Summary of Design & Documentation – Reinforcement Learning Puddle World Implementation

Introduction to Application, Communication protocol & Architecture

The application that I built is a client-server architecture-based reinforcement learning puddle world implementation. The puddle world is a reinforcement learning benchmark problem that involves a grid world where an agent is free to move around in, and it contains puddles or states that can significantly reduce the cumulative reward achieved by the agent at the end of the reinforcement learning training process' episodes. The objective of the training process is teaching the reinforcement learning (RL) agent how to effectively navigate around the puddles better as it goes through more training episodes and maximize the final reward that it ends with (ending with a high positive cumulative episode reward) in the final episode of training. The client-server architecture was adopted in this application to maintain a consistent architecture pattern as that of the Game of Amazons in the COSC 322 course here at UBC, as well as to delegate the action choosing and training logic to the client side for students of the COSC 322 course to complete, and the server to closely monitor the training process of the RL agent through a messaging protocol that enables full transparency during the RL agent's training process. The SmartFoxServer's Java server and client APIs were both used by my application's parts to communicate with each other.

The communication protocol for the client-server communication was defined as containing the following reinforcement-learning specific variables that were important to ensure that the training process is conducted as expected and in accordance with the Q-learning formula [1, see references]:

Client to server messages:

1. The current state of the RL agent (GAME_STATE)
2. The available actions for an RL agent from the current state (GAME_AVAILABLE_ACTIONS)
3. The available rewards for the states that can be traversed from the current state (GAME_AVAILABLE_REWARDS)
4. The action chosen by the client side ((GAME_ACTION_MOVE)
5. The reward of the action taken by the RL agent ((GAME_ACTION_REWARD)
6. The client reaching the terminal state ((GAME_FINAL_STATE)
7. The client asking the server to reset the RL environment (GAME_RESET)
8. The client sending Q table updates (GAME_Q_UPDATE)
9. The client sending V table updates (GAME_V_UPDATE)
10. The client asks for an episode summary (with the reward and the termination condition) (GAME_INFO)
11. The client sending a message to the server that the client side agent has finished training (GAME_TRAINING_COMPLETE)

Server to client responses:

1. The server responding with the reward of the action taken by the RL agent (GAME_ACTION_REWARD)
2. The server responding with the current state of the RL agent (GAME_STATE_RESPONSE)
3. The server responding with the available actions for the client side RL agent from the current state (GAME_AVAILABLE_ACTIONS_RESPONSE)
4. The server responding with the available rewards for the states that can be traversed from the current state (GAME_AVAILABLE_REWARDS_RESPONSE)
5. The server responding with the reward of the action taken by the client (GAME_ACTION_REWARD_RESPONSE)
6. The server responding with notifying that the terminal state has been reached (GAME_FINAL_STATE_RESPONSE)
7. The server responding with resetting the environment for the RL agent's training (GAME_RESET_RESPONSE)
8. The server responding with an episode summary (with the reward and the termination condition) (GAME_INFO_RESPONSE)
9. The server responding with an error message based on incorrect parameters or format of the request sent by the client (GAME_ERROR)

Setup & How to Run the Application

The zipped code folder is called "deliverables 2". Here are the dependencies that a user must have installed on their computer before unzipping the code:

1. SmartFoxServer 2X Java Server API in the Community tab:
https://www.smartfoxserver.com/download/sfs2x#p=installer [2, see references]
2. SmartFoxServer 2X Java Client API in the Client API tab:
https://www.smartfoxserver.com/download/sfs2x#p=client [3, see references]
3. Java:
https://www.oracle.com/ca-en/java/technologies/downloads/ [3, see references]
4. Maven
5. Dependencies in the "deliverable 2/ygraph-ai-smartfox-server/pom.xml" and "deliverable 2/ygraph-ai-smartfox-client/pom.xml"

Below is the process that must be followed to run the application:

1. Download and unzip the "deliverables 2" folder from any folder on your device.
2. Navigate to the directory titled "ygraph-ai-smartfox-client/target" and you will see the client side code from where the RL agent will be run from.
3. Open up your file system and look for this directory "/Applications/SmartFoxServer_2X/SFS2X/extensions" or something similar, and copy the file titled "ygraph-ai-smartfox-server-extension-2.1.jar" in the "JAR file" directory in the deliverables 2 folder that you have open.
4. Create a directory called cosc322-2 in the "/Applications/SmartFoxServer_2X/SFS2X/extensions" directory and insert the copied JAR file into it.

5. Now, go into to the directory called "Zone file" in the "ygraph-ai-smartfox-client/target" directory and copy the file titled "cosc322-2.zone.xml".
6. Next, go into the directory titled "/Applications/SmartFoxServer_2X/SFS2X/zones" or something similar, and paste the copied file into this directory.
7. Navigate to the "deliverable 2/ygraph-ai-smartfox-server/pom.xml" file, and ensure that the text between <sfs.extensions.dir> </sfs.extensions.dir> has the path to your "/Applications/SmartFoxServer_2X/SFS2X/extensions/" in it, and the text between <sfs.working.dir> </sfs.working.dir> has the path to your "/Applications/SmartFoxServer_2X/SFS2X/" or something similar to make sure the RL agent will run without any errors. Here is a snippet of my pom.xml file:

```
<properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>

        <sfs.extensions.dir>/Applications/SmartFoxServer_2X/SFS2X/extensions/</sfs.extensions.dir>
        <sfs.working.dir>/Applications/SmartFoxServer_2X/SFS2X/</sfs.working.dir>
</properties>
```

8. Launch SFS2XLauncher on your device.
9. Now, open up a terminal and make sure that your current working directory "ygraph-ai-smartfox-client/target", and run the following the command to run the RL agent's training process:

```
java -jar ygraph-ai-smartfox-client-2.1.jar agent1 password1 localhost 9933 cosc322-2 RLRoom
```

10. Once you hit enter, you will see logs on the terminal you have open (client-side logs) and on the SFS2XLauncher app (server-side logs), which you can scroll up and down to monitor the training process of the RL agent.
11. Please take a look at the parameters on both client-side and server-side .env files in the "ygraph-ai-smartfox-client" and "ygraph-ai-smartfox-server" directories, which can be changed to change the RL agent's training process' termination condition, and if you choose to change the parameters in one .env file, please also change it in the other .env file.
12. Before running the code again after changing the parameters in **both .env files, please comment out the entire test files called "/deliverable 2/ygraph-ai-smartfox-server/src/test/java/ygraph/ai/smartfox/rl/test/java/yraph/ai/smartfox/rl/test/RLGameMessageTest.java" and "/deliverable 2/ygraph-ai-smartfox-server/src/test/java/ygraph/ai/smartfox/rl/test/java/yraph/ai/smartfox/rl/test/RLWorldTest.java".**
13. Then, navigate to the "ygraph-ai-smartfox-server" directory and run the following command in your terminal:

```
mvn clean package
```

14. Once you have run the command, copy the file generated in the “deliverable 2/ygraph-ai-smartfox-server/target” directory titled “ygraph-ai-smartfox-server-extension-2.1.jar” and paste it into the “/Applications/SmartFoxServer_2X/SFS2X/extensions/cosc322-2” directory and replace the old JAR file that was there.
15. Now you can rerun the application and the modified parameters will be used to run the application.
16. It is necessary to repeat this process everytime the parameters in the .env files are changed to ensure that the server can read the new values in the server side .env file.

How does the application work?

