1. Concurrency
2. AI
Concurrency is hard...
Recap

- Process: program in execution. Unit of sequential execution.

- We can structure programs so that they can be executed as a set of concurrent processes
  - On a single processor
  - On multiple processors

- Processes may coordinate their actions using
  - Shared memory
  - Message passing

- A race condition is a situation in which multiple processes read and write a shared data item and the final result depends on the order of execution.
There are many ways to execute two processes concurrently.

The green process executes steps S1 S2 S3 in the given order.
The blue process executes steps S1 S2 S3 in the given order.

Several possible interleavings of steps.
Assumption

In the rest of the lecture we will use some programs to illustrate concepts such as race conditions, interference, and deadlock. For the purposes of this lecture we assume that a single line of program is executed atomically: you can think of one line of code as corresponding to one step in the previous slide whose execution cannot be broken down into smaller steps.
Critical Sections

- Often, a process really needs exclusive access to some data.

- A critical section is a sequence of steps that have exclusive access to the shared resource
  - If multiple processes are sharing a resource only one should be executing its critical region

- Real Life Examples where critical sections are needed
  - Crossing a traffic intersection
  - A bank with many ATMs
Consider a bank with multiple ATM’s.

At one, Mr. J requests a withdrawal of $10.

At another, Ms. J requests a withdrawal of $10 from the same account.

The bank’s computer executes:

1. For Mr. J, verify that the balance is big enough.
2. For Ms. J, verify that the balance is big enough.
3. Subtract 10 from the balance for Mr. J.
4. Subtract 10 from the balance for Ms. J.

The balance went negative if it was less than $20!
Vocabulary Reminder

- **Race condition**: A behavior in concurrent processing where proper functioning depends on the timing of other uncontrollable events.

- **A critical section** is a piece of code that accesses a shared resource that must **not** be concurrently accessed by more than one process.
What can we do to prevent one processor from entering the critical section while another is in it?

Locate the J’s account data containing the balance

if balance < 10:
    error
else:
    balance = balance - 10

Dispense $10 from ATM

Process executed for each concurrent request
Careful Driver Method: Don’t enter the intersection unless it’s empty.

In shared memory: \( \text{free} = \text{True} \) \# initially unlocked

# Process 1
while True :
    Non-Critical_Section
    while not free:
        pass
    free = False
Critical_Section
free = True

# Process 2
while True :
    Non-Critical_Section
    while not free:
        pass
    free = False
Critical_Section
free = True

Interference is possible!
Careful Driver Method: Don’t enter the intersection unless it’s empty.

In shared memory: \( \text{free} = \text{True} \) #initially unlocked

# Process 1
while True :
    Non-Critical_Section
    while not free :
        pass
    free = False
    Critical_Section
    free = True

# Process 2
while True :
    Non-Critical_Section
    while not free :
        pass
    free = False
    Critical_Section
    free = True

If these two processes leave their non-critical sections at precisely the same time, then strictly alternate lines, they will both end up in the Critical_Section.
Computers vs. Real Life

- The careful driver method works in real life because
  - The number of times in your life you cross the intersection is low. Twice a day for forty years is about 29,000.
  - The chance of two drivers arriving at the intersection simultaneously is low.
  - Cars move slowly enough that if you don’t see anyone coming, you’ll get across before anyone comes.
1. Signal your intention (by stopping).
2. Wait until cross road has no one waiting or crossing.
3. Cross intersection.
4. Renounce intention (by leaving intersection).
The Stop and Look Method

# Shared Memory
free0 = True  # P0 is not stopped at sign
free1 = True  # P1 is not stopped at sign

# Process 0
while True :
    Non-Critical_SECTION
    free0 = False
    while not free1 :
        pass
    Critical_SECTION
    free0 = True

# Process 1
while True :
    Non-Critical_SECTION
    free1 = False
    while not free0 :
        pass
    Critical_SECTION
    free1 = True

This version of the code does not suffer from interference. Does it now guarantee safe execution of the critical section?
The Stop and Look Method

