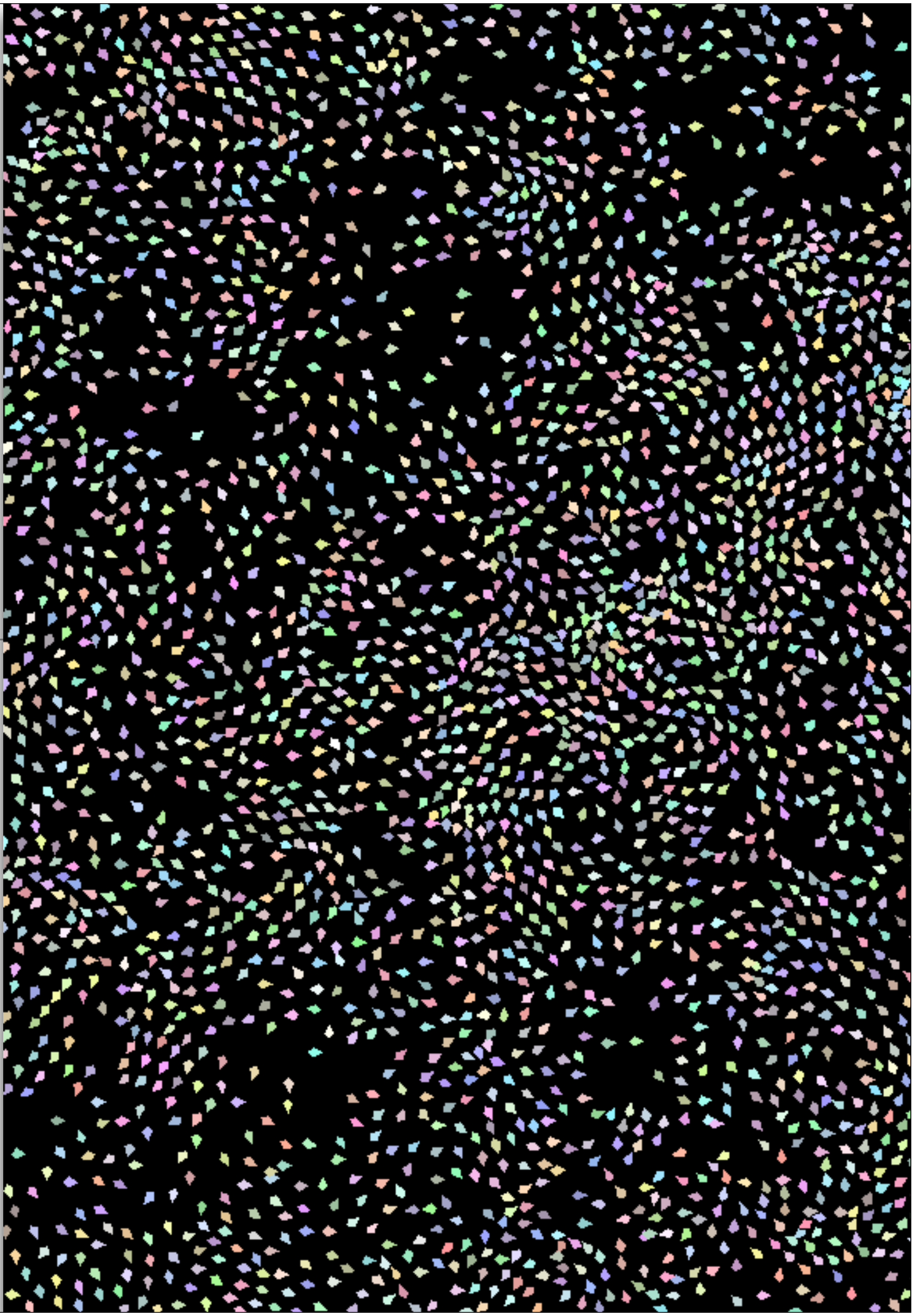
Multiagent Systems and Swarms

Sean Luke

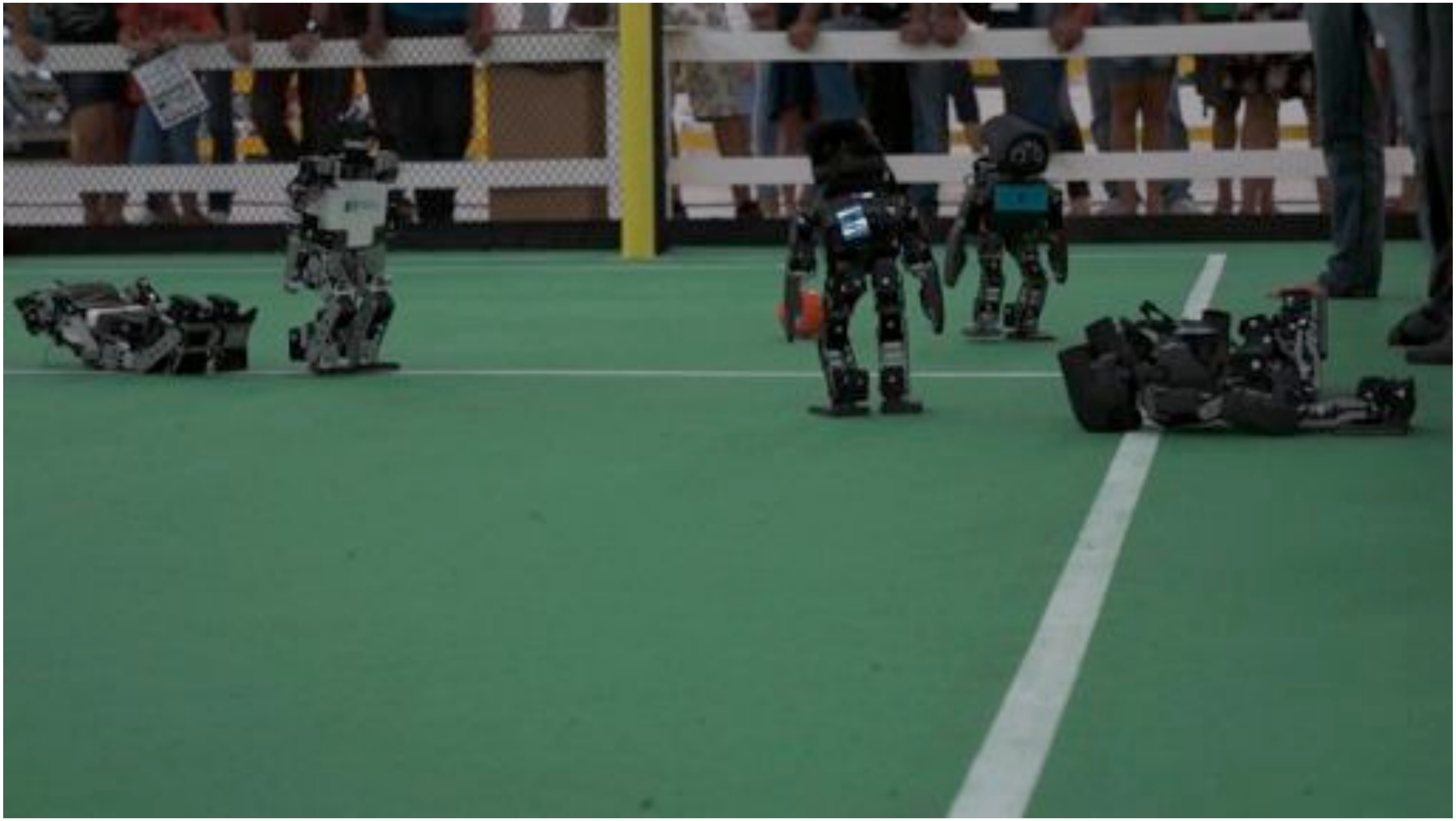
Department of Computer Science
George Mason University

