Fully Observable + Multiple Agents

- If agents act sequentially and can observe the state before acting: **Perfect Information Games**.
- Can do dynamic programming or search: Each agent maximizes for itself.
- Multi-agent MDPs: value function for each agent. Each agent maximizes its own value function.
- Multi-agent reinforcement learning: each agent has its own $Q$ function.
Fully-observable Game Tree Search

1: procedure GameTreeSearch(n)
2: Inputs
3: n a node in a game tree
4: Output
5: A pair: value for each agent for node n, path that gives this value
6: if n is a leaf node then
7: return \{i : evaluate(i, n)\}, None
8: else if n is controlled by agent i then
9: max := −∞
10: for each child c of n do
11: score, path := GameTreeSearch(c)
12: if score[i] > max then
13: max := score[i]
14: res := (score, c : path)
15: return res
What happens with this game? Payoff is for Andy, Barb

What if the 2,0 payoff was 1.9,0.1?
Should Barb be rational / predictable?
What should Andy do if Barb threatens to not do her best action?
Pruning Dominated Strategies

Special case: two person, competitive (zero sum) game. Utility for one agent is negative of utility of other agent. \(\Rightarrow\) minimax.

- square MAX nodes controlled by maximizing agent score
- round MIN nodes are controlled by a minimizing adversary
- leaves without a number do not need to be evaluated.

\[ \rightarrow \alpha-\beta \] pruning.
Each agent decides what to do without seeing the other agent’s action.

What should each agent do?
• Assume an $n$-player game in normal form

• A strategy for an agent is a probability distribution over the actions for this agent.

• A strategy profile is an assignment of a strategy to each agent.

• A strategy profile $\sigma$ has a utility for each agent. Let $utility(\sigma, i)$ be the utility of strategy profile $\sigma$ for agent $i$.

• If $\sigma$ is a strategy profile:
  - $\sigma_i$ is the strategy of agent $i$ in $\sigma$,
  - $\sigma_{-i}$ is the set of strategies of the other agents.

  Thus $\sigma$ is $\sigma_i \sigma_{-i}$
Nash Equilibria

\[ \sigma_i \text{ is a best response to } \sigma_{-i} \text{ if for all other strategies } \sigma'_i \text{ for agent } i, \]

\[ \text{utility}(\sigma_i, \sigma_{-i}, i) \geq \text{utility}(\sigma'_i, \sigma_{-i}, i). \]

A strategy profile \( \sigma \) is a **Nash equilibrium** if for each agent \( i \), strategy \( \sigma_i \) is a best response to \( \sigma_{-i} \). That is, a Nash equilibrium is a strategy profile such that no agent can do better by unilaterally deviating from that profile.

Theorem [Nash, 1950] Every finite game has at least one Nash equilibrium.
Stochastic Policies

Probability of a goal.

<table>
<thead>
<tr>
<th>kicker</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>right</td>
<td>0.3</td>
<td>0.9</td>
</tr>
</tbody>
</table>

$p_k$ is $P(kicker = right)$

$p_j$ is $P(goalkeeper = right)$
Multiple Equilibria

Hawk-Dove Game:

<table>
<thead>
<tr>
<th>Agent 1</th>
<th>doves</th>
<th>hawk</th>
</tr>
</thead>
<tbody>
<tr>
<td>dove</td>
<td>R/2,R/2</td>
<td>0,R</td>
</tr>
<tr>
<td>hawk</td>
<td>R,0</td>
<td>-D,-D</td>
</tr>
</tbody>
</table>

$D$ and $R$ are both positive with $D >> R$. 
Just because you know the Nash equilibria doesn’t mean you know what to do:

<table>
<thead>
<tr>
<th></th>
<th>Agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>shopping</td>
</tr>
<tr>
<td>Agent 1</td>
<td>2,1</td>
</tr>
<tr>
<td>shopping</td>
<td>0,0</td>
</tr>
<tr>
<td>football</td>
<td></td>
</tr>
</tbody>
</table>
Two strangers are in a game show. They each have the choice:
- Take $100 for yourself
- Give $1000 to the other player

This can be depicted as the payoff matrix:

<table>
<thead>
<tr>
<th></th>
<th>Player 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>take</td>
</tr>
<tr>
<td>Player 1</td>
<td></td>
</tr>
<tr>
<td>take</td>
<td>100,100</td>
</tr>
<tr>
<td>give</td>
<td>0,1100</td>
</tr>
</tbody>
</table>
Example:

