Assignment 2

● Graded

Student

Longpeng Xu

Total Points

9.1 / 10 pts

Autograder Score

6.0 / 6.0

Passed Tests

TestDesign (max_score=0) (0/0)
TestInheritance (max_score=0) (0/0)
TestTile (max_score=0.2) (0.2/0.2)
TestFloor (max_score=0.2) (0.2/0.2)
TestWall (max_score=0.2) (0.2/0.2)
TestGoal (max_score=0.2) (0.2/0.2)
TestGoal7030 (max_score=0.001) (0.001/0.001)
TestGoal7030Extra (max_score=0.001) (0.001/0.001)
TestEntity (max_score=0.2) (0.2/0.2)
TestCrate (max_score=0.2) (0.2/0.2)
TestPotion (max_score=0.2) (0.2/0.2)
TestStrengthPotion (max_score=0.2) (0.2/0.2)
TestMovePotion (max_score=0.2) (0.2/0.2)
TestFancyPotion (max_score=0.1) (0.1/0.1)
TestPlayer (max_score=0.2) (0.2/0.2)
TestConvertMaze (max_score=0.3) (0.3/0.3)
TestConvertMazeExtra (max_score=0.3) (0.3/0.3)
TestSokobanModel (max_score=0.5) (0.5/0.5)
TestSokobanModelExtra (max_score=0.5) (0.5/0.5)
TestSokobanModelAttemptMove (max_score=0.7) (0.7/0.7)
TestSokobanModelAttemptMove2 (max_score=0.2) (0.2/0.2)
TestSokobanModelAttemptMoveExtra (max_score=0.6) (0.6/0.6)
TestSokoban (max_score=0.4) (0.4/0.4)
TestSokoban2 (max_score=0.3) (0.3/0.3)
TestSokoban3 (max_score=0.3) (0.3/0.3)
TestSokoban7030 (max_score=0.5) (0.5/0.5)
TestSokoban7030Extra (max_score=0.5) (0.5/0.5)

Question 2

Readability **0.475** / 0.5 pts

2.1 Program Structure **0.25** / 0.25 pts

✔ **+ 0.25 pts** All of the following criteria has been met:

- Vertical whitespace has been used appropriately to separate logical blocks of code.
- Horizontal whitespace has been used to avoid terse lines of code.
- There are no sections of code which create undue burden on the reader.
- All lines of code conform to PEP8 style rules such as a maximum line length of 80 characters.

+ 0.125 pts Most of the following criteria has been met:

- Vertical whitespace has been used appropriately to separate logical blocks of code.
- Horizontal whitespace has been used to avoid terse lines of code.
- There are no sections of code which create undue burden on the reader.
- All lines of code conform to PEP8 style rules such as a maximum line length of 80 characters.

+ 0 pts At least one of the following criteria has been majorly violated:

- Vertical whitespace has been used appropriately to separate logical blocks of code.
- Horizontal whitespace has been used to avoid terse lines of code.
- There are no sections of code which create undue burden on the reader.
- All lines of code conform to PEP8 style rules such as a maximum line length of 80 characters.

2.2 Identifier Names **0.125** / 0.125 pts

✔ **+ 0.125 pts** All of the following criteria has been met:

- All identifier names conform to the correct casing for python code.
- All non-counter variables have a meaningful name which describes the variable independent of its context.
- Variable types are not included in the name when the type does not describe the variable.

+ 0.1 pts Most of the following criteria has been met:

- All identifier names conform to the correct casing for python code.
- All non-counter variables have a meaningful name which describes the variable independent of its context.
- Variable types are not included in the name when the type does not describe the variable.

+ 0 pts At least one of the following criteria has been majorly violated.

- All identifier names conform to the correct casing for python code.
- All non-counter variables have a meaningful name which describes the variable independent of its context.
- Variable types are not included in the name when the type does not describe the variable.

2.3 **Named constants** — **0.1** / 0.125 pts

+ **0.125 pts** All, non-trivial, fixed values (literal constants) in the code are represented by informative, named (symbolic) constants.

✔ + **0.1 pts** Most, non-trivial, fixed values (literal constants) in the code are represented by informative, named (symbolic) constants.

+ **0 pts** Only some, non-trivial, fixed values (literal constants) in the code are represented by informative, named (symbolic) constants.

The provided named constants have been used in some cases, but not all.

