Bazaar Activity: Deploy an agent for your domain.

Deliverables are marked in **bold**. Turn in the final report to your instructors.

1. Deploy the agent as-is.

- Follow the setup instructions (below) to get Bazaar talking in a web chat client.
- **Celebrate this early triumph with a screenshot.**

2. Adapt the agent to suit your domain.

- Imagine a scenario or specific task where you might deploy a conversational agent to support your would-be course. **Describe the setting or task you have in mind.**
- Make at least three substantial changes to the given configuration to suit your domain. **Describe the changes you plan to make, with a brief justification for each.** Here's some ideas:
  - Replace the Accountable Talk exemplar statements (and associated synonym files) with statements relevant to your own domain. In a perfect world, you'd have collected these from earlier student data, but for the sake of exploration you can follow your gut. Aim for at least a dozen statements. You might also adapt the content of the agent's Accountable Talk responses, to suit your domain or student level.
  - Change the content or structure of the included macro-script or tutorial dialogues, or create new ones from scratch. In the case of triggerable dialogues, you should also edit the dictionaries that trigger them.
  - Remove any agent behaviors that you think are unsuitable for your domain. Edit *operations.properties* to turn off the Actors or Preprocessors that you don't want. *(You can also uncheck many Actors' behaviors in the Agent UI.)*
  - Adjust the properties of the included components - for example, you might lower the triggering threshold for some of the Accountable Talk moves, or extend the time that a dialogue waits before giving up.
- **Include any files you modify in your submission.**
- **Celebrate this hard-won victory with a screenshot of your in-domain agent in action.**

3. Make it better!

- What would your conversational agent do differently, or in addition, if you had more time, experience, or a full-time developer on your payroll?
- Propose at least one new supportive behavior you'd like to see for a conversational agent in your domain. It can be purely conversational, or it might involve things that the web chat client doesn't yet offer.
- **Describe the behavior, and explain why you'd consider it appropriate. Explain how it might be implemented** - what would the agent have to pay attention to in order to provide this support?
- **Illustrate this utopian future with an imagined interaction.**
  Include at least a few lines of pretend dialogue between your would-be agent and your users that demonstrates the behavior you have in mind.
- Send any comments, questions, feedback, or wishes to **gtomar@cs.cmu.edu**

Atlas of the Bazaar

Set up your environment

Bazaar Demo:
Download http://erebor.lti.cs.cmu.edu/bazaar/BazaarAgent.zip and unzip it somewhere nice.
Java: You'll need Java 6 or newer to run Bazaar.

A Decent Text Editor: You'll be editing several plain-text and XML files.
YOUR LIFE WILL BE BETTER IF YOU NEVER USE NOTEPAD EVER AGAIN.
Use an XML-aware editor, like Notepad++ for Windows, or SubEthaEdit or TextWrangler for Mac OS X.

Try it out

- Join your chat room by via OLI: ***bazaar.lti.cs.cmu.edu:80/chat/***<*your chat room name*>***/***<your account id> .The macro-script should start once you join. Please make a note to use <*your chat room name*> in the "Room Name" field of the agent.

- Double-click ***BazaarAgent.jar*** in the BazaarAgent folder. If it doesn't double-click, try "java -jar BazaarAgent.jar" from a command-line terminal.
  On newer Macs, you might need to right-click and choose "Open"
  to explicitly give this unsigned app permission to run.

- The "Conditions" checkboxes turn certain agent behaviors on and off:
  turn on "**tutorial_trigger**", "**social**", and "**revoice**" for now.

- Enter <*your chat room name*> in the "Room Name" field, and then press "Start Agent"!

- Interact with the example agent - its macro-script highlights several of the behaviors you'll be able to customize.

What's Inside?

BazaarAgent/ contents organized in order of the likelihood of your needing to do something with it.

