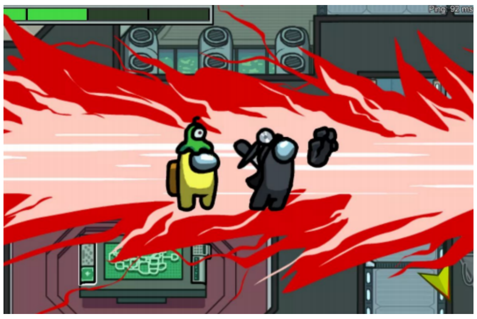
National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2021/2022

Extra Practice 8

Question 1

In a game of "Among Us", there are two groups of characters - normal crewmates and impostors.



The rules for our version are:

- Everyone starts off without a location and has to move to a location first (we'll assume this is not going to be a problem)
- Only the impostors can kill crewmates, and can only kill a crewmate if they are in the same location
- If a crewmate dies, he becomes a ghost and can still move around like a usual crewmate and do tasks
- Both impostors and crewmate can report a dead body, but must be in the same location as the dead body
- Impostors can pretend to do tasks as well

Define a class `Place` that is initialized with a name. It should have the following methods:

- `get_name()` that returns the name of the place
- `get_people()` that returns the names of all the people (dead or alive) in the location currently in a string `"[Person A], [Person B], [Person C] in [name of place]"`. If there is no one, return `"Nobody here"`

Define another class `Person` that is initialized with a name. It should have the following methods:

- `get_name()` that returns the name of the place
- `get_state()` that returns `"Alive"` if the person is still alive and `"Killed by [Murderer's name]"` if he/she is dead
- `do_task(task)` that takes in a task as an input string and returns the following sentence `"[Name of person] does [Name of task]."`

- `move(location)` that takes in a place object and moves the person from his current place to that new location
  - At the start of the game when this is called, return `"[Name of person] moves to [Name of place]"`
  - If the location is his current location, return `"Already here"`
  - Otherwise, return `"[Name of person] moves from [Current location name] to [New location name]"`
- `report(person)` that takes in a person object and tries to report it
  - If the person reporting is already dead, return `"Ghosts cannot report"`
  - If the person he is trying to report is oneself, return `"Cannot report oneself"`
  - If the person he is trying to report is not in the same location as he is in, return `"[Name of other person] is in [Name of location]"`
  - If the person he is trying to report is still alive, return `"[Name of person] is still alive"`
  - Otherwise, return `"[Name of person] reports [Name of other person]..."`
    He also immediately suspects everyone alive in the room at that moment and additionally returns the sentence `"... and suspects [Person A's name], [Person B's name] ... [Person D's name]"`
    If there is nobody else that he can suspect, the sentence `"... and nobody to suspect"` is returned

We now want to define another class `Impostor` that is a subclass of `Person`, and is initialized with two inputs - name and weapon. It should have the following additional methods:

- `kill(person)` that takes in a person object and attempts to kill it
  - If the person is oneself, return `"Cannot kill oneself"`
  - If the person is not in the same room as him, return `"[Victim's name] is in [Name of location victim is in]. Cannot kill"`
  - If the person is a fellow impostor, return `"Cannot kill another impostor"`
  - If the person is already dead, return `"Already killed"`
  - Otherwise, kill that person and return `"[Name] killed [name of victim] with a [name of weapon]"`
- `victim_list()` that returns the names of all the people he killed in alphabetical order `"[Name] killed [Person A], [Person B] ... [Person D]."`
  - If he has not killed anyone yet, return `"Killed nobody"`

