# California Polytechnic State University, San Luis Obispo

## Senior Project

*Roborodentia Scoring System*

By
Justin Kikuchi
Computer Engineering
June 2012

Senior Project Advisor: Dr. John Seng

# Table of Contents


| Content | Page # |
|---|---|
| 1. Title Page | 1 |
| 2. Introduction | 3 |
| 3. Design | |
| A. Overview of Design | 4 |
| B. System architecture | 5 |
| i. Tournament Data Structure | 5 |
| ii. WxPython Drawing Implementation | 7 |
| iii. Twisted Webserver | 10 |
| iv. Mobile Scoring Webpage | 11 |
| C. Software algorithms | 13 |
| D. Design decisions | 15 |
| 4. System Integration and Testing | 16 |
| 5. Conclusion | 17 |
| 6. References | 18 |
| 7. Appendices | |
| A. runScore.py | 19 |
| B. tournament.py | 21 |
| C. team.py | 33 |
| D. roboServer.py | 44 |
| E. Mobile Scoring Webpage | 48 |

# Introduction

Every year at Cal Poly there is a robotics competition called Roborodentia . Roborodentia is sponsored by various industry leaders to provide a valuable and fun learning experience for Cal Poly students and alumni. The competition is held during the Cal Poly open house and draws a large number of spectators. In order to provide the spectators with a fun experience, it is necessary to present them with a live scoreboard, with visual and auditory effects.

The objective of this senior project was to design and build a fully functional scoreboard and tournament bracket with real-time scoring and timer. The project was implemented in software using the Python programming language and the WxPython library. The automatic scoring system was also built using Python with a library called Twisted to build a fully functional server. The scoring system used a mobile webpage using l HTML, CSS and JavaScript. The scoring system allowed judges to score the match using a mobile device and a Wi-Fi connection.

The scoring system featured team names, team pictures, match scores, and the double elimination bracket layout. Teams and spectators would also be able to see the next matches that were about to start. Each time a team would gain or loose a point the system would also pay sound effect. The system was implemented and used successfully for the most recent Roborodentia tournament and possibly for future Roborodentia tournaments.

# Design

## Overview

The system was designed with a few different parts that were developed separately. The first major component developed was the structure and functions for the tournament data structure. It was built using a series of Python classes to represent each object in the system. The second component developed was the drawing and position calculation algorithms used along with WxPython. That piece was very crucial in order to make the bracket look visually appealing. Another component was the Twisted Python webserver. Its primary function was to communicate with the mobile scoring device. The final component was the mobile webpage that was used to present the judge with an appealing way to enter real time scores. The visual overview of the system can be seen below in figure 1.

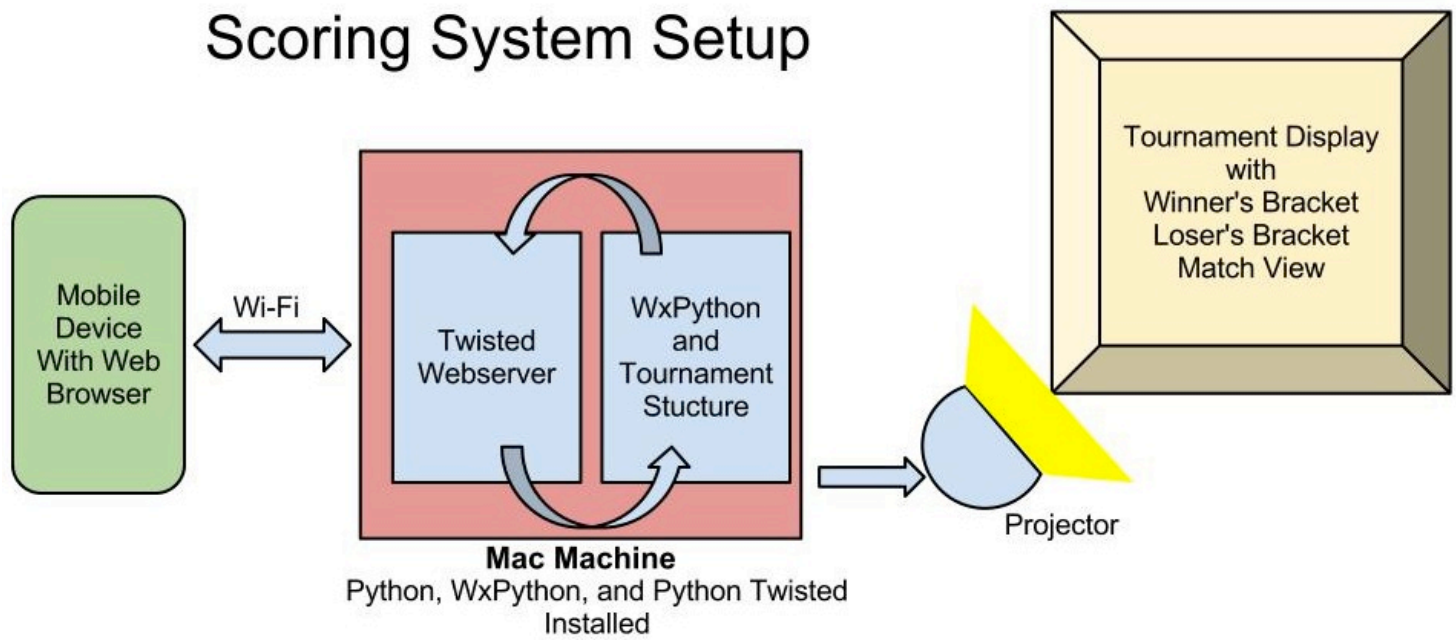


Figure 1

# System Architecture:

## Tournament Data Structure

The tournament data structure was built using a series of Python classes built on top of each other. The Python source code for this part can bee seen in appendix C. As seen in figure 2 below, the hierarchy is as follows: team, match, loser's bracket, winner's bracket, tournament. Each class contains information needed to fulfill its duty. For example, the team class contains the team name and location of team picture. The tournament data structure is based on a simple binary tree, but certain matches, or leaves are added or removed to represent a double elimination tournament.

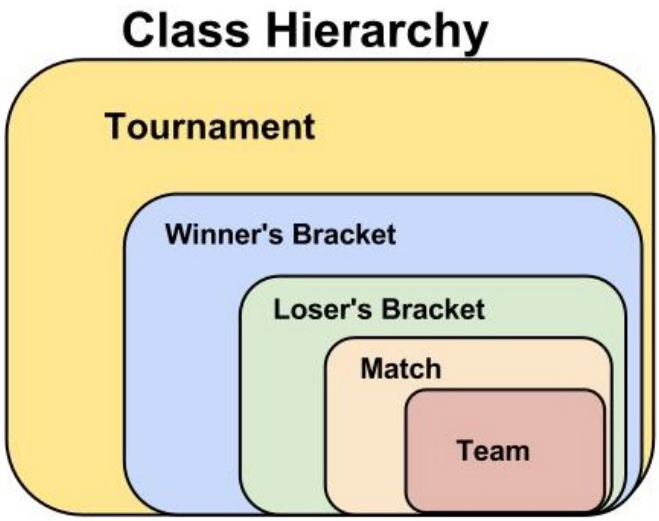


Figure 2

The match class is one of the main structures for the program. It contains the main information for the tournament tree structure. It has the link the bottom two matches and the link to the next match in the tournament. The match class also contains the two teams that are playing in the match along with the match scores. It also will send the winning team to the next match.

The loser's bracket contains the tree that represents teams that have lost exactly one match. This data structure is not a full binary tree, because rounds have to be

duplicated in order to allow for new teams to come into the bracket from the winner's bracket.

The winner's bracket is almost a complete binary tree of matches. It is not a complete binary tree because the bottom branches in the tree are not fully built, if the number of teams is not a multiple of a complete binary tree. The winner's bracket also contains the championship match and the second championship match if the undefeated team looses.

The tournament class is a shell wrapper that encases and sets up the whole tournament data structure. It also contains the algorithms to build both the winners and losers bracket. The tournament class runs the backend capabilities of the application. It moves teams around and puts teams in the correct matches. The tournament class also has added functionality to constantly be writing the tournament's state to a file incase the application crashes. Upon restart, the application can pick up exactly where it left off. The tournament structure runs all the backend operations of the application.

# WxPython Drawing Implementation

The wxPython library is a very useful tool in building a GUI for a software system. The Roborodentia scoring system is build using a double-buffered screen using wxPython. The screen is drawn every time the elements presented on it change. For example, it is refreshed every time a team scores, or the time left in the match changes. The objects are all drawn to a frame using pixel coordinates, so everything must be calculated beforehand. Each match object has an X, Y coordinate of where it is supposed to be drawn. The coordinates are calculated by the number of matches in each round and the size of the screen. If the round has more matches, then the coordinates must be set closer together.

There are three different screens that the audience saw. First there is the main winner's bracket, then the loser's bracket, and finally the match screen. All three screens can be seen below in figures 3, 4, and 5. The match screen shows the audience the selected match with the team names, team picture, team scores, and time left in the match. Drawing is done by first calling a makenewdata() function that creates new data and its coordinates to be drawn. Next, a function is called to create the frame with all the correct elements drawn in place. Then it is switched to the front of the UI to present the user with a seamless transition. All code for this part of the project can be found in Appendix B.

