OPTIMIZING RESOURCE PRESERVATION THROUGH DECEPTIVE MANEUVERING: A MOVER'S PLAYBOOK

Strategies for Deception and Resource Optimization: Applications in Dynamic Games and Systems

1st Dhanush Kumar Antharam
dept. electrical and computer engineering
University of North Carolina Charlotte
Charlotte, USA
danthara@uncc.edu

2nd Uma Nagirireddi
dept. electrical and computer engineering
University of North Carolina Charlotte
Charlotte, USA
unagirir@uncc.edu

3rd Kusuma Veerapalli
dept. electrical and computer engineering
University of North Carolina Charlotte
Charlotte, USA
kveerapa@uncc.edu

4th 3. Sai Kishan Kapavarapu
dept. electrical and computer engineering
University of North Carolina Charlotte
Charlotte, USA
skapavar@uncc.edu

***Abstract*—This research investigates the dynamics of deception in a strategically complex game, termed the Mover project. In this two-player game, the Mover, who knows the location of the true goal, seeks to retrieve resources while minimizing their consumption by a competing player, the Eater. The Eater, conversely, attempts to maximize consumption by predicting the Mover's goal. Utilizing a novel approach that abandons traditional estimative techniques, our study integrates real-time strategic responses into the game's design. This allows for a direct assessment of deception's efficacy through observed outcomes, rather than inferred predictions. The research demonstrates that strategic ambiguity, maintained even in adverse scenarios, can significantly enhance resource protection. The results provide novel insights into the application of deception in dynamic strategic contexts, presenting implications for both theory and real-world scenarios where deceptive strategies are pivotal.**

***Index Terms*—Monte Carlo First Visit,**

I. Introduction

Strategic deception plays a critical role in various competitive and adversarial contexts, ranging from security defenses to competitive sports and business negotiations. In these settings, the ability to mislead an opponent about one's true intentions or strategies can confer significant advantages. The Mover project introduces a dynamic game framework where deception is a central mechanism. Here, the Mover, with privileged information about the true goal location, aims to outmaneuver the Eater, who must decide where to maximize efforts based on observed movements. Unlike conventional game-theoretic models that often presume a static observer, this project models both players as active participants whose strategies evolve based on ongoing game dynamics. This approach provides a richer understanding of strategic deception under conditions where both players are continuously adapting to each other's moves.

Identify applicable funding agency here. If none, delete this.

II. Problem Statement

In the landscape of game theory, the utilization of dynamic strategies, especially involving deception, is often constrained by models that lack interactive complexity and realism. Most traditional approaches are rooted in assumptions where one player (observer or estimator) remains passively deceived without any real-time strategic adaptation. This passive role is unrepresentative of many real-world situations in which entities actively respond to and counteract deception based on ongoing interactions. The Mover project introduces a significant challenge by situating both participants—the Mover and the Eater—as active players within a dynamic game setting. The Mover, aware of the true goal location, seeks to retrieve resources while minimally alerting the Eater, who is intent on maximizing resource consumption based on the Mover's perceived actions. The key problem addressed here is the development of a robust strategic framework that enables the Mover to effectively conceal its goal amidst a constantly evolving game scenario. The

critical issues include: Developing a dynamic model that accommodates continuous strategic adaptations rather than static estimations. Ensuring that the model accurately represents scenarios with active deception where both players are continuously strategizing against each other.Formulating and validating deceptive strategies that are effective even when the opponent is actively trying to counteract them. Quantitatively measuring the effectiveness of deception in altering the game's outcome, which requires innovative approaches to game theory modeling.

III. Motivation

The motivation for delving into this research stems from the widespread relevance of deception in numerous strategic domains, such as military tactics, cybersecurity, competitive sports, and business strategy. In these fields, the ability to effectively mislead opponents or protect key information can dictate the success or failure of operations. The current theoretical models and strategies often fall short in dynamic situations where the opponent is neither static nor predictable.