Question 3

Algorithmic Logic — **0.625** / 0.75 pts

3.1 **Single Instance of Logic** — **0.125** / 0.25 pts

+ **0.25 pts** Almost no code has been duplicated in your program. You have well designed functions with appropriate parameters to modularise your code.

✔ + **0.125 pts** Some code has been duplicated in your program. You have used some functions to modularise your code.

+ **0 pts** Large amounts of code are duplicated in your program. Poor use of functions to modularise your code.

Early classes duplicate code a lot. Inheritance should be used to reduce this.

furthermore, functions should not have nested functions inside them (convert_maze)

3.2 **Variable Scope** — **0.25** / 0.25 pts

✔ + **0.25 pts** Variables are declared locally in the functions, methods, and classes in which they are needed. Global variables, or code which functions as a global variable, has not been used.

+ **0 pts** Global variables, or code which functions as a global variable, have been used, reducing the clarify of function logic.

3.3 **Control Structures** — **0.25** / 0.25 pts

✔ + **0.25 pts** Logic is structured simply and clearly through good use of control structures.

+ **0.125 pts** A small number of control structures are unnecessarily complex.

+ **0 pts** Many control structures are poorly designed (e.g. excessive nesting, overly complex conditional logic, loops with multiple unnecessary exit points, ...).

Question 4

Object-Oriented Programming Structure **1.25** / 2 pts

4.1 **Classes & Instances** **0.75** / 0.75 pts

✔ **+ 0.75 pts** Methods, attributes and comments indicate each class is treated as a definition of a type and objects are used well in implementation. System behaviour is implemented in terms of objects sending messages to each other. State is never duplicated within a class through extraneous instance variables.

+ 0.5 pts Methods and attributes are based on provided design and objects are mostly used appropriately in implementation. Most behaviour is implemented in terms of objects sending messages to each other, with at most, minor duplication of state.

+ 0 pts Classes are treated as modules with functions, not as self-contained entities. Method implementations depend on external logic or variables.

4.2 **Encapsulation** **0** / 0.5 pts

+ 0.5 pts Classes are designed as independent modules with state and behaviour. Methods only directly access the state of the object on which they were invoked.

+ 0.25 pts Classes are almost always designed as independent modules with state and behaviour. At most one instance of breaking encapsulation is present.

✔ **+ 0 pts** Multiple methods directly access or modify instance variables (or the state) of instances of another class.

multiple cases of breaking encapsulation (accessing/modifying private variables directly)

4.3 **Inheritance & Polymorphism** **0.5** / 0.75 pts

+ 0.75 pts Subclasses extend the behaviour of their superclass without re-implementing behaviour, or breaking the superclass behaviour or design. Abstract classes have been used to effectively group shared behaviour amongst subclasses.

✔ **+ 0.5 pts** Subclasses extend the behaviour of their superclass with only minor instances of re-implementing behaviour.

+ 0 pts Provided class interfaces have been modified in ways detrimental to the design, or there are several places in which behaviour has been re-implemented among subclasses.

Inheritance should be used to cut down on duplicated code.

Furthermore, Potion is an abstract class, it shouldn't need to know anything about its concrete implementations (get_type)

Question 5

Documentation **0.75** / 0.75 pts

5.1 **In-Line Comment Clarity** **0.25** / 0.25 pts

✔ **+ 0.25 pts** Inline comments are used to assist readability in all of the following cases:

- Single lines with complex logic.
- Blocks of code with a singular purpose which should be documented.

+ 0.125 pts Inline comments are used to assist readability in most of the following cases:

- Single lines with complex logic.
- Blocks of code with a singular purpose which should be documented.

+ 0 pts More than one case of missing inline comments in the following situations:

- Single lines with complex logic.
- Blocks of code with a singular purpose which should be documented.

  Or has needless inline comments.

5.2 **Informative Docstrings** **0.5** / 0.5 pts

✔ **+ 0.5 pts** All modules, classes, methods and functions are clearly and concisely described via informative and complete docstrings.

+ 0.25 pts Most modules, classes, methods and functions are clearly and concisely described via informative and complete docstrings.

+ 0 pts Few modules, classes, methods and functions are clearly described via informative and complete docstrings.