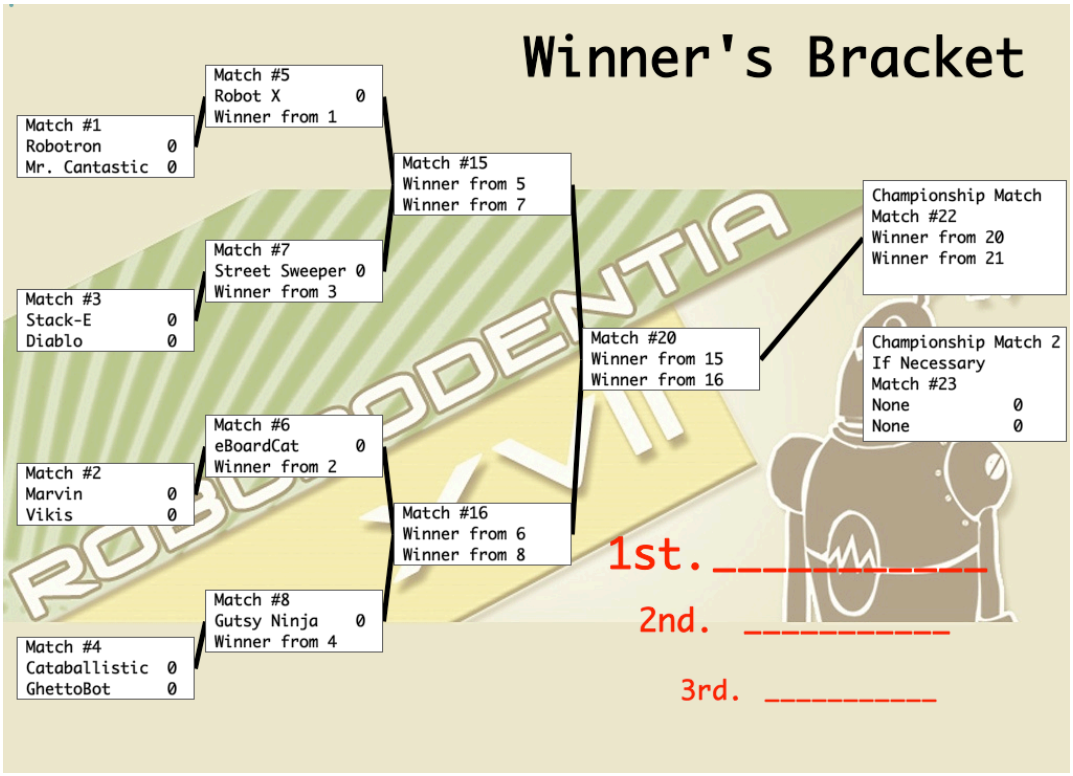


Figure 3

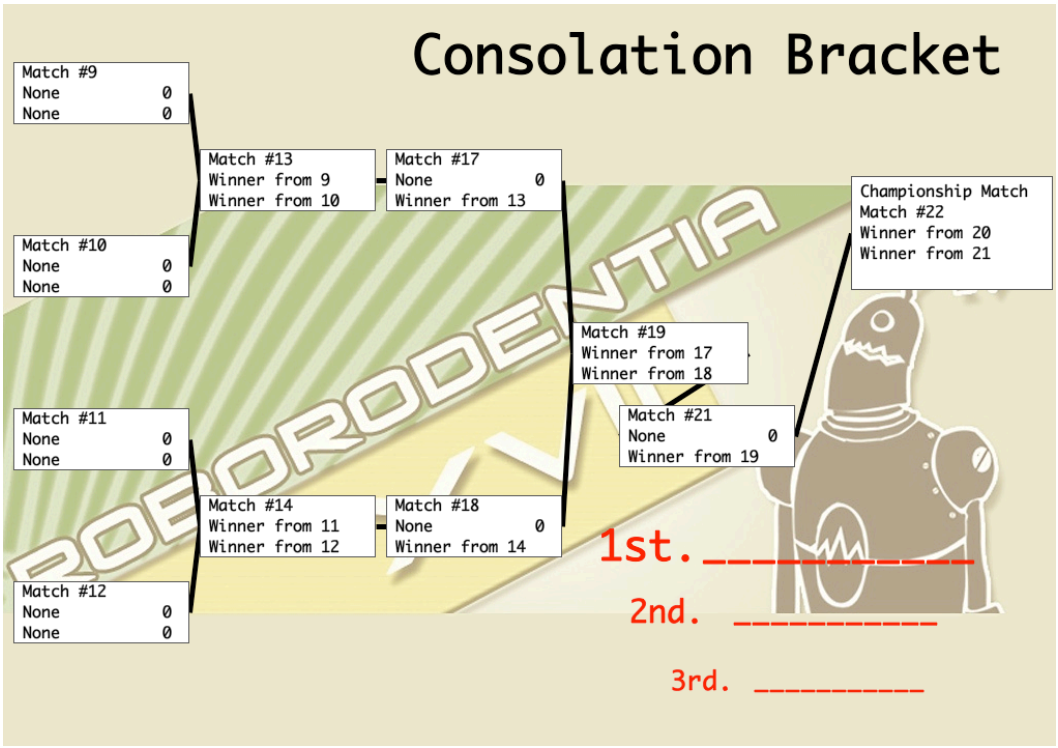


Figure 4


Figure 5

# Twisted Python Server

The Twisted Python server is used to serve requests from the mobile scoring webpage and the actual tournament application. Twisted is the standard Python networking library and is included in most Python instillations.. It stores the current match team names and their scores. The scores on each part of the program will all remain constant. The Python server allows for almost immediate updates of live scoring.

The server could not be run within the wxPython application because the two daemons running infinitely for the open window and to serve requests interfered with each other causing them not to work. The server also had the ability to not only increment or decrement the scores, but set them to a specific number. The server operated by serving json objects to the scoring webpage and the tournament application. All of the code for the webserver can be found in Appendix D.

# Mobile Scoring Webpage

The mobile scoring webpage was built using HTML, CSS, and JavaScript. The JavaScript relied heavily on JQuery for handling requests to the Twisted webserver. As seen below in figure 6, the webpage consisted of two team names with plus and minus buttons on either side of them. Then the score was in the middle. Outlined in figure 7, the score was a text box that enabled the judge to also manually enter any score. The webpage also featured a loading screen to ensure the user that the system was actually updating with the score and sending the requests to the server, seen in figure 8.


**Figure 6**

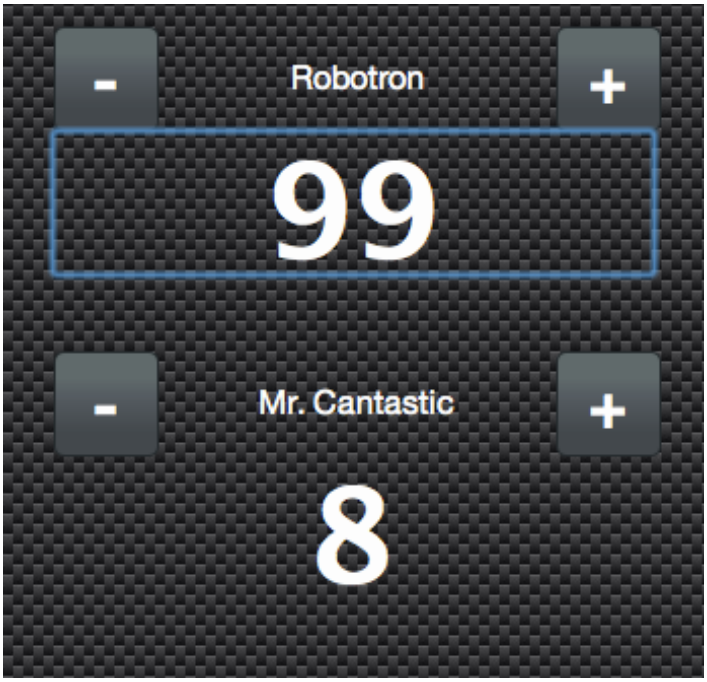


Figure 7

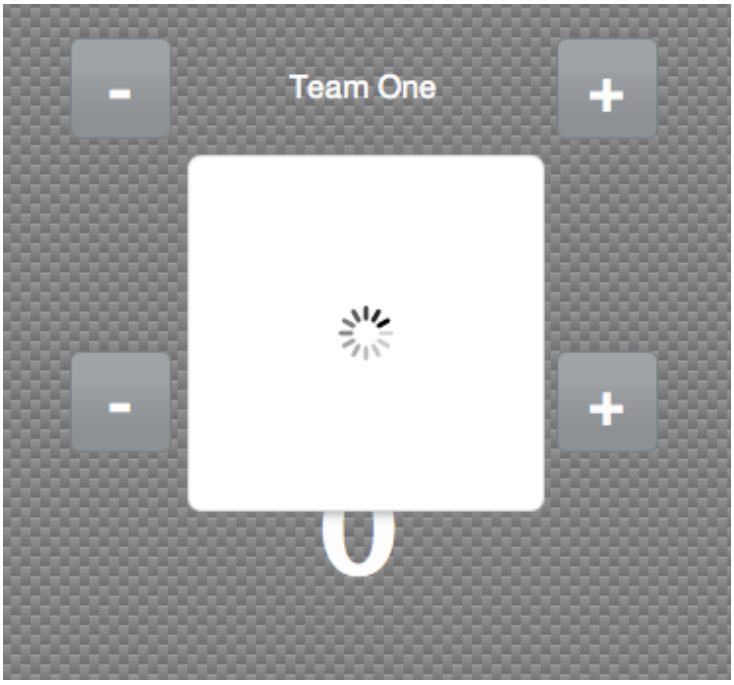


Figure 8

# Software Algorithms

The two most complicated software algorithms used to create the tree structure of matches in the tournament and determining the location of where to draw the matches on the screen.