File	What's Inside	What You Might Change
BazaarAgent.jar	This is the double-clickable agent launcher. If it doesn't double-click, try "java -jar BazaarAgent.jar" from a command-line terminal.	Chatroom name, active conditions. You can also wizard-of-oz tutor behavior, or pretend to be a student user.
properties/	Most of the "settings" are in here.	Behavior timing, priorities, and pointers to content files. "operation.properties" in particular controls several high-level agent settings worth investigating. Most properties files are thoroughly documented.
plans/	Macro-scripts and prompt files	Scripts! Sequencing and phrasing.
dialogues/	XML files defining each tutorial dialogue, and an "index" (dialogue-examples) that determines how the tutorials are triggered	Dialogue content, sequencing, and triggering. The example "animals" dialogue is heavily commented for your learning pleasure.
dictionaries/	Files (in nested folders) with word and regular expression lists that are used to "annotate" each student turn. The filename becomes the (ALL CAPS) annotation name.	Add or edit annotation dictionaries to support other behaviors (annotations are used to trigger tutorials, mitigate AT responses, etc)
accountable/	Accountable talk data and prompt files	Replace the contents of exemplar_statements and content_synonyms with something domain appropriate, rephrase the AT prompts
agent.xml	Architectural configuration - high-level agent pipeline stuff.	Agent name. You can also swap out the WebsocketChatClient for the DummyClient (and operate only through the agent wizard), if you don't want to deal with Moodle.
logs/	Annotated chat transcripts and also less useful logs.	take a look, if you like!
dict/	WordNet data files - used by the Accountable Talk behaviors.	nothing!
planstatus/	Keeps track of a macro-script in progress.	You might delete the contents of this folder to avoid "a plan is already in progress" messages.
behaviors/	Leftover behavior logging. Research code artefact.	nothing!
log.properties	Logger configuration. Research code artefact.	nothing!

Accountable Talk Behaviors

The four included Accountable Talk facilitation behaviors are Revoice, Agree-Disagree, Say More, and Ask for Explanation. All four work on the same principle - identify a candidate student statement by making a fuzzy match against a list of "exemplar" sentences, wait to see if students follow up on their own, and follow up with a facilitation move if they don't. Exactly what counts as a candidate or a followup statement is controlled by the properties files for each behavior.
See http://www.cs.cmu.edu/~dadamson/pubs/IJAIED2013_Adamson_Agents.pdf for much more about these behaviors.

File	**Description**
accountable/accountable_prompts.xml	Response variations for each facilitation behavior.
accountable/exemplar_statements.txt	List of sentences that would count as candidates. It's possible to have a different list for each behavior.
accountable/content_synonyms.txt	Sets of domain-specific content words and synonyms (upweighted in statement matching). One set per line.
accountable/synonyms.txt	Supplemental sets of domain-general synonyms. One synonym set per line.
accountable/stopwords.txt	List of "common" English words (downweighted in statement matching)
properties/AgreeDisagreeActor.properties	Configure the Agree-Disagree behavior. Nicely documented!
properties/AskForExplanationActor.properties	Configure the Explain-to-Others behavior. Nicely documented!
properties/RevoiceActor.properties	Configure the Revoicing behavior. Nicely documented!
properties/SayMoreActor.properties	Configure the Say More behavior. Nicely documented!

Social Support

This behavior implements social support strategies for group cohesion, using some handcrafted rules to notice and respond to various individual and group behaviors. There are two conditions mapping to this module, **social** for a broad set of supportive responses to student turns, and **participation** for nudges when the system notices an individual or group of students isn't actively participating. See http://www.rohitkumar.net/papers/published-rk-its10.pdf for more.

File	**Description**
plans/social_prompts.xml	Response variations for the various social supports.
properties/RuleBasedTriggerComputer.properties	Specifies the "social threshold" for how frequently the system should act on the social cues it notices.
properties/SocialController.properties	Defines the priority and timing of social moves.

Macro-Script

The PlanExecutor delivers a more-or-less timed sequence of steps, starting when a Launch Event is received - either when enough students have joined the room, or when a timeout has expired. You can also start the script manually from the wizard UI. Each step "type" is controlled by a different Step Handler.

Every step can have a "delay" or "timeout" attribute. "Delay" is how long to wait after the step completes on its own before beginning the next step. "Timeout" is how long to wait before cutting off the step and beginning the next step. See *plans/plan_steps.xml* for a documented example.

