35

A Poker Simulation Application

35.1 Introduction

This final application is possibly less serious than the other applications in the book, but it should be interesting to poker players, and it contains some interesting VBA code. In case you are not a poker player, a player is dealt five cards from a 52-card deck. There are several types of hands the player can be dealt, as described in the following list:

- A pair: two of some denomination and three of other distinct denominations
- Two pairs: two of one denomination, two of another denomination, and another card
- Three of a kind: three of one denomination and two of other distinct denominations
- A straight: five denominations in progression, such as 4, 5, 6, 7, 8
- A flush: five cards of the same suit, such as five hearts
- A full house: three of one denomination and two of another denomination
- Four of a kind: four of one denomination and another card
- Straight flush: a straight, all of the same suit
- A bust: none of the above

Except for a bust, the hands in this list are shown in increasing value. For example, three of a kind beats two pairs, and they all beat a bust.

The application simulates 100,000 five-card hands, all from a "well-shuffled" 52-card deck, and counts the number of each type of hand in the above list. It is interesting to see whether the probabilities of the various hands go in the opposite order of their values. For example, is two pairs more likely than three of a kind? The simulation will help answer this question.

New Learning Objectives: VBA

- To illustrate how VBA can perform a simulation completely with code—no spreadsheet model.
- To illustrate how rather complex logic can be accomplished with the use of appropriate `If` constructs, loops, and arrays.

New Learning Objectives: Non-VBA

- To show how simulation can be used to see how a game like poker works and whether its rules are reasonable. (Do the values of the hands go along with their likelihoods?)

35.2 Functionality of the Application

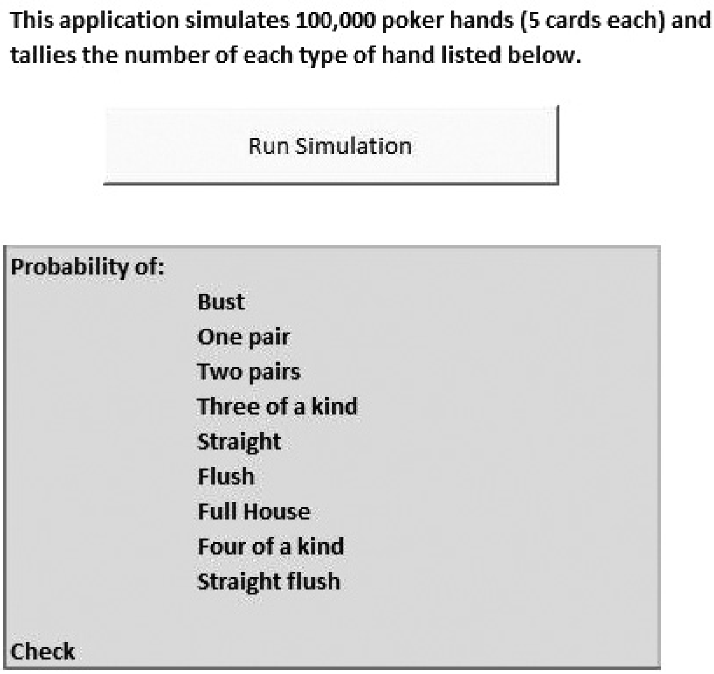
The purpose of this application is to repeatedly simulate five-card hands from a 52-card deck, tally the numbers of hands of each type, and display the relative frequencies in a worksheet.

35.3 Running the Application

The application is stored in the file **Poker.xlsm**. This file contains a single worksheet named Report, shown in Figure 35.1, which the user sees upon opening the file. Each time the user clicks the button, 100,000 *new* five-card hands are simulated, all from a fresh 52-card deck, and the results are displayed in the worksheet, as shown in Figure 35.2.