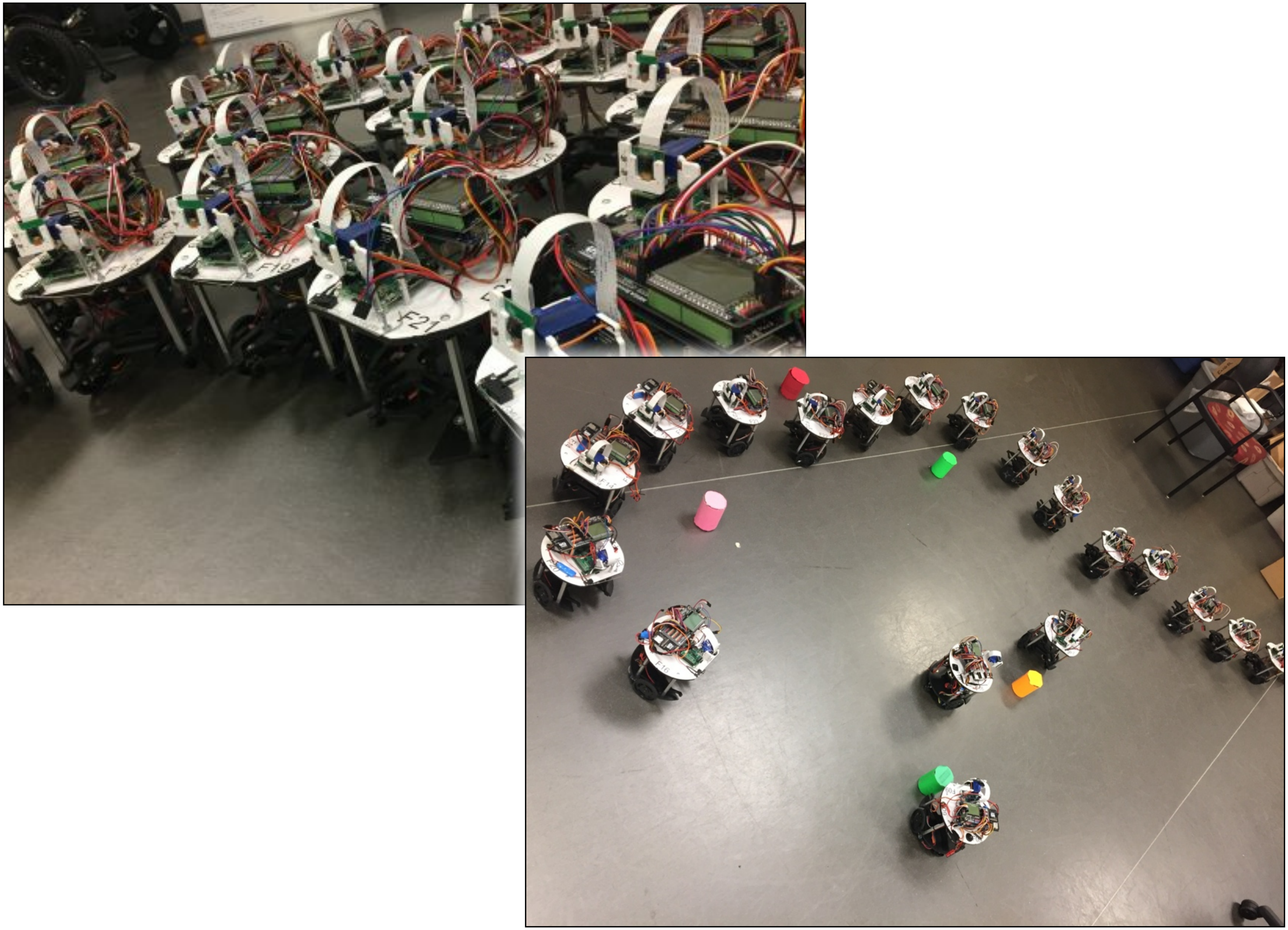
Washington, DC



My Current Multiagent Systems Problem

- Giacomo and Matteo, 5 years old




F21
F16

My Plan

- **Lecture 1**
  Agents
  Why Multiagent Systems are Hard
  Communication and Coordination
  Basic "Small" Multiagent Systems Topics (Game Theory, Voting, Auctions)
  Example from Multiagent Reinforcement Learning

- **Topics for Lectures 2 and 3: mostly my own research**
  Swarms
  Multiagent Task Allocation
  Multirobotics and Swarm Robotics
  Scalable Multiagent Learning
  Indirect Communication Methods
  Agent-Based Modeling

An Agent (in Artificial Intelligence)

- An ***autonomous*** computational mechanism.
  - ***Responsive.*** A mechanism which iteratively **manipulates** its environment in response to **feedback** or **sensor information** it receives from its environment. Engineers call this a **closed loop system.**
  - ***Self-Contained.*** There is no person with joysticks controlling the agent. (The agent has ***agency*** — it does things on its own).
- **"Autonomous agent"** is a redundant term.
- An **autonomous robot** is an agent whose environment is the physical world.

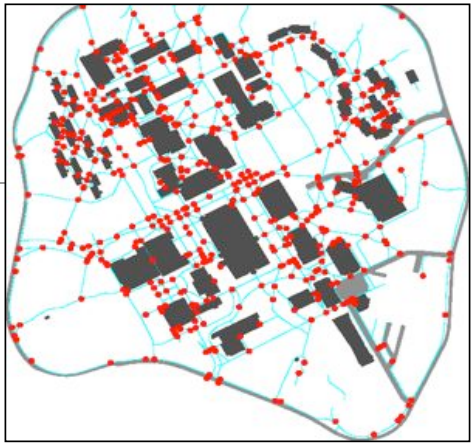
A Distributed System

- A system of **processors**.
- Your objective is to connect and program these processors to perform a task much faster / better than a single processor could do it.
- You have lots of **freedom** in how you design the system architecture.
- A distributed system is typically **optimistic** (we're going to do something **better** than a single agent!)
  - Parallel or cluster computing
    Distributed ad-hoc wireless networks
    Distributed sensor networks

