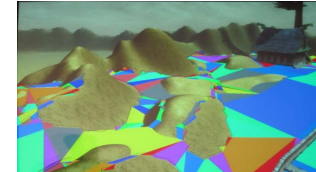


## Artificial Intelligence



## AI @ GDC

- Pathfinding
  - Planning & A\*
- Key ideas:
  - Reduce search space
- Steering
  - Following
  - Flocking
  - Grouping
  - Separation
  - Arrival
  - Avoidance
- Collisions (pushing)
  - Influence & unit circles

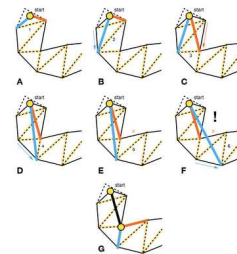
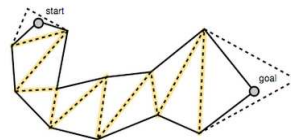


CSC404: Video Game Design © Steve Engels

Slide 2 of 57

## Funnel Algorithm

- Used to find quick paths through levels.
- Assumes that level has been decomposed into large polygons.
- Iterate through polygon corners to find narrowest funnel through passage.
- Multiple levels with different granularity
- Note: Always search for straight-line path first ☺

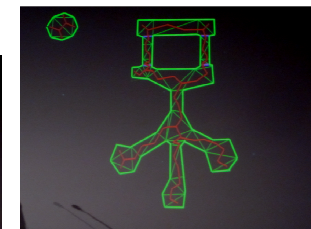
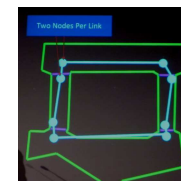
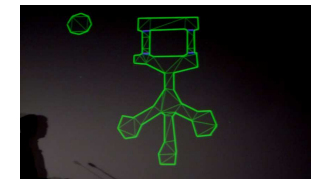


CSC404: Video Game Design © Steve Engels

Slide 3 of 57

## Pathfinding: Portals

- Create spots in each triangle edge that pathfinders use as intermediate points between regions.
- Example:
  - Playstation Move Heroes

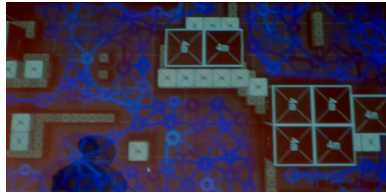


CSC404: Video Game Design © Steve Engels

Slide 4 of 57

## Influence Maps

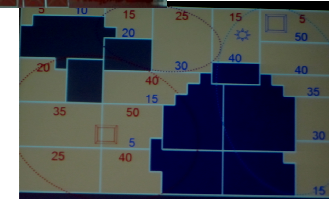
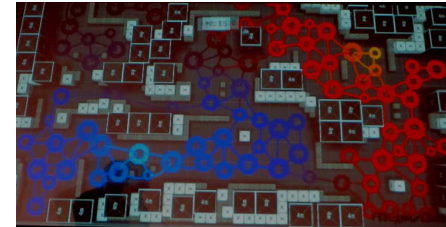
- Shows areas of control and influence for players.
- Implications:
  - Shows possible actions, future moves.
  - Defend where threatened, attack where weakest.
  - Emergent feigns and feints, teamwork.
- Based off spatial function:
  - Travel time, line-of-sight, A\* penalty, path speed, target bias, weapon choice, multipliers.



CSC404: Video Game Design © Steve Engels

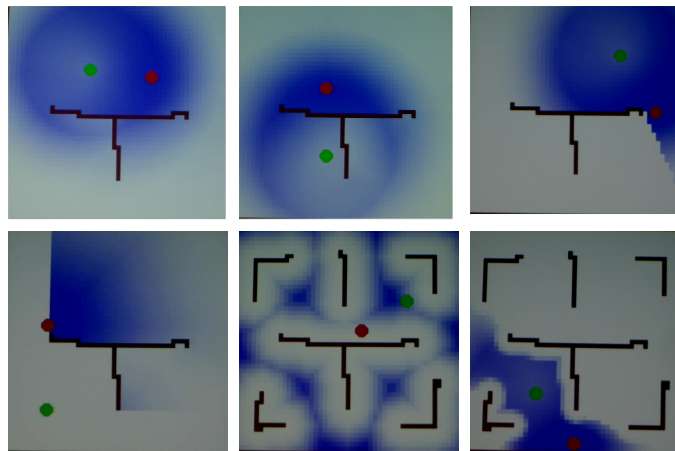
Slide 5 of 57

## Influence Maps



CSC404: Video Game Design © Steve Engels

Slide 6 of 57

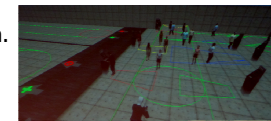
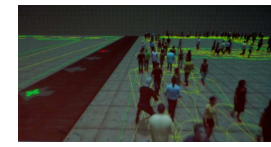
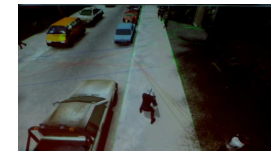


CSC404: Video Game Design © Steve Engels

Slide 7 of 57

## Intelligent NPCs

- Flow
  - Dynamic splines, dynamic lane forming.
  - Problems: twitching, piling up.
- Obstacle avoidance
  - Case-sensitive steering behaviour.
  - Social rules, self-organizing lanes.
- Action stations
  - e.g. benches, ATMs.
  - Stations “capture” NPCs in given area, take over brains & animation.
  - Once done, release NPC.
- More nuanced characters.