Autograder Results

Functionality Test Results

TestDesign (max_score=0) (0/0)

1. test_classes_defined_correctly (weight=1):
   PASSED
2. test_clean_import (weight=1):
   PASSED
3. test_doc_strings (weight=1):
   PASSED
4. test_functions_defined (weight=1):
   PASSED
5. test_functions_defined_correctly (weight=1):
   PASSED

TestInheritance (max_score=0) (0/0)

1. test_class_hierarchy (weight=1):
   PASSED

TestTile (max_score=0.2) (0.2/0.2)

1. test_blocking (weight=1):
   PASSED
2. test_repr (weight=1):
   PASSED
3. test_str (weight=1):
   PASSED
4. test_type (weight=1):
   PASSED

TestFloor (max_score=0.2) (0.2/0.2)

1. test_blocking (weight=1):
   PASSED
2. test_repr (weight=1):
   PASSED
3. test_str (weight=1):
   PASSED
4. test_type (weight=1):
   PASSED

TestWall (max_score=0.2) (0.2/0.2)

1. test_blocking (weight=1):
   PASSED
2. test_repr (weight=1):
   PASSED
3. test_str (weight=1):
   PASSED
4. test_type (weight=1):
   PASSED

TestGoal (max_score=0.2) (0.2/0.2)

1. test_blocking (weight=1):
   PASSED
2. test_fill (weight=1):
   PASSED
3. test_is_filled (weight=1):
   PASSED
4. test_repr (weight=1):
   PASSED
5. test_repr_filled_goal (weight=1):
   PASSED
6. test_str (weight=1):
   PASSED
7. test_str_filled_goal (weight=1):
   PASSED
8. test_type (weight=1):
   PASSED
9. test_type_filled (weight=1):
   PASSED

TestGoal7030 (max_score=0.001) (0.001/0.001)

1. test_unfill (weight=1):
   PASSED

TestGoal7030Extra (max_score=0.001) (0.001/0.001)

1. test_str_unfill_goal (weight=1):
   PASSED
2. test_repr_unfill_goal (weight=1):
   PASSED

TestEntity (max_score=0.2) (0.2/0.2)

1. test_movable (weight=1):
   PASSED
2. test_repr (weight=1):
   PASSED
3. test_str (weight=1):
   PASSED
4. test_type (weight=1):
   PASSED

TestCrate (max_score=0.2) (0.2/0.2)

1. test_movable (weight=1):
   PASSED
2. test_repr (weight=1):
   PASSED
3. test_str (weight=1):
   PASSED
4. test_strength (weight=1):
   PASSED
5. test_type (weight=1):
   PASSED

TestPotion (max_score=0.2) (0.2/0.2)

1. test_effect (weight=1):
   PASSED
2. test_movable (weight=1):
   PASSED
3. test_repr (weight=1):
   PASSED
4. test_str (weight=1):
   PASSED
5. test_type (weight=1):
   PASSED

TestStrengthPotion (max_score=0.2) (0.2/0.2)

1. test_effect (weight=1):
   PASSED
2. test_movable (weight=1):
   PASSED
3. test_repr (weight=1):
   PASSED
4. test_str (weight=1):
   PASSED
5. test_type (weight=1):
   PASSED

TestMovePotion (max_score=0.2) (0.2/0.2)

1. test_effect (weight=1):
   PASSED
2. test_movable (weight=1):
   PASSED
3. test_repr (weight=1):
   PASSED
4. test_str (weight=1):
   PASSED
5. test_type (weight=1):
   PASSED

TestFancyPotion (max_score=0.1) (0.1/0.1)

1. test_effect (weight=1):
   PASSED
2. test_movable (weight=1):
   PASSED
3. test_repr (weight=1):
   PASSED
4. test_str (weight=1):
   PASSED
5. test_type (weight=1):
   PASSED

TestPlayer (max_score=0.2) (0.2/0.2)

1. test_add_move (weight=1):
   PASSED
2. test_add_move_neg (weight=1):
   PASSED
3. test_add_move_neg_multi (weight=1):
   PASSED
4. test_add_move_neg_multi_alt (weight=1):
   PASSED
5. test_add_strength (weight=1):
   PASSED
6. test_apply_moves (weight=1):
   PASSED
7. test_apply_strength (weight=1):
   PASSED
8. test_get_moves (weight=1):
   PASSED
9. test_get_strength (weight=1):
   PASSED
10. test_movable (weight=1):
    PASSED
11. test_movable_after_move (weight=1):
    PASSED
12. test_repr (weight=1):
    PASSED
13. test_str (weight=1):
    PASSED
14. test_type (weight=1):
    PASSED

