GITHUB LINK: https://github.com/sarmatejas1006/MonoRL

MonoRL: Reinforcement Learning Agent for Intelligent Monopoly

May 10, 2018

Group 9

Tejas Sarma

Maitrai Kansal

Meghana Sanjay

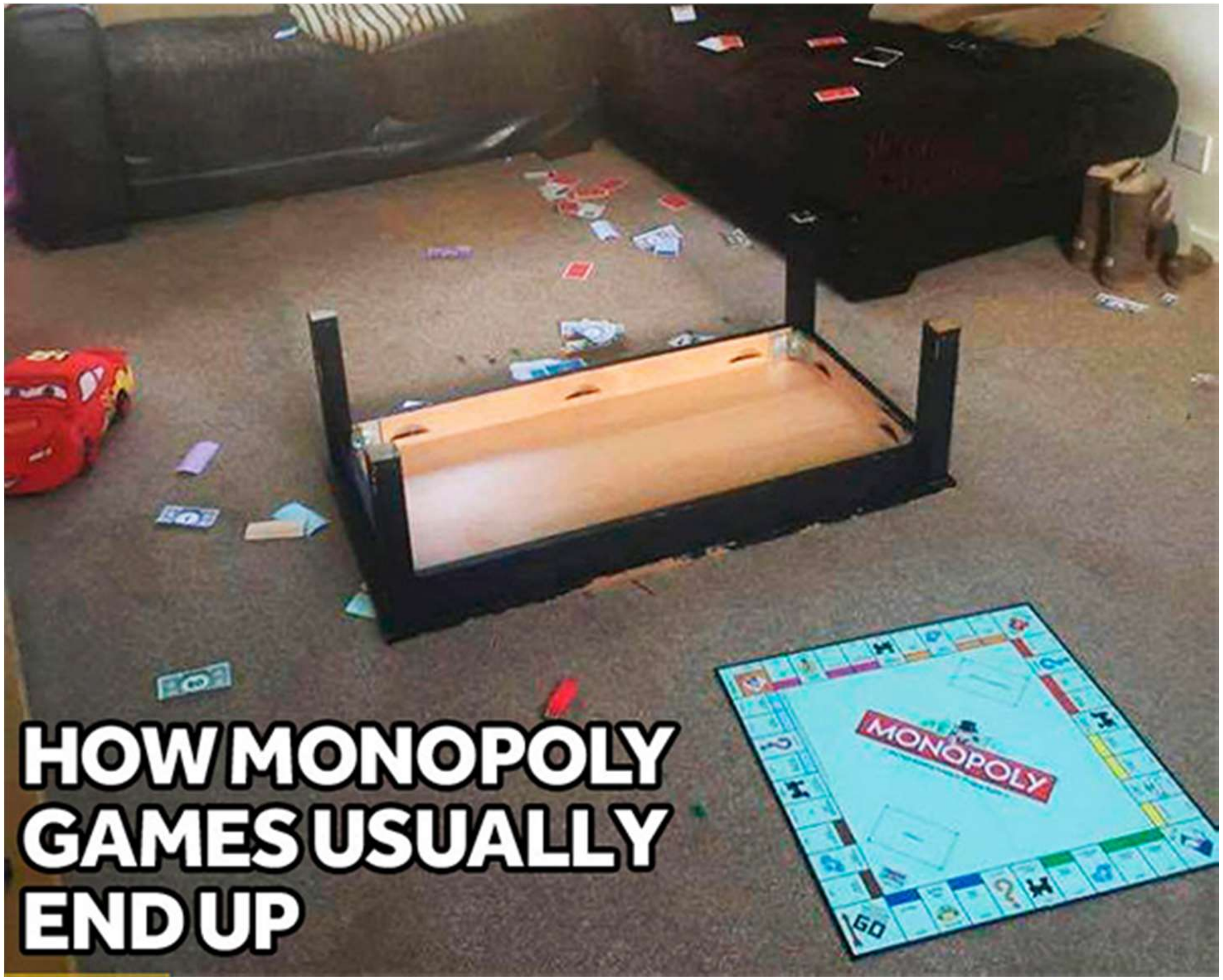
CONTENTS

INTRODUCTION

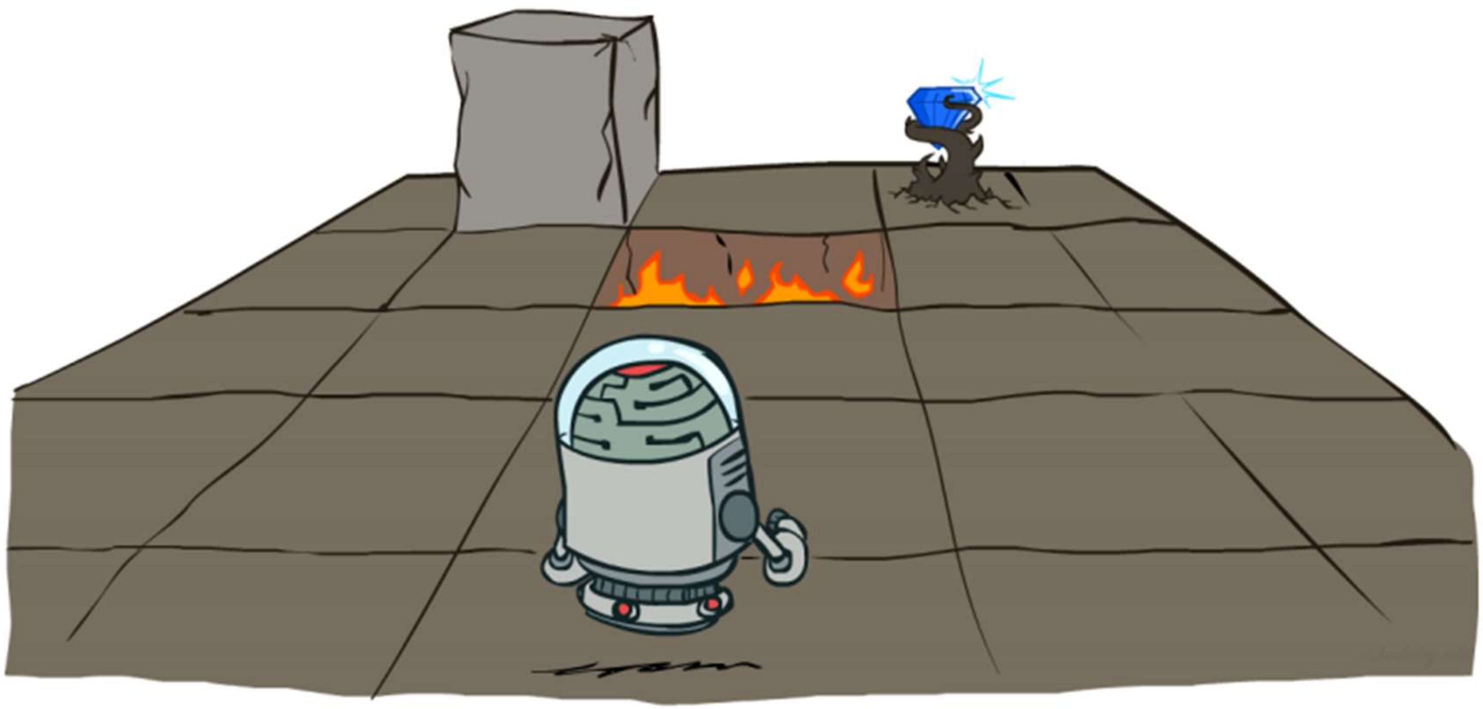
- Why RL for Monopoly?
- Markov Decision Process
- Reinforcement Learning

Why AI for Monopoly?

- Monopoly just a puppet problem
