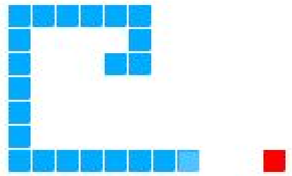
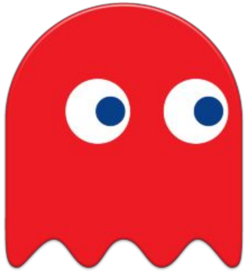
Naggin-Naagin: The Self Learning Snake

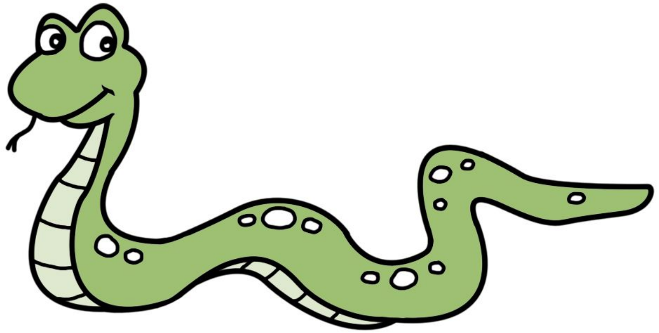


Don Kurian Dennis (1301CS17)
Ritobroto Maitra (1301CS50)

Outline

- Introduction
- Modeling Adversarial Games
- Min-Max trees and Optimum Policy
- Evaluation Functions
- Expectimax Trees
- Modelling Uncertain Outcomes:: The Markov Decision Process
- Q learning
- Q Learning: Snake Agent
- Q Learning: State Space
- Q Learning: Video Demonstration
- Future Work





Introduction

Modelling the Snake game has two aspects:

1. Game play with a model of the world (perfect or partial information).
2. Learning world model by train (no domain knowledge)

Part 1:

- Exploring partial information search (running model Pacman)
- Exploring Adversarial games
- Exploring uncertain event games
- Markov Decision Processes
- Q-Learning

Part 2:

- Deep Q learning for 1-cell snake
- MDP modelling for n-cell snake
- Deep Q Learning for n-cell snake

Modeling Adversarial Games

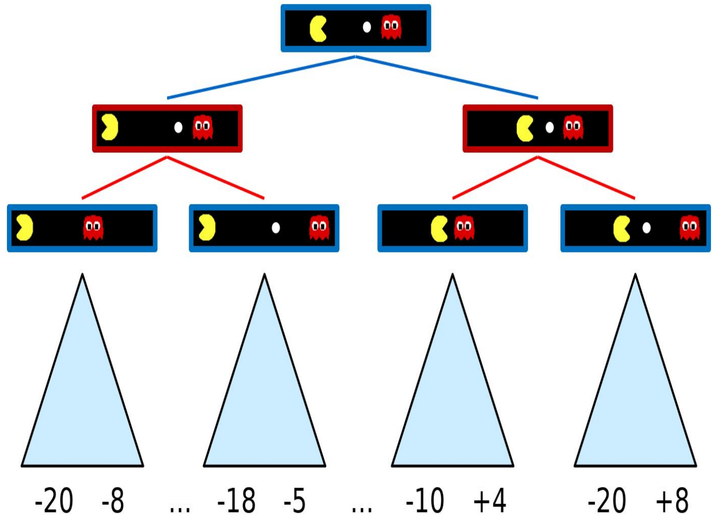
- Games like Chess, Checkers, Go, Tic-tac-toe, Snake and Pacman can be modelled as adversarial search problems.

- Pacman world model is a non-deterministic, zero-sum adversarial problem. Snake world model is deterministic, zero-sum but with other complications.

- Non-Zero-Sum: Cooperation, indifference, competition possible.

Modeling Adversarial Games

- A zero-sum game like pacman can be modelled as a MinMax Tree.
- This assumes a mastermind or perfect adversary.
- Overly pessimistic approach.
- Exponential search space. We are literally predicting all possible outcomes.



[Image: Dan Klein, Pieter Abbeel, http://ai.berkeley.edu.]

