CSCI 2270

Data Structures & Algorithms

Gabe Johnson

Lecture 39 April 26, 2013

AI Techniques (for the Game Project)

Upcoming Homework Assignment

HW #11 **Due: Friday, Apr 26**

Game

The Game assignment is due tonight. All group members should email it to Gabe's Gmail account with the correct subject line (in order to prevent it from getting lost):

To: **gabe.johnson@gmail.com**
Subject: **CS 2270 Project**

See the HW 11 README.md for more info.

Lecture Goals

1. Finish the AI overview PDF
2. Blackjack AI (imperfect info)
3. Chess AI (perfect info)
4. Rock-Scissor-Paper
5. Snake

Blackjack

- ★ **n** Standard decks of cards (52 per deck)
- ★ Dealt one card face down
- ★ Dealt one card face up
- ★ Optionally take more cards from the deck
- ★ Score over 21 = loss
- ★ Score = 21 = win
- ★ Score > anybody else's score = win
- ★ In case of tie, dealer wins
- ★ Players go in order

Imperfect Information

Blackjack is a good example of ‘imperfect information’. An AI (or person) who does not cheat has no way of knowing the composition of the deck.

You can *count cards* by recording what has been played already, and can no longer be used.

If we’ve seen two aces so far, we know there are only two left. If we’ve seen 10 cards so far, the odds that the next card is an ace is 2 / 42 = 5%.

AI for Blackjack

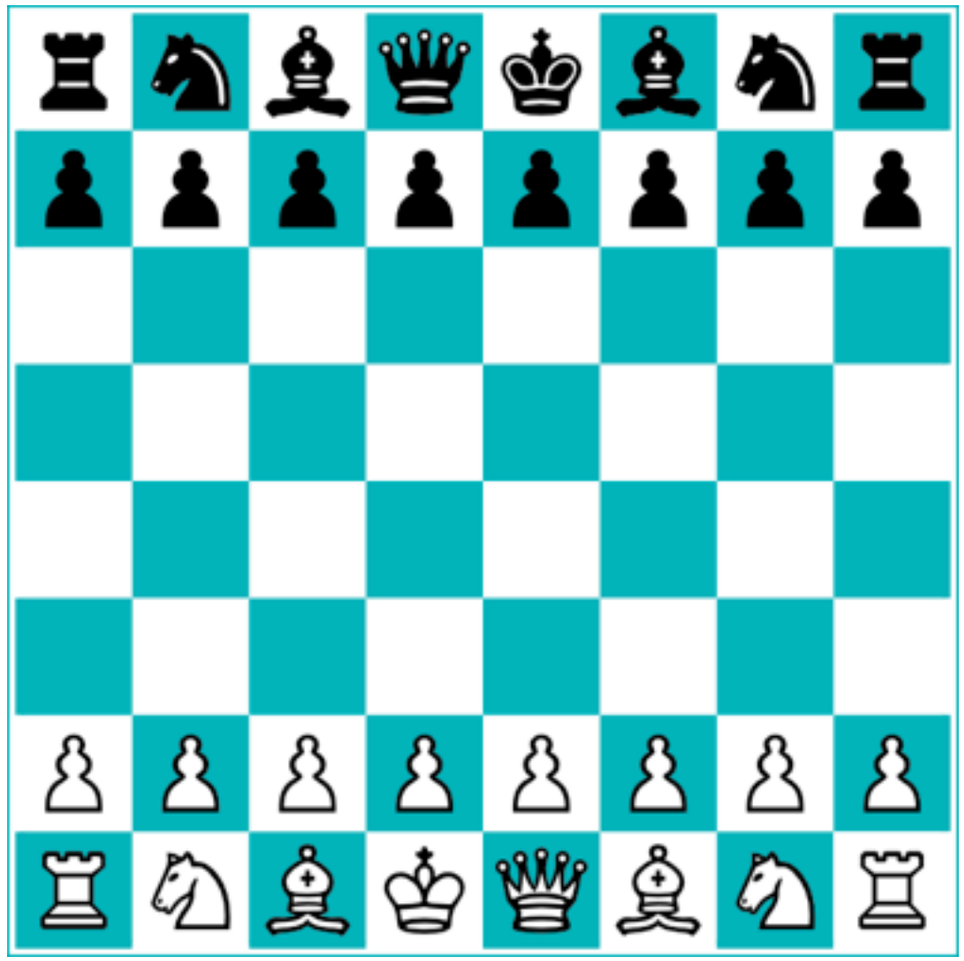
Several AI strategies:

- count cards like a human
  - > compute probabilities perfectly, or:
  - > ‘forget’ on occasion to make it less effective
- don’t count cards, ‘hit me’ if sum < 16
- cheat and look at the deck’s data structure
- flip a coin to decide to hit or stay

Chess

AI for chess has a long history and is often considered the benchmark for progress in the field.