CSC404: Video Game Design © Steve Engels

Slide 8 of 57

## Architecture for AI

- AI algorithms are notorious short on resources.
  - Cycles, memory
- AI components: analog to electrical components.
  - Broad classification, key properties, defined I/O, interchangeable
- Class design
  - Minimal classes, data lifetime, locality of reference.
- Multithreading
  - Run planners in parallel (SIMD)
  - Break down engine into modules (like entities)
    - Perception, behaviour tree, pathfinder, targeting, animation, standard movement (wolf/shark example).
    - Physics, sensory, movement, behaviour, reasoning, animation.
  - Maximize read-only data

CSC404: Video Game Design © Steve Engels

Slide 9 of 57

## AI Issues

- Nearest neighbour searches are slow
- Player intent
  - What does a click mean?
- Destructive interference (conflicting goals)
- Grid resolution
  - Grid elements < body size
- Hierarchical searching
  - Problems with aiming for section, then searching in section.
- Randomness
  - Can produce seemingly oppressive behaviour.
  - Use Gaussians, filter out results (especially in near-win conditions).



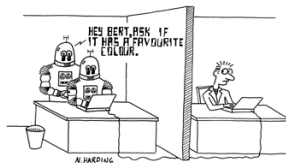
CSC404: Video Game Design © Steve Engels

Slide 10 of 57

## Artificial Intelligence

- **Artificial intelligence (AI)** is the field of creating intelligent behaviour in machines.
  - “Intelligence” understood to be measures relative to humans.
  - Labeled as **gameplay** from a developer’s point of view.
- So how do you measure intelligence?
  - Combination of perception, processing and expression.

### Turing Test:



CSC404: Video Game Design © Steve Engels

Slide 11 of 57

## Areas of Artificial Intelligence

- **Perception**
  - Language
  - Vision
- **Processing**
  - Searching
  - Planning
  - Game Trees
- **Learning**
  - Neural networks



CSC404: Video Game Design © Steve Engels

Slide 12 of 57

## AI Entities

- When creating artificial intelligence, the purpose is to produce entities that are able to operate independent of human direction
  - Often these entities are called **non-player characters** (NPCs)
- These entities need to have the following properties:
  - **autonomy** = needs no direct involvement to perform duties
  - **reactivity** = must be able to perceive and react to its environment
  - **proactivity** = must exhibit goal-directed behaviour
  - (**sociability** = interacts with other agents)

CSC404: Video Game Design © Steve Engels

Slide 13 of 57

## Intelligent Agents

- **Agents** = a software entity that exists in an **environment** and acts on that environment based on its **perceptions** and **goals**.



- Another possible agent example:
- NAVLAB (video)

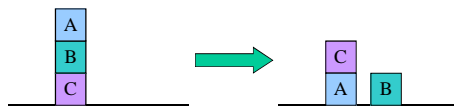


CSC404: Video Game Design © Steve Engels

Slide 14 of 57

## Agent Environments

- Usually described in terms of "worlds"
  - e.g. "**Blocks world**" = planning domain, moving blocks from one configuration to another, given movement rules



- Steve's favorite: **Vacuum-cleaner world**
  - **environment**: rooms with connections between the rooms and dirt in zero or more rooms
  - **perceptions**: current room, adjacent rooms, existence of dirt
  - **actions**: suck, move, **no-op**
  - **goal**: remove dirt from all rooms

CSC404: Video Game Design © Steve Engels

Slide 15 of 57

## Multi-Agent Systems

- When multiple agents work towards a collective goal, the rules for each agent change. It is no longer sufficient for agents to act solely in their own interests
  - Example: The Prisoner's Dilemma

		Agent A	
		Confess	– Confess
Agent B	Confess	A: 5 years B: 5 years	A: 10 years B: 1 year
	– Confess	A: 1 year B: 10 years	A: 2 years B: 2 years

What is the optimal strategy here?

CSC404: Video Game Design © Steve Engels

Slide 16 of 57

## Multi-Agent Applications

- Example: RoboCup

- robot soccer league
- international competition
- also offers search & rescue, RoboCup junior, and a dance competition



- Game example: Sports Games

- Game AI has to coordinate multiple team members for a common goal, not just for their individual goals.



## Perception

- Computer Vision

- To understand computer vision, it's good to understand human vision.
- The human retina is made up of **rods** and **cones**, which are sensitive to three main image features:

- Edges
- Corners
- Movement

- When creating devices with computer vision, it's good to incorporate these ideas.

