

Spring 2004 - CSI 535 – Introduction to Artificial Intelligence: Term Project

30% of Final Grade,

You must select **one** of the four topics to work on.

I will pair you up with another partner. Note well, this is **not** a joint project but you can share common *infrastructure* code, but not any code that implements the techniques described in this course. Please nominate what common code is shared.

Part A) Due 04/05/2004 5pm. Select a topic and email to me. Missing this deadline costs 10 points per day.

Part B) Questions 1) and 2): Due: 04/12/2004 at 9am for topics 1-3. For topic 4 produce a detailed sketch of your line of attack.

One grade penalty for each week (including partial weeks) late.

Part C) Remaining questions due 05/10/2003 5pm. No extensions are possible. Zero grade if not handed in by this date.

Please read all instructions carefully.

Topic #1: Playing 3D Tic-Tac-Toe with Mini-Max and Alpha-Beta Pruning

You will write agents to solve 4-in-a-row Tic-Tac-Toe in a 4x4x4 cube. You will design an agent to play crosses (X) and I will tell you how noughts (O) will play. Your agent will be MAX. **Read the question clearly, answer all questions succinctly, and make sure you answer addresses the question.**

Note, for questions 1 through 4 you will need to turn in the moves that each agent made for each game and the result. You will also need to turn your code in.

Question 1) (20 points)

Write a program (any language) that plays Tic-Tac-Toe using the mini-max strategy. Crosses will play a **general purpose** evaluation function of your choice looking no more than five moves (both your own and the opponents) in advance. The utility of your evaluation function is important. You cannot use the same function as your opponent. Your opponent (noughts) in this question will use the evaluation function that follows and also look no more than five moves (both players) in advance.

Number of open rows/columns/diagonals for noughts - Number of open rows/columns/diagonals for crosses + `RANDBETWEEN[0,1]`

A row is open, for say noughts, if and only if noughts can win the game using that row. The game ends in a draw if there are no open rows.

Describe your evaluation function. Recall that the intermediate node in a tree should approximate the leaf nodes under it.

Play 100 games (episodes), reporting on how many times your agent won, the average (and variance) number of moves and the average (and variance) number of nodes searched in the game tree. Describe using the PEAS description of the task environment the above situation (page 38, RN) (5 points). Describe the properties of the environment (page 40, RN) (5 points).

Question 2) (20 points)

Change your implementation from question 1 to incorporate alpha-beta pruning.

Once again play 100 games (episodes), reporting on how many times your agent won, the average (and variance) number of moves and the average (and variance) number of nodes searched in the game tree

Question 3) (30 points)

Alpha beta pruning of the game tree is most profitable when the moves are ordered in monotonically decreasing or increasing order of the evaluation function. Suggest a **principled** method of determining the order of how to expand nodes in the game tree. This will depend on your evaluation function.

Play 100 games (episodes), reporting on how many times your agent won, the average (and variance) number of moves and the average (and variance) number of nodes searched in the game tree. The expected number of nodes searched in the game tree should be less than Question 2). Empirically verify the theoretical bound on the best case analysis discussed in class.

Question 4) (30 points)

Play your agent against an agent who has learnt a strategy using Q-learning (topic #2). Play 1000 games (episodes), reporting on how many times your agent won, the average (and variance) number of moves and the average (and variance) number of nodes searched in the game tree.

Suggest an improvement to your agent. Demonstrate it improves its performance.

Topic #2: Learning to Play 3D Tic-Tac-Toe with Q-Learning

In this project you will write a **learning** agent to play 4-in-a-row Tic-Tac-Toe in a 4x4x4 cube. **Read the question clearly, answer all questions succinctly, and make sure your answers addresses the question.**

Write a program (any language) that plays Tic-Tac-Toe using any reasonable scheme. For example a reasonable scheme could be to use the next action according to:

Number of open rows/columns/diagonals for noughts - Number of open rows/columns/diagonals for crosses.

Your aim is to use Q-learning to learn a policy that can hopefully defeat this (and other schemes).

Question 1) (15 points)

Describe the states, actions, delta function and reward function. Note you cannot assume your agent has access to the delta or reward function. Is the environment (as you model it) deterministic or stochastic?

Describe using the PEAS description of the task environment the above situation (page 38, RN) (5 points)

Describe the properties of the environment (page 40, RN) (5 points).

Question 2) (20 points)

Document the Q-learning algorithm you will use in detail. After the algorithm has estimated the Q table, describe your policy to use the Q estimates. Be sure to read the section in RN on exploration versus exploitation. Train you agent for 100 episodes. Illustrate that the agents performance has improved.

Question 3) (20 points)

Allow your Learning Agent to play against its opponent for 1000 episodes. Plot the proportion of wins per ten game intervals as a function of the number of episodes.

Reproduce your Q table estimates after the 1000 episodes.

Question 4) (25 points)

Allow your Learning Agent to play against **itself** for 10,000 episodes. Plot the proportion of wins per ten game interval as a function of the number of episodes.

Reproduce your Q table estimates of both agents after 10,000 episodes.

