Decision Making in Baseball:
A Computer Programming Approach

Summer Research Participant: Tim David
Summer Research Faculty Mentor: Chris Jones

School of Science
Saint Mary's College of California
Summer Research 2010

Introduction

Baseball is a game of numbers. While other sports track statistics such as points scored, assists, and minutes played, baseball documents nearly every action that takes place such as batting average, slugging percentage, and earned run average. Statistics are prevalent in baseball because they are used primarily as predictors. Over a hundred years of statistics have shown us statistical norms as well as regression indicators within the game of baseball. Teams that implement statistical research are at an advantage in the professional ranks, as many professional baseball teams continually hire team statisticians and analysts. However, not much research has been put into amateur leagues. Statistical analysis allows professional coaches to optimize line-ups, scout players, and make in-game managerial decisions. The success of statistics with professional coaches indicates that amateur coaches could benefit from them as well.

Some college coaches do their best to use their own statistics, but without organizations or researchers following closely, it becomes a burden. As a result, most amateur coaches go with their gut instincts. There is nothing inherently wrong with this approach, as most coaches have had years of both playing and coaching experience, but there is a better way to approach the situation. We believe that there is a way to bring high-level statistics and decision-making tools to amateur coaches, and it can be as easy as keeping score. Most coaches keep their scorebook with them throughout the game anyway. However, the goal of our project was to digitize this process, as well as allow these digital stats to easily translate into an aid for amateur coaches -- a tool that they can use in order to make smarter decisions, as well as predict, with fairly good accuracy, the outcome resulting from their decisions.

In order to be practical, this program needed to be on something portable that could allow coaches to bring it to their games as they would a scorebook. It had to be as simple to use and access as possible or else it would not get used because it would not be practical because most decisions need to be made quickly. The iPad platform was an ideal solution.

Background

There have been areas of our research that people have touched on, but it has never been centralized. Score-keeping capabilities already exist through apps such as GameChanger^{TM} and ESPN's iScore^{TM}. Both of these apps track baseball games allow the user to track the game as it progresses, storing the statistics in a user friendly database. Additionally, several statistical predictors have been developed through Sabermertics (Society for American Baseball Research). These statistics aim to remove error from classic baseball statistics which have been shown to be poor indicators of skill. Sabermetric statistics focus in on the areas that professional baseball players can control. Without the element of luck, Sabermetrics allows statisticians to better predict the outcome of baseball events. Although both of these capabilities are vastly important in what we are trying to do, no one has created a program that tracks stats and then instantly sends the info that is gathered to compute the probability of certain situations occurring. The goal of our project was for a coach to be able to track the game they are in and, based on the situation of the game and the current players, be able to see what affect certain managerial decisions would have on the outcome of the game, such as bunting, base stealing, or substitutions. As a result, managers could instantaneously opt for the decision that gives them the highest percentage of winning the ballgame. Additionally, this type of research has not been done with amateur athletes in mind. Score-keeping and statistical databases have been developed before, but not in the way that we are proposing.

Resources and Method

In order to study the problem, we had to gain an understanding of a wide range of topics. First, we had to understand basic and advanced baseball statistics, such as batting average, slugging percentage, and on base percentage. In addition, we also had to learn advanced baseball statistics such as OBP, OPS qERA, and SIERA, as well as the significance and statistical relevance of each. Each statistic has its own metric, its own expected values, and its own regression rate. Sabermetrics was essential to this research because it is the most up to date research in baseball statistics and provides the most relevant outcomes. In addition to Sabermetric knowledge, we also had to learn how to utilize the iPhone Software Development Kit (SDK) that is essential in developing applications for the iPad. Unique to Apple iOS products, the SDK includes a programming platform, C++, as well as a graphic user interface (GUI) that they call the interface builder. Coming in with very little knowledge of computer programming, we had to learn all of the language before we could begin the project. We also utilized the iPhone SDK portion of the Apple website that provides programming instructions as well as tutorials that helped us get started with our project.

Results

In the given 10 weeks, we were not able to reach our goals, but we made great progress. We created five separate programs, that each contributed to what we were hoping to accomplish. Here are the five programs, listed by the names that we have given them:

Baseball Counter

The first program that we wrote attempted to track basic game functions. Tracking the game is a vital step of our overall goal, as the ability to make a decision is dependent upon the situation in the game. Additionally, tracking the game will eventually allow us to automatically add statistics to our individual player database. In this program, we were able to keep track of balls, strikes and outs for the given at bat, as well as the total number of balls and strikes thrown by the current pitcher. The program also has the ability to clear the total number of pitches thrown in preparation for a new pitcher, as well as the ability to clear all statistics in the event of a new game. We also designed the program so that all of these values could be inputted manually if the user should make a mistake. The ability to count balls and strikes was a simple counting system. Whenever the user clicks on the button labeled “Balls” or “Strikes”, the program adds one to the total in the text field above (See Figure 1). These counters continue to increase until “Strikes” reaches three or “Balls” reaches four. At this point, both counters revert back to zero, as the at bat is over. Additionally, if the program reaches three “Strikes” before four “Balls,” then the program adds one to the box labeled “Outs.” In a similar fashion, as soon as “Outs” reaches three, its respective counter reverts back to zero. The program also has the ability to track the total number of strikes, balls, and overall pitches thrown by the current pitcher. This function is extremely similar to that of the “Balls” and “Strikes” counter, except there is no limit on how many pitches that can be thrown. The only way that the total number of strikes, balls, and overall pitches will revert to zero is if the user clicks the “New Pitcher” button. Until the “New Pitcher” button is pressed, every time that the user presses the “Strikes,” “Balls,” or “Foul” buttons, it adds to the total number of pitches thrown and its respective result of strike or ball. The “Foul” button adds to the total number of strikes thrown, the total number of pitches thrown, and the total number of strikes, as long as the number of strikes within the at bat is zero or one. If the number of strikes within the at bat is two, clicking foul will not increase the total, it will only add to total strikes thrown and total pitches thrown.

Strike Zone

In addition to tracking balls and strikes, we also wanted to track pitch location. Statistics can help to predict what pitch type and pitch location a hitter can expect in a certain situation based on historical trends and patterns. To best predict pitch type and location, we first had to develop a way to record them. The strike zone program is set up with a visual display of a strike zone (See Figure 2). For simplicity, we designed the zone to have twenty-five boxes arranged in a five by five square. The sixteen boxes on the border of the zone represent balls, while the nine inner boxes represent strikes. Using a counting formula similar to the one used to track the total number of pitches thrown, we were able to track total balls and strikes from this program. We developed the program so that coaches can watch the pitch thrown in the game and then determine its location. Then the user will go to the program and click on the box that best represents where the pitch was thrown. If the selected box is within the strike zone, then the program adds a strike and if it is outside of the strike zone, then it adds a ball. Regardless of which box the user selects, immediately after a box is clicked, a menu labeled pitch type appears. On this menu, four standard pitches appear: fastball, change-up, curve-ball and slider. Similar to balls and strikes in this program, when the user selects which pitch type was thrown, one gets added to its total at the bottom of the page.

Markov Chain

One of the major statistical predictors in baseball is the Markov Chain. These chains can be utilized when a process can be separated into a finite number of states. An inning of baseball satisfies the needs of a Markov Chain. In every inning, there are twenty-five possible “states” or situations with outs and runners. All of the possible offensive situations revolve around a runner’s occupancy of bases. There can be a runner at first, second, third, first and second, first and third, second and third, every base, or no runners on base. Each of these situations can occur with zero, one, or two outs. Additionally, there is the final state for every inning, which is three outs and no runners on base. The Markov Chain becomes a predictor when they examine the transition between these states. An example of a transition between states would be going from a runner on first with one out to a runner on first with two outs. By inputting the statistics of individual players into this twenty-five by twenty-five matrix, we were able to calculate the probability of all possible outcomes of an at bat (See Figure 3). Multiplying these matrices by themselves an infinite amount of times allows us to predict the number of runs a team would score per inning if it was made up of nine of the same player. This number is a good indication of the ability of the hitter.

