In this homework, you will experiment with Simulated Dialogues, in which a computer system has a conversation with a simulated user.

Simulated dialogues are useful for a number of reasons. First, they allow researchers to experiment with what a suitable information state representation should be to allow a computer system to carry on interesting and natural conversations (Sidner 1994). Second, they allow researchers to investigate why certain language phenomena might exist, by studying the dialogue cost when a system and simulated user use the feature and do not use the feature (Walker 1995, Yang and Heeman 2004). Third, they allow researchers to apply reinforcement learning to automatically learn a dialogue policy for the system. This is our main purpose for simulating dialogues.

In this homework, we will have a computer system, specified using IS, and a simulated user, also specified using IS, talk to each other. Below is some research work that has made use of computer simulation of dialogues.


In this homework, you will be starting with IS3-Agent.tcl and IS3-Engine.tcl. IS3-Engine.tcl is very similar to IS2-Engine.tcl. IS3-Agent.tcl is a simplified version of the system agent for the banking domain.

**Question 1: Specification for System and User**

In IS3-Agent.tcl, you have update rules for the system for the banking domain. The rules are a simplified version of the ones used in the previous two homeworks. In particular, no agenda is used, nor are there any deliberate rules.

Explain how the new rules allow the system to ask the user questions without an agenda or deliberate rules being used.

You also have update rules for the simulated user. The behavior of the simulated user is very straight-forward, as it will just answer the system’s questions. In fact, due to how simplistic the user is, the information state of the user does not include an agenda nor a QUD, as the user just uses LastMove to determine how to respond. Thus, an understanding rule is not even needed for the user.

Note that for more complex domains, we might need to specify the behavior of the simulated user with a number of rules, and we might even use some of the same rules for the simulated user and the system, such as for updating the QUD. However, for the banking application, we can keep things as simple as possible, and just use one rule.

What IS variables are being used to describe B’s state? What code determines what B’s task is (how much does it want to transfer between which accounts).

**Question 2: Change the IS Engine to Support a User Simulation**

Change the code in ISEngine3.tcl so that it will run a dialogue between the system and the simulated user. You will need to make a small change in the Run procedure, and more extensive changes to RunSide.
The procedure RunSide is not well-named. What it does is run one cycle of the control strategy, running the appropriately understanding rules to understand what was just said, running any deliberation rules, and then running an action rule if the system has the turn.

You should change this code so that RunSide still does one cycle of the control strategy but for both the system (A) and the user (B). For RunSide, don’t bother passing it any parameters, you can just assume that the two agents are A and B. The first thing you want to do is figure out is who is the Speaker and who is the Hearer. Have the variable S indicate who the speaker is and H who the hearer is. Also, have isS be the information state of the speaker and isH be the information state of the hearer, which you can do with the following code:

\[
\text{upvar ::is$S isS} \\
\text{upvar ::is$H isH}
\]

In the remainder of the RunSide code, do not directly refer to A and B, but only through the variables S, H, isS, and isH. Also, the code should work for any user and system specification. In other words, even though this user does not have any understanding or deliberation rules, your code should still check for them.

As the user is being simulated, you can remove all of the GUI code, and just use the Tcl put command to output to the console.

After you make this change, you should be able to watch the system and simulated user have a dialogue.

Writeup 2.3. Marks 3

Hand in a copy of the code for Run and RunSide and highlight what you changed.

Question 3: Make a Simple System for Car Buying Domain

For this question and the next, we return to the car-buying domain of the previous homework. In the next question, you will be make a simulated user that will interact with a system in order to create simulated dialogues.

In this question, you will simplify how the system works, so that the user simulations does not need to be overly complicated. You can either start with your own formulation of the system behavior or use the solution that was given in the answers for the previous homework.

Make sure your system behaves as follows:

- Assume that the user always has a goal of a car that is actually in the car database. Hence, you should remove support for the system to be able to re-ask the same question, as there will always be at least one car that matches the user’s answers.