# Shared Memory
free0 = True  # P0 is not stopped at sign
free1 = True  # P1 is not stopped at sign

# Process 0
while True :
    Non-Critical_Section
    free0 = False
    while not free1:
        pass
    Critical_Section
    free0 = True

# Process 1
while True :
    Non-Critical_Section
    free1 = False
    while not free0:
        pass
    Critical_Section
    free1 = True

Once again, if the two processes exit the non-critical section at the same time and strictly alternate lines they will end up stuck in their while loops. This is called Deadlock.
Deadlock

Deadlock is the condition when two or more processes are all waiting for some shared resource, but no process actually has it to release, so all processes to wait forever without proceeding.

It’s like gridlock in real traffic.
The Stop Sign Method with Tie Breaking

1. Signal your intention (by stopping).
2. Wait until cross road has no one else waiting or crossing.
3. If two of you are both waiting, yield to the car to your right.
4. Cross intersection.
5. Renounce intention (by leaving intersection).
Peterson’s algorithm avoids all bugs!

```python
free0 = True
free1 = True
priority = 0

# Process 0
while True:
    Non-Critical_Section0
    free0 = False
    priority = 1
    while not free1 and priority==1:
        pass
    Critical_Section0
    free0 = True

# Process 1
while True:
    Non-Critical_Section1
    free1 = False
    priority = 0
    while not free0 and priority==0:
        pass
    Critical_Section1
    free1 = True
```
Peterson’s algorithm avoids all bugs!

```python
free0 = True
free1 = True
priority = 0

# Process 0
while True:
    Non-Critical_Section0
    free0 = False
    priority = 1
    while not free1 and priority==1:
        pass
    Critical_Section0
    free0 = True

# Process 1
while True:
    Non-Critical_Section1
    free1 = False
    priority = 0
    while not free0 and priority==0:
        pass
    Critical_Section1
    free1 = True
```

Entrance to the critical section is granted for process P0 if P1 does not want to enter its critical section (free1 == True) or if P1 has given priority to P0 by setting priority to 0 (priority == 0).
A Probabilistic Approach

There is a conceptually easier way to solve synchronization problem by embracing probable thinking. We just use the stop sign approach but wait for a random amount of time when a conflict occurs.
Types of HeisenBugs*

In decreasing order of seriousness:

1. Interference: multiple process in critical section.
2. Deadlock: two processes idle forever, neither entering their critical or non-critical sections.
3. Starvation: one process needlessly idles forever while the other stays in its non-critical section.
4. Unfairness: a process has lower priority for no reason.

Note: We did not discuss 3 and 4 in detail. You can learn more about them in the future.

* In computer programming jargon, a heisenbug is a software bug that seems to disappear or alter its behavior when one attempts to study it. Source: Wikipedia
Dining Philosophers’ Problem

Aristotle

Socrates

Homer

Plato
The Dining Philosophers

- Each philosopher thinks for a while, then picks up his left fork, then picks up his right fork, then eats, then puts down his left fork, then puts down his right fork, thinks for a while...
  - We assume here that each philosopher thinks and eats for random times, and a philosopher cannot be interrupted while he picks up or puts down a single fork.

- Each fork models a "resource" on a computer controlled by an OS.

- Original problem was proposed by Edsgar Dijkstra.
Dining Philosophers’ Problem

- There are N philosophers.
- Philosopher i does the following:
  1. THINK
  2. Pick up fork i.
  3. Pick up fork \((i+1) \mod N\).
  4. EAT
  5. Put down fork i.
  6. Put down fork \((i+1) \mod N\).
  7. Go to step 1.

NOTE: \((i+1) \mod N = i+1\), if \(0 \leq i < N-1\)
\((i+1) \mod N = 0\), if \(i = N-1\)
Dining Philosophers’ Problem

• There are N philosophers.

• Philosopher i does the following:
  1. THINK
  2. Pick up fork i.
  3. Pick up fork (i+1) modulo N.
  4. EAT
  5. Put down fork i.
  6. Put down fork (i+1) modulo N.
  7. Go to step 1.