In addition to the Markov Chain, this program also calculates other baseball statistics that can be used as predictors. The first four, BA, OBP, SLG, and OPS, are standard hitting statistics used in Major League Baseball. Batting Average (BA), is calculated by taking a hitter’s total number of hits and dividing by at bats. It essentially calculates how often a player gets a hit. Slugging Percentage (SLG) is similar to BA, but it is the total number of bases a hitter gains per at bat. For example, if a player hits a double, he would gain two total bases. On Base Percentage (OBP) is the total number of times a hitter reaches based divided by total plate appearances, which appears as follows:

$$(H+BB+HBP)/(AB+BB+HBP+SF)$$

On Base Plus Slugging (OPS) is simply the addition of OPS and SLG. Lastly, we calculated Runs Created (RC), which is an advanced statistic used in Sabermetrics. According to The Harball Times, RC "is a very good measure of the number of runs a batter truly contributed to his team's offense." The formula we used for RC is as follows:

$$\frac{(H+BB+HBP-CS-GIDP)\times(TB+0.26(BB-IBB+HBP)+0.52(SH+SF+SB)}{(AB+BB+HBP+SH+SF)}$$

Pitching Metric

Similar to hitting statistics, we also calculated relevant pitching statistics (See Figure 4). The first two, Earned Run Average (ERA) and Walks and Hits per Innings Pitched (WHIP), are standard statistics in Major League Baseball. WHIP is derived by taking all walks and hits accrued by a pitcher and dividing by their total number of innings pitched. ERA is derived by multiplying a pitcher's total number of earned runs by nine and dividing by their total number of innings pitched. We also calculated Strike Out to Walk Ratio (K/BB) and Strikeouts Per Nine Innings (K/9), which are not as commonly used, but are simple to calculate. Lastly, we calculated two stats, SIERA and qERA, that are not commonly used, but are highly praised in the Sabermetric community. According to Baseball Prospectus, Skill-Interactive Earned Run Average (SIERA) is "an estimate that more accurately gauges the run-prevention skills of a pitcher relative to his controllable skills." The formula for SIERA is as follows:

$$6.22-18.05\times(SO/PA)+11.292\times(BB/PA)-1.721\times((GB-FB-PU)/PA)+10.169\times((SO\times PA)^2)$$

$$-7.069\times(((GB-FB-PU)/PA)^2)+9.561\times(SO/PA)-4.027\times(BB/PA)\times((GB-FB-PU)/PA)$$

According to Baseball Prospectus, qERA "estimates what a pitcher's ERA should be based solely on his strikeout rate, walk rate, and his groundball to flyball ratio." The formula for qERA is as follows:

$$(2.69+K\%\times(-3.4)+BB\%\times 3.88+GB\%\times(-.66))^2$$

There are several more predictive statistical available through Baseball Prospectus, The Hardball Times, and other baseball research sites, but we felt that these were the most relevant pitching statistics for the purposes of our project.

Core Data

Another necessary component of our overall goals was data storage. While we had developed many counters and calculators, until we could persist data, we would have to manually input all of our statistics. The main method of data persistence in the Apple iOS is Core Data. Using Core Data, we began to create a database that we hoped to be able to integrate with each of our individual programs. With Core Data, we were able to create a simple, user friendly database, but we have yet to discover a way to use it with our other programs. The main page of the database has a list of players, sorted alphabetically and arranged by team (See Figure 5). Clicking on the name of the player takes the user to the player information page (See Figure 6). This page lists all of the statistics that are relevant to the Markov Hitting Chain as well as all of the pitching metrics. It is simple to manually add a new player from the main page as well, where the user can input all of the necessary statistical information. Our goal was for this player information page to connect with the Baseball Counter and Strike Zone pages so that players could have their statistics updated automatically. Additionally, we hoped that selecting a player would automatically input his statistics into either the Markov Chain Program or the Pitching Metric Program.

Conclusion

Although we made great progress in our allotted ten weeks of research, there is still a lot to be done with each of our programs in order to reach our goals. The first program that could use improvement is the Baseball Counter. Although it is a basic program and serves its purpose well, it will need a few additions once everything else is developed. We will need the program to track innings because certain situations depend upon leverage and the situation in the game. For example, some decisions, such as bunting, are more appropriate later in a game than earlier. In addition, we will have to develop a way to track the results of each at bat. As the program functions now, it only records outs, regardless of how they occurred. Developing the program so that it includes results will allow us to update player databases automatically. In order to do this, we will also have to develop a way to indicate which player is at bat or on the pitcher's mound so that we know who to credit with the statistics.

We were also hoping to add a feature to the Strike Zone Program as well. While this page is mostly complete, we were hoping to add the occurrence of pitch types in specific counts. With this feature, a batter could go to the database and see what the opposing pitcher tends to throw when he has, for example, two balls and two strikes on the hitter.

We were also working on connectivity between pages. As I mentioned earlier, we would like to be able to program a line-up into the Baseball Counter Program and the Strike Zone Program so that they could actively add statistics to our database.

Lastly, we would have liked to implement the decision making aspect of our program. However, with only ten weeks, the majority of our time was devoted to developing the preliminary programs that were necessary in order to develop the decision making portion. When we make the necessary changes to our basic programs, we will be able to implement the decision making aspect of the program.

Overall, in ten weeks we accomplished quite a bit. We developed a greater understanding for computer programming an addition to developing five separate programs that we hope to be able to integrate in the near future. While there is still work to do, we made great strides in our ten weeks of summer research.

Works Cited

"Glossary." *Baseball Prospectus*. Web. 13 Sept. 2010. <http://www.baseballprospectus.com/glossary/>.

Goldstein, Neal, and Tony Bove. *IPad Application Development for Dummies*. Hoboken, NJ: Wiley Pub., 2010. Print.

The Hardball Times. "THT Glossary." *The Hardball Times*. Web. 13 Sept. 2010. <http://www.hardballtimes.com/main/statpages/glossary/>.

"Introducing SIERA." *Baseball Prospectus*. Web. 13 Sept. 2010. <http://www.baseballprospectus.com/article.php?articleid=10032>.

"IOS Dev Center." *Apple Developer*. Web. 13 Sept. 2010. <http://developer.apple.com/devcenter/ios/index.action>.

Mark, Dave, and Jeff LaMarche. *Beginning IPhone 3 Development: Exploring the IPhone SDK*. Berkeley, CA: Apress, 2009. Print.

Appendix I: Statistics Glossary

<u>Recorded Statistics</u>:
AB: At Bats
PA: Plate Appearances
SO: Strike Outs
BB: Walks
HBP: Hit By Pitch
TB: Total Bases
SF: Sacrifice Fly
SH: Safcrifice Hit
GB: Ground Ball
FB: Fly Ball
PU: Pop Up
SB: Stolen Bases
CS: Caught Stealing
GIDP: Gets In Double Play

<u>Calculated Hitting Statistics</u>:
BA: (H/AB)
OBP: $(H+BB+HBP)/(AB+BB+HBP+SF)$
SLG: (TB/AB)
OPS: $(OBP+SLG)$
RC: $$\frac{(H+BB+HBP-CS-GIDP)\times(TB+0.26(BB-IBB+HBP)+0.52(SH+SF+SB)}{(AB+BB+HBP+SH+SF)}$$

<u>Calculated Pitching Statistics</u>:

ERA: $\frac{(ER\times 9)}{IP}$

WHIP: $\frac{(BB+H)}{IP}$

QERA: $(2.69+K\%\times(-3.4)+BB\%\times 3.88+GB\%\times(-.66))^2$

K/9: $\frac{(SO\times 9)}{IP}$

K/BB: SO/BB

SIERA: $$6.22-18.05\times(SO/PA)+11.292\times(BB/PA)-1.721\times((GB-FB-PU)/PA)+10.169\times((SO\times PA)^2)$$
$$-7.069\times(((GB-FB-PU)/PA)^2)+9.561\times(SO/PA)-4.027\times(BB/PA)\times((GB-FB-PU)/PA)$$

