UC Berkeley – Computer Science

CS188: Introduction to Artificial Intelligence Josh Hug and Adam Janin

Midterm I, Fall 2016

This test has **8** questions worth a total of **100** points, to be completed in 110 minutes. The exam is closed book, except that you are allowed to use a single two-sided hand written cheat sheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided.

Write the statement out below in the blank provided and sign. You may do this before the exam begins. Any plagiarism, no matter how minor, will result in an F.

'I have neither given nor received any assistance in the taking of this exam."		
	Signature:	
Name:	Your EdX Login:	
SID:	Name of person to left:	
Exam Room:	Name of person to right:	
Primary TA:		

- O indicates that only one circle should be filled in.
- o indicates that more than one box may be filled in.
- Be sure to fill in the \(\cap \) and \(\cap \) boxes completely and erase fully if you change your answer.
- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- There are a lot of problems on this exam. Work through the ones with which you are comfortable first. **Do not get overly captivated by interesting problems or complex corner cases you're not sure about.** Fun can come after the exam: There will be time after the exam is try them again.
- Not all information provided in a problem may be useful.
- Write the last four digits of your SID on each page in case pages get shuffled during scanning.

Problem	1	2	3	4	5	6	7	8
Points	11.5	9.5	14.5	12	8.5	16	19	9

Optional: Mark along the line to show your feelings	Before exam:	[:(😊 🛚
on the spectrum between :(and \odot .	After exam:	[:(⊙]

CS188 MI	DTERM I,	FALL	2016
Last 4 SID:			

1. Munching (11.5 pts)

i) (7.5 pts) Suppose Pac-Man is given a map of a maze, which shows the following: it is an *M*-by-*N* grid with a single entrance, and there are **W** walls in various locations in the maze that block movement between empty spaces. Pac-Man also knows that there are **K** pellets in the maze, but the pellets are not on the map, i.e. **Pac-Man doesn't know their locations in advance**. Pac-Man wants to model this problem as a search problem. Pac-Man's goal is to plan a path, before entering the maze, that lets him eat the hidden pellets as quickly as possible. **Pac-Man can only tell he has eaten a pellet when he moves on top of one.**

Which of the following features should be included in a minimal correct state space representation? For each feature, if it should be included in the search state, calculate the size of this feature (the number of values it can take on). If it should **not** be included, **very briefly** justify why not.

Feature	Include?	Size (if included) or Justification (if not included)
Current location	• Yes / O No	Size: MN Justification: We need to keep track of where Pac-Man is.
Dimensions of maze	O Yes / ● No	Justification: This is an input to the problem, and does not need to be included.
Locations of walls	O Yes / ● No	Justification: This is an input to the problem, and does not need to be included.
Places explored	• Yes / O No	Size: 2^{MN} Justification: We need to visit every square in the maze in order to ensure that we eat all of the pellets, so we must track where we've explored.
Number of pellets found	O Yes / ● No	Justification: We don't know where the pellets are, so we <i>can't</i> know when we've found a pellet when we're planning a path.
Time elapsed	O Yes / ● No	Justification: This does not need to be in the state, but is instead part of the cost of transitioning between states.

			Ve need to visit every of the pellets, so we m		maze in order to ensure we've explored.
Number of pellets found	O Yes / • No		Justification: We don't know where the pellets are, so we <i>can't</i> know whe we've found a pellet when we're planning a path.		
Time elapsed	O Yes / • No		Justification: This does not need to be in the state, but is instead part of to cost of transitioning between states.		
to one of the followin State Space S iii) (2 pts) Now assulocations and colors	earch Comme that there are sof all the pelle i.e. one red, one	SP \bigcirc Minimax $\in K_R$ red pellets, K_G goets, and that Pac-Mane green, and one blue	\bigcirc Expectimax reen pellets, and K_B blawants the shortest p	MDP ue pellets, that path that lets	ORL t Pac-Man knows the him eat at least one state space, assuming
Solution: The state n any pellets of that col	•	ck of where Pac-Man	is, and, for each color	, whether or n	ot Pac-Man has eaten
• •	broach is best su	uited for solving the	one-pellet-of-each-co	lor problem?	Completely fill in the
• State Space So	earch OCS	SP	Expectimax	\bigcirc MDP	○RL

CS188 MI	DTERM I, FALL 2016
Last 4 SID:	

2. Uninformed Search (9.5 pts)

Part A: Depth Limited Search (DLS) is a variant of depth first search where nodes at a given depth L are treated as if they have no successors. We call L the **depth limit.** Assume all edges have uniform weight.

i) (1 pt) Briefly describe a major <u>advantage</u> of <u>depth</u> <u>limited</u> <u>search</u> over a simple depth first tree search that explores the entire search tree.

Solution: Because we define a depth limit in DLS, we won't get stuck exploring an infinite path.

ii) (1 pt) Briefly describe a major <u>disadvantage of depth limited search</u> compared to a simple depth first tree search that explores the entire search tree.

Solution: Because we define a depth limit in DLS, if the solution is beyond the depth limit, we will not reach it.

Part B: IDDFS. The iterative deepening depth first search (IDDFS) algorithm repeatedly applies depth limited search with increasing depth limits L: First it performs DLS with L=1, then L=2, and so forth, until the goal is found. IDDFS is neither BFS nor DFS, but rather an entirely new search algorithm. Suppose we are running IDDFS on a tree with branching factor B, and the tree has a goal node located at depth D. Assume all edges have uniform weight.

i) (2 pts) Is IDDFS optimal? • Yes / \bigcirc No Is IDDFS complete? • Yes / \bigcirc No **Solution:** If the solution is at depth D, IDDFS is guaranteed to find it on the Dth iteration of the algorithm (when it runs DLFS with depth D).

ii) (1 pt) In the worst case asymptotically, which algorithm will use more memory looking for the goal?