A Multiagent System

- A distributed system of **agents**.
- The agents are autonomous: they're typically not under anyone's control. The agents tend to **interact** with or **coordinate with** one another, and through interaction, they **mess each other up a lot**.
- Agents do not have a **full understanding of the world** and **restrictions on communication** prevent them from telling each other everything**.**
- Your task is to get the agents to solve a problem **despite** the agents stepping on each other's toes.
- A multiagent system is **pessimistic** (how can I get the agents to solve this problem **at all?**)

Multiagent Systems are Everywhere



- **Nature**
  Multicellular organisms

  Coevolving arms races

  Swarms/flocks/schools of ants, termites, gnats, bees, zombies, fish, birds, cows, sheep, horses

  Societies/packs of dogs, apes, dolphins, lions, **humans**

- **Engineering**
  Autonomous car traffic
  Warehouse fulfillment robots
  Micro-air vehicles (MAVs, or drones)
  Social models of human behavior

DDOS attacks
Automated stock trading software
Distributed video game agents
Animated movies of killer robots

Why Multiagent Systems are Hard

- Multiagent systems are **high dimensional.**
  - Large numbers of agents.
  - Heterogeneous agents.
  - Computationally sophisticated agents.
  - High degree of agent interaction.
  - High degree of agent communication or coordination.
  - Agents with different goals: cooperative, competitive, etc.
- Hard to **design** for a high dimensional system.
- Multiagent systems **scale poorly** with centralized methods:
  - Broadcast communication
  - Centralized controllers

Why Multiagent Systems are Hard

- Multiagent systems are **hard to predict.**
  - **“Complexity”** or **“Emergent Behavior”**
- A multiagent system is made up of ***agents***, each with various ***behaviors***, and which ***interact*** in various ways. It is hard to predict the **emergent macrophenomena** in some **closed form.**
- **M(*agents, behaviors, interactions, etc.*)** ➡ ***emergent macrophenoma***

Why Multiagent Systems are Hard

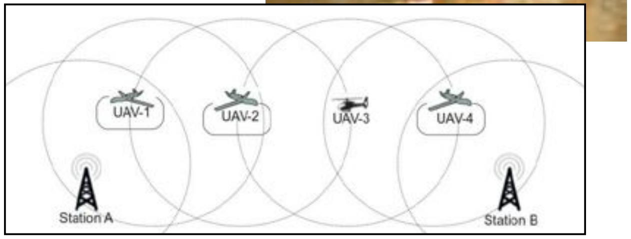
- Multiagent Systems have **difficult inverse functions**.
- **M(*agents, behaviors, interactions, etc.*) ➞ *emergent macro-phenomena***
  This function exists (though it likely has no closed form): it's called a simulator!
- **M'(*emergent macro-phenomena*) ➞ *agents, behaviors, interactions, etc.***
  This **inverse function** function does not exist. But we need it!
- I want my multiagent system to do a certain macro-phenomenon. What micro behaviors should I use to do it? ¯_(ツ)_/¯ This is an **inverse problem.**
- Inverse problems are normally solved using **optimization.** Multiagent systems are arbitrary, so their most common optimization approaches are **metaheuristics** (genetic algorithms, etc.) and **reinforcement learning.**

Communication

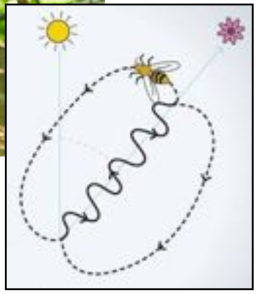
- **Global communication scales poorly (O(n^2)) with number of agents**
- Methods that fail quickly due to scaling issues:
  - Broadcast (Blackboard), Star, Tree, High-Degree Graphs
- **Dynamic agents (such as mobile robots) require rapid reconfiguration of multi-hop architectures**
  - Ad-hoc wireless network routing tables (for example) are not designed to be constantly revised.

Alternative Communication Methods

- **Signaling**
  When I raise my hand, everyone get ready.
  When I in a certain position, that you should kick to me.
  Potentially Global Communication, but limited data



- **Line-of-Sight Communication**
  Local Broadcast Communication
  I can talk to my local neighbors only.
  Multihop methods are hard with dynamic agents



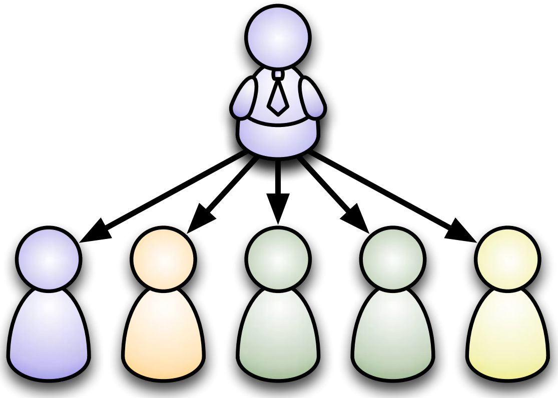
- **Indirect Communication**
  Leaving information in the environment for other agents to discover. Post-It Notes, Predators Marking Territory, Ant Pheromones.

- **Conference**
  Every once in a while everyone gets together to trade notes.
  Bee Waggle Dance. *Global, but only occasional*

Centralized Coordination

- **Centralized coordination does not scale with number of agents (O(n))**
  Far too high a load on the centralized coordinator

- **Centralized coordination is brittle**
  What if the centralized coordinator dies?



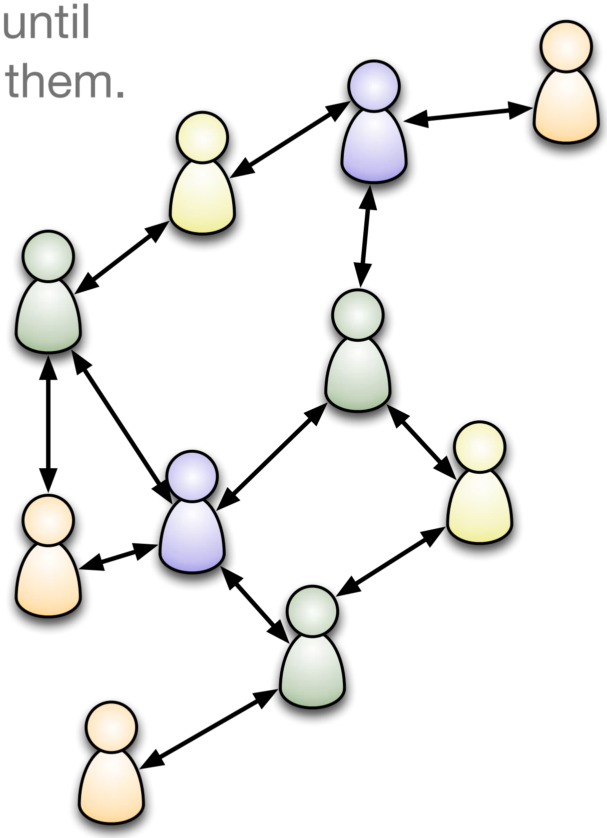
Local or Distributed Coordination

- **Social Laws** Everyone must drive on the right side of the road (in Italia)

- **Local Behaviors** When I see a nearby person has the ball, I will get open. When I have the ball, I will wait until someone is open, then pass to them.