How can deadlock occur here?

Fork 0 Fork 1 Fork 2 Fork 3

N=4
Removing the Deadlock

Philosopher i does the following:

1. THINK
2. If i is not equal to N-1:
   a. Pick up fork i
   b. Pick up fork i + 1
3. If i is equal to N-1:
   a. Pick up fork 0
   b. Pick up fork N - 1
4. EAT
5. If i is not equal to N-1:
   a. Put down fork i
   b. Put down fork i + 1
6. If i is equal to N-1:
   a. Put down fork 0
   b. Put down fork N - 1
7. Go to step 1

This philosopher picks up the right fork first

This philosopher picks up the right fork first
Afterthoughts

Some counter-intuitive ideas about bugs and risks.
This man removed all the traffic lights and signs!

It reduced accidents by 50% and reduced congestion. Pedestrian and driver anxiety increased.
Why did Jared Diamond sleep under a tree when his aborigine companion wouldn’t?

There is only a $1/10,000$ chance of a particular tree falling on any given night. But aborigines sleep in the woods every night so if they tempted fate every night for 40 years there is only a 23% chance of not dying that way, i.e. $(1-1/10,000)^{(40*365)} = 0.23$. 
Why is a 1% chance of a bug biting better than a 0.1% chance?

- If there is a 1% chance of error, the bug will show up during 100 days of testing.

- If there is a 0.1% chance, the bug will show up after three years when the system is deployed.
Overview

- What is AI?
- Games and state-space search
- When can we say a program is intelligent?
Artificial Intelligence

- Branch of computer science that studies the use of computers to perform computational processes normally associated with human intellect and skill.

- Some areas of AI:
  - Game playing
  - Knowledge representation
  - Robotics
  - Machine learning
  - Natural language processing
  - Music, Speech & Vision

Turing Award recipients in late ’60s—early ’70s for contributions to AI:

- Newell
  - CMU
- Simon
  - CMU
- Minsky
  - MIT
- McCarthy
  - Stanford
A Cynic’s View

- AI is the study of how to get computers to do things we don’t understand.
  - “thinking”, “learning”, “creativity”, etc.

- When we do understand something, it’s no longer AI: it’s just programming.
  - *Examples:*
    - speech recognition,
    - computer chess and checkers,
    - robotics, ...
Games and Search

A laboratory for artificial intelligence
Why Study Games?

- Thin end of the wedge for AI research
  - Characteristically human activity
  - Small so potentially solvable
  - Easy to measure success or failure
  - Solutions *might* tell us something about intelligence in general
  - But are we just “looking under the lamp post”?
Why Study Games?

Arthur Samuels, 1960:

Programming computers to play games is but one stage in the development of an understanding of the methods which must be employed for the *machine simulation of intellectual behavior*. As we progress in this understanding it seems reasonable to assume that these newer techniques will be applied to real-life situations with increasing frequency, and the effort devoted to games . . . will decrease. Perhaps we have not yet reached this turning point, and we may still have much to learn from the study of games. [Emphasis ours]
Two-player games
Game Properties

- Two players
  - alternating turns

- Perfect information
  - No hidden cards or hidden Chess pieces...

- Finite
  - Game must end in finite number of moves

- Deterministic
  - no randomness, e.g., dice

- Zero sum
  - Total winnings of all players
Imagine a two-player game with two possible moves at each point. A goes first:
Game of Nim

- Two players
- Some piles of stones
- At each turn, player can remove any number of stones from any single pile
- If it is your turn and there are no stones left, you lose.
Playing Nim
P1 has first move
Evaluation Functions

- We can define the **value** (goodness) of a certain game state (board).

  - For example,
    1 = Player A wins,
    -1 = Player B wins
    (0 = tie, when ties are possible)
How do we play?

- Traverse the “game tree”.
  - Enumerate all possible moves at each node.

- Start by evaluating the terminal positions, where the game is over.

