**PART 1**

**VALUE ITERATION**

Here is the procedure used to implement Value iteration:

* Start with utility values of every state, U(s) as 0.
* Iterate until convergence, such the utility values between successive trials using the epsilon value of 1.0:

Solve the Bellman equation to compute the utilities of all the states, using the following formula:



**Parameters Used:**

Gamma value: 0.99

Epsilon (for convergence): 1.0

**Number of Iterations:** 459

**Final Utility Values:**

State (0 , 0) : 99.007901

State (0 , 1) : 0.0

State (0 , 2) : 94.053359

State (0 , 3) : 92.882903

State (0 , 4) : 91.662516

State (0 , 5) : 92.336405

State (1 , 0) : 97.401263

State (1 , 1) : 94.890919

State (1 , 2) : 93.552899

State (1 , 3) : 93.405616

State (1 , 4) : 0.0

State (1 , 5) : 89.925826

State (2 , 0) : 95.956402

State (2 , 1) : 94.594329

State (2 , 2) : 92.302329

State (2 , 3) : 92.184175

State (2 , 4) : 92.11027

State (2 , 5) : 90.802772

State (3 , 0) : 94.56174

State (3 , 1) : 93.460395

State (3 , 2) : 92.240447

State (3 , 3) : 90.123158

State (3 , 4) : 90.822308

State (3 , 5) : 90.895986

State (4 , 0) : 93.320421

State (4 , 1) : 0.0

State (4 , 2) : 0.0

State (4 , 3) : 0.0

State (4 , 4) : 88.556314

State (4 , 5) : 89.574666

State (5 , 0) : 91.945376

State (5 , 1) : 90.736679

