Group Member: R04922029 薛祐婷、R05922062 楊宗翰、R05944004 巫承威
R05922136 黃博政、R05922131 祝子軒 (demo link here, github-repo here)

Intelligent Course Bot

NTUCB - Milestone 2

✤ User Simulation

We implemented our user simulator based on the template of Li, Xiujun, et al. [1]. According to the "Recipes" part in [1], we referred to the files `usersim.py` (which is the basic user simulator class and provides a few common interfaces for general users to implement their customized user simulators) and `usersim_rule.py` (which implements a rule-based user simulator), and then modified them to fit our system.

Briefly speaking, we retained the function `initialize_episode` in the `usersim.py` and modified some contents in it. In `usersim_rule.py`, we changed some parameters such like the numbers of turns added after each turn, renamed the names of the slots by our own naming rules, replaced the sets of the action and slot with our pre-defined sets and altered the goal of the user (which could be treated as the condition of the ending of the dialogue). In addition, we also modified the file `kb_helper.py` to match the knowledge base of our topic (NTU-Course) and also put the Django-based query function in it for querying our knowledge base. The last but not the least, we constructed a online demo web page of our user simulator in `demo.py`, so it's easy to manipulate our user simulator (link here).

1. Definition of Goals

- Information Access
  - Query (for course information)

2. Error Model Construction

Since we adopted NLU to be our communication bridge between the user simulator and dialogue state tracker, the characteristics of NLU will automatically generate the error in this communication flow. Therefore, we combined the error model with natural language understanding in our system.

- Natural Language Understanding
  - Errors come from misunderstanding

3. Support User Actions

- Request
- Inform
- Thanks
- Deny
- Closing

4. Template-based NLG (**N**atural **L**anguage **G**eneration)

- Rule-based Templates (**30** combinations ↑)
  - Django-based Template
    - Request_template
      → `["title", "instructor", "schedule_str", "classroom"]`

- Inform_template

  → `["title", "instructor", "schedule_str", "classroom"]`

- End_template

  → `["thanks"]`

- SSIF Design (**S**calability x **S**implicity x **I**ntuition x **F**lexibility)
  - Extend the template via appending a "Template" element to each attribute's list:

    → `'title': [Template('請列出課程名稱'), Template('什麼課')] += [Template('課名是')]`

5. Reward Model Construction

We built a simple reward model which gives a positive reward (**Success**) if the whole dialogue ends successfully (users receive the correct information they really want); otherwise, it will give a **negative feedback** (**Fail**). Additionally, in order to reduce the total numbers of turns of the dialogue, the system will get some **few penalties** for **extra turns**.

- Success/Fail
  - Success → +100
  - Fail → -100
- #Turn
  - Each turn → -1

✤ Basic DM (Dialogue Manager)

1. DST (**D**ialogue **S**tate **T**racker)

- Rule-based Combination of Single-Turn Language Understanding

This DST employs the single-turn LU implemented in Milestone 1 turn by turn. The slots specified by user are recorded across turns. However, information from previous turns is not considered when understanding user's utterance in current turn.

- Multi-Turn Language Understanding
  - **state** (**8-dim**)
    - We set eight dimensions as states which contains four aspects (**title**, **instructor**, **when**, **classroom**). Each aspects contains two possible states ("**request**", "**inform**")
  - **history**
    - To keep the information of history, we store the total dialogue history and the last state. And each turn contains ten word vectors. If the #turns is less than ten or the sentence is less than ten words, we use **zero padding** to maintain the same length of history.
  - **model**
    - We referred to Yun-Nung Chen, et al. [2] to create a multi-turn language understanding system which can output a BIO tag and the intent of the current sentence. After comparing the intent of current sentence and last state, we could update the state by adding new "**inform**" to state.

2. Dialogue Policy

We implemented a simple rule-based dialogue policy. The system action is determined by the number of valid courses under the given constraints, that is, the slot-value pairs the user has specified.

- System Actions (`diaact`)

- `inform`
  - Tell user the value of the specified slot(s).
- `request`
  - Ask user the value of some slot.
- `close`
  - The given constraints are not satisfiable.

- Request Slot Selection

(1) Slot-level

For each slot, we compute the number of different values in the current set of courses. And then select the slot with **maximum** #values.

(2) Conflict #values

If there are more than one slot with the same #values, we select by the following order:

`title > instructor > schedule_str`

The decision of this ordering is made according to the naturalness of system requests. For example, "請問是哪位老師開的?" (Which `instructor`?) is more natural than "請問是哪個時間上課的?"(Which `classroom` ?).

The slots the user is requesting will not be requested by the system.