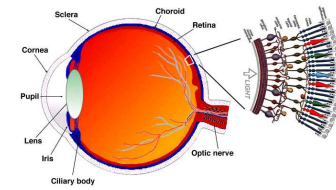
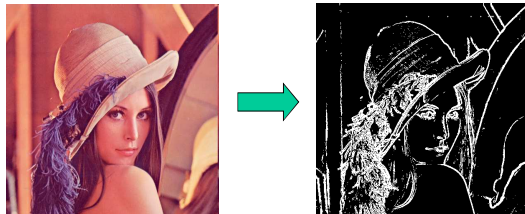


Fig. 1.1. A drawing of a section through the human eye with a schematic enlargement of the retina.

## Perception

- Vision is like computer graphics, but in reverse.
  - Start with overall image, and extract features that suggest the underlying component objects.
  - Edge detection algorithms scan the image, and produce edges wherever a change in colours occurs between neighbouring pixel values.



## Perception

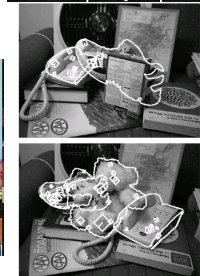
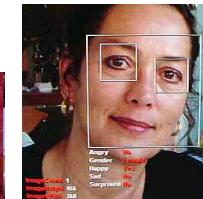
- To detect important features, scan image for edges that match a particular template image.

- Template image might be scaled, skewed and/or rotated.



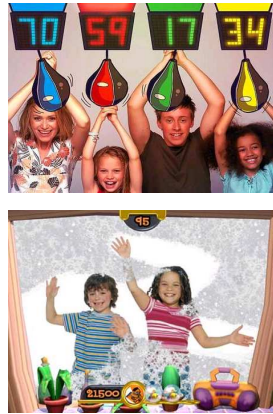
- Examples:

- Face detection
- Object recognition



## Computer Vision Example

- EyeToy



CSC404: Video Game Design © Steve Engels

Slide 21 of 57

## Perception

- Natural Language Processing
  - Natural Language is the task of translating auditory input into knowledge and back again.
- Perception stage (hard)
  - speech recognition (speech signals → words)
  - syntactic analysis (words → structure & roles)
  - semantic processing (structure & roles → meaning)
- Generation stage (easier)
  - language generation (meaning → words)
  - speech synthesis (words → speech signals)

CSC404: Video Game Design © Steve Engels

Slide 22 of 57

## “How to wreck a nice beach”

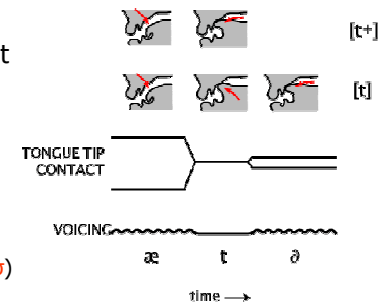
- Suppose you had a speech signal. How would you figure out what words the speech signal represents?
  - Usually, each portion of speech signal matches with a basic sound, called a **phone**, or the mental abstraction of that sound, called a **phoneme** (e.g. /k/, /a/ or /t/). Several phones can match to a single phoneme, which are considered **allophones** of each other.
  - What happens when a phone doesn't match clearly with a particular phoneme? Is it sufficient to choose the closest available match?
  - Certain classes of phones can be easily mistaken for one another
    - e.g. fricatives (/f/, /tʰ/, /v/), nasals (/m/, /n/, /ŋ/), plosives (/p/, /b/, /t/, /d/, /k/, /g/)

CSC404: Video Game Design © Steve Engels

Slide 23 of 57

## Recognizing Phonemes

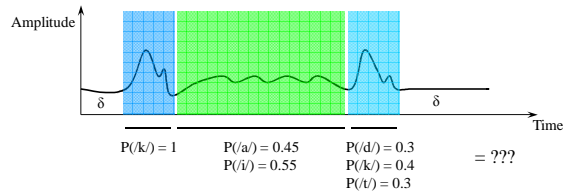
- To determine what sounds are being spoken, one must not only look at the phoneme possibilities, but also the context
  - requires large sample of labeled speech sounds to calculate the phoneme probabilities
- Requires:
  - probability of phoneme, given speech signal (S or σ)
  - probability of phoneme, given previous phoneme



CSC404: Video Game Design © Steve Engels

Slide 24 of 57

## Phoneme Sequences

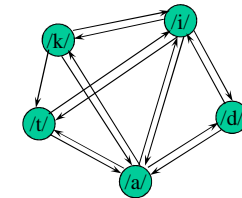


- Example: "kick" or "cat"?
  - From examining the individual probabilities alone, one would assume that this signal corresponds to the word "kick", since the /i/ and /k/ phonemes are the most likely
  - But if we consider the context...?