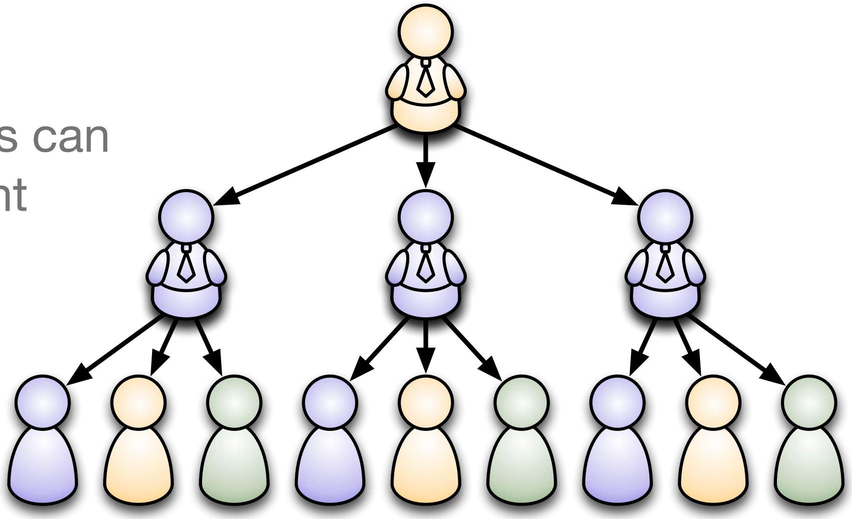
- **Local-only coordination is very restricted**
  What if agent A in Parioli must work with agent B in Trastevere? Who will coordinate them?

  [Most large multiagent systems assume no multihop communication **at all!**]



Hierarchical Coordination

- **Scales well.** No matter how large the hierarchy, every agent must coordinate with at most *N* other agents.



- **Global coordination.** A top-level boss can coordinate the entire group. If an agent needs to coordinate with a remote agent, he can do so via their shared boss.

- **Heterogenous.** Hierarchies can be organized into different groups of similar agents, or into similar groups of different agents.

- **Brittle.** If a boss dies, his entire subhierarchy is disconnected.

Theory

- **Most theory has concentrated on simple things:**
  - Typically 2 agents (often useless) (Game Theory) *or*
  - Independent agents, each with a voice (Auctions, Voting)

    Note: Kenneth Arrow died on Wednesday
- **Theory in swarms has is limited and poor.**
  - Largely complexity, dynamical systems models. Things that don't help.
- **There is a strong need for better theory. But it may be impossible!**

Practice

- **Small numbers (teams) of agents / robots**
  Computationally sophisticated agents, rich interactions, no communication problems, heterogeneous
  Example: RoboCup

- **Large numbers (swarms) of agents / robots**
  Computationally *trivial* agents, limited and distributed interactions, no communication or limited communication, almost always homogeneous
  Example: Kiva Systems

  - **There are almost *no* examples of large numbers of heterogeneous sophisticated robots with sophisticated interactions / communication.**

- **Modular Robots**

- **Agent-Based Models**

Extensive-Form Games

- Agents Sean and Alberto are **players** playing a game (Chess?). The game will an **outcome** for both Sean and Alberto. We'll call that a **reward** or **payoff.**
- Sean has a **strategy.** Let **s** be a *game state:* a current board configuration where Sean is trying to determine what move to make. Let $\mathbf{a} \in \mathbf{A}_s$ be the set of actions **a** Sean can do in a given state **s.**
- Sean's strategy is some function **f(s)➔ a** This is a **pure strategy**.
- Alberto's strategy is some other function **g(s)➔ a**
- The set of pure strategies is countable (and finite if the set of actions and states is finite). **Why?**
- Perhaps Sean makes random actions in a given state. Then his strategy is **f(s)➔** $\mathscr{A}_s$ where $\mathscr{A}_s$ is a **probability distribution** over $\mathbf{A}_s$. The action **a** to perform is selected under this distribution. This is a **mixed strategy.**
- The set of mixed strategies is uncountable. **Why?**

Normal-Form Games

- Pure strategies are deterministic. Since we can enumerate all of them, each combination of Sean's pure strategy and Alberto's pure strategy result in a specific reward for Sean and a specific reward for Alberto. We can enumerate this in a table.

Outcomes for Sean

Sean's Pure Strategy

Alberto's Pure Strategy

	A	B	C	D	E	F	G	H	I
U	7	2	-1	3	6	14	3	9	2
V	8	9	-5	3	2	1.4	0	9	0
W	8	9	-2	-9	2.3	5	1.1	6	7
X	8	8	8	8	8	8	8	8	8
Y	0	0	4	6	10	14	9.2	1	3
Z	0	0	2.1	6	4	7	8	2	5

Outcomes for Alberto

Sean's Pure Strategy

	A	B	C	D	E	F	G	H	I
U	6	2.3	7	7	1.0	2	3	5	12
V	2	4	2	3	0.9	9	4	3	1.1
W	4	5	1	4	0	5	3	3	3
X	6	-3	65	6	0	2	-2	4	6
Y	1	-5	9.1	5	0	0	0	0	0
Z	3	6	3	3	0	9	10	2	3

Normal-Form Game Theory

- A mixed strategy is just a distribution over possible pure strategies! **Why?**
- If Sean and Alberto are following a **mixed strategy** then this table still represents all possible combinations of pure strategies. The payoff of a mixed strategy is the **expected payoff** given the distribution.
- In Normal Form, it's often convenient to think of Sean and Alberto as selecting **actions.** A pure strategy always selects a specific action. A mixed strategy selects an action under its distribution. Now the table consists of the actions that Sean and Alberto can perform, and the payoffs are the payoffs for those actions.
- Sean does action **a.** Alberto does action **b.** The pair **<a, b>** is called a **joint action.** It's the action performed by the *entire system.*

Special Kinds of Games

- **Some special games:** *S is Sean's outcome table, A is Alberto's*
  - Fully cooperative $S = A$
  - Symmetric $S = A^T$
  - Constant Sum $\exists$ c $\forall$ i, j: $S_{ij} + A_{ij} = c$ Competitive
  - Zero Sum $\forall$ i, j: $S_{ij} + A_{ij} = 0$ Competitive
    - In all four of these examples, there's no need for two tables! **Why?**
  - General Sum **Any game** (S_{ij} and A_{ij} don't have any relationship)

- There can be >2 players. Then the table is >2-dimensional.
  - Unless the players cannot communicate, >2 players could result in **collusion.**

What Kinds of Games are These?

