Abalone

Team members

Zachary Potter, Channon Price

Game description

Abalone is a two person board game played with black and white marbles on a hexagonal grid. The object of the game is to knock six of the opposing player's pieces off the board. This is accomplished by moving pieces in-line (on the same row or column) or broadside (diagonally) one space. Each turn, a player may move from one to three pieces together either in-line or broadside. To move the opposing player's pieces, a player must make an in-line move while adjacent to the opponent's piece. However, there must be fewer pieces being moved than are being used in the in-line move. For instance, two black pieces can move one white piece, but not two white pieces.

Description of the messages

RequestJoin
Sent by the client to the server requesting to join a game. If the requested game doesn't exist, it will be created. A ResponseJoined will be sent back.

RequestLeave
Notifies the server that the client is leaving before disconnecting.

RequestMove
Sends a move, requesting to make it.

ResponseBoardUpdate
Sent by the server to notify the client that the board has been updated. Sends the board, and the color of the player who's turn it is.

ResponseJoined
Sent by the server to notify the client that they have either joined or failed to join the game.

ResponseLeftGame
Sent by the server to notify the client that they have been kicked from the game, or the game has ended.

ResponseMoveRejected
Sent by the server to notify the client that their last move was rejected.

Description of the Message Encoding

Messages are passed between client and server via message objects. Each event, such as a player joining or leaving a game, results in an object with all of the necessary data to handle the event being passed to the client or server. This technique works well for large data sets, like sending an entire board worth of pieces, or highly varied data sets. However, it is less efficient for small messages that contain little or no packaged data in the object.

Description of the client program design

The client program consists of three main classes: the modelProxy, playerView, and the abaloneClient.

The modelProxy is responsible for listening for server messages and calling the appropriate methods in the playerView object. It also sends messages back to the server in response to server messages, or when the user selects a move.

The playerView is responsible for displaying the game to the user and for taking user input for moves. It stores a local copy of the current game.

The abaloneClient class is responsible entirely for setting up a connection to the game server and launching the modelProxy and playerView objects. It interfaces with the user via command line arguments.

Description of the server program design

The server program consists of four main classes: the sessionManager, viewProxy, abaloneModel, and abaloneServer.

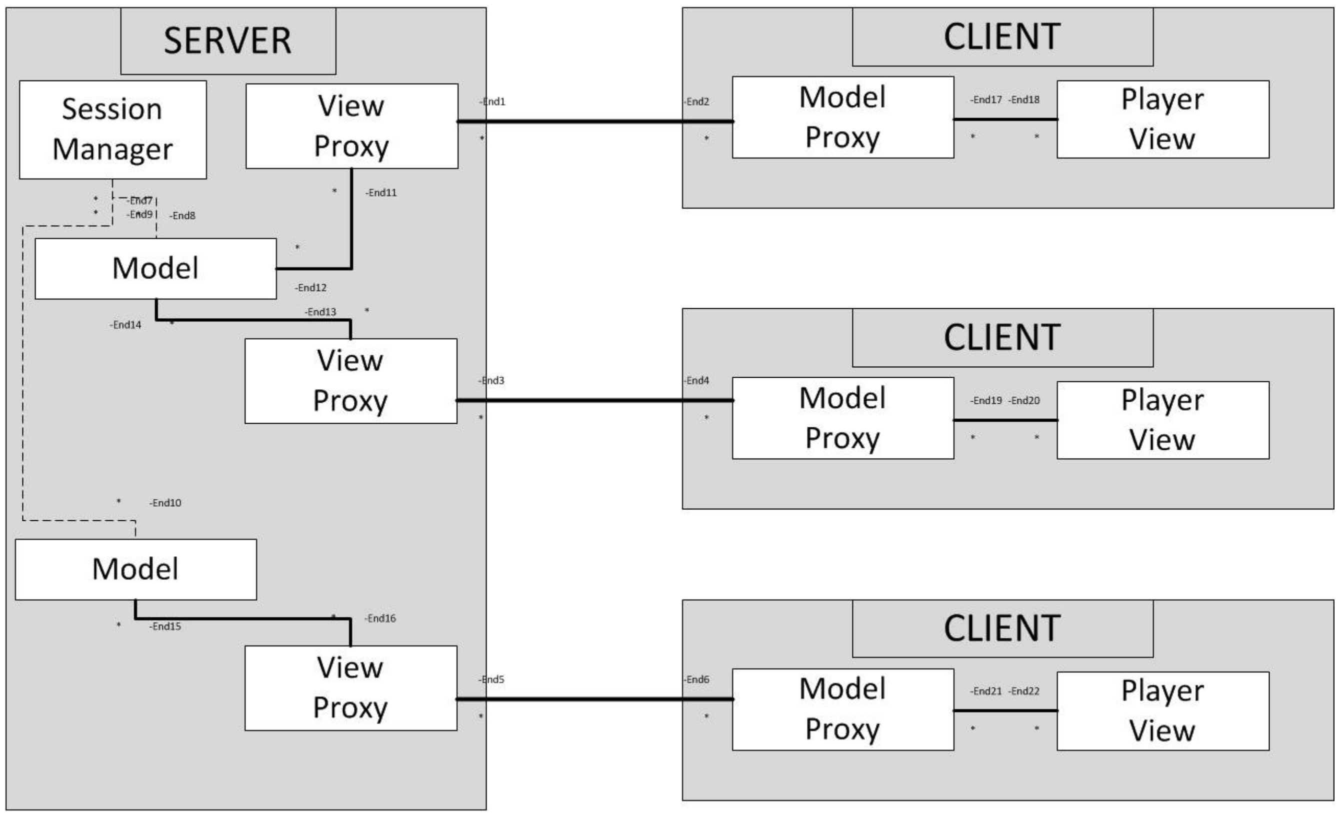
The sessionManager creates a new model for each session, and is responsible for connecting clients to their session's model.

