

Pontoon Distributed System

By 100136054

Pontoon Rules

A simplified version of the card game of “Pontoon” (also known as “twenty-one” or “Black-Jack”) is as follows: The aim of the game is to collect a hand of cards that sum to a value as close as possible to, but not exceeding 21. Face cards (the jack, queen and king) have a value of ten, other cards have their normal face value, except the ace can count as either 1 or 11 at the players discretion. A hand that has a minimum value greater than 21 is said to be “bust” and is worth nothing. A hand of 5 cards with a value of 21 or less is known as a “five card trick” and is worth more than any hand other than a “Pontoon”, a hand consisting of an ace and a face card.

Base classes and interfaces

In socket programming, the client and the server exchange information by messages. For the sake of simplifying the processes, I designed a very simple system which send and receive one message a time. The following classes and interfaces explain the idea in details.

Speaker interface

Due to the nature of socket programming, I designed the Speaker interface. In a socket program, the client and the server exchange information by messages. To simplify the process, I let the client and the server send only one message at a time. Thus, I designed the Speaker interface, which response to one coming message by a reply message. The communication between the client and the server are controlled by two speakers, one on each side.

In this particular pontoon game, the two speakers are the dealer and the player. The player firstly says “start” and the dealer says “your first two cards are ...”. The player then says “twist” or “stick”. The dealer tells the cards of the player every time and gives the result of the game at last.

This interface can also be generalised to other purposes. A new speaker class can be generated by implement another card game.

SpeakingStrategy interface

The SpeakingStrategy interface can also be called the “language” interface. It was named SpeakingStrategy because I used strategy design pattern. From the design point of view, this name is clearer. The SpeakingStrategy translates the objects to strings. The strings can be represented by different languages. Those languages include human language and machine language. I implemented one of each of them.

PlainSpeakingStrategy Class

The PlainSpeakingStrategy class speaks the human language. It gives information like “you hand is Ace of Hearts, Five of Clubs”. It is readable by any human player.

JsonSpeakingStrategy Class

The JsonSpeakingStrategy class speaks messages with JSON format. Its messages look like

```
{GameResult:Continue;Card1:Ace of Hearts;Card2:Five of Clubs;}
```

The SpeakingStrategy is expendable. In the future, new formats or protocols can be added easily. For example, the XML messages can be added easily.

Messages

In a socket system, the client and the server exchange information by messages. Thus it is important that the two generate messages according to the same protocol. The following table explains the process.

	Message	Client/Player	Server/Dealer
1	GameType	Duel/Melee	(Repeat the same)
2	MessageType	Plain/Json/XML	(Repeat the same)
3	Start	Start	(Gives the first two cards)
4	Twist	Twist	(Gives 1 more cards, end the game if player bust)
5	Stick	Stick	(Gives game result)
6	START/QUIT	START/QUIT	(Gives the first two cards) or close connection

In step 1, the player tells the dealer the game type. For the time being, it is either “DUEL” or “MELEE”. “DUEL” means the player requests a duel with the dealer, “MELEE” means the player request a game with other players, the dealer’s duty is simply giving cards. In the system, the message type can only be Plain or JSON. When the server has received this message, the SpeakingStrategy property of the speaker will be assigned with a concrete object which translates information to Plain English or JSON strings. The third message tells the server/dealer that the client/player is ready. The server will then send the message containing the information of the first two cards. Step 4 is repeated from 0 to 3 times. The player sends the twist message to the dealer. The dealer then calculates the result of the game. If the overall score is less than 21, the game continues and the player decides whether to send the twist message again. If the overall score exceeds 21, the game ends immediately. Step 5 is skipped. In step 5, the client sends the message “stick” and the server simply sends the result of the game to the client. In the last step, the player send the start/quit message indicating whether to restart a game or sever the connection.