## Transition Probabilities

- A table of **transition probabilities** must be obtained empirically from large labeled datasets

		(current phoneme)				
(previous phoneme)		/a/	/d/	/i/	/k/	/t/
/a/		0.4	0.2	0.01	0.15	0.24
/d/		0.6	0	0.4	0	0
/i/		0.05	0.2	0.4	0.15	0.2
/k/		0.5	0	0.4	0	0.1
/t/		0.5	0	0.5	0	0



- Transition table is similar to a graph's transition matrix

## Phoneme Sequences (cont'd)

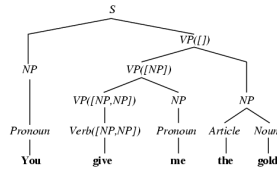
- Looking for highest probability of phonemes given speech signal
  - $= P(/k/, /i/, /k/ | \sigma) \rightarrow$  "kick", for example
  - $= P(/k/ | \sigma_1) * P(/i/ | \sigma_2) * P(/k/ | \sigma_3) * P(/k/) * P(/i/ | /k/) * P(/k/ | /i/)$
- So calculation of phoneme sequence probabilities produces the following:
  - $P(\text{"kick"}) = (1)(0.4)(0.55)(0.15)(0.4) = 13.2\%$
  - $P(\text{"kit"}) = (1)(0.4)(0.55)(0.2)(0.3) = 13.2\%$
  - $P(\text{"kid"}) = (1)(0.4)(0.55)(0.2)(0.3) = 13.2\%$
  - $P(\text{"kack"}) = (1)(0.5)(0.45)(0.15)(0.4) = 13.5\%$
  - $P(\text{"cad"}) = (1)(0.5)(0.45)(0.2)(0.3) = 13.5\%$
  - $P(\text{"cat"}) = (1)(0.5)(0.45)(0.24)(0.3) = \mathbf{16.2\%}$

## Syntactic Analysis

- Assuming that a sentence's words have been recognized correctly, how do you figure out what the words mean?
- First, one needs to figure out the syntactic role of each word in the phrase (also called **tokens**)
  - Java analogy: the `if` keyword has a different meaning if it comes at the beginning of a statement, or after the `else` keyword, or within a set of quotes.
  - The structure of a sentence and the placement of a word within that structure reveals information about the role of the word, and thus its meaning

## Parsing Methods

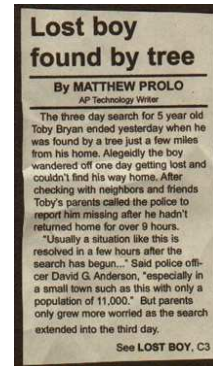
- A **parse tree** illustrates how a sentence can be broken down into component parts, until you reach the word level.
- Two basic approaches:



- **Top-down** = starting from the S symbol (representing sentence in this context, not a speech signal), search for a decomposition that results in a tree with the sentence's words as the leaves (each state represents a possible decomposition of the original S symbol)
- **Bottom-up** = given the sequence of words, search for a unification of adjacent components into non-terminals until a single tree is created with S as the root. Unification takes place by finding sequences of terminals or non-terminals that fit the right-hand side of a grammar rule, and replacing that sequence with the non-terminal on the left-hand side.

## Limits of Syntactic Analysis

- Problems with ambiguous parses
  - **Example:** Newspaper headlines
    - "Eye drops off shelf"
    - "Squad helps dog bite victim"
    - "Dealers will hear car talk at noon"
    - "Enraged cow injures farmer with ax"
    - "Two sisters reunite after eighteen years at checkout counter"
- Reference resolution (**anaphora**)
  - More newspaper headlines
    - "Grandmother of 8 makes hole in one"
    - "Two Soviet ships collide - one dies"



## Semantic Processing

- Parsing produces syntactic roles, but only produces a limited intuition of the meaning of the sentence
- For that, the system must also obtain an understanding of the sentence, based on the structure
- Semantic understanding is important for many important **natural language processing (NLP)** problems
  - interpreting commands
  - question answering
  - text summarization
  - automatic translation
    - Example: "The spirit is willing but the flesh is weak"
    - "The wine is good but the meat has gone bad"

## Basic Semantic Analyzers

- ELIZA (1966)
  - Natural language-based "therapist"
  - rearranges and substitutes certain phrases to emulate a Rogerian psychotherapist
- WordNet (Princeton University)
  - "semantic lexicon" for English
  - contains ~150,000 words grouped into synonym categories, with semantic definitions for each category
  - parse tree disambiguates part-of-speech, helps define deeper semantic context for that word
  - problematic when distinguishing between two different meanings for same part-of-speech → need semantic context



## Natural Language Example

- Façade



CSC404: Video Game Design © Steve Engels

Slide 33 of 57

## Processing

- Searching
  - Most gameplay algorithms are a form of search.
- Two main kinds of search domains:
  - solitary games
  - adversarial games
- In general, similar principles are used with each.
- Let's start with the basic searching principles of solitaire-style games...



