

**CZ4046 Intelligent Agents**

Assignment 1

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING  
NANYANG TECHNOLOGICAL UNIVERSITY**

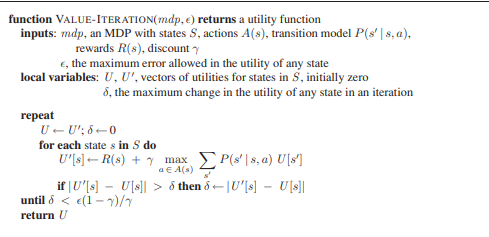
U1821787L Marcus Tang Zi Yang

**IMPLEMENTED SOLUTION – AN OVERVIEW**

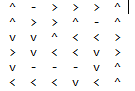
The implemented solution follows an object-oriented approach. We propose a grid-world environment (Environment) consisting of an array of boxes (Squares[][]). Each Square consists of a SquareState, an enum value that indicates the type of the square. The environment is parsed and a MDP (Policy) is formed, which includes an array of states (State[][]), a transition model (TransitionModel), an array of actions indicating the optimal policy (Action[][]), and a reward model used to implement the rewarding mechanics (RewardModel). An action consists of the environment it is in, the square it originates from, and the vector of the change (Vector2D). Vector2Ds are used to indicate directional change and displacement from origin (0,0) to facilitate computation for value/policy iteration.

**IMPLEMENTED SOLUTION – VALUE ITERATION**

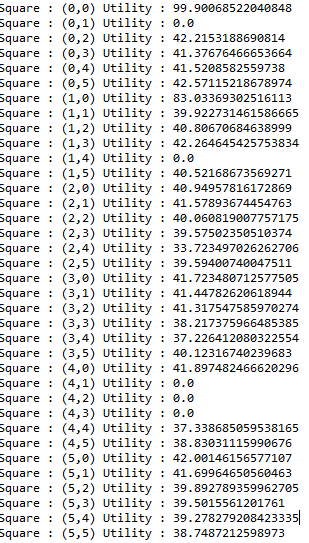
The implementation of value iteration follows the proposed implementation. In the program, a MDP is passed into the function as a Policy object and is broken down into its components. The expectimax utility is provided by the TransitionHandler static object, which handles the transition probability calculations and returns an utility value to be included in the Bellman equation used in the Bellman update. Each state is updated through the Bellman update process, and at every iteration, the changes are effected together. When the change in the utility diminishes below the threshold detailed by the equation ,the process is stopped. This results in utility values very close to the unique solutions of the Bellman equations, and computation of the optimal policy can be done in a single pass.



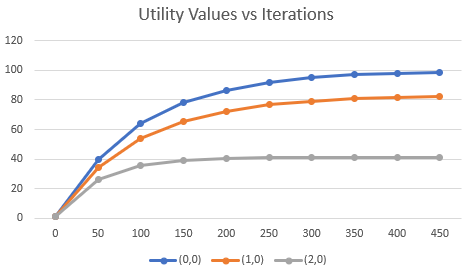
**VALUE ITERATION – PLOT OF OPTIMAL POLICY**



**VALUE ITERATION – UTILITY VALUES**

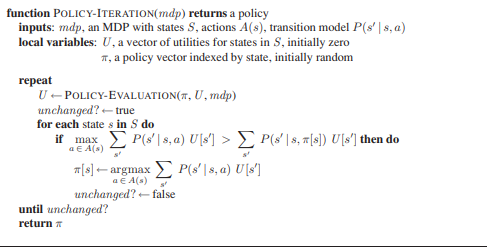


**VALUE ITERATION – UTILITY AS A FUNCTION OF ITERATIONS**

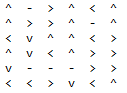


**IMPLEMENTED SOLUTION – POLICY ITERATION**

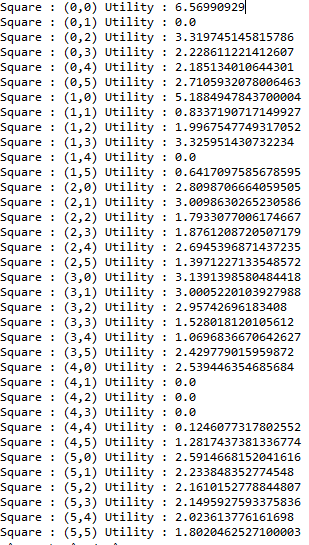
The implementation of value iteration follows the proposed implementation. In the program, a MDP is passed into the function as a Policy object and is broken down into its components. An initially random policy is initialized, and we start a loop where between policy evaluation and policy improvement. In policy evaluation, the program computes the utility values given the proposed policy, once again relying on TransitionHandler to provide the utility values. In the policy improvement step, we calculate an optimal policy using the utility values calculated in the previous evaluation step. The process stops once there is no change to the policy.



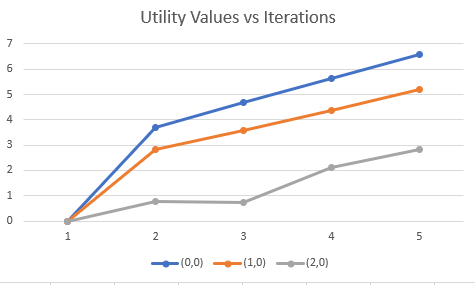
**VALUE ITERATION – PLOT OF OPTIMAL POLICY**

****

**POLICY ITERATION – UTILITY VALUES**



**POLICY ITERATION – UTILITY AS A FUNCTION OF ITERATIONS**



**APPLICATIONS TO LARGER WORLDS**

We created a larger world with dimensions (12,12) and tested both value and policy iteration on the world. Value iteration was able to achieve convergence with a similar number of iterations (459 vs 688). However, policy iteration did not achieve convergence within a reasonable time. The increase in the number of states allowed value-iteration to learn the right policy within a reasonable increase in time, but policy iteration failed to achieve results