Creating the tree structure of the winner's bracket was fairly straightforward. It would create a complete binary tree for one match less than the number of teams in the tournament. Then level order tree traversal was used to get a list of all the matches, from there they were numbered accordingly and team classes were added to the match.

The loser's bracket was little bit more complicated because it didn't not consist of a full binary tree. It was necessary to allow for new teams to be inserted into the bracket as it went on because, of the new first time losers from the winner's bracket would need to be inserted into it. As seen in figure 4 above, the algorithm used to create the loser's bracket created a tree structure that had duplicate levels at each increase of the level. For example, there is two rounds with one match, two rounds with two matches, two rounds with 4 matches, and so on.

Winning teams would be inserted in to the parent node of the match that it was in and losing teams would either out of the tournament, or inserted into the first open spot in the loser's bracket.

The algorithm to draw the match objects in the appropriate spot used the total pixel height and width of the screen it was on to calculate the appropriate location of the match on the screen. The more matches in the current round then the less space that match would put between itself and the match above and below it. Since the screen size

was known of the projector that the application was going to be run on, some values were hard coded to allow for the best presentation of the bracket.

# Design Decisions

Python was used as the main programming language for this project because it is an object orientated language that with a wide variety of libraries built for it. WxPython was a suggestion by Professor Seng as a good GUI application library. It works on most operating systems and has a clean look to the GUI.

Twisted was used as the webserver because it is the standard Python API for network communication. Twisted is included in most installations of Python and there is a lot of documentation online for Twisted.

Everything was done using a web based wireless scoring approach because wireless technology is very fast and reliable for these types of systems. Using a simple HTML page as the scoring interface was done to allow for any mobile device or computer to use it and it did not require an installation of an application on the device. Judges were able to move around and could score the match from any Wi-Fi connection.

# System Integration and Testing

Every part of the application was built and tested separately. When the system was integrated, there were not that many hiccups because each part was tested beforehand. The tournament application was first tested using keystrokes instead of communicating with a server to update scores. By using the keystrokes the application was able to be tested before the implementation of the server or webpage. Each part was built in small increments and tested thoroughly through out the development cycle. \

Development was done in week sprint periods with deliverables each week. This method allowed for a quality, tested product to be delivered in a short amount of time. This project was fairly complicated and had many different pieces that needed to built in a short amount of time.

# Conclusion

Through out the development of the Roborodentia Scoring System, there were many challenges that I faced. Python, wxPython, Twisted, and webpage development were all new topics that had to be researched, explored, and implemented. Each step was a new subject that I had to learn about and develop. This was a very slow development process, but it required me to learn a lot of new topics and information about the API or subject.

Since I was the sole developer of this project, I managed the operation from start to finish. Professor John Seng served as the client for this project, and provided certain constraints and functionality for the project. Professor Seng is the faculty advisor for the Cal Poly Roborodentia Tournament. This project was developed as a software consultant doing a project for a client. I researched, development, and ran the scoring system for the 2012 Roborodentia tournament. I familiarized myself with many new and relevant technologies that were necessary to develop this project. Since there was a variety of parts and technologies used for this project, I learned a very wide range of new, important web development, and GUI interface programming.

This project was a very valuable learning experience. The multiple pieces each required knowledge of a different piece of software structure. I was able to see a product from planning to research, then from research to implementation, and finally from implementation to delivery. This project was very similar to a real world project implementation.

# References

Dunn, Robin. "FrontPage - WxPyWiki." *WxPyWiki*. WxPython, 26 Apr. 2012. Web. 31 May 2012. <http://wiki.wxpython.org/FrontPage>.

Fettig, Abe, and Glyph Lefkowitz. *Twisted Network Programming Essentials*. Sebastopol, CA: O'Reilly, 2006. Print.

Gardner, Justin J. "HTML, CSS, JavaScript Methods." Personal interview. 10 Apr. 2012.

# Appendix A
## runScore.py

```
"""
runScore.py

created by Justin Kikuchi

Main application for running roborodentia scoring system.
Instanciates main frame and window to ask for the team file.
Then builts the tournament based on teams.

"""
import wx
import random
from team import Tournament
from tournament import MainFrame


class MainApp(wx.App):
    def OnInit(self):
        frame = MainFrame()
        self.SetTopWindow(frame)
        teamFile = frame.textentry()
        if not teamFile:
            return False
        tour = Tournament(teamFile)
        frame.setTournament(tour)
        tour.renumberMatches()

        return True

if __name__ == "__main__":
    app = MainApp(0)
    app.MainLoop()
```