The Mover project aims to bridge this gap by:

- Extending traditional game theory to include complex, interactive scenarios that better mirror real-world dynamics. This extension is crucial for developing strategies that are both theoretically sound and practically viable in scenarios where adversaries are intelligent and adaptive.
- Exploring the strategic use of information (or misinformation) in an environment where every action can lead to a series of reactive maneuvers. This exploration is pertinent to understanding how information asymmetry can be leveraged to one's advantage.
- Contributing to the foundational theories of behavioral game theory, where the focus shifts from purely rational models to those that consider the psychological and adaptive behaviors of opponents.
- Providing a framework that could inform the development of algorithms or protocols in security-sensitive applications, where deception is a key element of defense.
- Enhancing the predictive capabilities of strategic game models, thus offering better tools for training and simulation in contexts requiring high-stakes decision-making.

IV. Mover Environment

A. Detailed Overview of the Mover Environment Class

The MoverEnvironment class in Python simulates the environment for the Mover project, a strategic game involving a Mover and an Eater with goals and resources. This class is designed to manage the state of the game, player movements, resource tracking, and interactions with game objectives.

1) Class Attributes and Initialization

size: Represents the dimensions of the game grid. Default size is set to 7x7.

true goal and fake goal: Fixed locations on the grid representing the objectives. The true goal is located at (6, 6), while the fake goal is at (1, 5).

initial resources: High initial value of resources (100,000) assigned to both goals to simulate a resource-rich environment that the Eater attempts to deplete.

resources: A dictionary mapping each goal location to its corresponding resource count.

eater actions: A list of potential actions the Eater can take, expressed as tuples that represent the proportion of resources consumed from the fake and true goals.

eater policy: A dictionary that can define specific actions the Eater will take based on the game state and the Mover's actions; defaults to an empty dictionary if not provided.

moves: A list of possible movements ('Up', 'Down', 'Left', 'Right') the Mover can make on the grid.

visited fake goal: A boolean flag to check whether the fake goal has been visited at least once, which influences the game's scoring.

state: Current position of the Mover on the grid, initialized at (0, 0).

B. Methods

reset(): Resets the game environment to its initial state. Reinitializes the Mover's position and the resources at both goals. Returns the initial state of the Mover.

1) step(curr state, action):

Processes a single movement of the Mover based on the given action.Calculates the new state by applying the movement offset associated with the chosen action. Ensures that the Mover does not move out of the grid bounds. Determines the consumption of resources at both goals based on the Eater's actions, which are decided either by a policy or default settings.

2) Updates the game's state and manages interactions with the goals:

Rewards or penalties are assigned based on whether the Mover first visits the fake goal before reaching the true goal, revisits the fake goal, or depletes the resources at any goal. A significant reward is given for reaching the true goal after visiting the fake goal, reflecting successful deception and strategic movement. Penalties are applied for reaching the true goal without prior visit to the fake

goal or for depleting the resources, simulating a failure in resource management or strategic deception. Checks if the game should end, which occurs when the Mover reaches the true goal. Returns the new state, reward or penalty incurred, and a boolean indicating if the game has ended.

C. Detailed Breakdown of Step Function Logic Movement Calculation:

The Mover's next position is determined based on the current state and the action taken. Movement offsets for each action ('Up', 'Down', 'Left', 'Right') are applied to compute the new position.Boundary checks ensure the Mover remains within the grid limits.

1) Eater Interaction:

An index is calculated from the Mover's new state to look up the corresponding Eater action from the eater policy. If no specific action is predefined, a default action is chosen. Based on the selected Eater action, the resources at the true and fake goals are decremented accordingly, simulating the Eater consuming the resources.

2) Goal Interactions and Rewards:

If the Mover visits the fake goal for the first time, a significant reward is provided. Subsequent visits continue to give rewards but less than the first time, incentivizing strategic movement across the grid.

3) Reaching the true goal incorporates major game decisions:

A large penalty is imposed if the true goal is reached without first visiting the fake goal, reflecting a strategic failure. Conversely, a substantial reward is given for reaching the true goal after visiting the fake goal, symbolizing successful deception. Resource depletion at either goal triggers a severe penalty, adding a resource management aspect to the strategic gameplay.

4) Game Termination Check:

The game ends either when the Mover reaches the true goal or if the resources at either goal are depleted, with the latter scenario indicating a failure in resource conservation.

