ROCHESTER INTITUTE OF TECHNOLOGY – DEPARTMENT OF COMPUTER SCIENCE

Risk

A Networked Application

Team Name: The Fighting Mongooses
Trevor Mack
Dylan Hall

November 4th, 2009

Description of Game

Risk is a turn-based board game for 2 or more players. The game board is a predefined grid of territories. At the beginning of the game, each territory is randomly assigned to a player and a random-sized army is put onto it. During a player's turn, he or she may launch an attack from a territory he or she controls to an adjacent territory which is owned by another player. The outcome of the territory is determined by rolling dice. The attacking territory rolls one fewer dice than the number of armies on that territory. The defending territory rolls the same number of dice as armies on that territory. The territory that rolls a higher number wins the battle. In the event of a tie, the defending territory wins the battle. If the attack is successful, the attacker gains control of the defending territory and all but one of the armies from the attacking territory are moved into it. If the attack is unsuccessful, all but one of the armies of the attacking territory are lost. A player may choose to end his turn at any time. Once a player ends his turn, his or her territories are randomly fortified. Fortification involves the addition of one army per two territories the player controls. The game is over once one player controls the entire world map, and that player is declared the winner.

Message Description and Encoding

Network messages in the java project, Risk, are encoded and constructed using the DataOutputStream Object in the Java 1.5.0 or later specification. Each argument listed alongside the below message types will be written to the DataOutputStream one at a time using the given argument type, [boolean, int, byte, UTF, etc...].

Server to Client Message Encoding

Message Opcode (First Byte of Message)	Arguments	Argument Datatype
Constants.GAME_STARTING	- none -	- none -
Constants.TURN_IND	player index	Int
Constants.ATTACK_MADE	Source territory index Destination territory index Attack die roll Defend die roll	Int Int Int Int
Constants.TERRITORY_STATUS	Territory index Owner index Army count	Int Int Int
Constants.PLAYER_INFO	Player index Color number Player name	Int Int Java Modified UTF-8
Constants.WHO_AM_I	Player index	Int
Constants.GAME_FINISHED	Winning player index	Int
Constants.GAME_ALREADY_STARTED	Game status	Boolean

