Go: Formal Rules

CMPUT 355: Games, Puzzles, and Algorithms

Lecture Outline

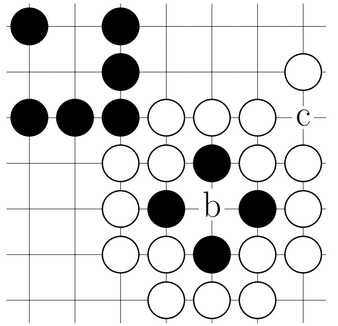
1. Recap
2. Demo! (leftover)
3. Tromp-Taylor rules of Go

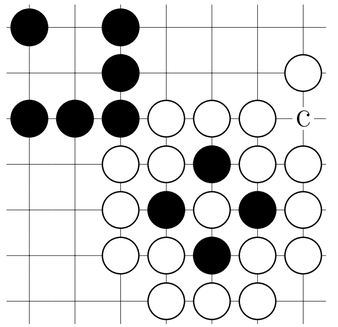
After this lecture, you should be able to:

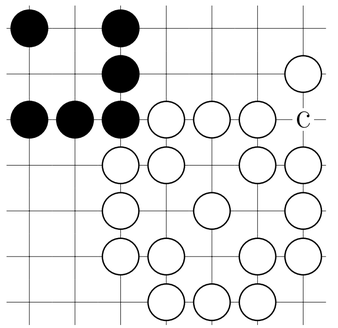
- Formally evaluate any move or position using Tromp-Taylor Rules

Recap: Rules of Go

- Board is a grid of lines; each intersection is a **point**
- Players move by placing a Black or White stone on empty intersections
- When a block loses its last **liberty**, it is **captured** and removed from the board
- A move that repeats a previous position is forbidden
  - This is called **superko**
- A move that causes the just-placed stone to be captured is **suicide**
  - We check for capture before we check for suicide
- Game ends when both players Pass instead of moving
- Scoring: Stones of own colour + "surrounded" territory + **komi** for White







Environments vs. Players vs. Solvers

- An **environment** is code that implements the rules of a game
  - E.g., **go/go_helper.py** in the sample code repository
- A **player** is code that chooses moves for playing the game
- A **solver** is a player that plays a game **optimally**
- In this class, we cover algorithms for all three types of program

Demo: Clone example code & try **go_helper.py**

1. Clone the repository:
   git clone https://github.com/jrwright/games-puzzles-algorithms.git
2. Run go_helper.py:
   cd games-puzzles-algorithms/go
   python3 go_helper.py
3. Try placing stones to demonstrate different rules:
   - capture
   - suicide detection
   - basic ko
   - superko
   - eyes

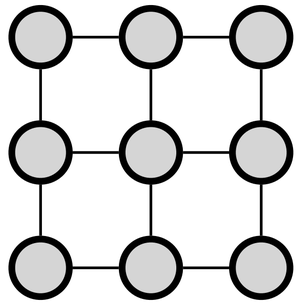
Tromp-Taylor Rules of Go

- Last lecture, we defined the rules of Go informally
- It's not always obvious how to interpret the rules in an algorithm
  - How to represent an empty point being "**surrounded**"?
  - How to formally define a **block** in a way that is straightforward to check?
- The **Tromp-Taylor Rules** formalize the rules from last lecture in the language of logic
  - This makes them easier to implement algorithmically

The Board

Rule 1: Go is played on a 19x19 square grid of points, by two players called Black and White.

- A rectangle of different dimensions may be used by prior agreement
- A "grid" is a graph in which each node has vertical and horizontal neighbours



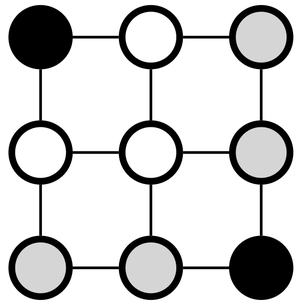
Rule 2: Each point on the grid may be **coloured** Black, White or Empty.

- Colouring a point Black or White means placing a black or white stone on the point
- Colouring a point Empty means removing the stone from it
- (In the example to the right, I represent "Empty" with grey)

Reachability

Rule 3: A point P, not coloured C, is said to **reach** C if there is a path of nodes of P's colour from P to a point of colour C.

- Connected stones of the same colour (sometimes called **strings**) all reach the same colours.
- Reaching empty means having empty points adjacent to the string, called **liberties**.



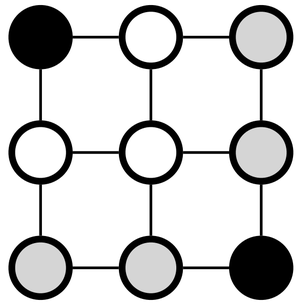
Examples:

- The Black stone in the upper left reaches White but not Empty
- The Black stone in the bottom right reaches Empty but not White
- Every stone in the White string reaches both Black and Empty

Clearing

Rule 4: Clearing a colour is the process of emptying all the points of that colour that don't reach Empty.

- Strings without liberties cannot exist at the end of a turn (because both Black and White will be cleared each turn)



Questions:

- What happens if we clear White in this example?
- What happens if we clear Black in this example?

Turns

Rule 5: Starting with an empty grid (i.e., all points coloured empty), the players alternate turns, starting with Black.

Rule 6: A **turn** is either a Pass, or a move that does not repeat an earlier grid colouring.

- That is, we forbid **positional superko**; the same colouring cannot be repeated even if it is for a different player-to-move.

Moves

Rule 7: A **move** consists of:

1. **Colouring** an empty point one's own colour, then
2. **Clearing** the opponent colour, then
3. **Clearing** one's own colour

- For any given move, at most one of the clearing processes can have effect (**why?**)
- If the first clearing process has an effect, then that is called **capture**
- If the second clearing process has an effect, then that is called **suicide**

End of Game

Rule 8: The game ends after two consecutive Passes.

Rule 9: A player's **score** is the number of points of her colour, plus the number of empty points that reach only her colour.

- This is called "area scoring"
- This is the rule that formalizes the notion of "territory" that is "surrounded" by either Black or White

Rule 10: The player with the higher score at the end of the game is the winner. Equal scores result in a tie.

- By prior agreement, a fixed amount called **komi** can be added to White's score.
- Using a non-integer such as 5.5 avoids ties

Summary

- This class is about efficiently implementing environments, players, and solvers
- The file **go/go_helper.py** in the example repository implements a Go environment
- Tromp-Taylor rules formally define the rules of Go
  - These are the rules that we will use in this class