TestConvertMaze (max_score=0.3) (0.3/0.3)

1. test_convert_maze (weight=1):
   PASSED
2. test_entities (weight=1):
   PASSED
3. test_player_position (weight=1):
   PASSED

TestConvertMazeExtra (max_score=0.3) (0.3/0.3)

1. test_convert_maze (weight=1):
   PASSED

TestSokobanModel (max_score=0.5) (0.5/0.5)

1. test_get_entities (weight=1):
   PASSED
2. test_get_maze (weight=1):
   PASSED
3. test_get_player_moves (weight=1):
   PASSED
4. test_has_won (weight=1):
   PASSED
5. test_player_position (weight=1):
   PASSED
6. test_player_strength (weight=1):
   PASSED

TestSokobanModelExtra (max_score=0.5) (0.5/0.5)

1. test_get_maze (weight=1):
   PASSED
2. test_get_entities (weight=1):
   PASSED
3. test_player_position (weight=1):
   PASSED
4. test_get_player_moves (weight=1):
   PASSED
5. test_player_strength (weight=1):
   PASSED

TestSokobanModelAttemptMove (max_score=0.7) (0.7/0.7)

1. test_attempt_move (weight=1):
   PASSED
2. test_attempt_move_false (weight=1):
   PASSED
3. test_attempt_move_false_moves_remaining (weight=1):
   PASSED
4. test_attempt_move_invalid (weight=1):
   PASSED
5. test_attempt_move_push_crate (weight=2):
   PASSED
6. test_attempt_move_remaining (weight=1):
   PASSED
7. test_attempt_move_onto_goal (weight=2):
   PASSED

TestSokobanModelAttemptMove2 (max_score=0.2) (0.2/0.2)

1. test_attempt_move_to_strength_potion (weight=1):
   PASSED
2. test_attempt_move_invalid_strength (weight=1):
   PASSED

TestSokobanModelAttemptMoveExtra (max_score=0.6) (0.6/0.6)

1. test_get_maze (weight=1):
   PASSED

TestSokoban (max_score=0.4) (0.4/0.4)

1. test_display (weight=1):
   PASSED
2. test_play_game (weight=1):
   PASSED
3. test_game_win (weight=1):
   PASSED
4. test_game_lost (weight=1):
   PASSED

TestSokoban2 (max_score=0.3) (0.3/0.3)

1. test_display (weight=1):
   PASSED
2. test_game_potions_and_heavy_crates (weight=1):
   PASSED

TestSokoban3 (max_score=0.3) (0.3/0.3)

1. test_display (weight=1):
   PASSED
2. test_game_fancy_potion (weight=1):
   PASSED

TestSokoban7030 (max_score=0.5) (0.5/0.5)

1. test_display (weight=0):
   PASSED
2. test_play_game_with_undo (weight=1):
   PASSED
3. test_play_game_with_undo_potion (weight=1):
   PASSED
4. test_play_game_with_undo_crate (weight=1):
   PASSED

TestSokoban7030Extra (max_score=0.5) (0.5/0.5)

1. test_display (weight=0):
   PASSED
2. test_play_game_with_undo (weight=1):
   PASSED

