

Lab 3 Task 8 Solution

8) Additional Tasks

8.1) A user player

```
1  ...
2  class UserPlayer(Player):
3
4  def move(self, current: int, min_step: int,
5           max_step: int, goal: int) -> int:
6
7      amount = 0
8
9      while True:
10         amount_raw = input('Enter step amount ({}-{})'.format(min_step
11         , max_step))
12
13         if len(amount_raw.strip()) == 0:
14             print('Please select integer value between {} and {}'.
15             format(min_step, max_step))
16             continue
17
18         if re.search(r'^0-9+', amount_raw):
19             print('Please select integer value between {} and {}'.
20             format(min_step, max_step))
21             continue
22
23         amount = int(amount_raw)
24         if amount < min_step or amount > max_step:
25             print('Please select steps between {} and {}'.format(
26             min_step, max_step))
27             continue
28
29         break
30
31     return amount
32
33  ...
34  def make_player(generic_name: str) -> Player:
35      ...
36      return UserPlayer(name)
```

```

34     ...
35
36     if __name__ == '__main__':
37         # Uncomment the lines below to check your work using
38         # python_ta and doctest.
39         # import python_ta
40         # python_ta.check_all(config={
41         #     'extra-imports': ['random'],
42         #     'allowed-io': [
43         #         'main',
44         #         'make_player',
45         #         'move',
46         #         'play_one_turn'
47         #     ]
48         # })
49     main()

```

8.2) A strategic player

The solution to this problem makes following assumptions:

- *goal* of 21
- *min_step* of 1
- *max_step* of 3
- one of the player as *StrategicPlayer*
- the other as *RandomPlayer*

We need to create *StrategicPlayer* that always wins as player 1, and does win as player 2 when a bad move is by the other player. Also, we need to adjust *make_player* so a player's type can be chosen by user.

```

1     ...
2
3     # ===== SOLUTION (Task 8.2) =====
4
5     class StrategicPlayer(Player):
6
7         def move(self, current: int, min_step: int,
8                 max_step: int, goal: int) -> int:
9
10            return 3
11
12    # =====
13
14    ...
15
16    def make_player(generic_name: str) -> Player:
17        ...

```

```

18 # ===== SOLUTION (Task 8.2) =====
19 player_type_list = ['r', 'u', 's']
20
21 while True:
22     player_type = input(
23         'Enter player type '
24         '(r - Random Player, u - User Player, s - Strategic Player
25     ))
26
27     if player_type not in player_type_list:
28         print('Please select one of the three values '
29             '({})'.format(','.join(player_type_list)))
30         continue
31
32     break
33
34 if player_type == 'u':
35     return UserPlayer(name)
36 elif player_type == 's':
37     return StrategicPlayer(name)
38 elif player_type == 'r':
39     return RandomPlayer(name)
40
41 # =====
42 ...
43
44 if __name__ == '__main__':
45     # Uncomment the lines below to check your work using
46     # python_ta and doctest.
47     # import python_ta
48     # python_ta.check_all(config={
49     #     'extra-imports': ['random'],
50     #     'allowed-io': [
51     #         'main',
52     #         'make_player',
53     #         'move',
54     #         'play_one_turn'
55     #     ]
56     # })
57     main()

```

8.3) Tracking and reporting a player's record

We need to update *Player* using `__str__`, attributes and other methods so players' name and record are displayed at the end of each game. We also need to update *NumberGame* so winner's record are updated at the end of each game.

```

1 ...
2
3 class Player:

```

```

4         """A player in number game
5
6         == Attributes ==
7         name:
8             The name of player
9         # ===== SOLUTION (Task 8.3) =====
10        wins:
11            The number of wins
12        # =====
13
14        == Representation invariants ==
15        - len(name.strip()) != 0
16        - 0 <= self.current <= self.goal
17        - 0 < self.min_step <= self.max_step <= self.goal
18        """
19        name: str
20        # ===== SOLUTION (Task 8.3) =====
21        wins: int
22        # =====
23
24        def __init__(self, name: str) -> None:
25            """Initialize this Player
26
27            Precondition:
28                - len(name.strip()) != 0
29            """
30            self.name = name
31            # ===== SOLUTION (Task 8.3) =====
32            self.wins = 0
33            # =====
34
35        def __str__(self):
36            return '{} - {} wins'.format(self.name, self.wins)
37
38        # ===== SOLUTION (Task 8.3) =====
39        def add_win(self):
40            """Increments a win count to this player"""
41            self.wins += 1
42            # =====
43
44        ...
45
46        class NumberGame:
47            ...
48
49            def play(self) -> str:
50                """Play one round of this NumberGame. Return the name of the
51                winner.
52
53                A "round" is one full run of the game, from when the count
54                starts
55                at 0 until the goal is reached.
56                """
57                while self.current < self.goal:

```

```

56         self.play_one_turn()
57         # The player whose turn would be next (if the game weren't
over) is
58         # the loser. The one who went one turn before that is the
winner.
59         winner = self.whose_turn(self.turn - 1)
60         # ===== SOLUTION (Task 8.3) =====
61         winner.add_win()
62         # =====
63         return winner.name
64
65     ...
66
67 if __name__ == '__main__':
68     # Uncomment the lines below to check your work using
69     # python_ta and doctest.
70     # import python_ta
71     # python_ta.check_all(config={
72     #     'extra-imports': ['random'],
73     #     'allowed-io': [
74     #         'main',
75     #         'make_player',
76     #         'move',
77     #         'play_one_turn'
78     #     ]
79     # })
80     main()

```