Comparing Baseball Players Using Expected Runs in Shiny

Spencer Rodrigues

Statistics Department

California Polytechnic State University, San Luis Obispo

June, 2017

Table of Contents

Introduction

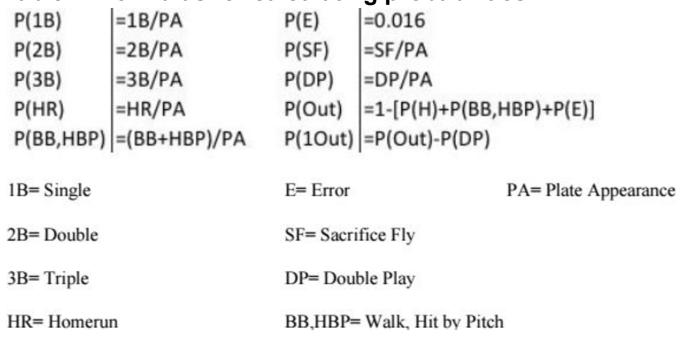
The game of baseball is not just a game of physical capability; it also consists of a large portion of intellect. According to the great Yogi Berra, "Baseball is ninety percent mental. The other half is physical." I have played baseball for seventeen years of my life, as both a pitcher and field player, and there is a lot of truth to Yogi's statement. There is never a resting moment in the game, in the mental aspect. There are so many decisions that have to be made before and during every play, from which pitch should be thrown to where the ball should be thrown once you field it. With all these decisions that needed to be made, I wanted to further investigate and then create an interface to compare players, which could lead to in game decisions for a team.

This project used portions of code from VJ Asaro's Senior Project, with some touch ups. VJ's code used Markov chains and a batter's data to calculate the expected runs from all the base states. A base state is a combination of which bases are occupied, and how many outs there are in the inning. For example, if we are in the (13,0) base state, there is a runner on 1st and 3rd with nobody out. I developed it by fixing some errors that he had, adding expected run calculations for pitchers, and making these into Shiny apps. One of the ideas for this project was to assess which player would be better in a given situation. For batting, if it is late in the game, we might want to put a pinch hitter into the game. With this analysis we would be able to make a better decision than just looking at batting average. For pitching, we would be able to decide which reliever we would want to put into the game. With the expected run calculations we will be able to use a more complex analysis to aid coaches in decisions. It can also allow fans to compare their favorite players to any other players in the same year, helping them in fantasy sports, or just to see who is better.

Expected Runs

When computing the expected runs for a half inning we will need to calculate the expected runs for a single at bat. To do this we will need the probabilities of a specific player's batting outcomes. This is calculated by the following table. The only value that will be exactly the same for every batter is the probability of an error. This is because we are using the proportion of plays where an error occurs in the MLB, according to baseball reference.

Table 1: Formulas for calculating probabilities

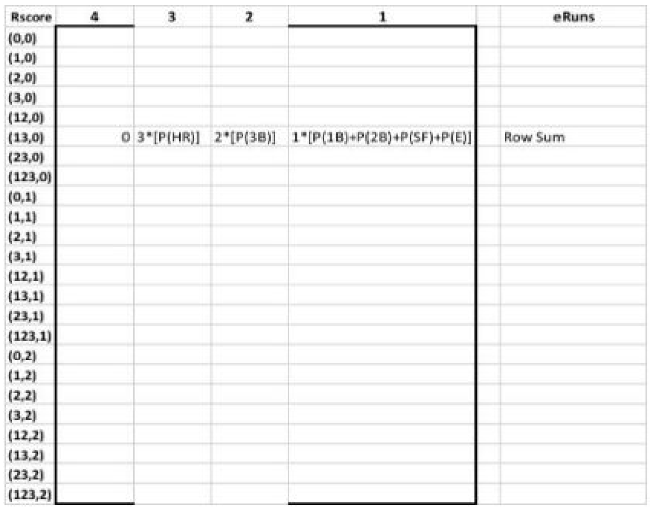


P(1B)	=1B/PA	P(E)	=0.016
P(2B)	=2B/PA	P(SF)	=SF/PA
P(3B)	=3B/PA	P(DP)	=DP/PA
P(HR)	=HR/PA	P(Out)	=1-[P(H)+P(BB,HBP)+P(E)]
P(BB,HBP)	=(BB+HBP)/PA	P(1Out)	=P(Out)-P(DP)

1B= Single	E= Error	PA= Plate Appearance
2B= Double	SF= Sacrifice Fly	
3B= Triple	DP= Double Play	
HR= Homerun	BB,HBP= Walk, Hit by Pitch	

With these probabilities we calculate the probability of scoring 4,3,2 and 1 runs in a base state, and then we sum the row for the expected runs in one at bat. For example, if there were runners on 1st and 3rd, the calculation for expected runs in that state is

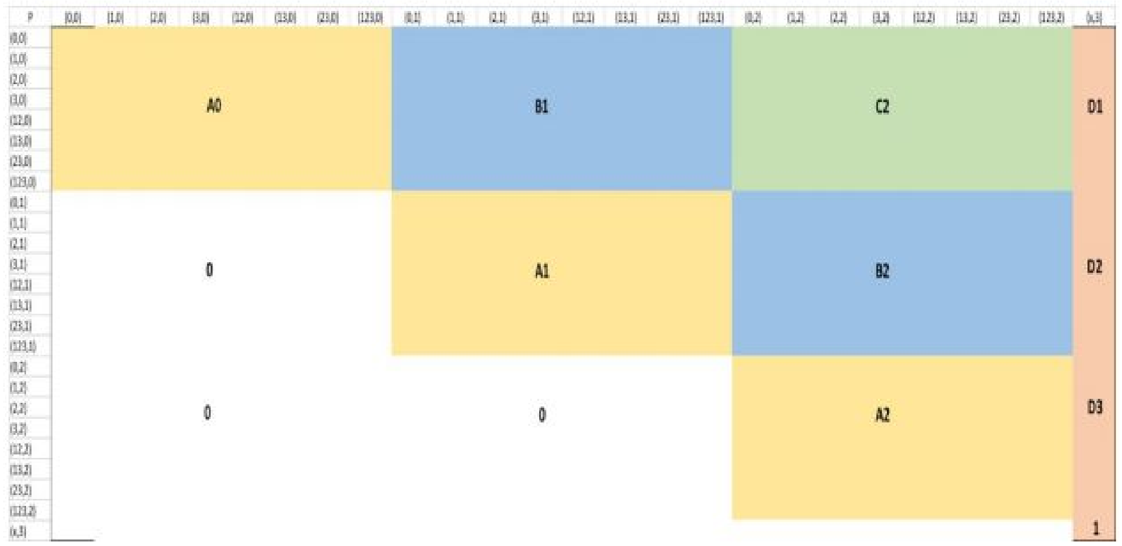
3*P(HR) + 2*P(3B) + 1*[P(1B) + P(2B) + P(SF) + P(E)] + 0*[P(BB,HBP) + P(Out)]. This type of computation will be done for all 24 base states. The table below shows the calculation of one element of **eRuns**, the vector of expected runs for a single at bat from each base state.



Rscore	4	3	2	1		eRuns
(0,0)						
(1,0)						
(2,0)						
(3,0)						
(12,0)						
(13,0)	0	3*[P(HR)]	2*[P(3B)]	1*[P(1B)+P(2B)+P(SF)+P(E)]		Row Sum
(23,0)						
(123,0)						
(0,1)						
(1,1)						
(2,1)						
(3,1)						
(12,1)						
(13,1)						
(23,1)						
(123,1)						
(0,2)						
(1,2)						
(2,2)						
(3,2)						
(12,2)						
(13,2)						
(23,2)						
(123,2)						

For the calculation of expected runs for an entire half inning we need to use a transition matrix. Each element in the transition matrix, **P**, contains the probability of moving from one base state to another, using the left column as your current state and the top row as the state you are moving to.

Table 2: Transition matrix, color coded corresponding to outs occurring on play



In the matrix **P** above, the yellow squares represent the states that have no outs occurring in the at-bat, the blue squares mean 1 out occurs and the green square means that 2 outs occur on the play. Once the red column is reached, it means that there are 3 outs and that the half inning has ended.

We need to solve this matrix to further progress our calculations for expected runs in a half inning. The equation we use for this is $\mathbf{E} = (\mathbf{I}-\mathbf{Q})^{-1}$ where **Q** is a 24x24 submatrix of **P**, with the absorbing three out states are removed (row and column) and **I** is the 24x24 identity matrix. Matrix **E** is then a 24x24 matrix whose values represent the expected number of visits to each state, starting from each state until the 3rd out occurs. The number of plate appearances before the absorbing state are calculated by the row sums of **E**. Our expected runs for the half inning is then calculated by taking our vector of expected runs, **eRuns**, and multiplying it by **E**.

New R Functions

There were three new R functions that had to be written to be able to do all these calculations. The first one is called **prob.batting**. In this function, we take all of the batting statistics for the player in one season and convert them all to the proportion of their at bats that occurred in that result. The proportions that it outputs are in **Table 1.**

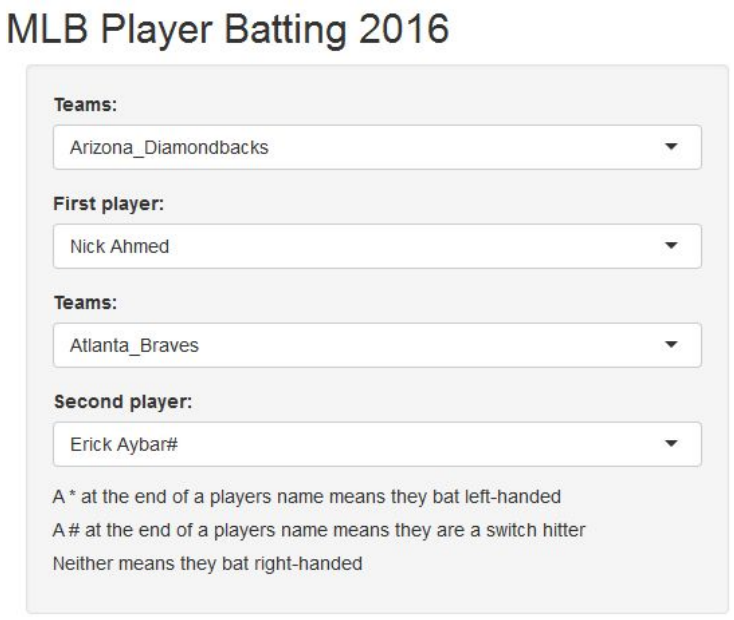
The next new function that was created was **trans.batting**. In this function, we are making the transition matrix, shown in **Table 2**. The transition matrix contains the probability of moving from the current base state (shown as the row) to every other possible base state (shown as the columns). This is computed individually for each batter, using their batting statistics from the **prob.batting** function. Once this is run we will have a 25x25 matrix full of probabilities to help us calculate the expected runs in the app. This function was written by VJ Asaro, but there were some small errors that occurred when changing the order of his code. I had a difficult time trying to fix them, but eventually I realized that his function was in need of another user input to account for iterating through the team data.

The last new function is **Mat.Exp.Runs**. This function requires the R package **Plotrix**. This function creates an 8x3 matrix of expected runs and each cell of the matrix is shaded depending on the value. The lighter a color is, the more runs we would expect to score from that state.

Shiny Batting App

When using the application, the first thing you will need to do is think of two batters that you want to compare. Once you have done that, select their respective teams under each **Teams:** pull down bar, and then select the player from the first team in the **First Player:** pull down bar then select the player from the second team in the **Second Player:** pull down bar.

The players that are available to select are players that had at least ten at bats in the season. If they played for more than one team in a season, the batting statistics in their expected runs calculation will only include their batting with that particular team.