Appendix II: Screen Shots

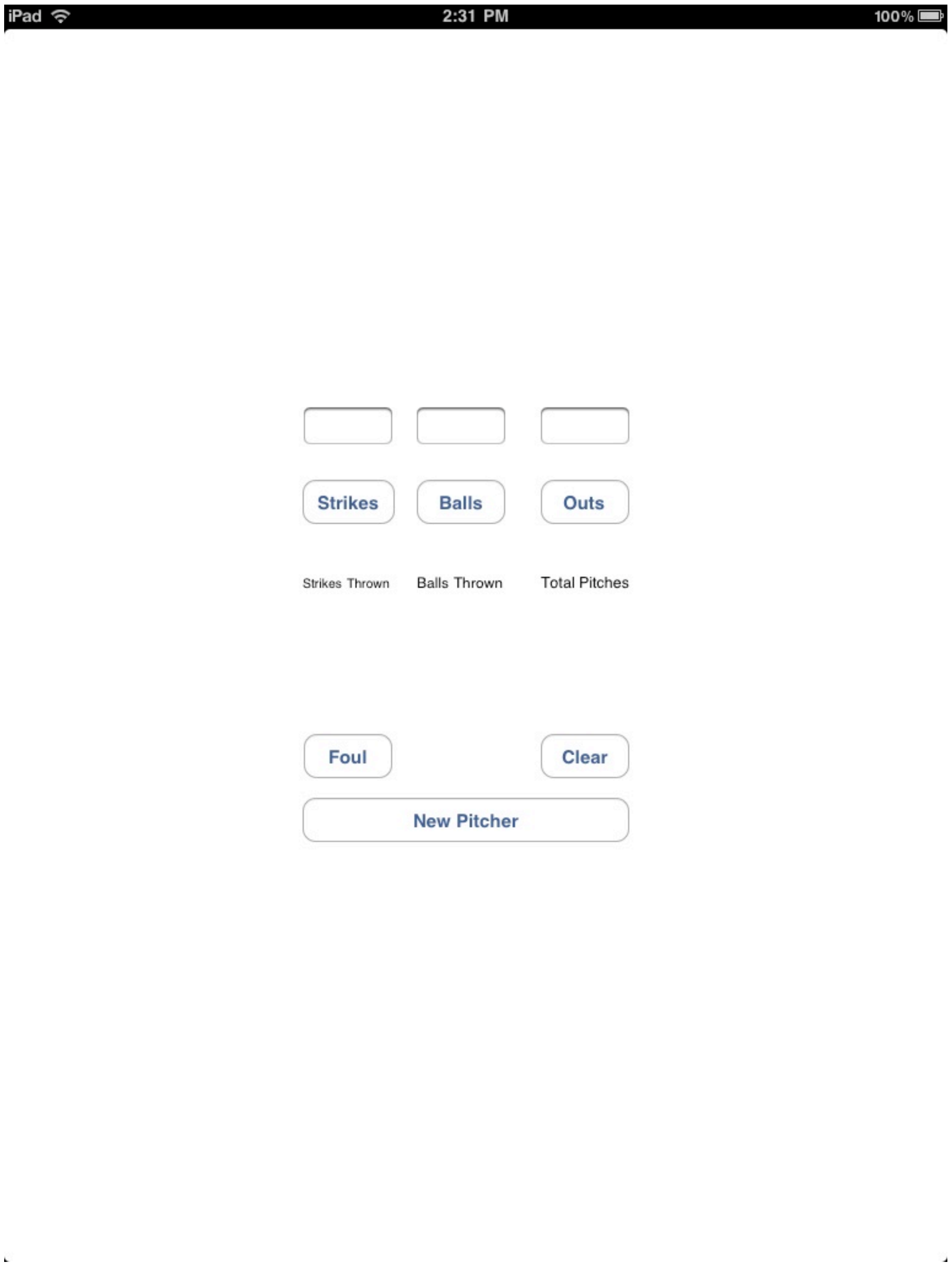
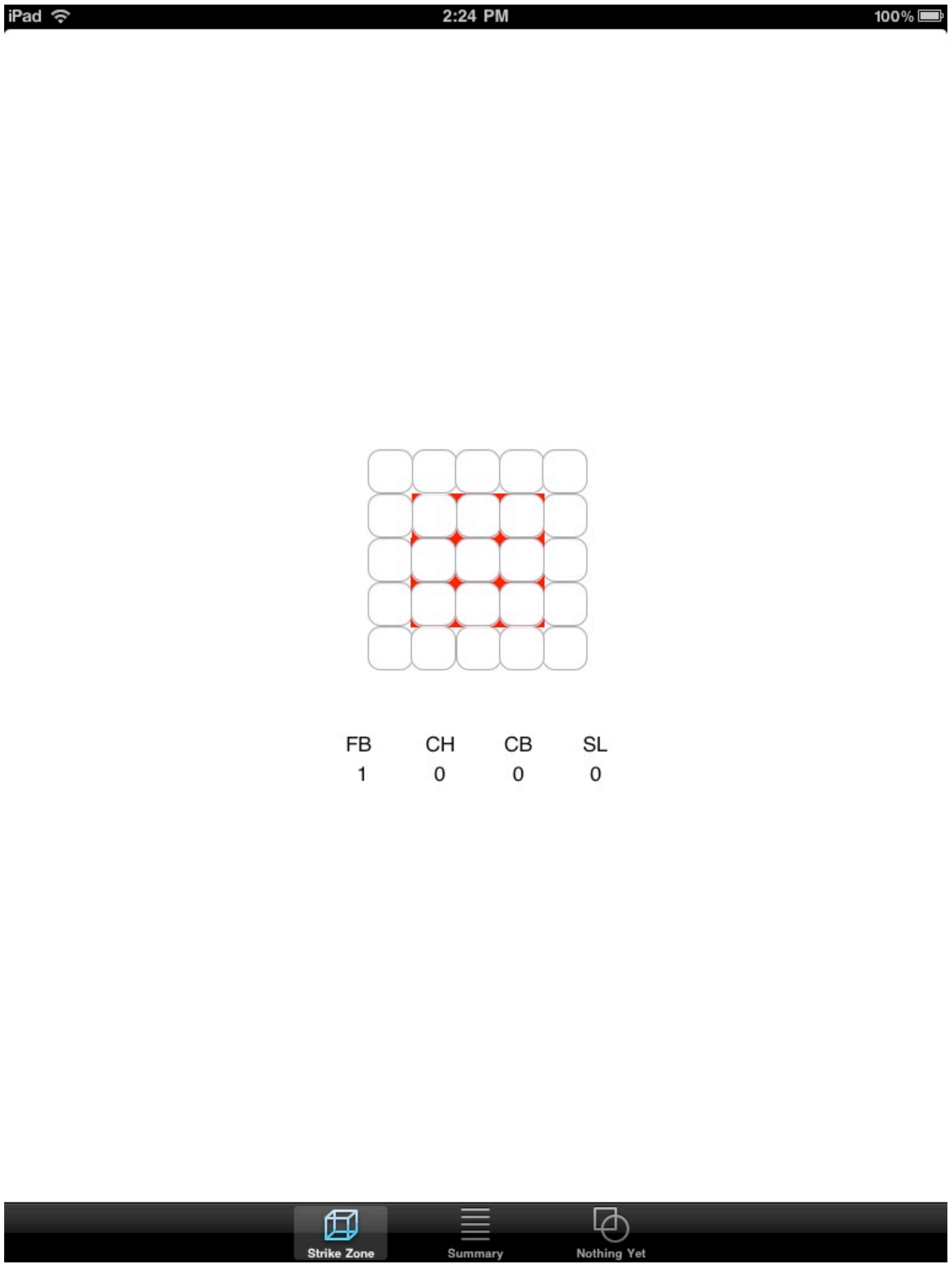
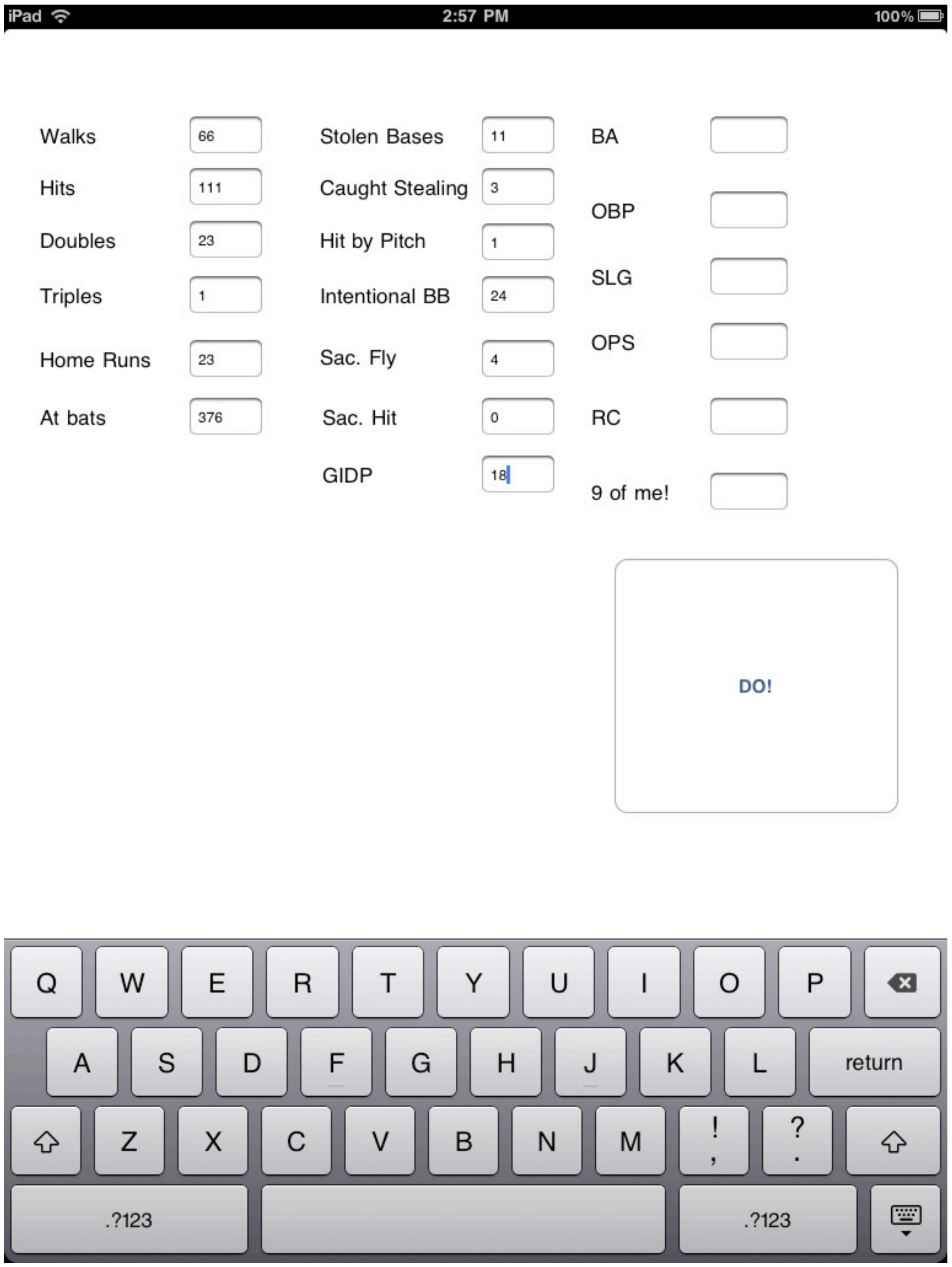