7	6	5	3	8
	3		2	
1		9		4
6	3	1	5	4
3	1		5	
6	4	5		9
	7		9	1
2	4	1	8	6

CSC404: Video Game Design © Steve Engels

Slide 34 of 57

## Intro to Searching

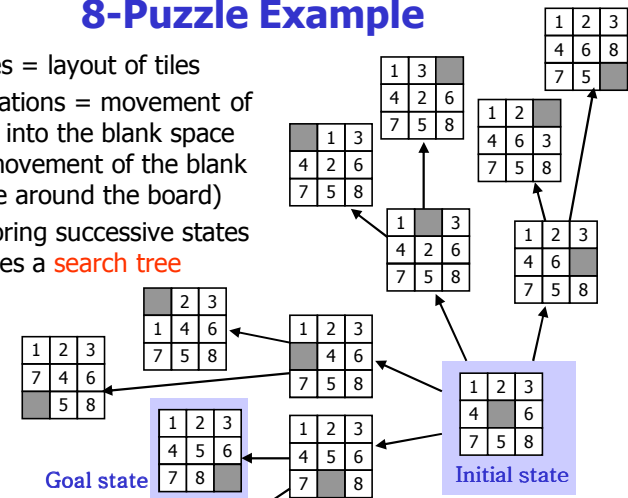
- Searching is the act of exploring possible **states** in a game or environment, to reach a specified goal
  - State in a game of chess = a layout of pieces on a board
  - State in an adventure game = your current position, where you're facing, what you're carrying and what you've done
  - State in a sports game = your current score, position, direction and player condition
- The actions that allow you to move from one state to another is called an **operation**
  - Operations in chess = moving a piece
  - Operations in an adventure game = moving your character, picking up an item, using an item, changing your clothes
  - Operations in sports game = move, throw, hit, jump, etc.

CSC404: Video Game Design © Steve Engels

Slide 35 of 57

## 8-Puzzle Example

- States = layout of tiles
- Operations = movement of a tile into the blank space (or movement of the blank space around the board)
- Exploring successive states creates a **search tree**

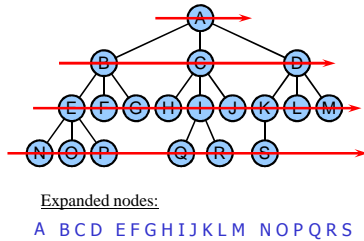


CSC404: Video Game Design © Steve Engels

Slide 36 of 57

## Breadth-first Search

- **Breadth-first search** expands the start state first, and then expands successive states, one level at a time.
- Slow, but guaranteed to find the closest solution (if one exists)
- Complexity analysis:
  - Time:  $\sim b^d$
  - Space:  $\sim b^d$
  - ( $b$  is the branching factor, the maximum number of successor states possible.  $d$  is the depth of the solution in the search tree)

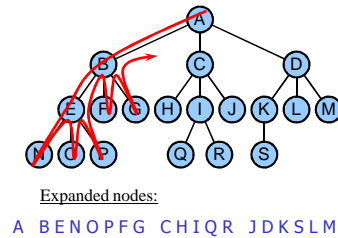


CSC404: Video Game Design © Steve Engels

Slide 37 of 57

## Depth-first Search

- **Depth-first search** expands states down a single branch of the search tree until the goal or a dead end is reached (requires backtracking)
- Faster, but dangerous. Can explore far past solution depth, and the first solution isn't guaranteed to be the best possible.
- Complexity analysis:
  - Time:  $\sim b^d$
  - Space:  $\sim b \cdot d$
  - ( $d$  is the maximum reasonable search depth)



CSC404: Video Game Design © Steve Engels

Slide 38 of 57

## Planning Example

- F.E.A.R.



CSC404: Video Game Design © Steve Engels

Slide 39 of 57

## Heuristic Searches

- Also known as **informed searches**
- A **heuristic** is a "rule of thumb" that allows an agent to estimate the distance between a particular state and the goal
- **Heuristic function** = estimated cost of path from current state  $n$  to goal state  $\rightarrow h(n)$ 
  - e.g. **straight-line heuristic**: direct distance between current position and destination on a map
- Heuristic searches ignore path cost information, and focus instead on seeking the solution

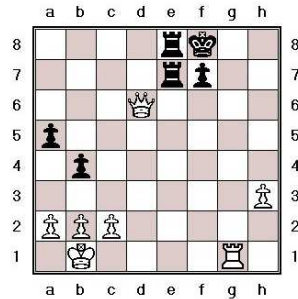


CSC404: Video Game Design © Steve Engels

Slide 40 of 57

## Adversarial Game Playing

- Most game-playing involves searching that takes place in an adversarial domain
- Not concerned with solitaire-like games that do not involve an opponent
- Main question of game theory: What is the best move do make in the current situation?

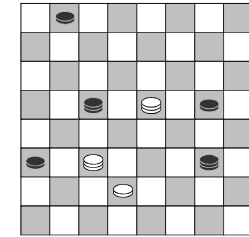


CSC404: Video Game Design © Steve Engels

Slide 41 of 57

## Assumptions

1. Domain can't be exhaustively explored
2. No **strictly dominating strategies**
3. Both the player's and opponent's strategies and positions are knowable
4. Both the player and the opponent are **rational**
5. Strategies are comprised of heuristics that can measure the "goodness" of any position with a numerical result
6. Game searches are pursuing a single goal
7. **Zero-sum** games being examined



What move by white guarantees victory?