MLB Player Batting 2016

Teams:
Arizona_Diamondbacks

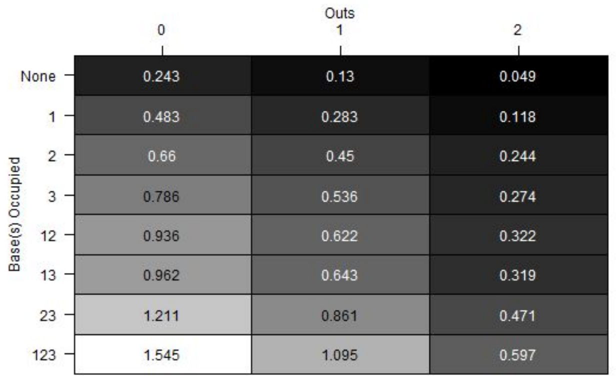
First player:
Nick Ahmed

Teams:
Atlanta_Braves

Second player:
Erick Aybar#

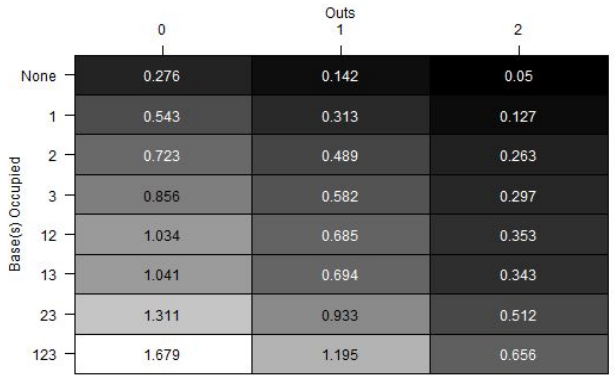
A * at the end of a players name means they bat left-handed
A # at the end of a players name means they are a switch hitter
Neither means they bat right-handed

Once the players have been selected, the expected runs matrices will appear to the right. Since there are eight different base states and three possible out states, we will be able to make our 24x1 expected runs vector into an 8x3 expected runs matrix. Each base state's expected runs is the amount of runs we would expect them to score for the rest of that inning from that state, given that our selected player has all the at bats.



Base(s) Occupied	Outs 0	1	2
None	0.243	0.13	0.049
1	0.483	0.283	0.118
2	0.66	0.45	0.244
3	0.786	0.536	0.274
12	0.936	0.622	0.322
13	0.962	0.643	0.319
23	1.211	0.861	0.471
123	1.545	1.095	0.597

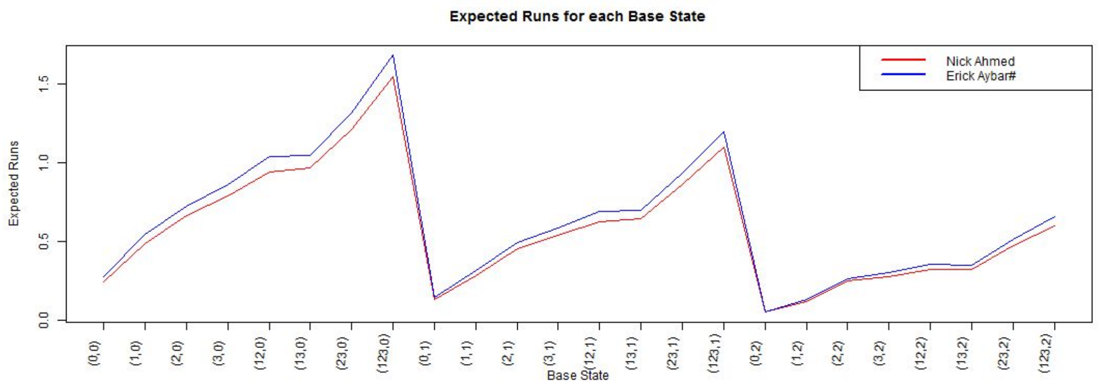
Nick Ahmed Expected Runs

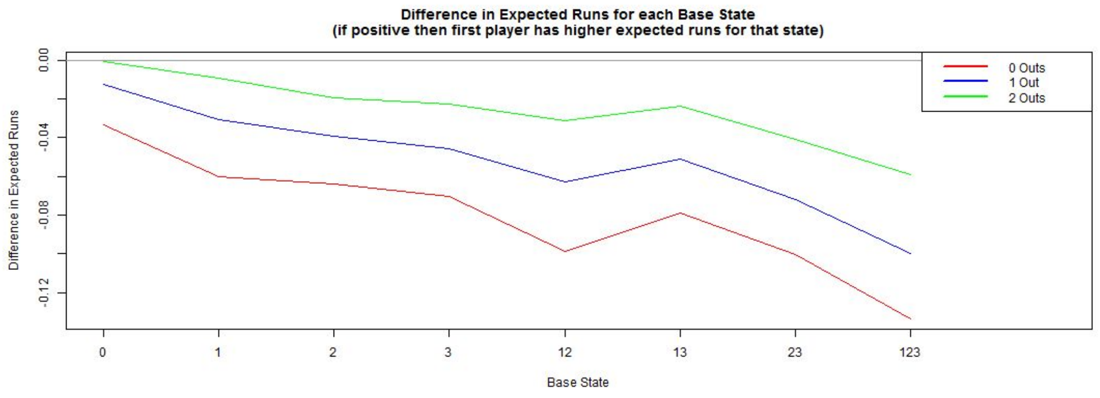


Base(s) Occupied	Outs 0	1	2
None	0.276	0.142	0.05
1	0.543	0.313	0.127
2	0.723	0.489	0.263
3	0.856	0.582	0.297
12	1.034	0.685	0.353
13	1.041	0.694	0.343
23	1.311	0.933	0.512
123	1.679	1.195	0.656

Erick Aybar# Expected Runs

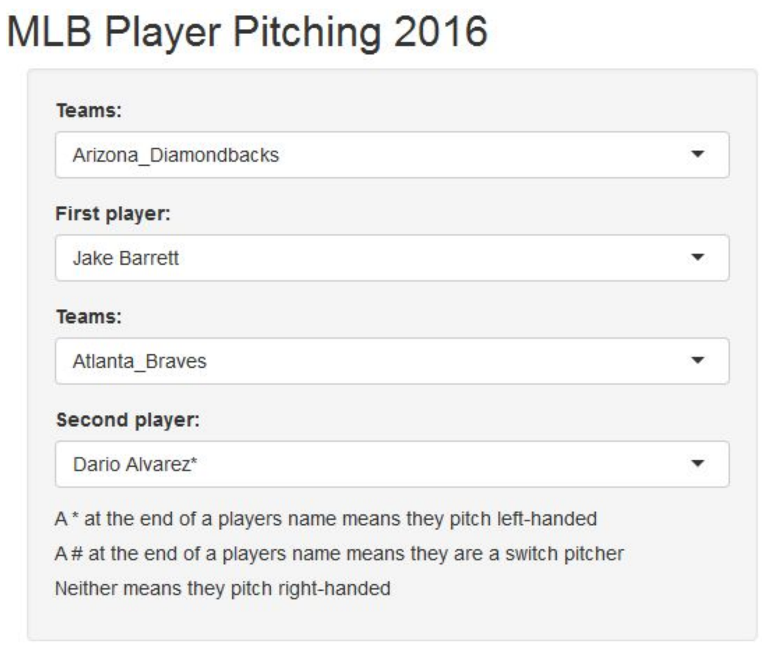
Under these matrices are two graphical components to aid the user in comparing the two players. The top graph has the two players' 24 base state graphs overlaid on each other. The first player's line is red and the second player's line is blue. The bottom graph shows the difference between the two players at the eight base states, using different colors to represent the different out states. If the value is negative at a particular state, it means that the second player has a higher number of expected runs in that state than the first player, and vice-versa if the difference is positive.





Shiny Pitching App

When using the application, the first thing you will need to do is think of two pitchers that you want to compare. Once you have done that, select their respective teams under each **Teams:** pull down bar, and then select the player from the first team in the **First Player:** pull down bar and the player from the second team in the **Second Player:** pull down bar. The players that are available to select are players that had at least three innings pitched in the season. If they played for multiple teams during the season, their expected runs allowed calculation will only include their pitching with that particular team.



MLB Player Pitching 2016

Teams: Arizona_Diamondbacks

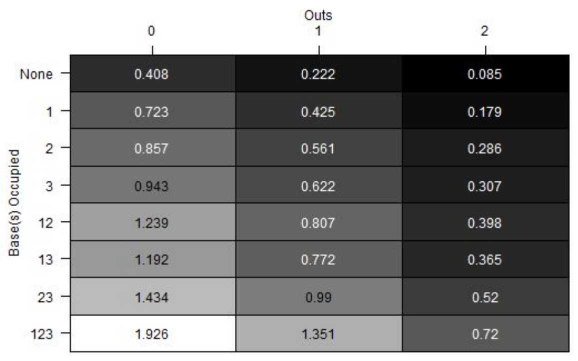
First player: Jake Barrett

Teams: Atlanta_Braves

Second player: Dario Alvarez*

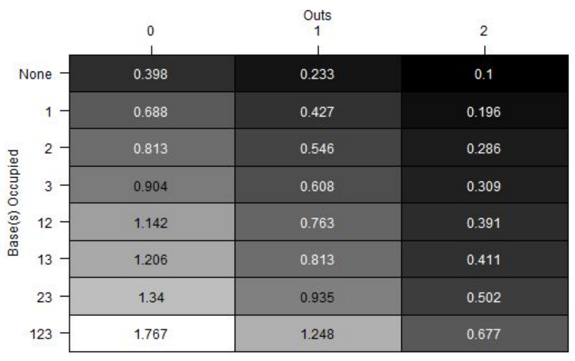
A * at the end of a players name means they pitch left-handed
A # at the end of a players name means they are a switch pitcher
Neither means they pitch right-handed

Once the players have been selected, the expected runs allowed matrices will appear to the right. Since there are eight different base states and three possible out states, we will be able to make our 24x1 expected runs vector into an 8x3 expected runs matrix. Each state's expected runs allowed is the amount of runs we would expect them to give up for the rest of that inning from that state.



Base(s) Occupied	Outs 0	Outs 1	Outs 2
None	0.408	0.222	0.085
1	0.723	0.425	0.179
2	0.857	0.561	0.286
3	0.943	0.622	0.307
12	1.239	0.807	0.398
13	1.192	0.772	0.365
23	1.434	0.99	0.52
123	1.926	1.351	0.72

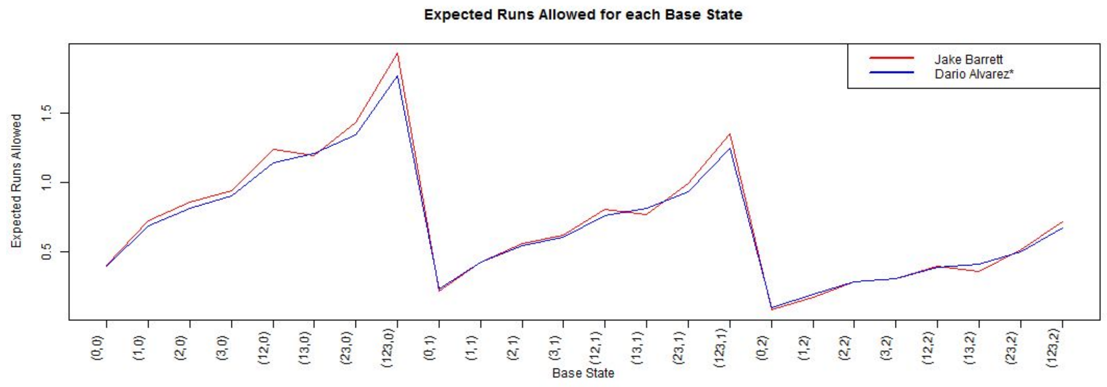
Jake Barrett Expected Runs Allowed

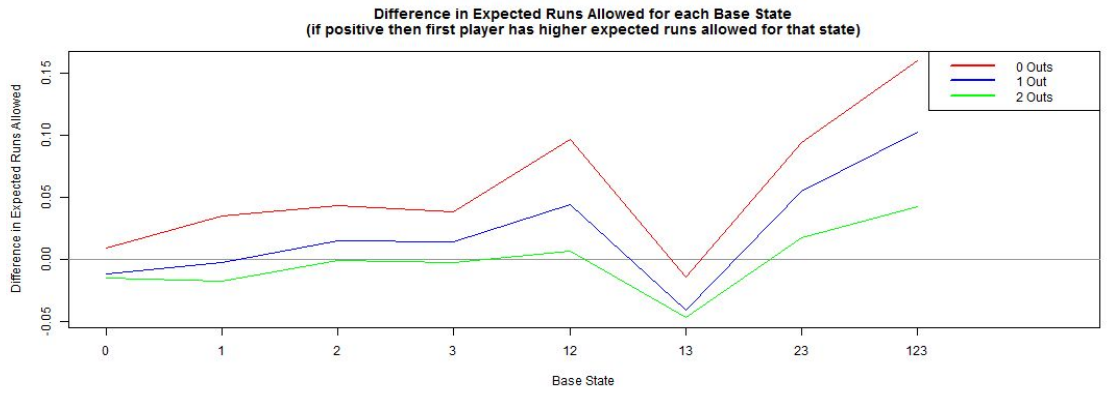


Base(s) Occupied	Outs 0	Outs 1	Outs 2
None	0.398	0.233	0.1
1	0.688	0.427	0.196
2	0.813	0.546	0.286
3	0.904	0.608	0.309
12	1.142	0.763	0.391
13	1.206	0.813	0.411
23	1.34	0.935	0.502
123	1.767	1.248	0.677

