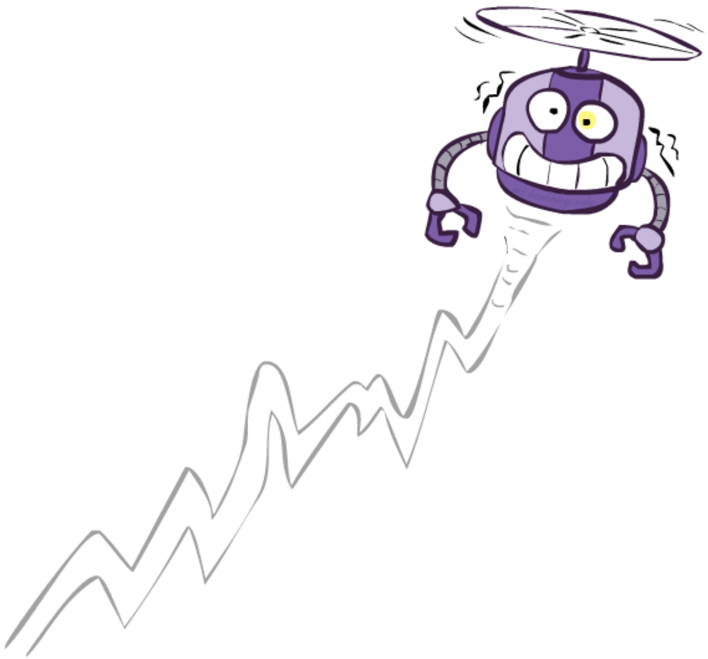
Simulated annealing

- Resembles the annealing process used to cool metals slowly to reach an ordered (low-energy) state
- Basic idea:
  - Allow “bad” moves occasionally, depending on “temperature”
  - High temperature => more bad moves allowed, shake the system out of its local minimum
  - Gradually reduce temperature according to some schedule
  - Sounds pretty flaky, doesn’t it?

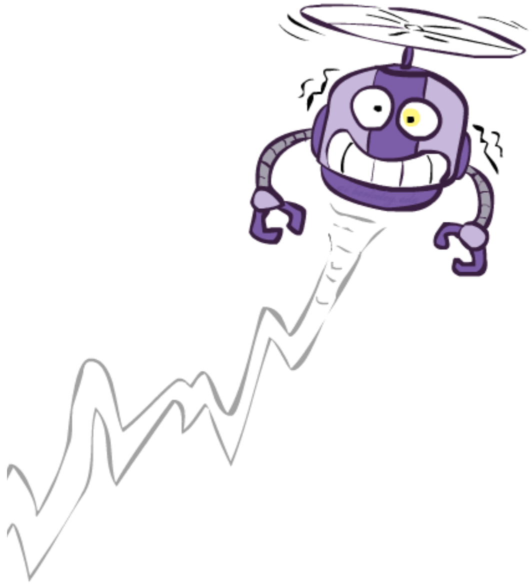
Simulated annealing algorithm

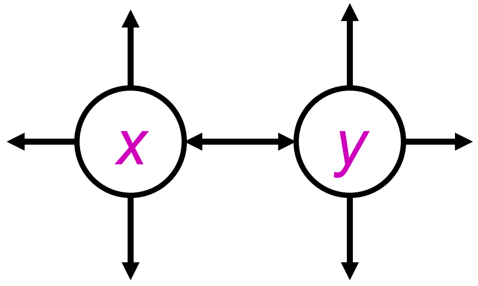
```
function SIMULATED-ANNEALING(problem,schedule) returns a  state
current ← problem.initial-state
for t = 1 to ∞ do
        T ←schedule(t)
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← next.value – current.value
        if ΔE > 0 then current ← next
                  else current ← next only with probability e^(ΔE/T)
```



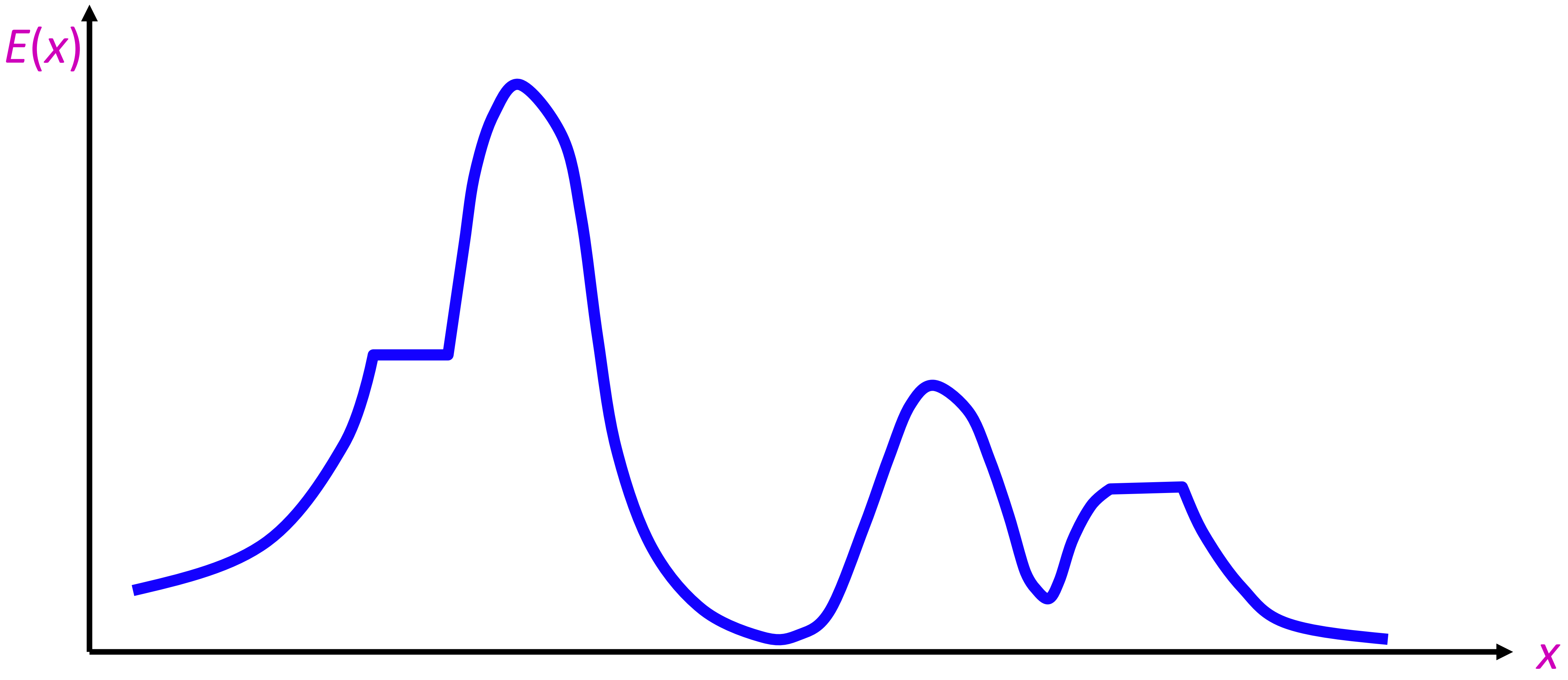
Simulated Annealing

- Theoretical guarantee:
  - Stationary distribution (Boltzmann): $P(x) \propto e^{E(x)/T}$
  - If T decreased slowly enough, will converge to optimal state!
- Proof sketch
  - Consider two adjacent states x, y with $E(y) > E(x)$ [high is good]
  - Assume $x \rightarrow y$ and $y \rightarrow x$ and outdegrees $D(x) = D(y) = D$
  - Let $P(x)$, $P(y)$ be the equilibrium occupancy probabilities at T
  - Let $P(x \rightarrow y)$ be the probability that state x transitions to state y