- Real task is to create a pseudo rational agent
- Planning and Learning through RL
- Research using Monopoly as an experimental problem as an equivalent to real life planning problems


HOW MONOPOLY
GAMES USUALLY
END UP
MONOPOLY
GO

Markov Decision Process

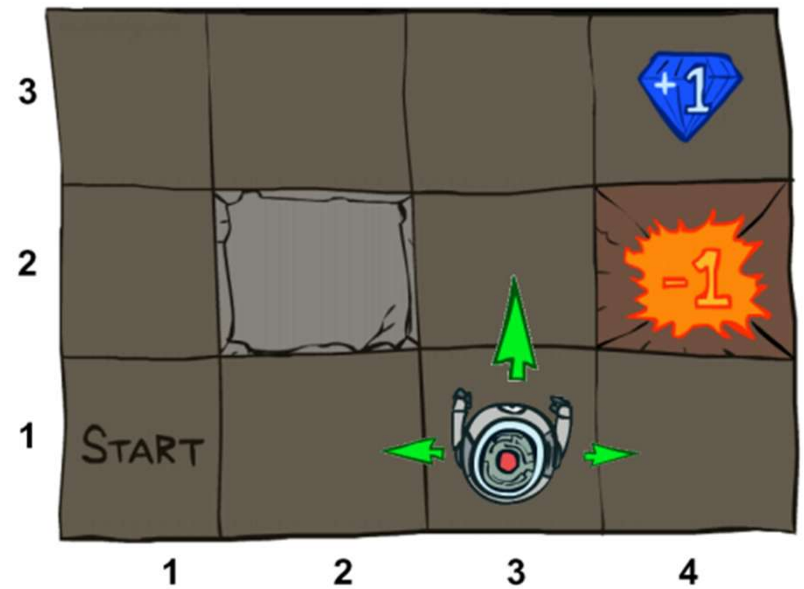


Markov Decision Process

"Markov" generally means that given the present state, the future and the past are independent