Dario Alvarez* Expected Runs Allowed

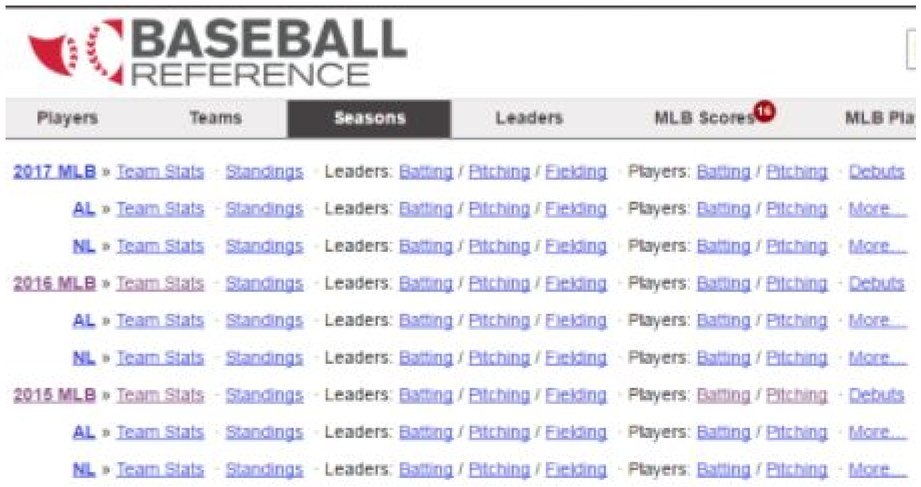
Under these matrices are two graphical components to aid the user in comparing the two players. The top graph has the two players' 24 base state graphs overlaid on each other. The first player's line is red and the second will be blue. The bottom graph shows the differences between the two players at the eight base states, using different colors to represent the different out states. If the value is negative at a particular state, it means that the second player has a higher expected runs allowed in that state than the first player, and vice-versa if it is positive.



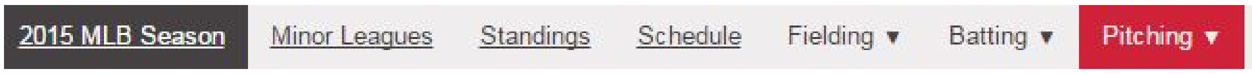


Updating the Apps

I am hoping that there will be someone to update the data on these apps at the end of each season. The steps to update the data for a new season are quite simple. For the batting data, you will need to go to http://www.baseball-reference.com/. You will see the following at the top of the web page. Hover your mouse over **Seasons** and then the select the **Players: Batting** option. Once that has been done you will be on the page for that seasons batters. Find the section labeled **Player Standard Batting**, click on the **Share & More Option,** and then click on **Get as Excel Workbook (experimental)**. Once the file is downloaded, you will need to save the file as **ALL_Batting_2017.csv**, or change the year to whatever year it is. Make sure that this file is saved as a CSV file and not a Web Page.


BASEBALL REFERENCE
Players
Teams
Seasons
Leaders
MLB Scores
MLB Pla
2017 MLB » Team Stats · Standings · Leaders: Batting / Pitching / Fielding · Players: Batting / Pitching · Debuts
AL » Team Stats · Standings · Leaders: Batting / Pitching / Fielding · Players: Batting / Pitching · More...
NL » Team Stats · Standings · Leaders: Batting / Pitching / Fielding · Players: Batting / Pitching · More...
2016 MLB » Team Stats · Standings · Leaders: Batting / Pitching / Fielding · Players: Batting / Pitching · Debuts
AL » Team Stats · Standings · Leaders: Batting / Pitching / Fielding · Players: Batting / Pitching · More...
NL » Team Stats · Standings · Leaders: Batting / Pitching / Fielding · Players: Batting / Pitching · More...
2015 MLB » Team Stats · Standings · Leaders: Batting / Pitching / Fielding · Players: Batting / Pitching · Debuts
AL » Team Stats · Standings · Leaders: Batting / Pitching / Fielding · Players: Batting / Pitching · More...
NL » Team Stats · Standings · Leaders: Batting / Pitching / Fielding · Players: Batting / Pitching · More...

For the pitching data, we will use the same site, except when we hover of the **Seasons** option, we will select **Players: Pitching**. Once there, find this bar.



Hover your mouse over the **Pitching** option, and then inside those options choose **Batting Against**. Find the section labeled **Player Batting Against**, click on the **Share & More Option,** then click on **Get as Excel Workbook (experimental)**. Once the file is downloaded, you will need to save the file as **ALL_PITCHERS_2017.csv**, or change the year to whatever year it is. Make sure that the file is saved as a CSV file and not a Web Page.

There will be no need to clean any of the data that is obtained because the app will do it. In the batting app, only the batters with more than 10 at bats for the season will be kept in the data. For the pitching app, pitchers with more than 3 innings pitched will be kept, and the rest will be removed. This is because some position players, like the center fielder, might come into the game to pitch if their team is losing by a lot. In our pitching data we only want to include actual pitchers to aid in decisions that can be made by the comparisons. Once the data is cleaned to the two specifications above, the data will be subset into teams, creating 2 new data sets for each team, 1 for pitchers and 1 for batters. Figuring out how to do this took me a decent amount of time. I originally downloaded the data for each team, one by one. This took a while to be completed. It wasn't until later that I noticed I could download all of the players into one data set and then have the app separate that data into the teams.

Conclusion

Baseball's great complexity means that these apps will not be perfect. There are specific calculations that are always changing and there are many other factors that just can't be accounted for in this formula. There are still some touch-ups that can be added to further improve these applications. The ability to compare players from one year to the next would be the first important step. Once that is possible it would be incredible to add all the seasons data from baseball reference, so that we could compare the greats of the past to those of the present. After the seasons of the past are added, the application should be updated at the end of every season to allow for comparisons with recent players.The next step would be to account for more diverse base-running decisions, to allow for a more accurate Expected Runs. One of the ways that this can be done is to include the probability that a runner scores from first on a double. Right now, there is only code in **trans.batting** that accounts for a base runner scoring from second on a single. The rest of the base running assumptions are that the base runners only advance the same amount of bases as the batter. For example, if there is a runner on first, and the batter hits a single, there will always be a batter on second and third. Another way to make the base running decisions more diverse would be to include the probability of multi-base errors. The only type of error that we include in the model is an error that lets the runners advance one base, but sometimes an error occurs where the base runner is able to advance two bases.

These applications have a variety of uses. These include: deciding which relief pitcher to put into the game, which batter you should put in as a pinch hitter, or deciding if you should bunt or try to get a hit. There are probably many other uses for these applications that I have not thought of, which I find very interesting and exciting. If this gets posted on the Cal Poly Statistics Department Shiny App page I would be interested to hear what the users want to get from their comparisons.

Bibliography

"Yogi Berra Quotes." *Baseball Almanac*. N.p., n.d. Web. 14 June 2017.

Sports Reference LLC. Baseball-Reference.com - Major League Statistics and Information. http://www.baseball-reference.com/. 10 June 2017

Asaro, VJ. “Markov League Baseball.” Senior project, California Polytechnic State University San Luis Obispo, 2016. DigitalCommons@Calpoly. Web. 14 June. 2017.