Client to Server Message Encoding

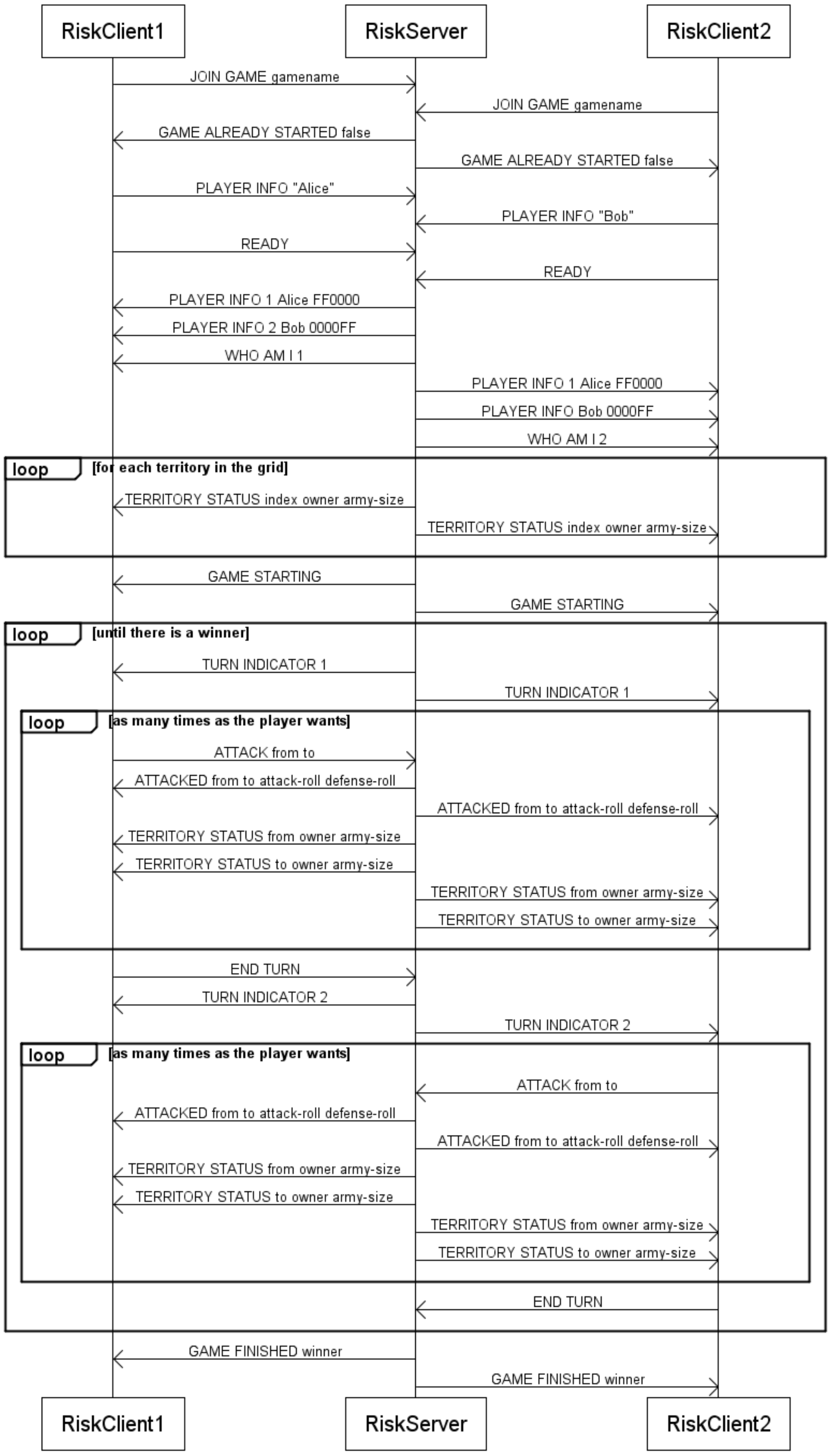
Message Opcode (First Byte of Message)	Arguments	Argument Datatype
Constants.ATTACK	Source territory index	Int

	Destination territory index	Int
Constants.SURRENDER	- none -	- none -
Constants.END_TURN	- none -	- none -
Constants.GAME_TO_JOIN	Game name	Java Modified UTF-8
Constants.PLAYER_INFO	Player name	Java Modified UTF-8

Example Message Encoding

```
//opcode associated with this message
outputstream.writeByte(Constants.TURN_IND);
//argument outlined for this message type, in this case (current player's index number)
outputstream.writeInt(0);
```