- Propagate the values up the tree, assuming we pick the best move for us and our opponent picks the best move for her.
Playing Nim with Minimax

Evaluation function:
1 if P1 wins
-1 if P2 wins
Playing Nim with Minimax

Evaluation function: 1 if P1 wins, -1 if P2 wins
Playing Nim with Minimax

Evaluation function: 1 if P1 wins (max), -1 if P2 wins
Playing Nim with Minimax

Evaluation function:
1 if P1 wins (max)
-1 if P2 wins (min)
Playing Nim with Minimax

Evaluation function:
1 if P1 wins (max)
-1 if P2 wins (min)
How do we play?

- What is the rule to compute the value of nonterminal node?
  - If it is our turn (P1) then we pick the maximum outcome from the children.
  - If it is the opponent’s turn (P2) then we pick the minimum outcome from the children.

- This process is known as the Minimax algorithm.

- We play the move that gives us the maximum at the root.
A pencil and paper game for two players, X and O, who take turns marking the spaces in a $3 \times 3$ grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

The following example game is won by the first player, X:

Tic Tac Toe Game Tree

Diagram of Tic Tac Toe game tree with various states and moves.
For the Enthusiast

- If you are interested in optimal strategies see the next 9 slides.
XKCD’s Optimal Tic-Tac-Toe

MAP FOR X:

COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

http://xkcd.com/832
MAP FOR 0:
X Move 1
O Move 1 (non-optimal play)
X Move 2
O Move 2
X Move 3
O Move 3
X Move 4 To Win
How Big is the Tic-Tac-Toe Tree?

- Assuming that all nine positions must be filled before the game ends, how big does this tree get?
How Big is the Tic-Tac-Toe Tree?

- Assuming that all nine positions must be filled before the game ends, how big does this tree get?

At most $9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 9! = 362,880$
How Big is the Tic-Tac-Toe Tree?

- Assuming that all nine positions must be filled before the game ends, how big does this tree get?

  At most $9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 9! = 362,880$

- Of course, in real Tic-Tac-Toe, a player can win without filling the whole board.
  - What is the first level of the tree where this can occur?
How Big is the Tic-Tac-Toe Tree?

- Assuming that all nine positions must be filled before the game ends, how big does this tree get?
  
  At most $9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 9! = 362,880$

- Of course, in real Tic-Tac-Toe, a player can win without filling the whole board.
  - What is the first level of the tree where this can occur?
    Ending on the 5th move. 3Xs, 2 Os
How Big is the Tic-Tac-Toe Tree?

- Assuming that all nine positions must be filled before the game ends, how big does this tree get?

  At most $9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 9! = 362,880$

- Of course, in real Tic-Tac-Toe, a player can win without filling the whole board.
  - What is the first level of the tree where this can occur?
    Ending on the 5th move. 3Xs, 2 Os
  - How big is this tree up to this level?
How Big is the Tic-Tac-Toe Tree?

- Assuming that all nine positions must be filled before the game ends, how big does this tree get?
  
  At most \(9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 9! = 362,880\)

- Of course, in real Tic-Tac-Toe, a player can win without filling the whole board.
  
  What is the first level of the tree where this can occur? Ending on the 5\(^{\text{th}}\) move. 3Xs, 2 Os

- How big is this tree up to this level?
  
  \(9 \times 8 \times 7 \times 6 \times 5 = 15,120\)
Tic-Tac-Toe Is Completely Searchable

- Using recursion, can grow the entire game tree with the current board state as the root

- fewer than 363,000 nodes, feasible

- Furthermore, if board symmetries are taken into account, two games are considered the same if rotating and/or reflecting the board makes one of the games into a copy of the other game.
Some Other *Solved* Games

- Connect Four
- Awari (Oware)
- Checkers (!) which took decades (see Chinook)
A complete analysis of a chess game is *computationally infeasible*.

(What other problem have we seen lately that is computationally infeasible? Hint: in that case, infeasibility was a *desirable feature*, not a problem.)

But human beings play chess, some very well!