Submitted Files

a2.py

Download

```python
from a2_support import *



class Tile():
    """ abstract class for tiles
    """
    def __init__(self) -> None:
        """ attributes of Tile
        """
        pass

    def is_blocking(self) -> bool:
        """ a tile default to be NOT blocked

        Outputs:
            : the attribute of blocking moving instances (bool)
        """
        return False

    def get_type(self) -> str:
        """ get the type of a tile

        Outputs:
            : the type of a tile (str)
        """
        if type(self).__name__ == 'Tile':
            return 'Abstract Tile'
        else:
            return type(self).__name__[0]

    def __str__(self) -> str:
        """ same as get_type(self)
        """
        return self.get_type()

    def __repr__(self) -> str:
        """ same as get_type(self)
        """
        return self.get_type()



class Floor(Tile):
    """ tiles as empty spaces which do not block moving instances
    """
```

```
47      def get_type(self) -> str:
48          return ' '
49
50
51
52  class Wall(Tile):
53      """ tiles as walls which block moving instances
54      """
55      def is_blocking(self) -> bool:
56          """ a wall default to be blocking
57
58          Outputs:
59              : the attribute of blocking moving instances (bool)
60          """
61          return True
62
63
64
65  class Goal(Tile):
66      """ tiles as a goal location for a crate
67      """
68      def __init__(self) -> None:
69          """ attributes of Goal: _filled: whether a goal is filled
70          """
71          super().__init__()
72          self._filled = False
73
74      def fill(self) -> None:
75          """ set a goal to be filled
76          """
77          self._filled = True
78
79      def is_filled(self) -> bool:
80          """ returns True when the goal is filled, else False
81
82          Outputs:
83              : the state of whether a goal is filled (bool)
84          """
85          if self._filled:
86              return True
87          else:
88              return False
89
90      def __str__(self) -> str:
91          """ returns FILLED_GOAL if a goal tile is filled, otherwise GOAL
92
93          Outputs:
94              : string representation of a goal (str)
95          """
```

```
96          if self._filled:
97              return FILLED_GOAL
98          else:
99              return GOAL
100
101     def __repr__(self) -> str:
102         """ same as __str__(self)
103         """
104         return str(self)
105
106     def unfill(self) -> None:
107         """ unfill a goal
108         """
109         self._filled = False
110
111
112
113 class Entity():
114     """ abstract class for entities
115     """
116     def __init__(self) -> None:
117         """ entities has no attributes beyond self
118         """
119         pass
120
121     def get_type(self) -> str:
122         """ get the type of an entity
123
124         Outputs:
125             : the type of an entity (str)
126         """
127         if type(self).__name__ == 'Entity':
128             return 'Abstract Entity'
129         else:
130             return type(self).__name__[0]
131
132     def is_movable(self) -> bool:
133         """ entities are not movable by default
134
135         Outputs:
136             : the movability of an entity (bool)
137         """
138         return False
139
140     def __str__(self) -> str:
141         """ same as get_type(self)
142         """
143         return self.get_type()
144
```

```
    def __repr__(self) -> str:
        """ same as get_type(self)
        """
        return self.get_type()


class Crate(Entity):
    """ entities as crates
    """
    def __init__(self, strength: int) -> None:
        """ attributes of crates

        Inputs:
            strength: the strength required to move a crate (int)
        """
        self._strength = strength

    def get_strength(self) -> int:
        """ get the strength required to move a crate

        Outputs:
            : the strength required to move a crate (int)
        """
        return self._strength

    def is_movable(self) -> bool:
        """ a crate is movable

        Outputs:
            : the movability of a crate (bool)
        """
        return True

    def __str__(self) -> str:
        """ the string representation of a crate

        Outputs:
            : the strength required to move a crate (str)
        """
        return str(self._strength)

    def __repr__(self) -> str:
        """ same as __str__(self)
        """
        return str(self)
```

```python
class Potion(Entity):
    """ entities as potions
    """
    def __init__(self) -> None:
        """ potions has no attributes beyond self
        """
        pass

    def get_type(self) -> str:
        """ get the type of a potion

        Outputs:
            : the type of a potion (str)
        """
        if type(self).__name__ == 'Potion':
            return 'Potion'
        else:
            return type(self).__name__[0]

    def effect(self) -> dict[str, int]:
        """ set a holder for the effects of different potions

        Outputs:
            : an empty holder for the effects (dict[str, int])
        """
        return dict()



class StrengthPotion(Potion):
    """ entities as potions of type StrengthPotion
    """
    def effect(self) -> dict[str, int]:
        """ a strength potion adds 2 'strength's

        Output:
            : the effect of a strength potion (dict[str, int])
        """
        return {'strength': 2}



class MovePotion(Potion):