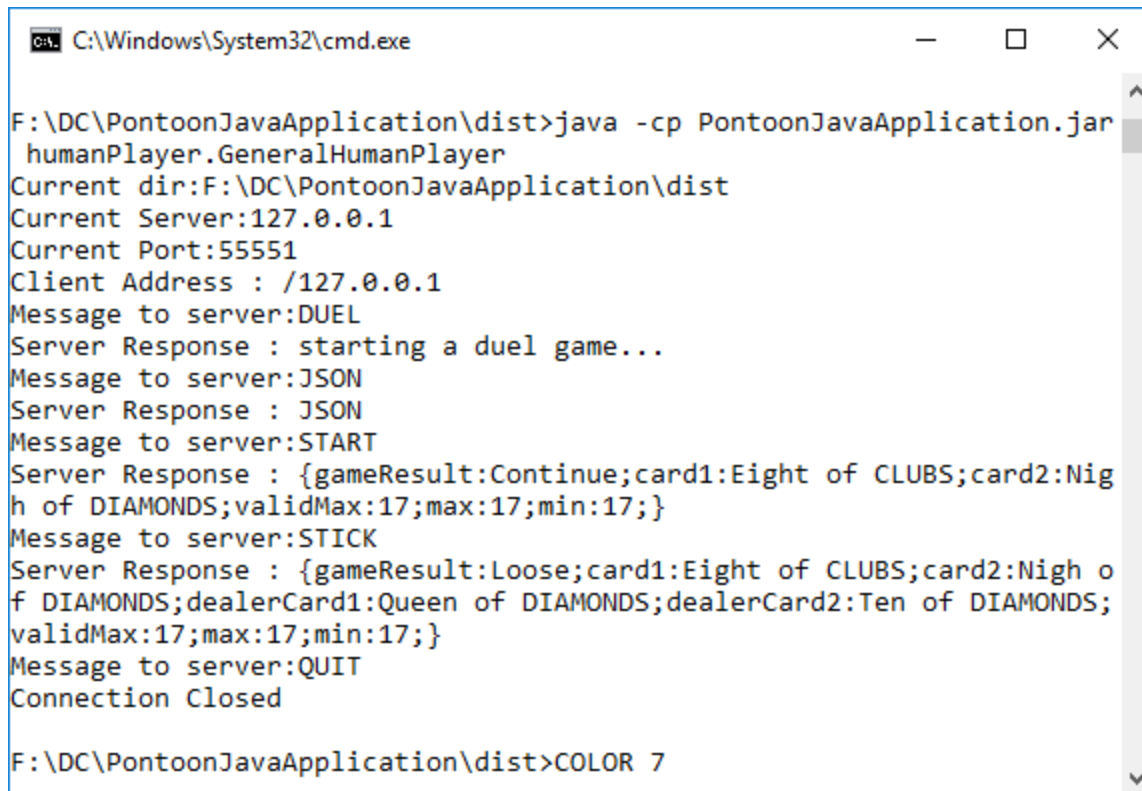
The following screenshots demonstrate the messages between the client and the server. The first message is DUEL, indicating a duel game. The second message is PLAIN, that is why the following messages are in human readable format. The third message is START, so the game starts. The fourth message is TWIST, because the player got only 8 points and it is safe to twist. The fifth message is STICK, because the player has already got 18 points and it is dangerous to ask for more. The server tells the

player the dealer has got 20 points and the player loses the game. The last message is QUIT. After that, the connection is closed.

```
C:\Windows\System32\cmd.exe
F:\DC\PontoonJavaApplication\dist>java -cp PontoonJavaApplication.jar
humanPlayer.GeneralHumanPlayer
Current dir:F:\DC\PontoonJavaApplication\dist
Current Server:127.0.0.1
Current Port:55551
Client Address : /127.0.0.1
Message to server:DUEL
Server Response : starting a duel game...
Message to server:PLAIN
Server Response : PLAIN
Message to server:START
Server Response : Your hand is :
Three of HEARTS
Five of SPADES
(value:8)
Twist or Stick?T|S
Message to server:TWIST
Server Response : Your hand is :
Three of HEARTS
Five of SPADES
Ten of DIAMONDS
(value:18)
Twist or Stick?T|S
Message to server:STICK
Server Response : Game Result:Loose
Your hand is :
Three of HEARTS
Five of SPADES
Ten of DIAMONDS
(value:18)
Dealer hand is:
Jack of SPADES
King of SPADES
(value:20)
Type start to start/restart the game.
Type quit to quit the game.
Message to server:QUIT
Connection Closed

F:\DC\PontoonJavaApplication\dist>COLOR 2
```