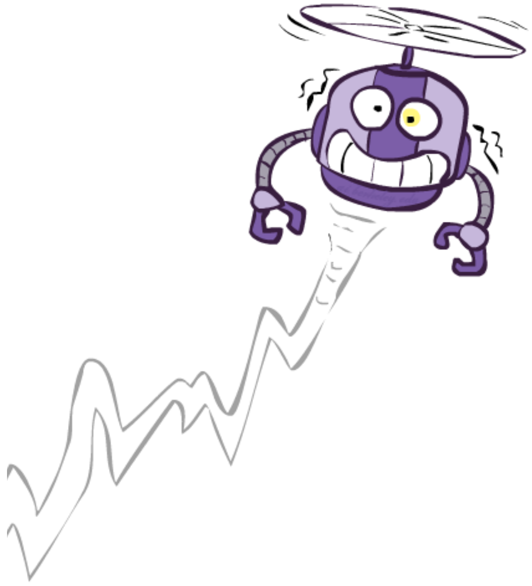


Occupation probability as a function of T



Simulated Annealing

- Is this convergence an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - “Slowly enough” may mean exponentially slowly
  - Random restart hillclimbing also converges to optimal state…
- Simulated annealing and its relatives are a key workhorse in VLSI layout and other optimal configuration problems

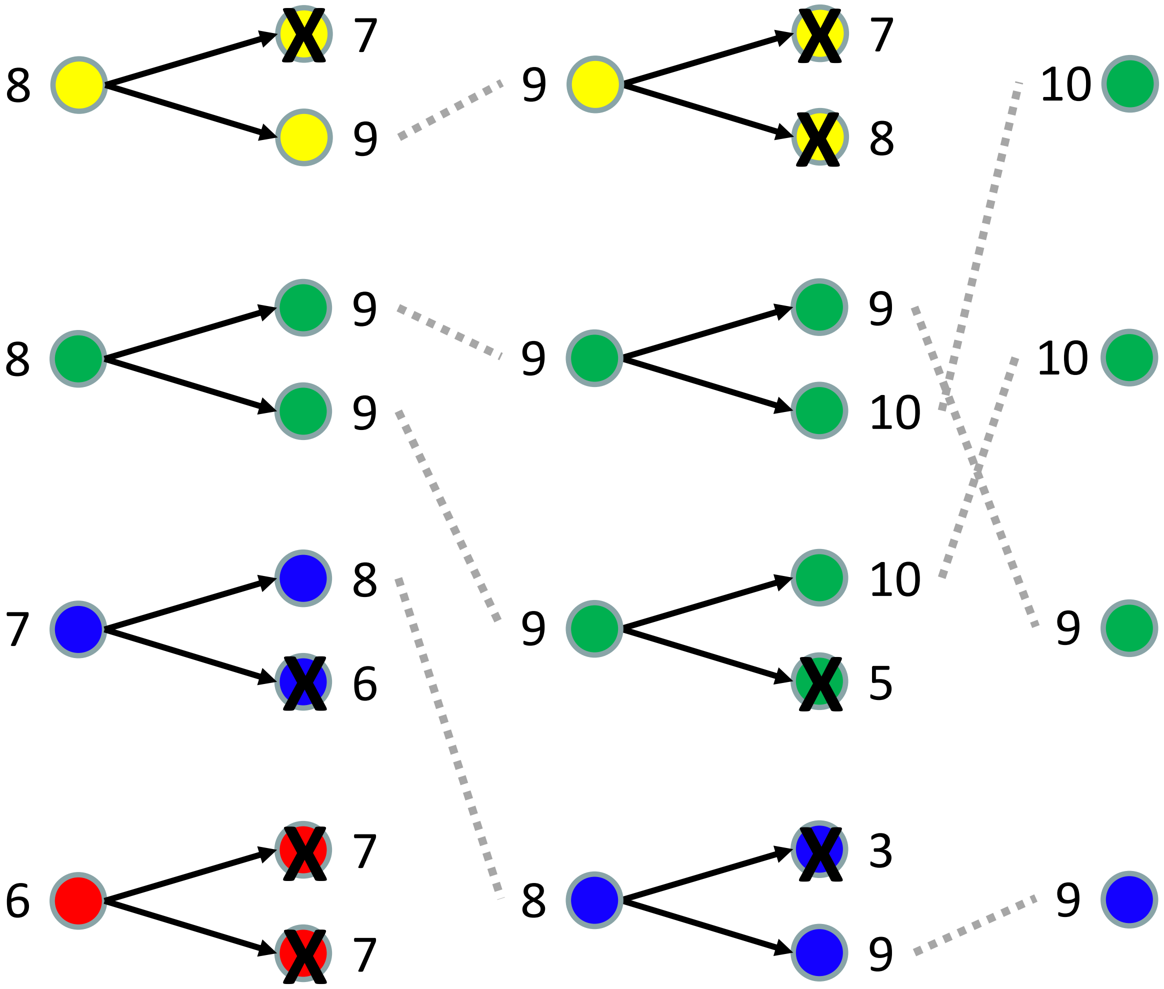


Local beam search

- Basic idea:
  - *K* copies of a local search algorithm, initialized randomly
  - For each iteration
    - Generate ALL successors from *K* current states
    - Choose best *K* of these to be the new current states

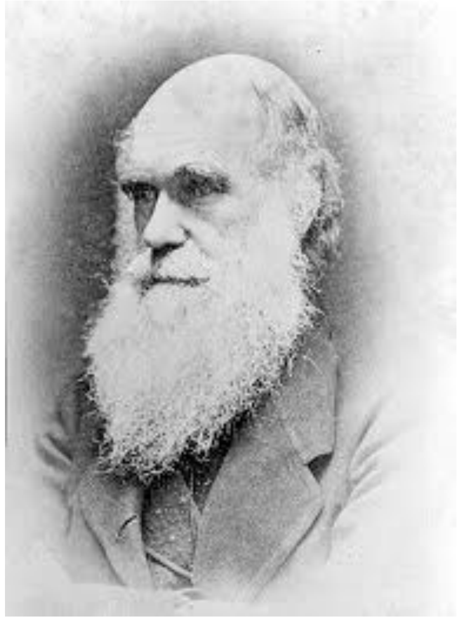
Or, K chosen randomly with a bias towards good ones

Beam search example (K=4)

