Project 6

Milestone 4 Revision – CSE 4904

William Sanville
**System Integration**

The testing strategy used for integrating all modules in the JokerStars project will be the bottom-up approach. Basic pieces of functionality will be developed and integrated together. After successful integration, more functionality will be developed. In the development phase, the most critical modules in the project, the Client and the Server modules, were implemented first. The integration testing will also follow the same priority. The Client and Server modules need to be integrated first because the entire system depends on their correctness. Also, in order for almost all other modules to be tested for integration, the Client and Server modules must be functioning. For example, integration testing between the User Interface module and the Game Engine can not take place without the Client and Server module, because the interface will be running on a different physical machine than the engine; the client and server modules are needed to bridge this gap. The following diagram illustrates the interaction between the modules in the JokerStars project.
Networking Module

A critical piece of both the Client and Server modules is the Networking module. This module contains common data structures, class definitions and methods used by both the Client and Server modules. The Networking module does not depend on any other module in the project. Instead, it only depends on core .NET libraries to function. In order for this module to integrate with the Client and Server, it must simply pass the unit tests for the Networking module. The Networking module provides class definitions for packets of data sent between the Client and Server, a buffer for receiving data, and methods to translate data into a byte array for transport over sockets.

In order for this module to successfully integrate with the modules that depend on it, namely the Client and Server modules, it must simply have the proper access specifiers for class definitions and methods, so that they can be instantiated and invoked by other modules. Since the project is written in .NET, all methods and structures that need to be used by the Client and Server modules must be declared as public. In addition, the Networking module must pass unit testing, and it will be able to successfully integrate with the other modules.

Client and Server Modules

The Client and Server modules are very similar, the only difference is that the Server module accepts connections. The two modules pass and receive messages in an identical way, using the same code defined in the Networking module. In order for these modules to integrate successfully, they must agree on the same protocol for passing messages. Both modules pass and receive messages using sockets as defined in the .NET
core libraries. Information travels between the two in the form of a packet defined in the Networking module. A packet consists of an enumeration which describes the purpose of the packet, an object, which is the data that needs to be sent, and an optional table identification number, so the system can determine to which table the packet corresponds.

An important aspect of testing the interaction between the client and server is to ensure that the client can connect and communicate with the server over the Internet, not only a local network. Firewalls will need to be set properly to allow the application to work as intended. In addition to the message passing functionality, the Server module needs to perform additional duties to manage connections and determine which client sent which message.

Test Equipment: Dedicated server and at least two separate client machines

Test Procedure:

1. Start the Server module on the dedicated machine.
2. Connect to the server using the Client module from the first client machine.
   
   Verify that the server has acknowledged that a user has connected.
3. Connect to the server using the Client module from the second client machine.
   
   Verify that the server has acknowledged that a different user has connected.
4. From the server, send a different message to each client. Verify that the message was received.
5. Send a message from each client to the server. The server should display which user sent which message.
6. Disconnect the first client machine. Verify that the server recognizes that a client
has disconnected.

7. Stop the server module from listening. Verify that the remaining connected client displays that the connection to the server was lost.

**Client and Interface Modules**

The Client module provides public methods and event handlers in order to allow other modules to use its functionality. The integration between the Client and User Interface modules is exactly the same as it would be for the Client module and any other module, regardless of the purpose or the overall program. The Client module provides a Connect method, which attempts to open a connection with a specified IP address and a SendMessage method, which passes a packet to the server. As long as a valid IP address is specified and the packet that is defined in the Networking module is used, any module will be able to consume the methods of the Client module. In order to pass messages, the User Interface module will need to specify the purpose of the packet, using the enumeration defined in the Networking module and specify any .NET object. The Client module provides two public event handlers for use with the User Interface. The DataReceived event is triggered whenever the Client module has finished receiving any piece of data from the server. This allows the module that is using the Client to handle all data processing, leaving the Client completely generic. The Client also has a public event handler, ConnectionLost, which is raised when the connection to the server is interrupted. This allows the module that is using the Client to update and gracefully respond to the connection problem.

Test Equipment: Dedicated server machine and a single client.
Test Procedure:

1. Using the User Interface as the driver program for the Client module, connect to the Server module.

2. Send a message from the server to the connected client. Verify that the data was passed to the User Interface for processing via the DataReceived event.

3. Stop the server program, verify that the ConnectionLost event was raised and the User Interface handles the event properly by displaying a dialog box to the user.

Server and Game Engine

Like the Client module, the Server module is generic and the integration of the Server module with the Game Engine is exactly the same as it would be for integrating the Server with any other module. The Server module provides a StartListen method, which takes no parameters and allows the Server to begin accepting connections. Like the Client module, the Server has a public SendMessage method, which sends a packet to a given user or a list of users. The Server also provides a method to close all connections with clients, if the game needs to shut down. The Server module provides two public event handlers, DataReceived, and UserConnection. Like the Client module, the Server provides the same DataReceived event that is triggered when the server has finished receiving a piece of data. The UserConnection event is triggered when a user either connects or disconnects from the server. These events allow the Server module to remain generic, because it delegates all processing of the received data to other modules.

Test Equipment: Dedicated server machine and a single client.

Test Procedure:
1. Using the Game Engine as the driver program for the Server module, start the Server module.