R/Shiny Batting Code

```
setwd("C:\\Users\\srodri40\\Desktop\\")

if (!require("shiny")) install.packages("shiny")
if (!require("xtable")) install.packages("xtable")
if (!require("ggplot2")) install.packages("ggplot2")
if (!require("plotrix")) install.packages("plotrix")
library(shiny)
library(ggplot2)
library(xtable)
library(plotrix)

data3=read.csv("ALL_Batting_2016.csv",header=T,as.is=T)[6:29]
data2=read.csv("ALL_Batting_2016.csv",header=T,as.is=T)[4]
data1=read.csv("ALL_Batting_2016.csv",header=T,as.is=T)[2]


data=cbind(data1,data2,data3)

#selecting all batters with more than 10
data= data[data$AB >=10,]

#Reading in teams, data name is selection name
Arizona_Diamondbacks=na.omit(data[data$Tm=="ARI",])
Atlanta_Braves=na.omit(data[data$Tm=="ATL",])
Baltimore_Orioles=na.omit(data[data$Tm=="BAL",])
Boston_Red_Sox=na.omit(data[data$Tm=="BOS",])
Chicago_Cubs=na.omit(data[data$Tm=="CHC",])
Chicago_White_Sox=na.omit(data[data$Tm=="CHW",])
Cincinnati_Reds=na.omit(data[data$Tm=="CIN",])
Cleveland_Indians=na.omit(data[data$Tm=="CLE",])
Colorado_Rockies=na.omit(data[data$Tm=="COL",])
Detroit_Tigers=na.omit(data[data$Tm=="DET",])
Houston_Astros=na.omit(data[data$Tm=="HOU",])
Kansas_City_Royals=na.omit(data[data$Tm=="KCR",])
Los_Angeles_Angels_of_Anaheim=na.omit(data[data$Tm=="LAA",])
Los_Angeles_Dodgers=na.omit(data[data$Tm=="LAD",])
Miami_Marlins=na.omit(data[data$Tm=="MIA",])
Milwaukee_Brewers=na.omit(data[data$Tm=="MIL",])
Minnesota_Twins=na.omit(data[data$Tm=="MIN",])
New_York_Mets=na.omit(data[data$Tm=="NYM",])
New_York_Yankees=na.omit(data[data$Tm=="NYY",])
Oakland_Athletics=na.omit(data[data$Tm=="OAK",])
Philadelphia_Phillies=na.omit(data[data$Tm=="PHI",])
Pittsburgh_Pirates=na.omit(data[data$Tm=="PIT",])
San_Diego_Padres=na.omit(data[data$Tm=="SDP",])
```

```
San_Francisco_Giants=na.omit(data[data$Tm=="SFG",])
Seattle_Mariners=na.omit(data[data$Tm=="SEA",])
St_Louis_Cardinals=na.omit(data[data$Tm=="STL",])
Tampa_Bay_Rays=na.omit(data[data$Tm=="TBR",])
Texas_Rangers=na.omit(data[data$Tm=="TEX",])
Toronto_Blue_Jays=na.omit(data[data$Tm=="TOR",])
Washington_Nationals=na.omit(data[data$Tm=="WSN",])


ui = bootstrapPage(
  titlePanel("MLB Player Batting 2016"),sidebarPanel(
  selectInput('team1', 'Teams:',
c('Arizona_Diamondbacks','Atlanta_Braves','Baltimore_Orioles','Boston_Red_Sox','Chicago_Cu
bs','Chicago_White_Sox','Cincinnati_Reds','Cleveland_Indians','Colorado_Rockies','Detroit_Tige
rs','Houston_Astros','Kansas_City_Royals','Los_Angeles_Angels_of_Anaheim','Los_Angeles_D
odgers','Miami_Marlins','Milwaukee_Brewers','Minnesota_Twins','New_York_Mets','New_York_
Yankees','Oakland_Athletics','Philadelphia_Phillies','Pittsburgh_Pirates','San_Diego_Padres','Sa
n_Francisco_Giants','Seattle_Mariners','St_Louis_Cardinals','Tampa_Bay_Rays','Texas_Ranger
s','Toronto_Blue_Jays','Washington_Nationals'),selected='Arizona_Diamondbacks'),
  selectInput('player1', 'First player:', 'Nick Ahmed' ),
  selectInput('team2', 'Teams:',
c('Arizona_Diamondbacks','Atlanta_Braves','Baltimore_Orioles','Boston_Red_Sox','Chicago_Cu
bs','Chicago_White_Sox','Cincinnati_Reds','Cleveland_Indians','Colorado_Rockies','Detroit_Tige
rs','Houston_Astros','Kansas_City_Royals','Los_Angeles_Angels_of_Anaheim','Los_Angeles_D
odgers','Miami_Marlins','Milwaukee_Brewers','Minnesota_Twins','New_York_Mets','New_York_
Yankees','Oakland_Athletics','Philadelphia_Phillies','Pittsburgh_Pirates','San_Diego_Padres','Sa
n_Francisco_Giants','Seattle_Mariners','St_Louis_Cardinals','Tampa_Bay_Rays','Texas_Ranger
s','Toronto_Blue_Jays','Washington_Nationals'),selected='Atlanta_Braves'),
  selectInput('player2', 'Second player:', 'Erick Aybar#'),
  h5("A * at the end of a players name means they bat left-handed"),
  h5("A # at the end of a players name means they are a switch hitter"),
  h5("Neither means they bat right-handed")),
  mainPanel(splitLayout(cellWidths = c("50%", "50%"), plotOutput("plot1"),
plotOutput("plot2")),
fluidRow(column(12,plotOutput("plot3"))),fluidRow(column(12,plotOutput("plot4"))))
)



server = function(input, output, session){
  #updates player selection based on team choice
  observeEvent(input$team1,{
    updateSelectInput(session, "player1",choices=as.character(get(input$team1)[,1]))
  })
  observeEvent(input$team2,{
    updateSelectInput(session, "player2",choices=as.character(get(input$team2)[,1]))
  })
```

```
            prob.batting= function(data){
              data$P.BB.HBP.= (data$BB+ data$HBP)/data$PA
              data$P.1B.= (data$H- data$X2B-data$X3B- data$HR)/data$PA
              data$P.2B.= data$X2B/data$PA
              data$P.3B.= data$X3B/data$PA
              data$P.HR.= data$HR/data$PA
              data$P.DP.= data$GDP/data$PA
              data$P.E.= (data$AB-data$SO)*(1-0.984)/data$PA
              data$P.Out.=
1-(data$P.BB.HBP.+data$P.1B.+data$P.2B.+data$P.3B.+data$P.HR.+data$P.E.)
              data$P.1Out.= data$P.Out.-data$P.DP.
              data$P.SF.= data$SF/data$PA

              # Extrabase is the probability of advancing from 2nd base to home on a single
              data$P.Att.2nd.0out. = 0.4673
              data$P.Att.2nd.1out. = 0.5927
              data$P.Att.2nd.2out. = 0.9079
              data$P.Safe.2nd.= 0.96

              prob= data[, c(1,27:40)]

              return(prob)
            }



            # End of prob function

            # trans function to compute the transition matrix and expected runs matrix
            trans.batting= function(prob,i){
              A0=
matrix(data=c(prob$P.HR.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$
P.3B.[i],0,0,0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],0,

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],prob$P.BB.HBP.[i],prob$P.1B.[i]*(1-prob$P.Att.2nd.0out.[i])
                    +prob$P.E.[i],0,0,

prob$P.HR.[i],prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$P.3B.[i],0,prob$P.BB.HBP.[i],0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.0out.
          [i])+prob$P.E.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],prob$P.BB.HBP.[i],
prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],0,prob$P.1B.[i]*(1-prob$P.Att.2nd.0out.[i])+prob$P.E.[i],0,prob
          $P.BB.HBP.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.0out.
          [i])+prob$P.E.[i]),
                      nrow=8, ncol=8, byrow=TRUE)

             A1=
matrix(data=c(prob$P.HR.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$
P.3B.[i],0,0,0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],0,

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],prob$P.BB.HBP.[i],prob$P.1B.[i]*(1-prob$P.Att.2nd.1out.[i])
                    +prob$P.E.[i],0,0,

prob$P.HR.[i],prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$P.3B.[i],0,prob$P.BB.HBP.[i],0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.1out.
          [i])+prob$P.E.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],prob$P.BB.HBP.[i],

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],0,prob$P.1B.[i]*(1-prob$P.Att.2nd.1out.[i])+prob$P.E.[i],0,prob
          $P.BB.HBP.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.1out.
          [i])+prob$P.E.[i]),
                      nrow=8, ncol=8, byrow=TRUE)


             A2=
matrix(data=c(prob$P.HR.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$
P.3B.[i],0,0,0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],0,

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],prob$P.BB.HBP.[i],prob$P.1B.[i]*(1-prob$P.Att.2nd.2out.[i])
                    +prob$P.E.[i],0,0,
prob$P.HR.[i],prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$P.3B.[i],0,prob$P.BB.HBP.[i],0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.2out.
        [i])+prob$P.E.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],prob$P.BB.HBP.[i],

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],0,prob$P.1B.[i]*(1-prob$P.Att.2nd.2out.[i])+prob$P.E.[i],0,prob
        $P.BB.HBP.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.2out.
        [i])+prob$P.E.[i]),
                nrow=8, ncol=8, byrow=TRUE)



          # creating B1 matrix: trans from 0 outs to 1 out
          B1= matrix(data=c(prob$P.Out.[i],0,0,0,0,0,0,0,
                    0,prob$P.1Out.[i],0,0,0,0,0,0,

0,prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i]),prob$P.Out.[i],0,0,0,0,0,
                    prob$P.SF.[i],0,0,prob$P.Out.[i]- prob$P.SF.[i],0,0,0,0,

0,0,0,0,prob$P.1Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out[i]*(1-prob$P.Safe.2nd.[i]),0,0,0,
                    0,prob$P.SF.[i],0,0,0,prob$P.1Out.[i]- prob$P.SF.[i],0,0,

0,prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i]),prob$P.SF.[i],0,0,0,prob$P.Out.[i]
- prob$P.SF.[i],0,

0,0,0,0,prob$P.SF.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i]),0,0,prob$P.1
Out.[i]- prob$P.SF.[i]),
                nrow=8, ncol=8, byrow=TRUE)

          # creating B2 matrix: trans from 1 out to 2 outs
          B2= matrix(data=c(prob$P.Out.[i],0,0,0,0,0,0,0,
                    0,prob$P.1Out.[i],0,0,0,0,0,0,

0,prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i]),prob$P.Out.[i],0,0,0,0,0,
                    prob$P.SF.[i],0,0,prob$P.Out.[i]- prob$P.SF.[i],0,0,0,0,

0,0,0,0,prob$P.1Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out[i]*(1-prob$P.Safe.2nd.[i]),0,0,0,
                    0,prob$P.SF.[i],0,0,0,prob$P.1Out.[i]- prob$P.SF.[i],0,0,
0,prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i]),prob$P.SF.[i],0,0,0,prob$P.Out.[i]
- prob$P.SF.[i],0,

0,0,0,0,prob$P.SF.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i]),0,0,prob$P.1
Out.[i]- prob$P.SF.[i]),
                    nrow=8, ncol=8, byrow=TRUE)

            # creating C2 matrix: trans from 0 outs to 2 outs
            C2= matrix(data=c(0,0,0,0,0,0,0,0,
                         prob$P.DP.[i],0,0,0,0,0,0,0,
                         0,0,0,0,0,0,0,0,
                         0,0,0,0,0,0,0,0,
                         0,0,0,prob$P.DP.[i],0,0,0,0,
                         prob$P.DP.[i],0,0,0,0,0,0,0,
                         0,0,0,0,0,0,0,0,
                         0,0,0,prob$P.DP.[i],0,0,0,0),
                    nrow=8, ncol=8, byrow=TRUE)

            # absorbing states
            D1= matrix(0, nrow=8, ncol=1)
            D2= matrix(data=c(0,prob$P.DP.[i],0,0,prob$P.DP.[i],prob$P.DP.[i],0,prob$P.DP.[i]),
nrow=8, ncol=1)
            D3= matrix(c(prob$P.Out.[i],
                      prob$P.Out.[i],
                      prob$P.Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out[i]*(1-prob$P.Safe.2nd.[i]),
                      prob$P.Out.[i],
                      prob$P.Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out[i]*(1-prob$P.Safe.2nd.[i]),
                      prob$P.Out.[i],
                      prob$P.Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out[i]*(1-prob$P.Safe.2nd.[i]),
                      prob$P.Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out[i]*(1-prob$P.Safe.2nd.[i])),
                    nrow=8, ncol=1)
            D4= matrix(1, nrow=1, ncol=1)

            abs0= matrix(c(0,0,0,0,0,0,0,0), nrow=1, ncol=8, byrow=TRUE)
            abs1= matrix(c(0,0,0,0,0,0,0,0), nrow=1, ncol=8, byrow=TRUE)
            abs2= matrix(c(0,0,0,0,0,0,0,0), nrow=1, ncol=8, byrow=TRUE)

            # zero matrices
            zero= matrix(0, nrow=8, ncol=8)

            # creating overall transisition matrix by combining above matrices
            trans=
matrix(c(rbind(A0,zero,zero,abs0),rbind(B1,A1,zero,abs1),rbind(C2,B2,A2,abs2),
rbind(D1,D2,D3,D4)),nrow=25, ncol=25,
                      dimnames= list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)",

"(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(12,1)","(13,1)","(23,1)","(123,1)",
"(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0
)","(12,0)","(13,0)","(23,0)",

"(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(12,1)","(13,1)","(23,1)","(123,1)",

"(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(x,3)")))

            R1=matrix(c(0,0,0,1*prob$P.HR.[i],0,
                     0,0,2*prob$P.HR.[i],1*prob$P.3B.[i],0,

0,0,2*prob$P.HR.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i]),
0,

0,0,2*prob$P.HR.[i],1*(prob$P.3B.[i]+prob$P.2B.[i]+prob$P.1B.[i]+prob$P.E.[i]+prob$P.SF.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P
.Safe.2nd.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.1B.[i]+prob$P.E.[i]+prob$P.SF.[i]),0,

0,3*prob$P.HR.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i]),1*
(prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.[i]*(1-prob$P.Att.2nd.0
out.[i])+prob$P.E.[i]+prob$P.SF.[i]),0,

4*prob$P.HR.[i],3*prob$P.3B.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.S
afe.2nd.[i]),
1*(prob$P.BB.HBP.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.
[i]*(1-prob$P.Att.2nd.0out.[i])+prob$P.E.[i]+prob$P.SF.[i]),0),
                    ncol=5,byrow=TRUE)

            R2=matrix(c(0,0,0,1*prob$P.HR.[i],0,
                     0,0,2*prob$P.HR.[i],1*prob$P.3B.[i],0,

0,0,2*prob$P.HR.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i]),
0,

0,0,2*prob$P.HR.[i],1*(prob$P.3B.[i]+prob$P.2B.[i]+prob$P.1B.[i]+prob$P.E.[i]+prob$P.SF.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P
.Safe.2nd.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.1B.[i]+prob$P.E.[i]+prob$P.SF.[i]),0,

0,3*prob$P.HR.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i]),1*
(prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.[i]*(1-prob$P.Att.2nd.1
out.[i])+prob$P.E.[i]+prob$P.SF.[i]),0,
4*prob$P.HR.[i],3*prob$P.3B.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.S
afe.2nd.[i]),
1*(prob$P.BB.HBP.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.
[i]*(1-prob$P.Att.2nd.1out.[i])+prob$P.E.[i]+prob$P.SF.[i]),0),
                    ncol=5,byrow=TRUE)

            R3=matrix(c(0,0,0,1*prob$P.HR.[i],0,
                     0,0,2*prob$P.HR.[i],1*prob$P.3B.[i],0,

0,0,2*prob$P.HR.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i]),
0,

0,0,2*prob$P.HR.[i],1*(prob$P.3B.[i]+prob$P.2B.[i]+prob$P.1B.[i]+prob$P.E.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P
.Safe.2nd.[i]),0,
                     0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.1B.[i]+prob$P.E.[i]),0,

0,3*prob$P.HR.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i]),1*
(prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.[i]*(1-prob$P.Att.2nd.2
out.[i])+prob$P.E.[i]),0,

4*prob$P.HR.[i],3*prob$P.3B.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.S
afe.2nd.[i]),
1*(prob$P.BB.HBP.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.
[i]*(1-prob$P.Att.2nd.2out.[i])+prob$P.E.[i]),0),
                    ncol=5,byrow=TRUE)

            Rscore= matrix(rbind(R1,R2,R3),nrow=24,ncol=5)
            eruns=matrix(rowSums(Rscore), nrow=24, dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)"),c(
"Exp Runs for Half-Inn")))

            list(trans,eruns)
          }
          # End of trans function



          # Expected runs matrix function team 1
          Mat.Exp = function(data){
            require(plotrix)
            color2D.matplot(data, show.values=3, axes=FALSE, xlab=paste0(input$player1,"
Expected Runs"), ylab="Base(s) Occupied")
            axis(side=2, at=7.5:0.5, labels=c("None","1","2","3","12","13","23","123"), las=1)
            axis(side=3, at=0.5:2.5, labels=c("0","1","2"))
            mtext(text="", side=2, line=2, cex.lab=1)
```

```
    mtext(text="Outs", side=3, line=2, cex.lab=1)
  }

  # Expected runs matrix function team 2
  Mat.Exp.Runs = function(data){
    require(plotrix)
    color2D.matplot(data, show.values=3, axes=FALSE, xlab=paste0(input$player2,"
Expected Runs"), ylab="Base(s) Occupied")
    axis(side=2, at=7.5:0.5, labels=c("None","1","2","3","12","13","23","123"), las=1)
    axis(side=3, at=0.5:2.5, labels=c("0","1","2"))
    mtext(text="", side=2, line=2, cex.lab=1)
    mtext(text="Outs", side=3, line=2, cex.lab=1)
  }


  output$plot1 <- renderPlot({
    prob.batting1=prob.batting(get(input$team1))
    nplayers.batting1 = nrow(prob.batting1)
    trans.store.batting1=array(NaN,c(25,25,nplayers.batting1),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
    R.store.batting1=array(NaN,c(24,5,nplayers.batting1))
    Exp.runs.store.batting1=matrix(NaN,nrow=24,ncol=nplayers.batting1)

    # player by player expected runs calculation
    for (i in 1: nplayers.batting1){
      temp.batting=trans.batting(prob.batting1,i)
      temp.trans.batting=temp.batting[[1]]
      temp.eruns.batting=temp.batting[[2]]

      # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
      I=diag(24)
      Q=temp.trans.batting[-25,-25]
      E=solve(I-Q)

      # Expected Runs for rest of inning at starting state: mult by 9 for full game
      Exp.Runs= E%*%temp.eruns.batting
      Nine.inn=Exp.Runs*9
      Nine.inn

      trans.store.batting1[,,i]=temp.trans.batting
      R.store.batting1[,,i]=temp.eruns.batting
      Exp.runs.store.batting1[,i]=Nine.inn
    }
```

```
            compare.batting1=cbind(get(input$team1)[,c(1,4)],Exp.runs.store.batting1[1,],
get(input$team1)$R/162, get(input$team1)$R)
            compare.batting1$Exp.runs.162.batting= Exp.runs.store.batting1[1,]*162
            compare.batting1$Percent.change=
round(((compare.batting1$Exp.runs.162.batting-get(input$team1)$R)/compare.batting1$Exp.run
s.162.batting)*100, 3)

            #matching player index from team data to input name
            index1=which(get(input$team1)$Name==input$player1)

player1.eruns=matrix(cbind(Exp.runs.store.batting1[,index1]/9),nrow=8,ncol=3,dimnames=list(c(
"0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
            Mat.Exp(player1.eruns)
          })

          output$plot2 <- renderPlot({
            prob.batting2=prob.batting(get(input$team2))
            nplayers.batting2 = nrow(prob.batting2)
            trans.store.batting2=array(NaN,c(25,25,nplayers.batting2),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
            R.store.batting2=array(NaN,c(24,5,nplayers.batting2))
            Exp.runs.store.batting2=matrix(NaN,nrow=24,ncol=nplayers.batting2)

            # player by player expected runs calculation
            for (i in 1: nplayers.batting2){
              temp.batting=trans.batting(prob.batting2,i)
              temp.trans.batting=temp.batting[[1]]
              temp.eruns.batting=temp.batting[[2]]

              # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
              I=diag(24)
              Q=temp.trans.batting[-25,-25]
              E=solve(I-Q)

              # Expected Runs for rest of inning at starting state: mult by 9 for full game
              Exp.Runs= E%*%temp.eruns.batting
              Nine.inn=Exp.Runs*9
              Nine.inn

              trans.store.batting2[,,i]=temp.trans.batting
              R.store.batting2[,,i]=temp.eruns.batting
              Exp.runs.store.batting2[,i]=Nine.inn
```

```
        }

        compare.batting2=cbind(get(input$team2)[,c(1,4)],Exp.runs.store.batting2[1,],
get(input$team2)$R/162, get(input$team2)$R)
        compare.batting2$Exp.runs.162.batting= Exp.runs.store.batting2[1,]*162
        compare.batting2$Percent.change=
round(((compare.batting2$Exp.runs.162.batting-get(input$team2)$R)/compare.batting2$Exp.run
s.162.batting)*100, 3)

        #matching player index from team data to input name
        index2=which(get(input$team2)$Name==input$player2)

player2.eruns=matrix(cbind(Exp.runs.store.batting2[,index2]/9),nrow=8,ncol=3,dimnames=list(c(
"0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
        Mat.Exp.Runs(player2.eruns)
       })


       output$plot3 <- renderPlot({

        prob.batting1=prob.batting(get(input$team1))
        nplayers.batting1 = nrow(prob.batting1)
        trans.store.batting1=array(NaN,c(25,25,nplayers.batting1),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
        R.store.batting1=array(NaN,c(24,5,nplayers.batting1))
        Exp.runs.store.batting1=matrix(NaN,nrow=24,ncol=nplayers.batting1)

        for (i in 1: nplayers.batting1){
         temp.batting=trans.batting(prob.batting1,i)
         temp.trans.batting=temp.batting[[1]]
         temp.eruns.batting=temp.batting[[2]]

         # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
         I=diag(24)
         Q=temp.trans.batting[-25,-25]
         E=solve(I-Q)

         # Expected Runs for rest of inning at starting state: mult by 9 for full game
         Exp.Runs= E%*%temp.eruns.batting
         Nine.inn=Exp.Runs*9
         Nine.inn

         trans.store.batting1[,,i]=temp.trans.batting
```

```
                R.store.batting1[,,i]=temp.eruns.batting
                Exp.runs.store.batting1[,i]=Nine.inn

              }


              compare.batting1=cbind(get(input$team1)[,c(1,4)],Exp.runs.store.batting1[1,],
get(input$team1)$R/162, get(input$team1)$R)
              compare.batting1$Exp.runs.162.batting= Exp.runs.store.batting1[1,]*162
              compare.batting1$Percent.change=
round(((compare.batting1$Exp.runs.162.batting-get(input$team1)$R)/compare.batting1$Exp.run
s.162.batting)*100, 3)

              #matching player index from team data to input name
              index1=which(get(input$team1)$Name==input$player1)

player1.eruns=matrix(cbind(Exp.runs.store.batting1[,index1]/9),nrow=8,ncol=3,dimnames=list(c(
"0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
              p1.eruns=as.vector(player1.eruns)



              prob.batting2=prob.batting(get(input$team2))
              nplayers.batting2 = nrow(prob.batting2)
              trans.store.batting2=array(NaN,c(25,25,nplayers.batting2),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
              R.store.batting2=array(NaN,c(24,5,nplayers.batting2))
              Exp.runs.store.batting2=matrix(NaN,nrow=24,ncol=nplayers.batting2)

              for (i in 1: nplayers.batting2){
                temp.batting=trans.batting(prob.batting2,i)
                temp.trans.batting=temp.batting[[1]]
                temp.eruns.batting=temp.batting[[2]]

                # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
                I=diag(24)
                Q=temp.trans.batting[-25,-25]
                E=solve(I-Q)

                # Expected Runs for rest of inning at starting state: mult by 9 for full game
                Exp.Runs= E%*%temp.eruns.batting
                Nine.inn=Exp.Runs*9
                Nine.inn
```

```
        trans.store.batting2[,,i]=temp.trans.batting
        R.store.batting2[,,i]=temp.eruns.batting
        Exp.runs.store.batting2[,i]=Nine.inn

      }


      compare.batting2=cbind(get(input$team2)[,c(1,4)],Exp.runs.store.batting2[1,],
get(input$team2)$R/162, get(input$team2)$R)
      compare.batting2$Exp.runs.162.batting= Exp.runs.store.batting2[1,]*162
      compare.batting2$Percent.change=
round(((compare.batting2$Exp.runs.162.batting-get(input$team2)$R)/compare.batting2$Exp.run
s.162.batting)*100, 3)

      #matching player index from team data to input name
      index2=which(get(input$team2)$Name==input$player2)

player2.eruns=matrix(cbind(Exp.runs.store.batting2[,index2]/9),nrow=8,ncol=3,dimnames=list(c(
"0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
      p2.eruns=as.vector(player2.eruns)

      #creating expected runs plot
      plot(c(1,24),c(min(p1.eruns,p2.eruns),max(p1.eruns,p2.eruns)),xaxt="n",
type="n",xlab="Base State",ylab="Expected Runs",las=3,main="Expected Runs for each Base
State")
      lines(p1.eruns,col="red")
      lines(p2.eruns,col="blue")

lablist.x=as.vector(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)",
"(2,1)","(3,1)","(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,
2)","(123,2)"))
      axis(1, at=1:24,labels=F)
      text(x = seq(.6, 23.6, by=1), par("usr")[3]-.15, labels = lablist.x, srt = 85, pos =1, xpd =
T)

legend("topright",c(input$player1,input$player2),lty=c(1,1),lwd=c(2.5,2.5),col=c("red","blue"))
     })

     output$plot4 <- renderPlot({

      prob.batting1=prob.batting(get(input$team1))
      nplayers.batting1 = nrow(prob.batting1)
      trans.store.batting1=array(NaN,c(25,25,nplayers.batting1),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
```

```
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
            R.store.batting1=array(NaN,c(24,5,nplayers.batting1))
            Exp.runs.store.batting1=matrix(NaN,nrow=24,ncol=nplayers.batting1)

            for (i in 1: nplayers.batting1){
              temp.batting=trans.batting(prob.batting1,i)
              temp.trans.batting=temp.batting[[1]]
              temp.eruns.batting=temp.batting[[2]]

              # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
              I=diag(24)
              Q=temp.trans.batting[-25,-25]
              E=solve(I-Q)

              # Expected Runs for rest of inning at starting state: mult by 9 for full game
              Exp.Runs= E%*%temp.eruns.batting
              Nine.inn=Exp.Runs*9
              Nine.inn

              trans.store.batting1[,,i]=temp.trans.batting
              R.store.batting1[,,i]=temp.eruns.batting
              Exp.runs.store.batting1[,i]=Nine.inn

            }


            compare.batting1=cbind(get(input$team1)[,c(1,4)],Exp.runs.store.batting1[1,],
get(input$team1)$R/162, get(input$team1)$R)
            compare.batting1$Exp.runs.162.batting= Exp.runs.store.batting1[1,]*162
            compare.batting1$Percent.change=
round(((compare.batting1$Exp.runs.162.batting-get(input$team1)$R)/compare.batting1$Exp.run
s.162.batting)*100, 3)

            #matching player index from team data to input name
            index1=which(get(input$team1)$Name==input$player1)

player1.eruns=matrix(cbind(Exp.runs.store.batting1[,index1]/9),nrow=8,ncol=3,dimnames=list(c(
"0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
            p1.eruns=as.vector(player1.eruns)



            prob.batting2=prob.batting(get(input$team2))
            nplayers.batting2 = nrow(prob.batting2)
            trans.store.batting2=array(NaN,c(25,25,nplayers.batting2),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
            R.store.batting2=array(NaN,c(24,5,nplayers.batting2))
            Exp.runs.store.batting2=matrix(NaN,nrow=24,ncol=nplayers.batting2)

            for (i in 1: nplayers.batting2){
              temp.batting=trans.batting(prob.batting2,i)
              temp.trans.batting=temp.batting[[1]]
              temp.eruns.batting=temp.batting[[2]]

              # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
              I=diag(24)
              Q=temp.trans.batting[-25,-25]
              E=solve(I-Q)

              # Expected Runs for rest of inning at starting state: mult by 9 for full game
              Exp.Runs= E%*%temp.eruns.batting
              Nine.inn=Exp.Runs*9
              Nine.inn

              trans.store.batting2[,,i]=temp.trans.batting
              R.store.batting2[,,i]=temp.eruns.batting
              Exp.runs.store.batting2[,i]=Nine.inn

            }


            compare.batting2=cbind(get(input$team2)[,c(1,4)],Exp.runs.store.batting2[1,],
get(input$team2)$R/162, get(input$team2)$R)
            compare.batting2$Exp.runs.162.batting= Exp.runs.store.batting2[1,]*162
            compare.batting2$Percent.change=
round(((compare.batting2$Exp.runs.162.batting-get(input$team2)$R)/compare.batting2$Exp.run
s.162.batting)*100, 3)

            #matching player index from team data to input name
            index2=which(get(input$team2)$Name==input$player2)

player2.eruns=matrix(cbind(Exp.runs.store.batting2[,index2]/9),nrow=8,ncol=3,dimnames=list(c(
"0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
            p2.eruns=as.vector(player2.eruns)

            #creating the difference in each base state between players
            diff=NULL
            for(i in 1:24){
              diff[i]=p1.eruns[i]-p2.eruns[i]
```

```
            }

            #creating expected runs plot
            plot(c(1,9.25),c(min(diff),max(diff)),xaxt="n", type="n",xlab="Base
State",ylab="Difference in Expected Runs",las=3,main=c("Difference in Expected Runs for each
Base State", "(if positive then first player has higher expected runs for that state)"))
            abline(h=0,v=0,col="gray60")
            lines(diff[1:8],col="red")
            lines(diff[9:16],col="blue")
            lines(diff[17:24],col="green")
            axis(1, at=1:8,labels=c(0,1,2,3,12,13,23,123))
            legend("topright",c("0 Outs","1 Out","2
Outs"),lty=c(1,1),lwd=c(2.5,2.5),col=c("red","blue","green"))
          })

        }

        shinyApp(ui=ui, server=server)
```