V. Algorithm

A. First-visit Monte Carlo

First-visit Monte Carlo is a model-free reinforcement learning algorithm. It does not require a model of the environment (i.e., the transition probabilities and reward functions do not need to be known). Instead, it learns directly from episodes of experience. Here's a detailed breakdown:

1) Initialization

Action-Value Function (Q): Before the learning process begins, initialize the action-value estimates, $Q(s, a)$, for all state-action pairs. These values are initially set to zero but are updated as learning progresses through the episodes. The Q-function estimates the expected return (total discounted future reward) of taking an action a in a state s and following a policy thereafter.

Returns: For each state-action pair, maintain a list that records all the returns (sum of discounted rewards) following the first visit to that state-action pair in each episode. These lists are used to calculate the average returns needed to update the Q values.

Policy: Initialize a policy (often randomly), which is a mapping from states to actions. This initial policy is iteratively improved based on the insights gained from the Q values.

2) Loop for Each Episode

- Generate an Episode:

  Episodes are sequences of states, actions, and rewards, which terminate when a final state is reached. The game begins in an initial state S_0 and an action is chosen and taken according to the ϵ-greedy policy: with probability ϵ, a random action is chosen (exploration), and with probability $1-\epsilon$, the action that maximizes the current Q estimate is chosen (exploitation). This ϵ-greedy approach ensures all state-action pairs are visited infinitely often, which is necessary for the convergence of the MC method.

- Calculate Returns:
  After generating an episode, traverse the episode from the end to the beginning to calculate the return for each state-action pair. This return is the sum of the rewards obtained after the first occurrence of each state-action pair, discounted by a factor γ at each timestep. Update the returns list for each visited state-action pair and then update the Q value for that pair to be the average of these returns. This step effectively implements the idea of averaging over all the sampled returns to estimate the expected return.

- Policy Improvement:
  With updated Q values, the policy can be improved by choosing the action that maximizes the Q value for each state. This is the essence of the policy improvement theorem, which states that the new policy will be as good as or better than the previous policy if it is greedy with respect to the action-value function derived from the previous policy.

- Output the Final Policy and Action-Value Function After the specified number of episodes, the algorithm outputs the final policy and Q values. These outputs represent the learned strategy and the corresponding value estimates, which guide the Mover's decisions to maximize its cumulative discounted reward.

3) Key Aspects and Benefits of the MC Method Model-Free:

- Directly learns from actual experiences without a model of the environment.
- Convergence: Under appropriate conditions (infinite visits to all state-action pairs), the MC method converges to the optimal policy and action-value function.
- Simplicity: Conceptually simple and easy to implement, especially when the environment's dynamics are unknown or difficult to model.

Algorithm Monte Carlo Control for Estimating Policy π

Initialize $Q(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
Initialize $\pi \leftarrow$ an arbitrary ϵ-soft policy
Initialize Returns$(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
loop for each episode
  Generate an episode following ϵ-greedy policy: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for $t = T-1$ downto 0 **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if (S_t, A_t) not in sequence $S_0, A_0, \ldots, S_{t-1}, A_{t-1}$ **then**
      Append G to Returns(S_t, A_t)
      $Q(S_t, A_t) \leftarrow$ average(Returns(S_t, A_t))
      $A^* \leftarrow \arg\max_a Q(S_t, a)$
      For all $a \neq A^*$, $\pi(a|S_t) \leftarrow \frac{\epsilon}{|\mathcal{A}(S_t)|}$
      $\pi(A^*|S_t) \leftarrow 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(S_t)|}$
    end if
  end for
end loop
return π, Q =0

B. Semi Gradient TD algorithm

$TD(0)$ - Learning for Estimating Q-values. Objective is to learn an optimal action-value function $Q(s,a)$ that estimates the expected return for taking an action a in state s and following a policy thereafter.

1) Inputs:
env: The game environment, which encapsulates the dynamics including states, actions, and rewards.
episodes: Total number of episodes to run for the TD learning.
alpha: Learning rate (0.1), which determines the step size in the updating process.
gamma: Discount factor (0.99), which discounts future rewards.
epsilon: Probability of choosing a random action (0.1), facilitating exploration.