CSC404: Video Game Design © Steve Engels

Slide 42 of 57

## Game Example: Othello

- In Othello, rows of opponent pieces are captured by surrounding them with two pieces of your own colour.
- As a result, positions on the edges and corners are more valuable than the middle.
- The value of an arrangement of pieces can be the sum of the weights of the player's pieces, minus the weight of the opponent's pieces.



8	4	4	4	4	4	4	4	8
4	1	1	1	1	1	1	1	4
4	1	1	1	1	1	1	1	4
4	1	1	1	1	1	1	1	4
4	1	1	1	1	1	1	1	4
4	1	1	1	1	1	1	1	4
4	1	1	1	1	1	1	1	4
8	4	4	4	4	4	4	4	8

CSC404: Video Game Design © Steve Engels

Slide 43 of 57

## Game Trees

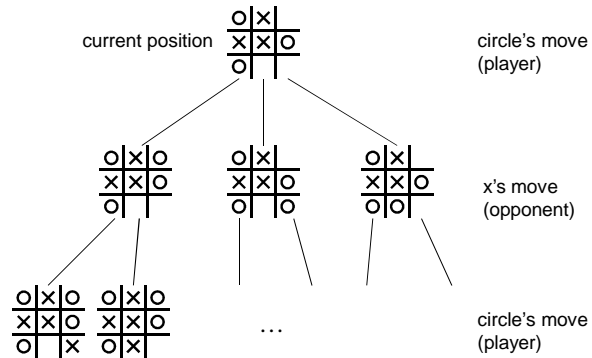
- **Game trees** are used to reflect all two-player game scenarios
  - multi-player scenarios are possible as well, and are an extension of two-player games
- Same as search trees, but take the opponent's goals into account as well
- Game trees form the foundation for all intelligent game-playing algorithms:
  - **Checkers**: AI can beat world champion
  - **Othello/Reversi**: Basic computer can beat best human player
  - **Go**: Amateur human can beat best computer player
  - **Chess**: Humans and computers still battling for top spot

CSC404: Video Game Design © Steve Engels

Slide 44 of 57

## Game Tree Example

- Tic-Tac-Toe:



CSC404: Video Game Design © Steve Engels

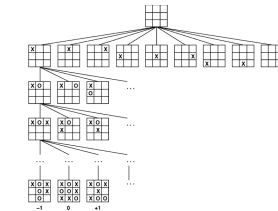
Slide 45 of 57

## Branching in Game Trees

- Like any search tree, game tree branching can get out-of-hand very quickly
  - tic-tac-toe has 97,162 potential game positions to consider
  - chess has an average branching factor  $b = 42$ 
    - two moves on each side produces over 3 million possible positions!

- Solution:**

- look limited moves ahead
- use good heuristic function
  - keep extensive databases
- use **minimax** principle
- prune search tree



CSC404: Video Game Design © Steve Engels

Slide 46 of 57

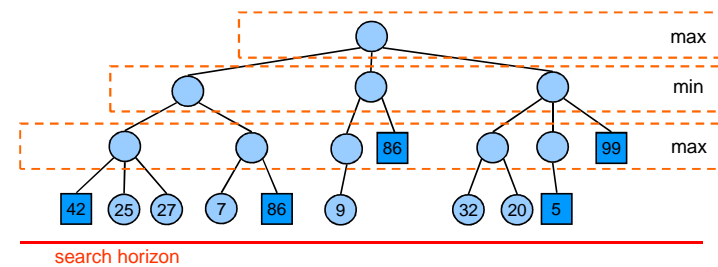
## Minimax Principle

- Invented by von Neumann and Morgenstern in 1944 as part of game theory.
- Involves growing a game tree to the **search horizon**.
  - The **search horizon** is defined by the number of moves the computer looks ahead. If the computer look  $n$  moves ahead for itself and  $n - 1$  moves for the opponent, we say the computer is playing  $2n - 1$  **ply**.
- Within the tree bounded by the search horizon, apply the heuristic function to all leaves to calculate the utility of the position at each leaf.
- Idea behind **minimax** is that the AI player tries to maximize the utility while the opponent tries to minimize it.
- Note:**
  - Leaf states are not necessarily end states
  - End states do not necessarily satisfy the goal conditions
  - Heuristic values at end states can vary

CSC404: Video Game Design © Steve Engels

Slide 47 of 57

## Minimax Example



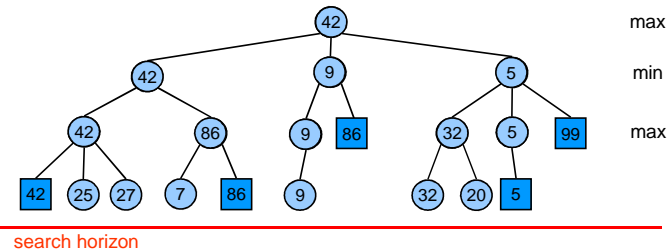
- From root position, what branch should the AI player pursue to achieve the highest eventual score?
  - Note: heuristic values are from the player's perspective, not the opponent's. It is assumed that the opponent wishes to minimize the player's score at each min stage.