- ★ 8x8 grid
- ★ Known initial state
- ★ Two players
- ★ Six piece types
- ★ Movement rules per piece
- ★ Players take turns
- ★ All information known
- ★ 50-move limit (*really*)



Perfect Information

There is **no secret information or randomness**.

Each player has full access to the same information, which is entirely encoded in the state of the chess board. Games progress according to the sequence of decisions each player makes.

AI for Chess

A chess AI could theoretically perform an exhaustive search of the board to determine if there is an unbeatable strategy (e.g. checkmate in 2 moves, other player can’t avoid it).

AI for Chess

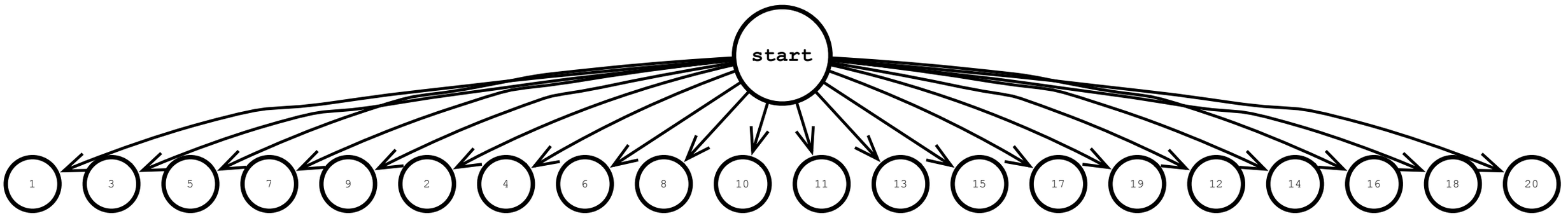
Exhaustive search is rather cumbersome, *but*, if the algorithm records the transitions, performing a lookup of that search is an O(1) operations.

Trouble is: this requires a MONDO memory. One estimate of the number of moves is:

$$10^{10^{50}}$$

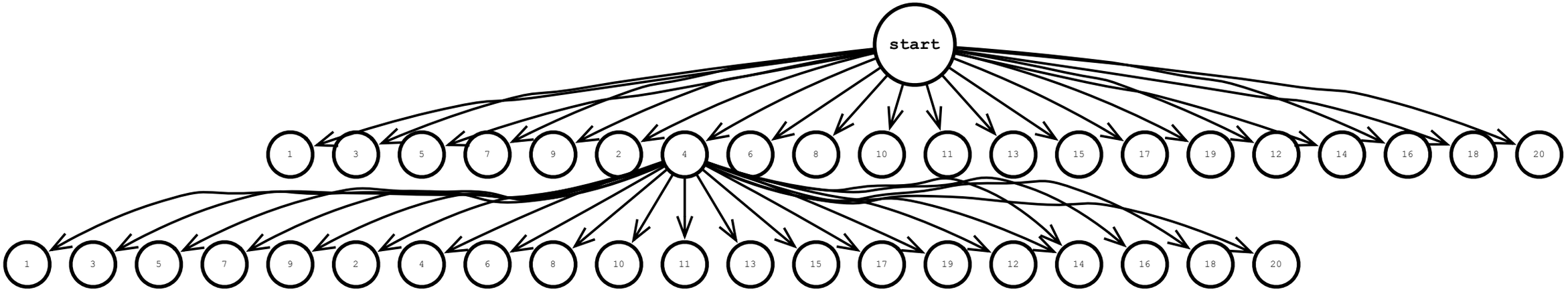
Try computing that number to get a sense of how insanely large this search space is.

AI for Chess



We can ‘search’ through possible moves if we represent actions as graphs. The graph above represents the very first move—it starts in a known state, but can transition into one of 20 possible states (16 pawn moves and 4 rook moves).