R/Shiny Pitching Code

```
setwd("C:\\Users\\srodri40\\Desktop\\")

if (!require("shiny")) install.packages("shiny")
if (!require("xtable")) install.packages("xtable")
if (!require("ggplot2")) install.packages("ggplot2")
if (!require("plotrix")) install.packages("plotrix")
library(shiny)
library(ggplot2)
library(xtable)
library(plotrix)

#Reading in teams, data name is selection name
data3=read.csv("ALL_PITCHERS_2016.csv",header=T,as.is=T)[7:29]
data2=read.csv("ALL_PITCHERS_2016.csv",header=T,as.is=T)[4:5]
data1=read.csv("ALL_PITCHERS_2016.csv",header=T,as.is=T)[2]

data=cbind(data1,data2,data3)
length(data)
#selecting all pitchers with more than 3 innings pitched
data= data[data$IP >=3,]

#Reading in teams, data name is selection name
Arizona_Diamondbacks=na.omit(data[data$Tm=="ARI",])
Atlanta_Braves=na.omit(data[data$Tm=="ATL",])
Baltimore_Orioles=na.omit(data[data$Tm=="BAL",])
Boston_Red_Sox=na.omit(data[data$Tm=="BOS",])
Chicago_Cubs=na.omit(data[data$Tm=="CHC",])
Chicago_White_Sox=na.omit(data[data$Tm=="CHW",])
Cincinnati_Reds=na.omit(data[data$Tm=="CIN",])
Cleveland_Indians=na.omit(data[data$Tm=="CLE",])
Colorado_Rockies=na.omit(data[data$Tm=="COL",])
Detroit_Tigers=na.omit(data[data$Tm=="DET",])
Houston_Astros=na.omit(data[data$Tm=="HOU",])
Kansas_City_Royals=na.omit(data[data$Tm=="KCR",])
Los_Angeles_Angels_of_Anaheim=na.omit(data[data$Tm=="LAA",])
Los_Angeles_Dodgers=na.omit(data[data$Tm=="LAD",])
Miami_Marlins=na.omit(data[data$Tm=="MIA",])
Milwaukee_Brewers=na.omit(data[data$Tm=="MIL",])
Minnesota_Twins=na.omit(data[data$Tm=="MIN",])
New_York_Mets=na.omit(data[data$Tm=="NYM",])
New_York_Yankees=na.omit(data[data$Tm=="NYY",])
Oakland_Athletics=na.omit(data[data$Tm=="OAK",])
Philadelphia_Phillies=na.omit(data[data$Tm=="PHI",])
Pittsburgh_Pirates=na.omit(data[data$Tm=="PIT",])
San_Diego_Padres=na.omit(data[data$Tm=="SDP",])
```

```
        San_Francisco_Giants=na.omit(data[data$Tm=="SFG",])
        Seattle_Mariners=na.omit(data[data$Tm=="SEA",])
        St_Louis_Cardinals=na.omit(data[data$Tm=="STL",])
        Tampa_Bay_Rays=na.omit(data[data$Tm=="TBR",])
        Texas_Rangers=na.omit(data[data$Tm=="TEX",])
        Toronto_Blue_Jays=na.omit(data[data$Tm=="TOR",])
        Washington_Nationals=na.omit(data[data$Tm=="WSN",])



        ui = bootstrapPage(
          titlePanel("MLB Player Pitching 2016"),sidebarPanel(
          selectInput('team1', 'Teams:',
c('Arizona_Diamondbacks','Atlanta_Braves','Baltimore_Orioles','Boston_Red_Sox','Chicago_Cu
bs','Chicago_White_Sox','Cincinnati_Reds','Cleveland_Indians','Colorado_Rockies','Detroit_Tige
rs','Houston_Astros','Kansas_City_Royals','Los_Angeles_Angels_of_Anaheim','Los_Angeles_D
odgers','Miami_Marlins','Milwaukee_Brewers','Minnesota_Twins','New_York_Mets','New_York_
Yankees','Oakland_Athletics','Philadelphia_Phillies','Pittsburgh_Pirates','San_Diego_Padres','Sa
n_Francisco_Giants','Seattle_Mariners','St_Louis_Cardinals','Tampa_Bay_Rays','Texas_Ranger
s','Toronto_Blue_Jays','Washington_Nationals'),selected='Arizona_Diamondbacks'),
          selectInput('player1', 'First player:', 'Jake Barrett' ),
          selectInput('team2', 'Teams:',
c('Arizona_Diamondbacks','Atlanta_Braves','Baltimore_Orioles','Boston_Red_Sox','Chicago_Cu
bs','Chicago_White_Sox','Cincinnati_Reds','Cleveland_Indians','Colorado_Rockies','Detroit_Tige
rs','Houston_Astros','Kansas_City_Royals','Los_Angeles_Angels_of_Anaheim','Los_Angeles_D
odgers','Miami_Marlins','Milwaukee_Brewers','Minnesota_Twins','New_York_Mets','New_York_
Yankees','Oakland_Athletics','Philadelphia_Phillies','Pittsburgh_Pirates','San_Diego_Padres','Sa
n_Francisco_Giants','Seattle_Mariners','St_Louis_Cardinals','Tampa_Bay_Rays','Texas_Ranger
s','Toronto_Blue_Jays','Washington_Nationals'),selected='Atlanta_Braves'),
          selectInput('player2', 'Second player:', 'Dario Alvarez*'),
          h5("A * at the end of a players name means they pitch left-handed"),
          h5("A # at the end of a players name means they are a switch pitcher"),
          h5("Neither means they pitch right-handed")),
          mainPanel(splitLayout(cellWidths = c("50%", "50%"), plotOutput("plot1"),
plotOutput("plot2")),
fluidRow(column(12,plotOutput("plot3"))),fluidRow(column(12,plotOutput("plot4"))))
        )



        server = function(input, output, session){
          #updates player selection based on team choice
          observeEvent(input$team1,{
            updateSelectInput(session, "player1",choices=as.character(get(input$team1)[,1]))
          })
          observeEvent(input$team2,{
            updateSelectInput(session, "player2",choices=as.character(get(input$team2)[,1]))
          })


          prob.pitching= function(data){
            data$P.BB.HBP.= (data$BB+ data$HBP)/data$PA
            data$P.1B.= (data$H- data$X2B-data$X3B- data$HR)/data$PA
            data$P.2B.= data$X2B/data$PA
            data$P.3B.= data$X3B/data$PA
            data$P.HR.= data$HR/data$PA
            data$P.DP.= data$GDP/data$PA
            data$P.E.= (data$AB-data$SO)*(1-0.984)/data$PA
            data$P.Out.=
1-(data$P.BB.HBP.+data$P.1B.+data$P.2B.+data$P.3B.+data$P.HR.+data$P.E.)
            data$P.1Out.= data$P.Out.-data$P.DP.
            data$P.SF.= data$SF/data$PA

            # Extrabase is the probability of advancing from 2nd base to home on a single
            data$P.Att.2nd.0out. = 0.4673
            data$P.Att.2nd.1out. = 0.5927
            data$P.Att.2nd.2out. = 0.9079
            data$P.Safe.2nd.= 0.96

            prob= data[, c(1,27:40)]

            return(prob)
          }



          # End of prob function

          # trans function to compute the transition matrix and expected runs matrix
          trans.pitching= function(prob,i){
            A0=
matrix(data=c(prob$P.HR.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$
P.3B.[i],0,0,0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],0,

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],prob$P.BB.HBP.[i],prob$P.1B.[i]*(1-prob$P.Att.2nd.0out.[i])
                    +prob$P.E.[i],0,0,

prob$P.HR.[i],prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$P.3B.[i],0,prob$P.BB.HBP.[i],0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.0out.
        [i])+prob$P.E.[i],
prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],prob$P.BB.HBP.[i],

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],0,prob$P.1B.[i]*(1-prob$P.Att.2nd.0out.[i])+prob$P.E.[i],0,prob
          $P.BB.HBP.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.0out.
          [i])+prob$P.E.[i]),
                      nrow=8, ncol=8, byrow=TRUE)

             A1=
matrix(data=c(prob$P.HR.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$
P.3B.[i],0,0,0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],0,

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],prob$P.BB.HBP.[i],prob$P.1B.[i]*(1-prob$P.Att.2nd.1out.[i])
                    +prob$P.E.[i],0,0,

prob$P.HR.[i],prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$P.3B.[i],0,prob$P.BB.HBP.[i],0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.1out.
          [i])+prob$P.E.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],prob$P.BB.HBP.[i],

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],0,prob$P.1B.[i]*(1-prob$P.Att.2nd.1out.[i])+prob$P.E.[i],0,prob
          $P.BB.HBP.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.1out.
          [i])+prob$P.E.[i]),
                      nrow=8, ncol=8, byrow=TRUE)


             A2=
matrix(data=c(prob$P.HR.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$
P.3B.[i],0,0,0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],0,

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
```

```
.[i],prob$P.BB.HBP.[i],prob$P.1B.[i]*(1-prob$P.Att.2nd.2out.[i])
                    +prob$P.E.[i],0,0,

prob$P.HR.[i],prob$P.1B.[i]+prob$P.E.[i],prob$P.2B.[i],prob$P.3B.[i],0,prob$P.BB.HBP.[i],0,0,

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.2out.
          [i])+prob$P.E.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]+prob$P.E.[i],0,prob$P.2B.[i],prob$P.BB.HBP.[i],

prob$P.HR.[i],prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i],prob$P.2B.[i],prob$P.3B
.[i],0,prob$P.1B.[i]*(1-prob$P.Att.2nd.2out.[i])+prob$P.E.[i],0,prob
          $P.BB.HBP.[i],

prob$P.HR.[i],0,0,prob$P.3B.[i],prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i],0,prob
$P.2B.[i],prob$P.BB.HBP.[i]+prob$P.1B.[i]*(1-prob$P.Att.2nd.2out.
          [i])+prob$P.E.[i]),
                      nrow=8, ncol=8, byrow=TRUE)



             # creating B1 matrix: trans from 0 outs to 1 out
             B1= matrix(data=c(prob$P.Out.[i],0,0,0,0,0,0,0,
                          0,prob$P.1Out.[i],0,0,0,0,0,0,

0,prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i]),prob$P.Out.[i],0,0,0,0,0,
                          prob$P.SF.[i],0,0,prob$P.Out.[i]- prob$P.SF.[i],0,0,0,0,

0,0,0,0,prob$P.1Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out[i]*(1-prob$P.Safe.2nd.[i]),0,0,0,
                          0,prob$P.SF.[i],0,0,0,prob$P.1Out.[i]- prob$P.SF.[i],0,0,

0,prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i]),prob$P.SF.[i],0,0,0,prob$P.Out.[i]
- prob$P.SF.[i],0,

0,0,0,0,prob$P.SF.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i]),0,0,prob$P.1
Out.[i]- prob$P.SF.[i]),
                      nrow=8, ncol=8, byrow=TRUE)

             # creating B2 matrix: trans from 1 out to 2 outs
             B2= matrix(data=c(prob$P.Out.[i],0,0,0,0,0,0,0,
                          0,prob$P.1Out.[i],0,0,0,0,0,0,

0,prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i]),prob$P.Out.[i],0,0,0,0,0,
                          prob$P.SF.[i],0,0,prob$P.Out.[i]- prob$P.SF.[i],0,0,0,0,

0,0,0,0,prob$P.1Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out[i]*(1-prob$P.Safe.2nd.[i]),0,0,0,
                          0,prob$P.SF.[i],0,0,0,prob$P.1Out.[i]- prob$P.SF.[i],0,0,
0,prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i]),prob$P.SF.[i],0,0,0,prob$P.Out.[i]
