Course Overview

CMPUT 355: Games, Puzzles, and Algorithms

Games, Puzzles, and Algorithms

- Introduction to the algorithms and theory behind programs that solve puzzles and games
- **Puzzles:** Find a sequence of actions to transform a starting position to a goal position
  - Key technique: **search**
  - Broadly applicable to non-game problems too (circuit layout, route finding, etc.)
- **Games:** An opponent gets to choose some of the actions
  - Search is still the key technique
  - But need to somehow account for the other player's moves

Why Games?

- Games are a classic artificial intelligence **challenge problem**
  - Reasoning about an opponent is hard!
  - Many AI advances start with game applications
- **Strategic reasoning** is important
  - Strategic interactions are fundamentally different from single-agent interations
  - In applications, frequently need to account for other agents
  - Real situations are often competitive
- Games are fun!

About Me

James R. Wright https://jrwright.info

- Associate Professor of Computing Science
- Amii Fellow
- Research at intersection of:
    1. Strategic Interactions
    2. Machine Learning
    3. Behavioral Modeling

Lecture Outline

1. Course Logistics
2. Course Topics

After this lecture, you should be able to:

- Understand the policies and procedures of the course
- Describe the basic overview of the course material
- Decide whether you want to take the course

Course Essentials

Course information: https://jrwright.info/gpaclass/

- This is the main source of class information
- Syllabus, slides, practice quizzes, deadlines

Lectures: Mondays, Wednesdays, Fridays 4:00-4:50pm in CCIS L2-190

- In person

Canvas: https://canvas.ualberta.ca/courses/32227

- Discussion forum for questions about course material

Email: james.wright@ualberta.ca for **private** questions

- If you ask about course material by email I will redirect you to the discussion forum

Office hours: After lecture, or by appointment

Evaluation

Grade breakdown

- Final exam: 35%
- Five **in-class** quizzes: 65% (13% per quiz)
  - Approximately every 2 weeks (see the schedule for exact dates)

Missed quizzes

- You can miss **up to 3** quizzes **for any reason**
  (no need to email or request EA)
- Weight of first 3 missed quizzes replaced with final exam mark
- Any additional missed quizzes get a mark of 0
- More than 2 missed quizzes may preclude a deferred final exam

Quizzes

- There will be **five quizzes**
- About a week before each quiz, I will post practice questions on the course website (with answers shortly afterward)
- The quizzes are **in-class** and **closed book**
- Types of questions:
  - Apply definitions and/or game rules
  - Compute quantities defined in class (e.g., minimax value)
  - Trace the execution of an algorithm discussed in class
  - Fill in missing code fragments
- Grades will be posted on Canvas

Academic Conduct

- Submitting someone else's work as your own is **plagiarism**.
- So is helping someone else to submit your work as their own.
- We report **all cases** of academic misconduct to the university.
- The university takes academic misconduct **very seriously**.
  Possible consequences:
  - Zero on the quiz or exam (virtually guaranteed)
  - Zero for the course
  - Permanent notation on transcript
  - Suspension or expulsion from the university

Prerequisites

- Formally: Any 200-level Computing Science course
- Comfort with **Python**
  - You don't need to be an expert programmer
  - But a lot of the course will involve exploring and understanding code
- Comfort with or interest in **formal mathematical/algorithmic reasoning**
  - We will reason about performance of algorithms, properties of data structures

Example code

- Much of the class will involve understanding example code
- This example code is available in a github repository
  - (forked from Ryan Hayward's original repository)
- Code is straightforward Python 3
- To clone the repository:

```
git clone https://github.com/jrwright/games-puzzles-algorithms.git
```

Course Topics

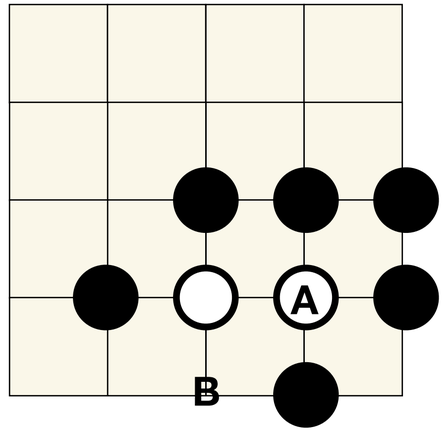
Typical topic structure:

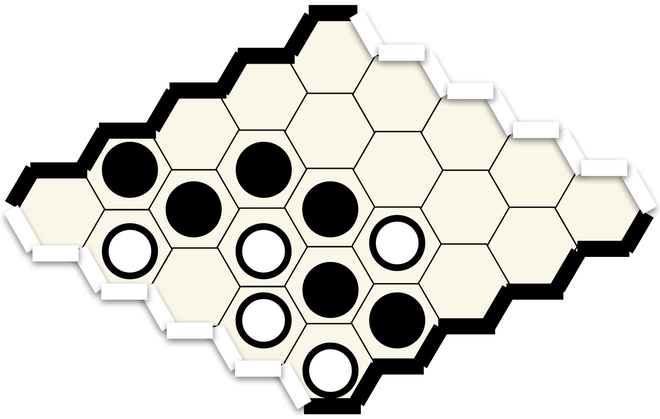
1. Example game(s) or puzzle(s)
2. Issue illustrated by the game, e.g.:
    - Positions vs states
    - Evaluating rules
    - Measuring position strength
3. Example code walkthrough, demo