Min-Max Trees and Optimum Policy

- Min-Max is very inefficient in its raw sense. (m depth, b branching factor)

  Time $O(b^m)$

  Space $O(bm)$
- Idea: Explore till some depth and then use a heuristic.
- Evaluation Function:

  - Imperfect
  - The deeper they are buried, the less the imperfections matter.
- Idea: Alpha-Beta Pruning

Demo: Evaluation
01_reflex_testclassic_ghost
02_reflex_testclassic_ghost_food
03_reflex_testclassic_ghost_food_score_capsule
04_reflex_mediumclassic_ghost_food_score_capsule
05_reflex_mediumClassic_multipleghost
06_snake_Manhattan

Expectimax Tree: Imperfect Opponent

- Min-Max assumes adversary is perfect and is playing optimally and there is no stochasticity.
- This is rarely the case in real scenarios.
- Use expectations instead of just min-max scores.
- Replace min-max values with expected utilities.

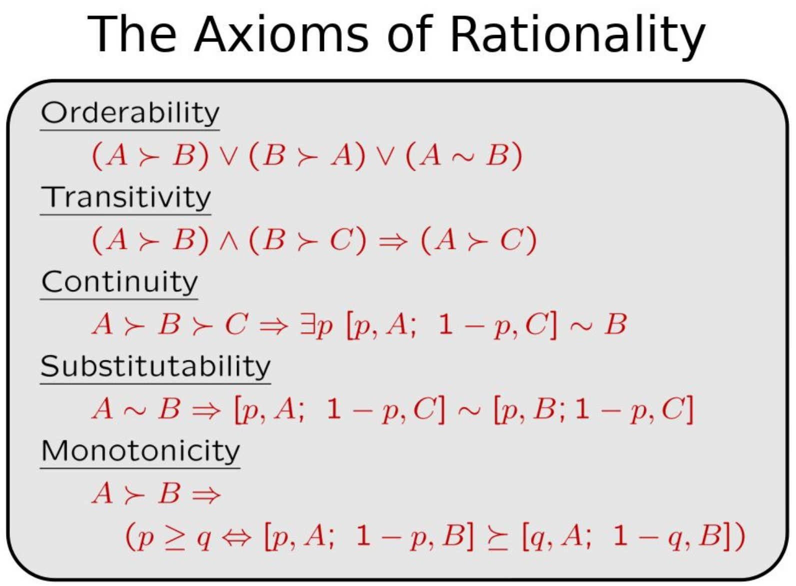
Markov Decision Process

A MDP is defined by

- A set of states $s \,\epsilon\, \boldsymbol{S}$
- A set of actions $a \,\epsilon\, \boldsymbol{A}$
- A transition function *T(s,a,s')*
- A reward function *R(s, a, s')*
- A start state
- A terminal state (optional)

Works on *'Rational Preferences'*

Has a notion of discounting



The Axioms of Rationality

Orderability

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

Transitivity

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

Continuity

$$A \succ B \succ C \Rightarrow \exists p\ [p, A;\ 1-p, C] \sim B$$

Substitutability

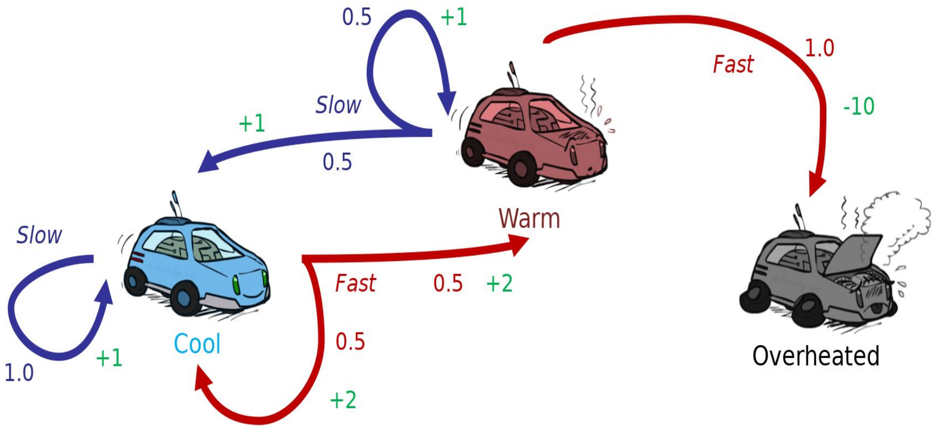
$$A \sim B \Rightarrow [p, A;\ 1-p, C] \sim [p, B; 1-p, C]$$

