Report: Exploration Strategies

Introduction

In the context of the grid-based game where the goal is for the agent (the snake) to find an apple, we explored several movement strategies to identify the most efficient approach. Specifically, we investigated the effectiveness of three exploratory strategies: **Randomized Walk**, **Spiral Walk**, and **Zigzag Walk**. Each of these methods was tested under the assumption that the agent needs to satisfy the constraint of locating the apple within a maximum of 35 steps, which we refer to as the **35S constraint**. For these 3 strategies, they fail short. So I thought of trying **Reinforcement learning** with designing the reward function to get a working policy as another strategy and it worked.

1. Randomized Walk

The **Randomized Walk** strategy involves the agent making random choices at each step. While this approach allows for exploration in all directions, it is inherently inefficient. Since the movements are random, the agent often revisits previously explored areas, leading to longer and more unpredictable paths. Consequently, the agent fails to converge towards the apple in a timely manner. The randomized nature of the walk makes it highly unlikely that the agent will find the apple within the 35 steps, as the movement does not follow any optimal search pattern.

Outcome:
Despite covering a large portion of the grid, the agent frequently fails to satisfy the 35S constraint. The random nature of the movements results in many redundant steps, increasing the likelihood of overshooting or missing the apple.

2. Spiral Walk

The **Spiral Walk** strategy was designed to systematically cover the grid in an expanding spiral pattern. The idea is that this method ensures the agent gradually explores all regions of the grid, starting from a central point and moving outward. While it is more structured than the Randomized Walk, it still does not guarantee optimal performance. The spiral pattern often leads the agent to move too far away from the apple, and because it covers the grid in a sequential manner, the agent may end up overshooting or not efficiently locating the apple.

Outcome:
Although the Spiral Walk provides better coverage of the grid, it still fails to satisfy the 35S constraint. The method does not adapt to the changing conditions of the game (i.e., the position of the apple), and the rigid pattern results in unnecessary steps, ultimately leading to suboptimal performance.

3. Zigzag Walk

The **Zigzag Walk** strategy involves the agent moving in alternating horizontal or vertical paths, essentially creating a zigzag pattern. This approach also ensures broad coverage of the grid but, like the Spiral Walk, is limited by its lack of adaptability. The agent is still following a fixed pattern, and while it covers the grid more evenly than the Spiral Walk, it does not adjust its search direction based on the agent's proximity to the apple.

Outcome:
The Zigzag Walk also fails to meet the 35S constraint. Similar to the Spiral Walk, it does not offer any strategic focus towards the goal and leads to inefficient exploration patterns that may result in wasted steps.

4. Introducing Q-Learning for Optimal Exploration

Given the failure of the aforementioned strategies to meet the 35S constraint, we turned to **Q-Learning** as a potential solution. Q-Learning is a model-free reinforcement learning algorithm that helps the agent learn the most optimal policy by interacting with the environment and adjusting actions based on the rewards received. The core advantage of Q-Learning lies in its ability to adapt to dynamic environments and optimize the agent's actions based on the state and reward structure.

By using Q-Learning, the agent can iteratively improve its decision-making process, rather than following predetermined patterns like in the Spiral or Zigzag Walk. With Q-Learning, the agent updates its action policy based on the rewards received for reaching certain states, eventually converging towards the most efficient path to the apple.

The main advantage of using Q-Learning in this scenario is its ability to learn from experience, ensuring that the agent prioritizes actions that minimize steps and maximize the chances of finding the apple within the 35S constraint. Through exploration and exploitation, the Q-Learning agent can adjust its behavior dynamically, making it far more efficient than random or structured walk strategies.

Implementation:

The winning strategy hinges on **tracking the sequence of recent actions** (up to the last three) and using this as the state representation for Q-learning. Here's why this approach is well-suited for the problem of navigating a wraparound grid as a snake trying to find an apple:

Compact State Representation

- **Why it works:** Instead of representing the full grid (which could be computationally expensive, especially for large or dynamically sized grids), the state is based on the *history of actions*. This leverages the fact that in a wraparound grid, the current position and heading direction can be inferred implicitly through recent movements.
- **Advantage:** This drastically reduces the state space, making Q-learning more efficient in terms of memory and computation.