Sample Execution:

```
>>> ravn = Impostor("Ravn", "Pistol")
>>> brendan = Impostor("Brendan", "Knife")
>>> daryl = Person("Daryl")
>>> tze = Person("Tze Lynn")
>>> ryan = Person("Ryan")
>>> clifton = Person("Clifton")
>>> daniel = Person("Daniel")
>>> room = Place("classroom")
>>> toilet = Place("toilet")
>>> hall = Place("hall")

>>> daniel.get_state()
'Alive'
>>> daniel.move(toilet)
'Daniel moves to toilet'
>>> daryl.move(room)
'Daryl moves to classroom'
>>> tze.move(hall)
'Tze Lynn moves to hall'
>>> ryan.move(toilet)
'Ryan moves to toilet'
>>> brendan.move(toilet)
'Brendan moves to toilet'
>>> ravn.move(room)
'Ravn moves to classroom'
>>> clifton.move(room)
'Clifton moves to classroom'
>>> room.get_people()
'Daryl, Ravn, Clifton in classroom'
>>> ravn.kill(ravn)
'Cannot kill oneself'
>>> ravn.victim_list()
'Killed nobody'
>>> tze.do_task("wiring")
'Tze Lynn does wiring'
>>> brendan.kill(clifton)
'Clifton is in classroom. Cannot kill'
>>> ravn.kill(clifton)
'Ravn killed Clifton with a Pistol'
>>> ravn.report(tze)
'Tze Lynn is in hall'
>>> ryan.report(brendan)
'Brendan is still alive'
>>> daniel.move(room)
'Daniel moves from toilet to classroom'
>>> ravn.report(clifton)
'Ravn reports Clifton and suspects Daryl, Daniel'
>>> ravn.victim_list()
'Ravn killed Clifton'
```

```
>>> clifton.get_state()
'Killed by Ravn'
>>> brendan.kill(ryan)
'Brendan killed Ryan with a Knife'
>>> brendan.move(room)
'Brendan moves from toilet to classroom'
>>> brendan.kill(ravn)
'Cannot kill another impostor'
>>> brendan.do_task("swipe card")
'Brendan does swipe card'
>>> tze.move(toilet)
'Tze Lynn moves from hall to toilet'
>>> tze.report(ryan)
'Tze Lynn reports Ryan and nobody to suspect'
>>> hall.get_people()
'Nobody here'
>>> room.get_people()    # order does not matter here
'Daryl, Ravn, Clifton, Daniel, Brendan in classroom'
>>> brendan.kill(daniel)
'Brendan killed Daniel with a Knife'
>>> ravn.report(daniel)
'Ravn reports Daniel and suspects Daryl, Brendan'
>>> brendan.victim_list()
'Brendan killed Daniel, Ryan'   # must be in alphabetical order
>>> brendan.kill(daniel)
'Already killed'
```