# Appendix B
## tournament.py

```
"""
tournament.py

created by Justin Kikuchi

Main wxpython file to draw images and windows.
It will assign coordinates to draw and create data to draw.

"""
import wx
import os
import time
import math
import sys
sys.path.append("/System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/")
import xmlrpclib
from time import time
useServer = True

class BufferedWindow(wx.Window):

    def __init__(self, *args, **kwargs):
        kwargs['style'] = kwargs.setdefault('style', wx.NO_FULL_REPAINT_ON_RESIZE)
        wx.Window.__init__(self, *args, **kwargs)

        wx.EVT_PAINT(self, self.OnPaint)
        wx.EVT_SIZE(self, self.OnSize)

        self.OnSize(None)
        self.paint_count = 0
        self.Bind(wx.EVT_LEFT_DOWN, self.onImageClick)

    def onImageClick(self, event):
        pass

    def Draw(self, dc):
        pass

    def OnPaint(self, event):
        dc = wx.BufferedPaintDC(self, self._Buffer)

    def OnSize(self,event):
        # The Buffer init is done here, to make sure the buffer is always
        # the same size as the Window
        #Size  = self.GetClientSizeTuple()
        Size  = self.ClientSize

        # Make new offscreen bitmap: this bitmap will always have the
        # current drawing in it, so it can be used to save the image to
        # a file, or whatever.
        self._Buffer = wx.EmptyBitmap(*Size)
        self.UpdateDrawing()

    def SaveToFile(self, FileName, FileType=wx.BITMAP_TYPE_PNG):
        ## This will save the contents of the buffer
        ## to the specified file. See the wxWindows docs for
```

```
        ## wx.Bitmap::SaveFile for the details
        self._Buffer.SaveFile(FileName, FileType)

    def UpdateDrawing(self):
        dc = wx.MemoryDC()
        dc.SelectObject(self._Buffer)
        self.Draw(dc)
        del dc # need to get rid of the MemoryDC before Update() is called.
        self.Refresh()
        self.Update()

class DrawWindow(BufferedWindow):
    def __init__(self, *args, **kwargs):
        ## Any data the Draw() function needs must be initialized before
        ## calling BufferedWindow.__init__, as it will call the Draw
        ## function.
        self.DrawData = {}
        self.Matches = {}
        self.Time = "00:00"
        self.Champ = None
        BufferedWindow.__init__(self, *args, **kwargs)


    def scale_bitmap(self, bitmap, width, height):
        image = wx.ImageFromBitmap(bitmap)
        image = image.Scale(width, height, wx.IMAGE_QUALITY_HIGH)
        result = wx.BitmapFromImage(image)
        return result

    def Draw(self, dc):
        dc.SetBackground( wx.Brush("Light Grey") )
        dc.Clear() # make sure you clear the bitmap!
        #sizeX, sizeY = wx.DisplaySize()
        sizeX = 1024
        sizeY = 768

        bitmap = wx.Bitmap("images/newrobo2.jpg")
        dc.DrawBitmap(bitmap, 0, 0 , True)
        # Here's the actual drawing code.
        l = []
        if len(self.Matches) == 1:
            dc.SetTextForeground('BLUE')
            if self.Matches[0].team1 != None:
                dc.SetFont(wx.Font(80, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                dc.DrawText(str(self.Matches[0].score1), (sizeX/4), 500)
                dc.SetFont(wx.Font(50, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                dc.DrawText(self.Matches[0].team1.name, (sizeX)/6 - (10*len(self.Matches[0].team1.name)),
20)
                if self.Matches[0].team1.logo != "None" and os.path.exists(self.Matches[0].team1.logo):
                    bitmap = wx.Bitmap(self.Matches[0].team1.logo)
                    imgX, imgY = bitmap.GetSize()
                    i = 1
                    while imgY/i > 400 or imgX/i > 400:
                        i+=.25
                    bitmap = self.scale_bitmap(bitmap, imgX/i, imgY/i)
                    imgX, imgY = bitmap.GetSize()
```

```
                dc.DrawBitmap(bitmap, (sizeX/4) - imgX/2, 100, True)
            else:
                dc.DrawText("None", sizeX/4, 20)
            dc.SetTextForeground('RED')
            if self.Matches[0].team2 != None:
                dc.SetFont(wx.Font(80, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                dc.DrawText(str(self.Matches[0].score2), (sizeX/4)*3, 500)
                dc.SetFont(wx.Font(50, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                dc.DrawText(self.Matches[0].team2.name, ((sizeX)/6) * 4 -
(10*len(self.Matches[0].team1.name)), 20)
                if self.Matches[0].team2.logo != "None" and os.path.exists(self.Matches[0].team2.logo):
                    bitmap = wx.Bitmap(self.Matches[0].team2.logo)
                    imgX, imgY = bitmap.GetSize()
                    i = 1
                    while imgY/i > 400 or imgX/i > 400:
                        i+=1
                    bitmap = self.scale_bitmap(bitmap, imgX/i, imgY/i)
                    imgX, imgY = bitmap.GetSize()
                    dc.DrawBitmap(bitmap, (sizeX/4 * 3) - imgX/2 , 100, True)
            else:
                dc.DrawText("None", sizeX/4 *3, 20)
            dc.SetTextForeground('BLACK')
            dc.SetFont(wx.Font(100, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
            dc.DrawText(self.Matches[0].getTime(), sizeX/5 * 2, 600)
            if self.Time != '-1':
                dc.SetTextForeground('RED')
                dc.DrawText(self.Time, sizeX/5 * 4, 600)

        else:
            for match in self.Matches:
                if match.parent:
                    pen = wx.Pen('#000000', 5, wx.SOLID)
                    pen.SetCap(wx.CAP_BUTT)
                    dc.SetPen(pen)
                    x1, y1 = match.getRightCoord()
                    x2, y2 = match.parent.getLeftCoord()
                    dc.DrawLine(x1, y1, x2, y2)
                dc.SetPen(wx.Pen('#4c4c4c', 1, wx.SOLID))
                dc.DrawRectangleRect(match.getCoord())
                label = match.getLabel()
                dc.SetFont(wx.Font(15, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                dc.DrawLabel(label, match.getCoord(), alignment=wx.ALIGN_LEFT)
            dc.SetFont(wx.Font(50, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
            if len(self.Matches) > 1 and self.Matches[0].isLoser:
                dc.DrawText("Consolation Bracket", 400, 20)
            else:
                dc.DrawText("Winner's Bracket", 500, 20)
                dc.SetFont(wx.Font(15, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                if self.Champ != None and self.Champ.parent != None:
                    dc.DrawRectangleRect(self.Champ.parent.getCoord())
                    self.Champ.parent.Y  += 5
                    dc.DrawLabel("\tChampionship Match 2\n\tIf Necessary\n"+self.Champ.parent.getLabel(),
self.Champ.parent.getCoord(),alignment=wx.ALIGN_LEFT)
            dc.SetFont(wx.Font(15, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
            if self.Champ != None:
                #dc.DrawText("Championship Match", sizeX-197, sizeY/3 + 40)
```

```
            dc.DrawRectangleRect(self.Champ.getCoord())
            self.Champ.Y  += 5
            dc.DrawLabel("\tChampionship Match\n"+self.Champ.getLabel(),
self.Champ.getCoord(),alignment=wx.ALIGN_LEFT)

            dc.SetTextForeground('RED')
            dc.SetFont(wx.Font(30, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
            dc.DrawText("2nd.", 610, 570)
            dc.SetFont(wx.Font(25, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
            dc.DrawText("3rd.", 650, 640)
            dc.SetFont(wx.Font(40, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
            dc.DrawText("1st.", 580, 500)
            if self.Champ.parent.team1 == None:
               if self.Champ.winner == None:
                  dc.DrawText("_____________", 680, 500)
                  dc.SetFont(wx.Font(30, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                  dc.DrawText("_____________", 710, 570)
                  dc.SetFont(wx.Font(25, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                  dc.DrawText("_____________", 730, 640)
               else:
                  dc.DrawText(self.Champ.winner.name, 680, 500)
                  dc.SetFont(wx.Font(30, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                  dc.DrawText(self.Champ.loser.name, 710, 570)
                  dc.SetFont(wx.Font(25, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                  dc.DrawText(self.Champ.left.loser.name, 730, 640)
            else:
               if self.Champ.parent.winner == None:
                  dc.DrawText("_____________", 680, 500)
                  dc.SetFont(wx.Font(30, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                  dc.DrawText("_____________", 710, 570)
                  dc.SetFont(wx.Font(25, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                  dc.DrawText("_____________", 730, 640)
               else:
                  dc.DrawText(self.Champ.parent.winner.name, 680, 500)
                  dc.SetFont(wx.Font(30, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                  dc.DrawText(self.Champ.parent.loser.name, 710, 570)
                  dc.SetFont(wx.Font(25, wx.MODERN, wx.NORMAL, wx.FONTWEIGHT_BOLD))
                  dc.DrawText(self.Champ.left.loser.name, 730, 640)




      for key, data in self.DrawData.items():
         if key == "Ellipses":
            dc.SetBrush(wx.Brush("GREEN YELLOW"))
            dc.SetPen(wx.Pen('CADET BLUE', 2))
            for r in data:
               dc.DrawEllipse(*r)
   def onImageClick(self, event):
      #print "Click"
      pass



class MainFrame(wx.Frame):
```

```
teams = []
tournament = None
num_teams = None
preDisplay = None
currMatch = None
display = "winners"
displayMatch = None
start = -4
tmp=0
loserList = []
def __init__(self, parent=None, ):
    wx.Frame.__init__(self, parent,
                      size = (1024, 768),
                      title="Tournament",
                      style=wx.DEFAULT_FRAME_STYLE)

    MenuBar = wx.MenuBar()

    file_menu = wx.Menu()
    fscreenMI = file_menu.Append(wx.ID_ANY,"Full Screen\tQ")
    displayMI = file_menu.Append(wx.ID_ANY,"Toggle Display\tT")
    item      = file_menu.Append(wx.ID_EXIT, text="&Exit")
    teamsMI   = file_menu.Append(wx.ID_ANY, "&Enter Teams\tF10")
    matchMI   = file_menu.Append(wx.ID_ANY, "Select Match\tM")

    self.Bind(wx.EVT_MENU, self.OnQuit, item)
    self.Bind(wx.EVT_MENU, self.OnFullScreen,id=fscreenMI.GetId())
    self.Bind(wx.EVT_MENU, self.OnDisplay,id=displayMI.GetId())
    self.Bind(wx.EVT_MENU, self.OnMatch, id=matchMI.GetId())
    self.Bind(wx.EVT_MENU, self.textentry, id=teamsMI.GetId())
    MenuBar.Append(file_menu, "&File")


    draw_menu = wx.Menu()
    item = draw_menu.Append(wx.ID_ANY, "&New Drawing")
    self.Bind(wx.EVT_MENU, self.NewDrawing, item)
    item = draw_menu.Append(wx.ID_ANY,'&Save Drawing\tAlt-I')