3	4	6
2	0	1
2	4	10

3	4	6
2	0	1
2	4	10

3	-4	6
-2	0	1
-2	-4	10

-3	4	-6
2	0	-1
2	4	-10

3	-4	6
-2	0	1
-2	-4	10

3	-2	-2
-4	0	-4
6	1	10

3	2	6
3	2	6
3	2	6

-3	-2	-2
-4	0	-4
6	1	10

Playing a Normal-Form Game

- **One-Shot.** Sean and Alberto independently pick strategies. Then they do one action each and each receive a payoff. **The Goal:** how do I maximize my **expected payoff**?
- **Repeated.** Sean and Alberto independently pick strategies. Then they do one action each and receive a payoff. Then Sean and Alberto possibly change their strategies. Then they again do an action each and both receive a payoff. This repeats forever. **The Goal**: how do I adjust/adapt my strategies over time to maximize my **total expected payoff?**
- **Stochastic.** Sean and Alberto independently pick strategies. Then they do one action each and receive a payoff, **and the game changes as a result of the *joint actions* done.** This **new game** (called a ***state***) has different actions and payoffs, and was chosen from a distribution based on the actions they chose. This process continues forever. Now a "strategy" is a collection of strategies, one for each game we might encounter. **The Goal**: how do I adjust/adapt my strategies over time to maximize my **total expected payoff?**

Equilibria

- An equilibrium is a convergence point in a repeated or stochastic game.

- **Nash Equilibrium.** Sean has strategy *S* right now. Alberto has strategy *A.* If Sean stays with *S,* Alberto has no reason to switch to **any other strategy** *A′* because it wouldn't increase Alberto's outcome. Likewise, Sean will not change if Alberto stays with A because it won't improve Sean's outcome. The pair <*S, A*> is a Nash Equilibrium.

- There can be many Nash Equilibria.

- John Nash proved: if Sean and Alberto are using mixed strategies, there always exists a Nash Equilibrium in the game.

    - This is not the case if Sean and Alberto are restricted to pure strategies (though it is often true).

Equilibria

- What is this game called? You have played it many times.

0	1	-1
-1	0	1
1	-1	0

0	-1	1
1	0	-1
-1	1	0

- What is the Nash Equilibrium (using mixed strategies)?

- What is the Nash Equilibrium in this cooperative game?

10	5	4
5	7	6
4	6	8

- How about this one? (And what is this called?)

-1	0
-3	-2

-1	-3
0	-2

The Prisoner's Dilemma

	Cooperate	Defect
Cooperate	-1	0
Defect	-3	-2

	Cooperate	Defect
Cooperate	-1	-3
Defect	0	-2

- **Tit for Tat**
  First Time: Cooperate
  Thereafter: Do whatever the other agent did last time

- What if there is noise the game?

- **Tit for Two Tats**
  First Time: Cooperate
  Thereafter: Defect only if the other agent has defected twice in a row

Pathologies in Repeated Cooperative Games

- Miscoordination
  - *The Gift of the Magi*
  - A **fundamental problem** in multiagent systems

1	-10
-10	1

-10	1
1	-10

-10	-1
-1	-10

- Relative Overgeneralization

10	0	0
0	5	6
0	6	7

Generalizations to Large Numbers of Agents

- **The *El Farol* Problem and the Tragedy of the Commons**
  In Santa Fe there is a bar called the *El Farol.* It is supposed to be very **cool.** After a workshop at the Santa Fe Institute, all the participants want to go to a cool bar. They all have heard about *El Farol.* But if they all go to *El Farol,* it stops being cool. What can we do?

- **Cooperative/Competitive Games**
  What would a payoff table look like for Soccer? Agents *cooperate* to form a team which *competes* against another team of *cooperating* agents.

- **Coalition Formation**
  You have a large number *N* of agents. An agent's actions are to *join a coalition* with another agent. After the agents have acted, each belongs to a coalition. The payoff function is based on which coalition an agent has joined and which other agents are in that coalition at the moment. Agents can *abandon* a coalition and move to another one. **What is does a Nash Equilibrium mean in this context?**

Voting (Social Choice)

- **Plurality Voting (particularly "First Past the Post" Voting)**
  Each agent can cast only one vote. Whoever has the most votes wins.
- **Approval Voting**
  Each agent can cast many votes, but only one vote per candidate.
  Whoever has the most votes wins.
- **Borda Count**
  Each agent *ranks* all the candidates 1...n . A candidate gets *n-1* points every time he was ranked #1, *n-2* points every time he was ranked #2, and so on, down to *n-n=0* points every time he was ranked #*n.* The candidate with the most points wins.
- **Instant Runoff Voting**
  Each agent *ranks* all the candidates. ITERATE: If a candidate has the majority of #1 ranks, he wins. Otherwise the candidate ranked last is eliminated. [And repeat]. Eventually someone must win.

Arrow's Theorem

- There are *N* candidates for office *x, y, z, ...*
  Each agent *i* has ranked the candidates in a total ordering. His ranking function $a >_i b$ says that *candidate a is preferred to candidate b.*
- We want a *social welfare function* which selects from the candidates based on the preferences of the agents. Three desirable properties of this function:
  - **Pareto Optimality.** If every agent prefers *a* to *b,* then the function will always pick *a* over *b.*
  - **Independence of Irrelevant Alternatives (IIA).** If the function picks *a* over *b,* it will continue to do so even if some agent changes his mind about how he ranks some other candidate *c*.
  - **No Dictatorship.** There is no agent whose ranking controls the function regardless of the other agents preferences.
- **There does not exist a social welfare function with all three of these properties, for >2 candidates.**

Single-Good Auctions

- **English Auction Open Outcry**: People repeatedly bid for an item, until no one wants to bid any higher. The winner is the person who bid the highest.

  Japanese Open Outcry: The auctioneer keeps raising the price until only one person is still willing to pay that price. That person is the winner.

  Dutch Open Outcry: The auctioneer starts with a high price and then starts lowering it. An agent can stop the process whenever he wants: he becomes the winner and pays that price.

  First Price Sealed Bid: Everyone submits a secret bid. Whoever bid the highest is the winner.

  Second Price Sealed Bid, *or* **Vickrey Auction**: Everyone submits a secret bid. Whoever bid the highest is the winner, but the price he pays is the bid of the *second highest bidder.*

- The **Blue** auctions cause *n* agents to bid their true valuation *v* of the product. The **Red** auctions optimally cause *n* agents to bid less than this! **v (n+1)/n**

Other Auctions

- **Multi-Good Auctions:** Multiple copies of an item available simultaneously
  - **Dutch Open Outcry:** The auctioneer starts with a high price and then starts lowering it. An agent purchase an item whenever he wants, and can purchase multiple items. When all the items have purchased, the buyers pay the price of the last purchase.
- **Position Auctions:** Bidders bid for a *rank position.* For example, bidders for keyword ads from Google each place a bid to display an ad. We assume earlier ads on the page are more desirable. Ads appear first-to-last in the order of bids (highest to lowest).
  - **Generalized Second Price Sealed Bid:** The highest bidder pays the price of the second-highest bid, the second-highest bidder pays the price of the third-highest bid, and so on. **Used by Google AdWords**
- **Combinatorial Auctions:** Many different things being sold, you can bid on *combinations* of items. Requires nontrivial combinatorial optimization to solve.

