Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page.

Name: _____

Python Review

1. Create a class called `State`. The initializer should take in an argument called `client`. The `State` should be modeled after a finite state machine. Therefore, it should have empty methods for `getEntryAction(self)`, `getExitAction(self)`, and `getAction(self)`. In addition, create the method `changeState(self, nextState)`, which transitions from one state to another.

2. Make an empty dictionary called `users`.

3. Determine if ``Bob'' exists in the `users` dictionary.

4. Given variables `name` (`Titus`) and `message` (`Hello world.`), print out the text ```<Titus> Hello world.'' using the `format` method.

5. Import the `protocol` and `reactor` modules from `twisted.internet`.

6. Import the `LineReceiver` class from the `twisted.protocols.basic` module.

Talkers

A talker is a chat system that people use to talk to each other over the Internet. The talker is a communication system that is the precursor to MMORPGs and other virtual worlds. In today's lecture, you will implement the basic functionality for talkers using the `Twisted` framework.

7. Identify the states necessary to implement a talker system.

Talker Protocol

8. You will now design a `Talker` protocol class, which extends the `LineReceiver`. The talker will have three methods: `connectionMade(self)`, `lineReceived(self, line)`, and `connectionLost(self, reason)`. Make these methods now and have them print something useful to the screen upon client connection.

9. Create a `TalkerFactory`. The `Talker` factory builds instances of protocols; in our case, the `TalkerFactory` will build `Talkers`. The class should extend `protocol.Factory`. In its initializer, you will create a dictionary of `users` to keep track of the users in the system.

10. The `reactor` is the event loop. In many ways, it is equivalent to the game loop in `Pygame`. Create a `reactor` that listens on port 1234.

11. To start the `reactor`, use the `run` method.

12. You can use `self.sendLine()` to send a message to the client. Instead of sending statements to IDLE, redirect them to the client.

13. Modify the initializer for `Talker` so that it starts in `IdleState`.

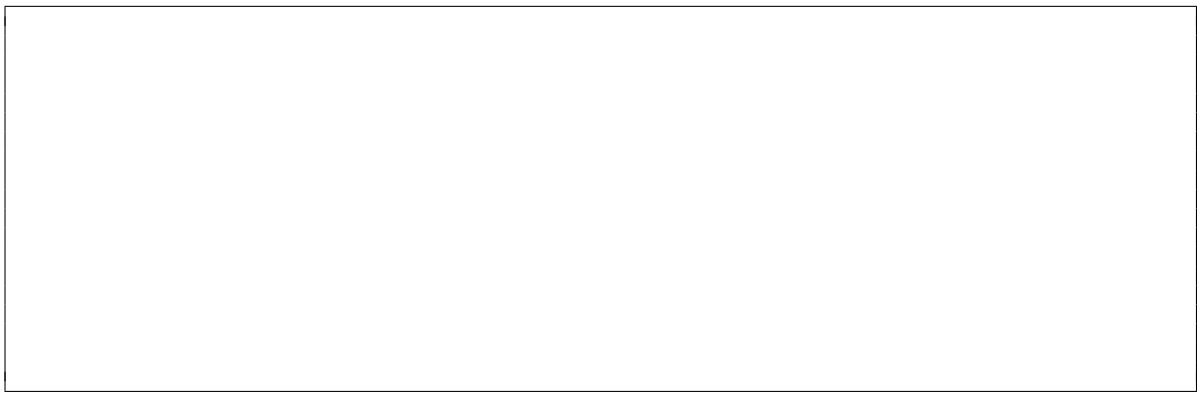
Idle State

14. You will immediately change to the `LoginState` when a connection is made. This is similar to bootstrapping the AI. The `__init__` method of `Talker` should do this initialization.

Login State

15. You will now implement the state `LoginState`. On `getEntryAction`, you should ask the client their name.

16. On `getAction`, you should determine if a user is logged in with the same name. If so, you need to retry the login state. Otherwise, add this client to the system, and change to the `ChatState`.



Chat State

17. The `ChatState` should print ``` ``You have entered the chatroom.'' ```, followed by the client's name. This should be in `getEntryAction`.

18. When receiving a line (`getAction`), you will send the line to all other clients except the sender.

19. When the connection is lost, the client should be removed from the users list.

Congratulations, you now have the basic building blocks needed for text-based MMOGs!