The application, at a high-level, has a server-side that works to receive the chosen actions from the client-side, and use that to update its representation of the RL agent on the server-side, and check for validity of actions, move the RL agent, and keep the RL agent’s training process going till a termination condition is hit. The server-side will respond to the client side with possible actions from a particular state, and their respective rewards, and the client can then use to decide on what action to choose based on the epsilon-greedy policy, that balances the ability of an RL agent to randomly choose an action (exploration) versus choosing the best action based on the computed Q-value in the Q-table (exploitation) [5, see references]. Q-learning is the process of training an RL agent to maximize its cumulative reward based on Q-tables (states are rows, and actions are columns) that store estimates of how good it is to be in state s and take action a to maximize rewards and are what guide the action suggestions of the RL agent from the client side. 4 possible actions are allowed for the RL agent: up, down, left, and right. I also defined the V-table, for which each box in the table tells us how much total reward I will see being in state s in total. The following Q-value update formula was used [1, see references]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]. \tag{6.8}$$

This update formula for Q-values in the Q-table guides the client side RL agent in making suggestions for actions, with S_t being the state that the RL agent is in at time interval t, A_t being the action that the RL agent took at time interval t, α is the speed at which the RL agent learns to avoid puddles (it priorities immediate rewards over future rewards), Q(S_t, A_t) represents the Q-value of going with action A_t from state S_t, R_{t+1} is the reward in time interval t+1, γ is the discount factor (it priorities future rewards over immediate rewards).

Each individual class on the client-side and server-side serve a different purpose in training the RL agent, at a high-level, they are explained below (please refer to the code files for further explanation of how the internal details of the code works):

Client-side classes:

1. RLClientGameMessage.java : This class creates and manages attributes of a message sent between the client and server - establishing a consistent protocol.
2. RLGameModel.java: This class holds the game's state specific to RL puddle world parameters like: state, available actions, rewards and the final state check.

3. RLGamePlayer.java: This class represents an RL agent's core logic and interactions with the Puddle World environment along with handling the connection and message processing to and from the server.

Server-side classes:

1. RLGameExtension.java: This class creates the RL environment within the server and sets up request handler to handle client-related RL logic.
2. RLGameManager.java: This class takes RLGameUser instances and binds them with RLWorld intances allowing for concurrent usage of the application.
3. RLGameMessage.java: This class defines the messaging protocol that understand what is being sent from the client side and can communicate back to the client side.
4. RLGameRequestHandler.java: This class handles RL-specific client-side requests and sends responses back to the client.
5. RLGameUser.java: This class represents a user within an RL game (server-side representation of the RL agent).
6. RLMultiHandler.java: This class manages non-RL-specific client requests like joining rooms and using RLGameManager to add and remove users.
7. RLWorld.java: This class defines the internal representation of the Puddle world on the server side.

The control flow of the program from start to end is as follows:
The RLGameExtension registers the RLGameRequestHandler to handle the client-side's requests, and then RLGamePlayer on the client side logs in and joins the room via RLMultiHandler, and then RLGamePlayer starts by requesting the initial state, and the user added to the hashmap storing the RLGameUser objects mapped to their usernames. Next, the server responds with the state that the RL agent is in, and the client-side requests possible actions and their respective rewards, and the server responds to those. The client side picks the next action with the epsilon-greedy policy, updates the Q and V tables, and sends the Q and V updates to the server along with the action that it picked. This process keeps going till the termination conditions are reached, till the number of episodes are set as the parameters in the .env files for the client and server side. The 3 termination conditions for an episode are as follows based on the STOP_METHOD variable in the .env files on the client and server side:

STOP_METHOD=0:

- With this training termination condition, the RL agent is left to traverse the puddle world until a number of maximum steps that are defined in the client and server side .env files.
- It is advised to keep the MAX_PUDDLES and PUDDLE_SIZE small (if setting MAX_PUDDLES=3 and PUDDLE_SIZE=3 on gridSize=6, the number of states that get covered with puddles is 3^3 states (out of 6^2 total states on the grid), as the puddles are created in grouped squares, so there will be three 3x3 squares of puddle states, therefore, please either maintain MAX_PUDDLES closer to 1 in smaller grid sizes, or if you prefer to keep the MAX_PUDDLES and PUDDLE_SIZE equal to each other, then maintain $MAX_PUDDLES^3 < \frac{1}{4}$th of grid size in puddle states) as it can take a while to generate puddles that do not overlap each other or the goal state.