AI for Chess



After each move, the board is in a different state. These can be modeled as nodes, and the transitions between nodes are possible legal moves.

Here's part of the search space for two moves. Imagine there are 20 nodes hanging off each from the first layer. That is a lot of moves!

AI for Chess

Two observations:

1) The search space is too big for any computer to handle—there are 10^{80} particles in the universe and $10^{(10^{\wedge}50)}$ possible chess games.

2) Some moves are substantially more likely than others because they are more likely to lead to victory (or defeat). So we can incorporate probability and search the most likely items first.

Rock Paper Scissors

In this game we have two players who reveal their decisions at the same time.

★ Paper beats Rock
★ Rock beats Scissors
★ Scissors beats Paper
★ Like types result in a tie

What do you think: imperfect or perfect information?

Rock Paper Scissors

You can examine your opponent's strategy. They might be completely random, so it is impossible to determine what they will do.

But humans are bad at being completely random: there will be some bias in their choices. You can record their possible choices and form a probability distribution of what they do.

Rock Paper Scissors

Opponent moves:

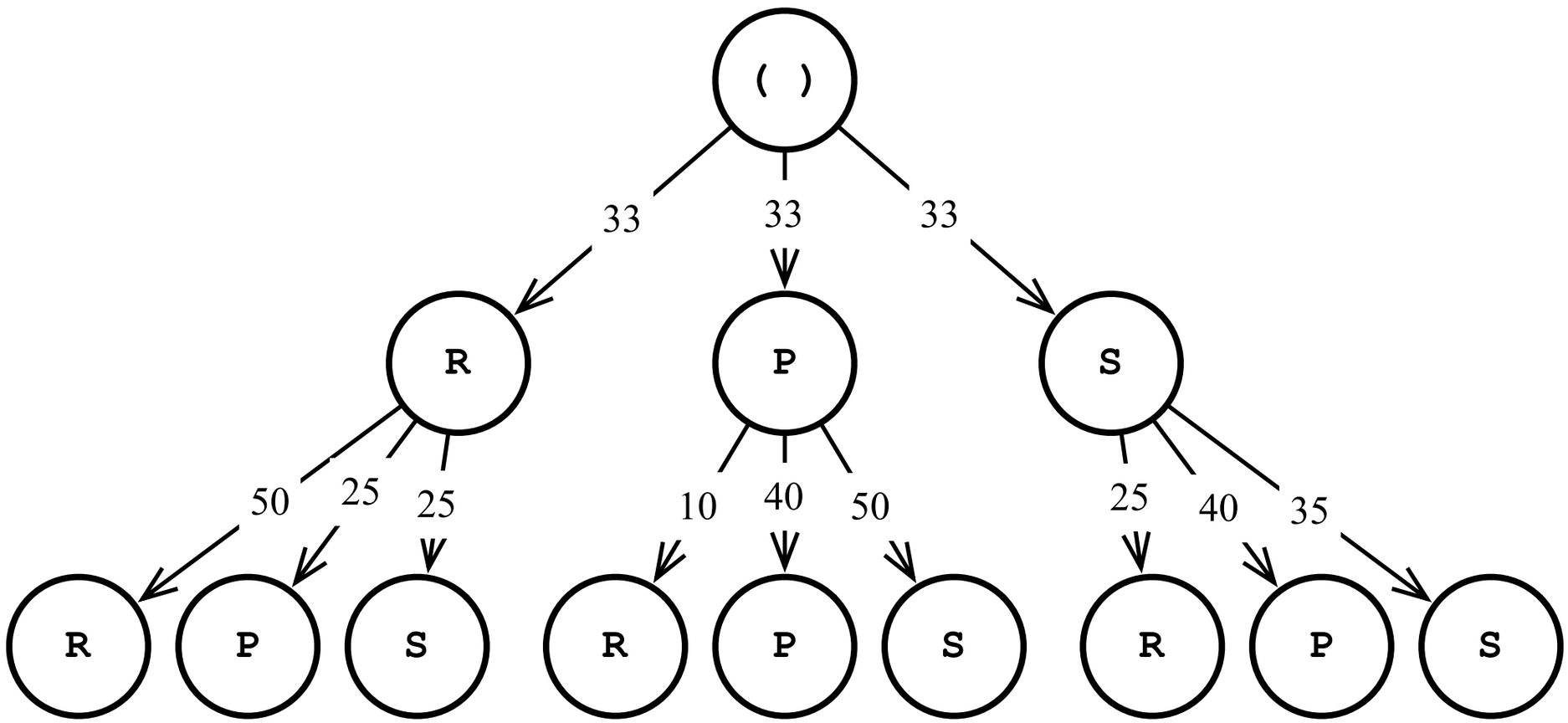
- Paper 50%
- Rock 30%
- Scissors 20%

What's a reasonable move against this player?