Q-Learning with a Model

- **Markov Decision Process (MDP)**
  Set of **states** $s \in S$

  Set of **actions** $a \in A$
  (can differ by state, but to keep it simple we'll assume every state has the same set of actions)
  Reward Function R(s, a)
  Transition Function T(s, a, s′) *or* P(s′ | s, a)

- The reward and transition functions are the **model**

- The agent wishes to maximize his **utility**, that is, **the expected total reward over the lifetime of the agent.**

- We wish to learn an **optimal policy** π*(s) ➔ a that tells us what action to do in a given state so as to **maximize his utility.**

Q-Learning with a Model

- **Q-Learning**
  Q*(s,a) is the utility the agent should expect to receive if he **starts** in state **s** and does action **a** first, and then **makes optimal moves from then on** (that is, from then on he follows policy π*)

- If we have the function Q*(s,a), we can compute π*:

  $\pi^*(s) = \text{argmax}_{\mathbf{a}}\ Q^*(s,a)$

- So to learn π* we will learn Q*(s,a). We do this with **bootstrapping.**

  1. Start with a wrong Q*(s,a)

  2. For every possible state and every possible action, update Q*(s,a) to more closely reflect the reward received R(s,a)

  3. Also for every possible state and every possible action, and every possible new state s**′**, update Q*(s,a) to be more similar to the best of Q*(s**′**, ...), averaged over the various possible s**′**. **Why?**

Q-Learning with a Model

```
$R(S, A) \leftarrow$ reward function for doing $a$ while in $s$, for all states $s \in S$ and actions $a \in A$
$P(S'|S, A) \leftarrow$ probability distribution that doing $a$ while in $s$ results in $s'$, for all $s, s' \in S$ and $a \in A$
$\gamma \leftarrow$ cut-down constant    ▷ $0 < \gamma < 1$. 0.5 is fine.

$Q^*(S, A) \leftarrow$ table of utility values for all $s \in S$ and $a \in A$, initially all zero
**repeat**
    $Q'(S, A) \leftarrow Q^*(S, A)$    ▷ Copy the whole table
    **for** each state $s$ **do**
        **for** each action $a$ performable in $s$ **do**
            $Q^*(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q'(s', a')$
**until** $Q^*(S, A)$ isn't changing much any more
**return** $Q^*(S, A)$
```

Model-Free Q-Learning

- We don't know P(s′ | s, a), so we can't average over various s′ any more.
- We don't know R(s, a)
- How can we still learn Q*(s,a)?
- We will learn an **estimate** of Q*(s,a) called Q(s,a). This is done as the agent wanders through the world and **experiences** rewards and transitions. It adds them into Q(s,a) as it discovers them.

  1. Start with a wrong Q(s,a)
  2. The agent performs an action **a** in state **s**, receiving some reward **r** and transitioning to state **s′**
  3. Update Q(s,a) to reflect a little bit of the reward **r** and a little bit of the best Q(s′,...)

- This more or less **averages in** all the rewards to form the Q(s,a) **(important!)**

Model-Free Q-Learning

```
$\alpha \leftarrow$ learning rate    ▷ $0 < \alpha < 1$. Make it small
$\gamma \leftarrow$ cut-down constant    ▷ $0 < \gamma < 1$. 0.5 is fine

$Q(S, A) \leftarrow$ table of utility values for all $s \in S$ and $a \in A$, initially all zero
**repeat**
    Start the agent at an initial state $s \leftarrow s_0$    ▷ It's best if $s_0$ isn't the same each time
    **repeat**
        Watch the agent make action $a$, transition to new state $s'$, and receive reward $r$
        $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$
        $s \leftarrow s'$
    **until** the agent's life is over
**until** $Q(S, A)$ isn't changing much any more, or we have run out of time
**return** $Q(S, A)$    ▷ As our approximation of $Q^*(S, A)$
```

Multiagent Reinforcement Learning (MARL)

- Multiple agents playing a **repeated** or **stochastic normal-form** game.
- Each agent is trying to learn what its **best strategy** is.
- Algorithms are designed for different combinations of:
  - Cooperative vs. General-Sum
  - Repeated vs. Stochastic Games
  - Independent Learner vs. Joint-Action Learner
    - Independent Learner: a player is told his reward only.
    - Joint-Action Learner: a player is told his reward, and also what action the other player did.
  - Self-Play: the algorithm knows it's playing against copies of the same algorithm.

Independent Learning of Pure Strategies in Repeated or Stochastic Games

- [Remember that a pure strategy is just an action. A mixed strategy is a distribution over actions from which an action is chosen.]
- A game is a **state.**
- When in game state **s** the action does action **a** and receives a reward **r** from **R(s, a).** He also transitions to a new game state **s′.** He is trying to learn the optimal pure strategy (action) for each game. That is, he's trying to learn **π(s)**. To do this he can maintain **Q(s,a)** as
    - $Q(s, a) \texttt{<-} (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$
- In a **repeated** game **there is only one state,** so we have **R(a), π( ),** and **Q(a).** So we have
    - $Q(a) \texttt{<-} (1-\alpha)Q(a) + \alpha\ r$
- This is (more or less) just averaging the rewards. **Will this be a problem?**

With Liviu Panait, Keith Sullivan, and Ermo Wei [GECCO 2006, JMLR 2008, 2015]

Lenient Multiagent Reinforcement Learning

Cooperative Independent-Learner Repeated Games

Repeat: 1. *N* agents (here *N*=2) each decide on an independent action
2. All agents receive the same *reward* based on their joint action
3. Agents *do not know* what actions the other agents took

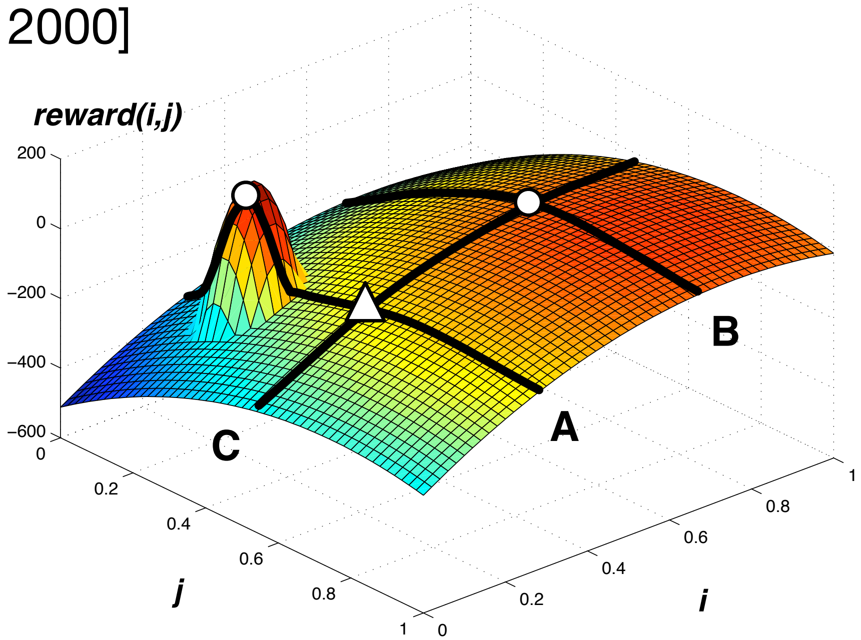
Goal: agents try to adapt to the good rewards.