- This is the default behavior, and the training will keep going on until the maximum number of steps is reached or the goal state is reached, whichever comes first.

STOP_METHOD=1:

- With this training termination condition, the RL agent is left to traverse the puddle world until it reaches the goal state, which is located at the state ID: $gridSize^2 - 1$ (it is advised to keep the gridSize < 8 or 9 to reduce the time it takes for an episode of training to complete).

STOP_METHOD=2:

- With this training termination condition, the RL agent is left to traverse the puddle world until it is stopped by a probability that is randomly generated. This probability is set on the client and server side .env files, and essentially the way this method works is that a random number is generated, if it is lesser than the probability that is in the .env files (STOP_PROB), then the episode is terminated immediately, but if the random number generated is greater than STOP_PROB, then the training process moves to the next step, and a new random number is generated before each step is taken by the RL agent.
- This method will keep going on until either the probabilistic stopping terminates the episode, or if the RL agent reaches the goal state, whichever comes first.

Output (Example Runs)

Here is what the .env files had in them when running the below tests of the application:

```
MAX_STEPS=30
EPISODE_COUNT=3
STOP_PROB=0.1
GRID_SIZE=5
ALPHA=0.1
GAMMA=0.9
EPSILON=1.0
DEFAULT_REWARD=-0.02
PUDDLE_REWARD=-2.0
GOAL_REWARD=20.0
MAX_PUDDLES=2
PUDDLE_SIZE=2
SUCCESS_REWARD_THRESHOLD=2.0
```

Snippets of the outputs on the client and server sides for each different stopping method are shown below, of which each visual has an episode summary to it, the number of steps in the episode, and the cumulative reward from each episode (this output is focusing on a single episode of training due to the length of the logs):

Continued in next page.

STOP_METHOD=0:
Client-side logs:

```
All received keys: [messageType, availableRewards]
Available Rewards: [-0.02, -0.02, -0.02, -2.0]
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
Parsing messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
All received keys: [availableActions, messageType]
Available Actions: [0, 1, 2, 3]
Requested available rewards for state 17
Deciding action for state: 17 with available actions: [0, 1, 2, 3]
Exploring with random action: 2
Chosen Action: 2
Awaiting server response. Action not sent.
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_AVAILABLE_REWARDS_RESPONSE
Parsing messageType: GAME_AVAILABLE_REWARDS_RESPONSE
All received keys: [messageType, availableRewards]
Available Rewards: [-0.02, -0.02, -0.02, -0.02]
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
Parsing messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
All received keys: [availableActions, messageType]
Available Actions: [0, 1, 2, 3]
Requested available rewards for state 17
Deciding action for state: 17 with available actions: [0, 1, 2, 3]
Exploring with random action: 0
Chosen Action: 0
Awaiting server response. Action not sent.
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_ACTION_REWARD_RESPONSE
Parsing messageType: GAME_ACTION_REWARD_RESPONSE
All received keys: [reward, messageType, nextStateId, action]
State transition: 17 -> 22
DEBUG: stopMethod=0

=== Episode 3 Information ===
Reason: Maximum steps (30) reached
Goal Reached: No
================================

Sent GAME_FINAL_STATE message to the server.

=== Training Complete ===
Total Episodes: 3
```

```
======================

Sent GAME_TRAINING_COMPLETE message to the server.
Disconnected from SmartFoxServer.
Updated Q-value for state 17, action 1: 0.2527783220164106
Sent Q-Table updates to the server.
Sent V-Table updates to the server.
Epsilon decayed to: 0.6369088258938781
Sent GAME_TRAINING_COMPLETE message to the server.
Disconnected from SmartFoxServer.
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_FINAL_STATE_RESPONSE
Parsing messageType: GAME_FINAL_STATE_RESPONSE
All received keys: [stepsThisEpisode, messageType, successfulEpisodes,
cumulativeReward, totalEpisodes, isTerminal]


=== End of Episode Summary ===
Steps Taken: 30/30
Episode Reward: -22.379999999999992
===============================================

Reset cumulative reward from -22.379999999999992 to 0.0
Starting new episode...
Requesting initial state for new episode...
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_TRAINING_COMPLETE
Received training complete message from server.
Training completed. Maximum number of episodes reached.
```