An MDP is defined by:

- A set of states s ∈ S
- A set of actions a ∈ A
- A transition function T(s, a, s') Probability that a from s leads to s', i.e., P(s'| s, a) Also called the model or the dynamics
- A reward function R(s, a, s') Sometimes just R(s) or R(s') A start state Maybe a terminal state

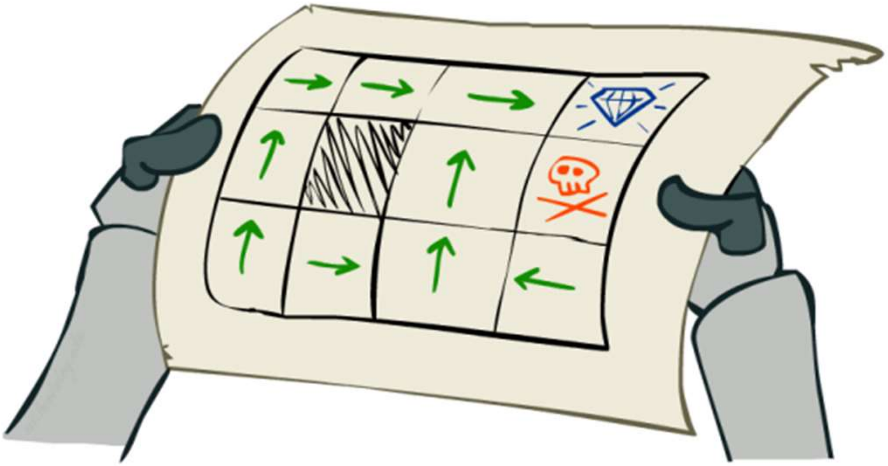


Markov Decision Process

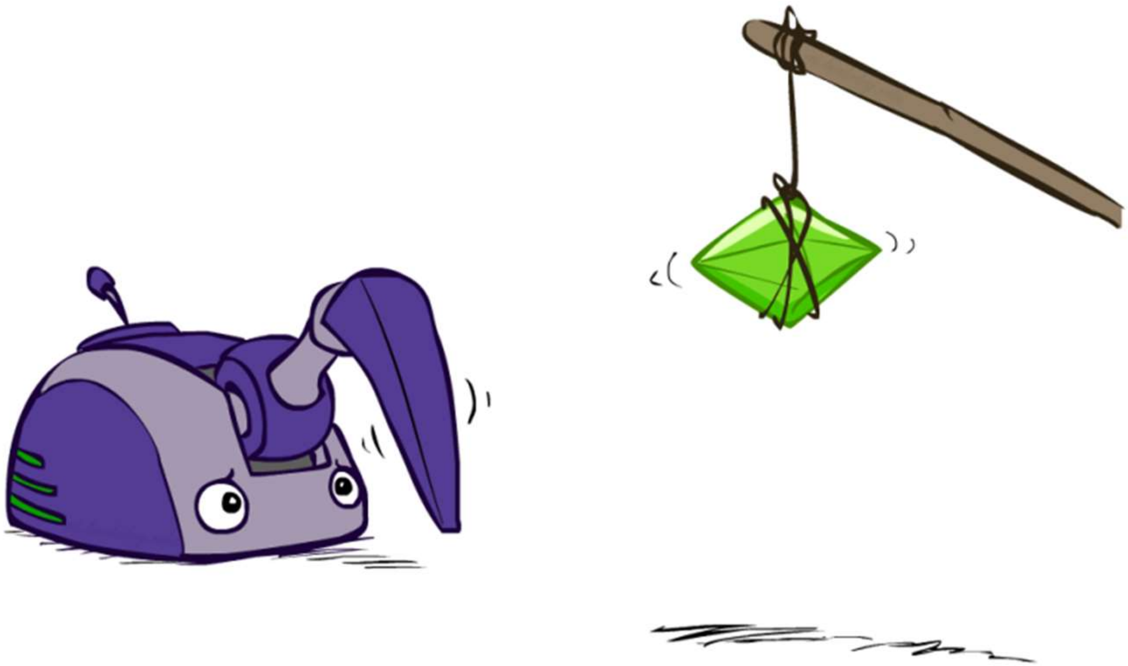
For MDPs, we want an optimal policy $\pi^*: S$