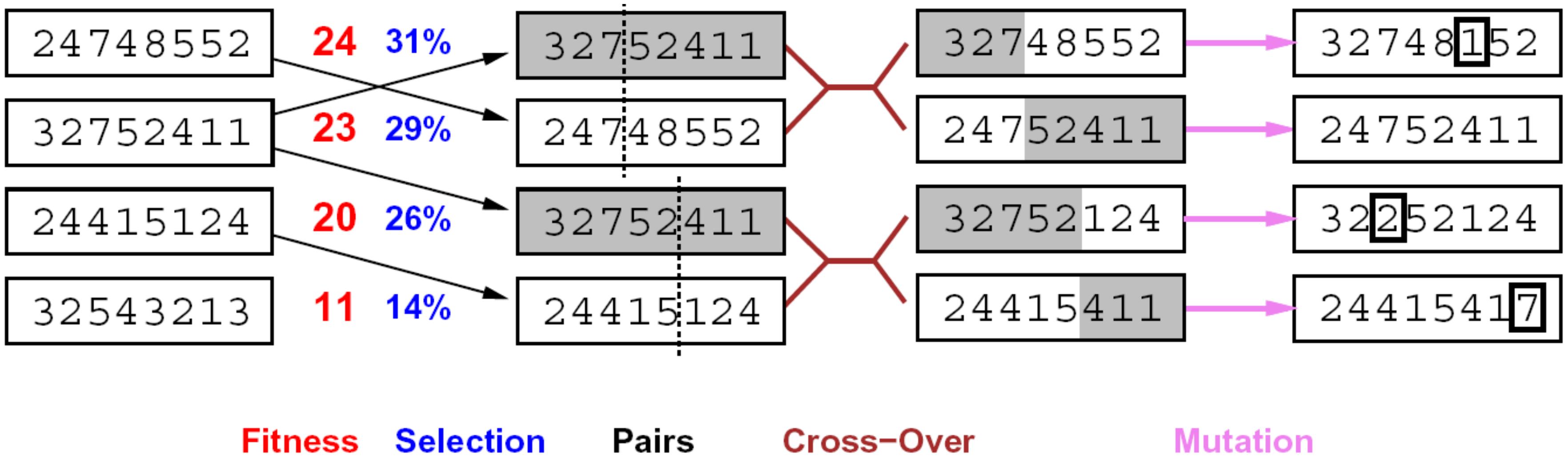


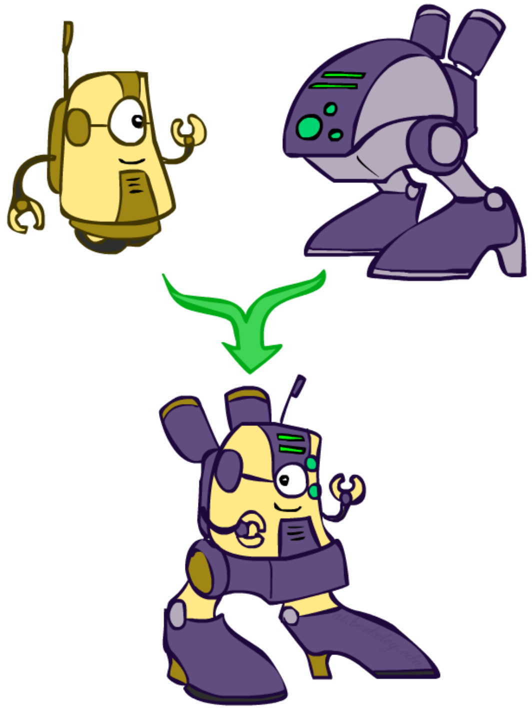
Local beam search

- Why is this different from *K* local searches in parallel?
  - The searches ***communicate***! "Come over here, the grass is greener!"
- What other well-known algorithm does this remind you of?
  - Evolution!



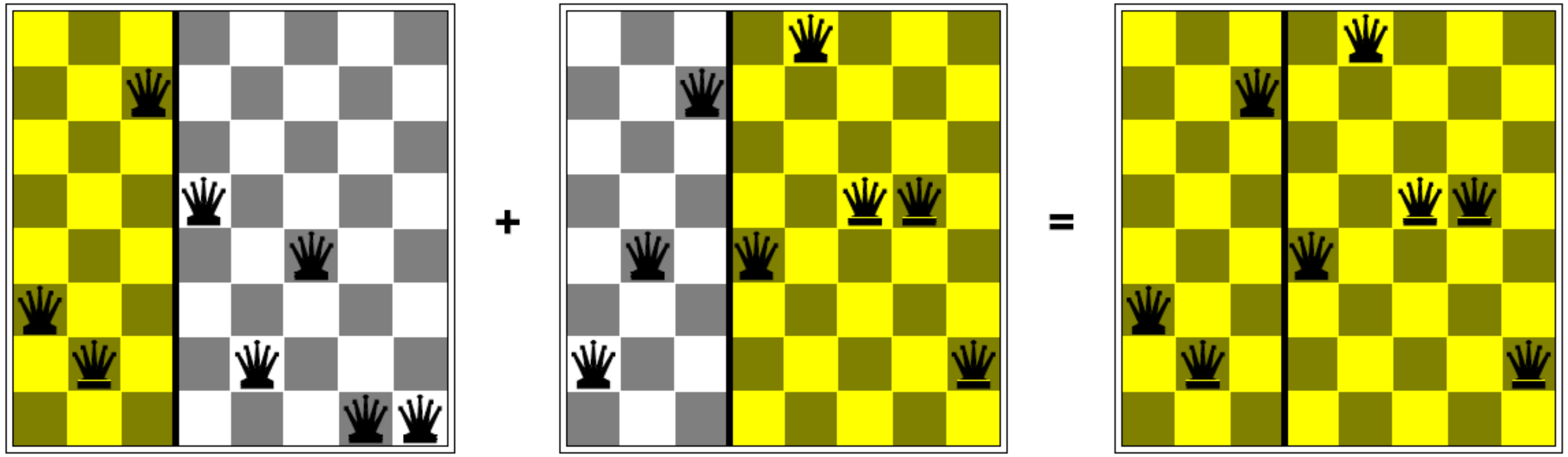
Genetic algorithms





- Genetic algorithms use a natural selection metaphor
  - Resample *K* individuals at each step (selection) weighted by fitness function
  - Combine by pairwise crossover operators, plus mutation to give variety