2) Outputs:
q values: A 3D numpy array containing the learned Q-values for each state-action pair.

3) Detailed Steps of the TD(0) Algorithm:
Initialization:
Initialize the q values array to zero for all state-action pairs in the environment. This array stores the estimated Q-values, which are updated iteratively throughout the learning process.

- Loop Over Episodes:
- Repeat for each episode:
- Initialize the Episode:
- Start by resetting the environment to get the initial state.
- Continue the episode until a terminal state is reached (done flag returns True).
- Choose and Execute Actions:
- For each step within an episode:

Action Selection (Epsilon-Greedy Policy):
Choose an action using an epsilon-greedy approach, where with probability epsilon a random action is chosen (exploration), and with probability 1 - epsilon the action that currently has the highest estimated Q-value in the current state is chosen (exploitation).
Execute the Action:
Apply the selected action to the environment, which returns the next state, the reward obtained, and whether the episode has ended.

- TD(0) Update:
- Calculate TD Target and TD Error:
- Identify the action with the highest Q-value in the next state (greedy action).
- Compute the TD Target as the reward received plus the discounted value of the estimated Q-value of the next state and the greedy action.

- Calculate the TD Error as the difference between the TD Target and the current Q-value for the state-action pair.
- Update Q-value: Adjust the Q-value for the current state-action pair by moving it a step alpha towards the TD Error.
- Transition to Next State:
- Update the current state to the next state to continue the episode.
- Return Q-values: After all episodes are completed, the final Q-values are returned. These values represent the learned policy's action preferences in various states.

4) Key Aspects of TD Learning:

- Bootstrap: Unlike Monte Carlo methods, TD learning updates estimates based partially on other learned estimates (bootstrapping), without waiting for a final outcome. This makes it suitable for non-terminating environments and faster learning over partial sequences.
- Update Frequency: Updates occur after every step within an episode, which allows faster learning and adaptation to changing dynamics in the environment.
- Efficiency: TD methods are generally more sample-efficient compared to Monte Carlo methods.

Algorithm Temporal Difference Learning (TD(0)) for Policy Evaluation

0: Initialize $Q(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ to zero.
0: Initialize a random policy π mapping states to actions.
0: Initialize ϵ for ϵ-greedy policy.
0: **loop** for each episode
0: Initialize state S_0
0: **for** each step of the episode until termination **do**
0: Choose action A_t from S_t using policy π derived from Q (e.g., ϵ-greedy)
0: Take action A_t, observe reward R_{t+1}, and next state S_{t+1}
0: $a' \leftarrow \arg\max_a Q(S_{t+1}, a)$ (best next action)
0: TD target $\leftarrow R_{t+1} + \gamma Q(S_{t+1}, a')$
0: TD error $\leftarrow$ TD target $-Q(S_t, A_t)$
0: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \times$ TD error
0: **end for**
0: Update policy π for each S_t visited during the episode:
0: $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
0: **end loop**
0: **return** policy π and action-value function Q =0

VI. Results

1) Experiment Results - 1

Step	Current State	Action	Next State	Reward	Resources After Action
1	(0, 0)	Up	(0, 1)	-1	gR: 99.8, gF: 99.2
2	(0, 1)	Up	(0, 2)	-1	gR: 99.5, gF: 98.5
3	(0, 2)	Right	(1, 2)	-1	gR: 99.3, gF: 97.7
4	(1, 2)	Up	(1, 3)	-1	gR: 99.1, gF: 96.9
5	(1, 3)	Up	(1, 4)	-1	gR: 99.1, gF: 95.9
6	(1, 4)	Up	(1, 5)	-1	gR: 98.8, gF: 95.2
7	(1, 5)	Right	(2, 5)	49	gR: 98.3, gF: 94.7
8	(2, 5)	Right	(3, 5)	-1	gR: 97.7, gF: 94.3
9	(3, 5)	Right	(4, 5)	-1	gR: 97.3, gF: 93.7
10	(4, 5)	Up	(4, 6)	-1	gR: 96.8, gF: 93.2
11	(4, 6)	Right	(5, 6)	-1	gR: 96.5, gF: 92.5
12	(5, 6)	Right	(6, 6)	814	gR: 95.7, gF: 92.3