CSC404: Video Game Design © Steve Engels

Slide 48 of 57

## Minimax Example (cont'd)

- Minimax algorithm:
  - from bottom level to top, fill in node with either the maximum of its children (on max levels) or the minimum of its children (on min levels)



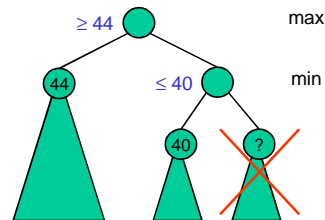
→ Best path is left-most branch

## Minimax & Pruning

- Assumptions:
  - Heuristic is known by both player and opponent
  - both will make the most sensible move, given their goals
- Minimax alone will not solve the search problem
  - need reduction in search space in order to increase chances of arriving at the goal
- Pruning is a major technique used to reduce the branch density and speed up the search
  - e.g. prior knowledge about certain positions can allow any further positions to be discounted
- $\alpha\beta$ -pruning can cut down the number of nodes searched *without losing information*

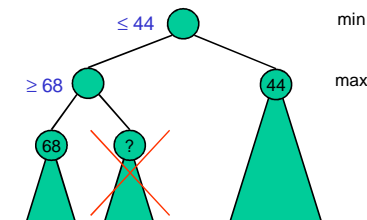
## $\alpha$ -Pruning

- When exploring nodes in game tree, assumption is a limited depth-first search, from left to right.
- $\alpha$ -pruning keys on the max level, and discards any branches that won't affect the max level value.
- Example:



## $\beta$ -Pruning

- $\beta$ -pruning is the same as  $\alpha$ -pruning, only from the perspective of the min levels



## αβ-Pruning (cont'd)

- Before a node and its subtree can be discarded, the algorithm requires tentative values for parent and grandparent nodes in tree
- Complexity reduction:
  - αβ-pruning reduces searching to  $\sim n^{1/2}$  states
  - search is able to explore twice the depth of a regular search
    - very helpful with real-time game applications



CSC404: Video Game Design © Steve Engels

Slide 53 of 57

## Search Enhancements

- Other techniques exist for speeding up search through game space
- **Transposition tables**
  - tables that record past positions, to avoid searching previously-explored subtrees
  - also used to eliminate subtrees that are permutations of other positions
    - e.g. Tic-tac-toe: initial branching reduces from  $b=9$  to  $b=3$
  - **Disadvantage:** most effective with iterative deepening search, which undermines use of αβ-pruning
  - must be careful not to equate two states whose positions are similar but situations are different
    - e.g. castling in chess, inventory in first-person shooters

CSC404: Video Game Design © Steve Engels

Slide 54 of 57

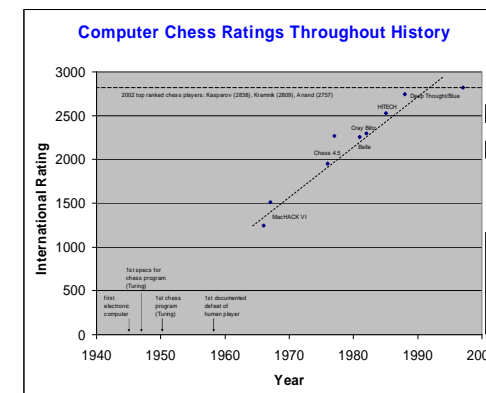
## Search Enhancements (cont'd)

- **Opening book**
  - games with set initial positions tend to have predetermined patterns (i.e. chess)
  - opening books reduce the search space in initial situations by performing one of a set of opening actions
- **Endgames / Killer moves**
  - certain situations that match recognized patterns can trigger a series of actions that guarantee an increase in the utility of the game state
  - endgames in particular lead to outcomes that guarantee a win
- **Variable depth**
  - the search horizon can be adjusted if a particular path requires more exploration (i.e. to realize a sub-goal)

CSC404: Video Game Design © Steve Engels

Slide 55 of 57

## Game Tree Example

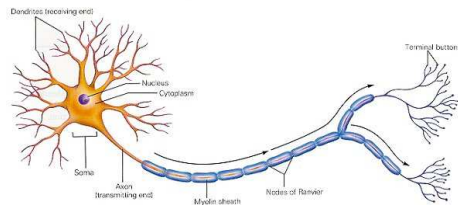


CSC404: Video Game Design © Steve Engels

Slide 56 of 57

## Learning

- Neural networks
  - Most rationalist approaches to AI involve modeling cognitive processes in human behaviour.
  - The connectionist approach takes this one step further, creating models that are based on the activity of **neurons** in the brain