I say that 100,000 *new* hands are simulated because each run uses a new set of random numbers for the simulation. Therefore, the results will be slightly

Figure 35.1 Report Worksheet Before Running Simulation



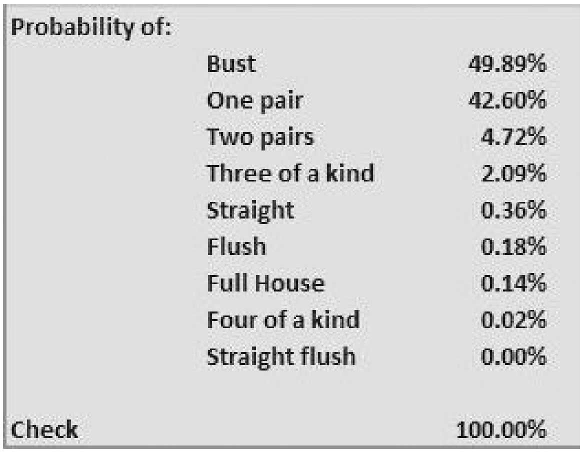
This application simulates 100,000 poker hands (5 cards each) and tallies the number of each type of hand listed below.

Probability of:

Bust
One pair
Two pairs
Three of a kind
Straight
Flush
Full House
Four of a kind
Straight flush

Check

Figure 35.2 Results from a Simulation Run

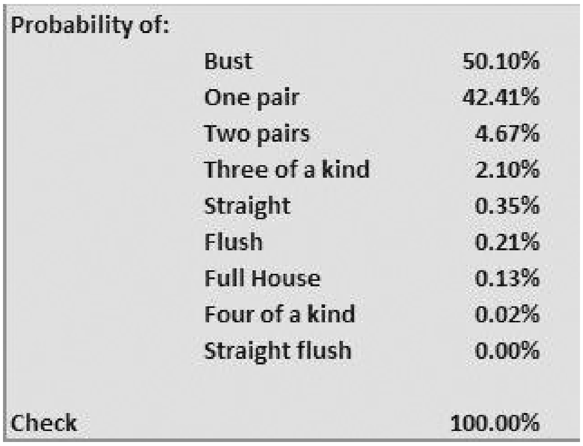


Probability of:		
	Bust	49.89%
	One pair	42.60%
	Two pairs	4.72%
	Three of a kind	2.09%
	Straight	0.36%
	Flush	0.18%
	Full House	0.14%
	Four of a kind	0.02%
	Straight flush	0.00%
Check		100.00%

different each time the application is run. Figure 35.3 shows results from a different set of 100,000 hands. They are very similar to the results in Figure 35.2, but they are not exactly the same. This is the nature of simulation. You will undoubtedly get slightly different results each time you run it.

Each of these runs illustrates what can be shown from a formal probability argument—the probabilities of the hands go in reverse order of the values of the hands. A bust is most likely, a pair is next most likely, and so on.[1] And if you are counting on getting four of a kind or a straight flush, dream on!

Figure 35.3 Results from Another Simulation Run

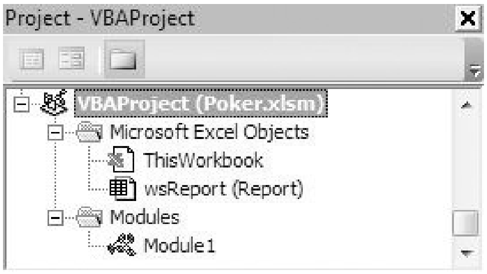


Probability of:		
	Bust	50.10%
	One pair	42.41%
	Two pairs	4.67%
	Three of a kind	2.10%
	Straight	0.35%
	Flush	0.21%
	Full House	0.13%
	Four of a kind	0.02%
	Straight flush	0.00%
Check		100.00%

---

[1] Again, because of the nature of simulation, it is *possible* that you will get results where, for example, there are more flushes than straights, but this is due to what statisticians call sampling error.

Figure 35.4 Project Explorer Window



35.4 Setting Up the Excel Sheets

There is really nothing to set up at design time other than to enter labels and format the Report worksheet, as shown in Figure 35.1. There is nothing “hidden” here. Other than labels, the worksheet is blank, waiting for the simulated results. Furthermore, the simulation occurs completely in VBA code. There is no worksheet for calculations.

35.5 Getting Started with the VBA

The application requires only a module—no user forms or references. After the module is added, the Project Explorer window will appear as in Figure 35.4.

Workbook_Open Code

The following code is placed in the `ThisWorkbook` code window. It clears results from any previous simulation run.

```
Private Sub Workbook_Open()
    With wsReport
        .Range("D10:D20").ClearContents
        .Range("C6").Select
    End With
End Sub
```

35.6 The Module

To this point, you might think this application is a fun little exercise for card players. However, the VBA code is far from trivial. It requires some careful logic, and it makes heavy use of arrays. It is an interesting illustration of how humans can easily perceive patterns that computers can discover only with intricate programming. For example, a poker player can look at his hand, without even rearranging the cards, and immediately see that he has a pair, a straight, or whatever. As the code will show, however, it takes a considerable amount of code to recognize these patterns.