    """ entities as potions of type MovePotion
    """
    def effect(self) -> dict[str, int]:
        """ a move potion adds 5 'move's

        Outputs:
```

```
            : the effect of a move potion (dict[str, int])
        """
        return {'moves': 5}



class FancyPotion(Potion):
    """ entities as potions of type FancyPotion
    """
    def effect(self) -> dict[str, int]:
        """ a fancy potion adds 2 'strength's and 2 'move's

        Outputs:
            : the effect of a fancy potion (dict[str, int])
        """
        return {'strength': 2, 'moves': 2}



class Player(Entity):
    """ entities as players
    """
    def __init__(self, start_strength: int, moves_remaining: int) -> None:
        """ attributes of a player

        Inputs:
            start_strength: the initial strength of a player (int)
            moves_remaining: the remaining moves of a player (int)
        """
        self._strength = start_strength
        self._moves_remaining = moves_remaining

    def is_movable(self) -> bool:
        """ a player is movable when his/her has moves available

        Outputs:
            : the movability of a player (bool)
        """
        return self._moves_remaining > 0

    def get_strength(self) -> int:
        """ get the strength of a player

        Outputs:
            : the strength of a player (int)
        """
        return self._strength

    def add_strength(self, amount: int) -> None:
```

```
292         """ add the given strength to a player
293
294         Inputs:
295             amount: the amount of strength to be added (int)
296         """
297         self._strength += amount
298
299     def get_moves_remaining(self) -> int:
300         """ get the remaining moves of a player
301
302         Outputs:
303             : the remaining moves of a player (int)
304         """
305         return self._moves_remaining
306
307     def add_moves_remaining(self, amount: int) -> None:
308         """ add the given moves to a player
309
310         Inputs:
311             amount: the number of moves to be added (int)
312         """
313         self._moves_remaining += amount
314
315     def apply_effect(self, potion_effect: dict[str, int]) -> None:
316         """ add available effects, namely moves and strength, to a player,
317             given specific potions
318
319         Inputs:
320             potion_effect: available effects with amounts (dict[str, int])
321         """
322         effects = list(potion_effect.keys())
323
324         if 'moves' in effects:
325             self.add_moves_remaining(potion_effect['moves'])
326         if 'strength' in effects:
327             self.add_strength(potion_effect['strength'])
328
329
330
331 def convert_maze(game: list[list[str]]) -> tuple[Grid, Entities, Position]:
332     """ convert string representation of a maze to be an object-oriented one,
333         and re-locate entities and player position information from the maze
334
335     Inputs:
336         game: the raw maze (list[list[str]])
337
338     Outputs:
339         : the formatted maze, the entities, and the player's position
340         (tuple[Grid, Entities, Position])
```

```
    """
    # default settings
    player_row, player_col = -1, -1
    entities = dict()

    def make_floor(i: int, j: int) -> None:
        """ helper function to make entity/tile on position (i, j) on the maze
            to be Floor()

        Inputs:
            i: row of the position (int)
            j: col of the position (int)
        """
        game[i][j] = Floor()
        return

    # handle the raw maze cell by cell
    for i in range(len(game)):
        for j in range(len(game[i])):

            # handle the player
            if game[i][j] == 'P':
                player_row, player_col = i, j
                make_floor(i, j)

            # handle any crates
            elif not game[i][j] in [WALL, GOAL, FLOOR, STRENGTH_POTION,
                                    MOVE_POTION, FANCY_POTION]:
                strength = int(game[i][j])
                entities[(i,j)] = Crate(strength)
                make_floor(i, j)

            # handle any potions
            elif game[i][j] == STRENGTH_POTION:
                entities[(i,j)] = StrengthPotion()
                make_floor(i, j)
            elif game[i][j] == MOVE_POTION:
                entities[(i,j)] = MovePotion()
                make_floor(i, j)
            elif game[i][j] == FANCY_POTION:
                entities[(i,j)] = FancyPotion()
                make_floor(i, j)

            # handle any tiles
            elif game[i][j] == WALL:
                game[i][j] = Wall()
            elif game[i][j] == GOAL:
                game[i][j] = Goal()
            elif game[i][j] == FLOOR:
```

```
390                 make_floor(i, j)
391
392     return (game, entities, (player_row, player_col))
393
394
395
396 class SokobanModel():
397     """ the model for the game
398     """
399     def __init__(self, maze_file: str) -> None:
400         """ the attributes of the model
401             player_last_met can be CRATE or 'CRATE then GOAL' or 'Potion' or -1
402
403         Inputs:
404             maze_file: the directory of a raw maze
405         """
406         raw_maze, player_stats = read_file(maze_file)
407         strength, moves = player_stats[0], player_stats[1]
408
409         # attributes for normal gameplay
410         self._maze, self._entities, self._player_position = \
411             convert_maze(raw_maze)
412         self._player = Player(strength, moves)
413
414         # attributes for undo