Step Type	Description	Attributes
prompt	Deliver a prompt. There's a built-in delay related to the length of the prompt, to allow time for reading. Additional delay may be specified with the "delay" attribute.	prompt="*NAME OF PROMPT*" (from the PromptHandler's prompt_file)
greet	Take time for introductions. If students give themselves a name, the agent can use that name throughout the session.	timeout=*seconds* (this is the maximum duration of the greet step - also redundantly specified in the IntroductionsHandler.properties file)
dialogue	Launch a dialogue, even if the tutorial-triggering condition is not set.	dialogue="NAME_OF_DIALOGUE" (from dialogues/dialogues-example.xml)
listen	Temporarily activate the full suite of Accountable Talk and Social behaviors (the corresponding condition checkboxes must be checked). Useful if you don't want AT facilitation all the time - remove the AT actors from operation.properties if this is the case.	timeout=*seconds* (this is the duration of the listening step)

File	Description
plans/plan_prompts.xml	Text (and variations) for each named prompt.
plans/plan_steps.xml	List of sentences that would count as candidates. It's possible to have a different list for each behavior.
properties/PlanExecutor.properties	Sets of domain-specific content words and synonyms (upweighted in statement matching). One set per line.
properties/PresenceWatcher.properties	Supplemental sets of domain-general synonyms. One synonym set per line.
properties/PromptStepHandler.properties	Specifies the prompt file from which prompt-step prompts are drawn, and also the words-per-minute delay after prompt steps.
properties/IntroductionsHandler.properties	Specifies the prompt file for introductions, and also the maximum duration of a greet step.

Tutorial Dialogues

TuTalk is an early PSLC project, specifying a rich and feature-full hierarchical dialogue system. Bazaar implements a limited variant of the TuTalk specification. While originally designed for two-party dialogue, we've used such scripts successfully for collaborative learning in previous studies. See http://www.pitt.edu/~tutalk/overview.pdf.

The three big ideas in a TuTalk script are **Concepts**, **Goals**, and **Steps**. A **Concept** represents something that a student might say, or that the dialogue system might say in response. For tutor turns, these are usually lists of phrases, from which the system will choose randomly. For student turns, a concept can be either a list of literal phrases (matched within a student turn), or of regular expression patterns, or a list of **annotations** provided by the Bazaar Message Annotator (via the wordlists and patterns in the dictionaries/ folder).

A dialogue **Goal** is a set of **steps** completed in sequence. Each **step** is composed of an **initiation** concept, which is something the student might say, and an optional set of **response** options. Each response option specifies a student concept between the <response> tags. The first concept to be matched will be activated, triggering a tutor response (a concept specified by the "say" attribute) and/or pushing a new **goal** into the dialogue system (with the "push" attribute, putting the current goal on hold until the sub-goal is completed). Goals should *not* recursively refer to themselves. Here's an example from *dialogues/scenario-animal.xml*:

```
<step>
        <!-- Every step begins with an initiating concept or literal phrase -->
        <initiation>animal_question</initiation>

        <!-- These are the "response" options. The response's say/push is triggered
           for the first student concept (between the tags) that matches. -—>
        <response say="recognize_fish">fish</response>
        <response say="recognize_bird">bird</response>
        <response say="recognize_mammal">mammal</response>

        <!--   unanticipated-response is anything that doesn't match one of the above. -->
        <response push="remediate_goal" say= "unrecognized">unanticipated-response</response>
</step>
```

Launching Dialogues

All the available dialogues must be listed in *dialogues/example-dialogues.xml*. A dialogue can be started either automatically, in response to a student statement, or as part of a macro-script (see previous section). A dialogue is started automatically when the **tutorial_trigger** condition is active and a student statement is annotated (by Message Annotator, from the lists in *dictionaries/*) with one of the **trigger** annotations listed for that dialogue. A dialogue may be accepted (typically with an AFFIRMATIVE statement) or canceled (with a NEGATIVE statement). If nobody responds, the dialogue is canceled after a while anyways.

File	Description
properties/TutorActor.properties	Controls the wait-for-response timing and some prompt options shared between dialogues, and points at the dialogue index file.
properties/TutorialTriggerWatcher.properties	Not much going on - just references the dialogue index file.
dialogues/dialogues-example.xml	The dialogue index file, specifying the trigger conditions for each dialogue.
dialogues/scenario-animals.xml	A moderately complex and thoroughly commented example.
dialogues/scenario-hugs.xml	A simple supportive dialogue example.
dialogues/TuTalkScenario.dtd	A document type description that your XML editor might use to "validate" a TuTalk script, or provide syntax hints.