Figure 1: Baseball Counter



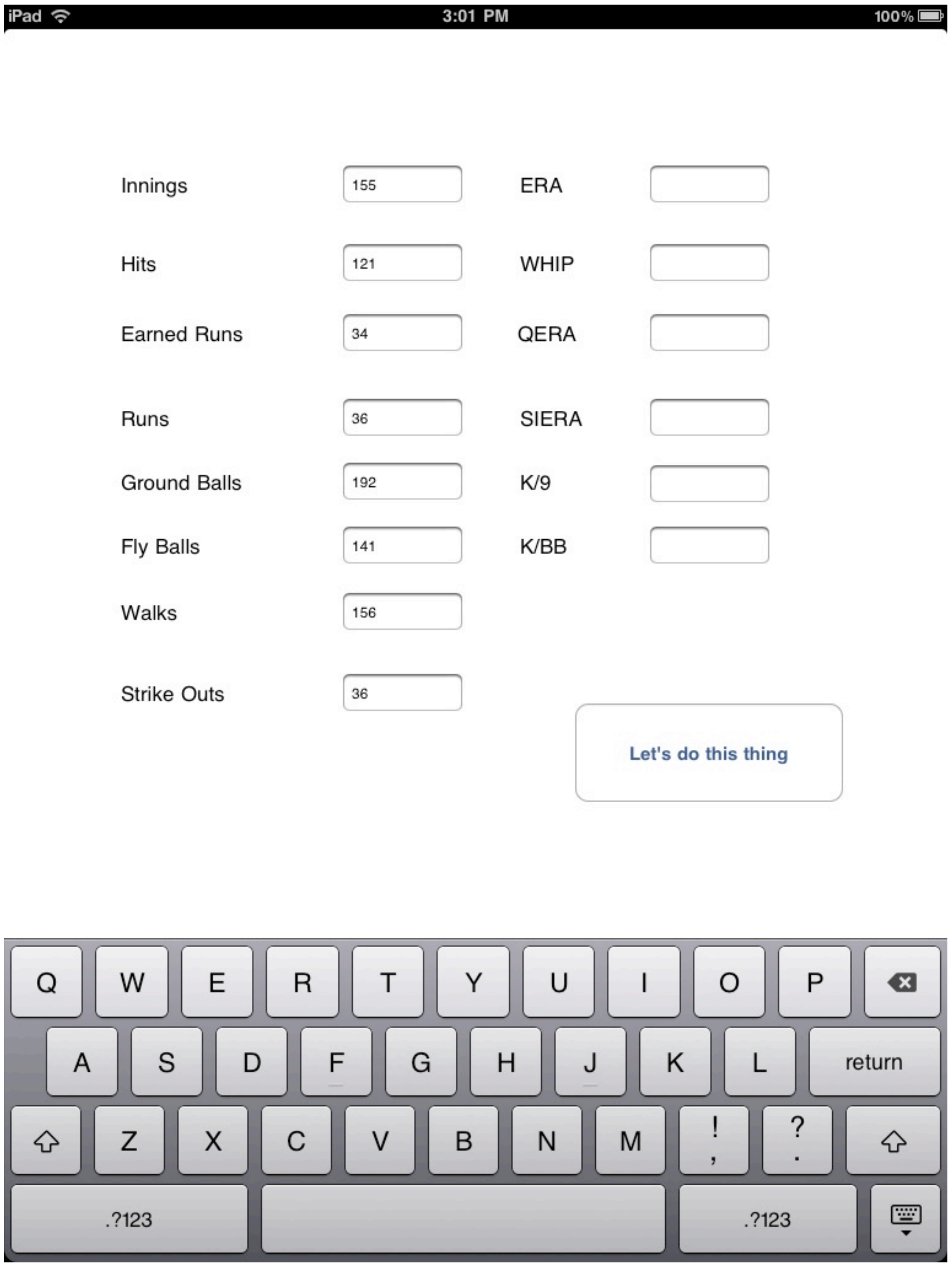
FB	CH	CB	SL
1	0	0	0

Figure 2: Strike Zone



Walks	66	Stolen Bases	11	BA	
Hits	111	Caught Stealing	3	OBP	
Doubles	23	Hit by Pitch	1	SLG	
Triples	1	Intentional BB	24	OPS	
Home Runs	23	Sac. Fly	4	RC	
At bats	376	Sac. Hit	0	9 of me!	
		GIDP	18		

Figure 3: Markov Hitting Chain



Innings	155	ERA	
Hits	121	WHIP	
Earned Runs	34	QERA	
Runs	36	SIERA	
Ground Balls	192	K/9	
Fly Balls	141	K/BB	
Walks	156		
Strike Outs	36		