Corresponding Server-side logs:

```
RLWorld retrieved for user: agent1 - Instance ID: 1070894385
Handling Available Actions Request for stateId: 17
Available Actions for state 17: UP, DOWN, LEFT, RIGHT
Available Actions Length: 4
Available Actions: UP, DOWN, LEFT, RIGHT
Available Actions Indices: [0, 1, 2, 3]
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
Sent Available Actions Response to user: agent1
Attempting to retrieve user: agent1 with memory reference: 451698926
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
State 22 is not a puddle.
moveAgentWithAction: stateId=17, action=1 (DOWN)
```

```
Action 'DOWN' from (4, 2) leads to newStateId=22 with reward=-0.02
Moved to state ID: 22 with Reward: -0.02
DEBUG (Server): stopMethod=0
User agent1 reached maximum steps per episode.
Action Taken: DOWN, New State: 22, Reward: -0.02
Sent GAME_ACTION_REWARD_RESPONSE with action: 1, reward: -0.02, nextStateId: 22
Sent GAME_FINAL_STATE_RESPONSE
End of Episode Summary:
 - Total Episodes: 3
 - Successful Episodes: 0
 - Steps Taken: 30
 - Episode Reward: -22.379999999999992
Maximum number of episodes reached. Ending training.
Puddle added at row: 1, col: 3
Puddle added at row: 1, col: 0
Total Puddles Initialized: 2
World reset. Current state set to 0.
Puddle at row: 1, col: 3
Puddle at row: 1, col: 0
Episode concluded for user: agent1
Attempting to retrieve user: agent1 with memory reference: 451698926
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
RLWorld retrieved for user: agent1 - Instance ID: 1070894385
Handling Available Rewards Request for stateId: 17
Available Actions for state 17: UP, DOWN, LEFT, RIGHT
Available Actions Length: 4
Available Actions: UP, DOWN, LEFT, RIGHT
State ID: 17
Available rewards array initialized with length: 4
Processing Action: UP
Simulating action 'UP' from state (3, 2)
Action 'UP' results in new state ID: 12
State 12 is not a puddle.
Normal state transition. Assigning reward: -0.02
Action 'UP' leads to state 12 with reward -0.02
Processing Action: DOWN
Simulating action 'DOWN' from state (3, 2)
Action 'DOWN' results in new state ID: 22
State 22 is not a puddle.
Normal state transition. Assigning reward: -0.02
Action 'DOWN' leads to state 22 with reward -0.02
```

STOP_METHOD=1:
Client-side logs:

```
Received messageType: GAME_ACTION_REWARD_RESPONSE
Parsing messageType: GAME_ACTION_REWARD_RESPONSE
All received keys: [reward, messageType, nextStateId, action]
State transition: 19 -> 24

!!! GOAL STATE REACHED !!!

=== Episode 2 Information ===
Reason: Goal state reached!
Goal Reached: Yes!
================================

Sent GAME_FINAL_STATE message to the server.

=== Starting Episode 3/3 ===
Steps: 0/30
Episode Reward: 0.0
===========================

Requesting initial state for new episode...
DEBUG: stopMethod=1
Updated Q-value for state 19, action 1: 2.320297771734851
Sent Q-Table updates to the server.
Sent V-Table updates to the server.
Epsilon decayed to: 0.18190617987607657
Requested available actions for state 24
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_FINAL_STATE_RESPONSE
Parsing messageType: GAME_FINAL_STATE_RESPONSE
All received keys: [stepsThisEpisode, messageType, successfulEpisodes,
cumulativeReward, totalEpisodes, isTerminal]


=== End of Episode Summary ===
Steps Taken: 152/30
Episode Reward: 3.120000000000015
===============================================
```