Relative Overgeneralization [Wiegand 2000]

Sets get trapped in local suboptima surrounding Nash Equilibria in the the joint space.

$$\sum_j reward(A, J_j) < \sum_j reward(B, J_j)$$

$$\max_j reward(A, J_j) > \max_j reward(B, J_j)$$



Lenient Multiagent Reinforcement Learning

Relative Overgeneralization is a *Very Common Problem*
And almost completely unstudied!

Record expected rewards using the Mean over Multiple Joint Actions
Gets trapped in suboptimal Nash Equilibria due to Relative Overgeneralization

Record expected rewards using the Maximum over Multiple Joint Actions
Highly susceptible to noise (the maximum reflects outliers)

		Agent 1		
		A	**B**	**C**
Agent 2	**A**	**11**	-30	0
	B	-30	7	6
	C	0	0	5

		Agent 1		
		A	**B**	**C**
Agent 2	**A**	**10 / 12**	5 / -65	8 / -8
	B	5 / -65	14 / 0	12 / 0
	C	5 / -5	5 / -5	10 / 0

Lenient Multiagent Reinforcement Learning

The LMRL2 Algorithm: be *Lenient*
Early on, make more random actions, and be **lenient to other agent's poor choices.** That is, record expected rewards using the **maximum.**

Later on, make actions largely based on the expected best rewards, and record expected rewards using the **mean.**

Degree of lenience and randomness is determined by a *temperature schedule* much like simulated annealing.

		Agent 1		
		A	**B**	**C**
Agent 2	**A**	**11**	-30	0
	B	-30	7	6
	C	0	0	5

		Agent 1		
		A	**B**	**C**
Agent 2	**A**	**10 / 12**	5 / -65	8 / -8
	B	5 / -65	14 / 0	12 / 0
	C	5 / -5	5 / -5	10 / 0

LMRL2 for Repeated Games

Q(a)	Initially minimum possible reward. *Note it's not Q(s,a)* **why?**
T(a)	Per-action temperature, initially *MaxTemp*

$\alpha \leftarrow 0.1$	learning rate
$\gamma \leftarrow 0.9$	discount for infinite horizon
$\delta \leftarrow 0.995$	temperature decay coefficient
$MaxTemp \leftarrow 50.0$	maximum temperature
$MinTemp \leftarrow 2.0$	minimum temperature
$\omega > 0$	action selection moderation factor (by default 1.0)
$\theta > 0$	lenience moderation factor (by default 1.0)

1. Compute the mean temperature as: $\overline{T} \leftarrow \text{mean}_a T(a)$

2. Select a as follows. If $\overline{T} < \mathit{MinTemp}$, or if $\max_a Q(a) = \infty$, select $\text{argmax}_a Q(a)$, breaking ties randomly. Otherwise use Boltzmann Selection:

   (a) Compute the action selection weights as: $\forall a : W_a \leftarrow e^{\frac{Q(a)}{\omega \overline{T}}}$

   (b) Normalize to the action selection probabilities as: $\forall a : P_a \leftarrow \frac{W_a}{\sum_i W_i}$

   (c) Use the probability distribution P to select action a.

3. The agent does action a and receives reward r.

4. Update $Q(a)$ and $T(a)$ only for the performed action a as:

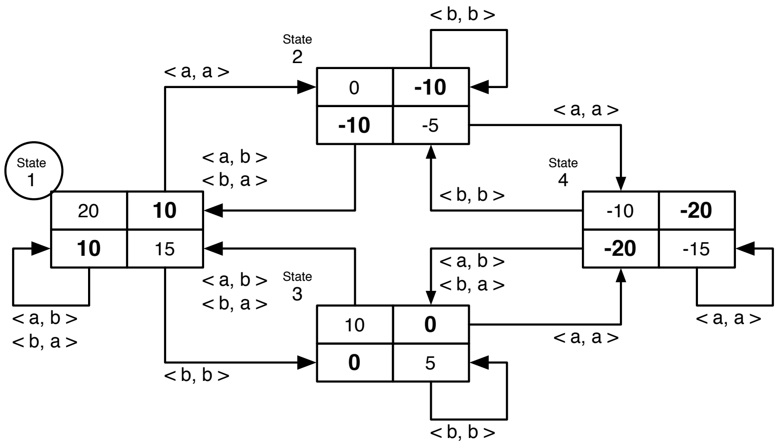
$$\mathit{Rand} \leftarrow \text{random real value between 0 and 1}$$

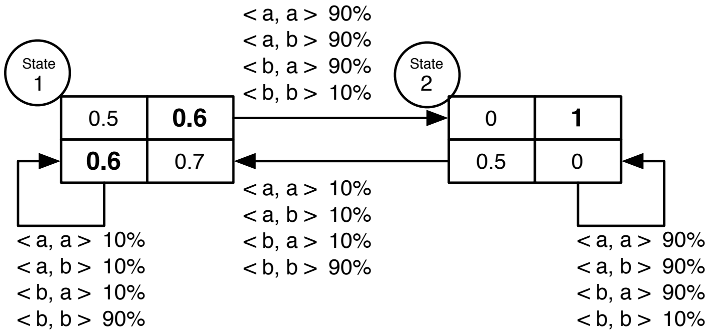
$$Q(a) \leftarrow \begin{cases} r & \text{if } Q(a) = \infty \qquad \text{(initialization was to infinity)} \\ (1-\alpha)Q(a) + \alpha r & \text{else if } Q(a) \leq r \text{ or } \left(\mathit{Rand} < 1 - e^{\frac{-1}{\theta T(a)}}\right) \\ Q(a) & \text{else} \end{cases}$$