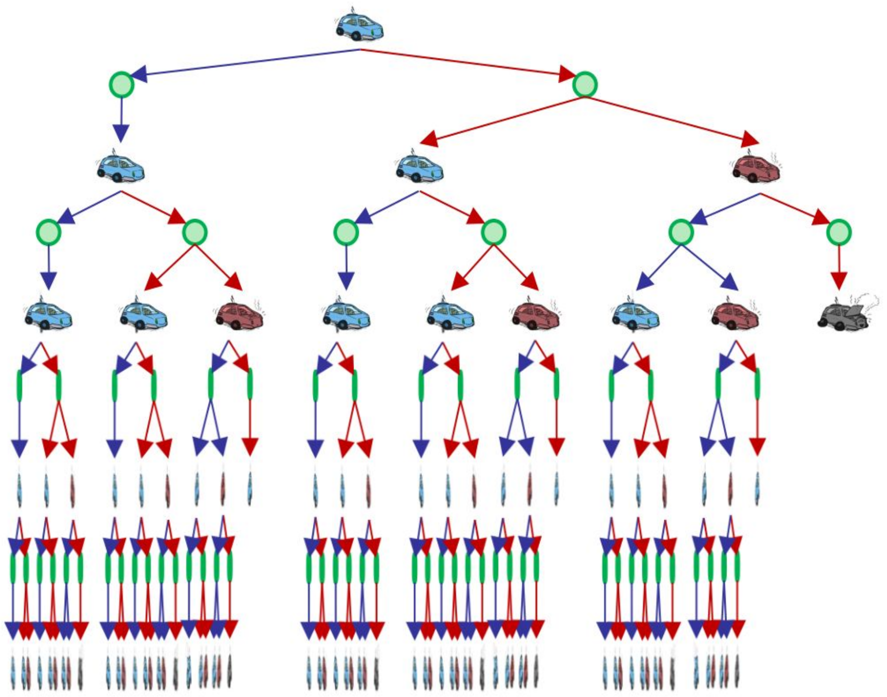
Monotonicity

$$A \succ B \Rightarrow$$
$$(p \geq q \Leftrightarrow [p, A;\ 1-p, B] \succeq [q, A;\ 1-q, B])$$

[Image: Dan Klein, Pieter Abbeel, http://ai.berkeley.edu.]