Figure 4: Pitching Metric

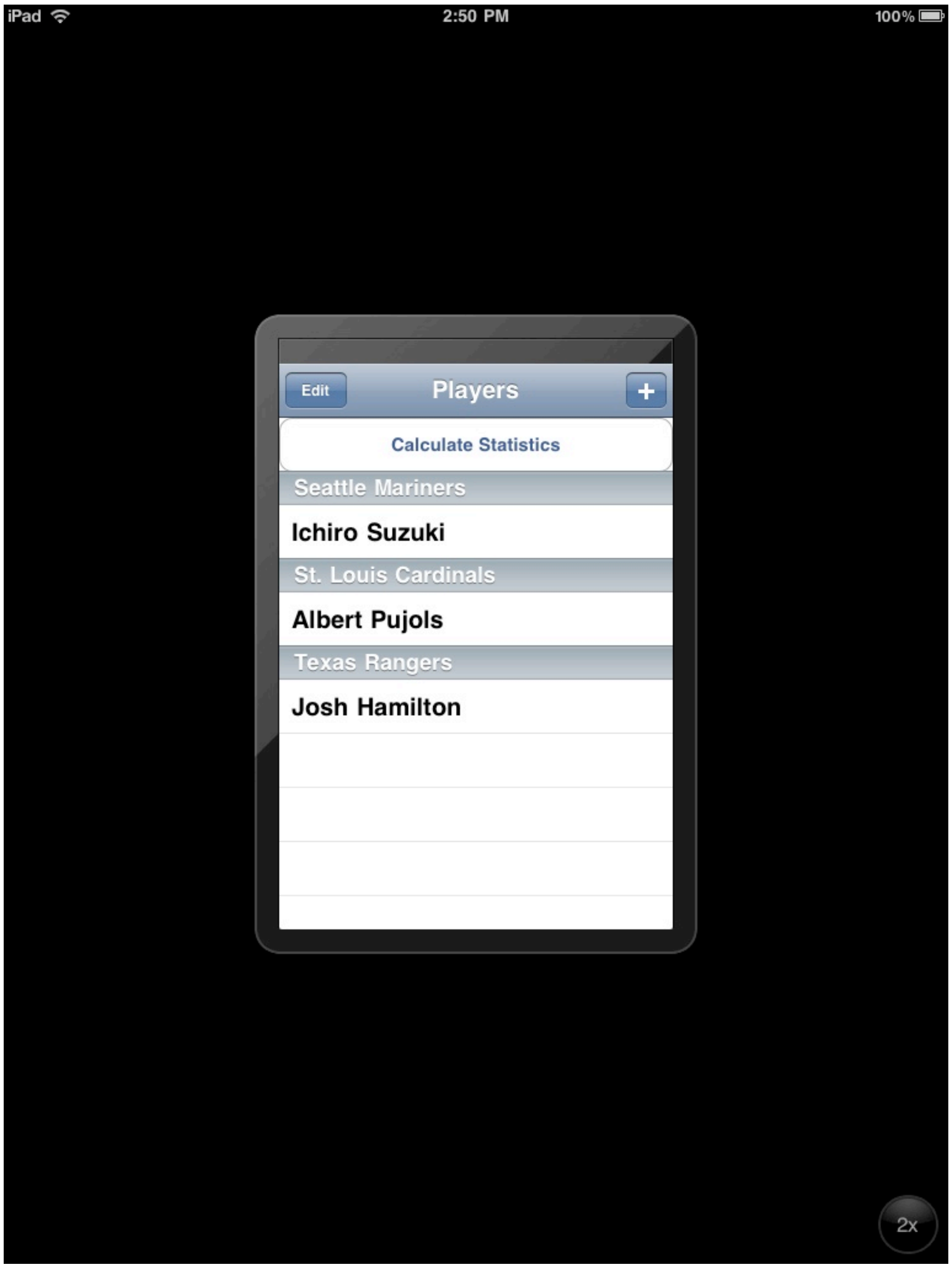
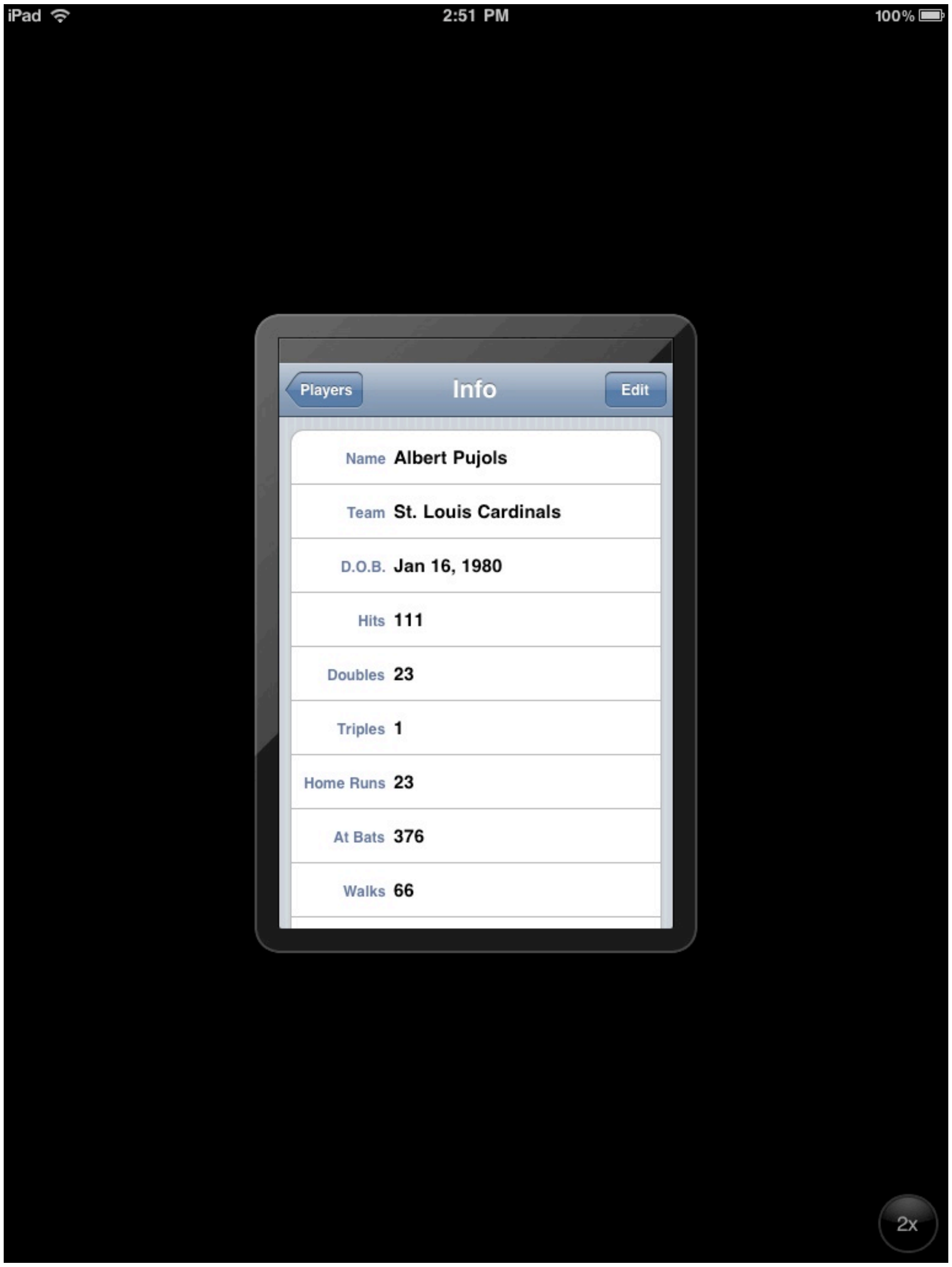


Figure 5: Core Data – Main Page



Name	Albert Pujols
Team	St. Louis Cardinals
D.O.B.	Jan 16, 1980
Hits	111
Doubles	23
Triples	1
Home Runs	23
At Bats	376
Walks	66

Figure 6: Core Data – Player Information Page

Appendix III: Code

Baseball Counter Code

```
//
//  BaseballCounterViewController.m
//

#import "BaseballCounterViewController.h"

@implementation BaseballCounterViewController


- (IBAction)clear {
  textField1.text = @"0";
  textField2.text = @"0";
  textField3.text = @"0";
}
- (IBAction)outs {
  float outs = ([textField3.text floatValue]);
  float outs1;
  outs1 = outs+1;
  textField3.text = [[NSString alloc] initWithFormat:@"%2.f", outs1];
  float x = ([textField3.text floatValue]);
  float c;
  float z = ([textField1.text floatValue]);
  float r = ([textField2.text    floatValue]);
  if (x<3) {
     c = x;
  }
  else
     c = 0;
     z = 0;
     r = 0;
  textField3.text = [[NSString alloc] initWithFormat:@"%2.f", c];
  textField1.text = [[NSString alloc] initWithFormat:@"%2.f", z];
  textField2.text = [[NSString alloc] initWithFormat:@"%2.f", r];
}

- (IBAction)newPitcher {
  label1.text = @"0";
  label2.text = @"0";
  label3.text = @"0";



}
- (IBAction)strike {
  float st = ([label1.text floatValue]);
  float st1;
  st1 = st+1;
  float tp = ([label3.text floatValue]);
```

```
    float tp1;
    tp1 = tp+1;
    label1.text = [[NSString alloc] initWithFormat:@"%2.f", st1];
    label3.text = [[NSString alloc] initWithFormat:@"%2.f", tp1];
    float x = ([textField1.text floatValue]);
    float y = ([textField2.text floatValue]);
    float c;
    float z;
    float r;
    if (x<2) {
        c = x+1;
    }
    else {
        z = ([textField3.text floatValue]);
        y = 0;

        if (z<2) {
            r = z+1;
            c = 0;}
        else {
            textField1.text = @"0";
            textField2.text = @"0";
            textField3.text = @"0";
        }
        textField3.text = [[NSString alloc] initWithFormat:@"%2.f", r];
    }
    textField1.text = [[NSString alloc] initWithFormat:@"%2.f", c];
}

- (IBAction)balls {
    float ba = ([label2.text floatValue]);
    float ba1;
    ba1 = ba+1;
    float tp = ([label3.text floatValue]);
    float tp1;
    tp1 = tp+1;
    label3.text = [[NSString alloc] initWithFormat:@"%2.f", tp1];
    label2.text = [[NSString alloc] initWithFormat:@"%2.f", ba1];
    float y = ([textField2.text floatValue]);
    float d;
    float x = ([textField1.text floatValue]);
    if (y<3) {
        d = y+1;
    }
    else {
        d = 0;
        x = 0;
    }
    textField2.text = [[NSString alloc] initWithFormat:@"%2.f", d];
    textField1.text = [[NSString alloc] initWithFormat:@"%2.f", x];
}
```

```
- (IBAction)foul {
   float st = ([label1.text floatValue]);
   float st1;
   st1 = st+1;
   float tp = ([label3.text floatValue]);
   float tp1;
   tp1 = tp+1;
   label3.text = [[NSString alloc] initWithFormat:@"%2.f", tp1];
   label1.text = [[NSString alloc] initWithFormat:@"%2.f", st1];
   float x = ([textField1.text floatValue]);
   float y = ([textField2.text floatValue]);
   float c;
   float z;

   if (x<2) {
       c = x+1;
   }
   else {
       z = ([textField3.text floatValue]);
       y = ([textField2.text floatValue]);
       c=x;
   }
   textField1.text = [[NSString alloc] initWithFormat:@"%2.f", c];
}


/*
// The designated initializer. Override to perform setup that is required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
    if ((self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil])) {
       // Custom initialization
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically, without using a nib.
- (void)loadView {
}
*/


/*
// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];
}
*/
```

```
/*
// Override to allow orientations other than the default portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning {
   // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

   // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
   // Release any retained subviews of the main view.
   // e.g. self.myOutlet = nil;
}


- (void)dealloc {
    [super dealloc];
}

@end
```