2. Connect to the server from the client machine. Verify that the Game Engine has handled the UserConnection event being raised.

3. Send a message from the client to the server. Verify that the Game Engine receives the message via the Server module's DataReceived event.

**User Interface and Game Engine**

The most complex part of simulating a card game over the Internet is the interaction between the engine and the executable on the end user's machine. These two modules are tightly coupled and must agree upon the format of every piece of data that is exchanged between the two modules. The User Interface will utilize the services of the Client module to communicate with the Game Engine, which uses the Server module. Although it is easy to communicate over the Internet using the Client and Server modules, the User Interface and Game Engine must agree upon the data types that are being sent to ensure that there are no runtime errors when typecasting the data from the packets into objects that the interface or engine need to work with. The following table specifies the format that the User Interface and Game Engine must use. For example, if the User Interface receives a packet from the Client module, and the packet's type is VisitorJoined, then the User Interface will treat the data inside the packet as a VisitorJoinedEventArgs object. Failure to abide by this contract will result in runtime errors.

<table>
<thead>
<tr>
<th>Packet Type (Enumeration)</th>
<th>Object that is sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>PacketType.PlayerAction</td>
<td>PlayerAction</td>
</tr>
<tr>
<td>PacketType, ChatMessage</td>
<td>ChatMessage</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>PacketType, PlayerSitDown</td>
<td>SitDownEventArgs</td>
</tr>
<tr>
<td>PacketType, VisitorJoined</td>
<td>VisitorJoinedEventArgs</td>
</tr>
<tr>
<td>PacketType, PlayerGetUp</td>
<td>LeaveTableEventArgs</td>
</tr>
<tr>
<td>PacketType, ActionRequest</td>
<td>ActionRequestEventArgs</td>
</tr>
<tr>
<td>PacketType, PlayerNeedsUpdate</td>
<td>Player</td>
</tr>
<tr>
<td>PacketType, CardsDealt</td>
<td>CardsDealtEventArgs</td>
</tr>
<tr>
<td>PacketType, BoardUpdate</td>
<td>BoardUpdateEventArgs</td>
</tr>
<tr>
<td>PacketType, ViewState</td>
<td>ViewState</td>
</tr>
<tr>
<td>PacketType, Showdown</td>
<td>ShowdownEventArgs</td>
</tr>
<tr>
<td>PacketType, TableList</td>
<td>GameTableList</td>
</tr>
</tbody>
</table>

In order to thoroughly test the Game Engine and the User Interface, the entire system will need to be operating to some degree. Therefore, testing of the Game Engine and User Interface modules will have to wait until System Testing is performed.

**Game Entities Module**

The Game Entities module contains class definitions for all object in a card game, such as cards, chips, players, and more. The Game Entities module does not depend on any other assembly in the JokerStars project, it relies only on core .NET libraries to function. In order for this module to integrate with the modules that use it, primarily the Game Engine and the User Interface, each class must have the correct access specifiers. In other words, the classes in the Game Entities module must be declared as public in order to integrate with the other modules in the JokerStars project. The Game Entities must also pass unit testing to successfully integrate with the other modules.

**SQL Module**

The purpose of the SQL module is to perform actions that interact with the
SQL database that runs on the game server machine. The SQL module relies only on the class definitions defined Game Entities module. The SQL module provides static methods for updating or retrieving data from the database, such as the amount of money a user has in his or her account.

Test Equipment: Server machine

Test Procedure:

1. Log into the database server. Access the ChipCount table and record the chip count for “user1.”

2. From the Game Engine, call the IncreaseChipCount method with the parameters “user1” and 100. Log into the database server and verify that the chip count has increased by 100.

3. Call the DecreaseChipCount method from the Game Engine with the same parameters. Verify that the user's chip count has returned to the original value.

Collections Module

The Collections module contains definitions for custom data structures and utilities to sort the collections. This module does not depend on any other module in the project, it relies only on core .NET libraries. In order for this module to successfully integrate with the other modules in the project, it must only pass unit testing to ensure that it works properly. Once that is verified, any other module in the project will be able to use the data structures defined in the Collections assembly.

Events Module
The Events module is used by the Game Engine and the User Interface. In most cases, data structures defined in the Events module are passed between the engine and the many clients that are connected. The purpose of this module is encompass all the details of any event in the game and package them into a single class for transport over the Internet. This module mimics the style of core .NET event arguments, such as the data structure that is passed when a button click event has been raised. In order for this module to integrate with the Game Engine and the User Interface, all of the classes defined in the module must be public and the accessor methods of each class must be public as well. The only testing for this module should be unit testing.

**Exit Criteria**

The JokerStars application will be ready for System Testing once the following core system features are verified in integration testing. The system must be able to transport complex classes over the Internet to clients both on campus and off campus. This is the foundation of the application, as it is a multi-player application that is distributed. Once the Client, Server, and Networking modules pass this test, the specific features of a poker game must pass integration testing. The SQL module must successfully authenticate user names and passwords with the database server in order to allow the system to recognize users and retrieve the amount of money the user has. The Game Engine must have the capability to broadcast the game tables that are active on the server, and the User Interface should process this data and allow the end user to select which table he or she wishes to play at. Once selecting a game table, the User Interface should display all graphical elements of a card game, and provide buttons and other controls for the user to provide input to play the game.