1. Group-based rules

Example games: Go, Hex

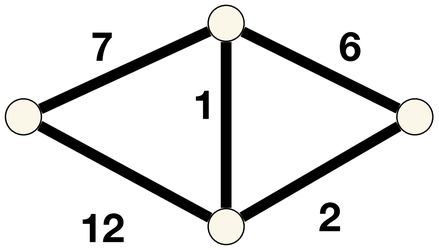
- Go is a popular territory-capturing game
  - Similar to game of life: stones with no "breathing room" die (are captured)
  - Different: breathing room is calculated for **contiguous groups**, not for individual stones
- Hex is a game where players try to connect their own two sides with a path of **contiguous** stones
- Surprisingly nontrivial to efficiently compute Go captures and Hex win conditions
  - Evaluate alternative algorithms and data structures





2. Brute-force and heuristic search

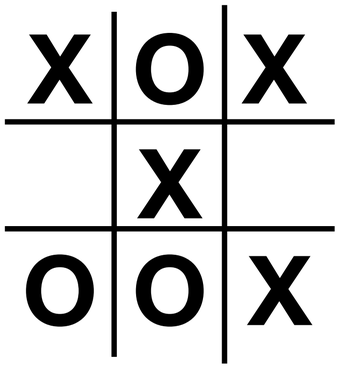
Example puzzles: Road maps and sliding tile puzzle



- Many problems representable as **states** that are modified by **actions**
  - E.g., locations in a road network; arrangements of tiles
- Can represent the problem as a **graph** of states
  - States s_1, s_2 **adjacent** if a single action in s_1 changes the state to s_2
- Can **solve** an instance of a puzzle by finding a path from start state to a solution state
  - E.g., path from start location to destination
  - E.g., sequence of moves from a scrambled start position to properly ordered
- **Heuristic** information can help search the graph more efficiently

1	2	4	
5	6	3	8
9	10	7	11
13	14	15	12

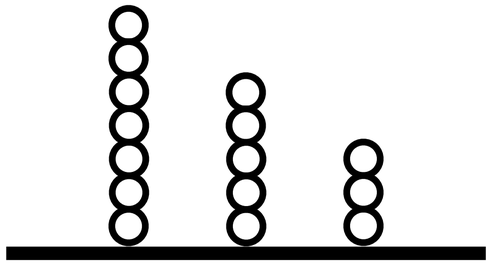
3. Minimax and alpha-beta search



Example game: Tic-tac-toe

- Games are fundamentally different from puzzles
  - Can't specify a full path, because opponent picks half the edges
- **Question:** How can search work in this setting? What is a **solution**?
- Solution if you can **force** a win no matter what the opponent does
- **Minimax search:** Inductive procedure to compute whether a position is a **win** (self can force a win), a **loss** (opponent can force a win), or neither
- **Alpha-beta search:** Procedure for pruning (i.e., not searching) subtrees that provably "don't matter" in a specific sense

4. Dynamic programming

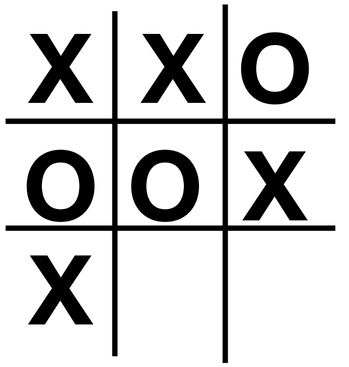


Example game: Nim

- Each turn: remove any number of stones from a single heap
- Player who removes the last stone wins

- Tic-tac-toe is solvable by brute force, because there are not that many paths to check
- Most games have exponentially many paths to evaluate
  - Exhaustive search is doomed, even with alpha-beta pruning
- *Idea:* **Cache** the results for previously-encountered positions (**memoization**)
- *Even better:* Evaluate positions in a carefully chosen **order** to maximize benefits of caching (**dynamic programming**)

5. Domain-specific pruning



Example games: Hex, tic-tac-toe, Nim

- Specific games have specific structure
- Sometimes can look at a position and easily tell if it's a win/loss/tie
  - E.g., this tic-tac-toe position
  - Can tell if a Nim position is a loss just by doing some arithmetic!
- This can be used for pruning
  - If a state is provably a win, just mark it as a win, no need to actually search to the end

6. Monte Carlo tree search

Example game: Hex

- A game is **solved** if some strategy is known to guarantee a win or draw for one player
  - i.e., must be able to prove that each move is a winning move no matter what the opponent does
- But a proof often requires searching the **whole state graph** (infeasible for Chess, Go, 11x11 Hex, etc.)
- **Question:** what can we do if we don't have time to search the whole tree?
- Monte Carlo tree search explores the state graph efficiently by using **simulations** ("roll-outs") to evaluate intermediate states
  - More promising subtrees are searched more intensively
- Doesn't necessarily **solve** the game, but often **plays** quite effectively

7. Noncooperative game theory

Example game: rock-paper-scissors

- Most of this course:
  - players choose actions sequentially
  - the state of the game is fully known by all players
  - one player's win is another player's loss
- Noncooperative game theory relaxes these assumptions
  - Players may have to move simultaneously (rock-paper-scissors)
  - Players may not know the current state of the game (Stratego, poker)
  - "Win-win" and "lose-lose" outcomes may exist (Prisoner's Dilemma)
- **Question:** What is even an optimal strategy for these games?

Summary

- **Course information:** https://jrwright.info/gpaclass/
- **Evaluation:** All in-person (in-class quizzes and a final exam)
- **Topics:** Algorithmic issues and techniques for solving games and puzzles
    1. Evaluating group-based position rules
    2. Search with and without heuristics for single-player puzzles
    3. Minimax and alpha-beta search for two-player perfect-information alternating-move games
    4. Dynamic programming for minimax search
    5. Domain-specific pruning for minimax search
    6. Monte Carlo tree search
    7. Noncooperative game theory for simultaneous-move games