The module-level variables are listed first. As in previous chapters, they are declared with the keyword `Dim`, not with `Public`. These module-level variables need to be declared at the top of the module, outside of the subs, so that all of the subs in the module can recognize them.

Option Statement and Module-Level Variables

```
Option Explicit

' Definitions of module-level variables and constant:
'    nBust - number of the 100,000 hands that results in a bust (with similar
'          definitions for nPair, n2Pair, etc.
'    denom - array that indicates which denomination (1 to 13) each card
'          in the deck is
'    card - array that indicates the cards in the hand - e.g., if card(3) = 37,
'          this means the third card dealt is the 37th card in the deck
'    nReps - number of simulated hands, in this case 100,000

Dim nBust As Long, nPair As Long, n2Pair As Long, n3ofKind As Long, _
    nFullHouse As Integer, n4ofKind As Integer, nStraight As Integer, _
    nFlush As Integer, nStraightFlush As Integer
Dim denom(1 To 52) As Integer
Dim card(1 To 5) As Integer

Const nReps = 100000
```

Main Code

The `Main` sub runs when the user clicks the button on the Report worksheet. It first calls the `InitializeStats` sub to set all counters to 0. Next, it calls the `SetupDeck` sub to "define" the cards in the deck. Then it uses a `For` loop to run the 100,000 replications of the simulation. In each replication it calls the `Deal` sub to deal the cards and the `EvaluateHand` sub to check what type of hand is obtained. Finally, it calls the `Report` sub to put the results in the Report worksheet. VBA's `Randomize` function is placed near the top of the `Main` sub to ensure that a new set of random numbers is used each time the simulation is run.

```
Sub Main()
    Dim iRep As Long ' replication index
    Randomize

    ' Set counters to 0.
    Call InitializeStats

    ' "Name" the cards in the deck.
    Call SetupDeck

    ' Deal out nReps poker hands and evaluate each one.
    For iRep = 1 To nReps
        Call Deal
        Call EvaluateHand
    Next

    ' Report the summary stats from the nReps hands.
    Call Report

    wsReport.Range("C6").Select
End Sub
```

InitializeStats Code

The InitializeStats sets all counters (the number of busts, the number of pairs, and so on) to 0.

```
Sub InitializeStats()
    nBust = 0
    nPair = 0
    n2Pair = 0
    n3ofKind = 0
    nStraight = 0
    nFlush = 0
    nFullHouse = 0
    n4ofKind = 0
    nStraightFlush = 0
End Sub
```

SetupDeck Code

The SetupDeck sub "defines" the deck by filling the denom array. It does this with two nested For loops. If you follow the logic closely, you will see that denom(1) through denom(4) are set to 1 (corresponding to the Aces), denom(5) through denom(8) are set to 2 (corresponding to the 2s), and so on. You can think of denomination 11 as the Jacks, denomination 12 as the Queens, and denomination 13 as the Kings. Also, there are no explicit hearts, diamonds, clubs, and spades, but you can think of cards 1, 5, 9, and so on as the hearts; cards 2, 6, 10, and so on as the diamonds; cards 3, 7,11, and so on as the clubs; and cards 4, 8,12, and so on as the spades.

```
Sub SetupDeck()
    Dim iDenom As Integer ' denomination index
    Dim iSuit As Integer ' suit index
    ' Give the first 4 cards denomination 1 (aces),
    ' the next 4 denomination 2 (deuces), and so on
    For iDenom = 1 To 13
        For iSuit = 1 To 4
            denom(4 * (iDenom - 1) + iSuit) = iDenom
        Next
    Next
End Sub
```

Deal Code

The `Deal` sub randomly chooses five cards from the 52-card deck. It is the only sub where any simulation takes place; that is, it is the only code that uses random numbers. It uses VBA's `Rnd` function (which is essentially equivalent to Excel's RAND function) to simulate a single random number uniformly distributed between 0 and 1. The following line generates a uniformly distributed *integer* from 1 to 52:

```
cardIndex = Int(Rnd * 52) + 1
```

Note how this works. The quantity `Rnd * 52` is a uniformly distributed *decimal* number between 0 and 52. Then VBA's `Int` function chops off the decimal, leaving an integer from 0 to 51. Finally, 1 is added to obtain an integer from 1 to 52.

