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

Data Gather Process

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

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

+ execute()
+ undo()

Question
- content: str
- variable: Variable

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

+ questions *

QuestionOccurrence
+ questions: Set[Question]
+ id: int

*

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()

UnansweredQuestionQueue
- unanswered_questions: Set[QuestionOccurrence]

+ pop(): QuestionOccurence
+ add_new_occurences_if_any(schedule: QuestionSchedule)
+ is_empty(): bool

1

RecurringQuestionSet
- days: Set[WeekDay]
- hour: int
- minutes: int
- questions: Set[Question]
- still_due: bool
- timestamp_previous_questions: float

+ get_days(): Set[WeekDay]
+ get_hour(): int
+ get_minutes(): int
+ is_due(current_time: float): bool
+ generate_questions(current_time: float = time.time()): Set[QuestionOccurence]

*

QuestionSchedule
- question_sets: Iterable[ReoccuringQuestionSet]

+ get_new_occurences(current_time: float): Tuple[QuestionOccurence]

1

QuestionCommunicator
- 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

«Abstract»
Message
- id: int

+ get_id(): int

VariableMessage
- var: Variable

+ get_variable: Variable

TextMessage
- text: str

+ get_text(): str

QuestionMessage

AnswerMessage
- value: Any

+ get_value(): Any

1 1

Pipe

Multiprocessing is used to run the question generation,
the GUI and the output generation in parallel.
Python's multiprocessing.Pipe is used to send data between processes.
Here it will be used to exchange QuestionMessages, AnswerMessages
and OutputMessages.
The QuestionCommunicator will be able to match a Question
to its AnswerMessage via their ID. It will also store this
ID in the corresponding InsertCommand, making it possible to undo
a command given only the question or answer that it was derived from.

GUI Process

GUI

1 1

1 1

Pipe

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: Set[Variable]

+ get_variables() -> Set[Variable]
+ __call__(db: DatabaseReader): Any
# _hook_check_output_value(output: Any): bool

TriggerExpression
_hook_check_output_value(output: Any): bool

MessageExpression
_hook_check_output_value(output: Any): bool

EvaluationExpression
_hook_check_output_value(output: Any): bool

1 1 1

Rule
- trigger: TriggerExpression
- messager: MessageExpression
- evaluator: EvaluationExpression

+ check_fireable(db: DatabaseReader): 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 indefinitely.

* rules

«Abstract»
torch.nn.Module
+ get_named_parameters(): Iterable[Tuple]
+ get_parameters(): Iterable[Parameter]
+ forward(x: Tensor): Tensor

OutputInvoker
- rules: Iterable[Rule]

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

generator *

OutputTextGenerator
- state: Tensor

1