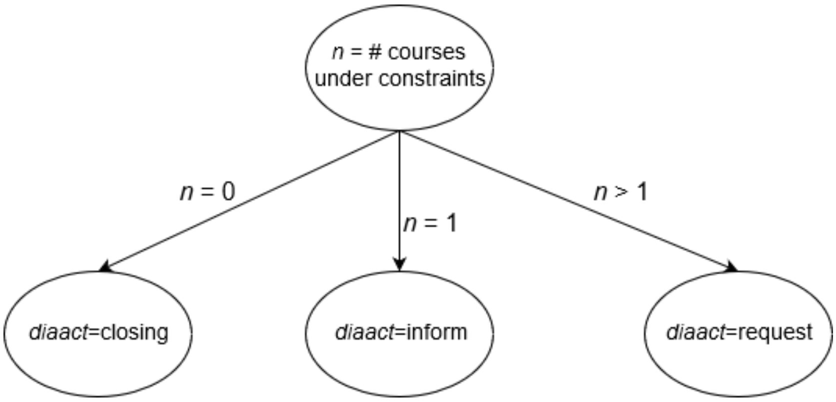

Figure 1. Dialogue Policy Structure

3. Dialogue Ending

When a unique course that satisfies all the constraints is found, our system will inform all the user-requested slots, along with the `serial_no` of the course.

4. Performance (full examples here, single-turn now)

In this part, we will use two tables (on the next page) to show part of the interactions between our user simulator and system. We finally generated 100 test cases and calculate the statistics as follows:

- Average Reward
  - **47.240000**
- Accuracy
  - **64% (64/100)**

We have built both single-turn and multi-turn systems. After comparing these two different structures' results, we found out that the single-turn system "looks" more robust than the multi-turn system. We think that the reason behind this issue is that the multi-turn system need more training data.

Case 1	
User	: 是量子力學三
System	: 請問是哪位老師開的?
User	: 老師是哪位
System	: 請問是哪位老師開的?
User	: 老師是哪位
System	: 請問是哪位老師開的?
Reward: -4	

Table 1. Case-1 Interaction between User Simulator and the System

Case 2	
User	: 在哪裡上課
System	: 請問要找哪門課?
User	: 國文下
System	: 請問是哪位老師開的?
User	: 是巫雪如老師
System	: 請問是哪個時間上課的?
User	: 星期一
System	: 流水號01059。課名是國文下。在共310上課。
Reward: 96	

Table 2. Case-2 Interaction between User Simulator and the System

✤ System Overview

The following flow diagram shows our system's structure:

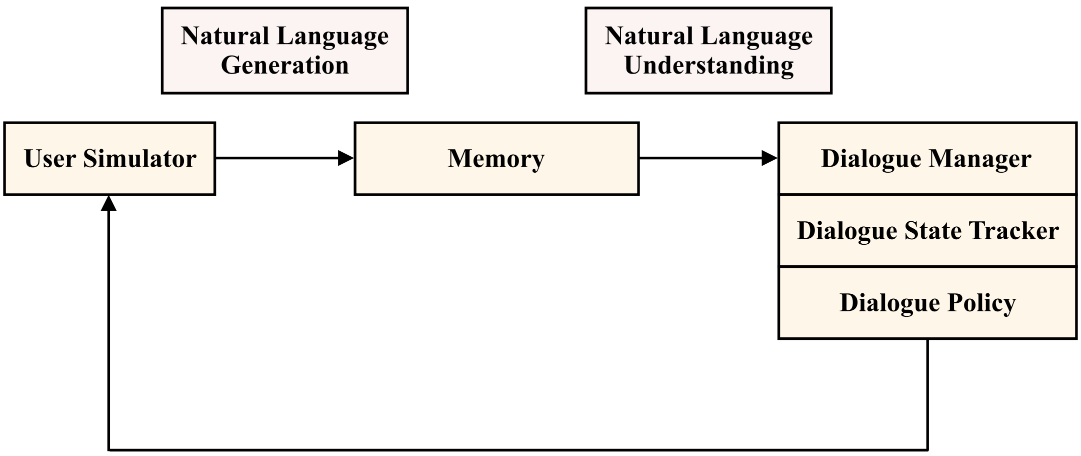

Figure 2. Flow Diagram of System Structure

✤ References

[1] Li, Xiujun, et al. "*A User Simulator for Task-Completion Dialogues.*" arXiv preprint arXiv:1612.05688 (2016). (link here, github-repo here)

[2] Chen, Yun-Nung, et al. "*End-to-end memory networks with knowledge carryover for multi-turn spoken language understanding.*" Proceedings of Interspeech. 2016. (link here)

[3] Bordes, Antoine, and Jason Weston. "*Learning end-to-end goal-oriented dialog.*" Proceedings of The 5th

International Conference on Learning Representations. 2017. (link here)

[4] Serban, Iulian Vlad, et al. "*Generative Deep Neural Networks for Dialogue: A Short Review.*" arXiv pr print arXiv:1611.06216 (2016). (link here)

[5] Shah, Pararth, Dilek Hakkani-Tür, and Larry Heck. "*Interactive reinforcement learning for task-oriented dialogue management.*" NIPS 2016 Deep Learning for Action and Interaction Workshop. 2016. (link here)

[6] Li, Xuijun, et al. "*End-to-end task-completion neural dialogue systems.*" arXiv preprint arXiv: 1703.01008 (2017). (link here, github-repo here)