Example: N-Queens

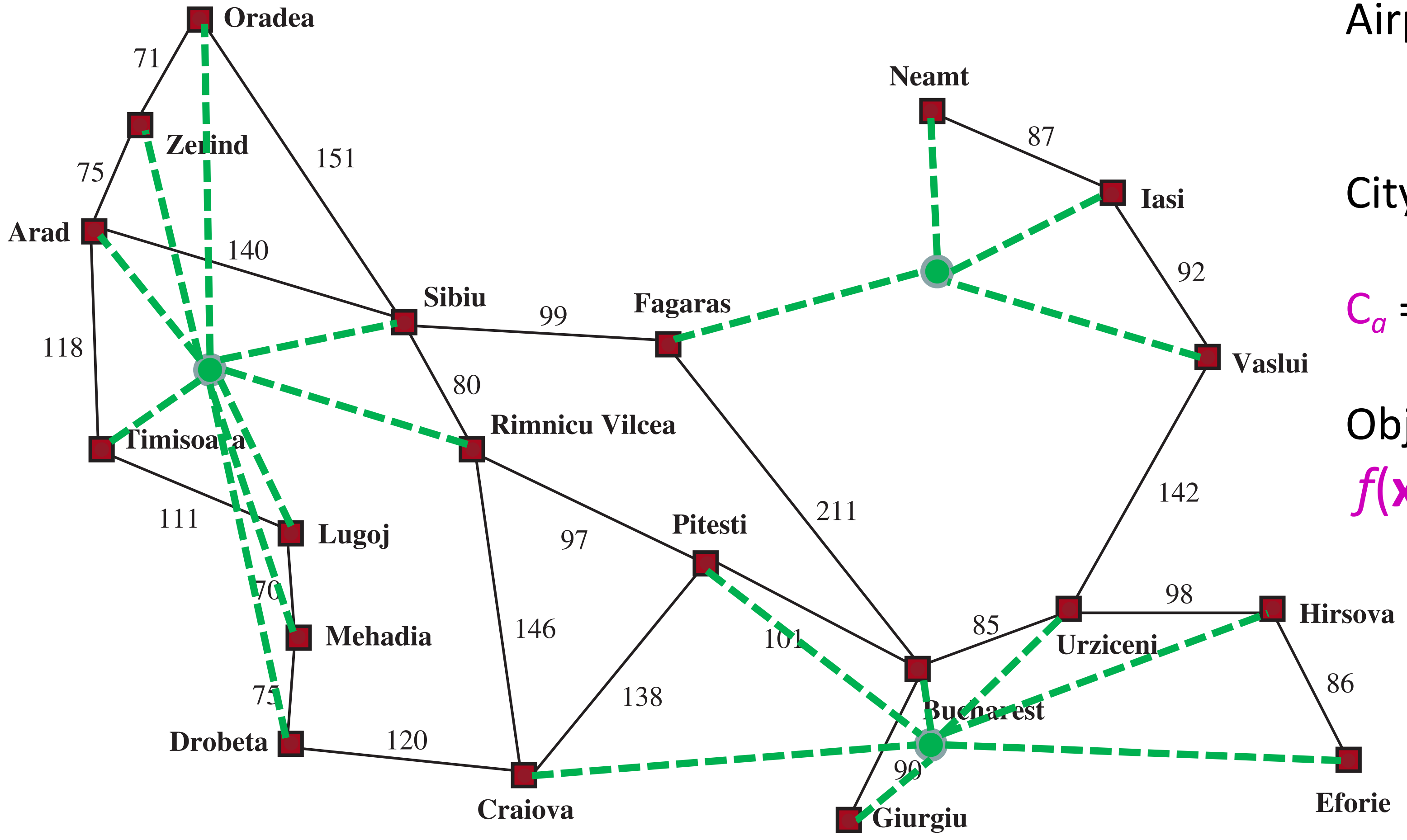


- Does crossover make sense here?
- What would mutation be?
- What would a good fitness function be?

Local search in continuous spaces

Example: Siting airports in Romania

Place 3 airports to minimize the sum of squared distances from each city to its nearest airport



Airport locations

$\mathbf{x} = (x_1,y_1), (x_2,y_2), (x_3,y_3)$

City locations (x_c,y_c)

C_a = cities closest to airport a

Objective: minimize

$$f(\mathbf{x}) = \Sigma_a \Sigma_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$$

Handling a continuous state/action space

1. Discretize it!
   - Define a grid with increment δ, use any of the discrete algorithms
2. Choose random perturbations to the state
   a. First-choice hill-climbing: keep trying until something improves the state
   b. Simulated annealing
3. Compute gradient of $f(\mathbf{x})$ analytically

Finding extrema in continuous space

- Gradient vector $\nabla f(\mathbf{x}) = (\partial f/\partial x_1, \partial f/\partial y_1, \partial f/\partial x_2, \ldots)^{\mathrm{T}}$
- For the airports, $f(\mathbf{x}) = \Sigma_a \Sigma_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$
- $\partial f/\partial x_1 = \Sigma_{c \in C_1} 2(x_1 - x_c)$
- At an extremum, $\nabla f(\mathbf{x}) = 0$
- Can sometimes solve in closed form: $x_1 = (\Sigma_{c \in C_1} x_c)/|C_1|$
- Is this a local or global minimum of f?
- Gradient descent: $\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla f(\mathbf{x})$
  - Huge range of algorithms for finding extrema using gradients

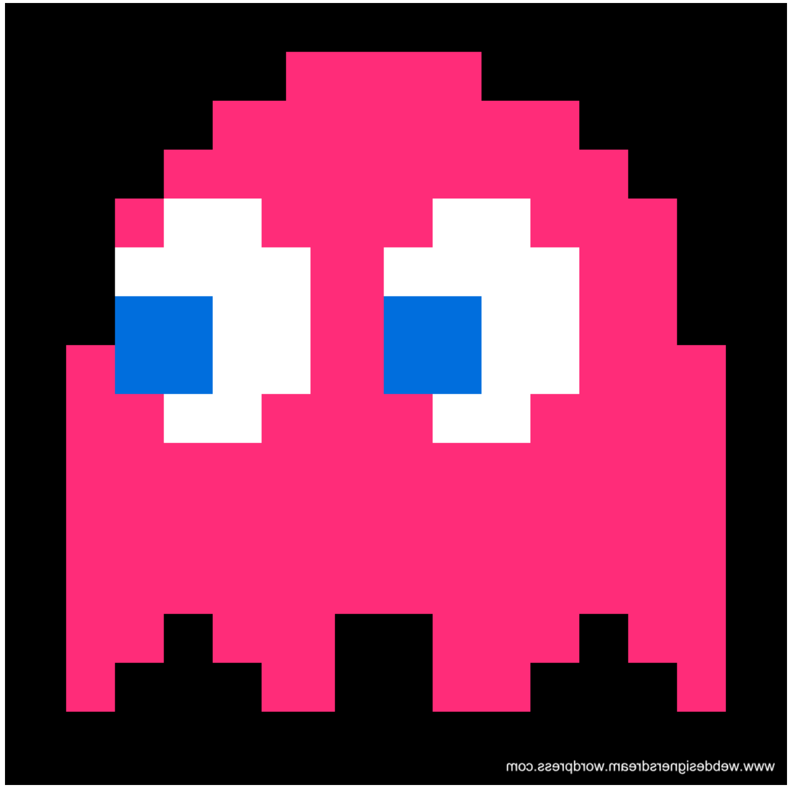
Summary

- Many configuration and optimization problems can be formulated as local search
- General families of algorithms:
  - Hill-climbing, continuous optimization
  - Simulated annealing (and other stochastic methods)
  - Local beam search: multiple interaction searches
  - Genetic algorithms: break and recombine states