TABLE I
Monte Carlo Simulation Results

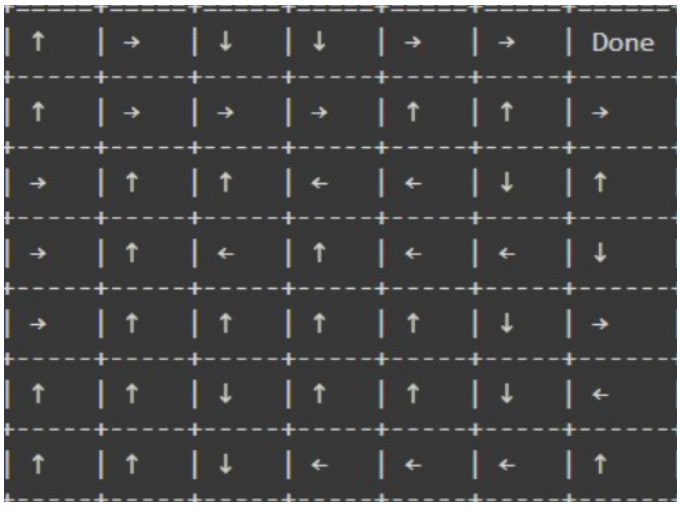

Fig. 1. Mover Policy

Step	Current State	Action	Next State	Reward	Resources After Action
1	(0, 0)	Up	(0, 1)	-1	gR: 99.8, gF: 99.2
2	(0, 1)	Up	(0, 2)	-1	gR: 99.5, gF: 98.5
3	(0, 2)	Up	(0, 3)	-1	gR: 98.9, gF: 98.1
4	(0, 3)	Right	(1, 3)	-1	gR: 98.3, gF: 97.7
5	(1, 3)	Up	(1, 4)	-1	gR: 98.3, gF: 96.7
6	(1, 4)	Up	(1, 5)	-1	gR: 98.0, gF: 96.0
7	(1, 5)	Right	(2, 5)	49	gR: 97.5, gF: 95.5
8	(2, 5)	Right	(3, 5)	-1	gR: 96.9, gF: 95.1
9	(3, 5)	Up	(3, 6)	-1	gR: 96.4, gF: 94.6
10	(3, 6)	Right	(4, 6)	-1	gR: 96.1, gF: 93.9
11	(4, 6)	Right	(5, 6)	-1	gR: 95.8, gF: 93.2
12	(5, 6)	Right	(6, 6)	814	gR: 95.0, gF: 93.0