$\rightarrow A$

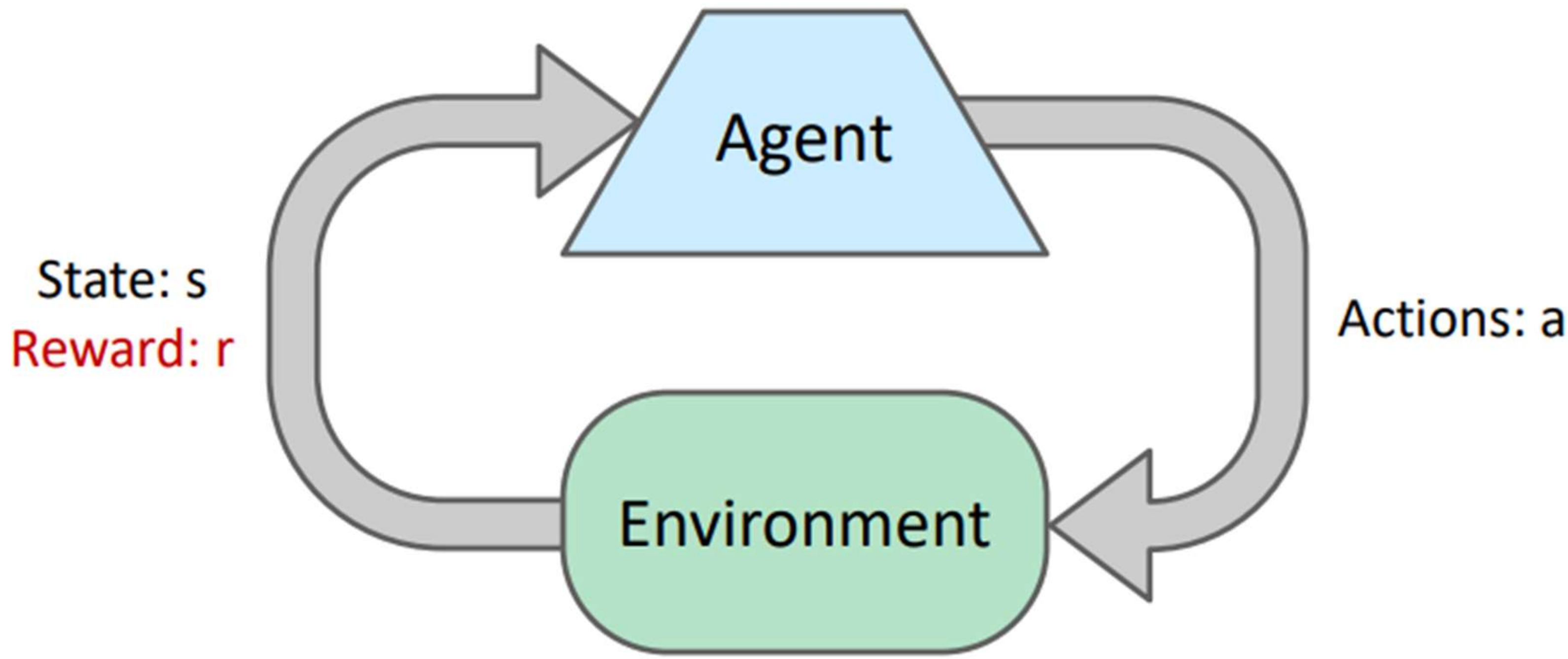
- A policy gives an action for each state
- An optimal policy is one that maximizes expected utility if followed
- An explicit policy defines a reflex agent