Sequence Diagram

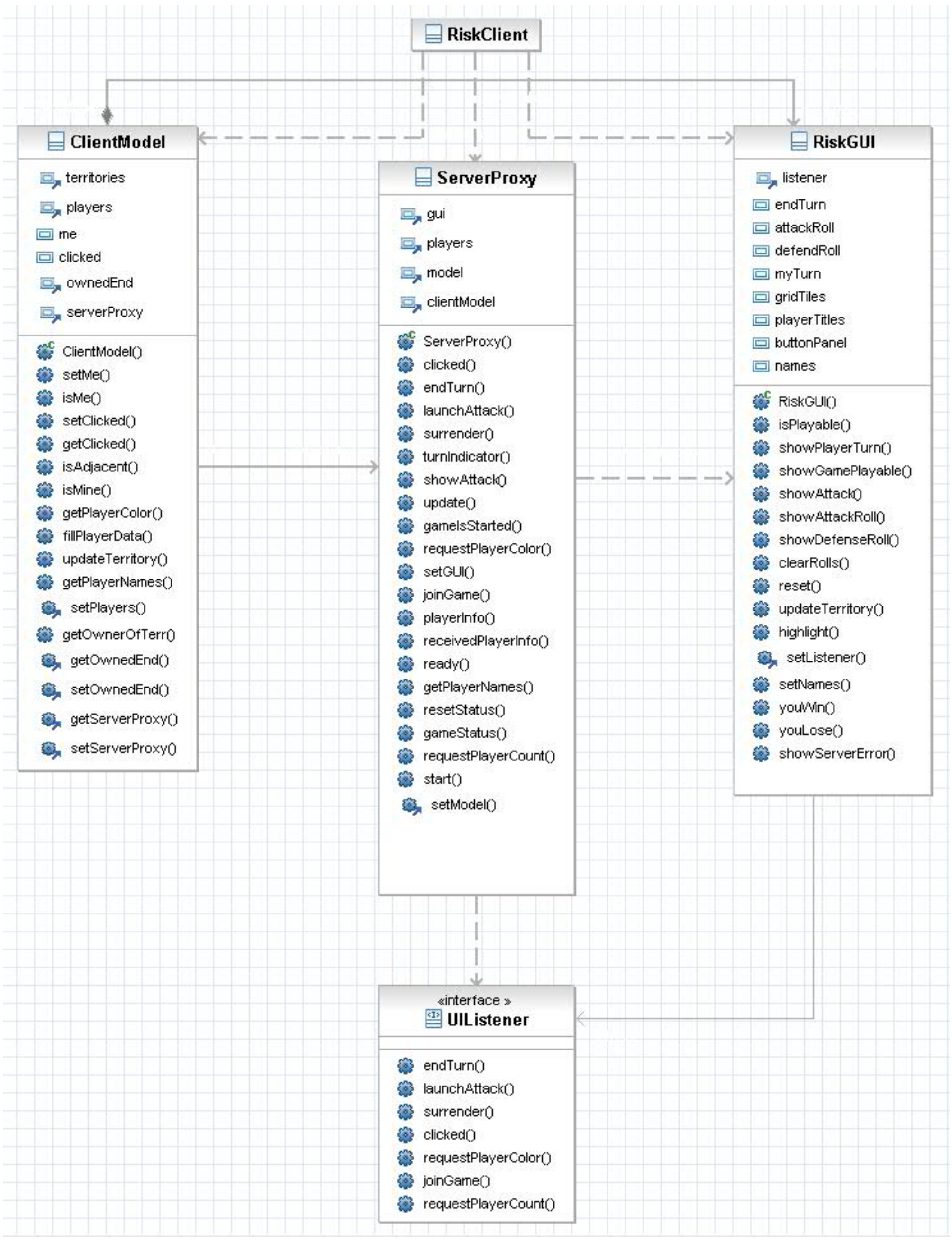

RiskClient1
RiskServer
RiskClient2
JOIN GAME gamename
JOIN GAME gamename
GAME ALREADY STARTED false
GAME ALREADY STARTED false
PLAYER INFO "Alice"
PLAYER INFO "Bob"
READY
READY
PLAYER INFO 1 Alice FF0000
PLAYER INFO 2 Bob 0000FF
WHO AM I 1
PLAYER INFO 1 Alice FF0000
PLAYER INFO Bob 0000FF
WHO AM I 2
loop
[for each territory in the grid]
TERRITORY STATUS index owner army-size
TERRITORY STATUS index owner army-size
GAME STARTING
GAME STARTING
loop
[until there is a winner]
TURN INDICATOR 1
TURN INDICATOR 1
loop
[as many times as the player wants]
ATTACK from to
ATTACKED from to attack-roll defense-roll
ATTACKED from to attack-roll defense-roll
TERRITORY STATUS from owner army-size
TERRITORY STATUS to owner army-size
TERRITORY STATUS from owner army-size
TERRITORY STATUS to owner army-size
END TURN
TURN INDICATOR 2
TURN INDICATOR 2
loop
[as many times as the player wants]
ATTACK from to
ATTACKED from to attack-roll defense-roll
ATTACKED from to attack-roll defense-roll
TERRITORY STATUS from owner army-size
TERRITORY STATUS to owner army-size
TERRITORY STATUS from owner army-size
TERRITORY STATUS to owner army-size
END TURN
GAME FINISHED winner
GAME FINISHED winner
RiskClient1
RiskServer
RiskClient2

Client Program Design

Description

The RiskClient class is the main class for the client part of the program. This class contains the connection to the server model via the ServerProxy. This ServerProxy is the link to the server-side model and exists as the client side ModelListener and implements the client side UIListener. We choose to represent a game using a ClientModel object so that there is less network traffic to handle the UI logic. For more details reference the below UML Diagram of the Server.

