koreascience.or.kr/article/JAKO201821464987344.pdf

To-do

Things to do by the next workshop:

- Work with your team to code up a Chomp AI strategy
- Think of a game you want to solve for the final project

Thanks for coming!