    self.Bind(wx.EVT_MENU, self.SaveToFile, item)
    MenuBar.Append(draw_menu, "&Draw")

    score_menu = wx.Menu()
    item = score_menu.Append(wx.ID_ANY, "Add Point Left Team\tD")
    self.Bind(wx.EVT_MENU, self.addPointL, item)
    item = score_menu.Append(wx.ID_ANY, "Minus Point Left Team\tF")
    self.Bind(wx.EVT_MENU, self.minusPointL, item)
    item = score_menu.Append(wx.ID_ANY, "Add Point Right Team\tJ")
    self.Bind(wx.EVT_MENU, self.addPointR, item)
    item = score_menu.Append(wx.ID_ANY, "Minus Point Right Team\tK")
    self.Bind(wx.EVT_MENU, self.minusPointR, item)
    item = score_menu.Append(wx.ID_ANY, "Start/Stop Clock\tV")
    self.Bind(wx.EVT_MENU, self.startClock, item)
    item = score_menu.Append(wx.ID_ANY, "Reset Clock\tN")
    self.Bind(wx.EVT_MENU, self.resetClock, item)
    item = score_menu.Append(wx.ID_ANY, "Reset Scores\tX")
    self.Bind(wx.EVT_MENU, self.resetScores, item)
    item = score_menu.Append(wx.ID_ANY, "Set Match\tZ")
```

```
        self.Bind(wx.EVT_MENU, self.setMatch, item)
        MenuBar.Append(score_menu, "&Scoring")

        self.SetMenuBar(MenuBar)

        self.timer = wx.Timer(self, -1)
        self.refreshTimer = wx.Timer(self, -1)
        # update clock digits every second (1000ms)
        self.timer.Start(1000)
        self.refreshTimer.Start(50)
        self.Bind(wx.EVT_TIMER, self.OnTimer, self.timer)
        self.Bind(wx.EVT_TIMER, self.OnRefresh, self.refreshTimer)

        self.startSound = wx.Sound("sound_effects/start.m4a")
        self.scoreSound = wx.Sound("sound_effects/score1.m4a")
        self.score2Sound = wx.Sound("sound_effects/score2.wav")
        self.minusSound = wx.Sound("sound_effects/minus.wav")
        self.minSound = wx.Sound("sound_effects/min.wav")
        self.endSound = wx.Sound("sound_effects/end.wav")


        self.Window = DrawWindow(self)
        self.Show()
        # Initialize a drawing -- it has to be done after Show() is called
        #   so that the Windows has teh right size.
        self.NewDrawing()

    def sendJson(self):
        if useServer:
            data = self.tournament.getJsonMatch(self.currMatch)
            #print data
            if data == None:
                return
            server = xmlrpclib.ServerProxy('http://localhost:8089/robo')
            status = server.setMatch(data)

    def startClock(self, event):
        if self.display == "match":
            if self.start > -4:
                self.start = -4
            else:
                self.startSound.Play()
                self.start = -3

    def resetClock(self, event):
        if self.display == "match":
            self.currMatch.time = 180
            self.currMatch.minSoundPlayed = False
            self.NewDrawing()

    def resetScores(self, event):
        if self.display == "match":
            self.currMatch.score1 = 0
            self.currMatch.score2 = 0
            self.sendJson()
```

```
def addPointL(self, event):
    if self.display == "match":
        self.currMatch.score1 +=1
        self.scoreSound.Play()
        self.sendJson()
        if not useServer:
            self.NewDrawing()


def minusPointL(self, event):
    if self.display == "match":
        self.currMatch.score1 -=1
        self.minusSound.Play()
        self.sendJson()
        if not useServer:
            self.NewDrawing()


def addPointR(self, event):
    if self.display == "match":
        self.currMatch.score2 +=1
        self.score2Sound.Play()
        self.sendJson()
        if not useServer:
            self.NewDrawing()

def minusPointR(self, event):
    if self.display == "match":
        self.currMatch.score2 -=1
        self.minusSound.Play()
        self.sendJson()
        if not useServer:
            self.NewDrawing()

def setMatch(self, event):
    if self.display == "match":
        if self.currMatch == self.tournament.champ:
            self.currMatch.setMatch(isChamp = True)
            if self.currMatch.loser.losses == 1:
                self.tournament.champ2.team1 = self.currMatch.team2
                self.tournament.champ2.team2 = self.currMatch.team1
        else:
            self.currMatch.setMatch(isChamp = False)
        if not self.currMatch.isLoser:
            #loserList.append(self.currMatch.getLoser())
            self.tournament.setLoser(self.currMatch)

def OnTimer(self, event):
    if self.display == "match" and self.start > -1:
        if self.currMatch.time == 180:
            self.startSysTime = time()
        elapse = time()
        self.currMatch.time = int(180 - (elapse  - self.startSysTime))
        if self.currMatch.time < 0:
            self.currMatch.time = 0
            self.start = -4
```

```
            self.endSound.Play()
        if self.currMatch.time < 119 and not self.currMatch.minSoundPlayed:
            self.currMatch.minSoundPlayed = True
            self.minSound.Play()
    if self.start > -4:
        self.start += 1
    self.NewDrawing()
    if self.tournament != None:
        self.tournament.saveState()

def OnRefresh(self, event):
    if self.currMatch != None:
        if useServer:
            if self.currMatch.getScore1() or self.currMatch.getScore2():
                #self.sendJson()
                self.NewDrawing()
    self.tmp +=1

def OnMatch(self, event):
    dlg = wx.TextEntryDialog(self, 'Please enter match number','Match Select')

    if dlg.ShowModal() == wx.ID_OK:
        self.displayMatch = dlg.GetValue()
    else:
        return

    if not self.displayMatch.isdigit():
        self.displayMatch = 0
        return
    self.display = "match"
    self.displayMatch = int(self.displayMatch)

    match = self.tournament.findMatch(int(self.displayMatch))
    self.currMatch = match
    self.sendJson()
    dlg.Destroy()
    self.NewDrawing()

def onImageClick(self):
    print "&&&&&&&&&&"
    self.NewDrawing()
    pos = event.GetPositionTuple()
    if self.tournament != None:
        matchList = self.tournament.getMatchList()
        for list in matchList:
            for match in list:
                if(match.checkClick(pos[0], pos[1])):
                    print "CLICKED: " + str(match.value)
    print(event.GetPositionTuple())
    print('CLICK')

def OnQuit(self,event):
    self.Close(True)

def OnDisplay(self,event):
    if self.display == "winners":
```

```
            self.display = "losers"
        else:
            self.display = "winners"
        self.NewDrawing()

    def OnFullScreen(self,event):
        self.ShowFullScreen(not self.IsFullScreen(), wx.FULLSCREEN_ALL)

    def NewDrawing(self, event=None):
        self.Window.DrawData, self.Window.Matches, self.Window.Time, self.Window.Champ =
self.MakeNewData()
        self.Window.UpdateDrawing()

    def SaveToFile(self,event):
        dlg = wx.FileDialog(self, "Choose a file name to save the image as a PNG to",
                            defaultDir = "",
                            defaultFile = "",
                            wildcard = "*.png",
                            style = wx.SAVE)
        if dlg.ShowModal() == wx.ID_OK:
            self.Window.SaveToFile(dlg.GetPath(), wx.BITMAP_TYPE_PNG)
        dlg.Destroy()

    def MakeNewData(self):
        MaxX, MaxY = self.Window.GetClientSizeTuple()
        DrawData = {}
        Matches = []
        #sizeX, sizeY = wx.DisplaySize()
        sizeX, sizeY = 1024, 768
        Champ = None
        offset = []
        if self.tournament != None:
            if self.display == "match":
                Matches.append(self.currMatch)
            else:
                self.preDisplay = self.display
                if self.display == "winners":
                    matchList = self.tournament.getMatchList()
                    if len(matchList)>4:
                        offset = [24, (sizeY-100)/16-24,(sizeY-100)/8-24, (sizeY-100)/4-24, (sizeY-100)/2-24]
                    elif len(matchList) >3:
                        offset = [24, (sizeY-100)/8-24, (sizeY-100)/4-24, (sizeY-100)/2-24]
                    elif len(matchList) >2:
                        offset = [24, (sizeY-100)/4-24, (sizeY-100)/2-24]
                    else:
                        offset = [24, (sizeY-100)/2-24]

                    i = 0
                    for list in matchList:
                        j = 0
                        for match in list:
                            if match != "Blank":
                                match.X = 16+180*i
                                match.Y = (((sizeY-100)/len(list)) * j) + (offset[i])
                                Matches.append(match)
                            j += 1
```

```
                i+=1

            elif self.display == "losers":
                matchList = self.tournament.getLoserList()
                if len(matchList)>5:
                    offset = [(sizeY-100)/16-24,(sizeY-100)/8-24,(sizeY-100)/4-24, (sizeY-100)/4-24,(sizeY-
100)/2-24, (sizeY-100)/2-24, (sizeY-100)/2-24]
                elif len(matchList)>4:
                    offset = [(sizeY-100)/8-24, (sizeY-100)/4-24,(sizeY-100)/4-24, (sizeY-100)/2-24, (sizeY-
100)/2-24]
                elif len(matchList) >3:
                    offset = [(sizeY-100)/4-24, (sizeY-100)/4-24, (sizeY-100)/2-24, (sizeY-100)/2-24]
                elif len(matchList) >2:
                    offset = [24, (sizeY-100)/2-24, (sizeY-100)/2-24]
                else:
                    offset = [24, (sizeY-100)/2-24]
                i = 0
                tmp = None
                for list in matchList:
                    j = 0
                    for match in list:
                        if match != "Blank":
                            match.X = 15+180*i
                            match.Y = (((sizeY-100)/len(list)) * j) + (offset[i])
                            Matches.append(match)
                            j += 1
                            tmp = match
                    i+=1
                #if len(matchList) > 5:
                tmp.X = tmp.left.X + 45
                tmp.Y = tmp.left.Y + 80

            Champ = self.tournament.champ
            Champ.X = sizeX-200
            Champ.Y = sizeY/7 + 60
            if Champ.parent != None:
                Champ.parent.X = sizeX-200
                Champ.parent.Y = sizeY/7 + 200
        # make some random ellipses
        l = []
        x = [1,10]
        l.append( (x[self.tmp%2],1,2,2) )
        DrawData["Ellipses"] = l

        if self.start > -4 and self.start < 2:
            Time = str(int(math.fabs(self.start-1)))
        else:
            Time = '-1'
        return DrawData, Matches, Time, Champ

    def textentry(self, event=None):
        dlg = wx.TextEntryDialog(self, 'Enter file name to read','Team Entry')

        if dlg.ShowModal() == wx.ID_OK:
            num_teams = dlg.GetValue()
        else:
```

```
            return
        dlg.Destroy()

        if num_teams.isdigit():
            return False
        return num_teams

    def setTournament(self, tournament):
        self.tournament = tournament
        self.NewDrawing()
```

