Lecture 29: Artificial Intelligence

Marvin Zhang
08/10/2016

Some slides are adapted from CS 188 (Artificial Intelligence)

Announcements

Roadmap

Introduction

Functions

Data

Mutability

Objects

Interpretation

Paradigms

Applications

Roadmap

Introduction

Functions

Data

Mutability

Objects

Interpretation

Paradigms

Applications

- This week (Applications), the goals are:

Roadmap

Introduction

Functions

Data

Mutability

Objects

Interpretation

Paradigms

Applications

- This week (Applications), the goals are:
  - To go beyond CS 61A and see examples of what comes next

Roadmap

Introduction

Functions

Data

Mutability

Objects

Interpretation

Paradigms

Applications

- This week (Applications), the goals are:
  - To go beyond CS 61A and see examples of what comes next
  - To wrap up CS 61A!

Artificial Intelligence (AI)

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:
  - Think like humans?

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:
  - Think like humans?
    - Well, we don’t really know *how* humans think

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:
  - Think like humans?
    - Well, we don’t really know *how* humans think
  - Act like humans?

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:
  - Think like humans?
    - Well, we don’t really know *how* humans think
  - Act like humans?
    - Quick, what’s 17548 * 44?

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:
  - Think like humans?
    - Well, we don’t really know *how* humans think
  - Act like humans?
    - Quick, what’s 17548 * 44?
    - Humans can often behave *irrationally*

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:
  - Think like humans?
    - Well, we don’t really know *how* humans think
  - Act like humans?
    - Quick, what’s `17548 * 44`?
    - Humans can often behave *irrationally*
  - Think rationally?

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:
  - Think like humans?
    - Well, we don't really know *how* humans think
  - Act like humans?
    - Quick, what's 17548 * 44?
    - Humans can often behave *irrationally*
  - Think rationally?
    - What we really care about, though, is behavior

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:
  - Think like humans?
    - Well, we don’t really know *how* humans think
  - Act like humans?
    - Quick, what’s 17548 * 44?
    - Humans can often behave *irrationally*
  - Think rationally?
    - What we really care about, though, is behavior
  - *Act rationally*

Artificial Intelligence (AI)

- The subfield of computer science that studies how to create programs that:
  - Think like humans?
    - Well, we don't really know *how* humans think
  - Act like humans?
    - Quick, what's 17548 * 44?
    - Humans can often behave *irrationally*
  - Think rationally?
    - What we really care about, though, is behavior
  - *Act rationally*
    - A better name for artificial intelligence would be *computational rationality*

Applications

Applications

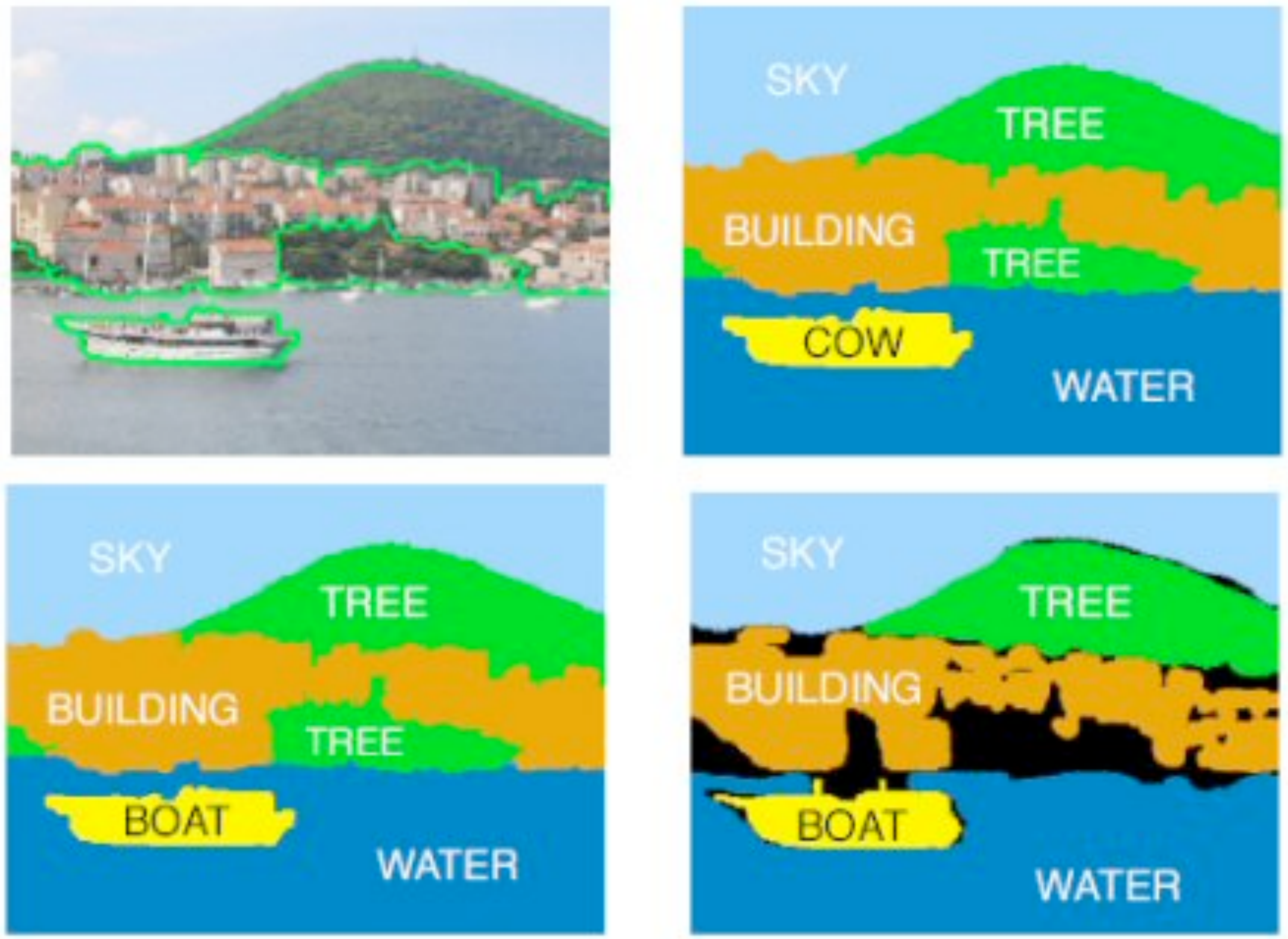
- Artificial intelligence has a wide range of applications, including examples such as:

Applications

- Artificial intelligence has a wide range of applications, including examples such as:
  - Natural language processing

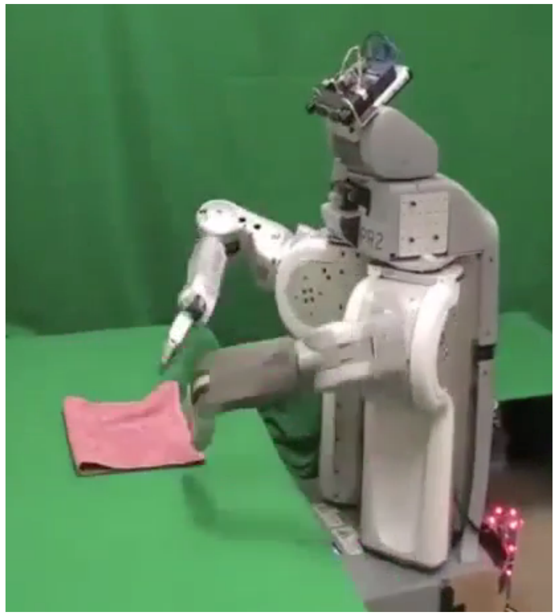
Applications

- Artificial intelligence has a wide range of applications, including examples such as:
  - Natural language processing
  - Computer vision



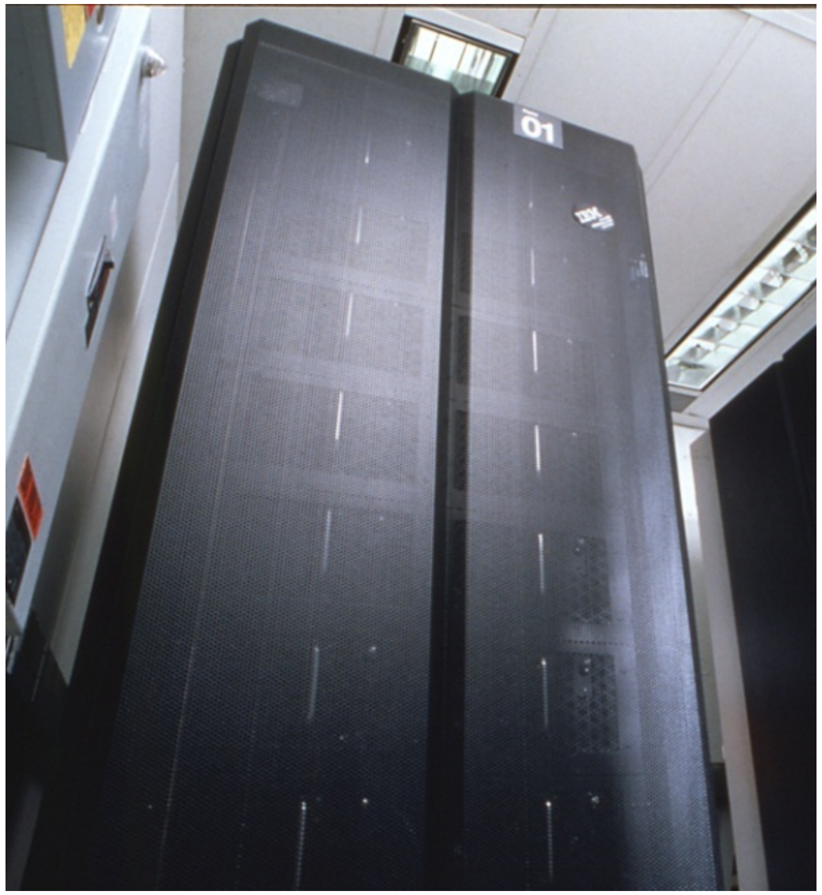
Applications

- Artificial intelligence has a wide range of applications, including examples such as:
  - Natural language processing
  - Computer vision
  - Robotics



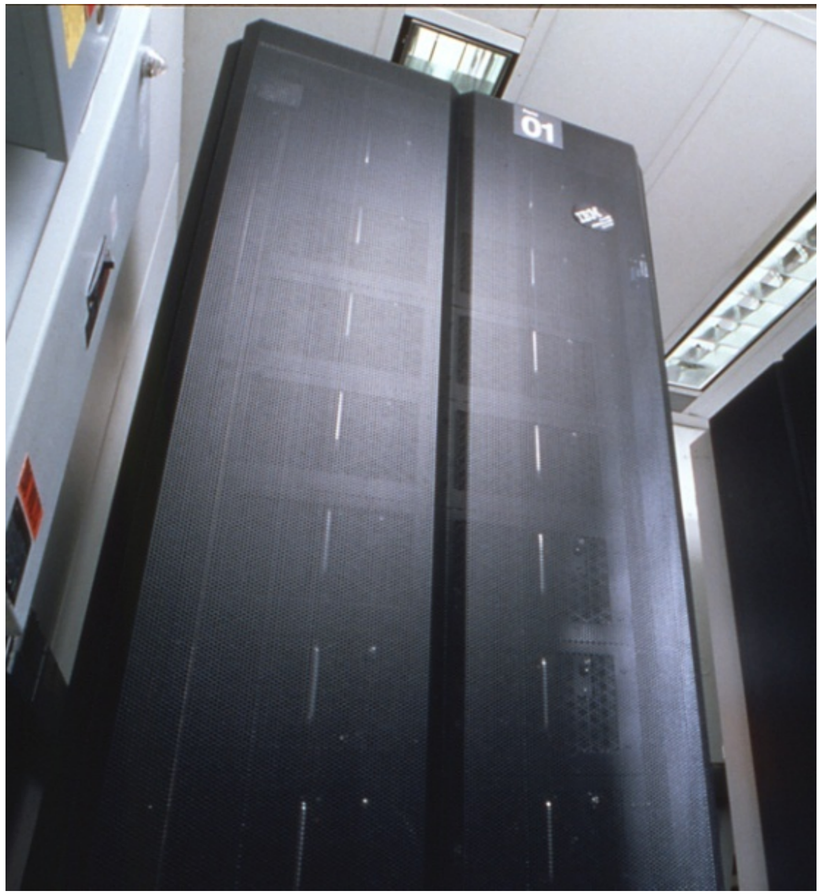
Applications

- Artificial intelligence has a wide range of applications, including examples such as:
  - Natural language processing
  - Computer vision
  - Robotics
  - Game playing