Many machine learning algorithms are local searches

CS 188: Artificial Intelligence

Propositional Logic I



Instructors: Stuart Russell and Peyrin Kao

University of California, Berkeley

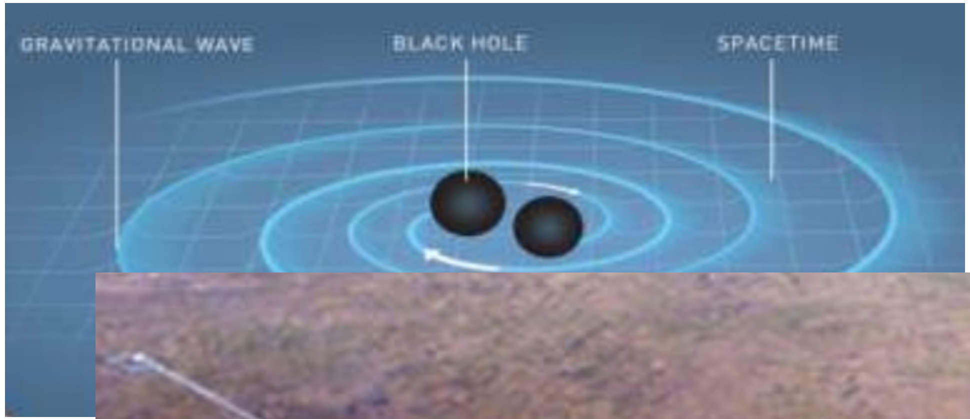
Outline

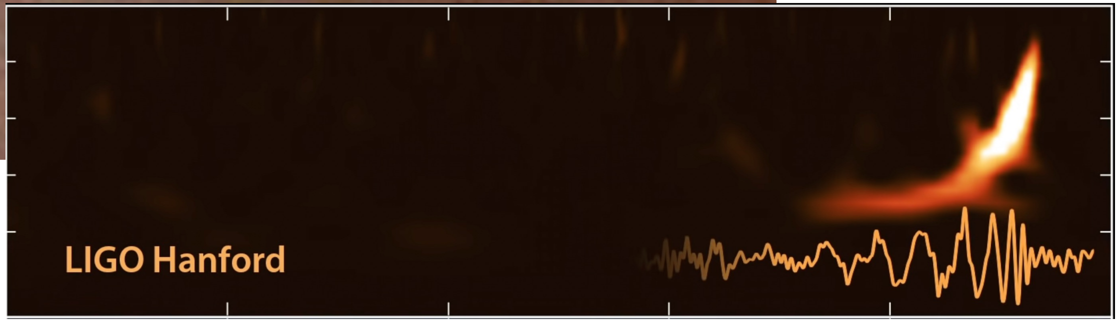
1. Propositional Logic I
   - Basic concepts of knowledge, logic, reasoning
   - Propositional logic: syntax and semantics, Pacworld example
2. Propositional logic II
   - Inference by theorem proving (briefly) and model checking
   - A Pac agent using propositional logic

Agents that know things

- Agents acquire knowledge through perception, learning, language
  - Knowledge of the effects of actions (“transition model”)
  - Knowledge of how the world affects sensors (“sensor model”)
  - Knowledge of the current state of the world
- Can keep track of a partially observable world
- Can formulate plans to achieve goals
- Can design and build gravitational wave detectors…..

LIGO

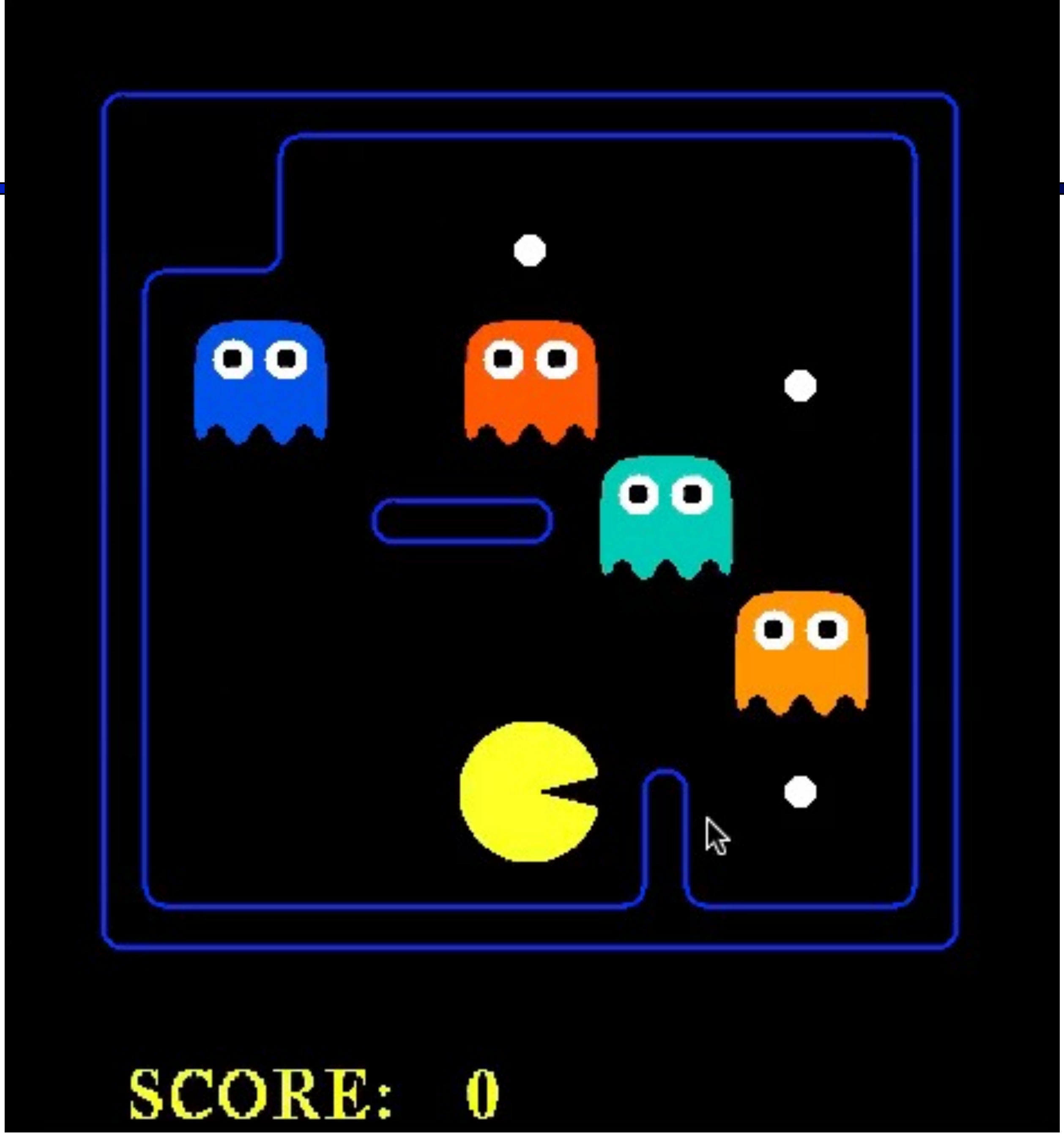




Knowledge, contd.