- Assume that the user cares about all of the parameter values, and so you should remove support for ‘don’t care’.

- Assume that the user always makes valid responses. So, you do not need to check if the user’s response is valid for the question that was asked.

- After the system makes a summary, it should only be able to use the bye action. Furthermore, this should be the only way that the system uses the bye action.

- You should have a deliberate rule for the system that sets the IS variables Cars and NumCars computes which cars match, and the number of cars that match.

- The system should just summarize when there is just a single car matches.

- Order the attributes so that they are asked in the following order: color, doors, powerwindows, power-brakes, powersteering, transmission, cylinders, type, airbags, year, and mileage.
As the changes to the system are very straight-forward, there is no reason to hand in your specification.

**Question 4: Make a User Simulation for Car Buying Domain**

In this question, you will make a user simulation.

First, you need to give the user simulation a goal: a particular car that it wants. As mentioned in the previous question, we will assume that the user always has one of the cars in mind that is in the actual car database. Use an IS variable to hold the user’s goal.

Later in this homework, we will be even running multiple dialogue simulations in a row, and we want to make sure that each time the user has a different goal. The best way to do this is with the procedure `InitAgent`. This procedure is called by `Run` after all IS variables are given the initial values specified in `SetupISVar`. For setting the user’s goal, randomly pick one of the cars in the database.

Second, you need to specify an action rule to answer the system’s question. Do not worry about using a separate understanding and action selection rule. Simply do all of the processing in a single action selection rule, as we did in Question 1.

**Writeup 4.4. Marks 2**

Hand in a copy of your code that defines your user simulation. You should include any `SetupISVar`, `InitAgent`, and the update rule for the user. Where possible, use the same IS variables for both the user and the system.

**Question 5: Costing a Dialogue**

Add support so that you give a dialogue cost to a simulated dialogue. The cost of a dialogue can be viewed as the sum of the costs of the system’s actions that have been applied. We will add an IS variable to the system called `Cost` that will be used to the system’s action rules to indicate the cost of performing the action. The IS engine will use this variable to compute the total cost of the dialogue. Below are more detailed instructions.

**IS Engine:** Change `Run` so that it initializes a global variable which will keep a tally of the action costs. Change `RunSide` so that after any action rule, it adds the cost of that was specified in the IS variable to the tally variable. Note that `ISEngine` should not know anything about how the costs should be set for the domain. That knowledge should just be in the agent file.

**Agent Specification:** Each utterance by the system (including the final `bye` action) should have a cost of 1.

For solution quality, the cost should be based on the final set of cars displayed to the user. The action selection preconditions should enforce that a summary action has occurred, and so at least one car will have been displayed to the user.

Due to how we have set up this problem, if any cars are displayed to the user, we are guaranteed that one of them will match (as we have the user pick its goal to be one of the cars in the database). But, you should charge 5 points for each extra car. So if 5 cars are displayed, one of them will be the solution. So 1 will be correct, and 4 of them will be wrong, giving a solution quality of 20. If only 1 car is displayed, it will be correct, and so the solution quality cost is 0. Note that none of the cars in `hw5cars.tcl` are the same, so you just need to see if the user has the same car in mind, rather than having to compare their slot value attributes.

In the hand-crafted system update rules, the `summary` action is only applied once immediately followed by the `bye` action. Although we could include the solution quality cost on the `summary` action, include it in the cost of the `bye` action, which is more representative of what is usually done in the literature. Note
that by having the solutional quality cost in the bye action, if we removed the preconditions that force a summary to occur, our code would still work correctly.

So, wait until the bye action (when you know the system is really finishing) to charge the solution quality cost. The summary action should save in an IS variable what cars it outputted to the user. Then, in the bye action, you should compare these cars with the user’s intended car, which you should have also saved in an IS variable. This logic should be in a subroutine that you call from the effects in the bye action. So the bye action will look like this:

```
SetupRule {A} action bye
AddEff {set is(NextMove) bye}
AddEff {set is(Cost) [finalcost]}
```

In the procedure `finalcost`, you can examine the value of A’s and B’s information state directly, using `::isA` and `::isB`. Note that `finalcost` should also take into account the dialogue cost of the bye action, in addition to the solution quality cost.

