Red means uncertain/major revision needed

Pink means it is a suggestion but maybe not necessary

Type is a reserved keyword in Python

Variable
- var_type: Class
- name: str

+ get_type(): Class
+ get_name(): str

«Enumeration»
WeekDay
Monday: 0
Tuesday: 1
Wednesday: 2
Thursday: 3
Friday: 4
Saturday: 5
Sunday: 6

«Enumeration»
QuestionStatus
unanswered: 0
answered: 1
snoozed: 2
cancelled: 3

* + variables

* + variables

DatabaseWriter
- conn: Connection
- variables: Set[Variable]

+ get_variables() -> Iterable[Variable]
+ execute_sql(sql_command: str)
+ execute_query(sql_command: str): List[Tuple]

1 1

«Abstract»
Command
- is_executed: bool = False
- db_writer: DatabaseWriter

+ execute()
+ get_is_executed(): bool

«Abstract»
ReversibleCommand
- is_executed: bool = False
- db_writer: DatabaseWriter
- id: int

+ get_is_executed(): bool
+ get_id(): int

+ execute()
+ undo()

InsertCommand
- is_executed: bool = False
- db_writer: DatabaseWriter
- id: in
- variable: Variable
- timestamp: int
- new_value: Any

+ get_is_executed(): bool
+ get_id()

+ execute()
+ undo()

Data Gather Process

Only allow sets of questions to be done together?
(Can always just use a single question in QuestionOccurence,
but we could enforce this)

Question
- content: str
- variable: Variable

+ get_content(): str
+ get_variable(): Variable

+ questions
*

QuestionOccurence
- status: QuestionStatus
- questions: Question
- snooze_time: int = 0
- id: int

+ get_id(): int
+ get_status(): QuestionStatus

+ set_status(new_status: QuestionStatus)
+ snooze(minutes: int)

Status superfluous?
Snooze needs revision?

*

RecurringQuestionSet
- days: Set[WeekDay]
- hour: int
- minutes: int
- variables: Set[Variable]

+ get_days(): Set[WeekDay]
+ get_hour(): int
+ get_minutes(): int
+ get_variables(): Set[Variable]
+ is_due(current_time: float, previous_check: float): bool
+ generate_questions(formulator: Formulator): Set[QuestionOccurence]

*

QuestionSchedule
- active_occurences: Dict[int, QuestionOccurence]
- snoozed_occurences: Dict[int, QuestionOccurence]
- finished_occurences: Dict[int, QuestionOccurence]
- prev_refresh_timestamp: float
- timeout_time: float

- question_sets: Set[ReoccuringQuestionSet]
- formulator: Formulator

+ get_active_occurences(): Set[QuestionOccurence]

+ register_answered(id: int)

Answer
- id: str
- timestamp: float
- variable: Variable
- value: variable.var_type

+ get_id(): int
+ get_value(): Any
+ extract_command(db_writer: DatabaseWriter): InsertCommand

Formulator
+ formulate_question(variable: Variable): str

1

Questions are stores in a dict
that maps the ID of an Occurence
to the QuestionOccurence itself.

1

BackendQuestioner
- executed_commands: List[Command]
- undone_commands: List[Command]
- schedule: QuestionSchedule
- db_writer: DatabaseWriter

+ get_active_questions(): Set[Question]

+ mainloop(pipe: multiprocessing.Pipe)
- add_new_variable(var: Variable)
- store_answer(answer: Answer)
- register_incoming_answers(): Iterable[Answer]
- send_questions(questions: Set[Question])

1

We will probably need to use multiprocessing.
Perhaps a 'Pipe' to send data between processes.
Here it will be used to exchange Questions and Answers.
The BackendQuestioner will be able to match a Question
to its Answer via their ID.

For commands such as undo or adding a variable
we can make new classes. Then the BackendQuestioner
can just extract stuff from the pipe,
and check the type of the object (answer, new var, undo)
to figure out what needs to be done with it.

1

Pipe

1

GUI Process

GUI

1 1

1

Pipe

1

DatabaseReader
- conn: Connection

+ get_values_var(var: Variable) -> ndarray
+ get_vars_dict(vars: Iterable[Variable]) -> Dict[str, ndarray]

Output Production Process

«Abstract»
RuleExpression
- expression: str
- variables: Tuple[Variable]

+ __forward__(): Any

TriggerExpression
- expression: str

+ __forward()__: bool

MessageExpression
- expression: str

+ __forward()__: float

«Abstract»
Rule
- trigger: TriggerExpression
- messager: MessageExpression

+ check_fireable(): bool
+ fire(db: DatabaseReader): float

After firing,
only becomes fireable again
when new changes occur in the
database that make the rule fire.
Otherwise the same output would
be generated indifferently.

*
rules

OutputInvoker
- rules: Iterable[Rule]

+ mainloop(pipe: multiprocessing.Pipe)
- find_invoked_rules(): Iterable[Rule]
- invoke_rules():

1

«Abstract»
torch.nn.Module

+ get_named_parameters(): Iterable[Tuple]
+ get_parameters(): Iterable[Parameter]
+ forward(x: Tensor): Tensor

generator
*

OutputTextGenerator
- state: Tensor