○ IDDFS / • BFS

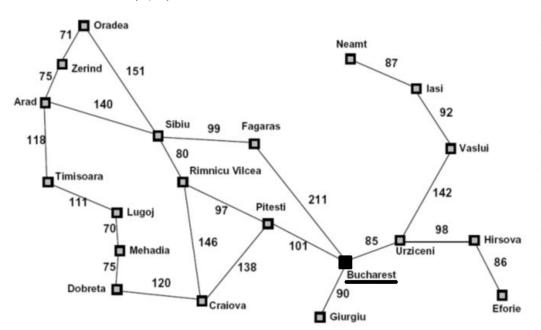
Solution: DLS and IDDFS both use asymptotically the same amount of memory as DFS (which only requires keeping track of the path to the current state), which is less than BFS (which requires keeping track of the entire fringe.)

iii) (2.5 pts) In project 2, you used DLS to implement the minimax algorithm to play Pac-Man. Suppose we are instead writing a minimax AI to play in a speed chess tournament. Unlike the Pac-Man project, there are many pieces that you or your opponent might move, and they may move in more complicated ways than Pac-Man. Furthermore, you are given only 60 seconds to select a move, otherwise you are given a random move. **Explain why** we'd prefer to implement minimax using IDDFS instead of DLS for this problem, and **explain how** would we need to modify IDDFS (as described above) so that it works for this problem.

Solution: With a pre-set depth L, DLS could either run out of time or not utilize all the time (L could be set too deep or too shallow). IDDFS will not have this problem. It would have to be modified such that the best found answer would be returned once time is up. (Possible answers for this include: keeping track of the time, keeping track of the best found answer at each layer)

3. Aliens in Romania (14.5 pts)

Here's our old friend, a road map of Romania. The **edges** represent **travel time along a road**. Our goal in this problem is to find the shortest path from some arbitrary city to Bucharest (underlined in black). You may assume that the heuristic at Bucharest is always 0. In this question, **L(X, Y)** represents the travel time if we went in a **straight line from city X to city Y**, ignoring <u>roads</u>, <u>teleporters</u>, <u>wind</u>, <u>birds</u>, etc. You should make no other assumptions about the values of **L(X, Y)**.



Part A: Warm up.

i) (6 pts) For each of the heuristics below, completely fill in the circle next to "Yes" or "No" to indicate whether the heuristic is admissible and/or consistent for **A* search**. You should fill in 6 answers (one has been provided for you). **Reminder**, **h(Bucharest)** is always zero.

Heuristic h(C) =	[except C = Bucharest]	Admissible	Consistent
0		• Yes / ONo	• Yes / O No
90		O Yes / ● No	O Yes / ● No
L(C, Bucharest)		• Yes / O No	• Yes / O No
min(L(C, Bucharest), 146)		• Yes / O No	

ii) (2.5 pts) For what values of x is h(C) = max(L(C, Bucharest), x) guaranteed to be admissible? Your answer should be in the form of an expression on x (e.g. "x = 0 OR x > 102").

Solution: If x > 85, then h(Urziceni) will be greater than 85, which would be an overestimate of the true cost to reach Bucharest from Urziceni (which is at most 85). Thus, we want $x \le 85$. (Notice that this is a strict inequality; x = 85 is fine.)

CS188 MI	DTERM I, FALL 2016	í
Last 4 SID:		

Part B: Teleporters

Aliens build a teleporter from Oradea to Vaslui. This means that we can now get from Oradea (top left) to Vaslui (near the top right) with zero cost.

- i) (1 pt) Update the drawing on the <u>previous page</u> so that your graph now reflects the existence of the teleporter. **Solution:** An arc between Oradea and Vaslui with weight 0.
- ii) (5 pts) Completely fill in the circle next to "Yes" or "No" to indicate which of the following are guaranteed admissible for A* search, given the existence of the alien teleporter. Reminder, h(Bucharest) is always zero, and L(A, B) is the travel time between A and B assuming a straight line between A and B ("as a bird flies").

Heuristic h(C) =	[except C = Bucharest]	Admissible
77		• Yes / O No
L(C, Bucharest)		O Yes / ● No
max(L(C, Bucharest), 80)		O Yes / ● No
max(L(C, Bucharest), L(C, Oradea))		O Yes / ● No
min(L(C, Bucharest), L(C, Oradea))		• Yes / O No

Solution: A key insight is that from C, we need to either go directly to Bucharest without using the teleporter, or go to Oradea before using the teleporter. However, neither, by itself is an admissible heuristic, so a *max* expression involving L(C, Bucharest) is necessarily inadmissible.

Survey Completion

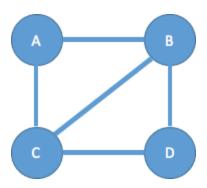
Enter the secret code for survey completion:	
If you took the survey but don't have the secret	code, explain:

4. CSPs (12 pts)

In a combined 3rd-5th grade class, students can be 8, 9, 10, or 11 years old. We are trying to solve for the ages of Ann, Bob, Claire, and Doug. Consider the following constraints:

- No student is older in years than Claire (but may be the same age).
- Bob is two years older than Ann.
- **B**ob is younger in years than **D**oug.

The figure below shows these four students schematically ("A" for Ann, "B" for Bob, etc).



i) (2 pts) In the figure, draw the arcs that represent the binary constraints described in the problem.

Note: Two directed arcs in place of each undirected arc is also acceptable.

ii) (1 pt) Suppose we're using the AC-3 algorithm for arc consistency. How many total arcs will be enqueued when the algorithm begins execution? Completely fill in the circle next to one of the following:

$$\bigcirc 0 / \bigcirc 5 / \bigcirc 6 / \bigcirc 8 / \bullet 10 / \bigcirc 12$$

Solution comment: Remember that a relationship between two variables will result in two arcs (since arcs are directed.)

iii) Assuming all ages {8, 9, 10, or 11} are possible for each student before running arc consistency, manually run arc consistency on **only** the arc *from* A *to* B.

A. (2 pts) What values on A remain viable after this operation? Fill in all that apply.

B. (2 pts) What **values on B** remain viable after this operation? Fill in all that apply.

Solution comment: Remember that enforcing the arc $A \rightarrow B$ should never change the viable domain of B.

C. (1 pt) Assuming there were no arcs left in the list of arcs to be processed, which arc(s) would be added to the queue for processing after this operation?

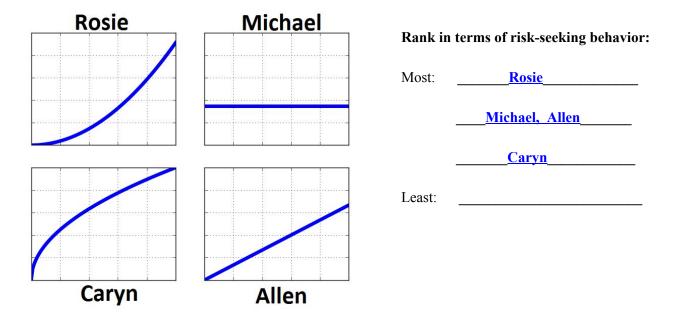
Solution: $B \to A$, $C \to A$ (in either order). They should be directed arcs, since that's critical to the algorithm.

iv) (4 pts) Suppose we enforce arc consistency on all arcs. What ages remain in each person's domain?