What if they pay attention to how often *you* play?

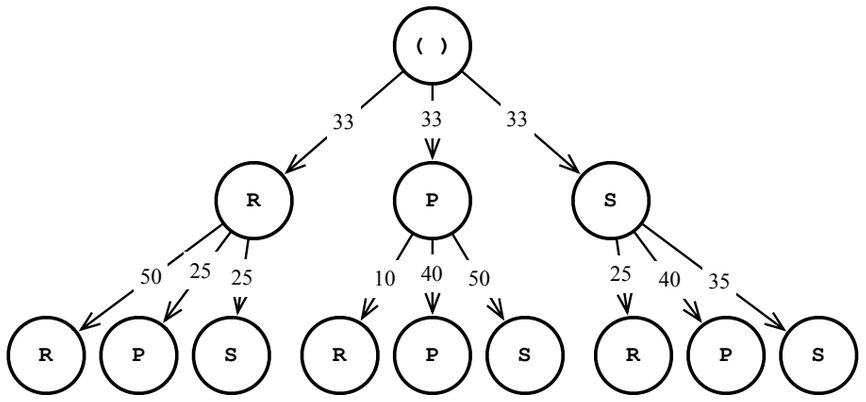
Rock Paper Scissors

Perhaps your opponent's individual moves are random, but the sequences they make them between several moves is not. Maybe:



Rock Paper Scissors

So if we've detected a pattern in the opponent's strategy, we can model it as a *decision tree*. Nodes represent moves, numbers represent the probability the player will make that move.



So here the thing we look at is the *sequence* of moves that have non-uniform probability distribution. And we can, of course, use this to crush our enemies.

Rock Paper Scissors

This is a good example of how to *learn* some behavior: here we might construct a decision tree as we play against a certain opponent. After every move we:

1. Possibly modify the structure of the tree (e.g. add nodes indicating a new sequence).
2. Update the probability distribution.

Rock Paper Scissors

Problem with the decision tree:

There's no end state. Our tree would grow indefinitely large unless we chose some (arbitrary) maximum tree height.

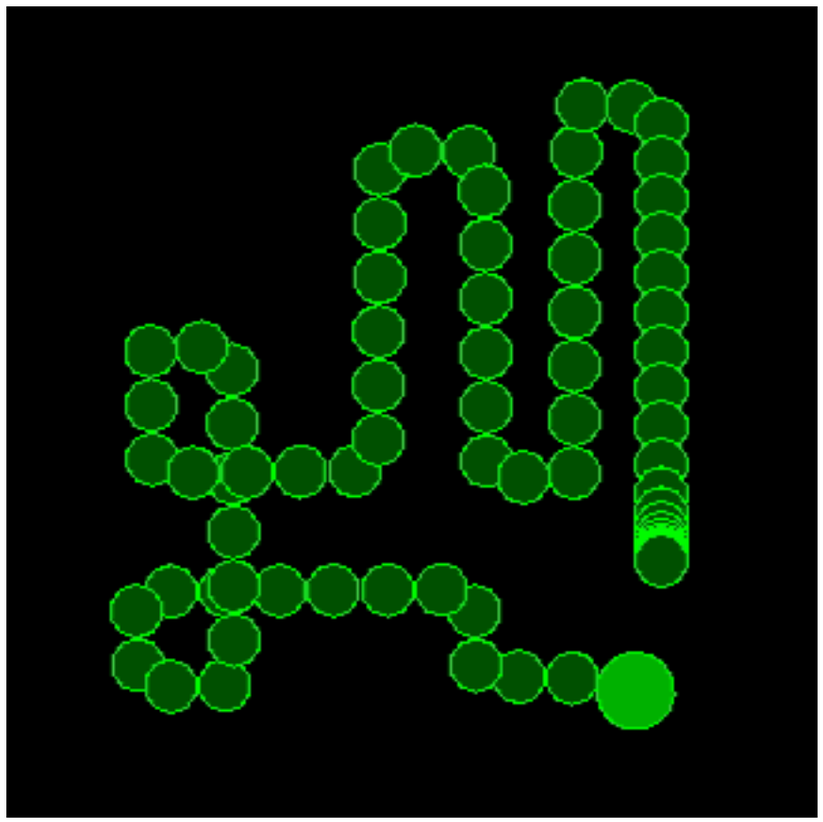
Maybe a better data structure would be a dynamically constructed finite state machine?

