Bernard Dickens

Thursday, October 28, 2010

CSC: 106 – Intro

Mini-MAR 2.0 BETA Write Up

**\_\_version\_\_ = "2.0b1";**

**\_\_author\_\_ = "Xunnamius of Dark Gray";**

This is the declaration of the typical python “superglobals” found in most python programs.

**import re, sys, hashlib, time, os, threading;**

**from string import ascii\_uppercase, ascii\_letters;**

**from random import randint;**

**from math import fabs, floor;**

**from httplib import BadStatusLine;**

**from collections import OrderedDict;**

**from NetworkInterface import NetworkInterface;**

**from DisplayInterface import Display;**

**from Challenger import Challenger;**

Importing all of the modules I’ll need to make this game work, including custom modules I’ve made for other projects.

**ascii\_uppercase = ascii\_uppercase[::-1];**

Reverses the string-list of letters

**# Establish our connection object**

**conn1 = NetworkInterface();**

**conn1.baseSite = "dignityice.com";**

**conn1.target = "/dg/Xunnamius/house2/pyGameInterface.php";**

**# Set up our environment**

**loggedIn = False;**

**time.clock(); # Clear the timer**

**# Sound-off!**

**# No color support in this version for now, so the Display.\*Msg distinctions are only code-deep**

**Display.sysMsg('DO NOT RUN THIS PROGRAM IN THE PYTHON INTERACTIVE CONSOLE (IDLE)! DO NOT!\n\n');**

**Display.sysMsg('Mini-MAR BETA version', \_\_version\_\_);**

**conn1.status();**

**Display.sysMsg('Bernard Dickens - Project II -', \_\_author\_\_);**

This code is simply establishing an environment with which to work in

**# Exit MAR**

**def MAR\_exit(status=0):**

**if(status == 0): Display.gameMsg('Program Terminated.');**

**else: Display.errorMsg('Program Terminated Unexpectedly ('+str(status)+')!');**

**Display.evnMsg('(C) Copyright 2010 - Dark Gray. All Rights Reserved.');**

**Display.pause();**

**os.\_exit(status);**

**# Is this JSON-like data?**

**def is\_JSON(data):**

**data = str(data).strip();**

**if(data[0] == '{' and data[-1] == '}'): return True;**

**return False;**

**# Decode JSON-like data**

**def json\_decode(json\_data, debug=False):**

**try:**

**if(not json\_data): raise BadStatusLine;**

**json\_data = str(json\_data).strip();**

**if(debug): print json\_data;**

**elif(is\_JSON(json\_data)): return eval(json\_data);**

**else: return json\_data;**

**except(Exception, BadStatusLine):**

**Display.errorMsg("Bad json\_data received"+('' if not debug else ' ('+str(debug)+')')+". Unable to continue.");**

**Display.errorMsg(">>", str(json\_data));**

**MAR\_exit(1);**

**# Wait for a response from the server, depending on the parameters**

**def block\_on\_challenge(ignore\_response, params, at\_end=1, stall\_time=60, debug=False, eval\_response=True):**

**global conn1, heartbeat;**

**if(not conn1 or not heartbeat): MAR\_exit(2);**

**xtime = 0;**

**# Give them 60 seconds**

**while(xtime < stall\_time):**

**if(xtime == (stall\_time-30)): Display.sysMsg('30 seconds remaining...');**

**elif(xtime == (stall\_time-10)): Display.sysMsg('10 seconds remaining...');**

**sleeptime = time.clock();**

**if(xtime % 5 == 0):**

**# Setup our internal environment**

**conn1.params = params;**

**conn1.makeNulls = True;**

**response = conn1.request();**

**# print 'tempresp -> ', response**

**if(response and response != ignore\_response):**

**if(eval\_response): response = json\_decode(response, debug);**

**conn1.makeNulls = False;**

**return response;**

**sleeptime = 1 - (time.clock() - sleeptime);**

**if(sleeptime > 0): time.sleep(sleeptime);**

**xtime = xtime + 1;**

**# We timed-out!**

**else:**

**if(at\_end == 1):**

**Display.evnMsg('Request TIMED OUT. Destroying room...');**

**conn1.params = {"u":u, "p":p, "python":1, "type":"kil", "SID":SID};**

**response = conn1.request();**

**Display.sysMsg('Room Destroyed!');**

**heartbeat.pauseFlag = False;**

**conn1.makeNulls = False;**

**return False;**

**# Small distance algorithm**

**# Calculates the relative distance between two elements in an array (by Xunnamius of Dark Gray)**

**def distanceAlgorithm(haystack, targetNeedle, needle, slightDistMod, farDistMod):**

**# Target array, element1, element2**

**target = (haystack, needle, targetNeedle);**

**# These percents modify the mangitude of the distance detection.**

**# The first percent denotes slight distance while the second one denotes the distinction**

**# between moderate and far distance.