State (5 , 2) : 89.543053

State (5 , 3) : 88.364311

State (5 , 4) : 87.577

State (5 , 5) : 88.305592

**Policy Estimates:**

↑ # ← ← ← ↑

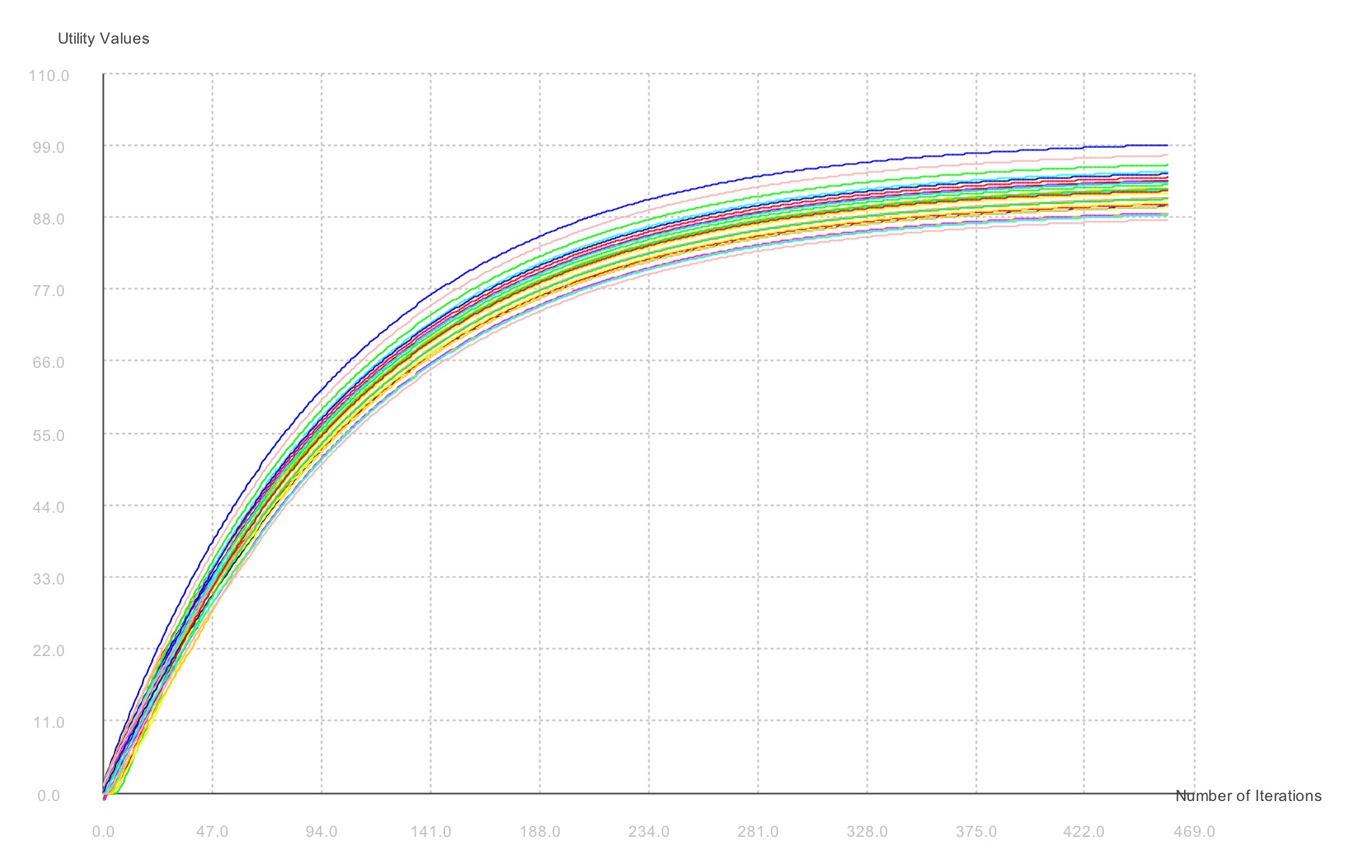
↑ ← ← ← # ↑

↑ ← ← ↑ ← ←

↑ ← ← ↑ ↑ ↑

↑ # # # ↑ ↑

↑ ← ← ← ↑ ↑

**Utility Values vs Number of Iterations Plot:**

**How to Run:**

javac -cp jmathplot.jar ValueIteration.java QLearning.java GiftFrenzy.java RandomCollection.java PlotGraph.java

java -cp jmathplot.jar: ValueIteration

**TD Q-LEARNING**

Here is the procedure used to implement Temporal Difference Q-Learning:

* Initialize all the action-utility function values for all State-Action pairs Q(s, a), and N(s, a) values (number of times the state-action pair is observed) to 0.
* Iterate until the Q(s, a) values for all state-action pairs converge between successive trials:

Loop for 2000 timestamps, and perform the following steps:

* From current state s, select an action a:



* Get the successor state **s’** by applying the max action **a**.
* Update the Q(s, a) value for current previous state, using successor states value:



**Parameters Used:**

Gamma value: 0.99

Epsilon (for convergence): 1.0

Learning Rate: 60 / (59 + N(s,a)), where N(s,a) -> Number of times action a has been taken from state s, as learning rate depends on the number of states explored