Reinforcement Learning



Reinforcement Learning



Reinforcement Learning

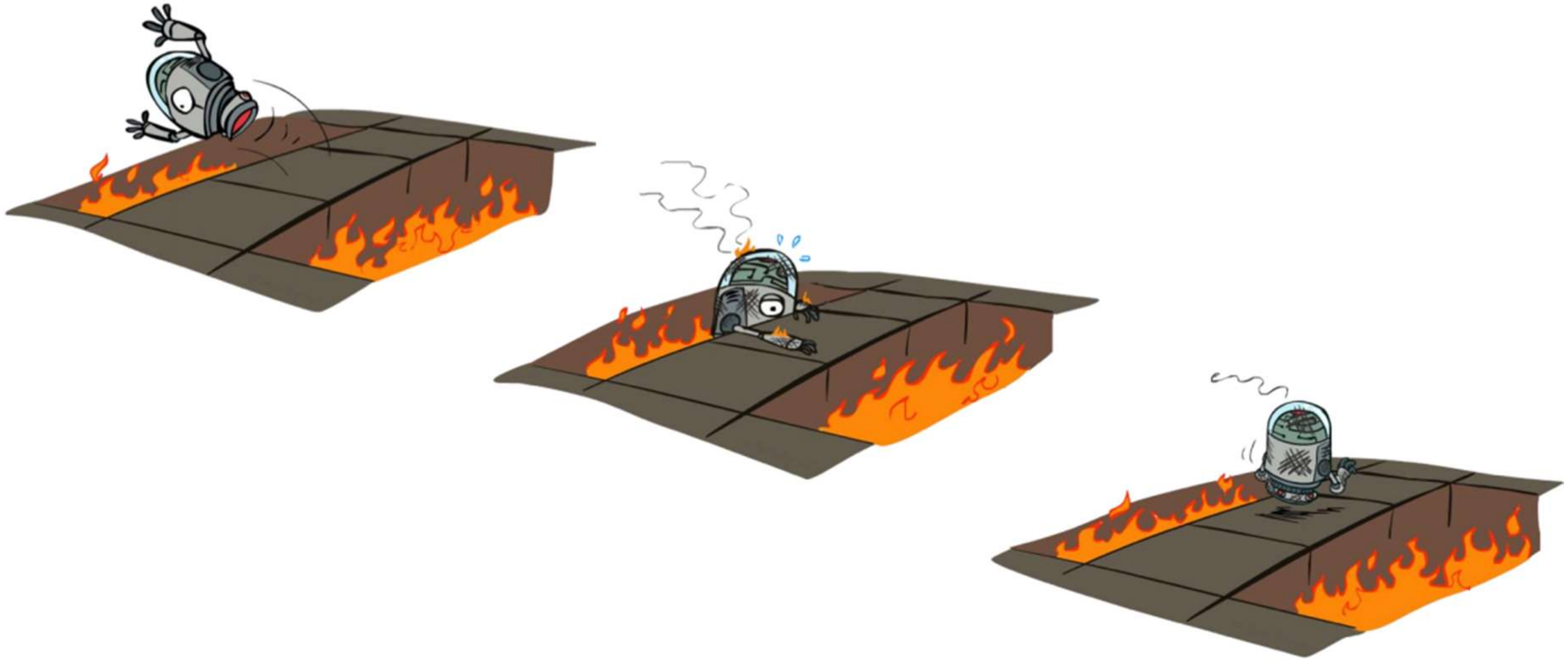
Still assume a Markov decision process (MDP):

- A set of states $s \in S$
- A set of actions (per state) A
- A model T(s,a,s') A reward function R(s,a,s')
- Still looking for a policy (s)

New twist:

- Don't know T or R I.e. we don't know which states are good or what the actions do
- Must actually try actions and states out to learn

Active Reinforcement Learning



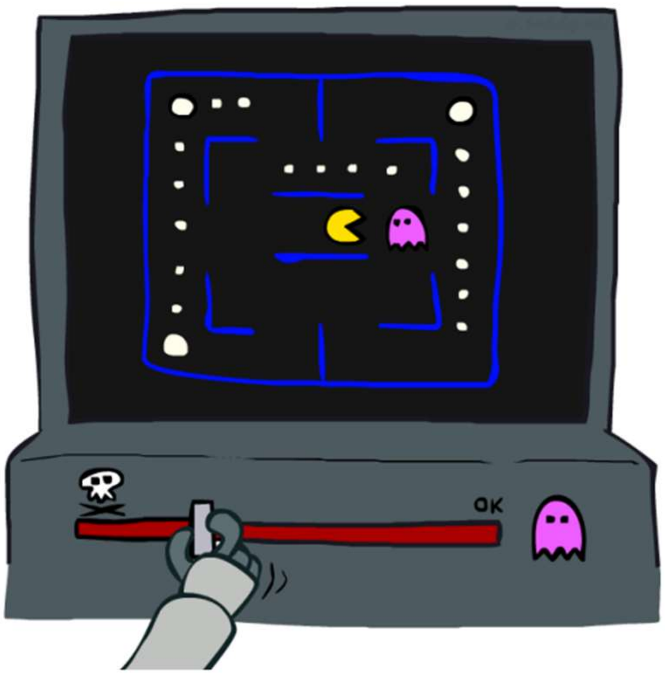
Active Reinforcement Learning

- You don’t know the transitions T(s,a,s’)
- You don’t know the rewards R(s,a,s’)
- You choose the actions now
- Goal: learn the optimal policy / values

In this case:

- Learner makes choices!
- Fundamental tradeoff: exploration vs. exploitation
- This is NOT offline planning! You actually take actions in the world and find out what happens

Approximate Q-Learning



Approximate Q-Learning

Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Learn Q(s,a) values as you go Receive a sample (s,a,s’,r)

$$\boxed{Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)}$$

Why Q-Learning?

Q-learning converges to optimal policy -- even if we're acting suboptimally!