Applications

- Artificial intelligence has a wide range of applications, including examples such as:
  - Natural language processing
  - Computer vision
  - Robotics
  - Game playing



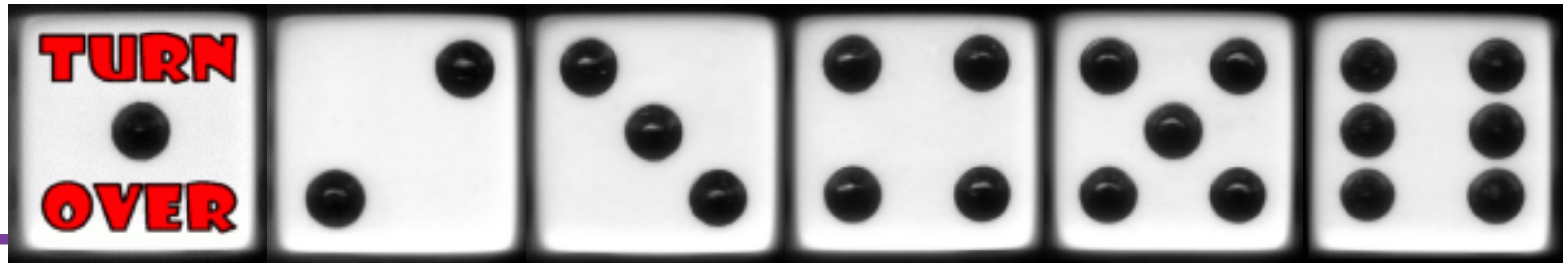
Game Playing

- Games have historically been a popular area of study in artificial intelligence, in part because they drive the study and implementation of efficient AI algorithms
  - If you're interested, two recent-ish results include playing Atari games at human expert levels and playing Go beyond top human levels
- Many breakthroughs in AI research have come from building systems that play games, including advances in:
  - Reinforcement learning (Checkers, Atari)
  - Rational meta-reasoning (Reversi/Othello)
  - Game tree search algorithms (Go)
- We will build AI systems today that play Hog and Ants!

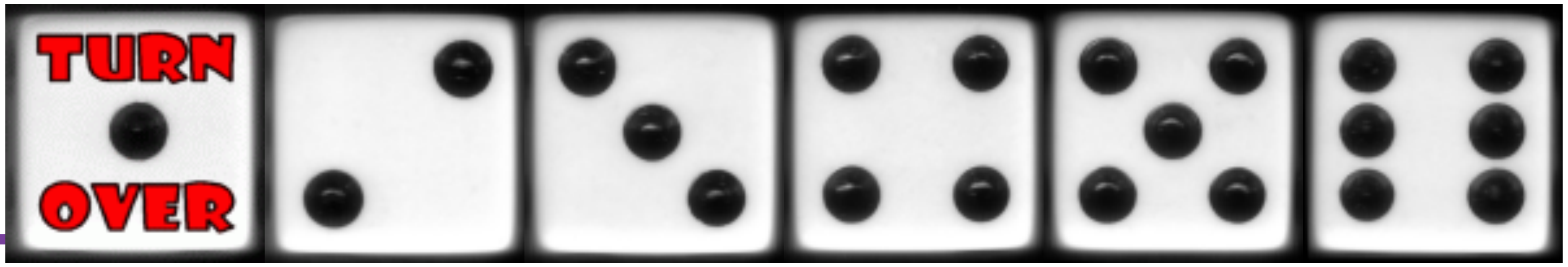
Playing Hog

Using Markov Decision Processes

Hog

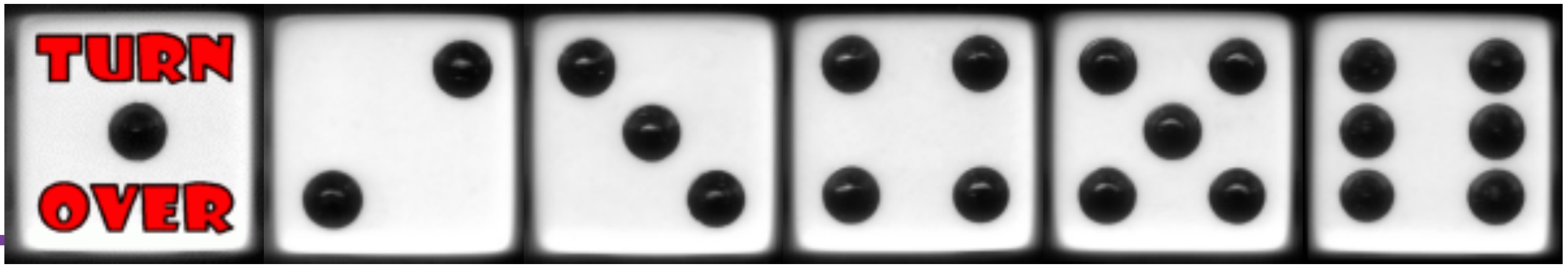


Hog



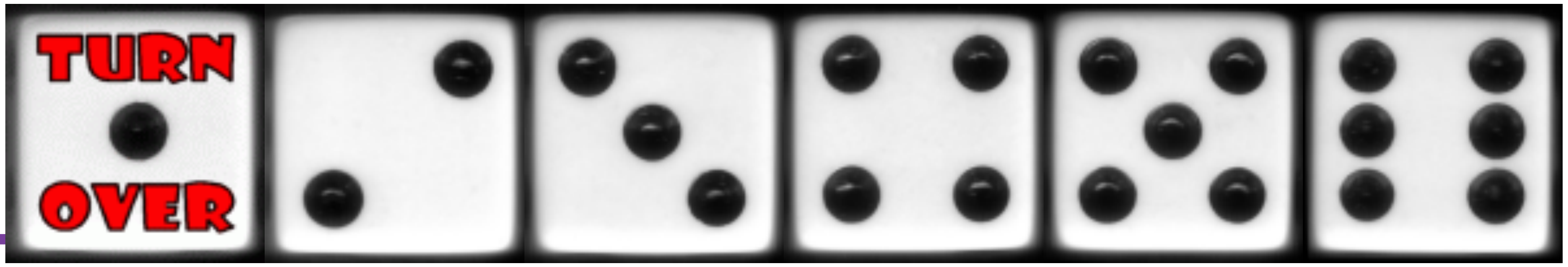
- Two player dice game

Hog



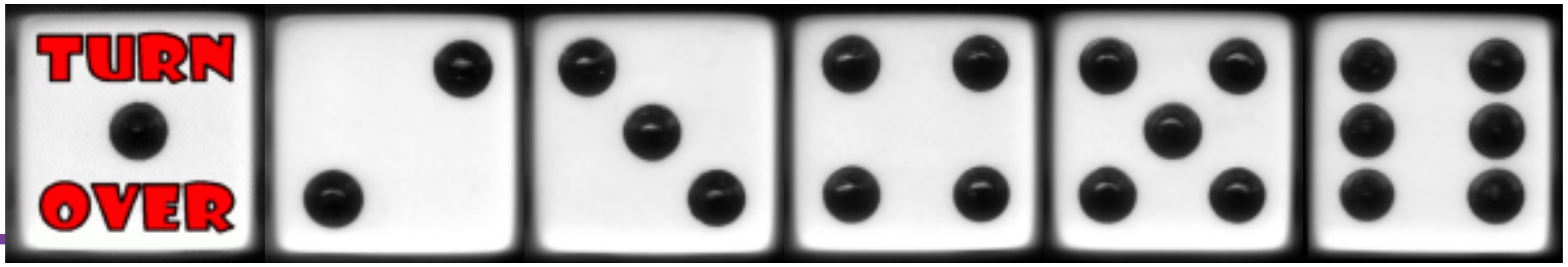
- Two player dice game
- Take turns rolling 0 to 10 dice and accumulating the sum into your overall score, until someone reaches 100

Hog



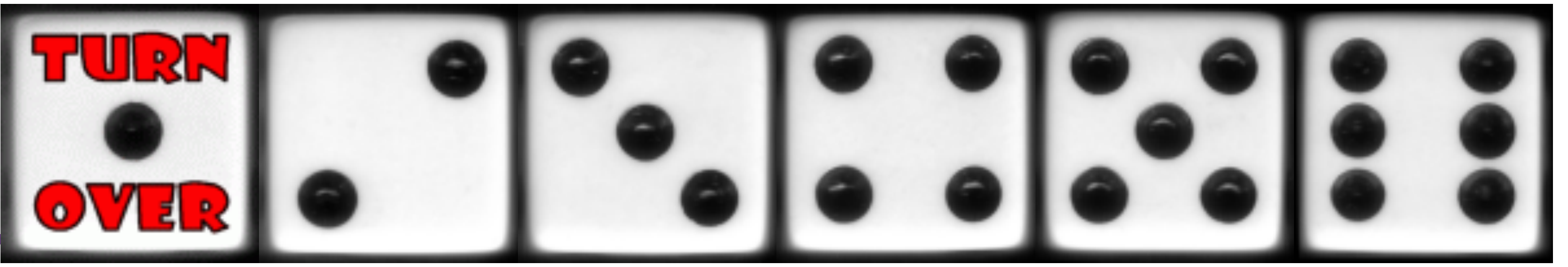
- Two player dice game
- Take turns rolling 0 to 10 dice and accumulating the sum into your overall score, until someone reaches 100
- Several special rules to keep track of:

Hog



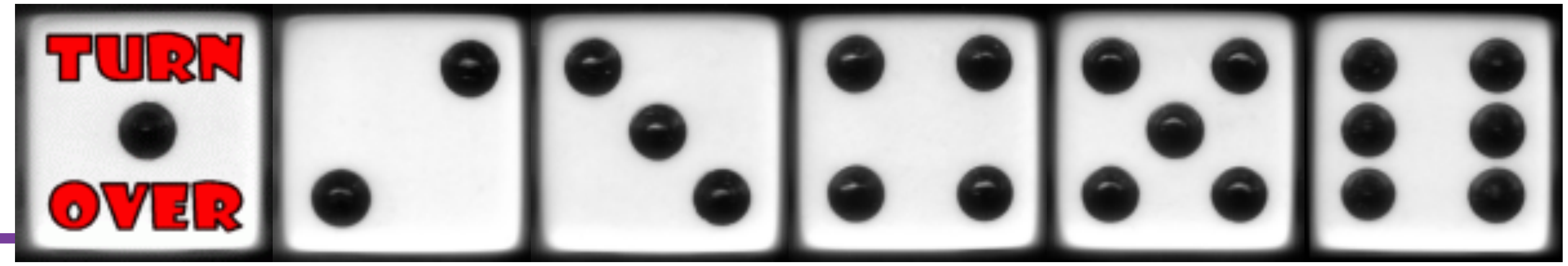
- Two player dice game
- Take turns rolling 0 to 10 dice and accumulating the sum into your overall score, until someone reaches 100
- Several special rules to keep track of:
  - Pig Out, Free Bacon, Hog Tied, Hog Wild, Hogtimus Prime