The Boolean `isUsed` array keeps track of which of the 52 cards in the deck have *already* been dealt in the current hand. Essentially, random integers are generated until five *distinct* integers have been obtained. When an integer is generated that is distinct from the previous integers, its `isUsed` value is set to `True`, so that it cannot be used again (in this hand). By the end of this sub, the indexes of the five cards dealt are stored in the `card` array. For example, if `card(4) = 47`, this means that the fourth card in the hand is the 47th card in the deck (the Queen of clubs).

```
Sub Deal()
    Dim i As Integer ' index of cards in deck
    Dim j As Integer ' index of cards in hand
    Dim cardIndex As Integer
    Dim used(1 To 52) As Boolean
    Dim newCard As Boolean

    ' Initially, no cards have been dealt.
    For i = 1 To 52
        used(i) = False
    Next

    ' For each of 5 cards, keep generating until a new card is dealt.
    For j = 1 To 5
        newCard = False
        Do
            cardIndex = Int(Rnd * 52) + 1
            If Not used(cardIndex) Then
                newCard = True
                used(cardIndex) = True
            End If
        Loop Until newCard = True

        ' Records the card number this card in this hand.
        card(j) = cardIndex
    Next
End Sub
```

EvaluateHand Code

The most difficult part of the program is the EvaluateHand sub. By this time, the card array has been generated. It might show that the hand contains the cards 2, 7, 19, 28, and 47. What kind of a hand is this? Is it a bust, a pair, or what? The Evaluate sub goes through the necessary logic to check all possibilities.

The first check is for a straight. It finds the denominations of the five cards and stores them in the cardDenom array. For example, the denomination of the first card is denom(card(1)), which is stored in cardDenom(1). These denominations might be out of order, such as 5, 3, 7, 6, 4, so it uses two nested For loops to sort them in increasing order. It then checks whether the sorted denominations form a progression, such as 3, 4, 5, 6, 7. (If this sounds overly complex, just try doing it any other way.)

The second check is for a flush. For example, the hand with cards 3, 15, 23, 39, 51 is a flush. This is because cards 3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, and 51 are the 13 cards of one suit (clubs, say). An easy way to check whether *any* five cards are of the same suit is to divide each of them by 4 and see whether the remainders are all equal. (This is the case for 3, 15, 23, 39, and 51; each has remainder 3.) This can be done with VBA's Mod function. For example, 51 Mod 4 is the remainder when 51 is divided by 4.