Ann: 8 Bob: 10 Claire: 11 Doug: 11
Solution comment: If we enforce the arcs $B \to A$, $B \to D$, $D \to B$, $A \to B$, $C \to D$, then we can quickly find the only satisfying solution.

5. Utilities (8.5 pts)

Part A: Interpreting Utility. (3.5 pts) Suppose Rosie, Michael, Caryn, and Allen are four people with the utility functions for marshmallows as described below (e.g. Michael doesn't care how many marshmallows he gets). The x-axis is the number of marshmallows, and the y-axis is the utility. Rank our four TAs from least risk-seeking (at the bottom) to most risk-seeking (at the top) using the four lines provided. If two people have the same degree of risk-seeking, put them on the same line. You may not need all four blanks (e.g. if two people are on the same line).



Solution explanation: This question asked the student to identify if a function was risk-seeking, risk-neutral, or risk-averse, based on the function's graph. Visually, we can see that Rosie's utility function is concave up (i.e. convex) and thus must be risk-seeking, Caryn's utility function is concave down and thus risk-averse, while both Michael's and Allen's functions are linear, and thus both have risk-neutral utility functions.

For visual intuition as to why concave up functions are risk-seeking: imagine connecting two points on the function's curve with a straight line segment. Notice that this line segment is above the curve. Now, for any point on this line segment, we can imagine a lottery between the two endpoints, such that the coordinates of this point are (the expectation of the lottery, the expected utility of the lottery), while the point below it on the curve is the utility of the expectation of the lottery.

Conversely, a concave down function is risk-averse, because the line segment (the expected utility of the lottery) is now below the curve (the utility of the expected value of the lottery), while a linear function is risk-neutral, because the line segment coincides with the curve.

Part B: Of Two Minds. (5 pts) Some utility functions are sometimes risk-seeking, and sometimes risk-averse. Show that $U(x) = x^3$ is one such function, by providing two lotteries: one for which it is risk-seeking, and one for which it is risk-averse. Provide justification by filling out the table below. Averse is just the opposite of "seeking". Specifically, it means "having a strong dislike for something," in this case, risk.

Example solution:

Example solution.			
	Risk-seeking:	Risk-averse:	
Lottery	[1/2, 0; 1/2, 2]	[1/2, 0; 1/2, -2]	
Utility of lottery	$\frac{1}{2} * 0^3 + \frac{1}{2} * 2^3 = 4$	$\frac{1}{2} * 0^3 + \frac{1}{2} * (-2)^3 = -4$	
Utility of expected value of lottery	$\left(\frac{1}{2} * 0 + \frac{1}{2} * 2\right)^3 = 1$	$\left(\frac{1}{2} * 0 + \frac{1}{2} * - 2\right)^3 = -1$	

Solution explanation: In general, for the given utility function, any lottery that can only take nonnegative values is risk-seeking, and any lottery that can only take on nonpositive values is risk-averse. (Lotteries that can take on both positive and negative values require a bit more effort to assess.)

The intention was for students to identify that, because the utility function is an odd function, simply negating the risk-seeking lottery would acquire a risk-averse function.

Common Mistakes:

- Not knowing how to identify a risk-seeking vs risk-averse function.
- More specifically: providing a fixed value as a lottery. (Note that we don't consider fixed values as lotteries, but even if we did, such a lottery would never, under any utility function, be anything other than risk-neutral.)
- Not knowing what "utility of lottery" vs "utility of expected value of lottery" means; more students had trouble with "utility of expected value of lottery".
- When calculating the expected value of the lottery: some students gave lotteries with a 50-50 chance between two rewards, correctly divided by two when calculating E[L], but then divided by two again after calculating U(E[L]).

Advice:

- As noted above, having one lottery be the negative version of the other halves the needed amount of computation for this problem.
- In general, choosing lotteries with easy numbers makes computation easier as work, For example, 0^3 is easy to compute; 5^3 is slightly less so. Multiple students had 100 as an outcome, and then lost a 0 or had some other careless error when calculating 100^3 .
- Similarly, choosing simple lotteries (e.g. only two possible outcomes that are equally likely) simplifies things as well.

6. Games / Mini-Max: The Potato Game (16 pts)

Aldo and Becca are playing the game below, where the left value of each node is the number of potatoes that Aldo gets, and the right value of each node is the number of potatoes that Becca gets (i.e. Aldo: left, Becca: right). Unlike prior scenarios where having more potatoes results in more utility, Becca and Aldo will have a more complex view of what makes a "good" distribution of potatoes.

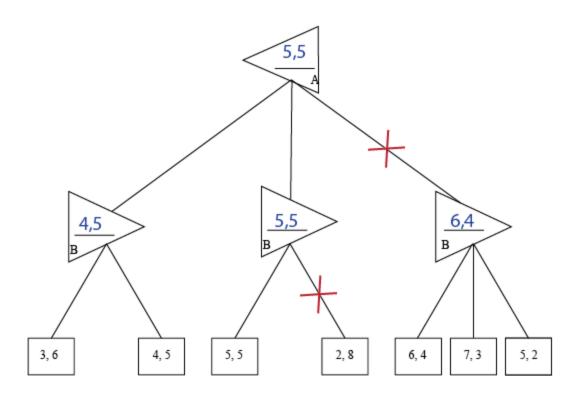
Becca will use the following thought process to decide which move to take:

- Among all choices where she gets at <u>least</u> as <u>many</u> potatoes as Aldo, she'll pick the one that maximizes Aldo's number of potatoes (very nice!).
- If there are no choices where she gets at <u>least as many</u> potatoes as Aldo, she'll simply maximize her own potato count (ignoring Aldo's value).

Aldo will do the same thing, but substituting "he" for "she", "her" for "his", and "Becca" for "Aldo". The rules above effectively tell us how Becca (and Aldo) rank the utility of any set of choices.

i) (4 pts) Fill in the blanks below with the choice that each player would make at each stage. Assume that both Aldo and Becca are trying to maximize their own utility as described above.

ii) (3 pts) Assume that Aldo and Becca know that **the sum at any leaf node is no more than 10**. Cross out the edges to any nodes that can be pruned (if none can be pruned then write "no pruning possible" <u>below the tree</u>).