Dynamic Adaptation

- **Why it works:** The wraparound nature of the grid eliminates boundaries, making movement repetitive and cyclic if not controlled. By maintaining a history of actions, the agent avoids repeating ineffective patterns (e.g., going in circles) and learns to explore new paths toward the apple.
- **Advantage:** The agent can dynamically adapt to the changing environment (random apple placements) without explicitly needing the apple's coordinates.

Reward Design

- **Step Penalty:** Encourages shorter paths by penalizing each step.
- **Apple Reward:** Provides a strong positive signal for successfully reaching the apple, ensuring that the agent prioritizes this goal.
- **Overstep Penalty:** Discourages excessively long searches, forcing the agent to learn efficient trajectories.

Generalization for Unknown Grid Sizes

- **Why it works:** The strategy is agnostic to the grid's dimensions. It doesn't depend on positional information or predefined boundaries. Instead, it relies on the sequence of actions and the rewards received.
- **Advantage:** Makes the approach scalable and adaptable to grids of varying sizes without any additional modifications.

Policy


```
Optimal_policy = {
 (): {'UP': 0.0, 'DOWN': 0.0, 'LEFT': 0.0, 'RIGHT': -0.1},
 ('RIGHT',): {'UP': 0.0, 'DOWN': 0.0, 'LEFT': 0.0, 'RIGHT': -0.1},
 ('RIGHT', 'RIGHT'): {'UP': 0.0, 'DOWN': 0.0, 'LEFT': 0.0, 'RIGHT': -0.1},
 ('RIGHT', 'RIGHT', 'RIGHT'): {'UP': 171.4695259214313, 'DOWN': 171.1724558022362, 'LEFT': 172.24941233620282, 'RIGHT': 171.84696044594395},
 ('RIGHT', 'RIGHT', 'LEFT'): {'UP': 195.7114637119111, 'DOWN': 196.21856594799388, 'LEFT': 197.7568119645967, 'RIGHT': 185.08102364049827},
 ('RIGHT', 'LEFT', 'LEFT'): {'UP': 168.59911941121587, 'DOWN': 168.2889395584269, 'LEFT': 167.9563760641255, 'RIGHT': 167.65919017022668},
 ('LEFT', 'LEFT', 'RIGHT'): {'UP': 178.0878099358454, 'DOWN': 157.36546818775926, 'LEFT': 177.76843517885285, 'RIGHT': 178.01312676173566},
 ('LEFT', 'RIGHT', 'LEFT'): {'UP': 268.8673553484859, 'DOWN': 152.1564943822199, 'LEFT': 163.7533214870835, 'RIGHT': 162.894713538009105},
 ('RIGHT', 'LEFT', 'RIGHT'): {'UP': 125.56504705464003, 'DOWN': 228.732770930930003, 'LEFT': 125.73382241533474, 'RIGHT': 133.368940887743},
 ('RIGHT', 'LEFT', 'UP'): {'UP': 172.16637598476183, 'DOWN': 171.85357596974578, 'LEFT': 170.635246540783, 'RIGHT': 171.79420951047211},
 ('LEFT', 'UP', 'LEFT'): {'UP': 155.94629916841328, 'DOWN': 156.8579304918671, 'LEFT': 155.50974555661503, 'RIGHT': 155.207051728745775},
 ('UP', 'LEFT', 'DOWN'): {'UP': 180.38001083700266, 'DOWN': 319.9879740417836, 'LEFT': 237.06217882924017, 'RIGHT': 246.63659514951314},
 ('LEFT', 'DOWN', 'LEFT'): {'UP': 270.4498268103302, 'DOWN': 280.384511533049, 'LEFT': 278.5587137903416, 'RIGHT': 279.8678505905368},
 ('DOWN', 'LEFT', 'LEFT'): {'UP': 241.69009830635318, 'DOWN': 245.65445817448685, 'LEFT': 245.15492198538254, 'RIGHT': 244.58857960681956},
 ('LEFT', 'LEFT', 'DOWN'): {'UP': 213.19793894003845, 'DOWN': 217.2617917630228, 'LEFT': 218.08288233766662, 'RIGHT': 216.560557551276455},
 ('LEFT', 'DOWN', 'DOWN'): {'UP': 269.6142996642989, 'DOWN': 249.6563660006079, 'LEFT': 271.45507054548904, 'RIGHT': 271.1304320264755555},
 ('DOWN', 'DOWN', 'DOWN'): {'UP': 149.39224786205895, 'DOWN': 140.9247844503192, 'LEFT': 136.452141041625, 'RIGHT': 301.755160197622995},
 ('DOWN', 'DOWN', 'UP'): {'UP': 220.18443835532133, 'DOWN': 224.181418392304555, 'LEFT': 262.29756118627125, 'RIGHT': 238.897781113231477},