If the hand is a straight or a flush (or both), no further checks are necessary. Otherwise, checks for a pair, two of a kind, and the others are necessary. All of these involve the numbers of like denominations in a hand. For example, a hand with two pairs contains two of some denomination, two of another, and one of another. The groups array is used to collect this information. A full house has groups(3) = 1 and groups(2) = 1, which says that it has one group of size 3 and one group of size 2. Similarly, a bust has groups(1) = 5, three of a kind has groups(3) = 1 and groups(1) = 2, and so on. So by filling the groups array and checking its contents, the program can discover which type of hand has been dealt.

```
Sub EvaluateHand()
    Dim i As Integer, j As Integer
    Dim count(1 To 13) As Integer, groups(1 To 4) As Integer
    Dim hasStraight As Boolean, hasFlush As Boolean
    Dim cardDenom(1 To 5) As Integer, temp As Integer

    ' First, check for a straight.
    hasStraight = False
    For i = 1 To 5
        cardDenom(i) = denom(card(i))
    Next
```

```
' Sort the denominations in increasing order.
For i = 1 To 4
    For j = i + 1 To 5
        If cardDenom(j) < cardDenom(i) Then
            temp = cardDenom(j)
            cardDenom(j) = cardDenom(i)
            cardDenom(i) = temp
        End If
    Next
Next

' Check if they are in a progression, like 4, 5, 6, 7, 8.
' If you consider Aces as denomination 13, then this code
' counts only "Ace high" straights.
If cardDenom(2) = cardDenom(1) + 1 And _
    cardDenom(3) = cardDenom(2) + 1 And _
    cardDenom(4) = cardDenom(3) + 1 And _
    cardDenom(5) = cardDenom(4) + 1 Then
    hasStraight = True
    nStraight = nStraight + 1
End If

' Next, check for a flush.
hasFlush = False
If card(1) Mod 4 = card(2) Mod 4 And card(2) Mod 4 = card(3) Mod 4 _
            And card(3) Mod 4 = card(4) Mod 4 And _
            card(4) Mod 4 = card(5) Mod 4 Then
    hasFlush = True
    nFlush = nFlush + 1
End If

' Next, check for a straight flush.
If hasStraight And hasFlush Then
    nStraightFlush = nStraightFlush + 1
    ' Don't count this a straight or a flush.
    nStraight = nStraight - 1
    nFlush = nFlush - 1
End If

' There's no need to check the rest if the hand is a straight
' or a flush (or both).
If hasStraight Or hasFlush Then Exit Sub

' Otherwise, check all the other possibilities.
' count(i) is the number of cards of denomination i in the hand.
For i = 1 To 13
    count(i) = 0
Next
For i = 1 To 5
    count(denom(card(i))) = count(denom(card(i))) + 1
Next

' groups(i) will be the number of "groups" of size i.
' For example, if groups(2) = 1, then there is one group of
' size 2, that is, one pair (of some denomination).
For i = 1 To 4
    groups(i) = 0
Next
For i = 1 To 13
```

```
            If count(i) > 0 Then groups(count(i)) = groups(count(i)) + 1
        Next

        ' Now go through all of the possibilities.
        If groups(1) = 5 Then
            nBust = nBust + 1
        ElseIf groups(1) = 3 And groups(2) = 1 Then
            nPair = nPair + 1
        ElseIf groups(1) = 1 And groups(2) = 2 Then
            n2Pair = n2Pair + 1
        ElseIf groups(1) = 2 And groups(3) = 1 Then
            n3ofKind = n3ofKind + 1
        ElseIf groups(2) = 1 And groups(3) = 1 Then
            nFullHouse = nFullHouse + 1
        Else
            n4ofKind = n4ofKind + 1
        End If
    End Sub
```

Report Code

The `Report` sub lists the results in the Report worksheet. Note that it reports the *relative* frequencies, such as the number of busts divided by the total number of replications. The formula in cell D20 is not really necessary, but it provides a comforting check that the relative frequencies sum to 1, as they should. If a number other than 1 appeared in cell D20, this would indicate a bug in the program.

```
Sub Report()
    With wsReport
        .Range("D10").Value = nBust / nReps
        .Range("D11").Value = nPair / nReps
        .Range("D12").Value = n2Pair / nReps
        .Range("D13").Value = n3ofKind / nReps
        .Range("D14").Value = nStraight / nReps
        .Range("D15").Value = nFlush / nReps
        .Range("D16").Value = nFullHouse / nReps
        .Range("D17").Value = n4ofKind / nReps
        .Range("D18").Value = nStraightFlush / nReps

        ' Check that they sum to 1.
        .Range("D20").Formula = "=Sum(D10:D18)"
    End With
End Sub
```