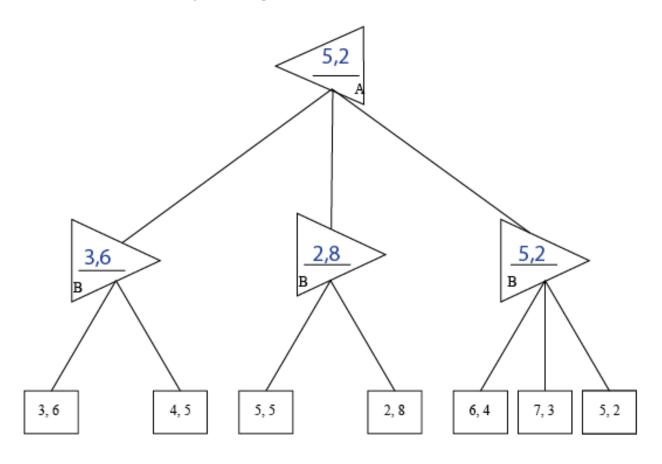


iii) (3 pts) Describe a general rule for pruning edges, assuming that the sum at any leaf node is no more than 10.

Solution: Prune all remaining edges after seeing a node with (5, 5).

Solution explanation: Since the sum at any leaf node is no more than 10, we know (5, 5) is the best possible (utility-maximizing) distribution of potatoes for <u>both</u> players, so we do not need to consider any other nodes. Note that based on the utility calculation rules, (5, 5) is better than other equal-potato distributions, like (4, 4) or (3, 3).

iv) (4 pts) Suppose that **Becca attempts to minimize Aldo's utility instead of maximizing her own utility, and that Aldo knows this, and tries to maximize his own utility -- taking into account Becca's strategy.** Repeat part ii. There is no need to cross out edges that are pruned.



v) (2 pts) Suppose that Becca chooses uniformly randomly and Aldo knows this and tries to maximize his own utility -- taking into account Becca's strategy. Which move will Aldo make to maximize his expected utility, assuming he treats Becca as a chance node: left, middle, or right? If there is not enough information, pick "Not Enough Information".

○ Left	Middle	Right	 Not Enough Information
--------	--------	-------	--

Solution: We do not know what Aldo's specific utility function is, so we lack information to calculate his average utility between two outcomes, which we need to compute the assigned value for a chance node.

7. The Spice Must Flow (19 pts)

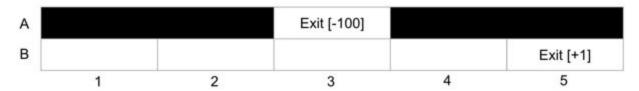
i) (3 pts) Suppose we have a robot that has 5 possible actions: $\{\uparrow, \downarrow, \leftarrow, \rightarrow, \text{Exit}\}$. If the robot picks a direction, it has an **80%** chance of going in that direction, a **10%** chance of going 90 degrees counterclockwise of the direction it picked (e.g. picked \rightarrow , but actually goes \uparrow), and a **10%** chance of going 90 degrees clockwise of the direction it picks.

If the robot hits a wall, it does not move this timestep, but time still elapses. For example, if the robot picks \downarrow in state B1 and is 'successful' (80% chance), it will hit the wall and not move this timestep. As another example, if it picks \rightarrow in state B1, but gets the 10% chance of going down, it will hit the wall and not move this timestep

In states B1, B2, B3, and B4, the set of possible actions is $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$. In states A3 and B5, the only possible action is $\{\text{Exit}\}$. Exit is always successful. The blackened areas (A1, A2, A4, and A5) represent walls.

All transition rewards are 0, except R(A3, Exit) = -100, R(B5, Exit) = 1.

In the boxes below, draw an optimal policy for states B1, B2, B3, and B4, if there is no discounting, i.e. $\gamma = 1$. Use the symbols $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$.



Solution: Any policy that went down at B3 and that never went left would work (i.e. guarantee never landing in the A3 exit and eventually reach the B5 exit).

ii) (2 pts) Give the expected utility for states B1, B2, B3, and B4 under the policy you gave above.

$$V^*(B1)$$
: ____1__ $V^*(B2)$: ____1__ $V^*(B3)$: ____1__ $V^*(B4)$: ____1

Solution: There is no discounting, so we don't have to worry about how long it takes to get to the exit; as long as we can guarantee that we will *eventually* reach the good exit (as we can with the above optimal policy), the optimal value of every state is going to be 1.

iii) (2.5 pts)What is V*(B4), the expected utility of state B4, if we have a discount $0 < \gamma < 1$ and use the optimal policy? Give your answer in terms of γ .

$$V^*(B4)$$
: $\frac{0.8\gamma}{1-0.2\gamma}$

Solution: Any optimal policy requires going right from B4. We can calculate this in one of two ways:

Direct Approach: The expected reward from B4 is equal to γ^N where N is the number of time steps until we get to B5; following the optimal policy, we have N following a geometric distribution with p = 0.8. Thus, we can calculate:

$$V^*(B4) = E[\gamma^{Geom(0.8)}] = \sum_{i=1}^{\infty} \gamma^i * P(Geom(0.8) = i) = 0.8\gamma \sum_{i=0}^{\infty} (0.2\gamma)^i = \frac{0.8\gamma}{1 - 0.2\gamma}$$

Recurrence Approach: If we try to go right and move successfully, we will get a reward of $\gamma * 1$; otherwise, we are at the same location (and have expected reward $\gamma * V^*(B4)$). Thus, we can use the recurrence to get $V^*(B4) = 0.8 * \gamma + 0.2 * \gamma * V^*(B4)$, which we can solve to get $V^*(B4) = \frac{0.8\gamma}{1-0.2\gamma}$.

Part B: Spice. Now suppose we add a sixth special action **SPICE**. This action has a special property that it always takes the robot **back to its current state**, with a reward of 0. This action might seem useless, except for one thing the robot will always get its top preference on its next action (instead of being subject to noise).

For example, suppose that the robot is in state s1 and picks action a which has a chance p2 of moving to s2, and a chance p3 of moving to s3. If the robot takes the **SPICE** action at timestep t, and a at timestep t+1, the robot will not end up randomly in s2 or s3, but will instead go to its **preference** of s2 or s3. It will still collect $R(s1, a, s^2)$, which is unchanged. Preferring s2 or s3 does not take an additional time step, e.g. if the robot is in B1 at timestep 0, chooses SPICE, then chooses \downarrow but prefers the outcome B2, the robot arrives in B2 at timestep 2.