Hog



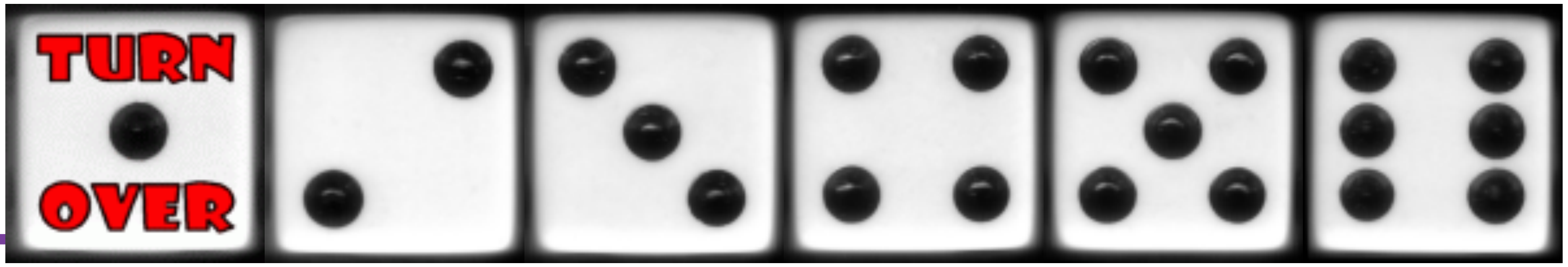
- Two player dice game
- Take turns rolling 0 to 10 dice and accumulating the sum into your overall score, until someone reaches 100
- Several special rules to keep track of:
  - Pig Out, Free Bacon, Hog Tied, Hog Wild, Hogtimus Prime
  - And the notorious Swine Swap

Hog



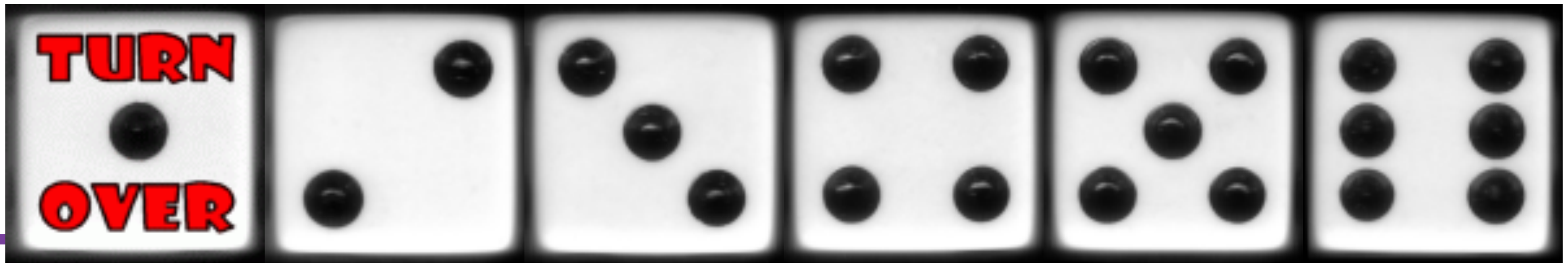
- Two player dice game
- Take turns rolling 0 to 10 dice and accumulating the sum into your overall score, until someone reaches 100
- Several special rules to keep track of:
  - Pig Out, Free Bacon, Hog Tied, Hog Wild, Hogtimus Prime
  - And the notorious Swine Swap
- In the last question of this project, you had to implement a final strategy that beats `always_roll(6)` at least 70% of the time

Hog



- Two player dice game
- Take turns rolling 0 to 10 dice and accumulating the sum into your overall score, until someone reaches 100
- Several special rules to keep track of:
  - Pig Out, Free Bacon, Hog Tied, Hog Wild, Hogtimus Prime
  - And the notorious Swine Swap
- In the last question of this project, you had to implement a final strategy that beats `always_roll(6)` at least 70% of the time
  - This is AI-like, except you (probably) hand-designed the "intelligence" into your strategy

Hog



- Two player dice game
- Take turns rolling 0 to 10 dice and accumulating the sum into your overall score, until someone reaches 100
- Several special rules to keep track of:
  - Pig Out, Free Bacon, Hog Tied, Hog Wild, Hogtimus Prime
  - And the notorious Swine Swap
- In the last question of this project, you had to implement a final strategy that beats `always_roll(6)` at least 70% of the time
  - This is AI-like, except you (probably) hand-designed the "intelligence" into your strategy
  - We can get up to ~85% win rate against `always_roll(6)`! I'll show you how, using AI techniques and algorithms

Agents and Environments

Agents and Environments

- Many, if not most, problems in AI are formalized using the concepts of an *agent* and an *environment*

Agents and Environments

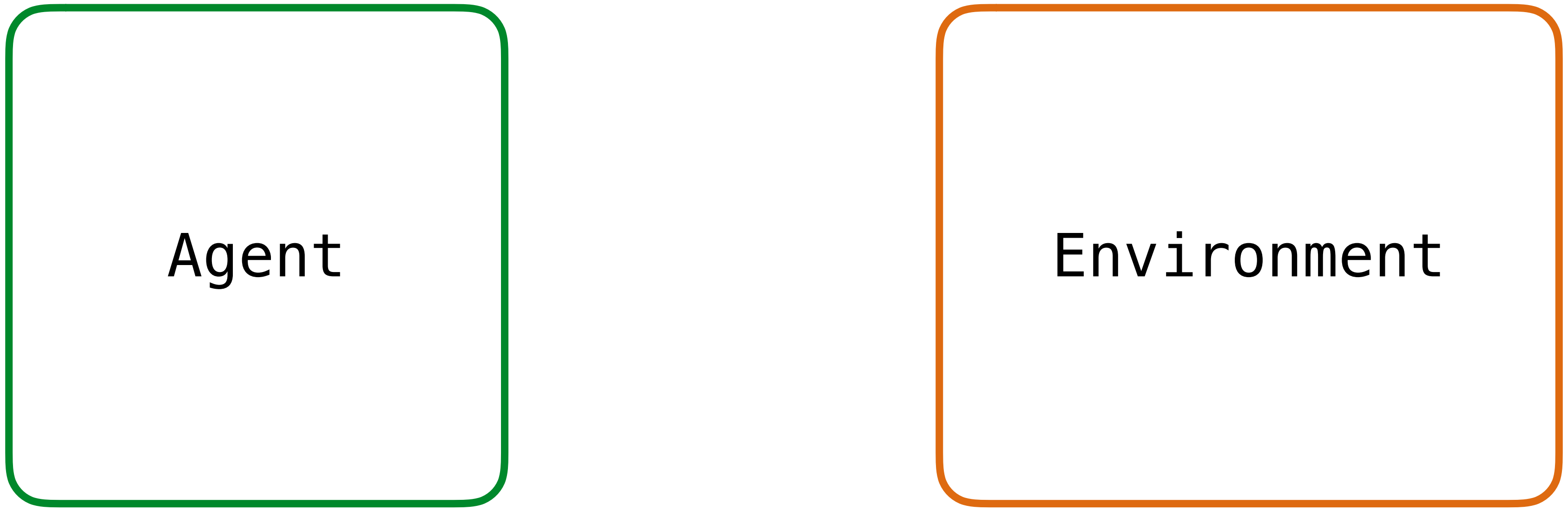
- Many, if not most, problems in AI are formalized using the concepts of an *agent* and an *environment*
- The agent *perceives* information about the environment and performs actions that may change the environment

Agents and Environments

- Many, if not most, problems in AI are formalized using the concepts of an *agent* and an *environment*
- The agent *perceives* information about the environment and performs actions that may change the environment
- This is a natural way to describe many games, robotic systems, humans, and much more

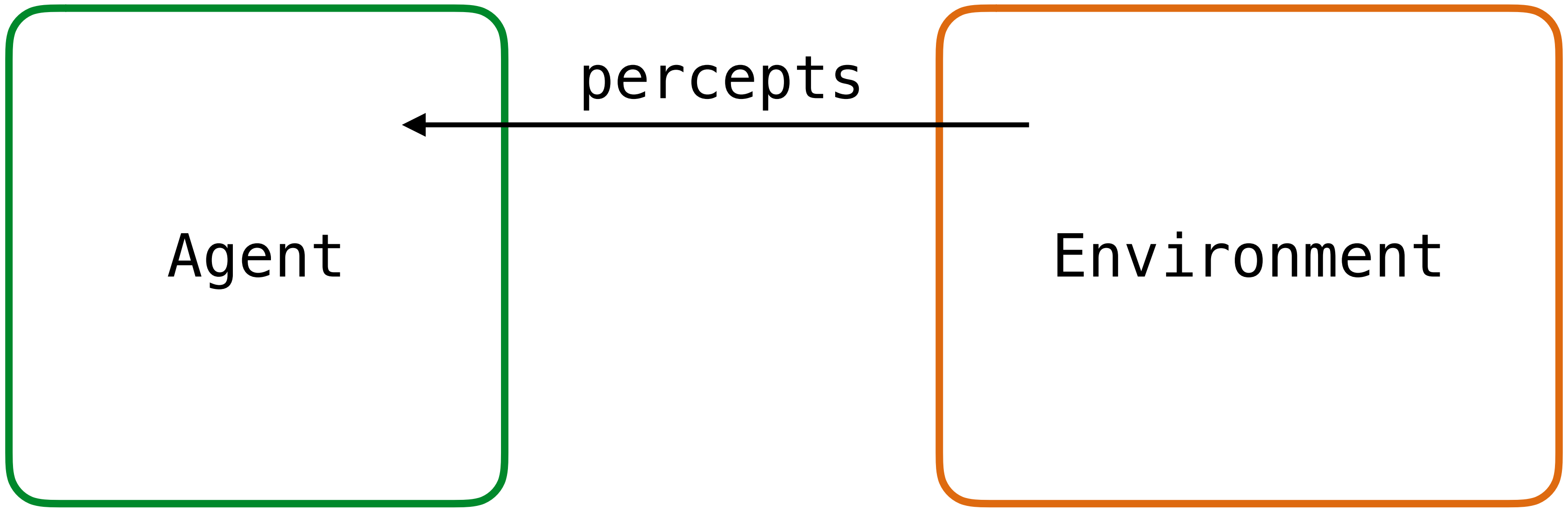
Agents and Environments

- Many, if not most, problems in AI are formalized using the concepts of an *agent* and an *environment*
- The agent *perceives* information about the environment and performs actions that may change the environment
- This is a natural way to describe many games, robotic systems, humans, and much more