TABLE II
SEMI GRADIENT TD SIMULATION RESULTS

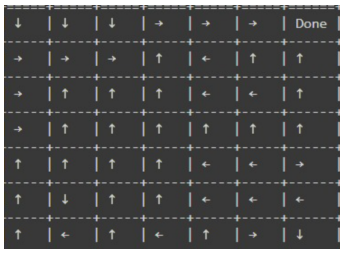

Fig. 3. Mover Policy

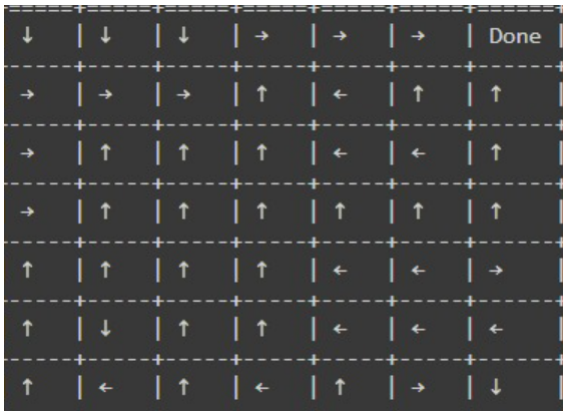

Fig. 2. Mover Policy

Step	Current State	Action	Next State	Reward	Resources After Action
1	(0, 0)	Up	(0, 1)	-1	gR: 99.7, gF: 99.3
2	(0, 1)	Up	(0, 2)	-1	gR: 99.4, gF: 98.6
3	(0, 2)	Up	(0, 3)	-1	gR: 99.4, gF: 97.6
4	(0, 3)	Up	(0, 4)	-1	gR: 99.2, gF: 96.8
5	(0, 4)	Up	(0, 5)	-1	gR: 99.0, gF: 96.0
6	(0, 5)	Right	(1, 5)	-1	gR: 98.8, gF: 95.2
7	(1, 5)	Right	(2, 5)	49	gR: 98.6, gF: 94.4
8	(2, 5)	Right	(3, 5)	-1	gR: 98.4, gF: 93.6
9	(3, 5)	Up	(3, 6)	-1	gR: 98.2, gF: 92.8
10	(3, 6)	Right	(4, 6)	-1	gR: 97.5, gF: 92.5
11	(4, 6)	Right	(5, 6)	-1	gR: 96.8, gF: 92.2
12	(5, 6)	Right	(6, 6)	814	gR: 96.1, gF: 91.9

TABLE IV
TD LEARNING RESULTS

3) Experiment Results - 3

Step	Current State	Action	Next State	Reward	Resources After Action
1	(0, 0)	Right	(1, 0)	-1	gR: 99.7, gF: 99.3
2	(1, 0)	Up	(1, 1)	-1	gR: 99.4, gF: 98.6
3	(1, 1)	Right	(2, 1)	-1	gR: 99.1, gF: 97.9
4	(2, 1)	Up	(2, 2)	-1	gR: 98.9, gF: 97.1
5	(2, 2)	Up	(2, 3)	-1	gR: 98.7, gF: 96.3
6	(2, 3)	Up	(2, 4)	-1	gR: 98.5, gF: 95.5
7	(2, 4)	Up	(2, 5)	49	gR: 98.5, gF: 94.5
8	(2, 5)	Up	(2, 6)	-1	gR: 97.8, gF: 94.2
9	(2, 6)	Right	(3, 6)	-1	gR: 97.1, gF: 93.9
10	(3, 6)	Right	(4, 6)	-1	gR: 96.4, gF: 93.6
11	(4, 6)	Right	(5, 6)	-1	gR: 95.7, gF: 93.3
12	(5, 6)	Right	(6, 6)	814	gR: 94.9, gF: 93.1

TABLE V
MC FIRST VISIT SIMULATION RESULTS

2) Experiment Results - 2

Step	Current State	Action	Next State	Reward	Resources After Action
1	(0, 0)	Right	(1, 0)	-1	gR: 99.7, gF: 99.3
2	(1, 0)	Right	(2, 0)	-1	gR: 99.4, gF: 98.6
3	(2, 0)	Right	(3, 0)	-1	gR: 99.1, gF: 97.9
4	(3, 0)	Up	(3, 1)	-1	gR: 98.8, gF: 97.2
5	(3, 1)	Left	(2, 1)	-1	gR: 98.5, gF: 96.5
6	(2, 1)	Up	(2, 2)	-1	gR: 98.2, gF: 95.8
7	(2, 2)	Right	(3, 2)	-1	gR: 98.2, gF: 94.8
8	(3, 2)	Up	(3, 3)	-1	gR: 98.0, gF: 94.0
9	(3, 3)	Up	(3, 4)	-1	gR: 97.8, gF: 93.2
10	(3, 4)	Left	(2, 4)	-1	gR: 97.6, gF: 92.4
11	(2, 4)	Up	(2, 5)	49	gR: 97.4, gF: 91.6
12	(2, 5)	Up	(2, 6)	-1	gR: 97.2, gF: 90.8
13	(2, 6)	Right	(3, 6)	-1	gR: 96.5, gF: 90.5
14	(3, 6)	Right	(4, 6)	-1	gR: 95.8, gF: 90.2
15	(4, 6)	Right	(5, 6)	-1	gR: 95.1, gF: 89.9
16	(5, 6)	Right	(6, 6)	814	gR: 94.4, gF: 89.6