Strike Zone Code

```
//
//  SecondViewController.m
//

#import "VolumeViewController.h"



@implementation VolumeViewController



-(IBAction)count1 {
   float pitchcounta = ([labelcount1.text floatValue]);
   float pitchcounta1;
   pitchcounta1=pitchcounta+1;
   labelcount1.text = [[NSString alloc] initWithFormat:@"%2.f", pitchcounta1];
}

...

-(IBAction)count25 {
   float pitchcounty = ([labelcount25.text floatValue]);
   float pitchcounty1;
   pitchcounty1=pitchcounty+1;
   labelcount25.text = [[NSString alloc] initWithFormat:@"%2.f", pitchcounty1];
}




-(IBAction)pushActionSheet{
   UIActionSheet *actionSheet = [[UIActionSheet alloc] initWithTitle:@"Pitch Type" delegate:self
cancelButtonTitle:@"Cancel" destructiveButtonTitle:nil otherButtonTitles:@"Fastball",@"Change
Up",@"Curve Ball",@"Slider",nil];

   [actionSheet showInView:self.view];
   [actionSheet release];
   [summaryViewController updateTotalSTRIKE];
   [summaryViewController updateTotalCountStrike];




}

-(void)actionSheet:(UIActionSheet *)actionSheet clickedButtonAtIndex:(NSInteger)buttonIndex {
   if (actionSheet == actionSheet) {
       if (buttonIndex == 0) {
           float fb = ([labelFB.text floatValue]);
           float fb1;
```

```
            fb1 = fb+1;
            labelFB.text = [[NSString alloc] initWithFormat:@"%2.f", fb1];
        [summaryViewController updateTotalFB];
        }
        else if (buttonIndex == 1) {
            float ch = ([labelCH.text floatValue]);
            float ch1;
            ch1 = ch+1;
            labelCH.text = [[NSString alloc] initWithFormat:@"%2.f", ch1];
        [summaryViewController updateTotalCH];
        }
        else if (buttonIndex == 2) {
            float cb = ([labelCB.text floatValue]);
            float cb1;
            cb1 = cb+1;
            labelCB.text = [[NSString alloc] initWithFormat:@"%2.f", cb1];
        [summaryViewController updateTotalCB];
        }
        else if (buttonIndex == 3) {
            float sl = ([labelSL.text floatValue]);
            float sl1;
            sl1 = sl+1;
            labelSL.text = [[NSString alloc] initWithFormat:@"%2.f", sl1];
        [summaryViewController updateTotalSL];
        }
    }
}


-(IBAction)pushSecondActionSheet{
    UIActionSheet *actionSheetTwo = [[UIActionSheet alloc] initWithTitle:@"Pitch Type"
delegate:self cancelButtonTitle:@"Cancel" destructiveButtonTitle:nil
otherButtonTitles:@"Fastball",@"Change Up",@"Curve Ball",@"Slider",nil];

    [actionSheetTwo showInView:self.view];
    [actionSheetTwo release];
    [summaryViewController updateTotalBALL];
    [summaryViewController updateTotalCountBall];



}

-(void)actionSheetTwo:(UIActionSheet *)actionSheetTwo
clickedButtonAtIndex:(NSInteger)buttonIndex {
    if (actionSheetTwo == actionSheetTwo) {
        if (buttonIndex == 0) {
            float fb = ([labelFB.text floatValue]);
            float fb1;
            fb1 = fb+1;
            labelFB.text = [[NSString alloc] initWithFormat:@"%2.f", fb1];
        [summaryViewController updateTotalFB];
        }
        else if (buttonIndex == 1) {
           float ch = ([labelCH.text floatValue]);
           float ch1;
           ch1 = ch+1;
           labelCH.text = [[NSString alloc] initWithFormat:@"%2.f", ch1];
        [summaryViewController updateTotalCH];
        }
        else if (buttonIndex == 2) {
           float cb = ([labelCB.text floatValue]);
           float cb1;
           cb1 = cb+1;
           labelCB.text = [[NSString alloc] initWithFormat:@"%2.f", cb1];
        [summaryViewController updateTotalCB];
        }
        else if (buttonIndex == 3) {
           float sl = ([labelSL.text floatValue]);
           float sl1;
           sl1 = sl+1;
           labelSL.text = [[NSString alloc] initWithFormat:@"%2.f", sl1];
        [summaryViewController updateTotalSL];
        }
    }
}


/*
// The designated initializer.  Override if you create the controller programmatically and want to
perform customization that is not appropriate for viewDidLoad.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically, without using a nib.
- (void)loadView {
}
*/

/*
// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];
}
*/
```

```
/*
// Override to allow orientations other than the default portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning {
   // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

   // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
   // Release any retained subviews of the main view.
   // e.g. self.myOutlet = nil;
}


- (void)dealloc {

    [super dealloc];
}


@end
```