*Idea:* computer techniques to deal with combinatorial explosion may reveal something important about intelligence.
For most games, the number of possible moves and potential outcomes is HUGE.

Example:
Chess: If there approximately 20 possible moves at each turn, looking ahead 15 moves requires examining about $3.3 \times 10^{19}$ sequences, which would take years.
Managing combinatorial explosion

- An AI technique used to manage the explosion is the use of a game tree.
  - A tree is built with a root node representing the current state of the game.
  - Child nodes are generated representing the state of the game for each possible move.
  - The tree is propagated down, building more child nodes for moves allowed by the next move, etc.
  - Leaves are terminal (win/lose/draw) states of the game.
  - Unlike the trees you’ve studied before, game trees are not usually computed in their entirety (they are too large!)
Search and heuristics

Coping with combinatorial explosion
Game Tree as Search Space

- We say we *search the game tree* as we try to compute a winning move
  - In reality AI programs generate a small part of the tree only!

- *Search space* or *solution space*—a powerful general idea in life as in computing
Heuristics

- Human thought is not purely deductive. It often relies on approximations or analogies.

- A heuristic is a “rule of thumb” that may not always be correct but usually gives useful results.

- Heuristic algorithms are helpful because they can find a reasonably good solution to a problem without requiring excessive search.
Dealing With **Huge** Game Trees

- How does a computer program that plays Chess or Go deal with the huge size of the game trees that can be generated?
  - **Chess**: size of game tree: \(~35^{100}\) (about \(10^{150}\)), beyond hope, even for fastest computers!
  - **Go**: much bigger (one estimate is \(10^{360}\))

- These programs use **heuristics** to reduce the part of the tree that must be examined.
Estimator functions

- Save time by **guessing** search outcomes
- **Chess**: Estimate the quality of a situation
  - Count pieces
  - Count pieces, using a weighting scheme
  - Count pieces, using a weighting scheme, considering threats
  - ...

Minimax, in reality

- Rarely do we reach an actual leaf
  - Use **estimator functions** to statically guess outcome of unevaluated subtrees (value of current position)

- **Max**
- **Min**
- **Max**

```
A moves
B moves
A moves
```
Minimax, bounded search

- Rarely do we reach an actual leaf
  - Use **estimator functions** to statically guess outcome of unevaluated subtrees (value of current position)

- Max
- Min
- Max

Note that minimax works with any real numbers, not just -1, 0, 1.
Minimax, in reality

- Rarely do we reach an actual leaf
  - Use **estimator functions** to statically guess outcome of unevaluated subtrees (value of current position)

```
Max
Min
Max
A moves
B moves
A moves
```
Minimax, bounded search

- Trade-off: Quality vs. Speed
  - Quality: deeper search
  - Speed: use of estimator functions

- Balancing
  - relative costs of move generation and estimator functions,
  - quality and cost of estimation function
Search and problem solving

not just for games!
State Space Search

- Searching game trees is a form of *state space search*

- "Good old-fashioned AI": idea that *any* intelligent behavior can be modeled as search in a state space tree

- Successes:
  - Chess, checkers, etc.
  - Route-finding for airline travel planning
  - Layout for integrated circuits in electronics
  - Automated manufacturing processes
  - Symbolic integration
  - Proving theorems

- Heuristics values:
  - The value of each piece. (1 for pawn up to 9 for queen)
  - The amount of control each side has over the board.
  - The safety of the king.
  - The quickness that pieces move into fighting position.

For more info:

Is Deep Blue intelligent?
“Chinook”

- Created by computer scientists from the University of Alberta to play checkers (draughts) in 1989.

- In 2007, the team led by Jonathan Schaeffer announced that Chinook could never lose a game. **Checkers had been solved**:

  The best a player playing against Chinook can achieve is a draw.

- Chinook's algorithms featured:
  - a library of opening moves from games played by grandmasters
  - a deep search algorithm
  - a good move evaluation function (based on piece count, kings count, trapped kings, player’s turn, “runaway checkers”, etc.)
  - an end-game database for all positions with eight pieces or fewer. and other minor factors.