**

**# The higher the percents, the biggerer their respective margins-of-error will be.**

**modifiers = (slightDistMod, farDistMod);**

**length = len(target[0]);**

**dist = target[0].index(target[1]) - target[0].index(target[2]);**

**fdist = fabs(dist);**

**# Based on the distance from the answer, print a different result (to the same line)**

**if(fdist <= length\*modifiers[0]/100): return 'slightly off.';**

**elif(dist > 0):**

**if(fdist <= length\*modifiers[1]/100): return 'too low.';**

**else: return 'WAY too low!';**

**else:**

**if(fdist <= length\*modifiers[1]/100): return 'too high.';**

**else: return 'WAY too high!';**

**# Attempt to activate a power**

**def activatePower(power, data):**

**# print 'power', power;**

**return data;**

**# Evalute the current standing of the specials array**

**def evaluateQueue(data, user=True):**

**# User=True means it's the user's data, False means the opponent's data!**

**# print data;**

**return data;**

**# Evaluate the specials array one last time, and then push the client's new stat data to the server**

**# Tell user the battle results**

**def finalEvaluation(data, opponent):**

**# print data;**

**conn1.params = {"u":u, "p":p, "python":1, "type":"upd", "SID":SID, 'data':'$Win'+data['username']};**

**conn1.request();**

**if(data['userdata']['victory'] != -1):**

**if(data['userdata']['victory'] == True):**

**data2 = {'XP':floor(data['userdata']['remainingTurns'] / 4), 'losses':0, 'wins':1};**

**Display.playerMsg('Good job,', data['username']+'!');**

**Display.gameMsg("You've defeated", opponent, 'in', (data['userdata']['maxTurns'] - data['userdata']['remainingTurns']), 'turns.');**

**Display.gameMsg("You've earned", data['userdata']['remainingTurns'], 'experience points!');**

**elif(data['userdata']['victory'] == False):**

**data2 = {'XP':1, 'losses':1, 'wins':0};**

**Display.errorMsg("You've failed", data['username']+".");**

**Display.errorMsg("You've lost to", opponent+".");**

**Display.errorMsg("The correct answer was " + data['userdata']['answer'] + ".");**

**conn1.params = {"u":u, "p":p, "python":1, "type":"psh", "SID":SID, 'data':data2};**

**response = conn1.request();**

**if(response == 'Updated'): Display.gameMsg('Server updated successfully. Game Over.');**

**else: Display.errorMsg('Server update failed ('+response+')! Game Over.');**

**def checkForCommands(response, actionQueue, opponent):**

**global conn1;**

**doBreak = False;**

**if(response == '$Skipped:'+actionQueue['username']):**

**conn1.params = {"u":u, "p":p, "python":1, "type":"upd", "SID":SID, 'data':'$ClearGameData'};**

**conn1.request();**

**Display.errorMsg('Temporarily desynched from the server. Your actions were discarded.');**

**elif(response == '$Skipped:'+opponent):**

**Display.errorMsg('Your opponent has desynched from the server. You have been declared winner.');**

**doBreak = True;**

**actionQueue['userdata']['victory'] = True;**

**finalEvaluation(actionQueue, opponent);**

**elif(response == '$Updated'): Display.gameMsg("Synchronization complete.");**

**elif(response == '$Win:'+opponent):**

**actionQueue['userdata']['victory'] = False;**

**finalEvaluation(actionQueue, opponent);**

**doBreak = True;**

**elif(response == '$Win:'+actionQueue['username']):**

**actionQueue['userdata']['victory'] = True;**

**finalEvaluation(actionQueue, opponent);**

**doBreak = True;**

**else:**

**if(response == '$Alive:'+opponent or response == '$Yes' or response == '$Skipped:'+opponent):**

**conn1.params = {"u":u, "p":p, "python":1, "type":"upd", "SID":SID, 'data':'$Die'};**

**conn1.request();**

**Display.errorMsg('Due to your opponent\'s excessive lag, the game has been canceled.');**

**elif(response == '$Alive:'+actionQueue['username']):**

**conn1.params = {"u":u, "p":p, "python":1, "type":"upd", "SID":SID, 'data':'$Die'};**

**conn1.request();**

**Display.errorMsg('Due to your excessive lag, the game has been canceled.');**

**elif(response == '$Die'):**

**Display.errorMsg('The game has been canceled unexpectedly. Probably due to lag.');**

**else:**

**Display.errorMsg('Bad response from server. Game Over.');**

**Display.errorMsg('>>', response);**

**actionQueue['userdata']['victory'] = -1;**

**doBreak = True;**

**return (actionQueue, doBreak);**

These are the main functions that allow MAR to operate in a more “object based” fashion. The methods are liberally commented above.

