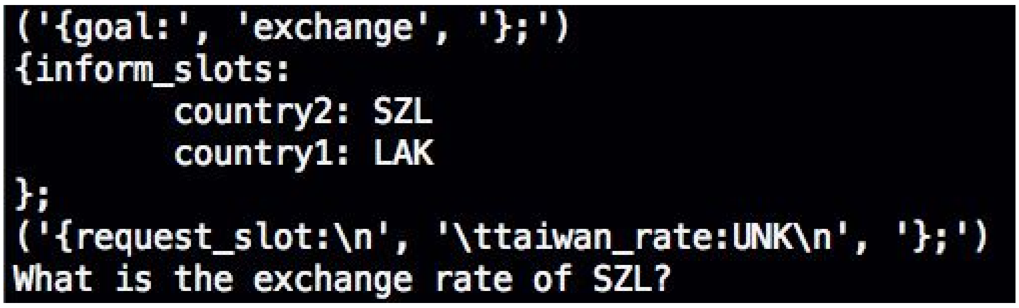
Milestone 2 report

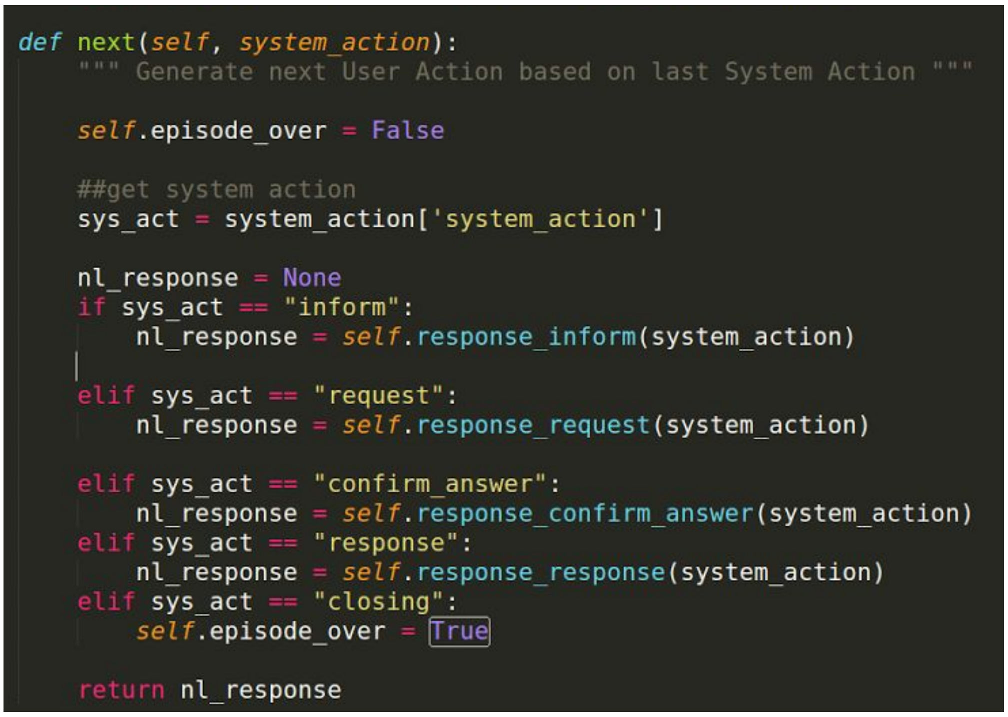
1. User Simulation:

a. We implement a rule-based user simulator Class based on the system action. There are two function, first of two is the initializer function, which will generate the first sentence of user base of a random user goal and let system know the user intent.
#an example of a user_goal and a templated_generated sentence.



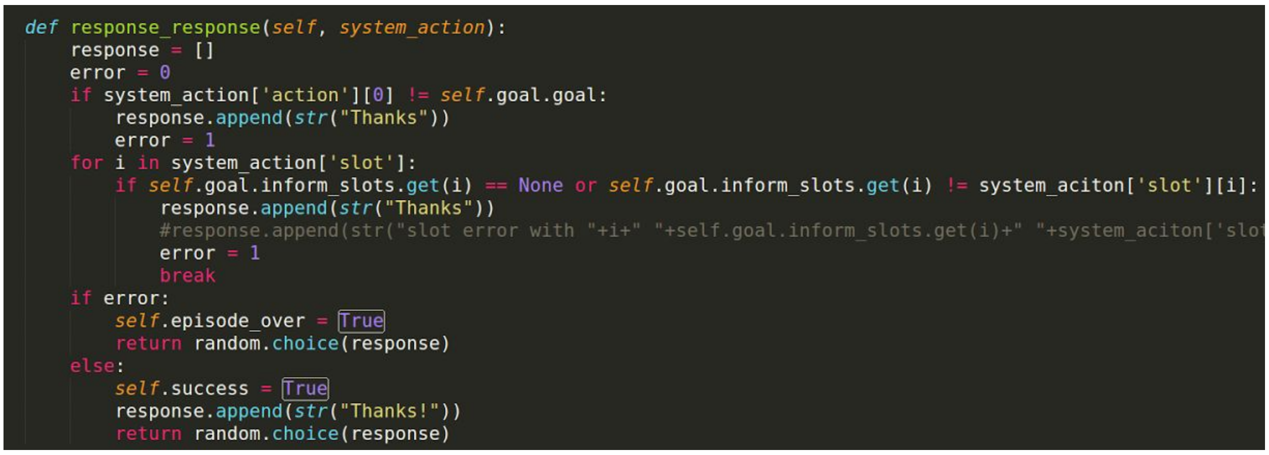
```
('{goal:', 'exchange', '};')
{inform_slots:
        country2: SZL
        country1: LAK
};
('{request_slot:\n', '\ttaiwan_rate:UNK\n', '};')
What is the exchange rate of SZL?
```