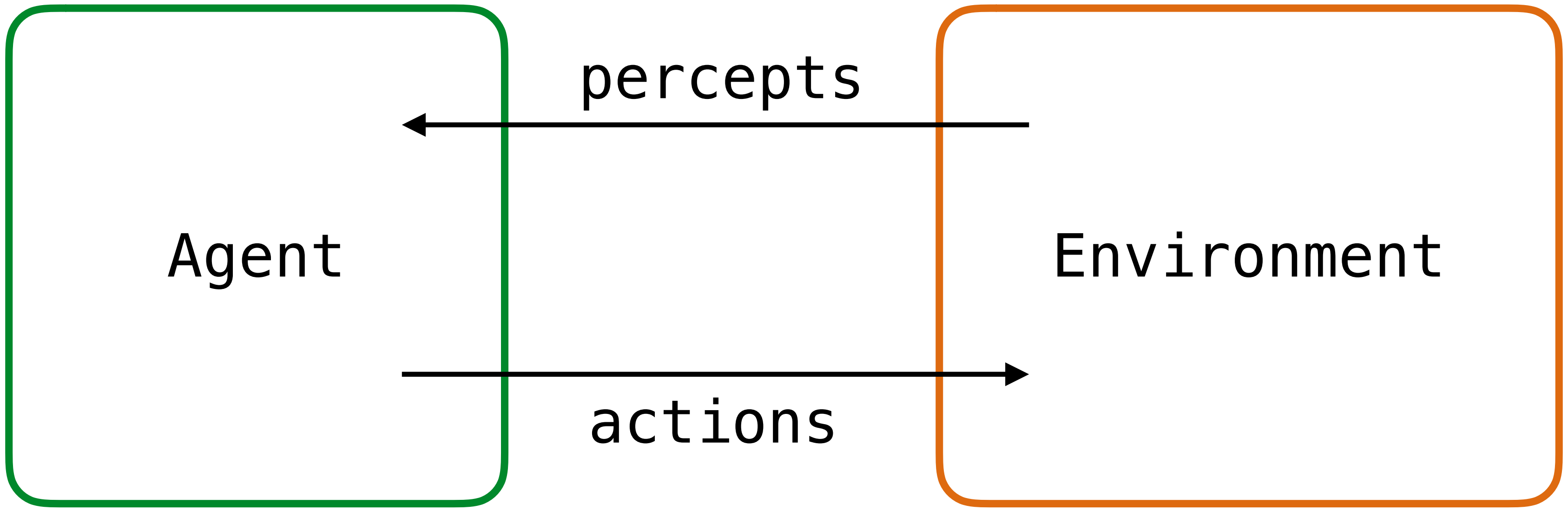
Agents and Environments

- Many, if not most, problems in AI are formalized using the concepts of an *agent* and an *environment*
- The agent *perceives* information about the environment and performs actions that may change the environment
- This is a natural way to describe many games, robotic systems, humans, and much more

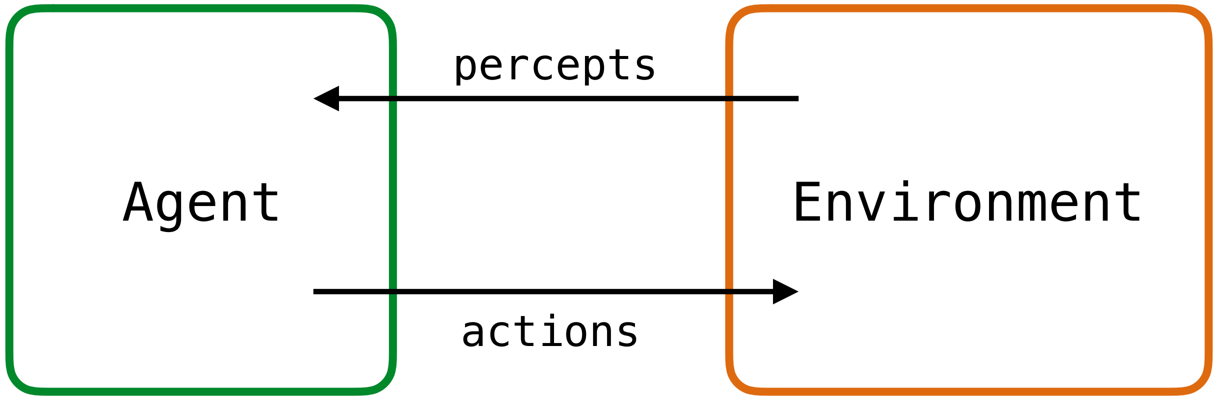


Agents and Environments

- Many, if not most, problems in AI are formalized using the concepts of an *agent* and an *environment*
- The agent *perceives* information about the environment and performs actions that may change the environment
- This is a natural way to describe many games, robotic systems, humans, and much more

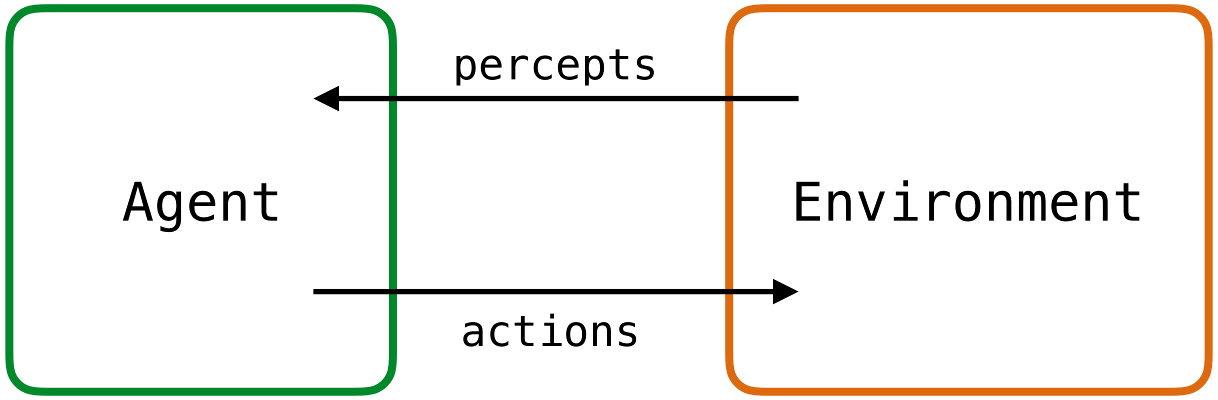


Hog Agents and Environments



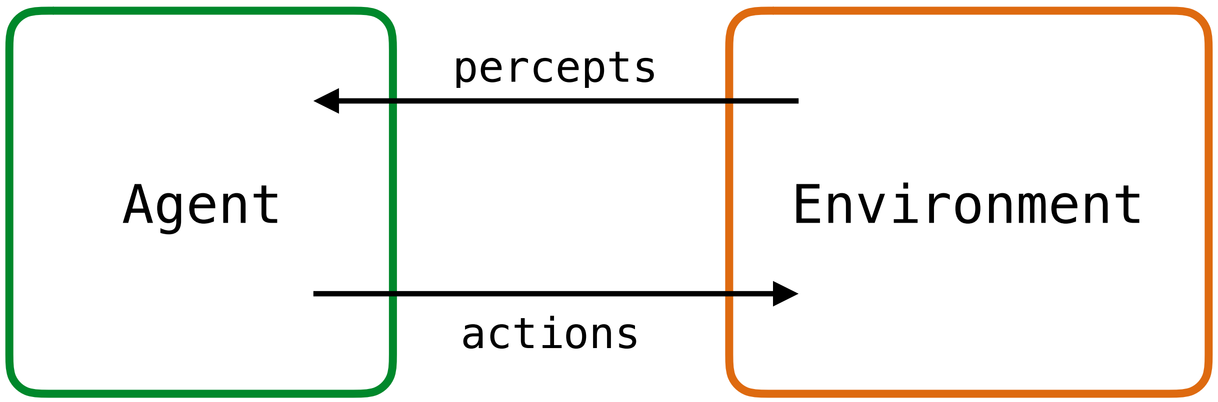
Hog Agents and Environments

- In the game of Hog, who is the agent?



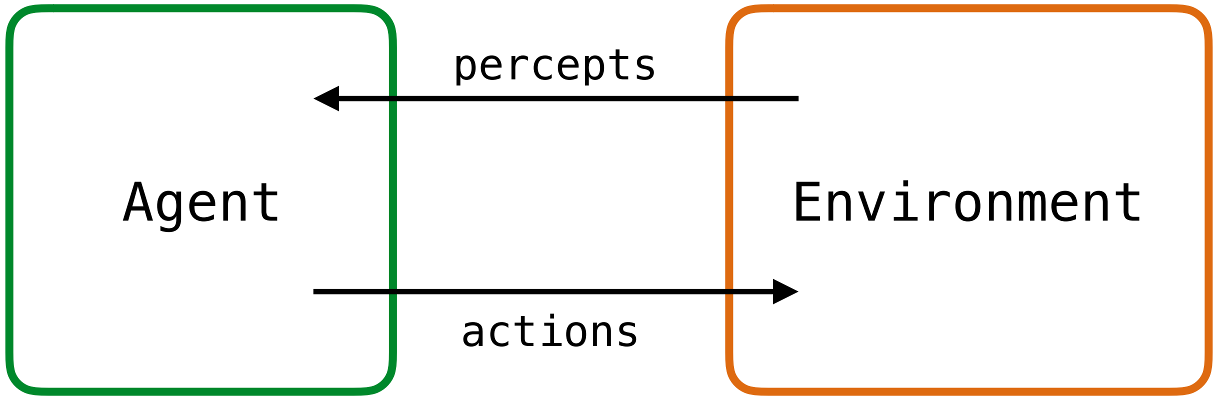
Hog Agents and Environments

- In the game of Hog, who is the agent?
  - You, or the computer



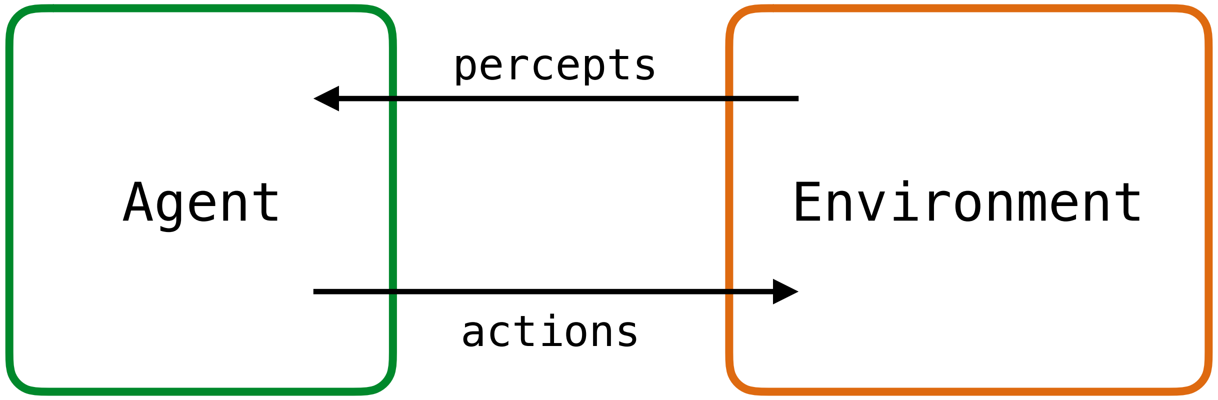
Hog Agents and Environments

- In the game of Hog, who is the agent?
  - You, or the computer
- What is the environment?