Markov Hitting Chain Code

```
//
//  markov_battingViewController.m
//

#import "markov_battingViewController.h"

@implementation markov_battingViewController
int i,j,k,pc,r1,c1,c2;
-(IBAction)calc {
    float PM0[25][25];
    float PM1[25][25];
    float PM2[25][25];
    float PM3[25][25];
    float PM4[25][25];
    float rowad[4][21][21];
    float u[26][21][25];
    float temp[4][21][25];
    float runs;

    float mwalks=([walks.text floatValue]);
    float mhits=([hits.text floatValue]);
    float mdoubles=([doubles.text floatValue]);
    float mtriples=([triples.text floatValue]);
    float mhruns=([hruns.text floatValue]);
    float matbats=([atbats.text floatValue]);
    float mhbp=([hbp.text floatValue]);
    float mcs=([cs.text floatValue]);
    float mgidp=([gidp.text floatValue]);
    float mibb=([ibb.text floatValue]);
    float msh=([sh.text floatValue]);
    float msf=([sf.text floatValue]);
    float msb=([sb.text floatValue]);



    float mplate;
    mplate=mwalks+matbats;

//    plate.text=[[NSString alloc] initWithFormat:@"%2.f", mplate];

    for(i=0;i<25;i++) {
        for(j=0;j<25;j++) {
            PM0[i][j]=0;
            PM1[i][j]=0;
            PM2[i][j]=0;
            PM3[i][j]=0;
            PM4[i][j]=0;
        }
    }
```

```
float fsingle=(mhits-mdoubles-mtriples-mhruns)/mplate;
float fdouble=mdoubles/mplate;
float ftriple=mtriples/mplate;
float fhrun=mhruns/mplate;
float fwalk=mwalks/mplate;
float fout=(matbats-mhits)/mplate;




PM0[0][1]=fwalk+fsingle;
PM0[8][9]=fwalk+fsingle;
PM0[16][17]=fwalk+fsingle;
PM0[0][2]=fdouble;
PM0[8][10]=fdouble;
PM0[16][18]=fdouble;
PM0[0][3]=ftriple;
PM0[8][11]=ftriple;
PM0[16][19]=ftriple;
PM0[1][4]=fwalk+fsingle;
PM0[9][12]=fwalk+fsingle;
PM0[17][20]=fwalk+fsingle;
PM0[1][6]=fdouble;
PM0[9][14]=fdouble;
PM0[17][22]=fdouble;
PM0[2][4]=fwalk;
PM0[10][12]=fwalk;
PM0[18][20]=fwalk;
PM0[3][5]=fwalk;
PM0[11][13]=fwalk;
PM0[19][21]=fwalk;
PM0[4][7]=fwalk;
PM0[12][15]=fwalk;
PM0[20][23]=fwalk;
PM0[5][7]=fwalk;
PM0[13][15]=fwalk;
PM0[21][23]=fwalk;
PM0[6][7]=fwalk;
PM0[14][15]=fwalk;
PM0[22][23]=fwalk;
PM0[0][8]=fout;
PM0[1][9]=fout;
PM0[2][10]=fout;
PM0[3][11]=fout;
PM0[4][12]=fout;
PM0[5][13]=fout;
PM0[6][14]=fout;
PM0[7][15]=fout;
PM0[8][16]=fout;
```

```
PM0[9][17]=fout;
PM0[10][18]=fout;
PM0[11][19]=fout;
PM0[12][20]=fout;
PM0[13][21]=fout;
PM0[14][22]=fout;
PM0[15][23]=fout;
PM0[16][24]=fout;
PM0[17][24]=fout;
PM0[18][24]=fout;
PM0[19][24]=fout;
PM0[20][24]=fout;
PM0[21][24]=fout;
PM0[22][24]=fout;
PM0[23][24]=fout;
PM0[24][24]=1;

PM1[0][0]=fhrun;
PM1[8][8]=fhrun;
PM1[16][16]=fhrun;
PM1[1][3]=ftriple;
PM1[9][11]=ftriple;
PM1[17][19]=ftriple;
PM1[2][1]=ftriple;
PM1[10][9]=fsingle;
PM1[18][17]=fsingle;
PM1[2][2]=fdouble;
PM1[10][10]=fdouble;
PM1[18][18]=fdouble;
PM1[2][3]=ftriple;
PM1[10][11]=ftriple;
PM1[18][19]=ftriple;
PM1[3][1]=fsingle;
PM1[11][9]=fsingle;
PM1[19][17]=fsingle;
PM1[3][2]=fdouble;
PM1[11][10]=fdouble;
PM1[19][18]=fdouble;
PM1[3][3]=ftriple;
PM1[11][11]=ftriple;
PM1[19][19]=ftriple;
PM1[4][4]=fsingle;
PM1[12][12]=fsingle;
PM1[20][20]=fsingle;
PM1[4][6]=fdouble;
PM1[12][14]=fdouble;
PM1[20][22]=fdouble;
PM1[5][4]=fsingle;
PM1[13][12]=fsingle;
PM1[21][20]=fsingle;
PM1[5][6]=fdouble;
```

```
PM1[13][14]=fdouble;
PM1[21][22]=fdouble;
PM1[7][7]=fwalk;
PM1[15][15]=fwalk;
PM1[23][23]=fwalk;

PM2[1][0]=fhrun;
PM2[9][8]=fhrun;
PM2[17][16]=fhrun;
PM2[2][0]=fhrun;
PM2[10][8]=fhrun;
PM2[18][16]=fhrun;
PM2[3][0]=fhrun;
PM2[11][8]=fhrun;
PM2[19][16]=fhrun;
PM2[4][3]=ftriple;
PM2[12][11]=ftriple;
PM2[20][19]=ftriple;
PM2[5][3]=ftriple;
PM2[13][11]=ftriple;
PM2[21][19]=ftriple;
PM2[6][1]=fsingle;
PM2[14][9]=fsingle;
PM2[22][17]=fsingle;
PM2[6][2]=fdouble;
PM2[14][10]=fdouble;
PM2[22][18]=fdouble;
PM2[6][3]=ftriple;
PM2[14][11]=ftriple;
PM2[22][19]=ftriple;
PM2[7][4]=fsingle;
PM2[15][12]=fsingle;
PM2[23][20]=fsingle;
PM2[7][6]=fdouble;
PM2[15][14]=fdouble;
PM2[23][22]=fdouble;

PM3[4][0]=fhrun;
PM3[12][8]=fhrun;
PM3[20][16]=fhrun;
PM3[5][0]=fhrun;
PM3[13][8]=fhrun;
PM3[21][16]=fhrun;
PM3[6][0]=fhrun;
PM3[14][8]=fhrun;
PM3[22][16]=fhrun;
PM3[7][3]=ftriple;
PM3[15][11]=ftriple;
PM3[23][19]=ftriple;

PM4[7][0]=fhrun;
```

```
PM4[15][8]=fhrun;
PM4[23][16]=fhrun;

for(i=0;i<4;i++) {
    for(j=0;j<21;j++) {
        for(k=0;k<21;k++){

            rowad[i][j][k]=0;


        }
    }
}

for(i=0;i<20;i++) {
    rowad[0][i+1][i]=1;
}

for(i=0;i<19;i++) {
    rowad[1][i+2][i]=1;
}

for(i=0;i<18;i++) {
    rowad[2][i+3][i]=1;
}

for(i=0;i<17;i++) {
    rowad[3][i+4][i]=1;
}


for(i=0;i<26;i++) {
    for(j=0;j<21;j++) {
        for(k=0;k<25;k++) {
            u[i][j][k]=0;
        }
    }
}
u[0][0][0]=1;

for(pc=0;pc<25;pc++) {

    for(i=0;i<4;i++) {
        for(j=0;j<21;j++) {
            for(k=0;k<25;k++) {
                temp[i][j][k]=0;
            }
        }
    }
```

```
        for(r1=0;r1<21;r1++) {
            for(c2=0;c2<25;c2++) {
                for(c1=0;c1<21;c1++) {
                    temp[0][r1][c2]+=(rowad[0][r1][c1]*u[pc][c1][c2]);
                    temp[1][r1][c2]+=(rowad[1][r1][c1]*u[pc][c1][c2]);
                    temp[2][r1][c2]+=(rowad[2][r1][c1]*u[pc][c1][c2]);
                    temp[3][r1][c2]+=(rowad[3][r1][c1]*u[pc][c1][c2]);
                }
            }
        }


        for(r1=0;r1<21;r1++) {
            for(c2=0;c2<25;c2++) {
                for(c1=0;c1<25;c1++) {


u[pc+1][r1][c2]+=(u[pc][r1][c1]*PM0[c1][c2])+(temp[0][r1][c1]*PM1[c1][c2])+(temp[1][r1][c1]*PM2[
c1][c2])+(temp[2][r1][c1]*PM3[c1][c2])+(temp[3][r1][c1]*PM4[c1][c2]);
                }
            }
        }

    }

    runs=0;

    for(i=0;i<21;i++) {
        runs+=u[25][i][24]*(i);
    }

//
runs=(u[25][0][24]*0)+(u[25][1][24]*1)+(u[25][2][24]*2)+(u[25][3][24]*3)+(u[25][4][24]*4)+(u[25][5][
24]*5)+(u[25][6][24]*6)+(u[25][7][24]*7)+(u[25][8][24]*8)+(u[25][9][24]*9)+(u[25][10][24]*10)+(u[2
5][11][24]*11);

    thingy.text=[[NSString alloc] initWithFormat:@"%.2f", runs];

    float mbatavg=mhits/matbats;

    batavg.text=[[NSString alloc] initWithFormat:@"%.3f", mbatavg];

    float mobp=(mwalks+mhits)/(mwalks+matbats);

    obp.text=[[NSString alloc] initWithFormat:@"%.3f", mobp];

    float mslg=(4*mhruns+3*mtriples+2*mdoubles+(mhits-mhruns-mdoubles-mtriples))/matbats;

    slg.text=[[NSString alloc] initWithFormat:@"%.3f", mslg];

    float mops=mslg+mobp;
    ops.text=[[NSString alloc] initWithFormat:@"%.3f", mops];

//    float mrunscreated=((mhits+mwalks+mhbp-mcs-
mgidp)*((4*mhruns+3*mtriples+2*mdoubles+(mhits-mhruns-mdoubles-mtriples))+0.26*(mwalks-
mibb+mhbp)+0.52*(msh+msf+msb))/(matbats+mwalks+mhbp+msh+msf));

    float ma=mhits+mwalks+mhbp-mcs-mgidp;
    float mb=(4*mhruns+3*mtriples+2*mdoubles+(mhits-mdoubles-mtriples-mhruns));
    float mc=0.26*(mwalks-mibb+mhbp);
    float md=0.52*(msh+msf+msb);
    float me=matbats+mwalks+mhbp+msh+msf;

    float mrunscreated=(ma*(mb+mc+md))/me;

    runscreated.text=[[NSString alloc] initWithFormat:@"%.3f", mrunscreated];


}
/*
// The designated initializer. Override to perform setup that is required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
   if ((self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil])) {
       // Custom initialization
   }
   return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically, without using a nib.
- (void)loadView {
}
*/


/*
// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
   [super viewDidLoad];
}
*/


// Override to allow orientations other than the default portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
   return YES;
}

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
```

```
  [super didReceiveMemoryWarning];

   // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
   // Release any retained subviews of the main view.
   // e.g. self.myOutlet = nil;
}


- (void)dealloc {
  [super dealloc];
}

@end
```

Pitching Metric Code

```
//
//  pitching_metricViewController.m
//

#import "pitching_metricViewController.h"

@implementation pitching_metricViewController
-(IBAction)calc {

    float minnings=([innings.text floatValue]);
    float mhits=([hits.text floatValue]);
    float meruns=([eruns.text floatValue]);
    float mruns=([runs.text floatValue]);
    float mgballs=([gballs.text floatValue]);
    float mfballs=([fballs.text floatValue]);
    float mwalks=([walks.text floatValue]);
    float msouts=([souts.text floatValue]);
    float gbpercent=mgballs/(mgballs+mfballs);
    float fbpercent=mfballs/(mgballs+mfballs);

    float wholeinnings=floor(minnings);

    float frac=minnings-wholeinnings;
    float frac1;
    frac1=0;
    if(frac<0.15) if(frac>0.05) frac1=0.333;
    if(frac<0.25) if(frac>0.15) frac1=0.667;

    minnings=wholeinnings+frac1;
    float hittersfaced=(minnings*3)+mhits+mwalks;


    float mera=meruns*9/minnings;

    era.text=[[NSString alloc] initWithFormat:@"%.2f", mera];

    float mwhip=(mwalks+mhits)/minnings;

    whip.text=[[NSString alloc] initWithFormat:@"%.2f", mwhip];

    float bbpercent=mwalks/hittersfaced;
    float kpercent=msouts/hittersfaced;

    float mqera=pow((2.69-(0.66*gbpercent)+3.88*bbpercent-3.4*kpercent),2);

    qera.text=[[NSString alloc] initWithFormat:@"%.2f", mqera];

    float msiera=6.262-18.055*kpercent+11.292*bbpercent-1.721*((mgballs-
mfballs)/hittersfaced)+10.169*(pow(kpercent,2))-7.069*(pow(((mgballs-
mfballs)/hittersfaced),2))+9.561*(kpercent)*((mgballs-mfballs)/hittersfaced)-
4.027*(bbpercent)*((mgballs-mfballs)/hittersfaced);


// Need to check the siera calc for some kind of +/-

    siera.text=[[NSString alloc] initWithFormat:@"%.2f", msiera];

    float mkpernine=msouts*9/minnings;

    kpernine.text=[[NSString alloc] initWithFormat:@"%.2f", mkpernine];

    float mkperb=msouts/mwalks;

    kperb.text=[[NSString alloc] initWithFormat:@"%.2f", mkperb];

}

/*
// The designated initializer. Override to perform setup that is required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
   if ((self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil])) {
      // Custom initialization
   }
   return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically, without using a nib.
- (void)loadView {
}
*/


/*
// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
   [super viewDidLoad];
}
*/


// Override to allow orientations other than the default portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
   return YES;
}

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
   [super didReceiveMemoryWarning];
```

```
    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


- (void)dealloc {
   [super dealloc];
}

@end
```