The second function is next function which based on the system actions.



```
def next(self, system_action):
    """ Generate next User Action based on last System Action """

    self.episode_over = False

    ##get system action
    sys_act = system_action['system_action']

    nl_response = None
    if sys_act == "inform":
        nl_response = self.response_inform(system_action)

    elif sys_act == "request":
        nl_response = self.response_request(system_action)

    elif sys_act == "confirm_answer":
        nl_response = self.response_confirm_answer(system_action)
    elif sys_act == "response":
        nl_response = self.response_response(system_action)
    elif sys_act == "closing":
        self.episode_over = True

    return nl_response
```

According to each system action, User simulator will have different response.
User simulator will detect if the agent is having the wrong direction by predicting the wrong intent or having the wrong slot tag in right intent, the situation above will head to an episode over and dialogue fail.



```
def response_response(self, system_action):
    response = []
    error = 0
    if system_action['action'][0] != self.goal.goal:
        response.append(str("Thanks"))
        error = 1
    for i in system_action['slot']:
        if self.goal.inform_slots.get(i) == None or self.goal.inform_slots.get(i) != system_aciton['slot'][i]:
            response.append(str("Thanks"))
            #response.append(str("slot error with "+i+" "+self.goal.inform_slots.get(i)+" "+system_aciton['slot
            error = 1
            break
    if error:
        self.episode_over = True
        return random.choice(response)
    else:
        self.success = True
        response.append(str("Thanks!"))
        return random.choice(response)
```

if it fail it will get a negative reward, and each turn of the dialogue will get a negative reward too, we construct a dialog_manager to update the reward for RL the next milestone. The reward of each episode is calculated as below: 1) one point is deducted from the reward after each turn. 2)at the end of an episode, increase reward points by [2*#maxturn](24 for now) if it is a successful dialogue, otherwise we deduct [#maxturn] points from the reward.

2. Dialogue State Tracking and Dialog Policy.

At first we will initialize a state for new user. Inside the state we record history conversation message for both system and user. Based on the history record we can help improve NLU accuracy. States are defined by it's presumed final goal. In other words, states indicate which slots we already have and which slots we still need to complete our action item. We exhaust all possible state and generate corresponding questions for user in order to complete the following conversation. We also use a threshold to check each NLU output. If our NLU's confidence about its' output is below the threshold, we will come up with a confirmation, then we decide whether to ask previous question or we can move on to the next state. For now, our response is based on policy based template and will be change into NLG based policy in milestone

3.Performance

an example of a successful dialogue:
turn 0,2,4,… is a natural language sentence from user simulator
turn 1,3,5,... is dialogue response specific for user simulator.

after 6 episodes,
reward: 18 ;   rate: 0.6666666666666666 (4success 2fail) avg_turn:5

How to run demo:

For user simulator, please follow the format of the successful dialogue. At first, user simulator will come with an user goal that it define to accomplish. Following the conversation, user simulator commit to finish this task. User simulator takes specific system response as input. :Which mainly comes in four ways:

{‘system_action’:['response’],’action_item’:[‘XXX’],’slot’:{‘slot_1’:'slot_value_1’, ‘slot_2’:'slot_value_2’}}
{‘system_action’:['request’],’action_item’:[‘XXX’],’slot’:{‘slot_1’:'slot_value_1’, ‘slot_2’:'slot_value_2’}}
{‘system_action’:[‘closing’],’action_item’:[‘closing’],’slot’:[]}
{‘system_action’”[‘confirm_answer’],’action_item’:[‘XXX’],’slot’:{‘slot_1’:'slot_value_1’}}

Request ask for more needed information, response return required system action response, closing indicate the closing of the dialogue and confirm answer is return when dialogue is not sure about slot tags.
But for demo version, just follow the instruction while running the code.
For dialogue state tracking, the are hint already implemented inside program, one can easily master its’ demo.
Run user simulator inside dir Code:

```
python3 dialogue_manager3.py
```

Run dialogue state tracker inside dir Code:

```
python3 dialogue_magager2.py
```