- Is Chinook intelligent?
Philosophical foundations

what is intelligence?
Well, What Is Intelligence?

- Many arguments over this, some fierce
  - Alan Turing
  - Roger Penrose (*The Emperor’s New Mind*)
  - Hubert Dreyfus
  - Daniel Dennett (*Consciousness Explained*)
  - Alan Newell and Herbert Simon
  - John Searle (the Chinese room argument)
  - Hilary Putnam
  - Many other philosophers and cognitive scientists
Intelligence as Symbol Processing

- Newell & Simon’s General Problem Solver (GPS) in 1957
- Idea: all intelligent behavior is symbol processing (the physical symbol system hypothesis)
- So is computing, ergo, AI should be conquered fairly soon
- “Brain in a box” based on a general problem-solving strategy applicable to mathematics problems, games, etc.
- Challenged by combinatorial explosion
The Imitation Game (Turing Test)

- Proposed by Alan Turing in 1948 as a litmus test for “thinking”.

- A human interrogator communicates with two players via teletype.
  - One player is human, the other a machine.
  - Each tries to convince the interrogator that they are the human.

*If the interrogator can’t tell which player is human, the machine must be thinking.*
**Hypothetical Conversation from Turing’s Paper**

- **Interrogator:** In the first line of your sonnet which reads “Shall I compare thee to a summer’s day,” would not “a spring day” do as well or better?

- **Computer:** It wouldn’t scan.

- **Interrogator:** How about “a winter’s day”? That would scan all right.

- **Computer:** Yes, but nobody wants to be compared to a winter’s day.
Interrogator: Would you say Mr. Pickwick reminded you of Christmas?

Computer: In a way.

Interrogator: Yet Christmas is a winter’s day, and I do not think Mr. Pickwick would mind the comparison.

Computer: I don’t think you’re serious. By a winter’s day one means a typical winter’s day, not a special one like Christmas.

What features of intelligence does this hypothetical conversation exhibit? Is this the same “intelligence” used to play chess?
Mid-60’s “trick” program fooled unsuspecting users:

User: Men are all alike.
Eliza: In what way?
User: They’re always bugging us about something or other.
Eliza: Can you think of a specific example?
Eliza exposed

Part of a conversation with a modern-day version:

User: You are a machine.
Eliza: You have your mind on machines, it seems.
User: Clever you!
Eliza: Are you sure that clever me?

But this is just a toy—what if someone is really trying?
A chatterbot written by someone really trying to win the prize:

JUDGE: What music do you like?

ZOE: I'm not really into music. What's your favorite song?

JUDGE: What is your favorite song?

ZOE: That's very interesting. Tell me more. I don't think I have a favorite. What is your favorite play?

JUDGE: I like Waiting for Godot. What is your favorite play?

ZOE: I like Waiting for Godot, too! I don't think I have a favorite. What is your favorite play?
A chat with a recent winner

Human: what will the weather be like in san jose for thanksgiving?

Mitsuku: I hope it's sunny in san jose for thanksgiving today.

Human: do you know what the weather will be like on that day?

Mitsuku: I don't know where you live to tell you what the weather will be.

Where do you live and I'll tell you the weather forecast.

Human: san jose california

Mitsuku: I hope it's sunny in san jose california today.
AI and Common Sense

- “Common sense” knowledge a huge challenge
- We take ours for granted but how can a computer “have” or “simulate” it?
- *Watson* (we’ll see more next time about this)
“Watson”

- IBM's “Watson” computer beat champions Ken Jennings and Brad Rutter in a 2-game match on Jeopardy! in 2011.

- Watson parsed clues into different keywords and fragments.

- Watson had 4TB of data content but was not connected to the Internet during the game.

- Watson executed thousands of language analysis algorithms concurrently. The more algorithms that pointed to the same answer, the more likely Watson would buzz in.

- Is Watson intelligent?
Summary

- Artificial Intelligence
- Games described as (large) decision trees
- Heuristics to guide search
- Would-be definitions of intelligence
- Turing Test