Question 2

Sample Execution (a very long one):

```
general = Channel("general-helpdesk", ["Owner", "Tutor", "Student"])
announcements = Channel("announcements", ["Owner", "Tutor"])
secret = Channel("foobar", ["Owner"])

russell = Owner("Russell")
clifton = Tutor("Clifton")
aeron = Student("Aeron")
kenghwee = User("Keng Hwee")

def display_hall_of_mute():
    channels = [general, announcements, secret]
    print()
    print("HALL OF MUTE")
    for channel in channels:
        print(f"{channel.get_name()}: {channel.hall_of_mute()}")
    print()

print(russell.get_role())               # Russell is the owner!
print(clifton.get_role())               # Clifton is a Tutor
print(aeron.get_role())                 # Aeron is a Student
print(kenghwee.get_role())              # Keng Hwee has no role
print()
print(russell.join(general))            # Russell joins #general-helpdesk
print(russell.join(general))            # Russell has already joined #general-he
print(clifton.join(general))            # Clifton joins #general-helpdesk
print(russell.mute(aeron, None, None))  # Russell muted Aeron indefinitely. Reas
print(aeron.join(general))              # Aeron is muted!
                                        # therefore cannot join
print(russell.unmute(aeron))            # Russell unmuted Aeron!
print(aeron.join(general))              # Aeron joins #general-helpdesk
print(kenghwee.join(general))           # Keng Hwee has no permission to join #g
print(general.get_members())            # ['Aeron', 'Clifton', 'Russell']
print()
print(russell.join(announcements))      # Russell joins #announcements
print(clifton.join(announcements))      # Clifton joins #announcements
print(aeron.join(announcements))        # Aeron has no permission to join #annou
print(kenghwee.join(announcements))     # Keng Hwee has no permission to join #a
print(announcements.get_members())      # ['Clifton', 'Russell']
print()
print(russell.join(secret))             # Russell joins #foobar
print(clifton.join(secret))             # Clifton has no permission to join #foo
print(aeron.join(secret))               # Aeron has no permission to join #fooba
print(kenghwee.join(secret))            # Keng Hwee has no permission to join #f
print(secret.get_members())             # ['Russell']
print()
print(russell.get_channels())           # ['announcements', 'foobar', 'general-h
print(clifton.get_channels())           # ['announcements', 'general-helpdesk']
print(aeron.get_channels())             # ['general-helpdesk']
print(kenghwee.get_channels())          # []
print()

print(russell.message(announcements, "Tutorial is canceled!"))
# #announcements --- Russell: Tutorial is canceled!
print(clifton.message(general, "Hooray!"))
# #general-helpdesk --- Clifton: Hooray!
print(aeron.message(announcements, "Tutorial is canceled!"))
# Aeron has not joined #announcements
print(kenghwee.message(announcements, "Tutorial is canceled!"))
# Keng Hwee has not joined #announcements
print(russell.message(secret, "I am alone!"))
# #foobar --- Russell: I am alone!
print(clifton.message(secret, "WHAT"))
# Clifton has not joined #foobar
print()

print(russell.mute(kenghwee, None, "Testing"))
# Russell muted Keng Hwee indefinitely. Reason: Testing
print(clifton.mute(russell, 10, "Revenge"))
# Cannot mute a fellow tutor
```

```
print(kenghwee.mute(russell, 100, "Revenge"))
# Keng Hwee is not allowed to send messages!
                                                                       # becaus
print(kenghwee.message(announcements, "Tutorial is canceled!"))
# Keng Hwee has not joined #announcements
print(clifton.unmute(kenghwee))
# Clifton unmuted Keng Hwee!
print(kenghwee.mute(russell, 100, "Revenge"))
# Keng Hwee doesn't have a permission to mute another user
print(aeron.mute(clifton, 3, None))
# Aeron doesn't have a permission to mute another user
print(clifton.mute(aeron, 5, "Why would you try to mute me?"))
# Clifton muted Aeron for 5 minutes. Reason: Why would you try to mute me?
print(kenghwee.mute(kenghwee, 10, "No idea"))
# Cannot mute oneself
print(russell.mute(russell, 10, "Same here"))
# Cannot mute oneself
print(russell.mute(aeron, 3, "Spam"))
# Aeron is muted!
display_hall_of_mute()
'''
HALL OF MUTE
#general-helpdesk: ['Aeron']
#announcements: []
#foobar: []
'''

print(aeron.message(secret, "Hello"))
# Aeron has not joined #foobar
print(aeron.message(general, "Yoooo"))
# Aeron is not allowed to send messages in #general-helpdesk
print(aeron.message(announcements, "Test"))
# Aeron has not joined #announcements
print(clifton.mute(russell, None, None))
# Cannot mute a fellow tutor
print(russell.mute(clifton, None, "Muting a fellow tutor is a can"))
# Russell muted Clifton indefinitely. Reason: Muting a fellow tutor is a can
display_hall_of_mute()
'''
HALL OF MUTE
#general-helpdesk: ['Aeron', 'Clifton']
#announcements: ['Clifton']
#foobar: []
'''

print(clifton.mute(aeron, 3, "Spam?"))      # Clifton is not allowed to send mes
print(russell.unmute(clifton))              # Russell unmuted Clifton!
display_hall_of_mute()
'''
HALL OF MUTE
```

```
#general-helpdesk: ['Aeron']
#announcements: []
#foobar: []
'''

print(clifton.mute(aeron, 3, "Spam?"))      # Aeron is muted!
print(aeron.message(general, "Yoooo"))      # Aeron is not allowed to send messa
                                            # since he's muted
print(aeron.mute(kenghwee, 2, "Lol"))       # Aeron is not allowed to send messa
                                            # again, because he's still muted
print(aeron.unmute(kenghwee))               # Keng Hwee is not muted :)
print(clifton.unmute(aeron))                # Clifton unmuted Aeron!
display_hall_of_mute()
'''
HALL OF MUTE
#general-helpdesk: []
#announcements: []
#foobar: []
'''

print(kenghwee.mute(russell, 10, None))                # Keng Hwee doesn't
print(kenghwee.message(general, "Hi guys I'm unmuted"))    # Keng Hwee has not
print(russell.message(general, "Hello"))               # #general-helpdesk
print(clifton.message(general, "Hello!"))              # #general-helpdesk
print(aeron.message(general, "I'm so happy!"))         # #general-helpdesk
print(aeron.message(announcements, "Test"))            # Aeron has not join
print(kenghwee.join(general))                          # Keng Hwee has no p
print(russell.unmute(kenghwee))                        # Keng Hwee is not m
print(russell.unmute(russell))                         # Russell is not mut
```