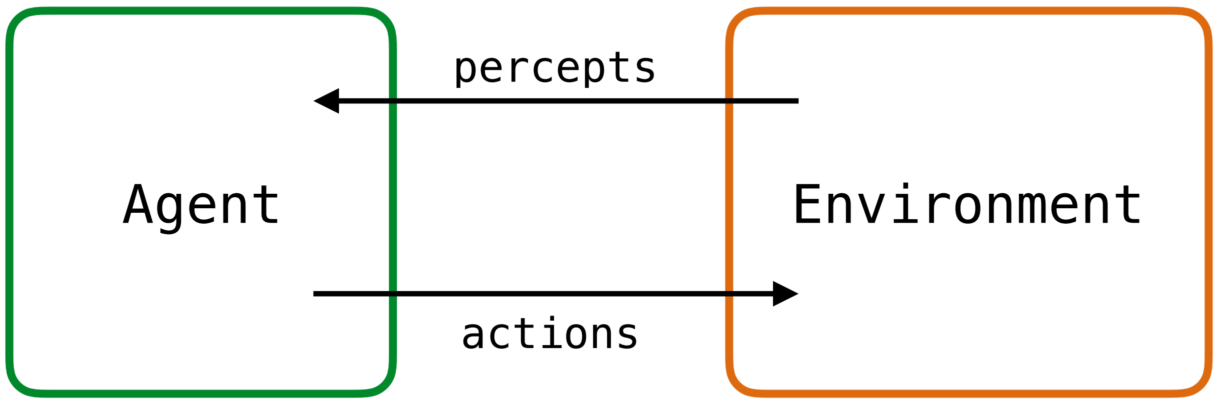
Hog Agents and Environments

- In the game of Hog, who is the agent?
  - You, or the computer
- What is the environment?
  - It's the whole game!



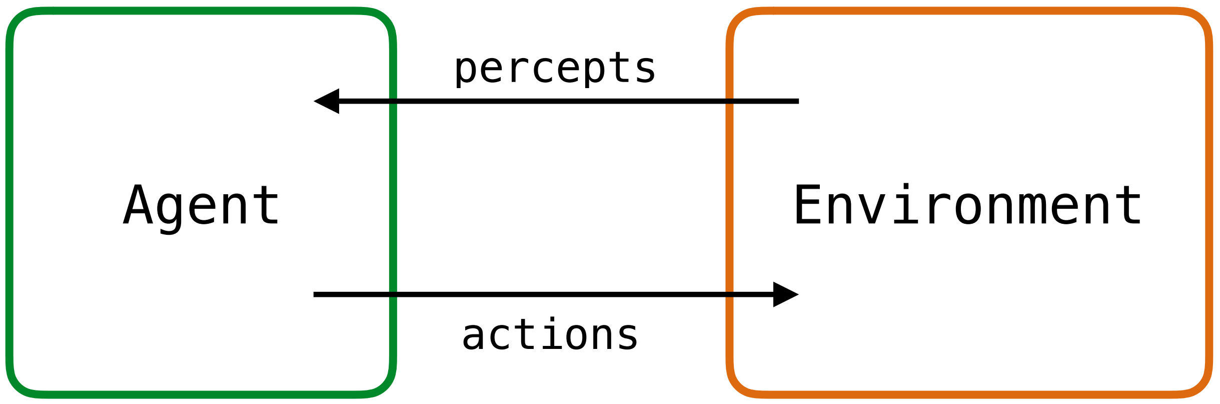
Hog Agents and Environments

- In the game of Hog, who is the agent?
  - You, or the computer
- What is the environment?
  - It’s the whole game!
    - Your opponent



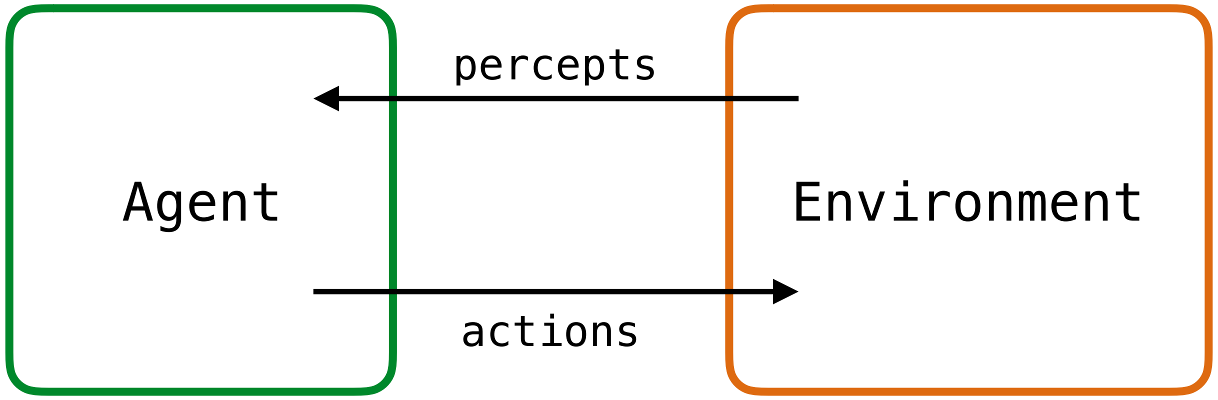
Hog Agents and Environments

- In the game of Hog, who is the agent?
  - You, or the computer
- What is the environment?
  - It’s the whole game!
    - Your opponent
      - (We are considering the opposing agent to be part of the environment, because it’s simpler this way)



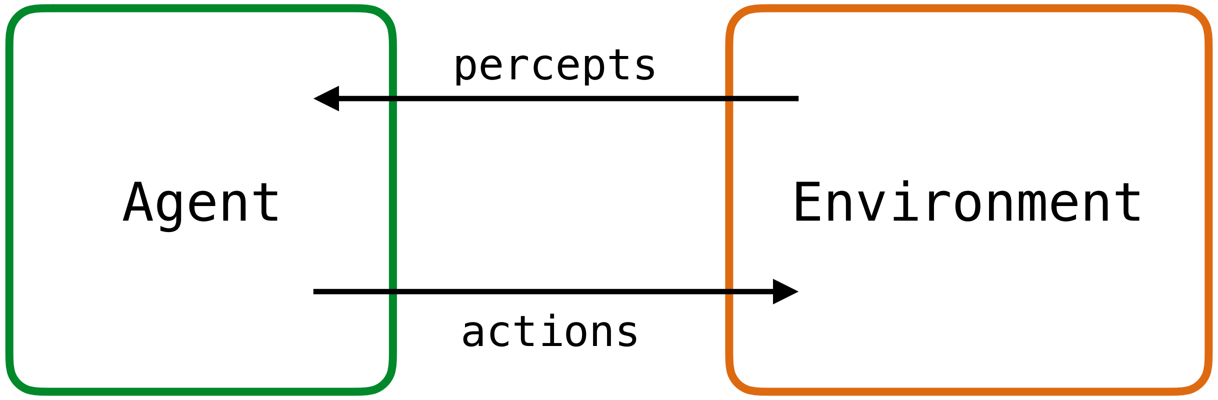
Hog Agents and Environments

- In the game of Hog, who is the agent?
  - You, or the computer
- What is the environment?
  - It’s the whole game!
    - Your opponent
      - (We are considering the opposing agent to be part of the environment, because it’s simpler this way)
    - You and your opponent’s score



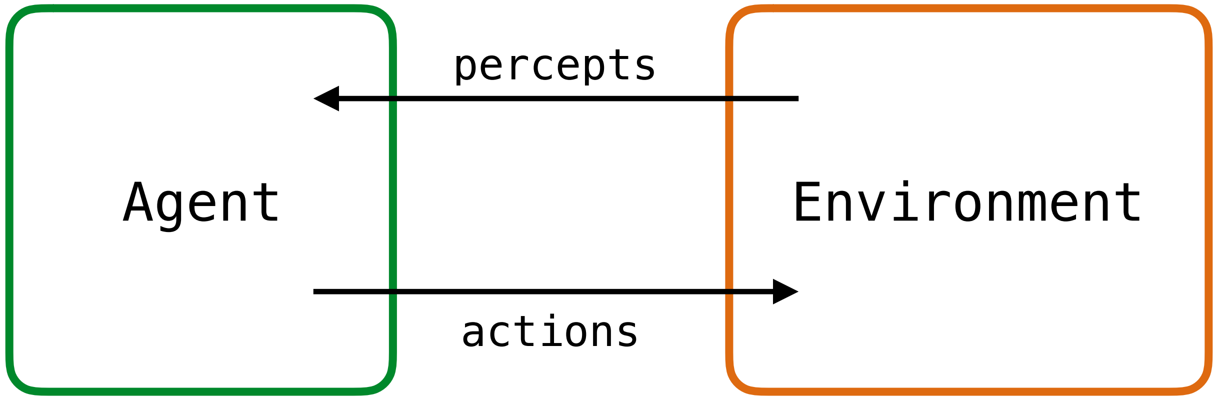
Hog Agents and Environments

- In the game of Hog, who is the agent?
  - You, or the computer
- What is the environment?
  - It's the whole game!
    - Your opponent
      - (We are considering the opposing agent to be part of the environment, because it's simpler this way)
    - You and your opponent's score
    - The rules of the game



Hog Agents and Environments

- In the game of Hog, who is the agent?
  - You, or the computer
- What is the environment?
  - It’s the whole game!
    - Your opponent
      - (We are considering the opposing agent to be part of the environment, because it’s simpler this way)
    - You and your opponent’s score
    - The rules of the game
- In AI, the problem we care about is figuring out how the agent should choose its actions, given what it perceives, so as to positively shape its environment



Markov Decision Processes

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)
- This means is that we have to specify:

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)
- This means is that we have to specify:
  - A set of *states* $\mathcal{S}$, which are the states of the environment

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)
- This means is that we have to specify:
  - A set of *states* S, which are the states of the environment
    - For Hog, we just need the two scores to represent states

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)
- This means is that we have to specify:
  - A set of *states* S, which are the states of the environment
    - For Hog, we just need the two scores to represent states
  - A set of *actions* A, which are the actions the agent can take

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)
- This means is that we have to specify:
  - A set of *states* $\mathbb{S}$, which are the states of the environment
    - For Hog, we just need the two scores to represent states
  - A set of *actions* $\mathbb{A}$, which are the actions the agent can take
    - This is how many dice the agent chooses to roll

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)
- This means is that we have to specify:
  - A set of *states* `S`, which are the states of the environment
    - For Hog, we just need the two scores to represent states
  - A set of *actions* `A`, which are the actions the agent can take
    - This is how many dice the agent chooses to roll
  - A *reward function* `R(s)`, which is the reward for each state `s`

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)
- This means is that we have to specify:
  - A set of *states* `S`, which are the states of the environment
    - For Hog, we just need the two scores to represent states
  - A set of *actions* `A`, which are the actions the agent can take
    - This is how many dice the agent chooses to roll
  - A *reward function* `R(s)`, which is the reward for each state `s`
    - We get a positive/negative reward only when we win/lose

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)
- This means is that we have to specify:
  - A set of *states* S, which are the states of the environment
    - For Hog, we just need the two scores to represent states
  - A set of *actions* A, which are the actions the agent can take
    - This is how many dice the agent chooses to roll
  - A *reward function* R(s), which is the reward for each state s
    - We get a positive/negative reward only when we win/lose
  - A *transition function* T(s, a, s′), which tells us the probability of going to state s′ starting from state s and choosing action a

Markov Decision Processes

- To do this for Hog, we will formalize our environment as a *Markov Decision Process* (MDP)
- This means is that we have to specify:
  - A set of *states* S, which are the states of the environment
    - For Hog, we just need the two scores to represent states
  - A set of *actions* A, which are the actions the agent can take
    - This is how many dice the agent chooses to roll
  - A *reward function* R(s), which is the reward for each state s
    - We get a positive/negative reward only when we win/lose
  - A *transition function* T(s, a, s′), which tells us the probability of going to state s′ starting from state s and choosing action a
    - We get this from dice probabilities and rules of the game