The following is another example with JSON messages:



```
C:\Windows\System32\cmd.exe

F:\DC\PontoonJavaApplication\dist>java -cp PontoonJavaApplication.jar
humanPlayer.GeneralHumanPlayer
Current dir:F:\DC\PontoonJavaApplication\dist
Current Server:127.0.0.1
Current Port:55551
Client Address : /127.0.0.1
Message to server:DUEL
Server Response : starting a duel game...
Message to server:JSON
Server Response : JSON
Message to server:START
Server Response : {gameResult:Continue;card1:Eight of CLUBS;card2:Nigh
h of DIAMONDS;validMax:17;max:17;min:17;}
Message to server:STICK
Server Response : {gameResult:Loose;card1:Eight of CLUBS;card2:Nigh o
f DIAMONDS;dealerCard1:Queen of DIAMONDS;dealerCard2:Ten of DIAMONDS;
validMax:17;max:17;min:17;}
Message to server:QUIT
Connection Closed

F:\DC\PontoonJavaApplication\dist>COLOR 7
```

Dealers - Game Types

A dealer class is simply a concrete class of the Speaker interface. A dealer class defines a set of rules. Adding a new dealer class is equal to adding a new game.

In this program, two dealers are implemented, representing two kinds of games. The first is the duel dealer, which regulates a one-vs.-one, client-vs.-server game. The server is a machine but the client can be either, because the server only deals with messages and it does not care whether a machine sends the message or a human.

The second dealer is the melee dealer, which hosts a many-vs.-many game. In the game, several clients are connected to the server. Again, the server is a machine but it does not know each of the client is a machine nor a human. It cares only the messages.