 ('DOWN', 'UP', 'UP'): {'UP': 242.04491644858476, 'DOWN': 247.4797636427259, 'LEFT': 247.99439471603733, 'RIGHT': 245.639422255061892},
 ('UP', 'UP', 'UP'): {'UP': 245.60514545292602, 'DOWN': 248.46275189115417, 'LEFT': 244.32848190296482, 'RIGHT': 246.39120976560018},
 ('UP', 'UP', 'RIGHT'): {'UP': 218.139662585192, 'DOWN': 211.57524096566306, 'LEFT': 219.1574048674022, 'RIGHT': 210.377623554452385},
 ('UP', 'RIGHT', 'UP'): {'UP': 299.501010246757, 'DOWN': 290.0327182820703, 'LEFT': 316.24921967377685, 'RIGHT': 313.008836392841665},
 ('RIGHT', 'UP', 'RIGHT'): {'UP': 108.27697303626046, 'DOWN': 107.343154727635655, 'LEFT': 108.05793055227467, 'RIGHT': 108.256934152520275},
 ('UP', 'RIGHT', 'DOWN'): {'UP': 212.800569914697, 'DOWN': 261.7476620463246, 'LEFT': 200.7583412559839, 'RIGHT': 264.20381790258983},
 ('RIGHT', 'DOWN', 'LEFT'): {'UP': 177.2550890390506, 'DOWN': 206.82282054271266, 'LEFT': 164.4968337850222, 'RIGHT': 163.018860348413825},
 ('LEFT', 'LEFT', 'UP'): {'UP': 205.9743575879479, 'DOWN': 203.05713180609933, 'LEFT': 208.10377035408652, 'RIGHT': 207.852281617940155},
 ('UP', 'LEFT', 'UP'): {'UP': 201.2408108355925, 'DOWN': 201.52883749109185, 'LEFT': 199.56337653121824, 'RIGHT': 177.3797264383868},
 ('LEFT', 'UP', 'RIGHT'): {'UP': 209.46572110886817, 'DOWN': 209.9436443108246, 'LEFT': 209.70298334993504, 'RIGHT': 220.66040122857885},
 ('UP', 'RIGHT', 'LEFT'): {'UP': 173.72582309554238, 'DOWN': 162.79663329771287, 'LEFT': 172.40435042025882, 'RIGHT': 128.0488363499449},
 ('RIGHT', 'LEFT', 'DOWN'): {'UP': 264.00476704844993, 'DOWN': 265.68446044729933, 'LEFT': 274.6626787166906, 'RIGHT': 264.619789910631775},
 ('LEFT', 'DOWN', 'UP'): {'UP': 211.06173455252602, 'DOWN': 211.5672469403516, 'LEFT': 212.92495009961584, 'RIGHT': 212.219902544595},
 ('DOWN', 'UP', 'RIGHT'): {'UP': 164.5134254861162, 'DOWN': 197.049166985655594, 'LEFT': 163.5672261440562, 'RIGHT': 171.602862711343165},
 ('RIGHT', 'DOWN', 'UP'): {'UP': 82.64611220051808, 'DOWN': 102.36257019015254, 'LEFT': 94.95907207051123, 'RIGHT': 114.98014287931254},
 ('DOWN', 'UP', 'LEFT'): {'UP': 187.84628859852788, 'DOWN': 249.194036122491075, 'LEFT': 278.6674650673249, 'RIGHT': 240.390833006988945},
 ('UP', 'LEFT', 'RIGHT'): {'UP': 152.31172853508033, 'DOWN': 163.99523892394268, 'LEFT': 154.8186584401845, 'RIGHT': 258.124496577007855},
 ('LEFT', 'RIGHT', 'RIGHT'): {'UP': 128.26854305483437, 'DOWN': 153.6188937523693, 'LEFT': 131.863799204035135, 'RIGHT': 128.265895539660635},
 ('RIGHT', 'RIGHT', 'UP'): {'UP': 254.8879909129139, 'DOWN': 254.3878082363013, 'LEFT': 255.1038328381349, 'RIGHT': 256.748505910369545},
 ('RIGHT', 'UP', 'DOWN'): {'UP': 115.13944729414872, 'DOWN': 123.79499740352743, 'LEFT': 115.50739460161907, 'RIGHT': 114.793422745601025},
 ('UP', 'DOWN', 'UP'): {'UP': 272.7573696651964, 'DOWN': 174.35182719239165, 'LEFT': 161.38841773446322, 'RIGHT': 156.251415258529825},
 ('DOWN', 'UP', 'DOWN'): {'UP': 149.60329995716322, 'DOWN': 149.7781915602563, 'LEFT': 221.2019525407084, 'RIGHT': 148.78206414547037},
 ('UP', 'DOWN', 'RIGHT'): {'UP': 175.26732521097898, 'DOWN': 190.48190914289518, 'LEFT': 174.3882957677575, 'RIGHT': 175.868389417044175},
 ('DOWN', 'RIGHT', 'DOWN'): {'UP': 167.6444510269338, 'DOWN': 170.74530328247405, 'LEFT': 169.65391870241245, 'RIGHT': 170.5961318388702},
 ('RIGHT', 'DOWN', 'RIGHT'): {'UP': 160.06431599235185, 'DOWN': 158.49997026163007, 'LEFT': 158.6587502067437, 'RIGHT': 158.324719460119555},