This is called off-policy l

RELATED WORK

Learning to play Monopoly: A Reinforcement Learning approach

Learning to play Monopoly: A RL approach

- Modelled Monopoly as an MDP
- RL agent
- Random Agent
- Fixed policy Agent

MODELLING THE GAME

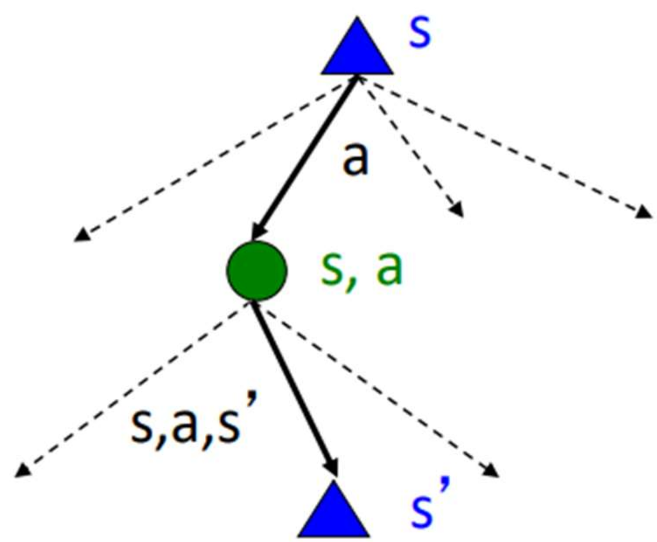
- Monopoly as a Reinforcement Learning Task
- Modeling Monopoly as an MDP
- Goals for RL Agent
- Reward Model

Monopoly as a Reinforcement Learning Task

- We attempt to model Monopoly as a single-agent RL task
- Despite the non-stationarity of the multiplayer Monopoly game invalidating most of the single-agent RL theoretical guarantees, single-agent RL algorithms have been extensively used in the literature in natively multi-agent settings
- Thus, this is considered suitable for a first approach on modelling Monopoly as a RL task

Modeling Monopoly as an MDP

- We formulate the state st as a 3-dimensional vector of objects containing information about the game's:
  - Area
  - Position
  - Finance current status , at time t
- Action Space

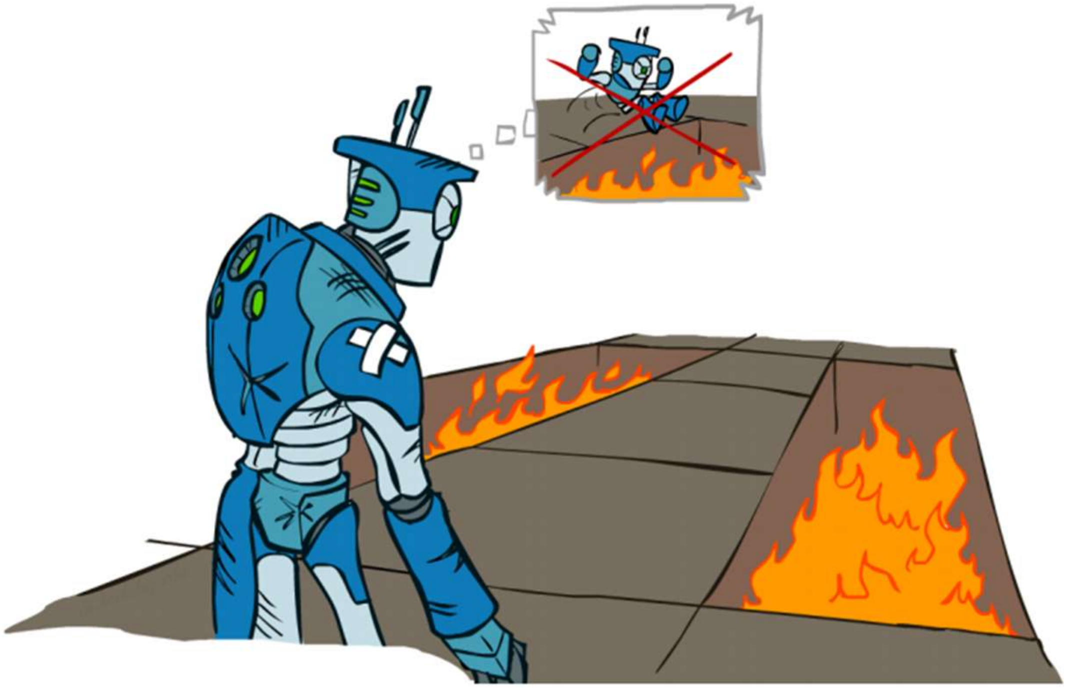


Modeling Monopoly as an MDP

- The area object, contains information about the game's properties, meaning the properties possessed from the current player and his opponents at time t
- The position variable determines the player's current position on the board in relation to its colour-group, scaled to [0,1]
- The finance vector consists of 2 values, specifying the current player's number of properties in comparison to those of his opponents' as well as a calculation of his current amount of money