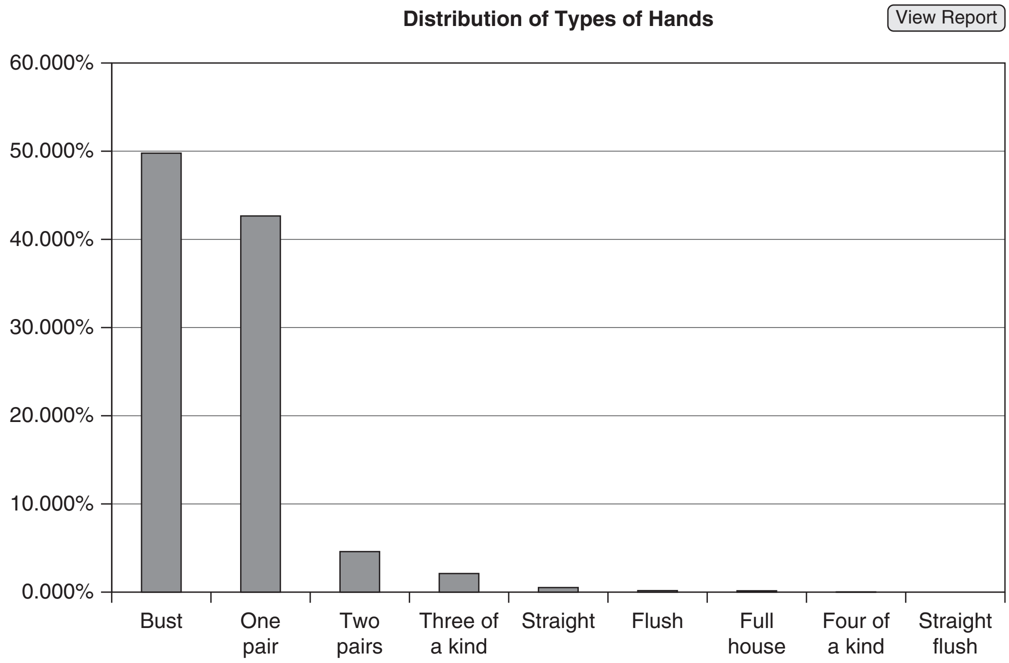
35.7 Summary

The application in this chapter is not earthshaking, except perhaps to avid poker players, but it does illustrate an interesting and certainly nontrivial use of logic, loops, and arrays. In addition, the results of the simulation agree with our intuition about the game of poker itself. They show that as hands become more valuable, they become less likely. And if you always thought you were unlucky because you got a lot of busts, you now realize that this happens about 50% of the time.

EXERCISES

1. Change the application so that it contains a chart sheet displaying the frequencies of the various types of hands, as in Figure 35.5. Put a button on the Report worksheet to navigate to this chart sheet. (Do you need to write any code to update the chart after each run?)
2. There are many versions of poker. Change the application so that it works for a version where the player is dealt six cards and then gets to discard any one of them. Assume that the player will discard the card that makes the remaining hand as valuable as possible. (*Hint*: Probably the simplest approach is to run the `EvaluateHand` sub on each of the possible five-card hands with one of the six cards omitted and take the best.)
3. A more realistic version of the previous exercise is where the player is dealt five cards. He can discard as many as four of these and request replacements from the remaining deck. The problem with simulating this version is that you have to know the player's strategy—depending on what he is dealt and what he will discard. Simulate the following strategy.

Figure 35.5 Frequency Chart for Exercise 1



- If dealt a bust, discard all but a single card. (Normally, a player would keep the highest card, but it doesn't make any difference here.)
- If dealt a pair, keep the pair and discard the other three cards.
- If dealt two pairs, keep the pairs and discard the other card.
- If dealt three of a kind, keep these three and discard the other two cards.
- If dealt any other type of hand, keep it and discard nothing.

4. (More difficult) In the preceding exercise, the player never tries to "fill in" partial straights or flushes. For example, if he has a 4, 5, 6, 7, and 10, he doesn't discard the 10, hoping to fill the straight with a 3 or an 8. Similarly, if he has four hearts and a spade, he doesn't discard the spade, hoping to fill the flush with another heart. Simulate such a strategy. Specifically, assume he first checks for a bust. If he has a bust, he checks whether he has a partial straight that could be completed on *either* end. (This means, for example, 4, 5, 6, 7, but not 1, 2, 3, 4. Trying to complete this latter straight is too risky because only a 5 will do it.) If he has such a partial "inside" straight, he discards the other card. Otherwise, still assuming he has a bust, he checks whether he has four cards of one suit. If so, he discards the other card. Otherwise, he discards any four cards from the bust. The rest of his strategy is the same as the last four bulleted points in the previous exercise. In other words, he tries to complete a straight or a flush only when he has a bust. Based on your simulation results, is this strategy better or worse than the strategy in the previous exercise?

5. In the game of bridge, each of four players is dealt 13 cards from a 52-card deck. Concentrate for now on a particular player. Develop a simulation similar to the poker simulation that finds the distribution of the number of aces the player is dealt. (*Note*: Since you are concentrating on one player only, you need to simulate 13 cards only; you can ignore what the other three players get.)

6. Continuing the previous exercise, again concentrate on a single player and simulate the distribution of the maximum number of any suit the player is dealt. For example, if the hand has five hearts, three diamonds, three clubs, and two spades, this maximum number is 5. How likely is it that a player will get at least 11 cards of some suit?