Final Project Specification
April 17, 2015

Project Title: "Machine Learning for Tic-Tac-Toe"
Team Members: Vincent Chow (chow@college.harvard.edu), Stephen Albro (salbro@college.harvard.edu), Brandon Price (bprice@college.harvard.edu), and Peter Hickman (phickman@college.harvard.edu)
TF Advisor: Armaghan Behlum

Detailed Description:
Signatures/Interfaces (see our GitHub Repository) with pseudocode -

- initializer.py
    - types:
        - type player = X | O
        - type action = int * int (* coordinates on gameboard *)
        - type board = (*array of arrays *)
        - type state = board * player
    - functions:
        - new () : state = (*empty board & player to move first*)
        - nextState(state, action) : state
        - Eval (state): player | tie | continue
        - opponent (player) = (* other player *)
        - valid(state, action) : bool = (* checks whether action is legal *)
- matrices.py
    - functions
        - makeKey (state) : string (* takes state/creates string representation *)
        - nextKey (state) : makeKey (nextState, opponent(player))
        - addKey (string) : (* adds row to rewardTable & qTable with this key, reward is 0 and q-value is 0 *)
    - setup matrices
        - type stateTable = dict (* dictionary with states as keys and actTables as values *)
        - type actTable = dict (* dictionary with squares as keys and floats as values *)
        - val mutable qTable : stateTable = (* floats [keys in inner actTable] are q-values; we'll update this *)
        - val rewardTable : stateTable = (* floats [keys in inner actTable] are rewards [0, 1, or -1]; don't update this *)

        - value (board, square, qTable): float (* looks up q-value for board and next move square*)
- learning.py
    - chooseMove (state) : (state*square)= (* identifies the state that maximizes the Q* algorithm and makes that move *)
    - val gamma : int = 0.8 (* discount factor *)
    - updateQvalues (stateKey, action, nextKey, reward) = (* updates the qTable using the function Q*(s,a) <- r + gamma*min_(a')Q*(s',a') *)
- graphics.py
    - functions
        - draw (state) : (displays gameboard)
- main.py (main file calls other modules and runs machine learning program)
    - gameLearning (maxGames) (* plays games until maxGames reached *)

Timeline:

- Week of 04/12 - 4/18:
    - setup distributed version control
    - create signatures/interfaces for modules
    - understand Q learning
    - begin implementations and pseudocode
- Week of 04/19 - 4/25:
    - drawn board interacts with user event (e.g. mouse clicks)
    - modules partially implemented
        - e.g. progress made on updating Q learning algorithm
        - e.g. capable of instantiating statetables
- Week of 04/26 - 5/2:
    - modules fully implemented
    - program capable of playing two algorithms against each other
    - graphics visualizes the game of two algorithms playing against each other
- Week of 5/3 - 5/6:
    - Humans compete against tic-tac-toe
    - Compare versions with different chooseMove and updateQvalue functions
    - Have different versions of algorithm competing
    - Reach: implement checkers
        - Create new reward matrix
        - Different set of valid moves, new dictionaries

Progress Report:

- We have successfully setup a git repository for collaborating on this project, and given Army access to view our code
- GUI interface has been partially implemented; capable of rendering a state of a tic-tac-toe board.
- Understanding of Q-learning and TD learning has been achieved, and we have pseudocode and signatures to begin implementing these algorithms
- We've found solid reference material from textbooks and other courses to teach us what we know about Reinforcement Learning and its application to Tic-tac-toe.
- We have two ideas for how to implement the chooseMove function, one using modified roulette wheel selection and the other using a (as yet) un-named formula from Tom Mitchell's *Machine Learning*.