# Appendix C
## team.py

```
"""
team.py

created by Justin Kikuchi

Contains team, match, and tournament classes.
Backbone for tournament object.

"""
import wx
import json
import sys
sys.path.append("/System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/")
import xmlrpclib

useServer = True
class Team:
    name = None
    totalScore = 0
    losses = 0

    def __init__(self, name, logo):
        self.name = name
        if logo == "None":
            self.logo = 'images/calpoly.gif'
        else:
            self.logo = logo

class Match:
    left = None
    right = None
    score1 = 0
    score2 = 0
    winner = None
    loser = None
    time = 180
    server = xmlrpclib.ServerProxy('http://localhost:8089/robo')
    loserMatch = None
    loserMatchTeam = 0

    def __init__(self, parent, value, isLoser=False, team1=None, team2=None, X = 0, Y = 0, width = 170,
height = 60):
        self.parent = parent
        self.team1 = team1
        self.team2 = team2
        self.value = value
        self.isLoser = isLoser
        self.X = X
        self.Y = Y
        self.width = width
        self.height = height
        self.minSoundPlayed = False
        self.score1Sound = wx.Sound("sound_effects/score1.m4a")
        self.score2Sound = wx.Sound("sound_effects/score2.wav")
        self.minusSound = wx.Sound("sound_effects/minus.wav")
```

```
def setMatch(self, isChamp=False):
    self.team1.totalScore += self.score1
    self.team2.totalScore += self.score2
    if self.score1 > self.score2:
        self.winner = self.team1
        self.loser = self.team2
        self.team2.losses +=1
    else:
        self.winner = self.team2
        self.loser = self.team1
        self.team1.losses +=1
    if self.parent != None and not isChamp:
        if self.parent.right == self:
            self.parent.team1 = self.winner
        else:
            self.parent.team2 = self.winner

def getTeams(self):
    teams = []
    teams.append(self.team1)
    teams.append(self.team2)
    return teams

def getCoord(self):
    return wx.Rect(self.X, self.Y, self.width, self.height)

def checkClick(self, x, y):
    if(x > self.X and x < self.X+width):
        if(y < self.Y and y > self.Y-height):
            return True
    return False

def getRightCoord(self):
    return self.X + self.width, self.Y + self.height/2

def getLeftCoord(self):
    return self.X, self.Y + self.height/2

def getLabel(self):
    try:
        label = " Match #" + str(self.value) + "\n\t%-14s %-5d\n" %(self.team1.name,self.score1, )
    except AttributeError:
        if(self.right == None):
            label = " Match #" + str(self.value) + "\n\t%-14s %-5d\n" %("None",self.score1, )
        else:
            label = " Match #" + str(self.value) + "\n" + " Winner from " + str(self.right.value) + "\n"
    try:
        label = label + "\t%-14s %-5d" %(self.team2.name,self.score2, )
    except AttributeError:
        if(self.left == None):
            label = label + "\t%-14s %-5d" %("None",self.score2, )
        else:
            label = label + " Winner from " + str(self.left.value)
    return label

def getScore1(self):
```

```
        status = False
        if useServer:
            score1up = self.server.getScore1("hello")
            #print score1up
            if self.score1 < score1up:
                self.score1Sound.Play()
                status = True
            elif self.score1 > score1up:
                self.minusSound.Play()
                status = True
            self.score1 = score1up
        return status

    def getScore2(self):
        status = False
        if useServer:
            score2up = self.server.getScore2("hello1")
            #print score2up
            if self.score2 < score2up:
                self.score2Sound.Play()
                status = True
            elif self.score2 > score2up:
                self.minusSound.Play()
                status = True
            self.score2 = score2up
        return status

    def getLoser(self):
        return self.loser

    def getTime(self):
        if self.time > 0:
            if(self.time%60 < 10):
                return str(int(self.time/60)) + ":0"+str(self.time%60)
            else:
                return str(int(self.time/60)) + ":"+str(self.time%60)
        else:
            return "0:00"

class Tournament:
    def __init__(self, teamFile):
        self.teamFile = teamFile
        self.teamList = {}
        backup = self.getTeamsLogos()
        self.champ2 = Match(None, self.numMatches+3, width=195, height=110)
        self.champ = Match(self.champ2, self.numMatches+2, width=195, height=110)
        self.root = Match(self.champ, self.numMatches+1)
        self.lroot = Match(self.champ, self.numMatches+1, isLoser=True)
        self.champ.left = self.lroot
        self.champ.right = self.root
        self.addTraverse()
        self.buildFullLosers()
        if backup != None:
            self.loadState(backup)
```

```
def addMatch(self, parent, matchNum, isLoser=False, team1=None, team2=None):
    #print  "added: " + str(parent.value) + ", " + str(self.matchNumLookUp[self.matchNum])
    #return Match(parent, self.matchNumLookUp[self.matchNum])
    return Match(parent, matchNum, isLoser=isLoser,team1=team1, team2=team2)


def setLoser(self, insert):
    if insert.loserMatch != None:
        if insert.loserMatchTeam == 1:
            insert.loserMatch.team1 = insert.loser
        else:
            insert.loserMatch.team2 = insert.loser
        print "rewrote loser"
        return
    loserList = self.getLoserList()
    for list in loserList:
        for match in list:
            if match.team1 == None:
                match.team1 = insert.loser
                insert.loserMatch = match
                insert.loserMatchTeam = 1
                return match
            if match.team2 == None:
                match.team2 = insert.loser
                insert.loserMatch = match
                insert.loserMatchTeam = 2
                return match
    return None

def traverse(self):
    thislevel = [self.root]
    tree = []
    while thislevel:
        nextlevel = list()
        for n in thislevel:
            tree.append(n)
            if n.left:
                nextlevel.append(n.left)
            if n.right:
                nextlevel.append(n.right)
        thislevel = nextlevel
    tree.reverse()
    return tree

def traverseLoser(self):
    thislevel = [self.lroot]
    tree = []
    while thislevel:
        nextlevel = list()
        for n in thislevel:
            tree.append(n)
            #print n.value,
            if n.left:
                nextlevel.append(n.left)
            if n.right:
                nextlevel.append(n.right)
```

```
        #print
        thislevel = nextlevel
    tree.reverse()
    return tree

def findMatch(self, matchNum):
    matchList = self.getMatchList()
    loserList = self.getLoserList()
    for list in matchList:
        for match in list:
            if match != "Blank":
                if match.value == matchNum:
                    return match
    for list in loserList:
        for match in list:
            if match != "Blank":
                if match.value == matchNum:
                    return match
    if self.champ.value == matchNum:
        return self.champ
    if self.champ2.value == matchNum:
        return self.champ2
    return None

def getJsonMatch(self, match):
    teams = {}
    if match == None:
        return None
    if match.team1 != None:
        teams['name1'] = match.team1.name
        teams['score1'] = match.score1
    else:
        return None
    if match.team2 != None:
        teams['name2'] = match.team2.name
        teams['score2'] = match.score2
    else:
        return None

    data = json.dumps(teams)
    return data

def getMatchList(self):
    thislevel = [self.root]
    tree = []
    while thislevel:
        nextlevel = list()
        for n in thislevel:
            if(n != "Blank"):
                if n.right:
                    nextlevel.append(n.right)
                else:
                    nextlevel.append("Blank")
                if n.left:
                    nextlevel.append(n.left)
                else:
```

```
                nextlevel.append("Blank")
        tree.append(thislevel)
        thislevel = nextlevel
    tree.reverse()
    tree.pop(0)
    return tree

def getLoserList(self):
    thislevel = [self.lroot]
    tree = []
    while thislevel:
        nextlevel = list()
        for n in thislevel:
            if(n != "Blank"):
                if n.right:
                    nextlevel.append(n.right)
                if n.left:
                    nextlevel.append(n.left)
        tree.append(thislevel)
        thislevel = nextlevel
    tree.reverse()
    #tree.pop(0)
    return tree

def addTraverse(self):
    curr = 0
    thislevel = [self.root]
    while thislevel and self.numMatches>0:
        nextlevel = list()
        for n in thislevel:
            if n.left == None and self.numMatches>0:
                print "added left to " + str(n.value)
                self.numMatches = self.numMatches-1
                n.left = self.addMatch(n, self.numMatches+1)
                nextlevel.append(n.left)
        for n in thislevel:
            if n.right==None and self.numMatches>0:
                print "added right to " + str(n.value)
                self.numMatches = self.numMatches-1
                n.right = self.addMatch(n, self.numMatches+1)
                nextlevel.append(n.right)

        thislevel = nextlevel
    self.numMatches = self.num
    tree = self.traverse()
    i=0
    for team in self.teams:
        for match in tree:
            if match.left == None:
                if(match.team1 == None):
                    match.team1 = Team(team, self.logos[i])
                    self.teamList[match.team1.name] = match.team1
                    i +=1
                    print "added " + team + " to " + str(match.value)
                    break
                elif(match.team2 == None):
```

```
                match.team2 = Team(team, self.logos[i])
                self.teamList[match.team2.name] = match.team2
                i+=1
                print "added " + team + " to " + str(match.value)
                break
        elif match.right == None:
            if(match.team1 == None):
                match.team1 = Team(team, self.logos[i])
                self.teamList[match.team1.name] = match.team1
                i+=1
                print "added " + team + " to " + str(match.value)
                break

def buildFullLosers(self):
    curr = 0
    oddeven = 0
    thislevel = [self.lroot]
    self.numMatches -= 1
    while thislevel and self.numMatches>0:
        nextlevel = list()
        for n in thislevel:
            if n.left == None and self.numMatches>0:
                print "loser added left to " + str(n.value)
                self.numMatches = self.numMatches-1
                n.left = self.addMatch(n, self.numMatches+1, isLoser=True)
                nextlevel.append(n.left)
            if oddeven % 2:
                if n.right==None and self.numMatches>0:
                    print "loser added right to " + str(n.value)
                    self.numMatches = self.numMatches-1
                    n.right = self.addMatch(n, self.numMatches+1, isLoser=True)
                    nextlevel.append(n.right)
        print
        oddeven += 1
        thislevel = nextlevel
    tree = self.traverse()
    loserTree = self.traverseLoser()
    i = 0
    for match in loserTree:
        if not match.left:
            match.team1 = tree[i].getLoser()
            i+=1
        if not match.right:
            match.team2 = tree[i].getLoser()
            i+=1

def renumberMatches(self):
    winners = self.getMatchList()
    losers = self.getLoserList()
    for list in winners:
        for match in list:
            if match == 'Blank':
                list.remove(match)
    i = 0
    start = 0
    full = [1, 2, 4, 8, 16]
```

```
    j = len(winners)-1
    next = False
    for list in winners:
            if next:
                for match in list:
                    start += 1
                    match.value = start
                next = False
            if full[j] == len(list):
                start = list[len(list)-1].value
                if i < len(losers):
                    for match in losers[i]:
                        start += 1
                        match.value = start
                    i +=1
                    next = True
                    if i < len(losers):
                        for match in losers[i]:
                            start += 1
                            match.value = start
                        i +=1

            j -=1
    start +=1
    self.champ.value = start
    self.champ2.value = start + 1
    return

def getTeamsLogos(self):
    backup = None
    if 'backup' in self.teamFile:
        try:
            f = open(self.teamFile, 'r')
        except IOError:
            print "Cannot open file1"
            exit()
        backup = f
        self.teamFile = f.readline().rstrip('\n')
    try:
        f = open(self.teamFile, 'r')
    except IOError:
        print "Cannot open file2"
        exit()
    teams = f.readlines()
    f.close()
    teams = map(lambda s: s.strip(), teams)
    num_teams = len(teams)/2
    self.numMatches = num_teams -2
    self.num = self.numMatches

    self.teams = teams[::2]
    self.logos = teams[1::2]

    return backup

def saveState(self):
```

```
try:
    f = open("t_backup", 'w')
except IOError:
    print "Cannot open save file"
    return
f.write(self.teamFile+'\n')
winners = self.getMatchList()
losers = self.getLoserList()
for list in winners:
    for match in list:
        if match != "Blank":
            if match.team1 != None:
                f.write(match.team1.name+'\n')
                f.write(str(match.score1) + '\n')
            else:
                f.write("None\n")
            if match.team2 != None:
                f.write(match.team2.name+'\n')
                f.write(str(match.score2) + '\n')
            else:
                f.write("None\n")
for list in losers:
    for match in list:
        if match.team1 != None:
            f.write(match.team1.name+'\n')
            f.write(str(match.score1) + '\n')
        else:
            f.write("None\n")
        if match.team2 != None:
            f.write(match.team2.name+'\n')
            f.write(str(match.score2) + '\n')
        else:
            f.write("None\n")
if self.champ.team1 != None:
    f.write(self.champ.team1.name + '\n')
    f.write(str(self.champ.score1) + '\n')
else:
    f.write("None\n")
if self.champ.team2 != None:
    f.write(self.champ.team2.name + '\n')
    f.write(str(self.champ.score2) + '\n')
else:
    f.write("None\n")
if self.champ2.team1 != None:
    f.write(self.champ2.team1.name + '\n')
    f.write(str(self.champ2.score1) + '\n')
else:
    f.write("None\n")
if self.champ2.team2 != None:
    f.write(self.champ2.team2.name + '\n')
    f.write(str(self.champ2.score2) + '\n')
else:
    f.write("None\n")

f.close()
```

```
def loadState(self, f):
    winners = self.getMatchList()
    losers = self.getLoserList()
    for list in winners:
        for match in list:
            if match != "Blank":
                team = f.readline().rstrip('\n')
                if team != "None":
                    match.team1 = self.teamList[team]
                    match.score1 = int(f.readline().rstrip('\n'))
                team = f.readline().rstrip('\n')
                if team != "None":
                    match.team2 = self.teamList[team]
                    match.score2 = int(f.readline().rstrip('\n'))

    for list in losers:
        for match in list:
            team = f.readline().rstrip('\n')
            if team != "None":
                match.team1 = self.teamList[team]
                match.score1 = int(f.readline().rstrip('\n'))
            team = f.readline().rstrip('\n')
            if team != "None":
                match.team2 = self.teamList[team]
                match.score2 = int(f.readline().rstrip('\n'))

    team = f.readline().rstrip('\n')
    if team != "None":
        self.champ.team1 = self.teamList[team]
        self.champ.score1 = int(f.readline().rstrip('\n'))
    team = f.readline().rstrip('\n')
    if team != "None":
        self.champ.team2 = self.teamList[team]
        self.champ.score2 = int(f.readline().rstrip('\n'))
    team = f.readline().rstrip('\n')
    if team != "None":
        self.champ2.team1 = self.teamList[team]
        self.champ2.score1 = int(f.readline().rstrip('\n'))
    team = f.readline().rstrip('\n')
    if team != "None":
        self.champ2.team2 = self.teamList[team]
        self.champ2.score2 = int(f.readline().rstrip('\n'))

    f.close()
```