Policies

Policies

- Now, with our MDP, we can formalize our problem

Policies

- Now, with our MDP, we can formalize our problem
- Our agent has a *policy* π, which is a function that takes in a state and outputs the action to take for that state

Policies

- Now, with our MDP, we can formalize our problem
- Our agent has a *policy* π, which is a function that takes in a state and outputs the action to take for that state
  - The policies that the computer uses were called strategies in the project

Policies

- Now, with our MDP, we can formalize our problem
- Our agent has a *policy* π, which is a function that takes in a state and outputs the action to take for that state
  - The policies that the computer uses were called strategies in the project
- Our goal is to find the *optimal policy* π^* that maximizes the expected amount of reward the agent receives

Policies

- Now, with our MDP, we can formalize our problem
- Our agent has a *policy* π, which is a function that takes in a state and outputs the action to take for that state
  - The policies that the computer uses were called strategies in the project
- Our goal is to find the *optimal policy* π^* that maximizes the expected amount of reward the agent receives
  - In our case, this means maximizing the win rate against some fixed opponent, such as `always_roll(6)`

Policies

- Now, with our MDP, we can formalize our problem
- Our agent has a *policy* π, which is a function that takes in a state and outputs the action to take for that state
  - The policies that the computer uses were called strategies in the project
- Our goal is to find the *optimal policy* π^* that maximizes the expected amount of reward the agent receives
  - In our case, this means maximizing the win rate against some fixed opponent, such as `always_roll(6)`
- How do we find this optimal policy? The reward function gives us very little information because it is 0 except for winning and losing states

Policies

- Now, with our MDP, we can formalize our problem
- Our agent has a *policy* π, which is a function that takes in a state and outputs the action to take for that state
  - The policies that the computer uses were called strategies in the project
- Our goal is to find the *optimal policy* π^* that maximizes the expected amount of reward the agent receives
  - In our case, this means maximizing the win rate against some fixed opponent, such as `always_roll(6)`
- How do we find this optimal policy? The reward function gives us very little information because it is 0 except for winning and losing states
  - We need something that will tell us about which states are more or less likely to win from

Value Functions

Value Functions

- Reward function: $R(s)$ = reward of being in state s

Value Functions

- Reward function: $R(s)$ = reward of being in state s
- *Value function*: $V(s)$ = value of being in state s

Value Functions

- Reward function: `R(s)` = reward of being in state `s`
- *Value function*: `V(s)` = value of being in state `s`
- The value is the *long-term expected reward*

Value Functions

- Reward function: $R(s)$ = reward of being in state s
- *Value function*: $V(s)$ = value of being in state s
- The value is the *long-term expected reward*
- How do we determine the value of a state? With recursion!

Value Functions

- Reward function: `R(s)` = reward of being in state `s`
- *Value function*: `V(s)` = value of being in state `s`
- The value is the *long-term expected reward*
- How do we determine the value of a state? With recursion!
  - The value of a state is the reward of the state plus the value of the state we end up in next.

Value Functions

- Reward function: `R(s)` = reward of being in state `s`
- *Value function*: `V(s)` = value of being in state `s`
- The value is the *long-term expected reward*
- How do we determine the value of a state? With recursion!
  - The value of a state is the reward of the state plus the value of the state we end up in next.

$$V(s) = R(s) + \max_a \sum_{s'} T(s, a, s') V(s')$$

Value Functions

- Reward function: `R(s)` = reward of being in state `s`
- *Value function*: `V(s)` = value of being in state `s`
- The value is the *long-term expected reward*
- How do we determine the value of a state? With recursion!
  - The value of a state is the reward of the state plus the value of the state we end up in next.

$$\boxed{V(s)} = R(s) + \max_a \sum_{s'} T(s, a, s')V(s')$$

Value Functions

- Reward function: R(s) = reward of being in state s
- *Value function*: V(s) = value of being in state s
- The value is the *long-term expected reward*
- How do we determine the value of a state? With recursion!
  - The value of a state is the reward of the state plus the value of the state we end up in next.

$$\boxed{V(s)} = \boxed{R(s)} + \max_a \sum_{s'} T(s,a,s') V(s')$$

Value Functions

- Reward function: R(s) = reward of being in state s
- *Value function*: V(s) = value of being in state s
- The value is the *long-term expected reward*
- How do we determine the value of a state? With recursion!
  - The value of a state is the reward of the state plus the value of the state we end up in next.

$$\boxed{V(s)} = \boxed{R(s)} + \max_a \sum_{s'} T(s,a,s') \boxed{V(s')}$$

Value Functions

- Reward function: R(s) = reward of being in state s
- *Value function*: V(s) = value of being in state s
- The value is the *long-term expected reward*
- How do we determine the value of a state? With recursion!
  - The value of a state is the reward of the state plus the value of the state we end up in next.

$$\boxed{V(s)} = \boxed{R(s)} + \max_a \sum_{s'} T(s,a,s') \boxed{V(s')}$$

- We take a maximum over all possible actions because we want to find the value for the optimal policy

Value Functions

- Reward function: R(s) = reward of being in state s
- *Value function*: V(s) = value of being in state s
- The value is the *long-term expected reward*
- How do we determine the value of a state? With recursion!
  - The value of a state is the reward of the state plus the value of the state we end up in next.

$$\boxed{V(s)} = \boxed{R(s)} + \max_a \sum_{s'} T(s, a, s') \boxed{V(s')}$$

- We take a maximum over all possible actions because we want to find the value for the optimal policy
- We use a summation and T(s, a, s') because there may be several different states we could end up in

Value Iteration

Value Iteration

- We may have to compute $V(s)$ multiple times in order to get it right, because the value of later states s' can change and this can affect the value of s

Value Iteration

- We may have to compute $V(s)$ multiple times in order to get it right, because the value of later states s' can change and this can affect the value of s
- This leads us to an algorithm known as *value iteration*:

Value Iteration

- We may have to compute $V(s)$ multiple times in order to get it right, because the value of later states s' can change and this can affect the value of s
- This leads us to an algorithm known as *value iteration*:
- Repeat:

Value Iteration

- We may have to compute $V(s)$ multiple times in order to get it right, because the value of later states s' can change and this can affect the value of s
- This leads us to an algorithm known as *value iteration*:
- Repeat:
    - For all states s, determine $V(s)$

Value Iteration

- We may have to compute $V(s)$ multiple times in order to get it right, because the value of later states s' can change and this can affect the value of s
- This leads us to an algorithm known as *value iteration*:
- Repeat:
    - For all states s, determine $V(s)$

$$V(s) = R(s) + \max_a \sum_{s'} T(s, a, s')V(s')$$

Value Iteration

- We may have to compute $V(s)$ multiple times in order to get it right, because the value of later states s' can change and this can affect the value of s
- This leads us to an algorithm known as *value iteration*:
- Repeat:
    - For all states s, determine $V(s)$

$$V(s) = R(s) + \max_a \sum_{s'} T(s, a, s')V(s')$$

    - If V doesn't change, return the policy π that, given a state s, chooses the action a that maximizes the expected value of the next state s'

Value Iteration

- We may have to compute v(s) multiple times in order to get it right, because the value of later states s′ can change and this can affect the value of s
- This leads us to an algorithm known as *value iteration*:
- Repeat:
  - For all states s, determine v(s)

$$V(s) = R(s) + \max_a \sum_{s'} T(s,a,s')V(s')$$

  - If v doesn't change, return the policy π that, given a state s, chooses the action a that maximizes the expected value of the next state s′

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')V(s')$$

Value Iteration

- We may have to compute V(s) multiple times in order to get it right, because the value of later states s′ can change and this can affect the value of s
- This leads us to an algorithm known as *value iteration*:
- Repeat:
  - For all states s, determine V(s)

$$V(s) = R(s) + \max_a \sum_{s'} T(s, a, s') V(s')$$

  - If V doesn't change, return the policy π that, given a state s, chooses the action a that maximizes the expected value of the next state s′

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s') V(s')$$

- We can show that this policy is optimal, under the correct assumptions! But let's not do the math

Algorithms for MDPs

Algorithms for MDPs

- We now have an algorithm that will find us the optimal policy for playing against `always_roll(6)`!

Algorithms for MDPs

- We now have an algorithm that will find us the optimal policy for playing against `always_roll(6)`!
  - It also does quite well against other opponents

Algorithms for MDPs

- We now have an algorithm that will find us the optimal policy for playing against `always_roll(6)`!
  - It also does quite well against other opponents
- This algorithm, value iteration, is just a special case of a family of algorithms for solving MDPs by alternating between two steps:

Algorithms for MDPs

- We now have an algorithm that will find us the optimal policy for playing against `always_roll(6)`!
  - It also does quite well against other opponents
- This algorithm, value iteration, is just a special case of a family of algorithms for solving MDPs by alternating between two steps:
  - *Policy evaluation:* Determine the value of each state `s`, but using the current policy rather than the optimal

Algorithms for MDPs

- We now have an algorithm that will find us the optimal policy for playing against `always_roll(6)`!
  - It also does quite well against other opponents
- This algorithm, value iteration, is just a special case of a family of algorithms for solving MDPs by alternating between two steps:
  - *Policy evaluation:* Determine the value of each state `s`, but using the current policy rather than the optimal
  - *Policy iteration:* Improve the current policy to a new policy using the value function found in the first step

Algorithms for MDPs

- We now have an algorithm that will find us the optimal policy for playing against `always_roll(6)`!
  - It also does quite well against other opponents
- This algorithm, value iteration, is just a special case of a family of algorithms for solving MDPs by alternating between two steps:
  - *Policy evaluation:* Determine the value of each state `s`, but using the current policy rather than the optimal
  - *Policy iteration:* Improve the current policy to a new policy using the value function found in the first step
  - Value iteration combines these two steps into one!

Algorithms for MDPs

- We now have an algorithm that will find us the optimal policy for playing against `always_roll(6)`!
  - It also does quite well against other opponents
- This algorithm, value iteration, is just a special case of a family of algorithms for solving MDPs by alternating between two steps:
  - *Policy evaluation:* Determine the value of each state `s`, but using the current policy rather than the optimal
  - *Policy iteration:* Improve the current policy to a new policy using the value function found in the first step
  - Value iteration combines these two steps into one!
- Let's see the optimal policy in action

Algorithms for MDPs (demo)

- We now have an algorithm that will find us the optimal policy for playing against `always_roll(6)`!
  - It also does quite well against other opponents
- This algorithm, value iteration, is just a special case of a family of algorithms for solving MDPs by alternating between two steps:
  - *Policy evaluation:* Determine the value of each state `s`, but using the current policy rather than the optimal
  - *Policy iteration:* Improve the current policy to a new policy using the value function found in the first step
  - Value iteration combines these two steps into one!
- Let's see the optimal policy in action

Playing Ants

Using rollout-based methods

Reinforcement Learning (RL)

Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function $R(s)$ or transition function $T(s, a, s')$

Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function $R(s)$ or transition function $T(s, a, s')$
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone

Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function $R(s)$ or transition function $T(s, a, s')$
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone

Reinforcement Learning (RL)