Markov Decision Process: Illustration





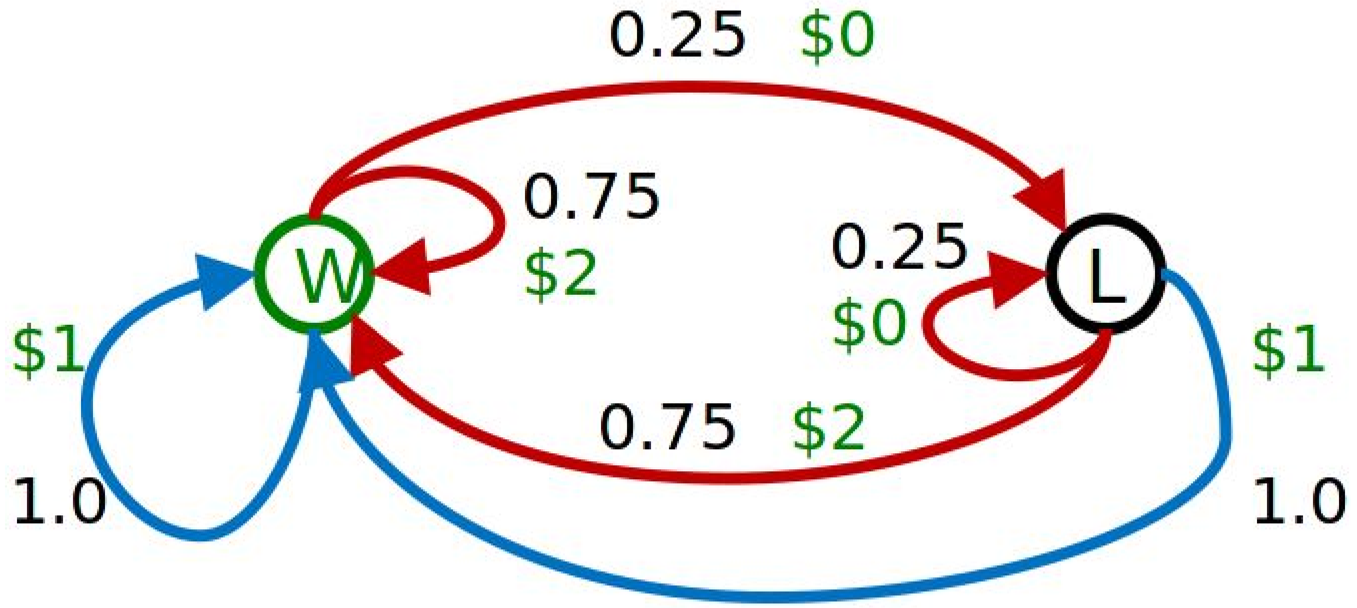
[Image: Dan Klein, Pieter Abbeel, http://ai.berkeley.edu.]

MDP To Reinforcement Learning: Double Bandit

$1
WINNER
777
DOUBLE
$2/$0
OR NOTHING

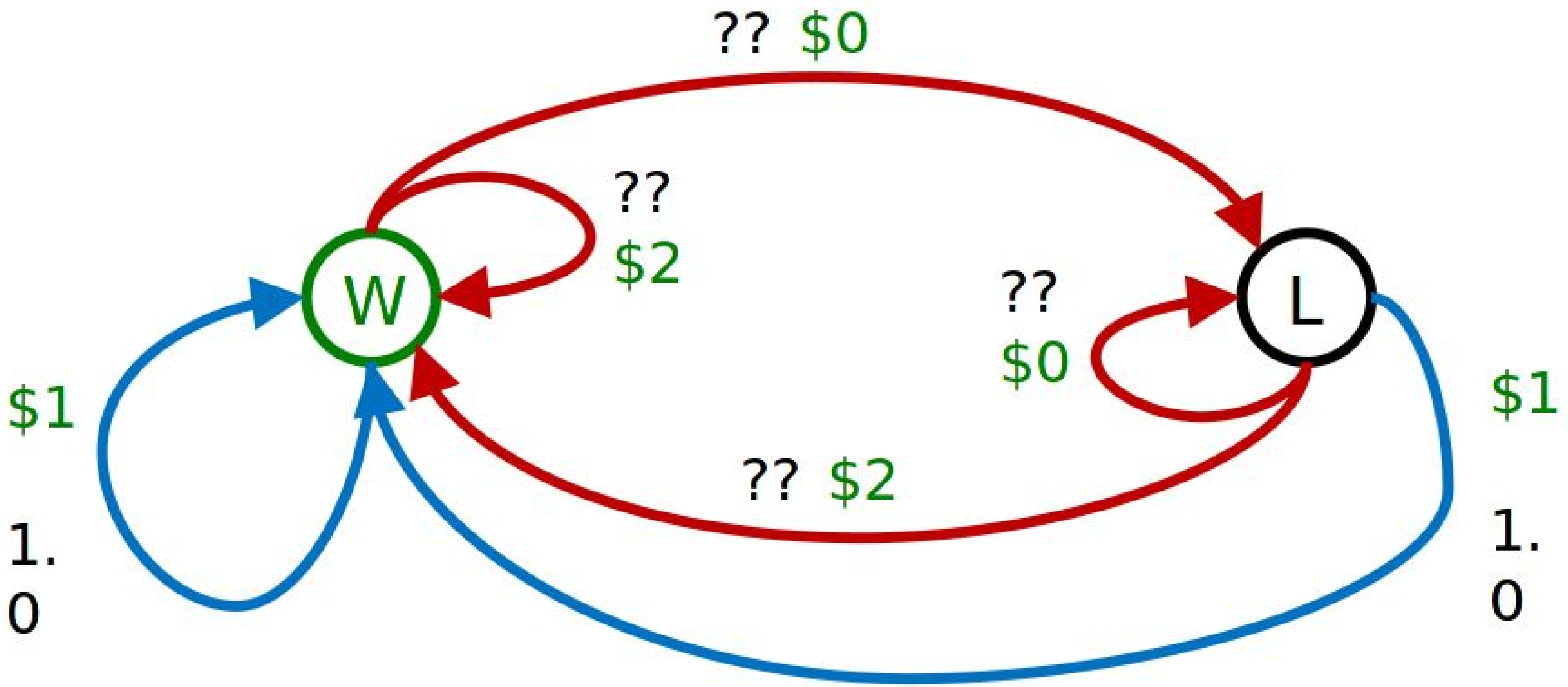
[Image: Dan Klein, Pieter Abbeel, http://ai.berkeley.edu.]