Corresponding Server-side logs:

```
Attempting to retrieve user: agent1 with memory reference: 79497150
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
Attempting to retrieve user: agent1 with memory reference: 79497150
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
```

```
moveAgentWithAction: stateId=19, action=1 (DOWN)
Action 'DOWN' from (4, 4) leads to newStateId=24 with reward=20.0
Moved to state ID: 24 with Reward: 20.0
DEBUG (Server): stopMethod=1
User agent1 has reached the terminal state: 24
Action Taken: DOWN, New State: 24, Reward: 20.0
Sent GAME_ACTION_REWARD_RESPONSE with action: 1, reward: 20.0, nextStateId: 24
Attempting to retrieve user: agent1 with memory reference: 79497150
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
Attempting to retrieve user: agent1 with memory reference: 79497150
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
 - agent1 (equals provided username? true)
User found: agent1
Sent GAME_FINAL_STATE_RESPONSE
User agent1 achieved success in episode 2
End of Episode Summary:
 - Total Episodes: 2
 - Successful Episodes: 1
 - Steps Taken: 152
 - Episode Reward: 3.120000000000015
Puddle added at row: 2, col: 1
Puddle added at row: 1, col: 3
Total Puddles Initialized: 2
World reset. Current state set to 0.
Puddle at row: 2, col: 1
Puddle at row: 1, col: 3
Game initialized for user: agent1 at state: 0
Game reset for user: agent1. New starting state: 0
Puddle added at row: 1, col: 0
Puddle added at row: 0, col: 2
Total Puddles Initialized: 2
World reset. Current state set to 0.
Puddle at row: 1, col: 0
Puddle at row: 0, col: 2
Episode concluded for user: agent1
```

STOP_METHOD=2:
Client-side logs:

```
State transition: 0 -> 5
DEBUG: stopMethod=2
Updated Q-value for state 0, action 1: 0.517982979042154
```

```
Sent Q-Table updates to the server.
Sent V-Table updates to the server.
Epsilon decayed to: 0.9137248860125932
Requested available actions for state 5
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
Parsing messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
All received keys: [availableActions, messageType]
Available Actions: [0, 1, 3]
Requested available rewards for state 5
Deciding action for state: 5 with available actions: [0, 1, 3]
Exploring with random action: 0
Chosen Action: 0
Sent Action: 0 (UP) for State: 5
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_AVAILABLE_REWARDS_RESPONSE
Parsing messageType: GAME_AVAILABLE_REWARDS_RESPONSE
All received keys: [messageType, availableRewards]
Available Rewards: [-0.02, -0.02, -0.02]
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
Parsing messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
All received keys: [availableActions, messageType]
Available Actions: [1, 3]
Requested available rewards for state 5
Deciding action for state: 5 with available actions: [1, 3]
Exploring with random action: 3
Chosen Action: 3
Awaiting server response. Action not sent.
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
Parsing messageType: GAME_AVAILABLE_ACTIONS_RESPONSE
All received keys: [availableActions, messageType]
Available Actions: [0, 1, 3]
Requested available rewards for state 5
Deciding action for state: 5 with available actions: [0, 1, 3]
Exploring with random action: 3
Chosen Action: 3
Awaiting server response. Action not sent.
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_ACTION_REWARD_RESPONSE
Parsing messageType: GAME_ACTION_REWARD_RESPONSE
All received keys: [reward, messageType, nextStateId, action]
State transition: 5 -> 0
DEBUG: stopMethod=2
Updated Q-value for state 5, action 0: 0.7205032890695144
Sent Q-Table updates to the server.
Sent V-Table updates to the server.
Epsilon decayed to: 0.9091562615825302
Requested available actions for state 0
```

```
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_FINAL_STATE_RESPONSE
Parsing messageType: GAME_FINAL_STATE_RESPONSE
All received keys: [stepsThisEpisode, messageType, successfulEpisodes,
cumulativeReward, totalEpisodes, isTerminal]


=== End of Episode Summary ===
Steps Taken: 8/30
Episode Reward: -0.16
===============================================

Reset cumulative reward from -0.16 to 0.0
Starting new episode...
Requesting initial state for new episode...
Received EXTENSION_RESPONSE: cmd=rl.action
Received messageType: GAME_TRAINING_COMPLETE
Received training complete message from server.
Training completed. Maximum number of episodes reached.
Disconnected from SmartFoxServer.
```

