Making New End-to-End Functionality in the Game

1 Background

The goal of this activity is to see how information flows through the system. We are going to hijack the behavior of sending the string "Magic" in the chat. If you type "Magic" in the chat, we don't want it to behave like chat. Instead, we want the server to output "Magic received from " + playerName to the console.

2 Make a play branch

Use "Team->Switch To" to create a new local branch based off master (don't call it master!).

3 Sequence Test

Since we are changing something that passes things between two machines, the first thing we should do is define the communication with a sequence test. Create MagicSequenceTest in the sequenceTest package of the GameSequenceTests project. It must extend SequenceTest.

Here are the details of that test:

initiating command An instance of CommandChatMessageSent with a message of"Hello" with a chat type of ChatType.LOCAL

initiating server type ServerType.THIS_PLAYER_CLIENT because this communication is started by the player doing something to the client.

initiating player ID PlayersForTest.MERLIN.getPlayerId();

reset data gateways The only singleton we are using is PlayerManager, so that is all you have to reset: PlayerManager.resetSingleton();

setUpMachines We need to let the machines on both ends know that Merlin is playing:

```
@Override
public void setUpMachines() throws DatabaseException
{
        ClientModelTestUtilities.setUpThisClientsPlayerForTest(
                PlayersForTest.MERLIN);
        PlayerManager.getSingleton().addPlayer(
                PlayersForTest.MERLIN.getPlayerID());
}
```

message flow You must make an array of the MessageFlow objects that describe the communication between machines. In this case, there only one message that goes from this player's client to the area server. It should be a PlayerSendsMagicMessage with Merlin's player id as the parameter to the constructor.

constructor The constructor of a sequence test must set up the message flow and the list of servers who are involved in this sequence

```
public MagicSequenceTest()
{
        for (MessageFlow mf : sequence)
        {
                messageSequence.add(mf);
        }
        serverList.add(ServerType.THIS_PLAYER_CLIENT);
        serverList.add(ServerType.AREA_SERVER);
}
```

Do what you have to in order to make that compile. Then, obviously, it should fail. You should see something saying that the player should have sent out a message, but didn't.

4 Client Model

The command that we are going to modify is CommandChatMessageSent in GameClient. When you look at that, the first thing you should notice is that someone left commented code laying around. That sucks, so clean it up!

4.1 The Command

You are going to make this command choose between two behaviors. If they entered something other than magic, normal chat functions should happen. "Magic", you want the model to send out a new type of report: a MagicSentReport containing the player id of the player that sent the magic. Here are some clues to help:

1. ClientPlayerManager is a singleton that manages all of players. You can ask it for this client's player and get the ID from that

2. As a report, it is really just a special kind of DTO object. The test for it just needs to be sure it can return the stuff given to its constructor

Since we are following TDD, you first need to modify the tests for the command. If you look at those tests, you will see that they only tested the constructor. In some ways, this is OK because they only call one method and it is tested elsewhere, but that says the interface we are using is pretty tightly-coupled. Coding a test that the command actually works would duplicate the test for the one method being called.

Now that the command is going to do two different things, we can make the tests better. Let's make a test for the functionality we are adding. The functionality that we need is that the model send a report (call it MagicSentReport). The report is just a DTO holding the player's id number.

In order to make that report, you new to create a new test class that verifies the report's constructor's behavior. You need to add that to the test suite for the client. Note that

the tests are listed in alphabetical order by package so that it is easy to see when one is missing.

Once you have the report, you are ready to make a change to the command. Here is the test for the new behavior (this uses EasyMock which you probably haven't seen - call me over to explain!):

```
@Test
public void testMagic() throws AlreadyBoundException, NotBoundException
{
        // Make the client think Merlin is this client's player
        ClientPlayerManager.getSingleton().initiateLogin(
                PlayersForTest.MERLIN.getPlayerName(),
                PlayersForTest.MERLIN.getPlayerPassword());
        ClientPlayerManager.getSingleton().finishLogin(
                PlayersForTest.MERLIN.getPlayerID());

        //set up a mock observer that expects to our new magic report
        MagicSentReport report =
                new MagicSentReport(PlayersForTest.MERLIN.getPlayerID());
        QualifiedObserver obs =
                EasyMock.createMock(QualifiedObserver.class);
        QualifiedObservableConnector.getSingleton().
                registerObserver(obs, MagicSentReport.class);
        obs.receiveReport(EasyMock.eq(report));
        EasyMock.replay(obs);

        // make and execute the command
        CommandChatMessageSent ccms = new CommandChatMessageSent(
                new ChatTextDetails("Magic", ChatType.Local));
        ccms.execute();

        // make sure the observer got the report
        EasyMock.verify(obs);
}
```

Once you understand the test, write the code to make it pass. Note, you will need to implement the "equals" method for EasyMock to be able to compare reports. Use the pulldown Sourse-¿Generate hashcode() and equals() and tell it to compare the instance variable holding the player's id.

4.2 The Packer

Now you need to make a packer that listens for your new report and translates it into the message that you created when you made the sequence test. Make sure you write the test (and add it to the test suite) first.

In order for your test to pass you probably need an equals method here, too (you certainly will need that for the sequence tests).

Once your packer works, you can make sure that it is actually being sent by rerunning the sequence tests. They will still fail, but now they should be complaining that the server

doesn't have a handler for your new message type.

5 The Server

The only thing the server needs to do is output a string to the console when it receives the message. You can accomplish that by building the appropriate handler and putting System.out.println in it.

At this point, you should be able to verify the behavior two ways:

- You should see the output in the output of the sequence tests
- if you run the system and enter "Magic" in the local chat, you should see the output in the console for the areaserver you are connected to.

6 The Server For Real

If you were doing this for real, your handler would have to make the model do something. That would require creation of a command that the handler could use. If you want, you could make that enhancement to this implementation for practice, but it is enough to be able to explain what that would require.