The powers granted by **SPICE** last one timestep. **SPICE** may be used any number of times.

i) (2 pts) Give an optimal policy for B1, B2, B3, and B4, assuming no discounting, i.e. $\gamma = 1$. On the left of each slash, write the optimal action if the previous action was not SPICE, and on the right side of each slash, write the optimal action if the previous action was SPICE, assuming the robot always "prefers" the outcome that points in the same direction as its action (e.g. if it picks \rightarrow , it prefers going right). In total, you should write 8 actions. Use the symbols $\{\uparrow, \downarrow, \leftarrow, \rightarrow, S\}$, where "S" represents indicate the spice action.

А			Exit [-100]		
В	1	1	1	1	Exit [+1]
	1	2	3	4	5

Solution: Since there was no discounting, there were a number of viable solutions. There were three particularly common solutions.

- 44% of students said to go right for every state except B3 without SPICE, where they said to use SPICE instead.
- 21% of students said to go right for every state except B3 without SPICE, where they said to go down instead.
- 19% of students simply said to always use SPICE to deterministically move right, regardless of location.

There are a few actions that would specifically *not* work:

- Going left, up, or right from B3 without SPICE all has a chance of accidentally leading to the bad exit.
- Whether or not it works depends on the other part of the policy, but going "left" was often incorrect because it would likely cause the robot to be caught in a loop and never be able to get to the exit.
- If SPICE is active, and the policy dictates for the robot to use SPICE again, then the robot would definitely be stuck in this state.

In addition, some students assumed that the robot started at B1 (or somewhere else), and then only wrote policies corresponding to a specific path. However, a policy should include an action for *all* states, even ones that might not be reachable from a particular starting state (not that there is a starting state in this problem at all).

CS188 MIDTERM I, FALL	2016
Last 4 SID:	

ii) (2.5 pts) Assuming we have a discount $0 < \gamma < 1$, what is V*(B4) if the previous action was not SPICE? Give your answer in terms of γ . It is OK to leave your answer in terms of a max operation.

$$V^*(B4): \underline{\hspace{1cm}} \max \left(\frac{0.8\gamma}{1 - 0.2\gamma}, \gamma^2 \right) \underline{\hspace{1cm}}$$

Solution: There are two potential policies: the robot can try right repeatedly until it reaches the exit, or it can take SPICE and move right. We thus want to compare the expected reward for each of these policies, and take the max over the two.

The value of the former is simply the same as the answer to part A.i: $\frac{0.8\gamma}{1-0.2\gamma}$.

The value of the latter is equal to γ^2 (since we will deterministically claim a reward of 1 in three time steps).

OFFICIAL RELAXATION SPACE. Draw or write anything here:

Part C: Bellman Equations. (You can do this problem even if you didn't do B, but it'll be harder)

In class, we derived the Bellman Equations for an MDP given below.

$$Q^*(s,a) = \sum_{s' \in S_{(s,a)}} T(s,a,s') [R(s,a,s') + \gamma V^*(s')]$$

$$V^*(s) = \max_{a \in A_s} Q^*(s, a)$$

You'll notice these look slightly different than the version on your cheat sheet, but these equations represent exactly the same idea. The only difference is that to improve the clarity of these equations, we've specified the sets over which we are summing and maximizing. Specifically, $S_{(s, a)}$ is the set of states that might result from the action (s, a), and A_s is the set of actions that one can take in state s. For this problem, assume that A_s does not include the special SPICE action.

i) (3.5 pts) Derive Q*(s, SPICE). Assume that the previous action was not **SPICE**. Your answer should have two max operators in it. Hint: Consider drawing the diagram we used to derive the Bellman Equation.

Solution: We need to choose the next action to take, and SPICE allows us to also choose the next state after that action; this leads to the two maxes that the question suggests. The rest of the recurrence is like the usual Bellman equation, except that we are getting R(s, a, s') at a one time-step delay, and s' is reached two time steps away:

$$\max_{a \in A} \max_{s' \in S_{(s,a)}} \gamma [R(s,a,s') + \gamma V(s')]$$

Common Mistakes:

- Some students included transition probabilities, or took the sum over multiple successor states; this goes against the point of consuming SPICE.
- Some students messed up on applying the discount factors correctly, most commonly forgetting the outermost γ.

ii) (3.5 pts) Derive $V^*(s)$, the value of a state in this MDP. You should account for the fact that the **SPICE** action is available to the robot. Assume that the previous action was not **SPICE**.

Solution: The optimal value of *s* is acquired either by taking a non-SPICE action (where the recurrence is the same as the regular Bellman equation), or by taking SPICE. We can thus calculate this by taking the max over these two possibilities:

$$\max\{Q^*(s, SPICE), \max_{a \in A_s} Q^*(s, a)\}$$

We can also merge these two cases into a single max expression:

$$\max_{a \in A_s \cup \{SPICE\}} \{Q^*(s, a)\}$$

CS188 MI	DTERM I,	FALL	2016
Last 4 SID:			

8. The Reinforcement of Brotherly Love (9 pts)

Consider two brother ghosts playing a game in an M-by-N grid world, Ghost Y is the Younger brother of Ghost O. Each ghost takes 100 steps, trying to pet cats which randomly appear in the maze. The reward for the game is equal to the number of cats petted. Each ghost has a policy for how to run through the maze. Ghost O being the older brother has a policy that has a **higher expected cumulative reward** (i.e. $V_Y(s) \leq V_O(s) \ \forall s \in S$). However, Ghost O is **not necessarily optimal** (i.e. $V_O(s) \leq V^*(s) \ \forall s \in S$). Ghost Y wants to learn from Ghost O instead of starting from scratch.

Part A: Q-Learning

O Yes / ● No

We will now explore how Ghost Y can achieve this, while performing **Q-Learning**. Ghost Y starts out with a Q function **initialized randomly** and wants to catch up to Ghost O's performance. Denote the Q functions for each Ghost as Q_Y , Q_O . In order for Ghost Y to converge to the optimal policy, he must properly balance **exploring** and **exploiting**. Consider the following exploration strategy: At state s with probability 1- ε choose $max_a Q_Y(s, a)$ otherwise choose $max_a Q_O(s, a)$.

(2 pts	s) Is this	guaranteed	to converge to	o the optimal	I value function	V*? Briefly justify.

Solution: Ghost Y is not guaranteed to visit all states and actions infinite times each, which prevents it from observing a potentially better action to apply. In particular, Ghost Y will only ever consider the initial $max_a Q_Y(s,a)$ action or the fixed $max_a Q_O(s,a)$, which may not include the true optimal action.