**Writeup 5.5**  Hand in a copy of RunSide with the changes highlighted. Also had in a copy of your agent rules, highlighting what you changed.

**Question 6: Running Multiple Dialogues**

Add code so that `Run` is called 50 times. Have the code keep track of the cost of each dialogue run. Also, have the code compute a histogram of the dialogue costs (number of dialogue runs for each different cost in ascending order by cost). Tcl’s associative arrays are very handy for this. You can use the command `array names` to determine all of the indices that you used, and use `lsort` to sort it in numerical ascending order.

Also, have the code compute the overall average. Report the average to two decimal places (which you can do using `[format %.2f $ave]`).

Note that if you are getting a decimal point of `.00`, you are probably doing your computation in integer mode. For example, if you compute `expr 7 / 2`, tcl does this in integer mode and returns you a value of 3. If you want the answer in floating point, you need to make sure that one of the arguments is a float. So, before dividing, you might want to multiply the numerator by 1.0 first. Many programming languages have a similar issue.

**Writeup 6.6**  Give the code that you added, the histogram of costs, and the average cost. Below is mine. Your average should be similar, within say 0.5 of mine.

```
Average cost is 11.52
7 1
9 3
10 6
11 4
12 30
13 6
```

**Question 7: Speeding up your code**

In the last question, you ran 50 dialogue runs. In the next homework, you might be running a 1,000,000 dialogue simulations or more! So, lets spend some time speeding up the code.
First, make sure all of the puts statements are inside of conditional statements that check if some debug flag is turned on. Do the same for any commands that update the tcl/tk window in Run and RunSide. When you want to run lots of dialogue runs, just turn off all of the puts statements.

Second, we need to speed up how the system determines which cars match. This is actually one of the main reasons why the code is so slow, as the system needs to go through every car in the database for each system action.

- In your initialization code, determine the full set of cars, and save it in a global variable, such as ::Cars. Then, in your declaration of Cars as an IS variable, you can specify that $::Cars is the initial value. Also, specify what the initial value for the IS variable NumCars should be.

- Just have the deliberation rule run if the user has given a new value. So, add an IS variable for the system, say NewValue, which is set to true in the system’s understanding rule for user answers. The deliberation rule should just be called if it is true, and should reset it in its effects.

- Speed up findcar. As we have removed the re-ask ability of the system, we actually know that as the dialogue processes, the matching cars is just a subset of the previous set of matching cars. So, we should just need to go over that list, rather than all of the cars.

With my laptop, I can process 1000 dialogues in 12 seconds. You should have comparable results (at most twice as slow).

Writeup 7.7 Hand in your new deliberation rule, the understanding rule, the code for findcars, and any other code that you changed.

Redo your previous question, but for 1000 dialogues. My version took 13 seconds to run.

Writeup 7.8 What is your new average? Is your average based on 1000 dialogues the same as for 50? If not, which do you think is closer to the real average?

Question 8: Variations

So far, you just have the system issuing summaries when there is just a single car left. Change the system specification so that it issues a summary when there are 5 or less cars available.

Writeup 8.9 What is the average dialogue cost over 1000 dialogues. Why did this change the average dialogue cost?

We are also forcing the attributes to be asked in a certain order. Change the order of the order that the attributes are asked so that year is asked first, mileage is asked second, and then the rest of the attributes in the same order as before. Also, the system should just issue a summary if there is 1 (or fewer) cars that match.

Writeup 8.10 What is the average dialogue cost now over 1000 dialogues? Why did just changing the order that the questions are asked change the change the average dialogue cost?

As summaries are only being done when there is one item, no penalty is being paid for solution quality. So the change in dialogue cost must be solely due to the number of system utterances.

Writeup 8.11 How much shorter, in terms of the number of system utterances, are the dialogues now?