Exploration-Exploitation function: Ne -> 25, R+ -> 100.0

Inner Loop Iterations: 1000

**Number of Trials:** 1471

**Final Utility Values:**

State : 0 , 0 Max Utility : 99.81271720657426

State : 0 , 1 Max Utility : 0.0

State : 0 , 2 Max Utility : 94.7400009127824

State : 0 , 3 Max Utility : 93.0284040748016

State : 0 , 4 Max Utility : 91.87462933769808

State : 0 , 5 Max Utility : 90.8914231158993

State : 1 , 0 Max Utility : 98.29967594123916

State : 1 , 1 Max Utility : 95.28658069485273

State : 1 , 2 Max Utility : 93.94882357148296

State : 1 , 3 Max Utility : 93.65067739386033

State : 1 , 4 Max Utility : 0.0

State : 1 , 5 Max Utility : 88.77488311806174

State : 2 , 0 Max Utility : 96.42338972310209

State : 2 , 1 Max Utility : 94.99164015778486

State : 2 , 2 Max Utility : 93.05485129233683

State : 2 , 3 Max Utility : 92.75017748260669

State : 2 , 4 Max Utility : 92.97461763169238

State : 2 , 5 Max Utility : 91.3512047832631

State : 3 , 0 Max Utility : 95.24095194925096

State : 3 , 1 Max Utility : 94.30641816591313