 ('DOWN', 'RIGHT', 'RIGHT'): {'UP': 159.22144849457455, 'DOWN': 144.12755981100112, 'LEFT': 159.0989150682184, 'RIGHT': 158.225261363705395},
 ('RIGHT', 'UP', 'UP'): {'UP': 208.60078566099858, 'DOWN': 204.00763515243946, 'LEFT': 205.00406728874685, 'RIGHT': 174.171084116346265},
 ('UP', 'UP', 'DOWN'): {'UP': 175.41826983378334, 'DOWN': 175.50427842721254, 'LEFT': 216.3737409754967, 'RIGHT': 174.200175205906285},
 ('UP', 'DOWN', 'DOWN'): {'UP': 125.27708702887355, 'DOWN': 123.63434448054687, 'LEFT': 123.41764676033239, 'RIGHT': 123.147735162530655},
 ('DOWN', 'DOWN', 'LEFT'): {'UP': 217.60261373180714, 'DOWN': 215.58061352095514, 'LEFT': 234.370458373813, 'RIGHT': 216.34454844873525},
 ('LEFT', 'LEFT', 'LEFT'): {'UP': 171.8359185685608, 'DOWN': 171.54996252672007, 'LEFT': 172.87175475083535, 'RIGHT': 171.90323925958472},
 ('LEFT', 'UP', 'UP'): {'UP': 219.46902850736265, 'DOWN': 220.1807312993921, 'LEFT': 220.6204514140457, 'RIGHT': 219.443359342159825},
 ('UP', 'UP', 'LEFT'): {'UP': 241.15640701814513, 'DOWN': 248.75895668321562, 'LEFT': 256.29781371195236, 'RIGHT': 256.05164772719645},
 ('UP', 'DOWN', 'LEFT'): {'UP': 179.88911194476051, 'DOWN': 178.56540978381804, 'LEFT': 180.31023224979964, 'RIGHT': 180.20703682512565},
 ('DOWN', 'LEFT', 'DOWN'): {'UP': 242.81589892295173, 'DOWN': 243.28264795558272, 'LEFT': 242.3367960418736, 'RIGHT': 244.484379552508565},
 ('UP', 'LEFT', 'LEFT'): {'UP': 238.72524710728388, 'DOWN': 238.93221813990309, 'LEFT': 237.9759334847711, 'RIGHT': 206.36788896789365},
 ('LEFT', 'RIGHT', 'UP'): {'UP': 180.09506889655568, 'DOWN': 179.75109313443838, 'LEFT': 208.1809316294303, 'RIGHT': 181.126627660682175},
 ('DOWN', 'LEFT', 'UP'): {'UP': 111.75053011992989, 'DOWN': 110.27900811513098, 'LEFT': 143.73542062213423, 'RIGHT': 117.207187696224225},
 ('RIGHT', 'DOWN', 'DOWN'): {'UP': 244.8013062624271, 'DOWN': 255.76272193348282, 'LEFT': 255.6046205807677, 'RIGHT': 252.80679247574235},
 ('DOWN', 'DOWN', 'RIGHT'): {'UP': 178.64835246199235, 'DOWN': 258.6045575985817, 'LEFT': 197.3457945113975, 'RIGHT': 201.488541307329545},
 ('DOWN', 'RIGHT', 'UP'): {'UP': 205.72834808561583, 'DOWN': 206.16708158246345, 'LEFT': 203.5937061011043, 'RIGHT': 208.963339324853255},
 ('LEFT', 'UP', 'DOWN'): {'UP': 140.1637326111832, 'DOWN': 140.39995045988638, 'LEFT': 141.5251474684387, 'RIGHT': 156.22718081042683},
 ('LEFT', 'RIGHT', 'DOWN'): {'UP': 132.41182116033153, 'DOWN': 131.95026368645216, 'LEFT': 131.49937415318678, 'RIGHT': 131.3897879805895},
 ('RIGHT', 'UP', 'LEFT'): {'UP': 145.27417026053155, 'DOWN': 122.72639077647608, 'LEFT': 121.33392554932648, 'RIGHT': 126.36705641361226},
 ('DOWN', 'RIGHT', 'LEFT'): {'UP': 131.35554229073477, 'DOWN': 132.38651103276007, 'LEFT': 216.18764271290235, 'RIGHT': 130.8737876377979},
 ('UP', 'RIGHT', 'RIGHT'): {'UP': 213.3031338315429, 'DOWN': 213.0696457859641, 'LEFT': 213.36645323612342, 'RIGHT': 213.0771433926361},
 ('RIGHT', 'RIGHT', 'DOWN'): {'UP': 183.00204715383103, 'DOWN': 183.56680066689754, 'LEFT': 183.5844980878126, 'RIGHT': 182.436580939628985},
 ('LEFT', 'DOWN', 'RIGHT'): {'UP': 157.157275169367, 'DOWN': 198.752446941707065, 'LEFT': 175.74483653569823, 'RIGHT': 168.181271314210825},
 ('DOWN', 'LEFT', 'RIGHT'): {'UP': 91.19250580133229, 'DOWN': 109.96953866574266, 'LEFT': 87.5481367381621, 'RIGHT': 255.513605595109775},
 None: {'UP': 0.0, 'DOWN': 0.0, 'LEFT': 0.0, 'RIGHT': 0.0}
}
```