- prob$P.SF.[i],0,

0,0,0,0,prob$P.SF.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i]),0,0,prob$P.1
Out.[i]- prob$P.SF.[i]),
                    nrow=8, ncol=8, byrow=TRUE)

            # creating C2 matrix: trans from 0 outs to 2 outs
            C2= matrix(data=c(0,0,0,0,0,0,0,0,
                          prob$P.DP.[i],0,0,0,0,0,0,0,
                          0,0,0,0,0,0,0,0,
                          0,0,0,0,0,0,0,0,
                          0,0,0,prob$P.DP.[i],0,0,0,0,
                          prob$P.DP.[i],0,0,0,0,0,0,0,
                          0,0,0,0,0,0,0,0,
                          0,0,0,prob$P.DP.[i],0,0,0,0),
                    nrow=8, ncol=8, byrow=TRUE)

            # absorbing states
            D1= matrix(0, nrow=8, ncol=1)
            D2= matrix(data=c(0,prob$P.DP.[i],0,0,prob$P.DP.[i],prob$P.DP.[i],0,prob$P.DP.[i]),
nrow=8, ncol=1)
            D3= matrix(c(prob$P.Out.[i],
                      prob$P.Out.[i],
                      prob$P.Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out[i]*(1-prob$P.Safe.2nd.[i]),
                      prob$P.Out.[i],
                      prob$P.Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out[i]*(1-prob$P.Safe.2nd.[i]),
                      prob$P.Out.[i],
                      prob$P.Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out[i]*(1-prob$P.Safe.2nd.[i]),
                      prob$P.Out.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out[i]*(1-prob$P.Safe.2nd.[i])),
                    nrow=8, ncol=1)
            D4= matrix(1, nrow=1, ncol=1)

            abs0= matrix(c(0,0,0,0,0,0,0,0), nrow=1, ncol=8, byrow=TRUE)
            abs1= matrix(c(0,0,0,0,0,0,0,0), nrow=1, ncol=8, byrow=TRUE)
            abs2= matrix(c(0,0,0,0,0,0,0,0), nrow=1, ncol=8, byrow=TRUE)

            # zero matrices
            zero= matrix(0, nrow=8, ncol=8)

            # creating overall transisition matrix by combining above matrices
            trans=
matrix(c(rbind(A0,zero,zero,abs0),rbind(B1,A1,zero,abs1),rbind(C2,B2,A2,abs2),
rbind(D1,D2,D3,D4)),nrow=25, ncol=25,
                     dimnames= list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)",

"(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(12,1)","(13,1)","(23,1)","(123,1)",
"(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0
)","(12,0)","(13,0)","(23,0)",

"(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(12,1)","(13,1)","(23,1)","(123,1)",

"(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(x,3)")))

            R1=matrix(c(0,0,0,1*prob$P.HR.[i],0,
                     0,0,2*prob$P.HR.[i],1*prob$P.3B.[i],0,

0,0,2*prob$P.HR.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i]),
0,

0,0,2*prob$P.HR.[i],1*(prob$P.3B.[i]+prob$P.2B.[i]+prob$P.1B.[i]+prob$P.E.[i]+prob$P.SF.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P
.Safe.2nd.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.1B.[i]+prob$P.E.[i]+prob$P.SF.[i]),0,

0,3*prob$P.HR.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.Safe.2nd.[i]),1*
(prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.[i]*(1-prob$P.Att.2nd.0
out.[i])+prob$P.E.[i]+prob$P.SF.[i]),0,

4*prob$P.HR.[i],3*prob$P.3B.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*prob$P.S
afe.2nd.[i]),
1*(prob$P.BB.HBP.[i]+prob$P.1B.[i]*prob$P.Att.2nd.0out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.
[i]*(1-prob$P.Att.2nd.0out.[i])+prob$P.E.[i]+prob$P.SF.[i]),0),
                  ncol=5,byrow=TRUE)

            R2=matrix(c(0,0,0,1*prob$P.HR.[i],0,
                     0,0,2*prob$P.HR.[i],1*prob$P.3B.[i],0,

0,0,2*prob$P.HR.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i]),
0,

0,0,2*prob$P.HR.[i],1*(prob$P.3B.[i]+prob$P.2B.[i]+prob$P.1B.[i]+prob$P.E.[i]+prob$P.SF.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P
.Safe.2nd.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.1B.[i]+prob$P.E.[i]+prob$P.SF.[i]),0,

0,3*prob$P.HR.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.Safe.2nd.[i]),1*
(prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.[i]*(1-prob$P.Att.2nd.1
out.[i])+prob$P.E.[i]+prob$P.SF.[i]),0,
4*prob$P.HR.[i],3*prob$P.3B.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*prob$P.S
afe.2nd.[i]),
1*(prob$P.BB.HBP.[i]+prob$P.1B.[i]*prob$P.Att.2nd.1out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.
[i]*(1-prob$P.Att.2nd.1out.[i])+prob$P.E.[i]+prob$P.SF.[i]),0),
                    ncol=5,byrow=TRUE)

            R3=matrix(c(0,0,0,1*prob$P.HR.[i],0,
                     0,0,2*prob$P.HR.[i],1*prob$P.3B.[i],0,

0,0,2*prob$P.HR.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i]),
0,

0,0,2*prob$P.HR.[i],1*(prob$P.3B.[i]+prob$P.2B.[i]+prob$P.1B.[i]+prob$P.E.[i]),0,

0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P
.Safe.2nd.[i]),0,
                     0,3*prob$P.HR.[i],2*prob$P.3B.[i],1*(prob$P.1B.[i]+prob$P.E.[i]),0,

0,3*prob$P.HR.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.Safe.2nd.[i]),1*
(prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.[i]*(1-prob$P.Att.2nd.2
out.[i])+prob$P.E.[i]),0,

4*prob$P.HR.[i],3*prob$P.3B.[i],2*(prob$P.2B.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*prob$P.S
afe.2nd.[i]),
1*(prob$P.BB.HBP.[i]+prob$P.1B.[i]*prob$P.Att.2nd.2out.[i]*(1-prob$P.Safe.2nd.[i])+prob$P.1B.
[i]*(1-prob$P.Att.2nd.2out.[i])+prob$P.E.[i]),0),
                    ncol=5,byrow=TRUE)

            Rscore= matrix(rbind(R1,R2,R3),nrow=24,ncol=5)
            eruns=matrix(rowSums(Rscore), nrow=24, dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)"),c(
"Exp Runs for Half-Inn")))

            list(trans,eruns)
          }
          # End of trans function



          # Expected runs matrix function team 1
          Mat.Exp = function(data){
            require(plotrix)
            color2D.matplot(data, show.values=3, axes=FALSE, xlab=paste0(input$player1,"
Expected Runs Allowed"), ylab="Base(s) Occupied")
            axis(side=2, at=7.5:0.5, labels=c("None","1","2","3","12","13","23","123"), las=1)
            axis(side=3, at=0.5:2.5, labels=c("0","1","2"))
            mtext(text="", side=2, line=2, cex.lab=1)
```

```
    mtext(text="Outs", side=3, line=2, cex.lab=1)
  }

  # Expected runs matrix function team 2
  Mat.Exp.Runs = function(data){
    require(plotrix)
    color2D.matplot(data, show.values=3, axes=FALSE, xlab=paste0(input$player2,"
Expected Runs Allowed"), ylab="Base(s) Occupied")
    axis(side=2, at=7.5:0.5, labels=c("None","1","2","3","12","13","23","123"), las=1)
    axis(side=3, at=0.5:2.5, labels=c("0","1","2"))
    mtext(text="", side=2, line=2, cex.lab=1)
    mtext(text="Outs", side=3, line=2, cex.lab=1)
  }


  output$plot1 <- renderPlot({
    prob.pitching1=prob.pitching(get(input$team1))
    nplayers.pitching1 = nrow(prob.pitching1)
    trans.store.pitching1=array(NaN,c(25,25,nplayers.pitching1),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
    R.store.pitching1=array(NaN,c(24,5,nplayers.pitching1))
    Exp.runs.store.pitching1=matrix(NaN,nrow=24,ncol=nplayers.pitching1)

    # player by player expected runs calculation
    for (i in 1: nplayers.pitching1){
      temp.pitching=trans.pitching(prob.pitching1,i)
      temp.trans.pitching=temp.pitching[[1]]
      temp.eruns.pitching=temp.pitching[[2]]

      # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
      I=diag(24)
      Q=temp.trans.pitching[-25,-25]
      E=solve(I-Q)

      # Expected Runs for rest of inning at starting state: mult by 9 for full game
      Exp.Runs= E%*%temp.eruns.pitching
      Nine.inn=Exp.Runs*9
      Nine.inn

      trans.store.pitching1[,,i]=temp.trans.pitching
      R.store.pitching1[,,i]=temp.eruns.pitching
      Exp.runs.store.pitching1[,i]=Nine.inn
    }
```

```
            compare.pitching1=cbind(get(input$team1)[,c(1,4)],Exp.runs.store.pitching1[1,],
get(input$team1)$R/162, get(input$team1)$R)
            compare.pitching1$Exp.runs.162.pitching= Exp.runs.store.pitching1[1,]*162
            compare.pitching1$Percent.change=
round(((compare.pitching1$Exp.runs.162.pitching-get(input$team1)$R)/compare.pitching1$Exp.
runs.162.pitching)*100, 3)

            #matching player index from team data to input name
            index1=which(get(input$team1)$Name==input$player1)

player1.eruns=matrix(cbind(Exp.runs.store.pitching1[,index1]/9),nrow=8,ncol=3,dimnames=list(c
("0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
            Mat.Exp(player1.eruns)
          })

          output$plot2 <- renderPlot({
            prob.pitching2=prob.pitching(get(input$team2))
            nplayers.pitching2 = nrow(prob.pitching2)
            trans.store.pitching2=array(NaN,c(25,25,nplayers.pitching2),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
            R.store.pitching2=array(NaN,c(24,5,nplayers.pitching2))
            Exp.runs.store.pitching2=matrix(NaN,nrow=24,ncol=nplayers.pitching2)

            # player by player expected runs calculation
            for (i in 1: nplayers.pitching2){
              temp.pitching=trans.pitching(prob.pitching2,i)
              temp.trans.pitching=temp.pitching[[1]]
              temp.eruns.pitching=temp.pitching[[2]]

              # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
              I=diag(24)
              Q=temp.trans.pitching[-25,-25]
              E=solve(I-Q)

              # Expected Runs for rest of inning at starting state: mult by 9 for full game
              Exp.Runs= E%*%temp.eruns.pitching
              Nine.inn=Exp.Runs*9
              Nine.inn

              trans.store.pitching2[,,i]=temp.trans.pitching
              R.store.pitching2[,,i]=temp.eruns.pitching
              Exp.runs.store.pitching2[,i]=Nine.inn
```

```
        }

        compare.pitching2=cbind(get(input$team2)[,c(1,4)],Exp.runs.store.pitching2[1,],
get(input$team2)$R/162, get(input$team2)$R)
        compare.pitching2$Exp.runs.162.pitching= Exp.runs.store.pitching2[1,]*162
        compare.pitching2$Percent.change=