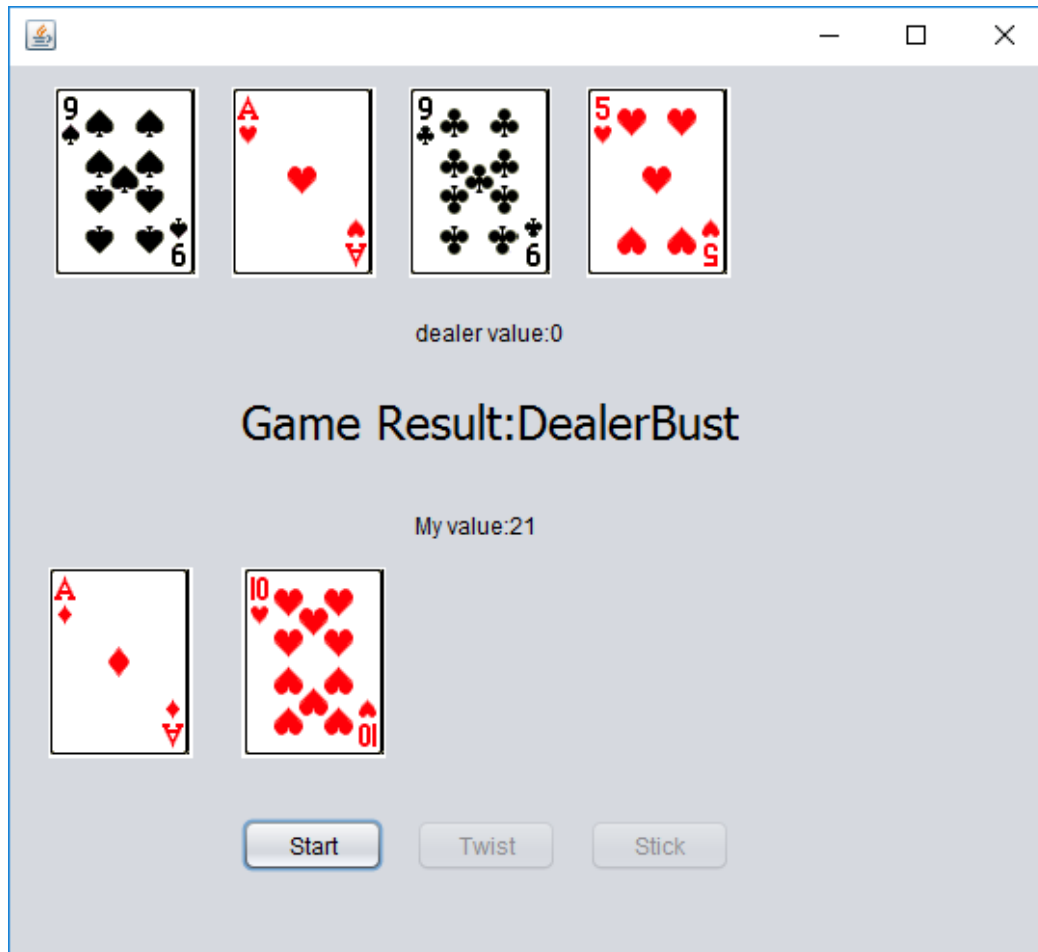
Server

The server does care whether the client is a human or a machine. What the server does is to simply create a thread for each player and plays with each of them at the same time.

Human Player

The DOC client in previous section can be regarded as a human client. However, it would be very inconvenient for the player to use a DOS window to play the game. Thus, I created the following GUI. Beneath the GUI, the client simply sends and receives JSON messages. However, this client will translate

the JSON messages and rebuild the objects. After that, the java window can create images via these objects.

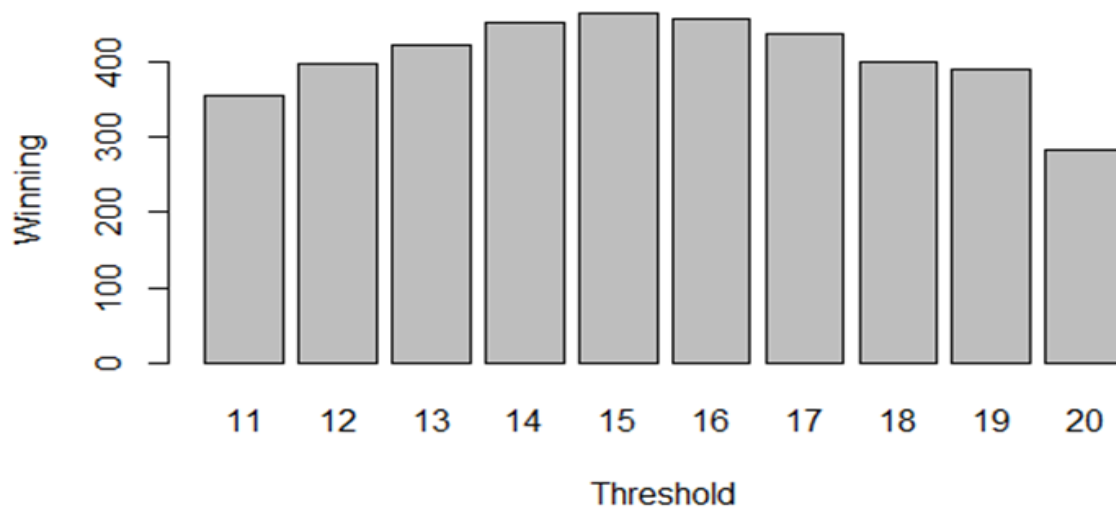


Machine Player

In order to find the best strategy in a one-vs.-one game, I created a machine player. This machine player plays the pontoon game again and again with different threshold, each repeats the same times. The mechanism is very simple. The machine client will set a threshold. Once the points exceed the threshold, the player will stick to it. By doing this, sufficient data has been collected which can then be used in statistical methods in searching for a good strategy.

Analysis of log file

I used threshold from 11 to 20, each 1000 times. The best strategy/threshold is 15, the winning rate is about 47%. Statistically, the dealer is always the winner at last.



Multi-Player

In the multi-player game, the server waits until there are enough players. Once, each of the players has sent the start message to the server, the server starts to deal cards to players one by one.

Each of the players is created by the class called player generator and each of them is a separate thread.

Analysis of log file

The following plot created in R Studio shows the best strategy of pontoon in multi-player game. I created 10 players with threshold from 11 to 20. They sit in the same table and play the game together. Each of them will decide whether to stick or twist independently, regardless of the valid highest points of other players. We can see that with winning rate increases with the threshold and the best strategy is to set the threshold at 20.

However, in a real game, a great number of factors have to be considered, such as the cards used and left, the highest points on the table and etc. the following plot is only suggestive.

MultiPlayer Threshold and winning(in 10000 games)