415         self._player_position_history = [self._player_position]
416         self._gone_crate_history, self._gone_potion_history = dict(), []
417         self._player_last_hit = -1
418
419     def get_maze(self) -> Grid:
420         """ get a formatted maze
421
422         Outputs:
423             : the formatted maze (Grid)
424         """
425         return self._maze
426
427     def get_entities(self) -> Entities:
428         """ get the entities of the model
429
430         Outputs:
431             : the entities of the model (Entities)
432         """
433         return self._entities
434
435     def get_player_position(self) -> tuple[int, int]:
436         """ get the position of the player
437
438         Outputs:
```

```
439             : the position of the player (tuple[int, int])
440         """
441         return self._player_position
442 
443     def get_player_moves_remaining(self) -> int:
444         """ get the remaining moves of the player
445 
446         Outputs: the remaining moves of the player (int)
447         """
448         return self._player.get_moves_remaining()
449 
450     def get_player_strength(self) -> int:
451         """ get the strength of the player
452 
453         Outputs: the strength of the player (int)
454         """
455         return self._player.get_strength()
456 
457     def attempt_move(self, direction: str) -> bool:
458         """ move the player in case of crates (incl. ones next to goals),
459             potions, and tiles, given user's prompt. Invalid moves cannot
460             move the player. Undo of the last valid move is accessible
461 
462         Inputs:
463             direction: the direction to move the player (str)
464 
465         Outputs:
466             : the validity of the move (bool)
467         """
468 
469         def get_next_position(current_position: tuple) -> tuple[int, int]:
470             """ helper function to get next position of player, given direction
471                 Warning
472                  get_next_position(), next(), and next_tile() are nested within
473                  attempt_move(direction), since they are dependent on direction
474 
475             Inputs:
476                 current_position: the current position of the player (tuple)
477 
478             Outputs:
479                 : his/her next position (tuple[int, int])
480             """
481             row, col = current_position
482 
483             # branching by direction
484             if direction == UP:
485                 return (row-1, col)
486             elif direction == DOWN:
487                 return (row+1, col)
```

```
488             elif direction == LEFT:
489                 return (row, col-1)
490             elif direction == RIGHT:
491                 return (row, col+1)
492 
493         def next(row_or_col: str) -> int:
494             """ helper function to get the row or column of the next position
495                 of the player
496 
497             Inputs:
498                 row_or_col: 'row' or 'col' (str)
499 
500             Outputs:
501                 row|col: the row or the column (int)
502             """
503             row, col = get_next_position(self._player_position)
504             if row_or_col == 'row':
505                 return row
506             elif row_or_col == 'col':
507                 return col
508 
509         def next_tile():
510             """ helper function to get the tile that the player will meet in the
511                 next step
512 
513             Outputs:
514                 : the tile (Wall|Floor|Goal)
515             """
516             row, col = get_next_position(self._player_position)
517             return self.get_maze()[row][col]
518 
519         # method variable that simplifies many codes
520         next_position = get_next_position(self._player_position)
521 
522         # if the move is invalid
523         if direction not in [UP, DOWN, LEFT, RIGHT]:
524             return False
525         elif isinstance(next_tile(), Wall):
526             return False
527 
528         # if next position of player is an entity
529         elif next_position in self._entities.keys():
530 
531             # if next position of player stands a crate
532             if type(self._entities[next_position]) == Crate:
533 
534                 # if the crate cannot be moved
535                 if self.get_player_strength() < self._entities[next_position]\
536                     .get_strength()\
```

```
537                 or next('row') not in range(len(self._maze)) \
538                 or next('col') not in range(len(self._maze[0])):
539                 return False
540 
541             else:
542                 # move the crate
543                 self._player_last_hit = CRATE
544                 row_2, col_2 = get_next_position(next_position)
545                 self._entities[(row_2, col_2)] = \
546                     self._entities[next_position]
547                 del self._entities[next_position]
548 
549                 # if next position of a crate stands an unfilled goal
550                 if type(self._maze[row_2][col_2]) == Goal and \
551                     not self._maze[row_2][col_2].is_filled():
552                     self._player_last_hit = 'CRATE then GOAL'
553                     self._gone_crate_history[(row_2, col_2)] = \
554                         self._entities[(row_2, col_2)]
555                     del self._entities[(row_2, col_2)]
556                     self._maze[row_2][col_2].fill()
557 
558         # if next position of player stands a potion