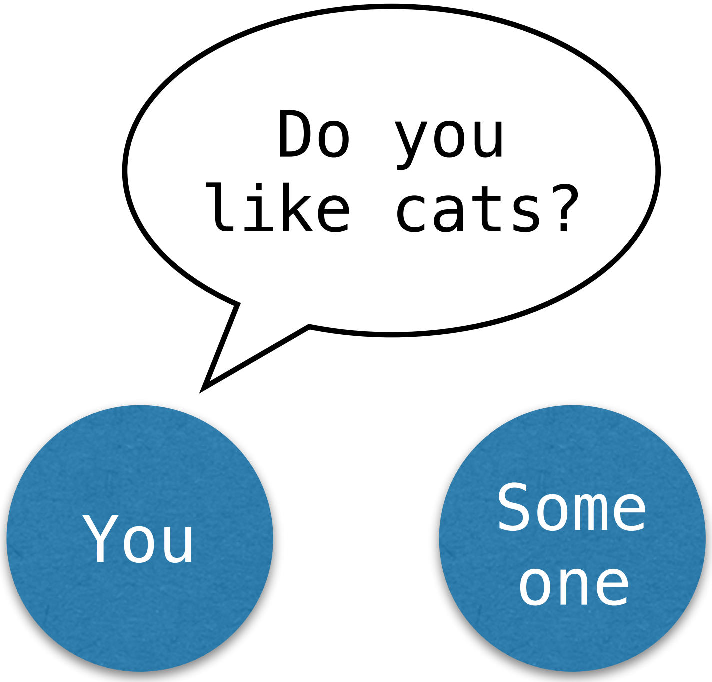
- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function $R(s)$ or transition function $T(s, a, s')$
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well

Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don’t know our reward function $R(s)$ or transition function $T(s, a, s')$
- This is very much like the real world, and here’s an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don’t know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds

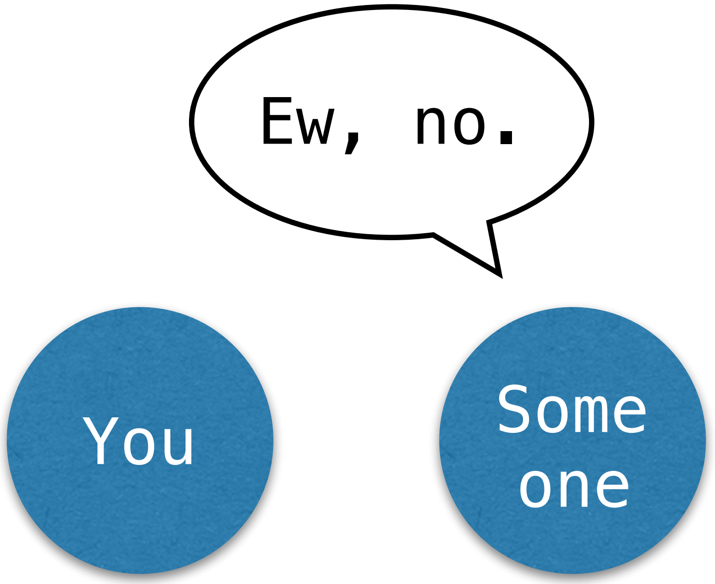
Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function R(s) or transition function T(s, a, s')
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds



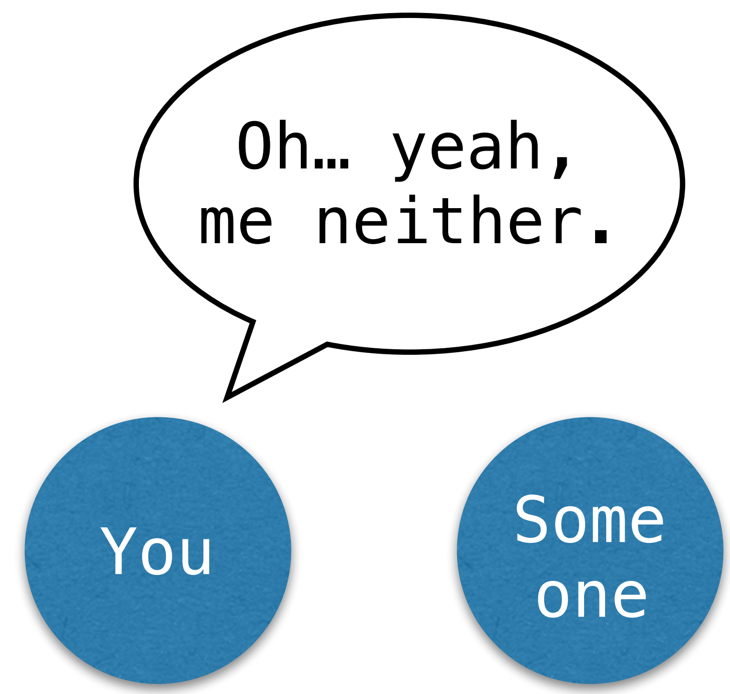
Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function `R(s)` or transition function `T(s, a, s')`
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds



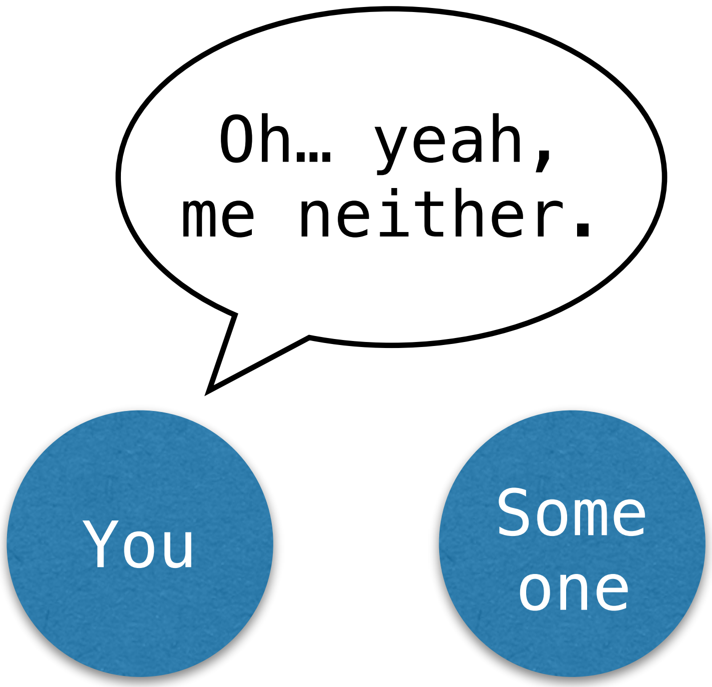
Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function `R(s)` or transition function `T(s, a, s')`
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds



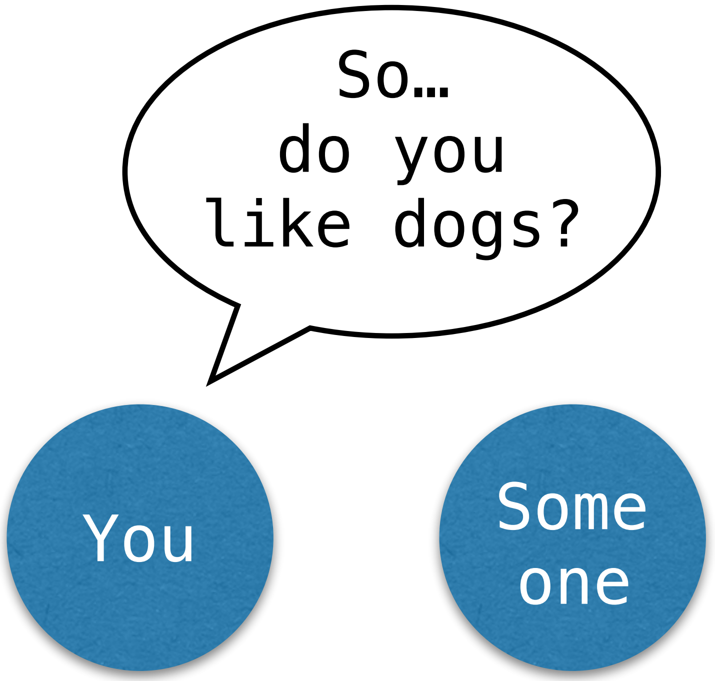
Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function `R(s)` or transition function `T(s, a, s')`
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds
- As the date goes on, you slowly figure out how you should act based on what you've tried so far, and how it went



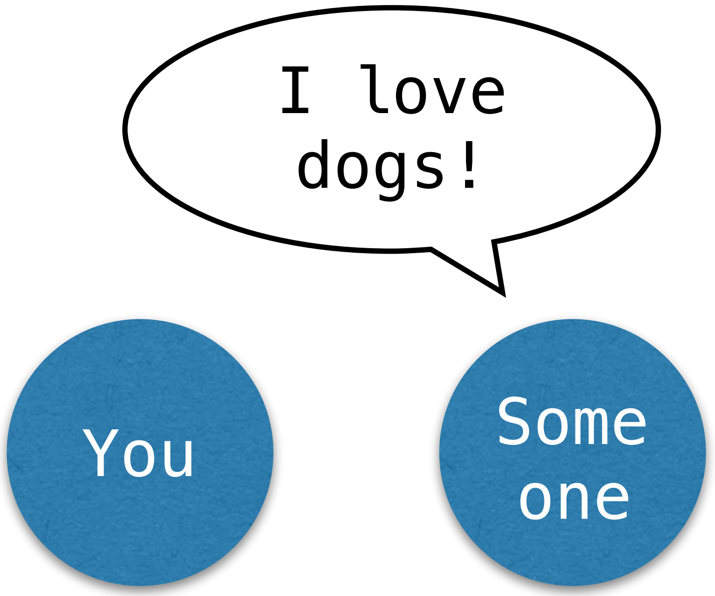
Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function `R(s)` or transition function `T(s, a, s')`
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds
- As the date goes on, you slowly figure out how you should act based on what you've tried so far, and how it went



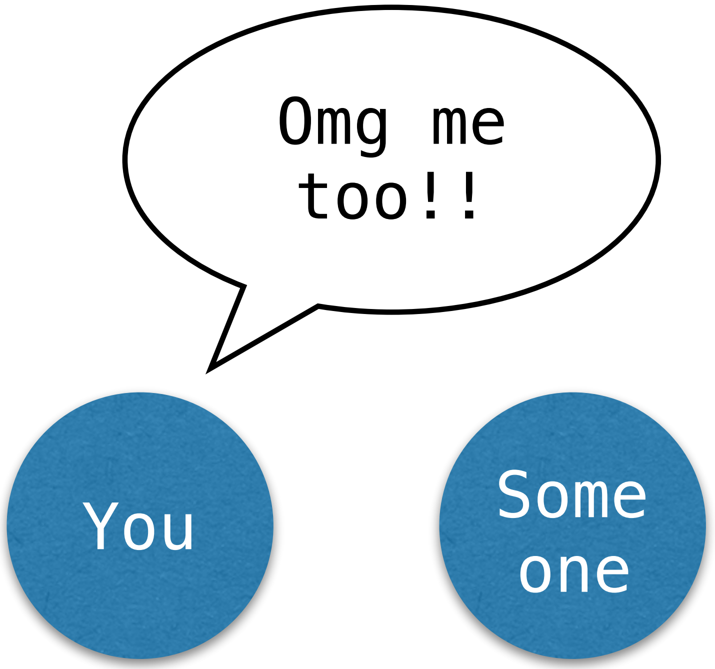
Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function `R(s)` or transition function `T(s, a, s')`
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds
- As the date goes on, you slowly figure out how you should act based on what you've tried so far, and how it went



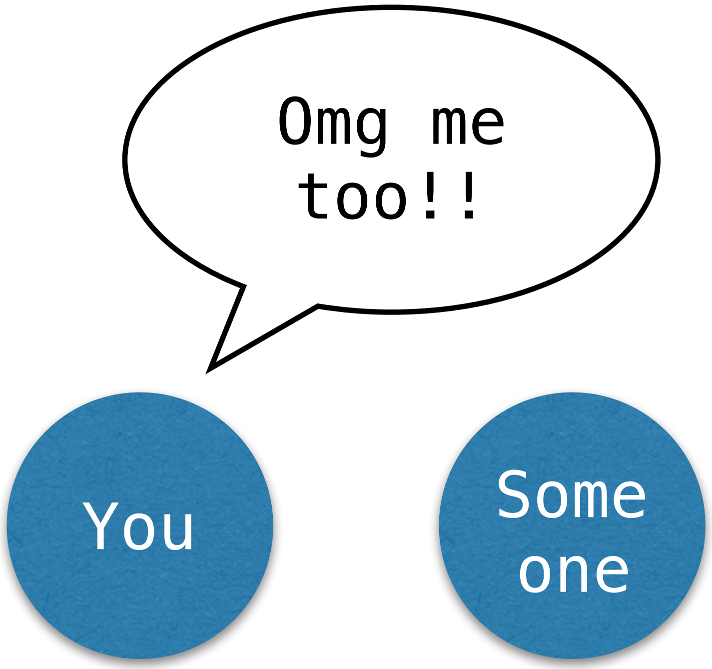
Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function $R(s)$ or transition function $T(s, a, s')$
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds
- As the date goes on, you slowly figure out how you should act based on what you've tried so far, and how it went



Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function `R(s)` or transition function `T(s, a, s')`
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds
- As the date goes on, you slowly figure out how you should act based on what you've tried so far, and how it went
- With some luck, and the right algorithm, you may learn how to act optimally!