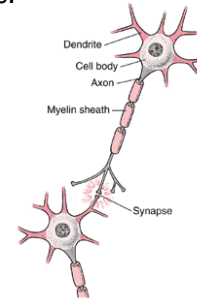


CSC404: Video Game Design © Steve Engels

Slide 57 of 57

## Biological Neural Cells

- Each neuron is composed of three parts:
  - **dendrites**: receive electrical signals from other neurons
  - **axon**: transmits electrical pulse to other neurons
  - **cell body**: decides what kind of electrical pulse to transmit, given input signals
- The body's nervous system and the brain's higher functions are composed of networks of these neurons
  - **biological neural network**
- Gave rise to computational models of these networks
  - **artificial neural networks**

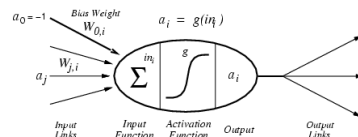


CSC404: Video Game Design © Steve Engels

Slide 58 of 57

## Artificial Neural Networks

- Artificial neural networks (or simply neural networks) are made up of a series of nodes that represent the neurons of the brain
  - Each node (or unit) is connected to other nodes through directed links
  - Each node is defined by its **input function**, which sums the activations energies from other nodes, and an **activation function** that produces an output value for this node, based on the result of the input function.



CSC404: Video Game Design © Steve Engels

Slide 59 of 57

## Neural Network Structure

- Every neural network has three layers:
  - **input layer**: the nodes that record the input values for the training case, one node per input value (no processing).
  - **output layer**: the nodes whose activation functions produce the output values for the neural network.
  - **hidden layer(s)**: the nodes between the input and output layers.
- **Prime time analogy: *House M.D.***
  - House has team of residents.
  - Residents examine many symptoms.
  - When symptoms occur, those "activate" certain residents.
  - House diagnoses based on residents.

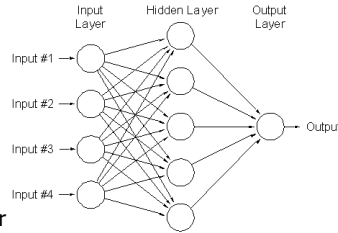


CSC404: Video Game Design © Steve Engels

Slide 60 of 57

## Hidden Layer "Rules"

- Hidden layers of neural networks store internal "knowledge", used in processing the output.
- **Rules of Thumb:**
  - only one layer is really necessary
  - no universal rule for number of nodes in the hidden layer, except that more nodes implies more detail in output distribution → potential for overfitting, in sparse data cases
  - some problems require no hidden layer (i.e. linear separation problems)
    - use single-layer feed-forward neural network → **perceptron**



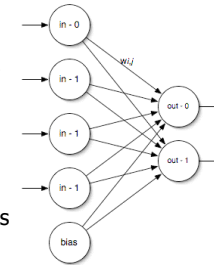
## Perceptrons

- Perceptrons can differentiate different sections in the input data, as long as they are linearly separable
  - similar to support vector machines
- How does learning occur within the network?

- want to minimize the squared error of the neural network

$$E = \frac{1}{2} \text{Err}^2 \equiv \frac{1}{2} (y - h_w(x))^2$$

- $y$  is the true output that should be expected, given the input  $x$ , weights  $w$  and actual output  $h_w(x)$



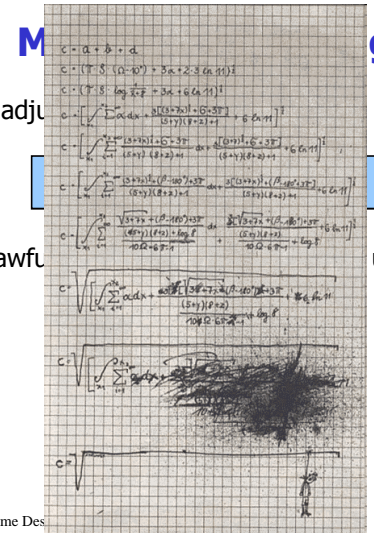
## Perceptron Learning

- In order to minimize this error, find the difference between the expected and actual output, and use that to adjust the weights that affect that output
- Using *House* analogy again:
  - If House is right, he thinks more highly of the residents he listened to, and tells them so. In turn, they think more highly of the symptoms they considered in their diagnosis.
  - If House is wrong, he gives less weight to the residents he listened to, and they in turn give less weight to the symptoms they considered.
  - This continues until steady-state is achieved.
- This is called the **back-propagation algorithm**.



- Weight adjustment

- This is awful until third year.





## Qualities of Neural Networks

- Advantages:
  - Biological basis
    - learning models human learning techniques; more sound than other contrived techniques
  - Auto-organization
    - internal representation of knowledge is not outlined or affected by human operators
  - Fault tolerance
    - partial destruction of network structure is compensated for by parallel operation of remaining network
  - Flexibility
    - learned model can work on unseen data
  - Speed
    - once model is trained, responses are obtained in real-time
- Disadvantages
  - Arcane quality
    - can't inspect internal workings to determine what it learned, since hidden state values don't mean anything at a higher level

CSC404: Video Game Design © Steve Engels

Slide 65 of 57

## Learning Example

- Black & White



CSC404: Video Game Design © Steve Engels

Slide 66 of 57