By inspecting the policy, I discovered that **it actively avoids falling into cycles**, ensuring the agent continues exploring instead of repeating redundant moves. For example:

Example from the Policy

1. **State:** `('RIGHT', 'RIGHT', 'RIGHT')`
    - **Best Action:** `LEFT`
    - **Explanation:** If the agent has been moving right repeatedly, the policy suggests taking a left turn to break the cycle of continuous "RIGHT" moves, which could lead to looping behavior in the wraparound grid.
2. **State:** `('RIGHT', 'RIGHT', 'LEFT')`
    - **Best Action:** `LEFT`
    - **Explanation:** Similarly, after moving "RIGHT" twice and then "LEFT," the policy still suggests continuing "LEFT" to avoid getting trapped in a back-and-forth movement between "RIGHT" and "LEFT."
3. **State:** `('LEFT', 'LEFT', 'RIGHT')`
    - **Best Action:** `UP`
    - **Explanation:** If the agent has alternated directions (e.g., "LEFT," "LEFT," then "RIGHT"), the policy opts for "UP" to ensure exploration and avoid repetitive horizontal movements.
4. **And so on if you continue to check actions.**
5. **Also when choosing an action, we have a condition that** `if random.random() < self.epsilon` **then we will be choosing a random action to break the cycles if we have**

Training the Q-Learning Agent by Simulating Games

In this setup, the Q-learning agent is trained using a simulation of a game, specifically the `SnakeGame` in this case. The main idea is that the agent learns by interacting with the environment (the game), where each action it takes has consequences in terms of rewards. The agent's goal is to maximize its cumulative reward over time. Here's how the training works:

1. Game Simulation:

- The agent interacts with the `SnakeGame` environment in a loop for a given number of episodes.
- During each episode, the agent starts with the initial state of the game (e.g., the position of the snake on the grid).
- In each step, the agent chooses an action, interacts with the game, and receives a reward. The environment provides feedback, which the agent uses to update its knowledge.