Reinforcement Learning (RL)

- In the *reinforcement learning* setting, we still model our environment as an MDP, except now we don't know our reward function `R(s)` or transition function `T(s, a, s')`
- This is very much like the real world, and here's an analogy: suppose you go on a date with someone
- You are the agent, the other person and the setting are the environment, and you don't know the environment that well
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds
- As the date goes on, you slowly figure out how you should act based on what you've tried so far, and how it went
- With some luck, and the right algorithm, you may learn how to act optimally!

RL Algorithms

RL Algorithms

- Algorithms for reinforcement learning must solve a more general problem than algorithms like value iteration, because we don’t know how our environment works

RL Algorithms

- Algorithms for reinforcement learning must solve a more general problem than algorithms like value iteration, because we don’t know how our environment works
- We have to make sure to try different actions to determine which ones work well in our environment

RL Algorithms

- Algorithms for reinforcement learning must solve a more general problem than algorithms like value iteration, because we don’t know how our environment works
- We have to make sure to try different actions to determine which ones work well in our environment
  - This is called *exploration*

RL Algorithms

- Algorithms for reinforcement learning must solve a more general problem than algorithms like value iteration, because we don’t know how our environment works
- We have to make sure to try different actions to determine which ones work well in our environment
  - This is called *exploration*
- However, we also want to make sure to use actions that we have already found to be good

RL Algorithms

- Algorithms for reinforcement learning must solve a more general problem than algorithms like value iteration, because we don't know how our environment works
- We have to make sure to try different actions to determine which ones work well in our environment
  - This is called *exploration*
- However, we also want to make sure to use actions that we have already found to be good
  - This is called *exploitation*

RL Algorithms

- Algorithms for reinforcement learning must solve a more general problem than algorithms like value iteration, because we don’t know how our environment works
- We have to make sure to try different actions to determine which ones work well in our environment
  - This is called *exploration*
- However, we also want to make sure to use actions that we have already found to be good
  - This is called *exploitation*
- Balancing exploration and exploitation is a key problem that RL algorithms must address, and there are many different ways to handle this

RL for Ants

RL for Ants

- It’s a little weird to use MDPs and RL for Ants. Why?

RL for Ants

- It’s a little weird to use MDPs and RL for Ants. Why?
  - Everything is *deterministic*

RL for Ants

- It’s a little weird to use MDPs and RL for Ants. Why?
  - Everything is *deterministic*
  - This means that we don’t need a transition function, and we actually do know how our environment works

RL for Ants

- It’s a little weird to use MDPs and RL for Ants. Why?
  - Everything is *deterministic*
  - This means that we don’t need a transition function, and we actually do know how our environment works
- However, the state space for Ants is very, very large

RL for Ants

- It’s a little weird to use MDPs and RL for Ants. Why?
  - Everything is *deterministic*
  - This means that we don’t need a transition function, and we actually do know how our environment works
- However, the state space for Ants is very, very large
  - So even though we could specify how our environment works, it is very difficult to code it and for our program to utilize all of this information

RL for Ants

- It’s a little weird to use MDPs and RL for Ants. Why?
  - Everything is *deterministic*
  - This means that we don’t need a transition function, and we actually do know how our environment works
- However, the state space for Ants is very, very large
  - So even though we could specify how our environment works, it is very difficult to code it and for our program to utilize all of this information
  - A more reasonable approach is thus to only look at a subset of states and actions, e.g., the more likely ones, and find an approximation that hopefully works for all states

RL for Ants

- It’s a little weird to use MDPs and RL for Ants. Why?
  - Everything is *deterministic*
  - This means that we don’t need a transition function, and we actually do know how our environment works
- However, the state space for Ants is very, very large
  - So even though we could specify how our environment works, it is very difficult to code it and for our program to utilize all of this information
  - A more reasonable approach is thus to only look at a subset of states and actions, e.g., the more likely ones, and find an approximation that hopefully works for all states
  - Now, it makes sense to use MDPs and RL for Ants

Rollout-based Policy Iteration

Rollout–based Policy Iteration

- In reinforcement learning and some other settings, a *rollout* is essentially a simulation, where the agent takes a certain number of actions in the environment

Rollout-based Policy Iteration

- In reinforcement learning and some other settings, a *rollout* is essentially a simulation, where the agent takes a certain number of actions in the environment
- Algorithms that use rollouts to find a policy are sometimes called rollout-based algorithms

Rollout–based Policy Iteration

- In reinforcement learning and some other settings, a *rollout* is essentially a simulation, where the agent takes a certain number of actions in the environment
- Algorithms that use rollouts to find a policy are sometimes called rollout–based algorithms
- One such algorithm is *rollout–based policy iteration*, which approximates the value function $V(s)$ using rollouts

Rollout-based Policy Iteration

- In reinforcement learning and some other settings, a *rollout* is essentially a simulation, where the agent takes a certain number of actions in the environment
- Algorithms that use rollouts to find a policy are sometimes called rollout-based algorithms
- One such algorithm is *rollout-based policy iteration*, which approximates the value function $V(s)$ using rollouts
  - For every state seen during the rollouts, the value of that state is the average of the rewards after that state for every rollout that included that state

Rollout–based Policy Iteration

- In reinforcement learning and some other settings, a *rollout* is essentially a simulation, where the agent takes a certain number of actions in the environment
- Algorithms that use rollouts to find a policy are sometimes called rollout–based algorithms
- One such algorithm is *rollout–based policy iteration*, which approximates the value function $V(s)$ using rollouts
    - For every state seen during the rollouts, the value of that state is the average of the rewards after that state for every rollout that included that state
    - For the unseen states, we assign them values by looking at the seen states that seem the most similar

Rollout–based Policy Iteration

- In reinforcement learning and some other settings, a *rollout* is essentially a simulation, where the agent takes a certain number of actions in the environment
- Algorithms that use rollouts to find a policy are sometimes called rollout–based algorithms
- One such algorithm is *rollout–based policy iteration*, which approximates the value function $V(s)$ using rollouts
  - For every state seen during the rollouts, the value of that state is the average of the rewards after that state for every rollout that included that state
  - For the unseen states, we assign them values by looking at the seen states that seem the most similar
  - We balance exploration and exploitation by sometimes selecting a random action, rather than using our policy

Rollout-based Policy Iteration

- In reinforcement learning and some other settings, a *rollout* is essentially a simulation, where the agent takes a certain number of actions in the environment
- Algorithms that use rollouts to find a policy are sometimes called rollout-based algorithms
- One such algorithm is *rollout-based policy iteration*, which approximates the value function $V(s)$ using rollouts
  - For every state seen during the rollouts, the value of that state is the average of the rewards after that state for every rollout that included that state
  - For the unseen states, we assign them values by looking at the seen states that seem the most similar
  - We balance exploration and exploitation by sometimes selecting a random action, rather than using our policy
- Let's see a policy trained using this algorithm in action

Rollout-based Policy Iteration (demo)

- In reinforcement learning and some other settings, a *rollout* is essentially a simulation, where the agent takes a certain number of actions in the environment
- Algorithms that use rollouts to find a policy are sometimes called rollout-based algorithms
- One such algorithm is *rollout-based policy iteration*, which approximates the value function $V(s)$ using rollouts
  - For every state seen during the rollouts, the value of that state is the average of the rewards after that state for every rollout that included that state
  - For the unseen states, we assign them values by looking at the seen states that seem the most similar
  - We balance exploration and exploitation by sometimes selecting a random action, rather than using our policy
- Let's see a policy trained using this algorithm in action

Summary

Summary

- Artificial intelligence is all about building programs that act rationally, i.e., *computational rationality*

Summary

- Artificial intelligence is all about building programs that act rationally, i.e., *computational rationality*
- Game playing is an important and natural domain for much of artificial intelligence research and development

Summary

- Artificial intelligence is all about building programs that act rationally, i.e., *computational rationality*
- Game playing is an important and natural domain for much of artificial intelligence research and development
  - We built an agent that plays Hog optimally against `always_roll(6)`, using MDPs and value iteration

Summary

- Artificial intelligence is all about building programs that act rationally, i.e., *computational rationality*
- Game playing is an important and natural domain for much of artificial intelligence research and development
  - We built an agent that plays Hog optimally against `always_roll(6)`, using MDPs and value iteration
  - We built an agent that plays Ants pretty well, using reinforcement learning and rollout-based methods

Summary

- Artificial intelligence is all about building programs that act rationally, i.e., *computational rationality*
- Game playing is an important and natural domain for much of artificial intelligence research and development
  - We built an agent that plays Hog optimally against `always_roll(6)`, using MDPs and value iteration
  - We built an agent that plays Ants pretty well, using reinforcement learning and rollout-based methods
- However, applications of AI go far beyond games and stretch into almost every area of everyday life

Summary

- Artificial intelligence is all about building programs that act rationally, i.e., *computational rationality*
- Game playing is an important and natural domain for much of artificial intelligence research and development
  - We built an agent that plays Hog optimally against `always_roll(6)`, using MDPs and value iteration
  - We built an agent that plays Ants pretty well, using reinforcement learning and rollout-based methods
- However, applications of AI go far beyond games and stretch into almost every area of everyday life
- If you're interested, take:

Summary

- Artificial intelligence is all about building programs that act rationally, i.e., *computational rationality*
- Game playing is an important and natural domain for much of artificial intelligence research and development
  - We built an agent that plays Hog optimally against `always_roll(6)`, using MDPs and value iteration
  - We built an agent that plays Ants pretty well, using reinforcement learning and rollout-based methods
- However, applications of AI go far beyond games and stretch into almost every area of everyday life
- If you're interested, take:
  - CS 188 (Introduction to Artificial Intelligence)

Summary

- Artificial intelligence is all about building programs that act rationally, i.e., *computational rationality*
- Game playing is an important and natural domain for much of artificial intelligence research and development
  - We built an agent that plays Hog optimally against `always_roll(6)`, using MDPs and value iteration
  - We built an agent that plays Ants pretty well, using reinforcement learning and rollout-based methods
- However, applications of AI go far beyond games and stretch into almost every area of everyday life
- If you're interested, take:
  - CS 188 (Introduction to Artificial Intelligence)
  - CS 189 (Introduction to Machine Learning)

Thank you