The viewProxy is responsible for listening for client messages and calling the appropriate methods in the abaloneModel object. It also sends messages back to the client messages, or when something is modified in the model.

The abaloneModel is the server side copy of the Abalone game. It is responsible for checking to see if moves are valid, and moving pieces on the board. It keeps score, and determines if the game has finished.

The abaloneServer class is responsible entirely for listening for connections from the game client and setting up a connection to the client and then launching the viewProxy, playerView objects. It interfaces with the user via command line arguments.

UML



<u>Developer's manual</u>

Compiling

Enter the Abalone root directory.

To compile the source files:

```
mkdir bin
javac -d bin src/edu/rit/datacom/abalone/client/* src/edu/rit/datacom/abalone/common/*
src/edu/rit/datacom/abalone/server/*
```

To create the JAR files:

Client:

```
jar cfm AbaloneClient.jar client.mf -C bin edu/rit/datacom/abalone/client -C bin
edu/rit/datacom/abalone/common
```

Server:

```
jar cfm AbaloneServer.jar server.mf -C bin edu/rit/datacom/abalone/server -C bin
edu/rit/datacom/abalone/common
```

<u>User's manual</u>

To launch the game as the server, run the command:
java -jar AbaloneServer.jar <host> <port>

Once the game server has been launched, launch the game client with the command:
java -jar AbaloneClient.jar <host> <port> <game-name>

where <host> is the ip of the server, and <port> is the port chosen when starting the server. The <game-name> is the name of the session you would like to join or create.

Once the game is launched and connected to, the interfaces is straightforward. Once it is your turn, observe the current board. The O's are black pieces, and the @'s are white pieces. The board is labeled on the left side with numbers representing the Y values and labeled on the bottom representing the X values. To move a piece, input the X location of the piece, and then hit enter. Then enter the Y location of the piece. If you wish to move more pieces, enter up three pieces locations or else type d and hit enter to select only the pieces entered so far. Once all of the pieces have been selected, select the number corresponding to the direction you wish to move your pieces. If the move was legal, it is now your opponent's turn. If the move was illegal, you must repeat the process of selection pieces and a direction again.

Discussion of what you learned

We learned about using serializable objects as messages in an ObjectStream. We also learned about the challenges of using a session manager to keep track of multiple sessions, each of which has multiple clients.

What each individual team member did

Zach wrote the Abalone game engine, including the game board, moves, and user input and output. Channon wrote the network interface, including the model and view proxies and listeners. We collaborated heavily on the design of the game to network interface.