- There are 100 agents.
- There is an common environment that is shared amongst all agents. Each agent has 1/100 of the shared environment.
- Each agent can choose to do an action that has a payoff of +10 but has a -100 payoff on the environment or do nothing with a zero payoff.
- For each agent, doing the action has a payoff of $10 - 100/100 = 9$
- If every agent does the action the total payoff is $1000 - 10000 = -9000$
To compute a Nash equilibria for a game in strategic form:

- Eliminate dominated strategies
- Determine which actions will have non-zero probabilities. This is the support set.
- Determine the probability for the actions in the support set
Eliminating Dominated Strategies

<table>
<thead>
<tr>
<th>Agent 1</th>
<th>Agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>d₂</td>
</tr>
<tr>
<td>b₁</td>
<td>3,5</td>
</tr>
<tr>
<td>c₁</td>
<td>1,1</td>
</tr>
<tr>
<td></td>
<td>2,6</td>
</tr>
</tbody>
</table>

- Can prune c₁ because it is dominated by a₁
- Can prune f₂ because it is dominated by [0.5 : d₂, 0.5 : e₂]
- Next prune b₁ then e₂
- Single Nash equilibrium is (a₁, d₂)
Computing probabilities in randomized strategies

Given a support set:

- Why would an agent will randomize between actions \( a_1 \ldots a_k \)? Actions \( a_1 \ldots a_k \) have the same value for that agent given the strategies for the other agents.

- This forms a set of simultaneous equations where variables are probabilities of the actions.

- A solution with all probabilities in range \((0,1)\) is a Nash equilibrium.

Search over support sets to find a Nash equilibrium.
Example: computing Nash equilibrium

<table>
<thead>
<tr>
<th>kicker</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>right</td>
<td>0.3</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Probability of a goal.

When would goalkeeper randomize?
Let $p_k$ be probability the kicker will kick right.

\[
P(goal \mid jump \ left) = p(goal \mid jump \ right) \\
p_k \times 0.3 + (1 - p_k) \times 0.6 = p_k \times 0.9 + (1 - p_k) \times 0.2 \\
0.6 - 0.2 = (0.6 - 0.3 + 0.9 - 0.2) \times p_k \\
p_k = 0.4
\]

Similarly for goal keeper: $P(jump \ right) = 0.3$

Probability of a goal is:

\[
(0.6 \times 0.7) \times 0.6 + (0.6 \times 0.3) \times 0.2 + (0.4 \times 0.7) \times 0.3 + (0.4 \times 0.3) \times 0.9 = 0.48
\]
Collect statistics of the other player.

Assuming those statistics reflect the stochastic policy of the other agent, play a best response.

Both players using fictitious play converges to a Nash equilibrium for many types of games (including two-player zero-sum games).

If an opposing agent knew the exact strategy (whether learning or not) agent A was using, and could predict what agent A would do, it could exploit that knowledge.
controller $Stochastic\_policy\_iteration(S, A, \alpha, \gamma, q\_init, p\_init)$

Inputs

$S$ is states, $A$ is actions, $\alpha$ is step size, $\gamma$ discount

$q\_init$ and $p\_init$ ($>0$) are initial $Q$ and $P$ values

Local

$P[S, A]$ unnormalized $P(a \mid s)$ ▷ Dirichlet

$Q[S, A]$ estimate of value of doing $A$ in state $S$

$P[s, a] := p\_init; \ Q[s, a] := q\_init$ for each $s \in S$ and $a \in A$

observe state $s$; select action $a$ at random

repeat

$do(a)$

observe reward $r$, state $s'$

select action $a'$ based on $P[s', a']/\sum_{a''} P[s', a'']$

$Q[s, a] := Q[s, a] + \alpha \star (r + \gamma \star Q[s', a'] - Q[s, a])$

$a\_best := \arg\max_a (Q[s, a])$

$P[s, a\_best] = P[s, a\_best] + 1$

$s := s'$; $a := a'$

until termination
Repeated playing goal-kick game with single state \((\alpha = 0.1, \gamma = 0, q_{\text{init}} = 1, p_{\text{init}} = 5)\).

AI\texttt{Python}: \texttt{masLearn.py}
AlphaZero – plays world-class chess, shogi, and Go. Improvement of program that beat Lee Sedol in 2016.

- implements modified policy iteration
- uses a deep neural network:
  - Input: the board position
  - Output: the value function and a stochastic policy
- To get a better estimate of the current state, it does stochastic simulation (forward sampling) of the rest of the game, using the stochastic policy with the upper confidence bound (UCB).
- This relies on a model to restart the search from any point.
- It was trained on self-play, playing itself for tens of millions of games.