Corresponding Server-side logs:

```
Attempting to retrieve user: agent1 with memory reference: 934437991
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
RLWorld retrieved for user: agent1 - Instance ID: 600257690
Handling Available Actions Request for stateId: 5
Available Actions for state 5: UP, DOWN, RIGHT
Available Actions Length: 3
Available Actions: UP, DOWN, RIGHT
Available Actions Indices: [0, 1, 3]
Attempting to retrieve user: agent1 with memory reference: 934437991
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
Attempting to retrieve user: agent1 with memory reference: 934437991
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
Sent Available Actions Response to user: agent1
Attempting to retrieve user: agent1 with memory reference: 934437991
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
State 0 is not a puddle.
moveAgentWithAction: stateId=5, action=0 (UP)
```

```
Action 'UP' from (0, 0) leads to newStateId=0 with reward=-0.02
Moved to state ID: 0 with Reward: -0.02
Attempting to retrieve user: agent1 with memory reference: 934437991
Current users in userMap:
DEBUG (Server): stopMethod=2
User agent1 stopped due to STOP_METHOD=2 random stopping condition.
Action Taken: UP, New State: 0, Reward: -0.02
 - agent1 (equals provided username? true)
User found: agent1
Sent GAME_ACTION_REWARD_RESPONSE with action: 0, reward: -0.02, nextStateId: 0
Sent GAME_FINAL_STATE_RESPONSE
End of Episode Summary:
 - Total Episodes: 3
 - Successful Episodes: 0
 - Steps Taken: 8
 - Episode Reward: -0.16
Maximum number of episodes reached. Ending training.
Puddle added at row: 2, col: 1
Puddle added at row: 1, col: 3
Total Puddles Initialized: 2
World reset. Current state set to 0.
Puddle at row: 2, col: 1
Puddle at row: 1, col: 3
Episode concluded for user: agent1
Attempting to retrieve user: agent1 with memory reference: 934437991
Current users in userMap:
 - agent1 (equals provided username? true)
User found: agent1
RLWorld retrieved for user: agent1 - Instance ID: 600257690
Handling Available Rewards Request for stateId: 5
Available Actions for state 5: UP, DOWN, RIGHT
Available Actions Length: 3
Available Actions: UP, DOWN, RIGHT
State ID: 5
```

As can see from the logs for all the stop methods, they stop exactly as how described above.

Limitations

The one limitation I believe that my code has is that it can be refactored to make the code cleaner and adhere to best practices when developing scalable software.

References

1. Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

2. https://www.smartfoxserver.com/download/sfs2x#p=installer – SmartFoxServer Java Server API downloads page
3. https://www.smartfoxserver.com/download/sfs2x#p=client - SmartFoxServer Java Client API downloads page
4. https://www.oracle.com/ca-en/java/technologies/downloads/ - Oracle Java Downloads page
5. https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/ - Epsilon-Greedy policy in RL