CS188 MI	DTERM I,	FALL	2016
Last 4 SID:			

Part B: Model-Based RL

Ghost Y decides model free learning is scary, and decides to try to learn a model instead. Ghost Y will watch Ghost O try to catch Pac-Man and estimate the **transition** probabilities and **rewards** from Ghost O's demonstrations (i.e. learn a model for T(s,a,s') and R(s,a,s')).

i) (2 pts) For Ghost Y to learn the correct and exact values of R(s,a,s') for all state/action/next_state transitions, what must be true about the episodes he watches Ghost O perform?

Solution: Ghost O must visit each state/action/next_state transition at least once. (Alternatively: ghost O must try each state/action pair enough times to see all possible successor states.)

Common Mistakes:

- Seeing the transition (s,a,s') once is enough to know exactly what R(s,a,s') is. Some students incorrectly put that Ghost O needs infinite transitions, but that is only the case for learning T(s,a,s') (as below.)
- Some students only tried every action from every state once. This didn't necessarily see every possible successor state.

ii) (2 pts) For Ghost Y to learn the correct and exact values of T(s,a,s') for all state/action/next_state transitions, what must be true about the episodes he watches Ghost O perform?

Solution: Ghost O must visit each state/action/next_state transition infinite times each. (Alternatively: Ghost O must try each action from each state infinite times.)

Common Mistakes:

- Some answers were not specific enough, and only said that ghost O visits each transition "many" or "enough" times.
- Some answers mentioned that there needed to be an infinite number of episodes, and some also noted that the episodes must visit every transition at least once, but did not clearly specify that each transition is visited infinite times.
- Some answers said that each state needed to be visited infinite times, but did not talk about trying each action.
- Conversely, some answers said to try each action infinite times. This is different from trying each action *from each state* infinite times.
- A few answers said that Ghost Y needs episodes that include every state/action pair, and needed to watch these episodes infinite times. This does not guarantee that the finite number of episodes are representative of the true probability distributions.

iii) (2 pts) Assuming T(s,a,s') and R(s,a,s') are learned <u>exactly</u>, what algorithm/s below could be helpful for Ghost Y to decide which actions to take from each state, based <u>only</u> on information in his learned model?

■ Value Iteration ■ Policy Iteration □ Q-Learning □ TD-Learning

Solution: Value Iteration and Policy Iteration. These both can generate a better policy and do not require further information to be gathered.

iv) (1 pt) Let π be any of the policies learned in part iii. Does following π result in maximized expected utility for every state, i.e. is $V^{\pi}(s) = V^{*}(s)$ for all states? [Consider all policies]

Last 4 SID:

DO NOT WRITE ON THIS PAGE. IT WILL NOT BE GRADED.