Rock Paper Scissors

Or, maybe you build several trees and FSMs using different rules (like max tree height), and see which one has the highest prediction accuracy.

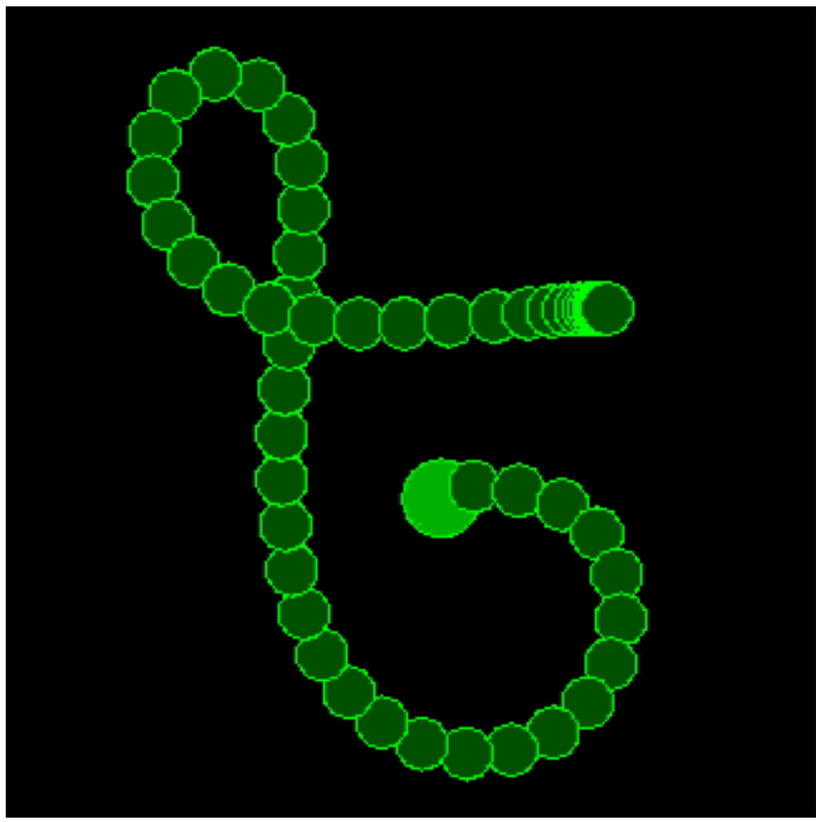
(Building *forests* of possible decision trees that are built using different assumptions and features is common. This applies to other learning data structures as well.)

Snake



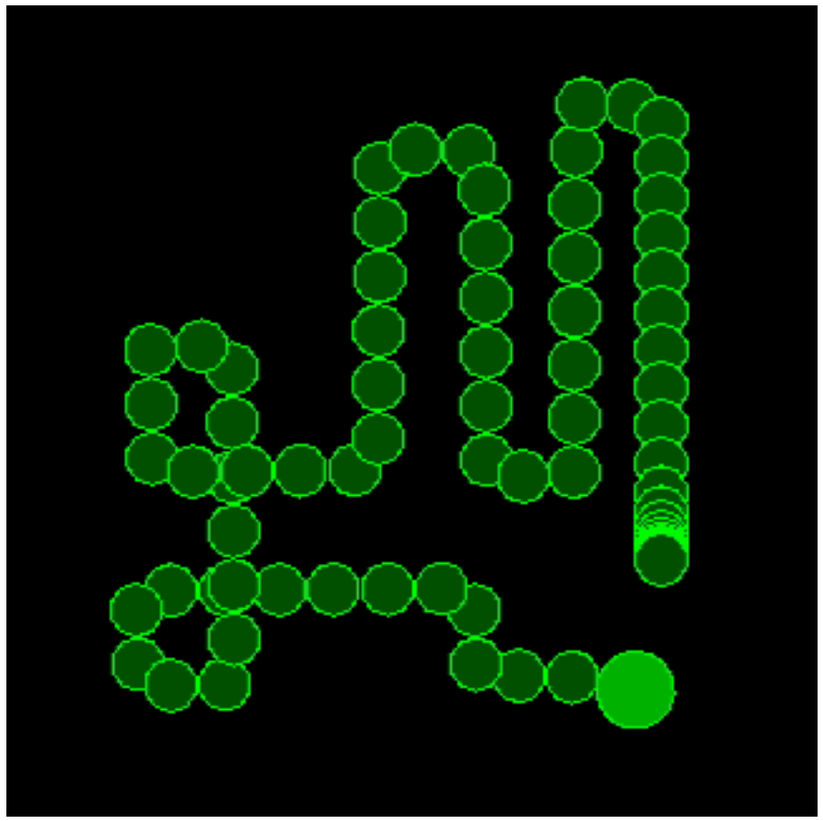
Retro Motion: snake body follows the head's exact path.