2. Agent's Actions:

- The agent chooses an action based on its current state. The `choose_action` function decides whether the agent should explore or exploit:

- **Exploration**: The agent might take a random action with probability `epsilon` (exploration factor), which ensures the agent tries new actions and doesn't get stuck in local optima.
- **Exploitation**: If the agent has learned from previous episodes, it exploits its knowledge by choosing the action that maximizes the expected reward, i.e., the action that maximizes the Q-value for the current state.

3. Updating the Q-table:

- After each action, the agent receives a reward and transitions to a new state. The `update_q_value` function updates the Q-table based on the following Q-learning update rule:
  - $Q(s,a)=Q(s,a)+\alpha(R+\gamma \cdot a' \max Q(s',a')-Q(s,a))$
  - s is the current state, a is the action taken, and s′ is the next state.
  - α is the learning rate (how much to update the Q-values).
  - γ is the discount factor (how much to consider future rewards).
  - R is the reward received after taking action a from state s.

4. Episode End Conditions:

- The game ends either when the agent eats the apple (reaches a terminal state) or when the number of steps exceeds the maximum allowed (a penalty is applied for this scenario). This allows the agent to experience both positive and negative rewards, which shape the learning process.

Reward Functions in the Q-learning Agent

The reward function plays a crucial role in shaping the agent's behavior. In this case, we have two main types of rewards: **positive rewards** and **negative penalties**.

1. Step Penalty:

- Every step the agent takes without eating the apple or exceeding the maximum number of steps incurs a small penalty (`step_penalty = -1`). This encourages the agent to act quickly and efficiently, avoiding unnecessary moves.
- Without a step penalty, the agent might move aimlessly and never converge to an optimal solution.

2. Apple Reward:

- When the agent eats the apple, a large positive reward is given (`max_apple_reward = (game.width * game.height) * 35 + 1`). This reward is much larger than the step penalty, driving the agent towards the goal (the apple).
- The larger reward for eating the apple teaches the agent to prioritize actions that lead to the apple's position.

3. Maximum Steps Penalty:

- If the agent exceeds the maximum number of allowed steps in the game, it receives a large negative penalty (`-max_apple_reward`). This penalty discourages the agent from taking too long to reach the apple or making inefficient moves.
- This penalty can also serve as a signal to the agent to avoid paths that might lead to longer games without success.

4. Terminal State:

- After either reaching the apple or exceeding the maximum number of steps, the agent is considered to have reached a terminal state. This means no further actions are taken, and the agent updates its Q-values using the terminal reward.
- The terminal state is important because it marks the end of the learning cycle for that episode, and the Q-table is updated based on the accumulated reward for that episode.

How the Agent Learns from Rewards

- The agent uses the rewards to refine its Q-values, adjusting its understanding of which actions are beneficial for specific states.

- Over time, through repeated simulations (training episodes), the agent learns the optimal policy that maximizes its cumulative reward.
  The learning process follows this pattern:

    - The agent starts with little to no knowledge of the environment and takes random actions (due to the epsilon-greedy strategy).
    - With each interaction, the agent learns which actions lead to better rewards, adjusting its Q-table to reflect this knowledge.
    - Eventually, the agent converges towards an optimal policy, where it can reliably predict the best action for any given state.

By running many episodes and experiencing the rewards and penalties, the agent learns the most effective strategies to reach the apple efficiently.

Conclusion

The exploration strategies of Randomized Walk, Spiral Walk, and Zigzag Walk all failed to meet the 35S constraint, primarily due to their lack of adaptability and efficiency in exploring the grid. These strategies, while simple and straightforward, could not effectively balance exploration and exploitation. The introduction of Q-Learning offers a promising solution, as it allows the agent to adapt and learn from the environment, leading to more efficient exploration and ultimately meeting the 35S constraint.