504B03041400090008005C694549D992860B96110000CA1100000C001C00676C7970686167652E706E675554090003505EF557FC5EF55775780B000104F501000004140000007A41EA7E54C508 5ADCCA44BADCC524A15475544046F748AE7372C6D2CD4E47E8B7E7CB6596470370A4BEBE144CEB2EE0E3E5DFFF994D70C833845C625F91596A82636CD38E61C369058A3AB8B54968A81ADC716B C604AFB759A0E81D88405B66B022B0FE543490278C3891DDDF4EBE49697FF9775EF828ECA9457A271D0E0209516E510E4CEEFD45DF58789D932A1BC43EC722D9D24048AF0F15DBDE9539AC905C39F65ACFE3C6EF24062195C174FAC0A8AD794ADB602124B06415F981E0EC7053722D3C75101B9B30BA9A42A6699B8C3B141934B53346145309A8BEE3381E0957A594E8AE6F66326B56D2BFF0E8 5DA3DA741CB2EDFD18F36026C91BFAF65E3BFFFCF1A40E9CB6901CAA6688A2870182E03562037B144FFE2A611C1BBAE646BCBFF26DC95FD8C4AF85B499C36084074E6906BB08DE2BDB85CAC29B 964F3R7RD136584C50F1C74631F7FFD4F2226F0A13D9024F76F8FD697R13840232198F4R6C00R124FD6F3D103081RFDAD890RA45R6F473R8045F872387RFRR44RF8R7826F8F9F26R12R086F52C 192789370E8666214D446C079A8FA1B0FD65E381F995F7BCF5A004016C0542C83B027959D74D19827A4C0CC8B499A49540D4A942ADA2196766C87AF7E343AA8DA1FDDEDF048C757532FAF42F7B C0AA380D8CF8DAC9C850D1609C4AF36EAD8570F38C9F990E187B3FFF384CA8151A3E112C458736AAFF65291AD73B998F659419E4181E6190203A2BDC0DE0F6A03045EC793DAC9F495675F3B3A7 449C8D0584667258A3C73A6633E5DCE90D18EC1FF8DBA7FC2E9B501972CA674CB6C6ECB981EE88C96CCEA40E77D1966FB0252F8E4D6F7F6DEACA986C3338F097A36654ED06BAC4D4ECE1DD79DDFC0F7AA5C52D129F9D21FBB082CBA1D1CC89A891D91B15BDA4BD8014058A7618A8C31A0A2BDD63C6E3A4D229B90F898168746690FE533C0B0505EA19A49D2F73A48B2B8E85E4AFD1FB6B20F131 ${\tt E130C9A7936FBA7A35111D6ED6F7309DC36C88DA4052D544A12A040FF96D946DB9471DBFDEBC736BA0F29BA703DF982BFAABBAAA001A9580564FC3764F89473D6D6A2381E0C57DD0DAE6298AED}$ D0C75D81EFD81C94C9EFC2AA7EA5900D85EBB706B51B1ACCD1949C18B4180BA3DE65FB9B6EC8D7F7533C02A18779FA6BBF8E0F386E6DC00CC046DDE29E72839537E5106862DCBB57056AE8F820 7CC855C65C1686FD335963F23928068388A9BE75E3C69B26E64A689C8B00E505CFA80243FA36B26B773346974678B4A8D7C99BEBF75E061D840B1137FA1967C1656C3DBFDF04C82D65D01513A8 R5RFD93R8F76075F6A8R979F4D1061AR0A64FF808F39819FDFD476R0124A55C892C8F14FR5CA197F4079578FR0397R6992134A637045A8C9FR8238F02CF0FCA834F194FCC0814FF9495D16F70D 5B2B7AFEF9199B9CA2FEF2A60E8913A247088345A6983C7ACDF01358CB5C4B5BB093380A48D788D24C1DBA0DDCF207C697A9454F064A16E28513E2DB92B0D0FDA62F221B86BB92F061F791CFE6 0FC1EA7A46DB9C5FA3A8063A265D80A035B37EDC859D40813BF1E1068488DA4B925088821713A979776B722A0ED17604A92E2D053B1A798C5F4F9AD08038153C6DDAAE3A31772EEF22249F3004 4C8450DC7D19513DC48897CA46C21A09FA13E7EFF01CF2D36EE3DD4474A3C460B4FE000A74DA090C1A31A9F768E0EDCAD2C967E6F1648D69D3D493ED3A68C0B9C46450880DB77CDD9ED6A82645FD9982AD419B1BAD0725FA2E34936A7A05AD6B3E55D88D4DF41E50103BAB4F282620D66D6C06B7EB14465B5C56F76CCAF2E07B9B052A57CFED32AD203E71F85267C3F40122E3EBA65080D26BBD EEB72D1C001528F682E1B6D6ACC9544758C99B42DDBCF7D58926E2DAD3975942C4E48DCBB7057454D305FB7A8A7197C33152A0F0C3AE7DA62D2ECEA6C9ACA34048DCA071AA7254AE67F3091048 CCF9926E44D8C4A48D9C081A37D52FB7BFB8AF1C415A99CE4FEF4133CE5DC6AE310A370CFE377F4FB63E3617FBE58C03E74A8ECE66C84EADBE87ACD500039460FD2DD02765C9E41DBC89D8F94E 808F973A9CF33E6C439C1DF00FCA297F2E02233A89113CEE671B2D77B988BC82266836CD6329232546E805DBB780DE19DECAE05AB6A874E897B4A61EF5EC608B7B1D8FD6BCBCB7D44A0E7E62B7 27D2FRD397813DFF94F499CF9F474488448239893DD85F94DR3664C28935FFF95F68R36393F32DR82D16D6F9424DC66R939541FR2F1DR4C5F6147R99RDFRF78818FF4RF1RFRF4D8R6859618DD8 AE9083C0C7C34E3D59F999538E69E1CA1A0B482404909F9732344E141658C82AC4535D3E553C7798216F6A415B902B420D50B3CA66F2AAD495811D9442936D0922DE67C18AA571AE2D70A68338 0CF41724F1C8E4DC20937CB67D9A06E8CAEF2A62ADD8BE97339F288A1B386A9F1F631315155C289F5734C4D785353B33AE7F37F109BDD05B80373E6FDE8F78DB4E9B087E69E44E5801F2F64BA3 8926C8FBF88BFB4BE2C4370CA938FD3921F51BF7EA911186DC37AFF11D1819698F9203FC46AEA5BAF54B775642352E6535B2D7ECA81D21DB9FEA9A1C624297B5B762D68BB5D05F91CAE2C1D0FD E59246911C4DABE3E13B1E933A8CBFB57DD6EB8F84A03F0999E6750A1581547BB7A6E01970B4DC30665C165E4C2BF5D1C3E9F95571CB6EA77F7D1E47C3B6D80A28999F6F1D56F3F7CEF7F1FCC0 033EFECA476DAD30265804181A6426C49F0A4420C9D73B3124318D0BDB52D64B79A73ECBCCE996C5482D9C36430C2A2DD0E38F0C4B2242D280D3B735A159D5B1624F9901481235E2F0A7AB11B92BF498B71E1C605A3D4A71C950F5FD4463181FA1CECE91BF921B017E347EB47749FC41EE0E4387D4619CE6966D22E0B6E0F9F4156FF88EB0A0E85C4EC8703F3EB3731236C842A609E43F3F80A7 4CE9441634F5F51226E126D4B7FBC431A710C5D44F87DE66901EA7BECE03D718D9A52CD0DA3CA3F1F3C65937FCF6031A0690B5BA994F0408D54A1017B5202D5E700E3A23F970703A94E01EB67D 79DEDCER637D46D18R5RD021C65E3819D6C1C5CF4D8D47CD093153EC79E5242R2992634E86RE54D5D44C2E5E0434819RD2F4C44402EC13404D08R1479E053386DC4D18028ED36DC43244D038DR7 0621B5C644DA7256213EB2E8883D1B92D8B2324FA13725EC3738839DE4C6A7243A142EA129DBAFD5995E29A3FB3D1220236AF15AA85A5B47FDB8CE66EDCE53A21905DBBB0DF5E5D26B7474BF2B 4B47E67619B02ED5DC83B801892034ADD285714FE5F329C48C9538918FE3193E18F8E726C462D4C69A4102E90BC8CB699EC95555EA2AFDDE601136EA36BC6B0E00540438DE8D11F771BA01E6DB 501E4D2B5939FEF803C0305F8976C5DD1FF7EB77BF09A6779ACA0C31A18A431176E7178875B94F5D6FFF71AAF94B09F3A709F00A3E6B9E45A3CC1F217E9CA488F85985F5182C2700BEB65FFB5F 