Core Data Code

```
//
//  core_data.m
//
#import "DetailViewController.h"
#import "Book.h"
#import "EditingViewController.h"


@implementation DetailViewController

@synthesize book, dateFormatter, undoManager;


#pragma mark -
#pragma mark View lifecycle

- (void)viewDidLoad {
        [super viewDidLoad];

        // Configure the title, title bar, and table view.
        self.title = @"Info";
    self.navigationItem.rightBarButtonItem = self.editButtonItem;
        self.tableView.allowsSelectionDuringEditing = YES;
}


- (void)viewWillAppear:(BOOL)animated {
    // Redisplay the data.
    [self.tableView reloadData];
        [self updateRightBarButtonItemState];
}


- (void)setEditing:(BOOL)editing animated:(BOOL)animated {
    [super setEditing:editing animated:animated];

        [self.navigationItem setHidesBackButton:editing animated:animated];
    [self.tableView reloadData];

        if (editing) {
          [self setUpUndoManager];
        }
        else {
          [self cleanUpUndoManager];
          // Save the changes.
          NSError *error;
          if (![book.managedObjectContext save:&error]) {
                  // Update to handle the error appropriately.
                  NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
```

```
                    exit(-1);  // Fail
                }
            }
        }


        - (void)viewDidUnload {
                    self.dateFormatter = nil;
        }


        - (void)updateRightBarButtonItemState {
                self.navigationItem.rightBarButtonItem.enabled = [book validateForUpdate:NULL];
        }


        #pragma mark -
        #pragma mark Table view data source methods

        - (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
            // 1 section
            return 1;
        }


        - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
            // 10 rows
            return 10;
        }


        - (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
        *)indexPath {

                static NSString *CellIdentifier = @"Cell";

            UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
            if (cell == nil) {
                cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue2
        reuseIdentifier:CellIdentifier] autorelease];
                    cell.editingAccessoryType = UITableViewCellAccessoryDisclosureIndicator;
            }

                switch (indexPath.row) {

                    case 0:
                            cell.textLabel.text = @"Name";
                            cell.detailTextLabel.text = book.title;
                            break;
                case 1:
                            cell.textLabel.text = @"Team";
                            cell.detailTextLabel.text = book.author;
                            break;
                case 2:
                            cell.textLabel.text = @"D.O.B.";
                            cell.detailTextLabel.text = [self.dateFormatter stringFromDate:book.copyright];
                            break;
                  case 3:
                            cell.textLabel.text = @"Hits";
                            cell.detailTextLabel.text = book.hits;
                            break;
                  case 4:
                            cell.textLabel.text = @"Doubles";
                            cell.detailTextLabel.text = book.doubles;
                            break;
                  case 5:
                            cell.textLabel.text = @"Triples";
                            cell.detailTextLabel.text = book.triples;
                            break;
                  case 6:
                            cell.textLabel.text = @"Home Runs";
                            cell.detailTextLabel.text = book.homeruns;
                            break;
                  case 7:
                            cell.textLabel.text = @"At Bats";
                            cell.detailTextLabel.text = book.atbats;
                            break;
                  case 8:
                            cell.textLabel.text = @"Walks";
                            cell.detailTextLabel.text = book.walks;
                            break;
                  case 9:
                            cell.textLabel.text = @"Strike Outs";
                            cell.detailTextLabel.text = book.strikeouts;
                            break;
            }
            return cell;
        }


        - (NSIndexPath *)tableView:(UITableView *)tv willSelectRowAtIndexPath:(NSIndexPath
        *)indexPath {
            // Only allow selection if editing.
            return (self.editing) ? indexPath : nil;
        }

        - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
        *)indexPath {

              if (!self.editing) return;

            EditingViewController *controller = [[EditingViewController alloc]
```

```
initWithNibName:@"EditingView" bundle:nil];

    controller.editedObject = book;
    switch (indexPath.row) {
        case 0: {
            controller.editedFieldKey = @"title";
            controller.editedFieldName = NSLocalizedString(@"Name", @"display name for title");
            controller.editingDate = NO;
        } break;
        case 1: {
            controller.editedFieldKey = @"author";
                        controller.editedFieldName = NSLocalizedString(@"Team", @"display name for
author");
                        controller.editingDate = NO;
        } break;
        case 2: {
            controller.editedFieldKey = @"copyright";
                        controller.editedFieldName = NSLocalizedString(@"Date of Birth", @"display
name for copyright");
                        controller.editingDate = YES;
        } break;
            case 3: {
            controller.editedFieldKey = @"hits";
                        controller.editedFieldName = NSLocalizedString(@"Hits", @"hits name for hits");
                        controller.editingDate = NO;
            } break;
            case 4: {
            controller.editedFieldKey = @"doubles";
                        controller.editedFieldName = NSLocalizedString(@"Doubles", @"doubles name
for doubles");
                        controller.editingDate = NO;
            } break;
            case 5: {
            controller.editedFieldKey = @"triples";
                        controller.editedFieldName = NSLocalizedString(@"Triples", @"triples name for
triples");
                        controller.editingDate = NO;
            } break;
            case 6: {
            controller.editedFieldKey = @"homeruns";
                        controller.editedFieldName = NSLocalizedString(@"Home Runs", @"homeruns
name for homeruns");
                        controller.editingDate = NO;
            } break;
            case 7: {
            controller.editedFieldKey = @"atbats";
                        controller.editedFieldName = NSLocalizedString(@"At Bats", @"atbats name for
atbats");
                        controller.editingDate = NO;
            } break;
            case 8: {
```

```
	  controller.editedFieldKey = @"walks";
			controller.editedFieldName = NSLocalizedString(@"Walks", @"walks name for
walks");
			controller.editingDate = NO;
		} break;
		case 9: {
		controller.editedFieldKey = @"strikeouts";
			controller.editedFieldName = NSLocalizedString(@"Strike Outs", @"strikeouts
name for strikeouts");
			controller.editingDate = NO;
		} break;

    }

    [self.navigationController pushViewController:controller animated:YES];
	[controller release];
}


- (UITableViewCellEditingStyle)tableView:(UITableView *)tableView
editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath {
	return UITableViewCellEditingStyleNone;
}


- (BOOL)tableView:(UITableView *)tableView
shouldIndentWhileEditingRowAtIndexPath:(NSIndexPath *)indexPath {
	return NO;
}


#pragma mark -
#pragma mark Undo support

- (void)setUpUndoManager {
	  if (book.managedObjectContext.undoManager == nil) {

	  NSUndoManager *anUndoManager = [[NSUndoManager alloc] init];
	  [anUndoManager setLevelsOfUndo:3];
	  self.undoManager = anUndoManager;
	  [anUndoManager release];

	  book.managedObjectContext.undoManager = undoManager;
	}

	NSUndoManager *bookUndoManager = book.managedObjectContext.undoManager;

	NSNotificationCenter *dnc = [NSNotificationCenter defaultCenter];
	[dnc addObserver:self selector:@selector(undoManagerDidUndo:)
name:NSUndoManagerDidUndoChangeNotification object:bookUndoManager];
	[dnc addObserver:self selector:@selector(undoManagerDidRedo:)
```

```
name:NSUndoManagerDidRedoChangeNotification object:bookUndoManager];
}


- (void)cleanUpUndoManager {

	[[NSNotificationCenter defaultCenter] removeObserver:self];

	if (book.managedObjectContext.undoManager == undoManager) {
	  book.managedObjectContext.undoManager = nil;
	  self.undoManager = nil;
	}
}


- (NSUndoManager *)undoManager {
	return book.managedObjectContext.undoManager;
}


- (void)undoManagerDidUndo:(NSNotification *)notification {
	[self.tableView reloadData];
	[self updateRightBarButtonItemState];
}


- (void)undoManagerDidRedo:(NSNotification *)notification {
	[self.tableView reloadData];
	[self updateRightBarButtonItemState];
}


- (BOOL)canBecomeFirstResponder {
	return YES;
}


- (void)viewDidAppear:(BOOL)animated {
	[super viewDidAppear:animated];
	[self becomeFirstResponder];
}


- (void)viewWillDisappear:(BOOL)animated {
	[super viewWillDisappear:animated];
	[self resignFirstResponder];
}


#pragma mark -
#pragma mark Date Formatter
```

```
- (NSDateFormatter *)dateFormatter {
        if (dateFormatter == nil) {
          dateFormatter = [[NSDateFormatter alloc] init];
          [dateFormatter setDateStyle:NSDateFormatterMediumStyle];
          [dateFormatter setTimeStyle:NSDateFormatterNoStyle];
        }
        return dateFormatter;
}


#pragma mark -
#pragma mark Memory management

- (void)dealloc {
    [undoManager release];
    [dateFormatter release];
    [book release];
    [super dealloc];
}

@end
```