Goals for RL Agent

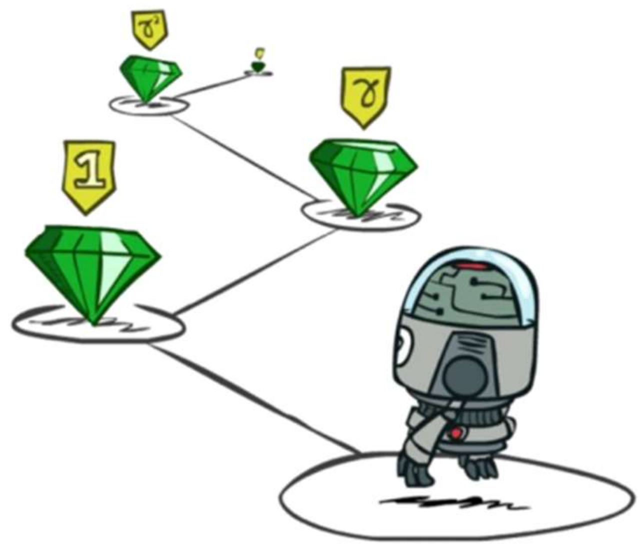
- Maximize Finance
- Minimize Regret
- Learn Optimum policies that Maximize finance and Minimize Regret
    - i.e. , learn which decisions are profitable, and which are not



Reward Model

The proposed reward model is represented by the following equation

$$r = \frac{\frac{v}{p} * c}{1 + |\frac{v}{p} * c|} + \frac{1}{p} * m;$$



- p is the number of players
- v is a quantity representing the player's total assets value
  (calculated by adding value of all properties in possession of player, minus properties of all opponents)
- m is a quantity representing the player's finance and is equal to the percentage of the money the player has to the sum of all players' money

IMPLEMENTATION

- Random Agent
- Fixed Policy Agent
- MonoRL

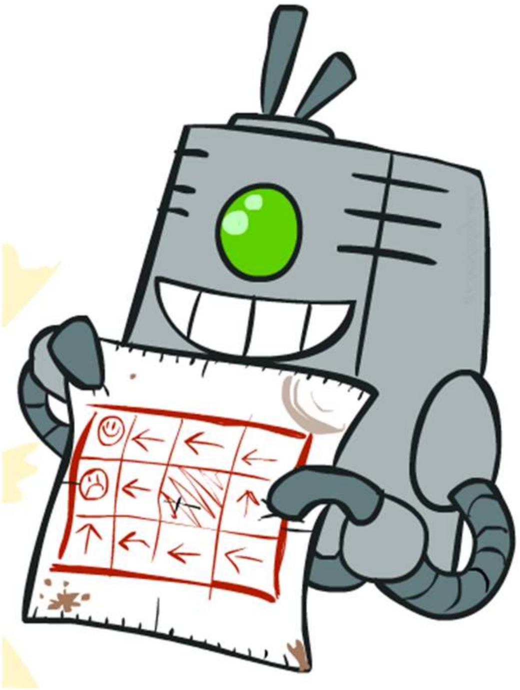
Q(λ) Learning Algorithm

1. $\hat{Q}(x,a) = 0$ and $Tr(x,a) = 0$ for all x and a
2. Do Forever:
   (A) $x_t \leftarrow$ the current state
   (B) Choose an action a_t according to current exploration policy
   (C) Carry out action a_t in the world. Let the short-term reward be r_t, and the new state be x_{t+1}
   (D) $e'_t = r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{Q}_t(x_t, a_t)$
   (E) $e_t = r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)$
   (F) For each state-action pair (x, a) do
   - $Tr(x,a) = \gamma\lambda Tr(x,a)$
   - $\hat{Q}_{t+1}(x,a) = \hat{Q}_t(x,a) + \alpha Tr(x,a) e_t$

   (G) $\hat{Q}_{t+1}(x_t, a_t) = \hat{Q}_{t+1}(x_t, a_t) + \alpha e'_t$
   (H) $Tr(x_t, a_t) = Tr(x_t, a_t) + 1$

Random Agent

- Random Agent is a random player whose actions are selected randomly ignoring the state signal.
- Chooses an action randomly from permitted actions



Fixed Policy Agent

- Fixed-policy player: Action selection is based on the money possessed.
- It sells if it has less than 150, buys when it has more than 350 and does nothing between.
- These thresholds were tuned appropriately for the best performance.
- Obtained from Reference paper

MonoRL

- RL agent with action selection based on the ε-greedy algorithm
- Performs optimum action with large probability 1- ε
- Performs random action with probability ε
- Exploration for better performance.