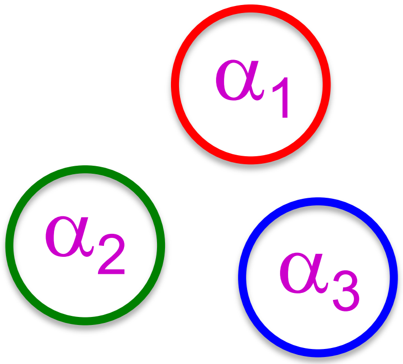
- Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent (or other system):
  - ***Tell*** it what it needs to know (or have it ***Learn*** the knowledge)
  - Then it can ***Ask*** itself what to do—answers should follow from the KB
- Agents can be viewed at the ***knowledge level***
  i.e., what they ***know***, regardless of how implemented
- A single inference algorithm can answer any answerable question

Knowledge base	Domain-specific facts
Inference engine	Generic code


SCORE: 0

Logic

- ***Syntax***: What sentences are allowed?
- ***Semantics***:
  - What are the ***possible worlds***?
  - Which sentences are ***true*** in which worlds? (i.e., ***definition*** of truth)



Syntaxland

Semanticsland

Different kinds of logic

- Propositional logic
  - Syntax: $P \vee (\neg Q \wedge R)$; $X_1 \Leftrightarrow (\text{Raining} \Rightarrow \neg\text{Sunny})$
  - Possible world: {P=true,Q=true,R=false,S=true} or 1101
  - Semantics: $\alpha \wedge \beta$ is true in a world iff is α true and β is true (etc.)
- First-order logic
  - Syntax: $\forall x\, \exists y\, P(x,y) \wedge \neg Q(Joe,f(x)) \Rightarrow f(x)=f(y)$
  - Possible world: Objects o_1, o_2, o_3; P holds for <o_1,o_2>; Q holds for <o_3>; $f(o_1)=o_1$; $Joe=o_3$; etc.
  - Semantics: $\phi(\sigma)$ is true in a world if $\sigma=o_j$ and ϕ holds for o_j; etc.

Different kinds of logic, contd.

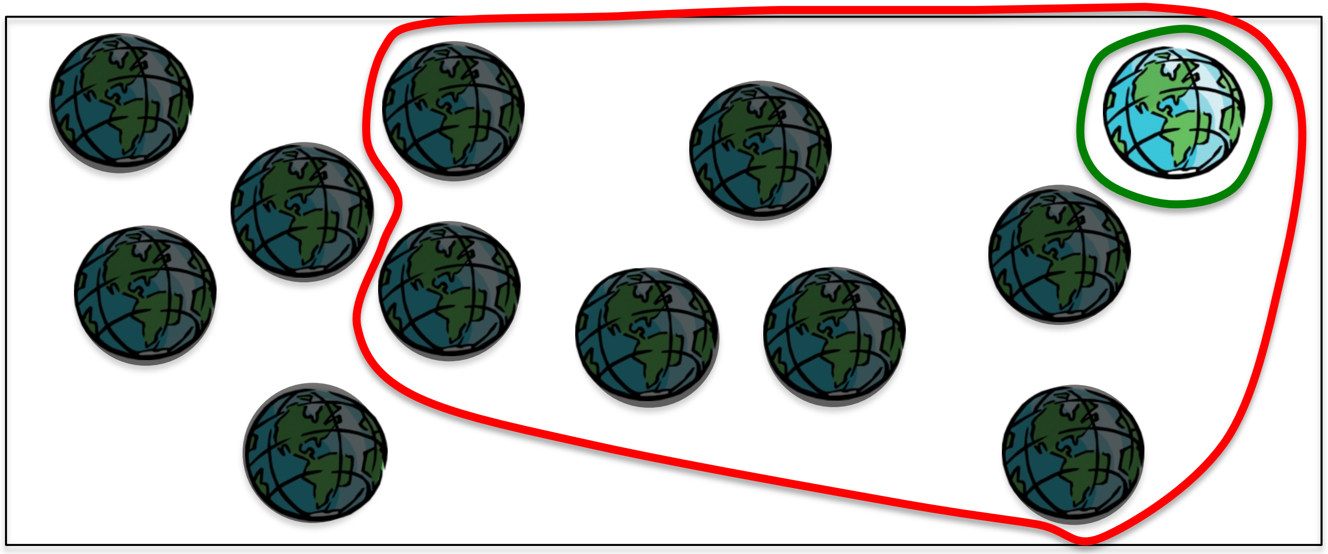
- Relational databases:
  - Syntax: ground relational sentences, e.g., *Sibling*(*Ali*,*Bo*)
  - Possible worlds: (typed) objects and (typed) relations
  - Semantics: sentences in the DB are true, everything else is false
    - Cannot express disjunction, implication, universals, etc.
    - Query language (SQL etc.) typically some variant of first-order logic
    - Often augmented by first-order rule languages, e.g., Datalog
  - Knowledge graphs (roughly: relational DB + ontology of types and relations)
    - Google Knowledge Graph: 5 billion entities, 500 billion facts, >30% of queries
    - Facebook network: 2.93 billion people, trillions of posts, maybe quadrillions of facts

Inference: entailment

- ***Entailment***: $\alpha \models \beta$ ("α entails β" or "β follows from α") iff <u>in every world where α is true, β is also true</u>
  - I.e., the α-worlds are a ***<u>subset</u>*** of the β-worlds [***models***(α) $\subseteq$ ***models***(β)]
- In the example, $\alpha_2 \models \alpha_1$
- (Say α_2 is $\neg Q \wedge R \wedge S \wedge W$
  α_1 is $\neg Q$)

α_1

α_2



Inference: proofs

- A proof is a ***demonstration*** of entailment between α and β
- ***Sound*** algorithm: everything it claims to prove is in fact entailed
- ***Complete*** algorithm: every that is entailed can be proved

Inference: proofs

- Method 1: ***model-checking***
  - For every possible world, if α is true make sure that is β true too
  - OK for propositional logic (finitely many worlds); not easy for first-order logic
- Method 2: ***theorem-proving***
  - Search for a sequence of proof steps (applications of ***inference rules***) leading from α to β
  - E.g., from P and $(P \Rightarrow Q)$, infer Q by ***Modus Ponens***

Propositional logic syntax

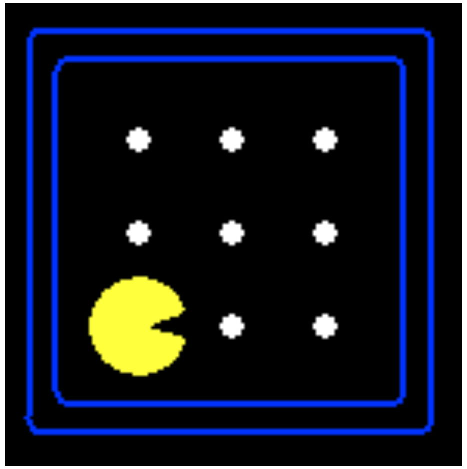
- Given: a set of proposition symbols $\{X_1, X_2, \ldots, X_n\}$
  - (we often add True and False for convenience)
- X_i is a sentence
- If α is a sentence then $\neg\alpha$ is a sentence
- If α and β are sentences then $\alpha \wedge \beta$ is a sentence
- If α and β are sentences then $\alpha \vee \beta$ is a sentence
- If α and β are sentences then $\alpha \Rightarrow \beta$ is a sentence
- If α and β are sentences then $\alpha \Leftrightarrow \beta$ is a sentence
- And p.s. there are no other sentences!