# Appendix D
## roboServer.py

```
#!/usr/bin/env python
# encoding: utf-8

"""
roboServer.py

created by Justin Kikuchi

Python server to service requests from a mobile device to a scoring system.

"""

from twisted.web import xmlrpc, server, resource
from twisted.internet import defer, threads
from twisted.internet import reactor
from twisted.cred import error

import json
import xmlrpclib
from xmlrpclib import Fault
import httplib

TCP_PORT = 8089
RPC_ROOT = 'robo'
rpcFacilitator = None

class XMLRPCFacilitator(xmlrpc.XMLRPC):
	""" The core for running deferred requests remotely. """

	score1 = 0
	score2 = 0
	currMatchJson = None
	def __init__(self):
		xmlrpc.XMLRPC.__init__(self)

	def render(self, request):
		m = getattr(self, 'render_' + request.method, None)
		request.setHeader('Access-Control-Allow-Origin', '*')
		if not m:
			from twisted.web.server import UnsupportedMethod
			raise UnsupportedMethod(getattr(self, 'allowedMethods', ()))

		return m(request)

	def xmlrpc_echo(self, echoString):
		""" Simple XMLRPC echo, responds on both client and server for testing. """
		sotelPrint(echoString)
		return echoString

	def xmlrpc_getScore1(self, string):
		#sotelPrint(string)
		return self.score1

	def xmlrpc_getScore2(self, string):
		#sotelPrint(string)
		return self.score2
```

```
def xmlrpc_setMatch(self, match):
        data = json.loads(match)
        if self.currMatchJson != None:
                data1 = json.loads(self.currMatchJson)
                if data["name1"] != data1["name1"] or data["name2"] != data1["name2"] :
                        self.score1 = 0
                        self.score2 = 0
                        data["score1"] = self.score1
                        data["score2"] = self.score2
                        print match
        self.currMatchJson = match
        self.score1 = data["score1"]
        self.score2 = data["score2"]
        return "yahhh"

def xmlrpc_getTeamNames(self):
        return self.currMatchJson

def xmlrpc_incScore1(self):
        print "inc1"
        self.score1 += 1
        data = json.loads(self.currMatchJson)
        data["score1"] = self.score1
        self.currMatchJson =  json.dumps(data)
        return self.currMatchJson

def xmlrpc_incScore2(self):
        print "inc2"
        self.score2 += 1
        data = json.loads(self.currMatchJson)
        data["score2"] = self.score2
        self.currMatchJson =  json.dumps(data)
        return self.currMatchJson

def xmlrpc_decScore1(self):
        print "dec1"
        self.score1 -= 1
        data = json.loads(self.currMatchJson)
        data["score1"] = self.score1
        self.currMatchJson =  json.dumps(data)
        return self.currMatchJson

def xmlrpc_decScore2(self):
        print "dec2"
        self.score2 -= 1
        data = json.loads(self.currMatchJson)
        data["score2"] = self.score2
        self.currMatchJson =  json.dumps(data)
        return self.currMatchJson

def xmlrpc_setScore1(self, num):
        print num
        self.score1 = num
        data = json.loads(self.currMatchJson)
        data["score1"] = self.score1
```

```
                self.currMatchJson =  json.dumps(data)
                return self.currMatchJson

        def xmlrpc_setScore2(self, num):
                print num
                self.score2 = num
                data = json.loads(self.currMatchJson)
                data["score2"] = self.score2
                self.currMatchJson =  json.dumps(data)
                return self.currMatchJson

def sotelPrint(message):
        print("<robo> " + message)

def initializeServer():
        rpcFacilitator = XMLRPCFacilitator()
        xmlrpc.addIntrospection(rpcFacilitator)
        root = resource.Resource()
        root.putChild(RPC_ROOT, rpcFacilitator)
        reactor.listenTCP(TCP_PORT, server.Site(root))
        sotelPrint("Server is all fired up!")
        reactor.run()

if __name__ == "__main__":
        initializeServer()
```