559         elif type(self._entities[next_position]) in \
560             [StrengthPotion, MovePotion, FancyPotion]:
561             # apply the potion to the player and remove it from entities
562             self._player_last_hit = 'Potion'
563             self._gone_potion_history += [self._entities[next_position]]
564             potion = self._entities[next_position]
565             self._player.apply_effect(potion.effect())
566             del self._entities[next_position]
567 
568     # if next position of player stands anything unrelated to undo
569     else:
570         self._player_last_hit = -1
571 
572     # update player information about moves
573     self._player._moves_remaining -= 1
574     self._player_position = next_position
575     self._player_position_history += [self._player_position]
576     return True
577 
578 def has_won(self) -> bool:
579     """ judge if the game has been won given the current maze. A game has
580         been won if having all goals be filled
581 
582     Outputs:
583         : if the game has been won (bool)
584     """
585     return not any(isinstance(tile, Goal) and not tile.is_filled()
```

```
                    for row in self._maze for tile in row)

    def undo(self) -> None:
        """ undo all the effects by the last valid move, w.r.t. crates, goals,
            potions, and the player
        """
        # reverse player information about moves
        self._player._moves_remaining += 1
        self._player_position = self._player_position_history[-2]
        row_move, col_move = [self._player_position_history[-1][i] -
                              self._player_position_history[-2][i]
                              for i in range(2)]

        # if a crate was moved to a goal, recover both
        if self._player_last_hit == 'CRATE then GOAL':
            position_crate, crate = list(self._gone_crate_history.items())[-1]
            row, col = position_crate
            self._entities[(row-row_move, col-col_move)] = crate
            self._maze[row][col].unfill()

        # if next position of player stood a crate
        elif self._player_last_hit == CRATE:
            row, col = self._player_position_history[-1]
            self._entities[(row, col)] = \
                self._entities[(row+row_move, col+col_move)]
            del self._entities[(row+row_move, col+col_move)]

        # if next position of player stood a potion
        elif self._player_last_hit == 'Potion':
            last_potion = self._gone_potion_history[-1]
            self._entities[self._player_position_history[-1]] = last_potion
            if type(last_potion) == StrengthPotion:
                self._player._strength -= 2
            elif type(last_potion) == MovePotion:
                self._player._strength -= 5
            elif type(last_potion) == FancyPotion:
                self._player._strength -= 2
                self._player._moves_remaining -= 2



class Sokoban():
    """ the controller of the game
    """
    def __init__(self, maze_file: str) -> None:
        """ attributes of Sokoban

        Inputs:
            maze_file: the directory of a maze file (str)
```

```
635         """
636         self._model = SokobanModel(maze_file)
637         self._view = SokobanView()
638 
639     def display(self) -> None:
640         """ display the game and the statistics of the player
641         """
642         self._view.display_game(self._model._maze, self._model._entities,
643                                 self._model._player_position)
644         self._view.display_stats(self._model.get_player_moves_remaining(),
645                                  self._model.get_player_strength())
646 
647     def play_game(self) -> None:
648         """ the whole process of the game
649         """
650         while self._model.has_won() == False:
651 
652             # if the game has been lost
653             if self._model.has_won() == False and \
654                 self._model.get_player_moves_remaining() <= 0:
655                 print('You lost!')
656                 return
657 
658             # display the game state
659             self._view.display_game(self._model._maze, self._model._entities,
660                                     self._model._player_position)
661             self._view.display_stats(self._model.get_player_moves_remaining(),
662                                      self._model.get_player_strength())
663 
664             # prompt a user for a move
665             move = input('Enter move: ')
666             if move == 'u':
667                 self._model.undo()
668             elif move == 'q':
669                 return
670             elif self._model.attempt_move(move):
671                 pass
672             else:
673                 print('Invalid move\n')
674 
675             # if the game has been won
676             if self._model.has_won() == True:
677                 self._view.display_game(self._model._maze,
678                                         self._model._entities,
679                                         self._model._player_position)
680                 self._view.display_stats(self._model.get_player_moves_remaining\
681                                          (), self._model.get_player_strength())
682                 print('You won!')
683                 return
```

```
684
685
686
687 def main():
688     """ run the maze file and the game
689     """
690     game = Sokoban('maze_files/maze1.txt')
691     game.play_game()
692     pass
693
694 if __name__ == '__main__':
695     main()
```