Propositional logic semantics

- Let *m* be a model assigning true or false to $\{X_1, X_2, \ldots, X_n\}$
- If α is a symbol then its truth value is given in *m*
- $\neg\alpha$ is true in *m* iff α is false in *m*
- $\alpha \wedge \beta$ is true in *m* iff α is true in *m* <u>and</u> β is true in *m*
- $\alpha \vee \beta$ is true in *m* iff α is true in *m* <u>or</u> β is true in *m*
- $\alpha \Rightarrow \beta$ is true in *m* iff α is false in *m* <u>or</u> β is true in *m*
- $\alpha \Leftrightarrow \beta$ is true in *m* iff $\alpha \Rightarrow \beta$ is true in *m* <u>and</u> $\beta \Rightarrow \alpha$ is true in *m*

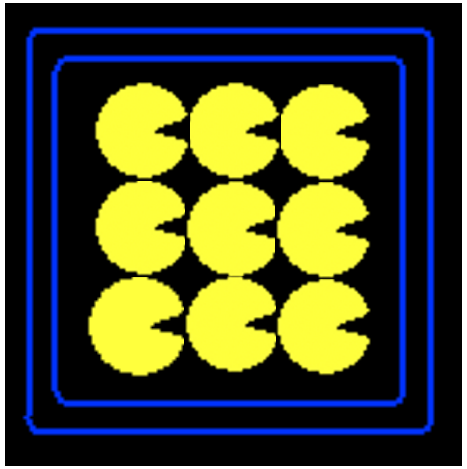
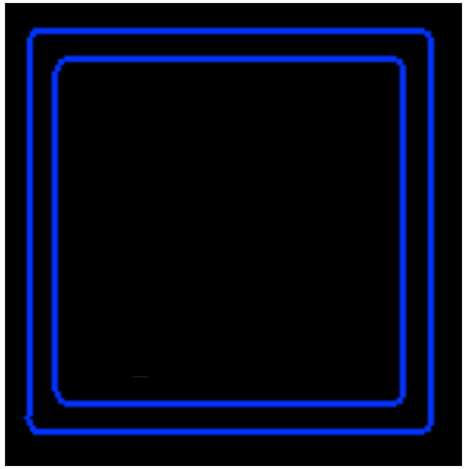
Example: Partially observable Pacman

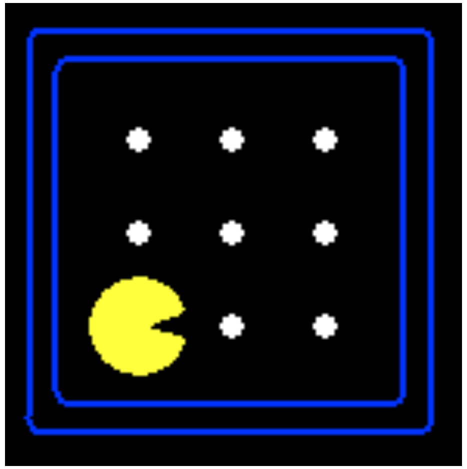
- Pacman knows the map but perceives just wall/gap to NSEW
- Formulation: ***what variables do we need?***
  - Wall locations
    - Wall_0,0   there is a wall at [0,0]
    - Wall_0,1   there is a wall at [0,1], etc. (*N* symbols for *N* locations)
  - Percepts
    - ~~Blocked_W (blocked by wall to my West) etc.~~
    - Blocked_W_0 (blocked by wall to my West ***at time 0***) etc. (4*T* symbols for *T* time steps)
  - Actions
    - W_0 (Pacman moves West at time 0), E_0 etc. (4*T* symbols)
  - Pacman's location
    - At_0,0_0 (Pacman is at [0,0] at time 0), At_0,1_0 etc. (*NT* symbols)



How many possible worlds?

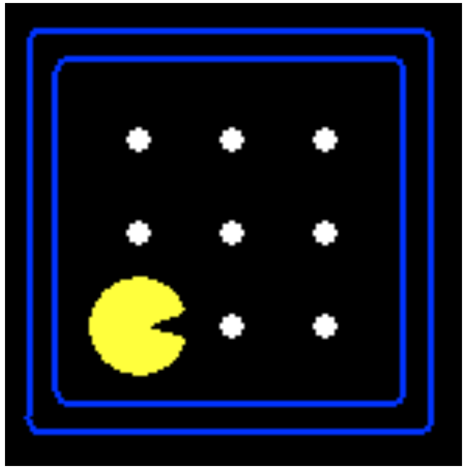
- *N* locations, *T* time steps => $N + 4T + 4T + NT = O(NT)$ variables
- $O(2^{NT})$ possible worlds!
- N=200, T=400 => ~10^{24000} worlds
- Each world is a complete "history"
  - But most of them are pretty weird!



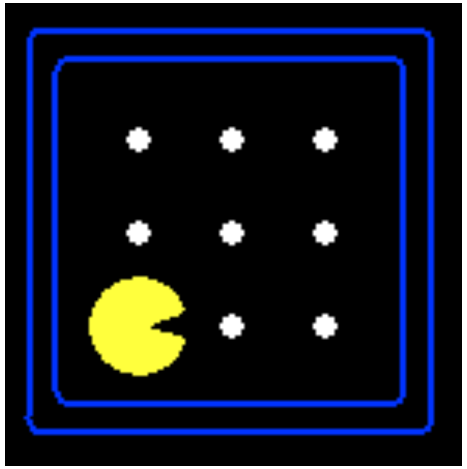
Pacman's knowledge base: Map

- Pacman knows where the walls are:
  - Wall_0,0 ∧ Wall_0,1 ∧ Wall_0,2 ∧ Wall_0,3 ∧ Wall_0,4 ∧ Wall_1,4 ∧ …
- Pacman knows where the walls aren't!
  - ¬Wall_1,1 ∧ ¬Wall_1,2 ∧ ¬Wall_1,3 ∧ ¬Wall_2,1 ∧ ¬Wall_2,2 ∧ …



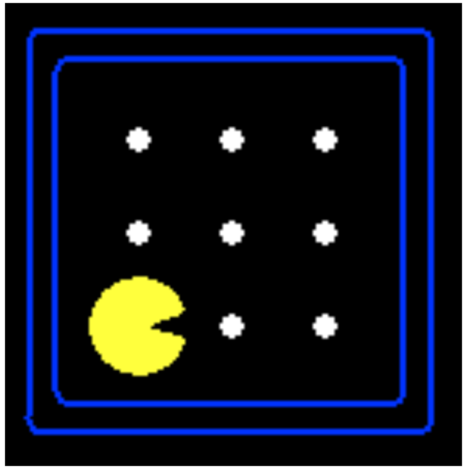
Pacman's knowledge base: Initial state

- Pacman doesn't know where he is
- But he knows he's somewhere!
  - At_1,1_0 ∨ At_1,2_0 ∨ At_1,3_0 ∨ At_2,1_0 ∨ …