# Appendix E
## Mobile Scoring Page

```
<!DOCTYPE html>
<html>
	<head>
		<!-- metadata -->
		<meta charset="utf-8" \>
		<meta name="viewport" content="width=device-width, initial-scale=1, maximum-
scale=1">

		<title>
			Roborodentia Mobile
		</title>

		<!-- css jazz. -->
		<link href="style.css" rel="stylesheet" type="text/css"\>

		<!-- javascript! -->
		<script src="jquery-1.7.2.min.js"></script>
		<script src="./robo.js"></script>
	</head>

	<body>
		<!-- We'll display a spinner so that people don't click things twice before the server loads.
-->
		<div id="spinnerContainer">
			<div id="spinner">
				 
			</div>
		</div>

		<div id="main">
			<!-- first team's data. -->
			<div class="team one">
				<div class="minus button">
					-
				</div>

				<div class="name">
					Team One
				</div>

				<div class="plus button">
					+
				</div>

				<input type="text" class="score" value="0" />
			</div>

			<!-- second team's data. -->
			<div class="team two">
				<div class="minus button">
					-
				</div>

				<div class="name">
					Team Two
				</div>
```

```
                                        <div class="plus button">
                                                +
                                        </div>

                                        <input type="text" class="score" value="0" />
                                </div>
                        </div>
                </body>
</html>

var xmlrpcStart = '<?xml version="1.0"?><methodCall><methodName>';
var xmlrpcEnd = '</methodName><params></params></methodCall>';

var xmlrpcMidParams = '</methodName><params><param><value><int>';
var xmlrpcEndParams = '</int></value></param></params></methodCall>';
var xmlrpcURL = "http://129.65.109.24:8089/robo";

function handleServerResponse(data, text, jqXHR)
{
        var result = $(data).find("string").text();
        console.log(result);
        result = $.parseJSON(result);
        $('.team.one .name').text(result.name1);
        $('.team.two .name').text(result.name2);
        $('.team.one .score').val(result.score1);
        $('.team.two .score').val(result.score2);
}

function getTeamData() {
        $.ajax({
                url: xmlrpcURL,

                type: "POST",

                data: xmlrpcStart + 'getTeamNames' + xmlrpcEnd,

                success: function(data, text, jqXHR) {
                        handleServerResponse(data, text, jqXHR);
                },

                error: function() {
                        console.log("err");
                },

                beforeSend: function() {
                        $('#spinnerContainer').show();
                },

                complete: function() {
                        $('#spinnerContainer').hide();
                }
        });
}

$(document).ready(function () {
```

```
$('#spinnerContainer').hide();
getTeamData();
//setInterval("getTeamData()", 10000);

$('.team.one .plus').on("click", function(ev) {
	console.log("team one plus");

	$.ajax({
		url: xmlrpcURL,

		type: "POST",

		data: xmlrpcStart + 'incScore1' + xmlrpcEnd,

		success: function(data, text, jqXHR) {
			handleServerResponse(data, text, jqXHR);
		},

		error: function() {
			console.log("err");
		},

		beforeSend: function() {
			$('#spinnerContainer').show();
		},

		complete: function() {
			$('#spinnerContainer').hide();
		}
	});
});

$('.team.two .plus').on("click", function(ev) {
	console.log("team two plus");

	$.ajax({
		url: xmlrpcURL,

		type: "POST",

		data: xmlrpcStart + 'incScore2' + xmlrpcEnd,

		success: function(data, text, jqXHR) {
			handleServerResponse(data, text, jqXHR);
		},

		error: function() {
			console.log("err");
		},

		beforeSend: function() {
			$('#spinnerContainer').show();
		},

		complete: function() {
			$('#spinnerContainer').hide();
```

```
                }
        });
});

$('.team.one .minus').on("click", function(ev) {
        console.log("team one minus");

        $.ajax({
                url: xmlrpcURL,

                type: "POST",

                data: xmlrpcStart + 'decScore1' + xmlrpcEnd,

                success: function(data, text, jqXHR) {
                        handleServerResponse(data, text, jqXHR);
                },

                error: function() {
                        console.log("err");
                },

                beforeSend: function() {
                        $('#spinnerContainer').show();
                },

                complete: function() {
                        $('#spinnerContainer').hide();
                }
        });
});

$('.team.two .minus').on("click", function(ev) {
        console.log("team two minus");

        $.ajax({
                url: xmlrpcURL,

                type: "POST",

                data: xmlrpcStart + 'decScore2' + xmlrpcEnd,

                success: function(data, text, jqXHR) {
                        handleServerResponse(data, text, jqXHR);
                },

                error: function() {
                        console.log("err");
                },

                beforeSend: function() {
                        $('#spinnerContainer').show();
                },

                complete: function() {
                        $('#spinnerContainer').hide();
```

```
                }
        });
});

$('.team.one .score').on("change", function(ev) {
        console.log("changed!");
        console.log(xmlrpcStart + 'setScore1' + xmlrpcMidParams +
                        $('.team.one .score').val() + xmlrpcEndParams);
        $.ajax({
                        url: xmlrpcURL,

                        type: "POST",

                        data: xmlrpcStart + 'setScore1' + xmlrpcMidParams +
                        $('.team.one .score').val() + xmlrpcEndParams,

                        success: function(data, text, jqXHR) {
                                handleServerResponse(data, text, jqXHR);
                        },

                        error: function() {
                                console.log("err");
                        },

                        beforeSend: function() {
                                $('#spinnerContainer').show();
                        },

                        complete: function() {
                                $('#spinnerContainer').hide();
                        }
                });
});

$('.team.two .score').on("change", function(ev) {
        console.log("changed!");
        console.log(xmlrpcStart + 'setScore2' + xmlrpcMidParams +
                        $('.team.two .score').val() + xmlrpcEndParams);
        $.ajax({
                        url: xmlrpcURL,

                        type: "POST",

                        data: xmlrpcStart + 'setScore2' + xmlrpcMidParams +
                        $('.team.two .score').val() + xmlrpcEndParams,

                        success: function(data, text, jqXHR) {
                                handleServerResponse(data, text, jqXHR);
                        },

                        error: function() {
                                console.log("err");
                        },

                        beforeSend: function() {
                                $('#spinnerContainer').show();
```

```
                                        },

                                        complete: function() {
                                                $('#spinnerContainer').hide();
                                        }
                                });
        });

});

* {
        padding: 0px;
        margin: 0px;
}

body {
        color: #fdfdfd;
        background: url('./img/carbon.png') repeat top left #1d1d1d;
        font-family: "Helvetica Neue", "Helvetica", "Arial", sans-serif;
}

#main {
        position: relative;
        left: -150px;
        display: block;
        width: 300px;
        margin-left: 50%;
}

.team {
        display: block;
        margin: 20px 0px 40px 0px;
        height: 120px;
}

.name {
        display: block;
        width: 196px;
        height: 50px;
        line-height: 50px;
        float: left;
        text-align: center;
}

.score {
        display: block;
        width: 296px;
        height: 70px;
        line-height: 70px;
        font-weight: bold;
        font-size: 64px;
        text-align: center;
        background: transparent;
        border: 0px;
        color: #fdfdfd;
}
```

```
.button {
	display: block;
	width: 50px;
	height: 50px;
	background: -webkit-gradient(linear, left bottom, left top,
		color-stop(0.21, rgb(67,72,76)),
		color-stop(0.61, rgb(87,94,99)),
		color-stop(0.75, rgb(96,106,107)));
	border: 1px solid #262c2f;
	border-radius: 5px;
	line-height: 50px;
	text-align: center;
	font-size: 32px;
	font-weight: bold;
	float: left;
}

.button:hover {
	cursor: pointer;
}

.button.minus {
	clear: both;
}

#spinner {
	background: #ffffff url('./img/spinner_transparent.gif') no-repeat center center;
	border: 1px solid #dedede;
	box-shadow: 0px 0px 7px #888888;
	border-radius: 7px 7px;
	width: 180px;
	height: 180px;
	position: absolute;
	margin-left: 50%;
	left: -90px;
	top: 80px;
}
#spinnerContainer {
	z-index: 100;
	display: block;
	width: 100%;
	height: 100%;
	top: 0px;
	left: 0px;
	bottom: 0px;
	position: absolute;
	background: rgba(255, 255, 255, 0.4);
}
```