MonoRL – Parameters Used

- Gamma = 0.95 = Discount factor
- Lambda = 0.85
- Q-learning rate = 0.2
- Neural Network Learning Rate = 0.2

EXPERIMENTAL RESULTS

- Random Agent
- Fixed Policy Agent
- MonoRL

Results

Our Results:

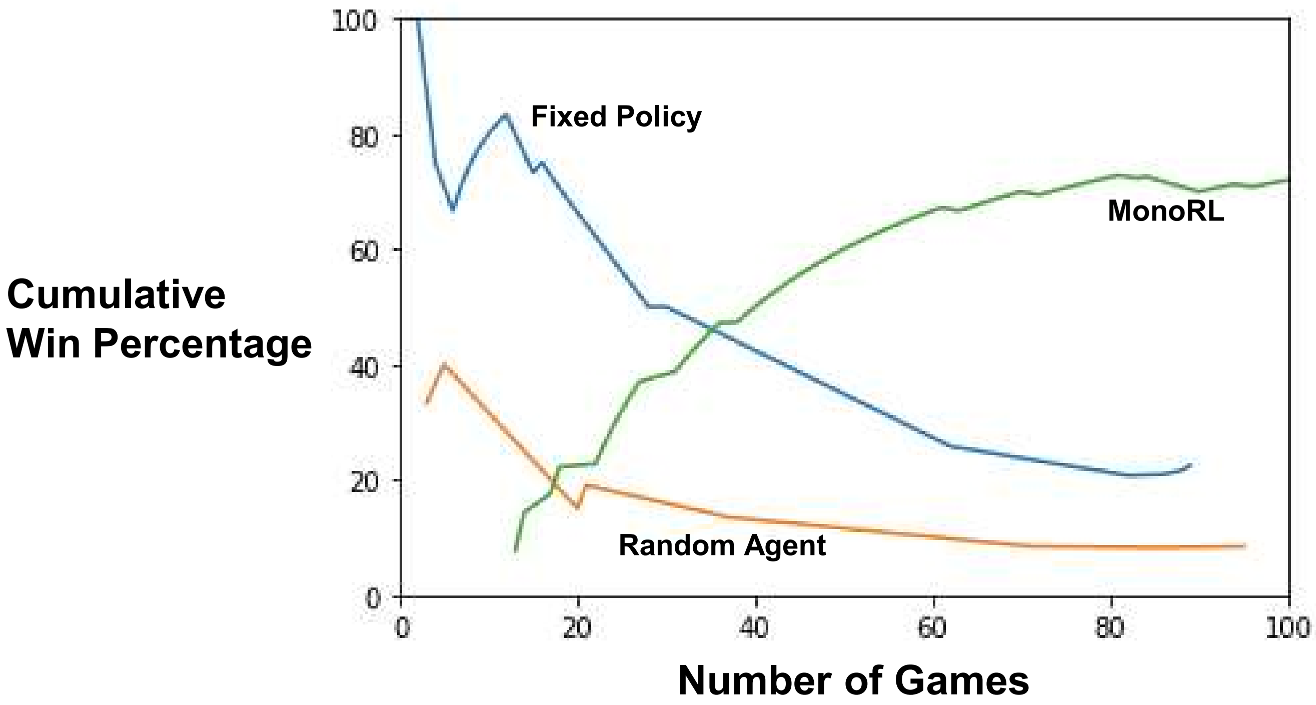
Relative Assets ~ 43% | Relative Money ~ 47%

- 100 Game runs
- Won by MonoRL: 61
- Won by Fixed Policy: 30
- Won by Random: 9
- Win % = 61%

Paper's Results

- 1000 Game runs
- Won by MonoRL: 694
- Won by Fixed Policy: 286
- Won by Random: 20
- Win % = 69.4%

Results



~700 Turns per game
(~3 * 2000 throws)

FUTURE SCOPE

Adversarial Q Learning

Adversarial Q Learning? (Evil9000?)

Take actions that maximize self Q values, while simultaneously minimizing Q Values of Opponents

Would it be as effective?

TECHNOLOGIES

Python: Most of the Modelling and Logic

Scikit Learn: Learning Q Values

CODE REVIEW

- Code Structure
- Code Walkthrough

Code Structure

Directories:

1. **Classes:** Models for the players, game and its components
2. **Data:** Command and Property Cards in XML format
3. **HelperUtils:** A few helper classes we built
4. **MonopolyHandlers:** Handles the rule based logic for the game
5. **RLClasses:** Actions, and observation data about the RL Agents (Finance, Area, Position)
6. **RLHandlers:** RL Agent and the RL Environment

CODE WALKTHROUGH

THANK YOU