4AFB7E123BC73A280E9AF18E405FFB4389667E87D4695B2E068FA8AD979108A59518D022913F8AE2B22EB3470A132FDF006B5F35BA23E6B9106517172090F22397FBCF2736AAD75DD2E5B0578D 7C344C175B516325654DF4F54D49F826040BD3A692A352D41CF191689A54CCB3C941779BEFBE38AE04ADF395D3EB59A62074887F1BDFC705D15148A10735C2C2EB5084B22551DEB3593BC5739BF48E66EE985D1915FAB316B612CDBAC862C24F9E644B6DEF0AC266D24D63C82AFD798AB6C644BF45740B7E5A1CFBAD946A1B5ACF961CE2ECB44A31DB0A30D54A1492C348AD939890C280E2E4D6 C4DE46FBAC1D6CA2B31421E60A36F14E891A0397EA353CA765A9F947BF0CE8BAA4EFE757BDF85AAD28FE061504C02E2C9ECC7D2CD1BF2ADB2DD6F687477053038182B19312ECF85E907F527AE7 131F1F09DR634F9F594307FC4CC91414D1D96R4473F62550952958R3DR377RF6D9C34R47AC4D17CDCF14493D24011R94FFC2D232D1R41562282DERF249424C34RR9837F4RDRD091324R781D181 0730995E28E8E5C2058B01E00B9726A920FDFBBE331E81E6BCAD5DE42E314407F0579A1D2E01870B2FF195EF036DD839BFCC05618E935E40EC0AEBE137B164148BF48C673C46DA112A333CE3A1 53D957CA1DD0082E832B22E5AA4E92CD6C901E8C25BF8D32A7A953CE7493EA0003D8B2686CBF7DBDE866BC48BAD103DE86B22034CA17E2FA812ED6D79C7B40808D928A8F106C8CCD7B8D410359 127139F149720E737D12C7C270E7A645519E6DE11912F55F523699C5417319E4B519AC93121F49B617F6CBD8EFC3A092A7A5916CB1AB66F1B31EBC3EE09D42DBB45B687CF6B55C2F49F19A11431D537581D9FFFF240D1DF68EBB27DAA1A54BDA148EABC0FFFE401C10D63F665F3F44DD079AFBEA69A6A1F152A4745EF4C2B49E184D6D9B2F38097B4B971DCA89461BBA5690B3618DA28F77BC6A DB679A9E8AB03A4840B614A0F683BAA4DFD242E98A91F87612218D16149DE4AA5406C531C4F3510750EFD9EB351558AE894495FF32E82D942C5762BD74AFCEFA489B80CC8E2D043C077A1F12E4 57A75B1AD704091CA910316851E3855EB08D580BF8D7DD15CAF656680630CD701786EC8BEFB5A33422F7CC4D78619D68542874D9916B962E3D7317AF118373480399D3508DA185394EA072C302 40C2FA219F15A87ABF7E4046AAE90210C10AC9492E5AEA7A26706C1356ADF8FDB6D5734C30E90CD632092E6B841EF26B87782EE2198DDBBBCEA0F217167D7991E67693D3D5D3EAAB1B51635847 221A35A5FAF99ADA2A86FAR655AC507F58851835CA311R29352FF09C66568F7CA260R0897CR0230F854RR5D00RA6AAF78A50FA0748811AF563754948371F0F574C0RFF1FF60R9859A612R0F3CF B462963B91D0C17B790F60662F868A82D7947C6D52279980C9738C489259852B3D787FD001929D969B464F3AA570EE41708A7FB9AE3599AAA19EF163038B5CDAE98CAA5BB79C3EC66EECB8D74 332EE683B50BA64D2C7C04E2F23056F4D7CE02DF26417B63745BF38374193B077CFC3A817B2957E162EDB895C4311CA5B49B84BAA00BA4484BEBD43E86BD3A9B9D98B89A69C106CE2DCDA74ADD 6C98AC39D09F077C6A4667A4AB4887CF6BB3C95D590A4C92BD89A883A44D28E40BB89F9855C4FAD6FD15D6947279AB7B7DABCC3A5A35B7274EE6A4E12EE7979AAC2B2C01C4D9FA36C089A2CEA6 1DD1B71F611E3E0D9501BB820239453FD66C1D99F4E12EE69CCF080E28AB311DEBDEAE869A4CBA7C31D7D846F7A103FEF6A358C571E3632B01482D75038C74BFD04DA323C60D7603F69420DDD7 E6491E3ADBB8B0D94907CE982CA288C02F7962033903EC0016AA81EBD7385F96FC890AE2C1A7EF4E4CE6E341A3E3E2567D1CBFE08378B4D0FF895D1757D266CC9BC0127DED3991ACD706A857CC 853722A262AD3C9407E21BB0E3F97CFD588D73E73BCBA83315177E9AC4BECA7B7D72ED2A5F7F7D3682A75C8AA51ACEA9F9A15E7C69B6B9E111379BFDE3B1B88EB24F0FF6563D05F686CAD9589D D231BC41489BF0169C7B46EE741EB8888BB870318F03FCA759304786B341EC267E82D59DF0D39D8638594F8D1DC355CB37862237FB7520949F119AE5C6FB2A0B535EFBBE91525DB19D06618B75 341C28439064F42D208649DR6C08F627R40F524FD75F8F37273647509C5393571F1R4543DC5DFF5C4F7749R54399CR2FC444D1476FF8274FDF95D12419184055354DCDF8449RCD401CF2044C108 6F9D7B6AAF1E3512A7B29EE3C129DECE22F3B403CA665FA5804028956551AD167958EFD1E4A89F4E7D85C7BD7600F9FB1958F66BA93FB68A2D2D7A506D65805D42FC893536D492584486F5FC4A DB63B9395DD0D0E36E07E7368D1FD0CF7B675E503683CDE26B7287AE3C7C9135FD9EC8B5E510AA0DCBE362744AA9D54C4B1D8D46EBC01103DA3DB2050702252EC9441D7CE8F224CF1219782F9E CF7E8D2EF96242B4353B3A91AD62125039554BA7F997AD0361175A0FA4BAD6F0CBCC4EFD809AB372BA53AB5405131725F133984B2A25C60AF0212BD13D69ECCA50B43AE441BC7045D8252C886E A36BE444DBB48A77A3C48D1DF44E1995D7AD73966B7A92B629C9BB7DF87D680C416D98DF5EDB06C0D1B7ED0324590A36BFD8FF5E2C715FFD0D49A3C0CA2C41D0215547956E852B59F62F577090 0000000000000000048100000000676C7970686167652E706E675554050003505EF55775780B000104F50100000414000000504B050600000001000100520000000EC1100000000--496E2074686F736520646179732C20696F2074686F7365206661722072656D6F7465

207265606F74652079656172732C20696E2074686F73652066617261776179206E69676874732C20696E2074686F73652066E172732C20696E2074686F73652066E172
207265606F74652079656172732C20617420746861742074696055207468652077697365206F6E652077686F20686E7720746F20737065616820696E207566C616226F726174652077
6F726473206C6976656420696E20746865204C616E64382043757275707061672C20746865207697365206F6E652C2077686F20686E657720686F7720646F2073065616B207769746820076
6F226473206C6976656420696E20746865204C616E642E204375727570706167206761766520696E737472756374696F6E7320746F2068697320736F6E2086697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E308697320736F6E30869737472756374696F6E7320966520736F6E308697320736F6E30869737472756374696F6E7320966520736F6E3086973073757261320906520737065616820073686F7508666794696F6E2120846F20966F6E2120846F20796F752073686F7532073686F756C642070617920617474656E74696F6E2120846F20956F6E73472756374696F6E7320946F0E73472756374696F6E7320946F0E73472756374696F6E7320946F0E73472756374696F6E7320946F0E7320946F6E7320946

















+++ATH0