Double Bandit: Offline Planning



[Image: Dan Klein, Pieter Abbeel, http://ai.berkeley.edu.]

Double Bandit: Reinforcement Learning



[Image: Dan Klein, Pieter Abbeel, http://ai.berkeley.edu.]

Snake World : MDP to Reinforcement Learning

- Given a good MDP model, we know how to find optimal policies
  - Value Iteration
  - Policy Iteration
- The goal remains the same: learn an optimal policy
  - But without a transition model
  - Knowing the current state and the current reward, but without access to the reward function
- Issues:
  - Non-deterministic effects
  - Delayed Reward problem (What move should you make now to have a checkmate 20 moves later?)
  - Credit assignment problem (How much is each move worth?)
- Advantage:
  - General Purpose (Even more so with Deep RL) and Zero Domain Data

Q-Learning

- Model-free reinforcement learning
- Proven result: For any finite MDP, Q-Learning eventually finds an optimal policy.
- Each State-Action Pair (S,A) returns a “Q” value, which is an indicator of how likely the pair is to be part of an optimal policy.
- We start with an arbitrary set of values, and iterate till we find the best set.
- Factors:
    - Exploration vs Exploitation
    - Initialization (Random vs Optimistic)

Q-Learning: Snake Agent

- In RL, we do not teach the agent.
- We simply have a way to tell the agent what’s good and what’s bad.
- The agent we are using now uses the following scores:
  - +50 for each time it eats the food
  - -1000 for each time there’s a termination condition

Q-Learning: Snake Agent

- However, there are two subtle complications:
  - The termination conditions need to include all possible combinations of the snake hitting itself, the snake hitting the border wall, the snake hitting the maze walls or the snake going into a corner
  - The second one is that in itself the two scores are not enough. It also needs a live penalty (otherwise the snake just go in a simple path and never try to reach the food).
  - So there’s a live penalty of -10.
- These scores are hand-tested. In our future work, we plan to do this more methodically.

State Space

- The state, if not represented correctly, can grow very fast very quickly (often superexponentially)
- So we avoid this by representing the state of the game relative to the snake’s head instead of tracking the snake’s head as an absolute.
- Essentially, we shift the origin at every movement to correspond to the snake’s head. This allows us to use the same calculations over and over again
- We divide the game canvas into quadrants (relative to the snake’s head) and proceed as we would - the food lies in the quadrant that provides the general direction vector, and the snake orients itself to move towards that (and iteratively so for every now ‘head’ position or origin).

Video Demonstration

Strategies that the snake has learnt:

- The snake often moves diagonally
- This is due to the relative state representation
- Observe the movement when the snake head and the food are on different sides of the middle wall.
- It first moves to avoid the wall (higher penalty) and then when it has cleared it, moves to capture the food.
- Example: between scores 6 and 7

Demo: Snake
07_qlearning_snake

Future Work

- Bonus Foods
- Comparative study of different sorts of mazes
- Different levels
- Introduce Deep-Q for 1-Cell Snake
- Deep-Q for (Dynamic) N-Cell Snake

Thank You