State : 3 , 2 Max Utility : 92.97545936909839

State : 3 , 3 Max Utility : 91.04989886824252

State : 3 , 4 Max Utility : 91.6493017414143

State : 3 , 5 Max Utility : 91.79006084303431

State : 4 , 0 Max Utility : 93.94656774376126

State : 4 , 1 Max Utility : 0.0

State : 4 , 2 Max Utility : 0.0

State : 4 , 3 Max Utility : 0.0

State : 4 , 4 Max Utility : 89.40797942416282

State : 4 , 5 Max Utility : 90.52047781100639

State : 5 , 0 Max Utility : 92.84116778419443

State : 5 , 1 Max Utility : 91.53110472829557

State : 5 , 2 Max Utility : 90.22539796253312

State : 5 , 3 Max Utility : 89.2910392481423

State : 5 , 4 Max Utility : 88.32207680394875

State : 5 , 5 Max Utility : 89.09617330676316

**Policy Estimates:**

**↑ # ← ↓ ← ←**

**↑ ← ← ← # ↑**

**↑ ← ← ↑ ← ←**

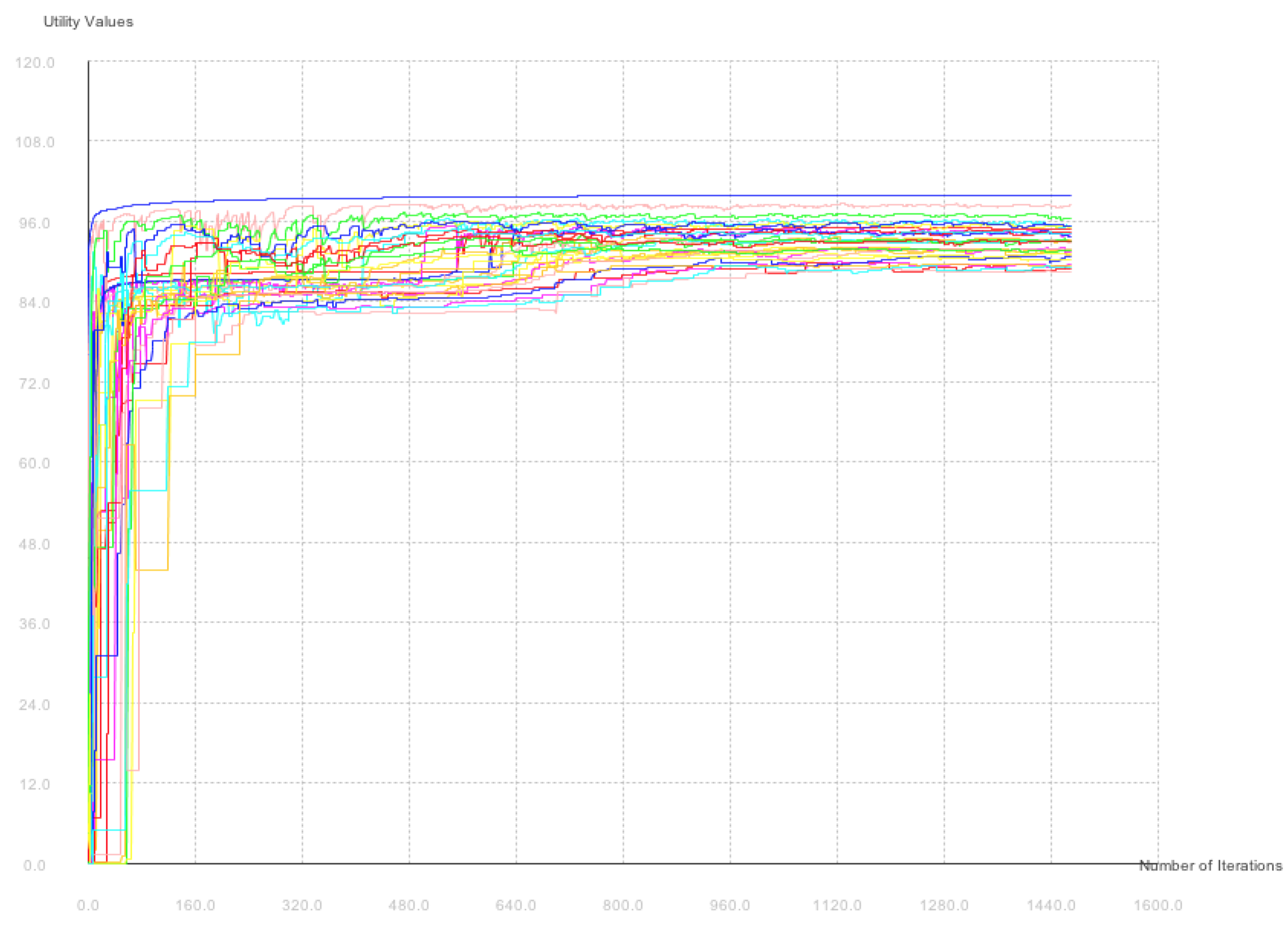
**↑ ← ← ↑ ↑ ↑**

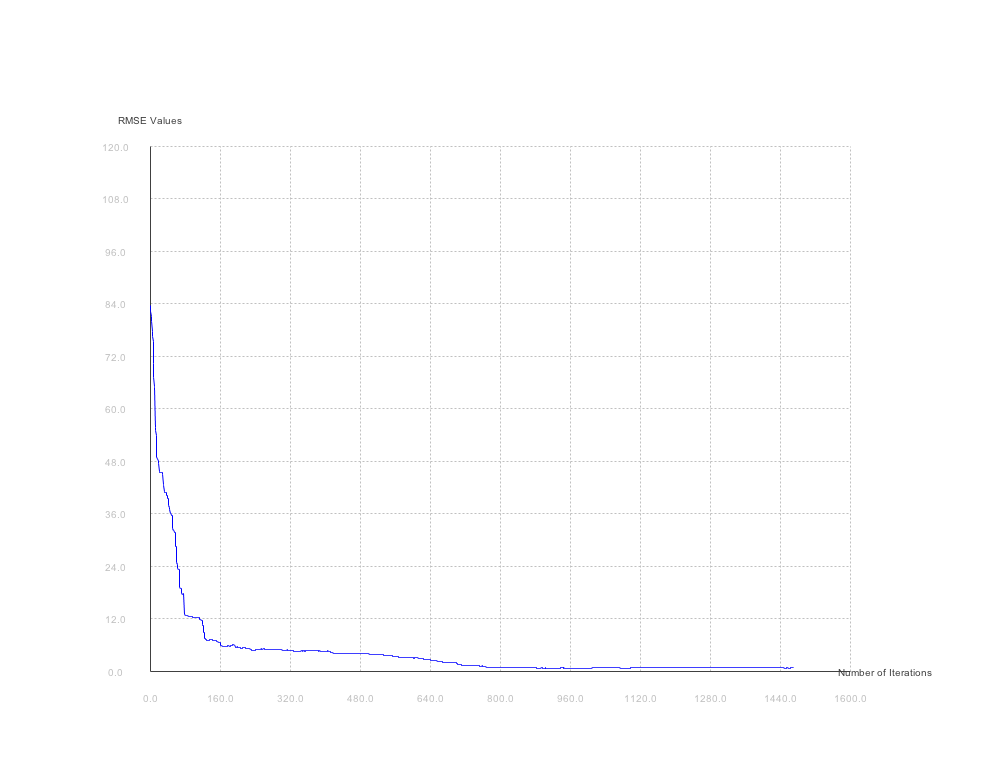
**↑ # # # ↑ ↑**

**↑ ← ← ← ↑ ↑**

From the above result we can infer that our Q learning policy differs from that of optimal policy by value iteration by **2** (i.e. only the policy for grid (0,3) and (0,5) differ)