UML Diagram

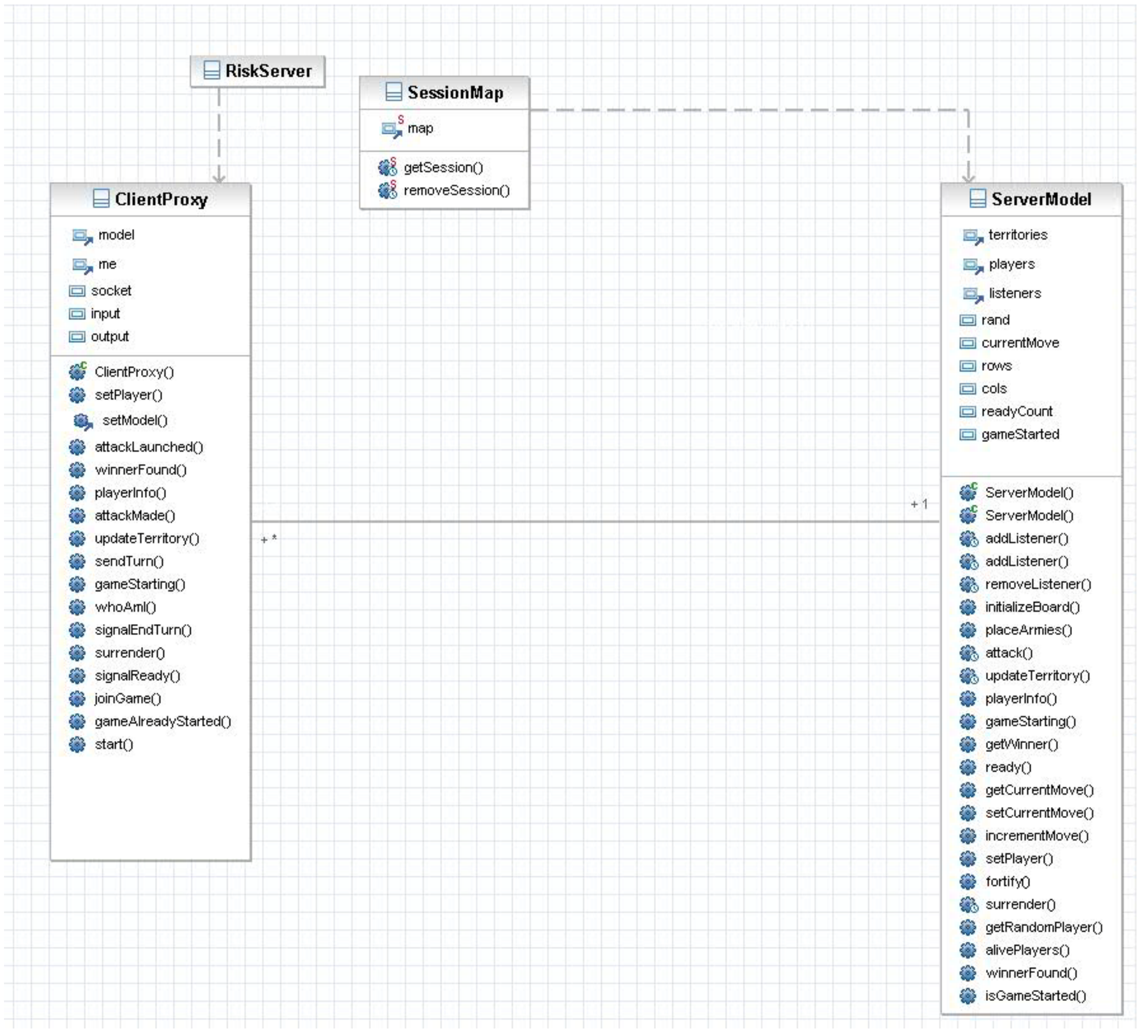


Server Program Design

Description

The RiskServer class is the main class for the server part of the program. This class contains the connection to the client model via the ClientProxy. This ClientProxy is the link to the client-side model and contains the network handling implementation. Also to handle multiple sessions simultaneously there is a static class known as SessionMap which handle the logic of mapping a game name to an actual ServerModel. This is declared as static because it is considered a Singleton. For more details reference the below UML Diagram of the Server.