TABLE III
MONTE CARLO SIMULATION RESULTS

Fig. 4. Mover Policy

Step	Current State	Action	Next State	Reward	Resources After Action
1	(0, 0)	Right	(1, 0)	-1	gR: 99.7, gF: 99.3
2	(1, 0)	Right	(2, 0)	-1	gR: 99.4, gF: 98.6
3	(2, 0)	Right	(3, 0)	-1	gR: 99.1, gF: 97.9
4	(3, 0)	Up	(3, 1)	-1	gR: 98.8, gF: 97.2
5	(3, 1)	Right	(4, 1)	-1	gR: 98.5, gF: 96.5
6	(4, 1)	Right	(5, 1)	-1	gR: 98.3, gF: 95.7
7	(5, 1)	Up	(5, 2)	-1	gR: 98.1, gF: 94.9
8	(5, 2)	Right	(6, 2)	-1	gR: 97.9, gF: 94.1
9	(6, 2)	Up	(6, 3)	-1	gR: 97.7, gF: 93.3
10	(6, 3)	Up	(6, 4)	-1	gR: 97.7, gF: 92.3
11	(6, 4)	Up	(6, 5)	-1	gR: 97.5, gF: 91.5
12	(6, 5)	Up	(6, 6)	-101	gR: 96.8, gF: 91.2

TABLE VI
SEMI GRADIENT TD SIMULATION RESULTS

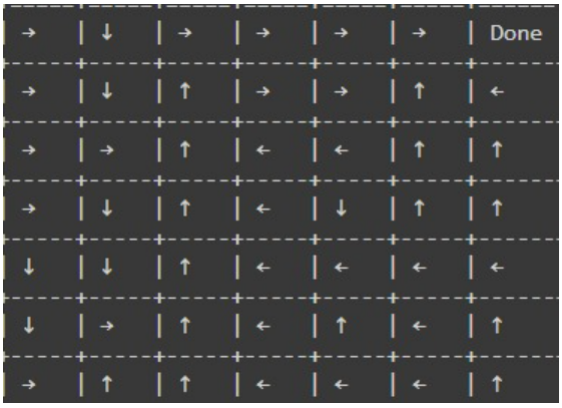

Fig. 5. Mover Policy

VII. CONCLUSION

In summary, from the exploration of the **Mover** project and the implementation of both the *Monte Carlo* and *Temporal Difference Learning* methods, several key insights emerge:

- **Model-Free Learning:** Both the Monte Carlo and TD Learning methods are model-free, meaning they do not require prior knowledge of the environment's dynamics. This makes them versatile and powerful for exploring unknown environments, such as the strategic game setting in the Mover project.
- **Dynamic Policy Improvement:** Both algorithms are designed to improve the policy dynamically based on the rewards received and the estimates of future states. Monte Carlo does this at the end of each episode, while TD Learning updates its estimates at each step, offering a more immediate integration of new information.
- **Exploration vs. Exploitation:** Utilizing an ϵ-greedy strategy ensures a balance between exploration (learning about new actions and states) and exploitation (leveraging known information to maximize rewards), which is crucial for developing robust strategies in dynamic and uncertain environments.
- **Efficiency and Responsiveness:** TD Learning particularly stands out for its efficiency and responsiveness, updating policies on a step-by-step basis, which is more suited for environments where the conditions change rapidly.
- **Practical Applications:** The learnings from implementing these algorithms in the Mover project extend beyond gaming into areas such as robotics, logistics, and automated decision systems, where adaptive strategies are crucial.

In conclusion, the integration of Monte Carlo and TD Learning methods within the Mover project framework provides valuable methodologies for enhancing decision-making processes in complex, dynamic environments. These techniques not only foster a deeper understanding of strategic interactions under uncertainty but also pave the way for more advanced developments in artificial intelligence and machine learning.

VIII. REFERENCE

[1]V. Rostobaya, Y. Guan, J. Berneburg, M. Dorothy and D. Shishika, "Deception by Motion: The Eater and the Mover Game," in IEEE Control Systems Letters, vol. 7, pp. 3157-3162, 2023, doi: 10.1109/LCSYS.2023.3291385. keywords: Games;Observers;Linear programming;Motion control;Measurement;Dynamic scheduling;Tracking;Agent-based systems;deception;game theory,

IX. ANNOTATIONS

gR - Real Goal at (6,6)
gF - Fake Goal at (1,5)
in a $7x7$ grid.