round(((compare.pitching2$Exp.runs.162.pitching-get(input$team2)$R)/compare.pitching2$Exp.
runs.162.pitching)*100, 3)

        #matching player index from team data to input name
        index2=which(get(input$team2)$Name==input$player2)

player2.eruns=matrix(cbind(Exp.runs.store.pitching2[,index2]/9),nrow=8,ncol=3,dimnames=list(c
("0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
        Mat.Exp.Runs(player2.eruns)
       })


       output$plot3 <- renderPlot({

        prob.pitching1=prob.pitching(get(input$team1))
        nplayers.pitching1 = nrow(prob.pitching1)
        trans.store.pitching1=array(NaN,c(25,25,nplayers.pitching1),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
        R.store.pitching1=array(NaN,c(24,5,nplayers.pitching1))
        Exp.runs.store.pitching1=matrix(NaN,nrow=24,ncol=nplayers.pitching1)

        for (i in 1: nplayers.pitching1){
         temp.pitching=trans.pitching(prob.pitching1,i)
         temp.trans.pitching=temp.pitching[[1]]
         temp.eruns.pitching=temp.pitching[[2]]

         # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
         I=diag(24)
         Q=temp.trans.pitching[-25,-25]
         E=solve(I-Q)

         # Expected Runs for rest of inning at starting state: mult by 9 for full game
         Exp.Runs= E%*%temp.eruns.pitching
         Nine.inn=Exp.Runs*9
         Nine.inn

         trans.store.pitching1[,,i]=temp.trans.pitching
```

```
            R.store.pitching1[,,i]=temp.eruns.pitching
            Exp.runs.store.pitching1[,i]=Nine.inn

          }


          compare.pitching1=cbind(get(input$team1)[,c(1,4)],Exp.runs.store.pitching1[1,],
get(input$team1)$R/162, get(input$team1)$R)
          compare.pitching1$Exp.runs.162.pitching= Exp.runs.store.pitching1[1,]*162
          compare.pitching1$Percent.change=
round(((compare.pitching1$Exp.runs.162.pitching-get(input$team1)$R)/compare.pitching1$Exp.
runs.162.pitching)*100, 3)

          #matching player index from team data to input name
          index1=which(get(input$team1)$Name==input$player1)

player1.eruns=matrix(cbind(Exp.runs.store.pitching1[,index1]/9),nrow=8,ncol=3,dimnames=list(c
("0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
          p1.eruns=as.vector(player1.eruns)



          prob.pitching2=prob.pitching(get(input$team2))
          nplayers.pitching2 = nrow(prob.pitching2)
          trans.store.pitching2=array(NaN,c(25,25,nplayers.pitching2),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
          R.store.pitching2=array(NaN,c(24,5,nplayers.pitching2))
          Exp.runs.store.pitching2=matrix(NaN,nrow=24,ncol=nplayers.pitching2)

          for (i in 1: nplayers.pitching2){
            temp.pitching=trans.pitching(prob.pitching2,i)
            temp.trans.pitching=temp.pitching[[1]]
            temp.eruns.pitching=temp.pitching[[2]]

            # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
            I=diag(24)
            Q=temp.trans.pitching[-25,-25]
            E=solve(I-Q)

            # Expected Runs for rest of inning at starting state: mult by 9 for full game
            Exp.Runs= E%*%temp.eruns.pitching
            Nine.inn=Exp.Runs*9
            Nine.inn
```

```
              trans.store.pitching2[,,i]=temp.trans.pitching
              R.store.pitching2[,,i]=temp.eruns.pitching
              Exp.runs.store.pitching2[,i]=Nine.inn

            }


            compare.pitching2=cbind(get(input$team2)[,c(1,4)],Exp.runs.store.pitching2[1,],
get(input$team2)$R/162, get(input$team2)$R)
            compare.pitching2$Exp.runs.162.pitching= Exp.runs.store.pitching2[1,]*162
            compare.pitching2$Percent.change=
round(((compare.pitching2$Exp.runs.162.pitching-get(input$team2)$R)/compare.pitching2$Exp.
runs.162.pitching)*100, 3)

            #matching player index from team data to input name
            index2=which(get(input$team2)$Name==input$player2)

player2.eruns=matrix(cbind(Exp.runs.store.pitching2[,index2]/9),nrow=8,ncol=3,dimnames=list(c
("0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
            p2.eruns=as.vector(player2.eruns)

            #creating expected runs plot
            plot(c(1,24),c(min(p1.eruns,p2.eruns),max(p1.eruns,p2.eruns)),xaxt="n",
type="n",xlab="Base State",ylab="Expected Runs Allowed",las=3,main="Expected Runs
Allowed for each Base State")
            lines(p1.eruns,col="red")
            lines(p2.eruns,col="blue")

lablist.x=as.vector(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)",
"(2,1)","(3,1)","(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,
2)","(123,2)"))
            axis(1, at=1:24,labels=F)
            text(x = seq(.6, 23.6, by=1), par("usr")[3]-.15, labels = lablist.x, srt = 85, pos =1, xpd =
TRUE)

legend("topright",c(input$player1,input$player2),lty=c(1,1),lwd=c(2.5,2.5),col=c("red","blue"))
          })

          output$plot4 <- renderPlot({

            prob.pitching1=prob.pitching(get(input$team1))
            nplayers.pitching1 = nrow(prob.pitching1)
            trans.store.pitching1=array(NaN,c(25,25,nplayers.pitching1),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
```

```
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
            R.store.pitching1=array(NaN,c(24,5,nplayers.pitching1))
            Exp.runs.store.pitching1=matrix(NaN,nrow=24,ncol=nplayers.pitching1)

            for (i in 1: nplayers.pitching1){
              temp.pitching=trans.pitching(prob.pitching1,i)
              temp.trans.pitching=temp.pitching[[1]]
              temp.eruns.pitching=temp.pitching[[2]]

              # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
              I=diag(24)
              Q=temp.trans.pitching[-25,-25]
              E=solve(I-Q)

              # Expected Runs for rest of inning at starting state: mult by 9 for full game
              Exp.Runs= E%*%temp.eruns.pitching
              Nine.inn=Exp.Runs*9
              Nine.inn

              trans.store.pitching1[,,i]=temp.trans.pitching
              R.store.pitching1[,,i]=temp.eruns.pitching
              Exp.runs.store.pitching1[,i]=Nine.inn

            }


            compare.pitching1=cbind(get(input$team1)[,c(1,4)],Exp.runs.store.pitching1[1,],
get(input$team1)$R/162, get(input$team1)$R)
            compare.pitching1$Exp.runs.162.pitching= Exp.runs.store.pitching1[1,]*162
            compare.pitching1$Percent.change=
round(((compare.pitching1$Exp.runs.162.pitching-get(input$team1)$R)/compare.pitching1$Exp.
runs.162.pitching)*100, 3)

            #matching player index from team data to input name
            index1=which(get(input$team1)$Name==input$player1)

player1.eruns=matrix(cbind(Exp.runs.store.pitching1[,index1]/9),nrow=8,ncol=3,dimnames=list(c
("0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
            p1.eruns=as.vector(player1.eruns)



            prob.pitching2=prob.pitching(get(input$team2))
            nplayers.pitching2 = nrow(prob.pitching2)
            trans.store.pitching2=array(NaN,c(25,25,nplayers.pitching2),dimnames=
list(c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)","(
12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)","(
x,3)"),c("(0,0)","(1,0)","(2,0)","(3,0)","(12,0)","(13,0)","(23,0)","(123,0)","(0,1)","(1,1)","(2,1)","(3,1)"
,"(12,1)","(13,1)","(23,1)","(123,1)","(0,2)","(1,2)","(2,2)","(3,2)","(12,2)","(13,2)","(23,2)","(123,2)",
"(x,3)")))
            R.store.pitching2=array(NaN,c(24,5,nplayers.pitching2))
            Exp.runs.store.pitching2=matrix(NaN,nrow=24,ncol=nplayers.pitching2)

            for (i in 1: nplayers.pitching2){
              temp.pitching=trans.pitching(prob.pitching2,i)
              temp.trans.pitching=temp.pitching[[1]]
              temp.eruns.pitching=temp.pitching[[2]]

              # creating E: rowsums(E)= expected number of batters at each starting state for the
inning
              I=diag(24)
              Q=temp.trans.pitching[-25,-25]
              E=solve(I-Q)

              # Expected Runs for rest of inning at starting state: mult by 9 for full game
              Exp.Runs= E%*%temp.eruns.pitching
              Nine.inn=Exp.Runs*9
              Nine.inn

              trans.store.pitching2[,,i]=temp.trans.pitching
              R.store.pitching2[,,i]=temp.eruns.pitching
              Exp.runs.store.pitching2[,i]=Nine.inn

            }


            compare.pitching2=cbind(get(input$team2)[,c(1,4)],Exp.runs.store.pitching2[1,],
get(input$team2)$R/162, get(input$team2)$R)
            compare.pitching2$Exp.runs.162.pitching= Exp.runs.store.pitching2[1,]*162
            compare.pitching2$Percent.change=
round(((compare.pitching2$Exp.runs.162.pitching-get(input$team2)$R)/compare.pitching2$Exp.
runs.162.pitching)*100, 3)

            #matching player index from team data to input name
            index2=which(get(input$team2)$Name==input$player2)

player2.eruns=matrix(cbind(Exp.runs.store.pitching2[,index2]/9),nrow=8,ncol=3,dimnames=list(c
("0","1","2","3","12","13","23","123"),c("0 OUTS","1 OUT","2 OUTS")))
            p2.eruns=as.vector(player2.eruns)

            #creating the difference in each base state between players
            diff=NULL
            for(i in 1:24){
              diff[i]=p1.eruns[i]-p2.eruns[i]
```

```
            }

            #creating expected runs plot
            plot(c(1,9.25),c(min(diff),max(diff)),xaxt="n", type="n",xlab="Base
State",ylab="Difference in Expected Runs Allowed",las=3,main=c("Difference in Expected Runs
Allowed for each Base State", "(if positive then first player has higher expected runs allowed for
that state)"))
            abline(h=0,v=0,col="gray60")
            lines(diff[1:8],col="red")
            lines(diff[9:16],col="blue")
            lines(diff[17:24],col="green")
            axis(1, at=1:8,labels=c(0,1,2,3,12,13,23,123))
            legend("topright",c("0 Outs","1 Out","2
Outs"),lty=c(1,1),lwd=c(2.5,2.5),col=c("red","blue","green"))
           })

         }

         shinyApp(ui=ui, server=server)
```