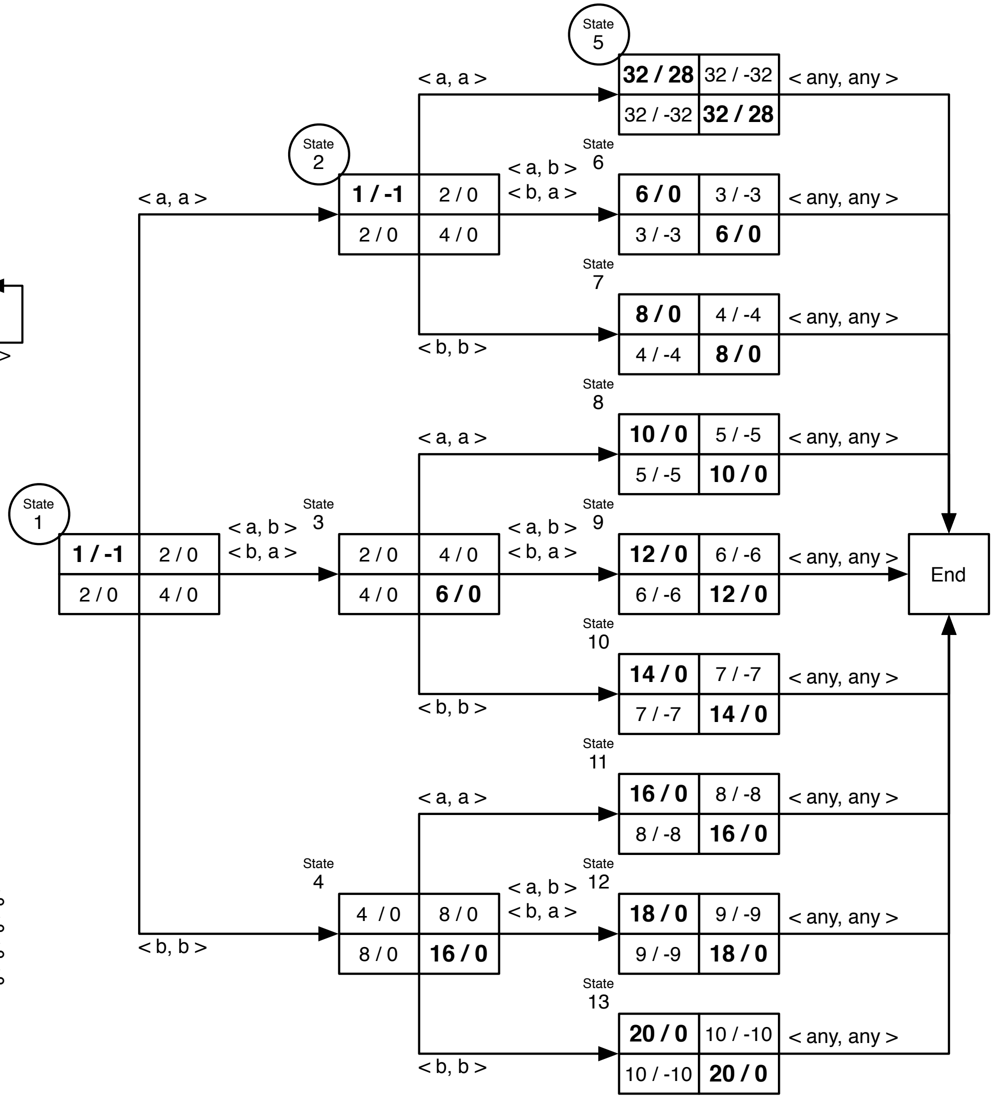
$$T(a) \leftarrow \delta T(a)$$

5. Go to 1.

Lenient Multiagent Reinforcement Learning







LMRL2 for Stochastic Games

- Q(s,a) Initially minimum possible reward.
- T(s,a) Per-state, per-action temperature, initially *MaxTemp*

$\alpha \leftarrow 0.1$	learning rate
$\gamma \leftarrow 0.9$	discount for infinite horizon
$\tau \leftarrow 0.1$	temperature diffusion coefficient
$\delta \leftarrow 0.995$	temperature decay coefficient
$\mathit{MaxTemp} \leftarrow 50.0$	maximum temperature
$\mathit{MinTemp} \leftarrow 2.0$	minimum temperature
$\omega > 0$	action selection moderation factor (by default 1.0)
$\theta > 0$	lenience moderation factor (by default 1.0)

```
Repeat:

  1. Current state $s \leftarrow$ initial state.

  2. Repeat until the current state $s$ is the end state (if any):

     (a) Compute the mean temperature for current state $s$ as: $\overline{T}(s) \leftarrow \text{mean}_a T(s,a)$

     (b) Select $a$ as follows. If $\overline{T}(s) < \mathit{MinTemp}$, or if $\max_a Q(s,a) = \infty$, select $\text{argmax}_a Q(s,a)$,
         breaking ties randomly. Otherwise use Boltzmann Selection:
           i. Compute the action selection weights in current state $s$ as: $\forall a : W_a \leftarrow e^{\frac{Q(s,a)}{\omega \overline{T}(s)}}$
          ii. Normalize to the action selection probabilities in current state $s$ as: $\forall a : P_a \leftarrow \frac{W_a}{\sum_i W_i}$
         iii. Use the probability distribution $P$ to select action $a$.

     (c) The agent, in current state $s$, does action $a$, receives reward $r$, and transitions to new
         state $s'$.

     (d) Update $Q(s,a)$ and $T(s,a)$ only for the performed action $a$ as:

           $\mathit{Rand} \leftarrow$ random real value between 0 and 1

           $$R \leftarrow \begin{cases} r & \text{if } \max_{a'} Q(s',a') = \infty \\ r + \gamma \max_{a'} Q(s',a') & \text{else} \end{cases}$$

           $$Q(s,a) \leftarrow \begin{cases} R & \text{if } Q(s,a) = \infty \qquad \text{(initialization was to infinity)} \\ (1-\alpha)Q(s,a) + \alpha R & \text{else if } Q(s,a) \leq R \text{ or } (\mathit{Rand} < 1 - e^{\frac{-1}{\theta T(s,a)}}) \\ Q(s,a) & \text{else} \end{cases}$$

           $$T(s,a) \leftarrow \delta \times \begin{cases} (1-\tau)T(s,a) + \tau \overline{T}(s') & \text{if } s' \text{ is not the end state (if any)} \\ T(s,a) & \text{else} \end{cases}$$

     (e) $s \leftarrow s'$
```

Lenient Multiagent Reinforcement Learning

Results

Two measures (“Complete” and “Correct”), shown as **Complete / Correct**

LMRL2 is *always* in the top tier for either Complete or Correct, often both.

LMRL2 requires you to tune only a *single* parameter.

Test Problem	LMRL2	Q-Learning	Distributed Q	Hysteretic Q	WoLF-PHC	FMQ
Boutilier	**10000/10000**	**10000/10000**	**10000/10000**	**10000/10000**	**9998/ 9998**	
Common Interest	**9971**	9942	2080	**9990**	**9968**	
Gradient 1	*8407/**10000**	8609/ 8695	**9999/ 9999**	9925/ 9926	2329/ 2335	
Gradient 2	548/ **9997**	2430/ 5995	0/ 1266	157/ 5609	**5147**/ 5897	
Heaven and Hell	**9991/10000**	8289/ 9942	**10000/10000**	9848/**10000**	9954/**10000**	
RO 1	†9368/**10000**	3350/ 3350	**10000/10000**	**10000/10000**	1943/ 2280	
RO 2	**9999/10000**	2/ 514	9258/ 9258	9897/**10000**	2/ 512	
RO 3	**7332/ 7332**	5337/ 5337	2272/ 2272	2737/ 2737	2769/ 2769	
Climb	**9999**	1661	**10000**	**10000**	387	9956
Climb-PS	**9930**	1820	2821	7454	391	9857
Climb-FS	**9016**	1763	3874	2558	676	3894
Penalty	**9999**	**9997**	**10000**	**10000**	9951	**10000**