UML Diagram



Developer's Manual

Compiling and running the Risk game is very simple. First, you must extract the java files from Risk.jar using the following command:

```
jar xvf Risk.jar
```

To compile the java files, run the following command:

```
javac *.java
```

Note that Risk requires at least Java 1.5.0 be installed. Previous versions of Java are not supported.

To run the server, simply run the following command:

```
java RiskServer
```

No command line arguments are necessary, and any provided will be ignored. The server always runs on port 1988.

To run a client, simply run the following command:

```
java RiskClient
```

No command line arguments are necessary, and any provided will be ignored. The client will always attempt to run on port 1988. Once the client is running, dialog boxes ask for your username, the host name, and the game to connect to before displaying the GUI. After a valid game name is given to the GUI the next DialogBox asks to see if the user is ready to begin play. Note that the game board will not launch until all users that joined that given game name are considered ready to play by the server.

User’s Manual

To run the Risk game, first make sure all the files are compiled. To do this, please see the Developer Manual.

To play a game of Risk, there must be a server running either on your local computer, or elsewhere. To start a server on your computer, run the following command in the folder where the Risk files are located:

```
java RiskServer
```

To start a game, run the following command in the folder where the Risk files are located:

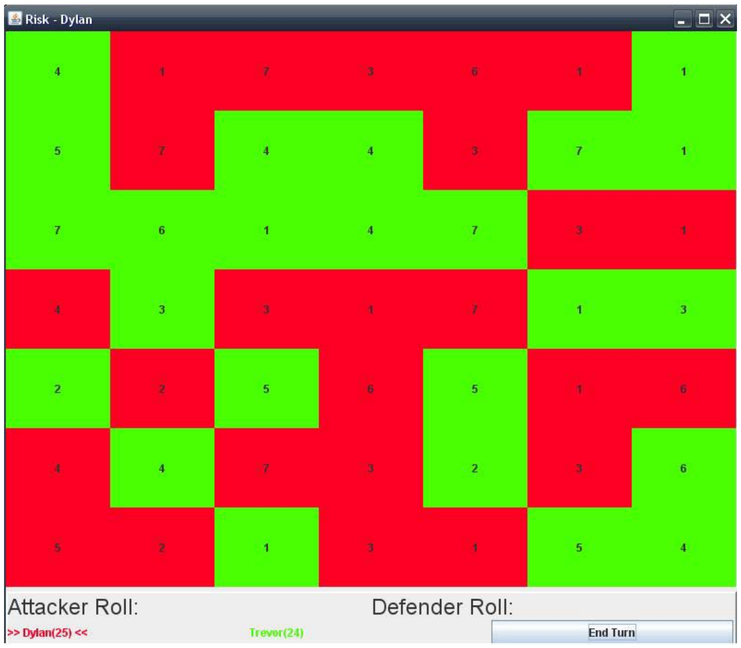
```
java RiskClient
```

After running this command, a dialog box will appear and ask for your name. This will be your username when playing the game. After this a second dialog box will appear and ask for the address of the Risk Server. If the server is running on your computer, leave the value as localhost, otherwise enter the location of the Risk server. Finally a third dialog box will appear asking for the name of the game to join. If the game with this name is already in session, it will ask for another. Clicking Cancel on any of the dialog boxes will exit the program.

Once you have selected a username, server, and game name, you will be presented with the prompt "Are you ready?" Click yes when all the players you want to play with are in the game. Clicking no will simply result in the prompt being displayed again. Clicking cancel will exit the program.