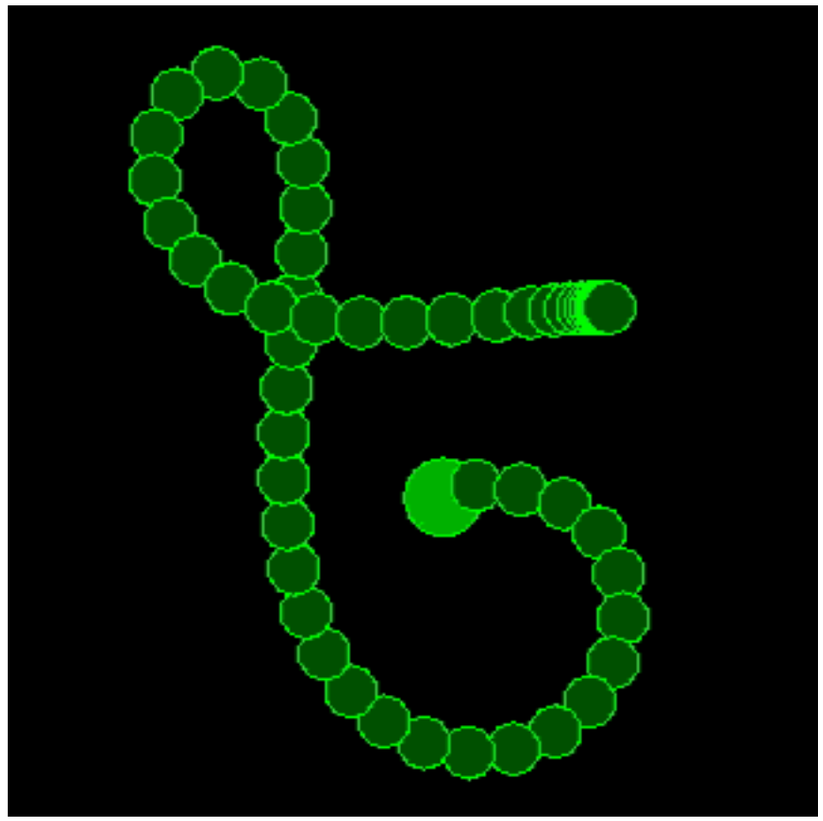
Snake



Smooth Motion: segments follow one just in front of it.