**Utility Values vs Number of Trials Plot:**

****

****

**RMS Error vs. Number of Trials plot:**

**How to Run:**

javac -cp jmathplot.jar ValueIteration.java QLearning.java GiftFrenzy.java RandomCollection.java PlotGraph.java

java -cp jmathplot.jar: QLearning

**GIFT FRENZY**

Implemented using the same procedure as Q-Learning. However, here the state representation changes, and the following additional properties are added to the state:

* HasGift -> whether or not the agent has gift
* HasMoney -> whether or not the agent has money

**Parameters Used:**

Gamma value: 0.99

Epsilon (for convergence): 0.001

Learning Rate: 60 / (59 + t), where t -> timestamp

Exploration-Exploitation function: Ne -> 100, R+ -> 5.0

Inner Loop iterations: 10000

**Number of Trials:** 4447

**Policy Estimates:**

***Policy for Gift -> yes, Money -> yes:***

# # # # # # # # # # # # # # #

# ← ↑ → ↑ ↓ → ↓ ↑ → ← ↑ ↑ ↑ #

# → → ↓ → → ↓ → → ↓ ← ↑ ↑ ↓ #

# ↑ ↑ ← # ↓ ↓ ↑ ← ↓ # ↑ ↑ ← #

# # # # # ← ← ↓ ↓ → # # # # #

# ↓ ↑ → → ↓ ↓ → ↓ ↓ # ↓ ↓ → #

# ↓ → ↑ → ↓ → ← → → → → ← → #

# → ↓ → ↓ → ↓ ↑ ← → ↓ ↓ ↑ ← #

# # # # # # # # # # # # # # #

***Policy for Gift -> yes, Money -> no:***

# # # # # # # # # # # # # # #

# ← ← ↑ ↑ ← → ↑ → ↓ ← ↓ ← ↓ #

# ↓ ← ↑ ↓ ← ↓ ↑ → ↑ → ← ← ↑ #

# ↓ ↓ ← # ↓ → → ↓ ← # ↑ ↑ ↑ #

# # # # # → ← ↓ ↓ ↓ # # # # #

# → ↑ ← ↑ ↓ ← ↑ ↓ ↓ # ↑ ↓ ↓ #

# → ↑ ← ↓ ↑ ← ← ↓ → → → → → #

# → ↓ → ↓ ↓ ↓ → → → → ↑ ↓ ↑ #

# # # # # # # # # # # # # # #

***Policy for Gift -> no, Money -> yes:***

# # # # # # # # # # # # # # #

# → ↓ ← → → ↓ ↓ ↓ → ↑ ← ↑ ← #

# → ↓ ← ← ← ↓ ↑ ↓ ↑ → ↓ → ↓ #

# → ↑ ↑ # ↓ ↓ ↑ ↓ ← # ← → → #

# # # # # ↓ ↓ ↓ → ↓ # # # # #

# → ↓ → → ← ↓ ↓ ↓ ← # ← → ← #

# ↑ → ← ← → ↓ ← ← ← ← ↓ ↑ ← #

# ↑ ← → → ↓ ↓ ← ↓ ← ↓ ← ← ← #

# # # # # # # # # # # # # # #

***Policy for Gift -> no, Money -> no:***

# # # # # # # # # # # # # # #

# → ← → ← ↑ ← ↓ ↑ ↓ → ← ← → #

# ← → ← → → ↑ ↑ ↑ ← ↑ ← ↓ → #

# → ↓ ↑ # ↓ → ↓ → ↓ # ← ← ↓ #

# # # # # ↑ ↓ ↑ ↓ ↓ # # # # #

# ← ← ← ← ← → ↑ ← ← # ↑ ↓ ↓ #

# → ← ↓ ↓ ← ↓ → ↑ ← ← ← ↓ ↓ #

# ↓ ↑ ← ↑ ↓ → ↓ → ← ↓ ← ↓ ↓ #

# # # # # # # # # # # # # # #

**How to Run:**

javac -cp jmathplot.jar ValueIteration.java QLearning.java GiftFrenzy.java RandomCollection.java PlotGraph.java

java -cp jmathplot.jar: GiftFrenzy