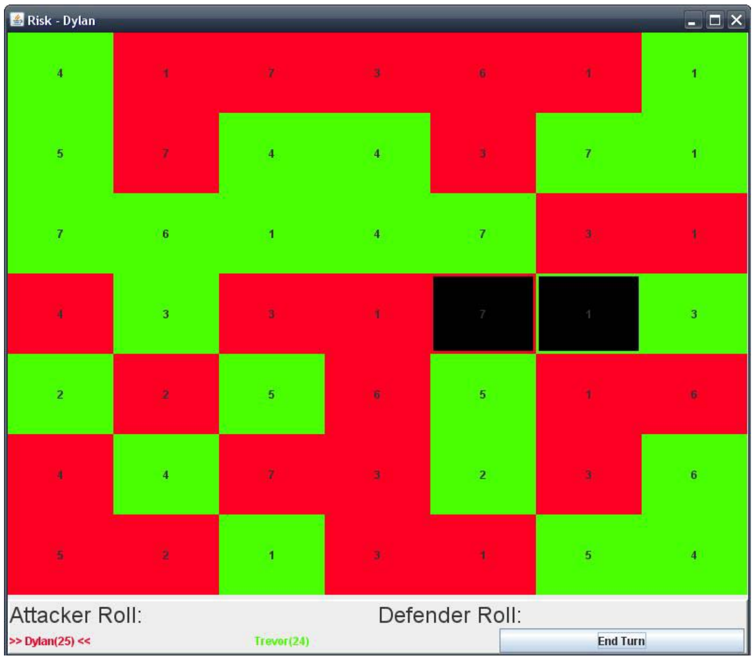
The game will not start until at least 2 players have joined and indicated that they are ready.

When the game starts, you will see a screen like the following:



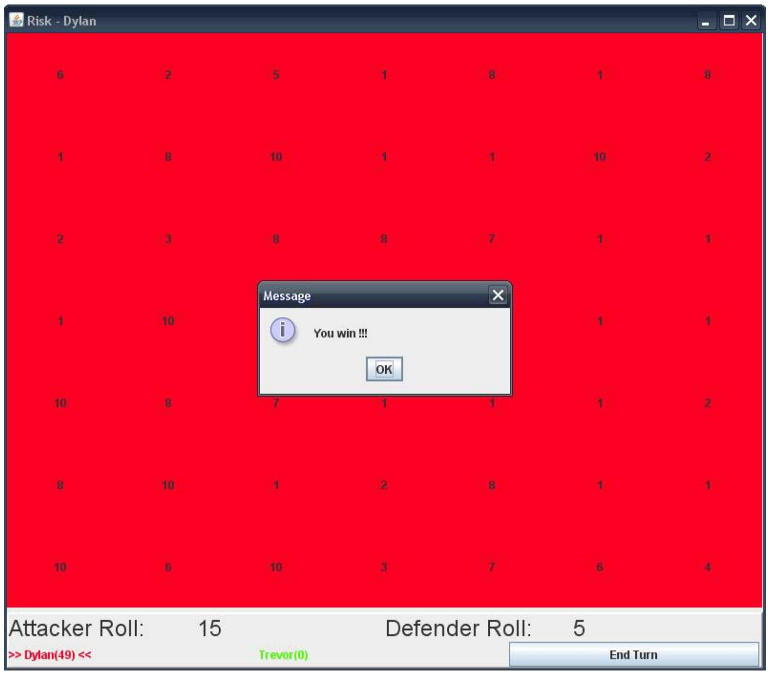
The main grid is the representation of the game map. Each tile is a territory that is owned by a player. The number in the territory is the size of the army in that territory. Below the grid is a row of space to display the attack roll and defense roll. At the bottom is a row of names of players in their color with the number of territories they control in parentheses. The current player is indicated with his name surrounded by “>> <<”. In the bottom right corner is the End Turn button. This button is disabled until it is your turn.

To play the game when it is your turn, first click on one of your territories, then on a neighboring enemy territory. The two territories will be highlighted and the rolls will be displayed.



Note that the rolls are calculated by a number of dice rolls, where the number of dice is the army size. If your attack was successful, the enemy territory will become yours, and all but one of your armies will move into that territory. (The one remaining army stays in the attacking territory.) Note that this means the attack roll uses one fewer die to roll than the army size. If the attack was unsuccessful, the enemy territory is unchanged, and all but one army is removed from the attacking territory. You are allowed to attack as many times as you are able in one turn. When you can make no more moves, click the End Turn button to end your turn.

The game is over when one player controls the entire game board.



Message

You win !!!

Attacker Roll: 15 Defender Roll: 5

>> Dylan(49) << Trevor(0)

What did we learn?

Through this project we learned some more intermediate strategies that are encountered in the creation of a networked application in java. We also expanded our knowledge of the MVC (Model-View-Controller paradigm). We discovered practical ways to implement and use a multi-user, multi-session server environment. Finally, used the material delivered in class lectures to design and implement network messages to be used for this project.