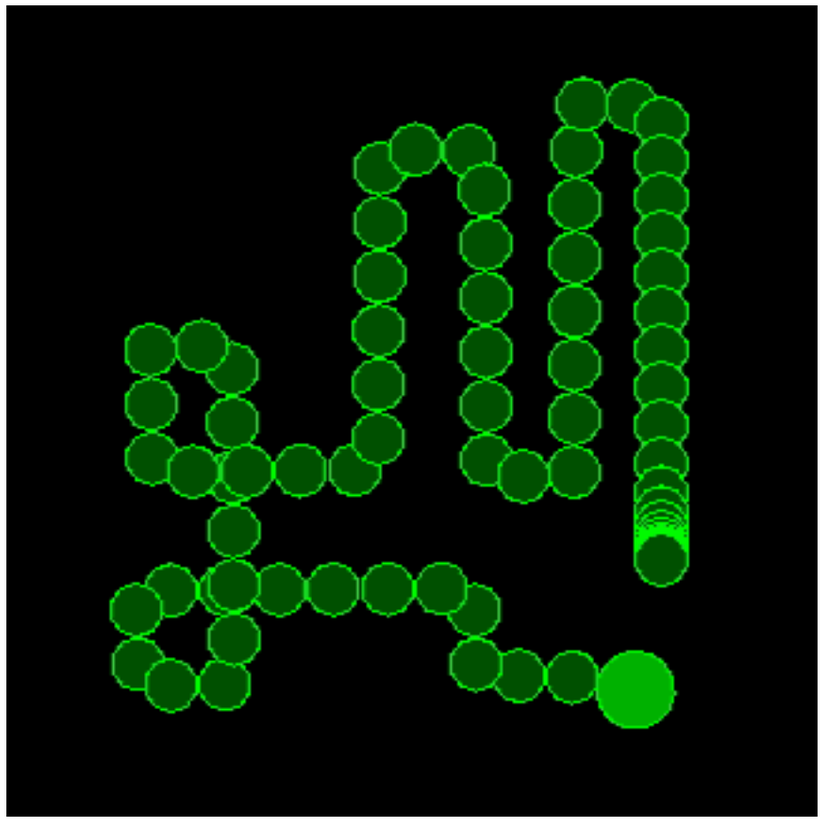
Snake

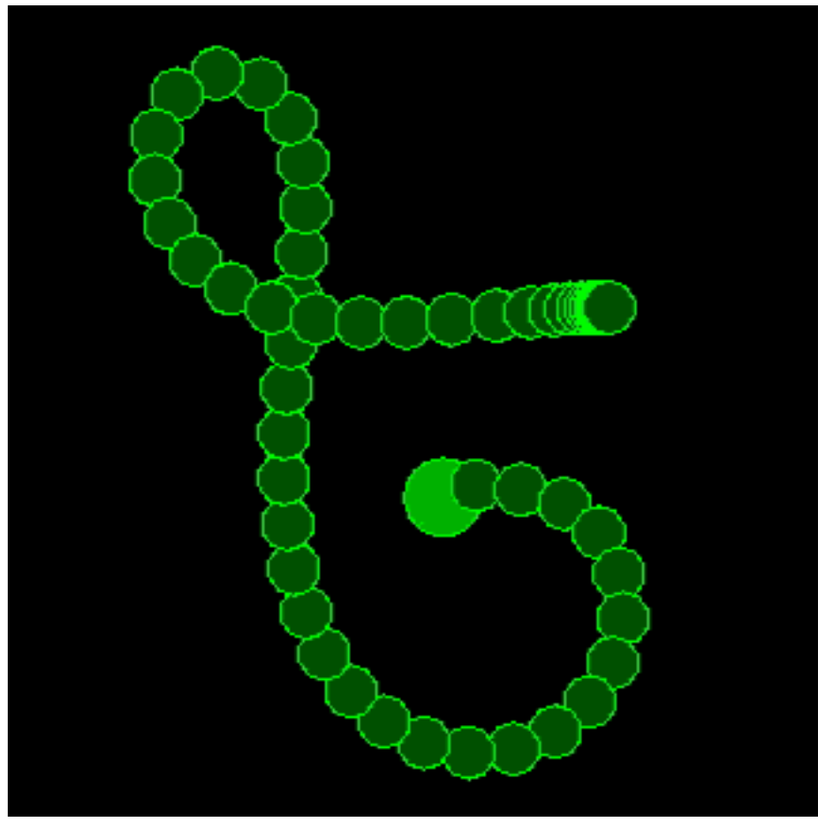




The game is to go as long as you can while having the snake eat dots without letting it intersect itself.

Snake





An AI to solve this would have to calculate the snake's body state into the future while also considering the path the snake has to take to get to the food.

Other Games?

If you're fuzzy on how you might write an AI for your game, we can talk about it now in class.

Otherwise: Monday and Wednesday are review for the final. I'll cover everything that will be on the test.

About 40—50% of the test will be on pointers and recursion. *Just so you know*.

#winning

Next Friday: Special lecture: How to win at Software.

Not required, but this will be advice from someone who has been there (me). How to hack, how to engineer, how to debug, how to design, how to research. And (importantly) how *not* to do these things.