Question 5) (20 points)

With one of the Q table's learnt in question 4) allow your agent to play against another class-mate who has chosen topic #1 one thousand times. Plot the proportion of wins per ten game interval as a function of the number of episodes both with and without your agent updating its Q-table.

Compare, contrast and discuss your results in questions 4) and 5). Suggest and justify a strategy to train a reinforcement learning agent to play 3D Tic-Tac-Toe against a **variety** of opponents.

Topic #3): Robocup – Simulation League

This topic will consist of designing autonomous agents that can kick a soccer ball into the goal. The agents can not communicate between each other. **Do not use material available on the Internet or outside of this course to answer these questions, if you do so, you will lose points.**

Question 1) (10 points) – Understanding the Environment

Consider a **single** agent designed to play soccer. At any given time the agent can query the state of the game and knows:

- a) If a ball is in their range of vision.
- b) You are in possession of the ball.
- c) If the goal is in their range of vision.
- d) If another player of the same team is in their range of vision.

A player can perform four actions

- a) Dash forward
- b) Turn right 90 degrees
- c) Turn left 90 degrees
- d) Kick the ball

Describe using the PEAS description of the task environment the above situation (page 38, RN) (5 points)

Describe the properties of the environment (page 40, RN) (5 points).

Background to Running the Robo-Cup simulator.

The machine `steve.cs.albany.edu` contains an account for each registered student. Your account login is your student ID, email me for a password. You may only access this machine using SSH. The ROBOCup Soccer simulator is a graphics intensive program, so you will need to do the following:

- 1) Perform `Xhost +steve.cs.albany.edu` from your local machine.
- 2) `ssh -l <your login> steve.cs.albany.edu`
- 3) After logging in then set the environmental variable `DISPLAY` to your local machine:0.0 ie. In a BASH shell, “`export DISPLAY=mymachine.cs.albany.edu:0.0`”
- 4) Test the connection by seeing if you can display `xclock`, if you can then read the `README` file in your directory to determine how to start the simulator and run an agent.

Note a team is always five agents.

Question 2) (20 points) – A Reflex Agents

The file `Sample.c` contains a very simple agent. You will modify this code and try to improve on its performance. Executing the script `StartAgents.sh` will start five agents. For the agent implemented in `Sample.c` report the times taken for the team to score a goal for **ten** kick offs.

Modify the code supplied in `Sample.c` to create a reflex agent. Provide a complete table stating each of your agents reflexes (5 points). Spend one paragraphs describing why you chose your agent to have the particular reflexes described (5 points). Save the reflex agent in the file `Reflex<YourStudentId>.c`. For this agent report the times taken for the team to score a goal for **ten** kick offs (10 points).

Question 3) (45 points) – A Belief Network Agent

The previous agent did not model the environment. You will now create a cyclic graph-based agent. For this agent each node in the graph represents *a piece of information* of the situation the agent is in. The arcs between the nodes represent causation or correlation, you may use either so long as your usage is consistent. Some nodes are given such as the game states a) b) and c) and actions described earlier. The remaining nodes are essentially latent and you must create them to provide intelligence (hopefully) to your agent. It may be beneficial to consider each node as being a cause or an effect.

Draw a graphical representation of your agent, explain succinctly why you choose your proposed architecture. Highlight which nodes are causes and which are effects (5 points)

At any point in time, how will you determine the action to perform (if any)? Show how you will calculate the probability your agent will dash forward using a sampler. Describe the sampler you will use. Include psuedo code for your sampler. (30 points).

Implement the agent and store it in a file `BeliefNet<YourStudentId>.c`. For this agent report the times taken for the team to score a goal for **ten** kick offs. (10 points).

Question 4) (25 points) – Competition

Choose a class-mate who is also doing topic #3. You are to schedule a best of three series of games. Produce a detailed two page report on the success or otherwise of your approach. Suggest an improvement to your belief network topology that could improve its performance. You may also wish to consider a learning component to your network. Implement the improvement and report the subsequent results of another three game series.

Topic #4): Analysis and Improvement of Algorithms – Term Paper Options (Needs my approval)

The Q-learning algorithms proof of convergence is rather unsatisfying as it requires visiting each state action pair and infinite number of times. Develop a useful set of bounds on how close the table estimates will be if:

- a) Each state action pair is visited m times on average/at most/at least

OR

Using sampling for inexact inference raises the issue of the number of draws from the joint distribution to make. Consider the query $P(Q|E)$. If our aim is to get an estimate that is within $\pm\epsilon$ with a chance of failure δ construct a lower and/or bound on the number of draws from the joint distribution to obtain this result. You may use any one of a number of standard bounds due to the various laws of large numbers such as a Chernoff, Chebychev or Markov inequality. For details see: Davidson and Aminian, *Using the Central Limit Theorem to Learn Belief Networks*, 8th A.I. and Math Symposium, 2004.

Note you cannot assume (without proof) that the sampler immediately converges to the stationary distribution.

Empirically verify the bound(s) you construct.

OR

Propose a theory question of your own.