Pacman's knowledge base: Sensor model

- State facts about how Pacman's percepts arise…
  - <Percept variable at t> ⇔ <some condition on world at t>
- Pacman perceives a wall to the West at time *t* ***if and only if*** he is in *x,y* and there is a wall at *x-1,y*
  - Blocked_W_0 ⇔ ((At_1,1_0 ∧ Wall_0,1) v
    (At_1,2_0 ∧ Wall_0,2) v
    (At_1,3_0 ∧ Wall_0,3) v ….)
  - 4T sentences, each of size *O*(*N*)
  - Note: these are valid for any map



Pacman's knowledge base: Transition model

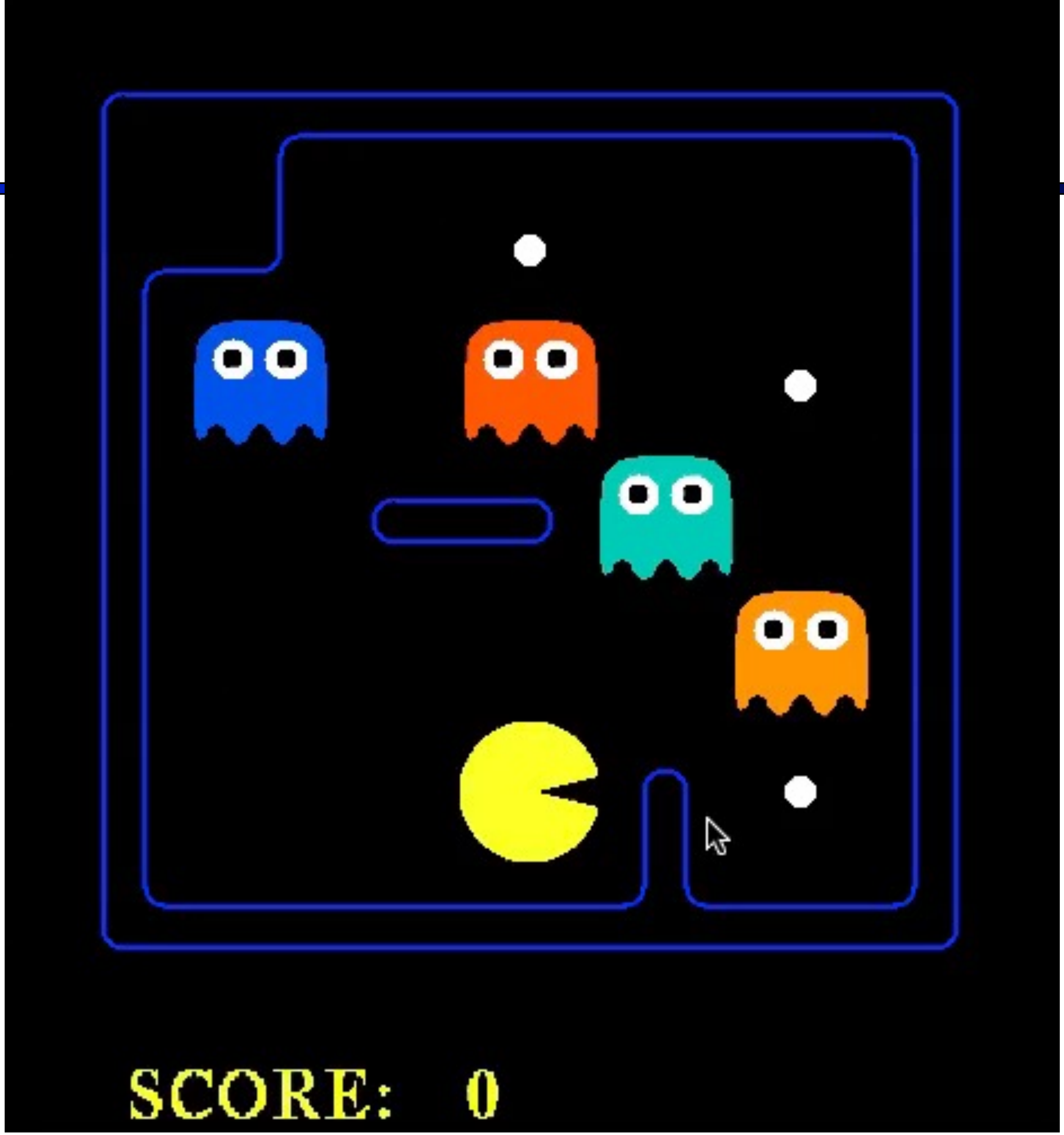
- How does each ***state variable*** at each time gets its value?
  - Here we care about location variables, e.g., At_3,3_17
- A state variable X gets its value according to a ***successor-state axiom***
  - X_t ⇔ [X_t-1 ∧ ¬(some action_t-1 made it false)] v
    [¬X_t-1 ∧ (some action_t-1 made it true)]
- For Pacman location:
  - At_3,3_17 ⇔ [At_3,3_16 ∧ ¬((¬Wall_3,4 ∧ N_16) v (¬Wall_4,3 ∧ E_16) v …)]
    v [¬At_3,3_16 ∧ ((At_3,2_16 ∧ ¬Wall_3,3 ∧ N_16) v
    (At_2,3_16 ∧ ¬Wall_3,3 ∧ N_16) v …)]

How many sentences?

- Vast majority of KB occupied by O(NT) transition model sentences
  - Each about 10 lines of text
  - N=200, T=400 => ~800,000 lines of text, or 20,000 pages
- This is because propositional logic has limited expressive power
- Are we really going to write 20,000 pages of logic sentences???
- No, but your code will generate all those sentences!
- In first-order logic, we need O(1) transition model sentences
- (State-space search uses atomic states: how do we keep the transition model representation small???)

Some reasoning tasks

- ***Localization*** with a map and local sensing:
  - Given an initial KB, plus a sequence of percepts and actions, where am I?
- ***Mapping*** with a location sensor:
  - Given an initial KB, plus a sequence of percepts and actions, what is the map?
- ***Simultaneous localization and mapping***:
  - Given …, where am I and what is the map?
- ***Planning***:
  - Given …, what action sequence is guaranteed to reach the goal?
- ***ALL OF THESE USE THE SAME KB AND THE SAME ALGORITHM!!***


SCORE: 0

Summary

- One possible agent architecture: knowledge + inference
- Logics provide a formal way to encode knowledge
  - A logic is defined by: syntax, set of possible worlds, truth condition
- A simple KB for Pacman covers the initial state, sensor model, and transition model
- Logical inference computes entailment relations among sentences, enabling a wide range of tasks to be solved