**# Start a head2head battle**

**def MAR\_battle(opponent, first):**

**print '';**

**global conn1, heartbeat, loggedIn, userdict, userpowers, ascii\_uppercase;**

**if(not conn1 or not loggedIn or not heartbeat or not userdict or not userpowers or not ascii\_uppercase): MAR\_exit(3);**

**#Letters need re-reversing for some reason. Bad python scoping system is bad and horrible.**

**ascii\_uppercase = ascii\_uppercase[::-1];**

**# Finish setting up our environment**

**actionQueue = {'username':userdict['username'], 'specialdata':{},**

**'userdata':{**

**'guess':None,**

**'miniTurns':0,**

**'remainingTurns':27,**

**'maxTurns':28,**

**'letterNum':1,**

**'answer':None,**

**'answerStatus1':None,**

**'answerStatus2':None,**

**'skipped': False,**

**'victory': False**

**}};**

**# Structure the user's powers into a workable menu object**

**powerStructure = [('guess', -1), ('skip', -2)];**

**for power in userpowers.keys(): powerStructure.insert(0, (str(power).lower(), str(power)));**

**powerStructure = OrderedDict(powerStructure);**

**# Begin the game**

**while(actionQueue['userdata']['remainingTurns'] > 0 and not actionQueue['userdata']['victory']):**

**if(actionQueue['userdata']['letterNum'] <= 4 and not actionQueue['userdata']['victory']):**

**# We're up!**

**if(first):**

**if(not actionQueue['userdata']['skipped']):**

**# Set up our environment**

**if(actionQueue['userdata']['answer'] == None):**

**actionQueue['userdata']['answer'] = ascii\_uppercase[randint(0, 25)];**

**Display.gameMsg('For you, this is letter '+str(actionQueue['userdata']['letterNum'])+' of 4.');**

**Display.gameMsg('You have 60 seconds until your turn is automatically skipped.');**

**# Prompt for selection of powers or guess (60 second time limit)**

**sleeptime = time.clock();**

**i = Display.menu(powerStructure, initMsg='Your move:', prefix='- ', time\_limit=60, limit\_phrase='skip');**

**# User wants to guess normally**

**if(i == -1):**

**if(actionQueue['userdata']['guess'] and actionQueue['userdata']['answer']): Display.gameMsg('-> You previous guess was:', actionQueue['userdata']['guess']);**

**actionQueue['userdata']['guess'] = Display.timed\_input('Guess a letter, '+actionQueue['username']+': ', 'skip', 60 - (time.clock() - sleeptime));**

**if(actionQueue['userdata']['guess'] == 'skip'): actionQueue['userdata']['skipped'] = True;**

**elif(actionQueue['userdata']['guess'] in ascii\_letters):**

**if(actionQueue['userdata']['guess'].upper() == actionQueue['userdata']['answer']):**

**Display.gameMsg("You've guessed correctly! (Total", actionQueue['userdata']['miniTurns'], "turns)");**

**actionQueue['userdata']['letterNum'] = actionQueue['userdata']['letterNum'] + 1;**

**actionQueue['userdata']['answerStatus1'] = actionQueue['username']+" guessed correctly and is onto the next letter! (Letter "+str(actionQueue['userdata']['letterNum'])+" of 4)";**

**actionQueue['userdata']['answerStatus2'] = '';**

**actionQueue['userdata']['miniTurns'] = 0;**

**actionQueue['userdata']['answer'] = None;**

**else:**

**dist = distanceAlgorithm(ascii\_uppercase, actionQueue['userdata']['guess'].upper(), actionQueue['userdata']['answer'], 4.0, 40);**

**actionQueue['userdata']['answerStatus1'] = actionQueue['username']+' guessed incorrectly!';**

**actionQueue['userdata']['answerStatus2'] = "Player's guess was "+dist;**

**print Display.errorWrapper() + "Incorrect! Your guess was", dist;**

**else:**

**Display.errorMsg("That's not a letter,", actionQueue['username']+". You've lost two turns for that!");**

**actionQueue['userdata']['answerStatus1'] = actionQueue['username']+' entered an invalid response and was penalized!\n What a fool!';**

**actionQueue['userdata']['remainingTurns'] = actionQueue['userdata']['remainingTurns'] - 2;**

**actionQueue['userdata']['miniTurns'] = actionQueue['userdata']['miniTurns'] + 1;**

**# User wants to skip their turn**

**elif(i == -2): actionQueue['userdata']['skipped'] = True;**

**# A power was chosen, attempt to add it to the specialdata array**

**else:**

**Display.errorMsg('Specials don\'t work yet. Sorry!');**

**actionQueue['userdata']['skipped'] = True;**

**# actionQueue['specialdata'] = activatePower(userpowers[i], actionQueue['specialdata']);**

**# Evaluate specialdata array & victory conditions**

**actionQueue = evaluateQueue(actionQueue);**

**Display.gameMsg('Turn ended!');**

**Display.gameMsg("You have", (actionQueue['userdata']['remainingTurns'] if actionQueue['userdata']['remainingTurns'] > 0 else 'no'), "turns remaining.");**

**Display.gameMsg("Synchronizing with server...");**

**conn1.params = {"u":u, "p":p, "python":1, "type":"upd", "SID":SID, 'data':actionQueue};**

**response = conn1.request();**

**# print 'resp:', response**

**if(response == '$UpdateFailed'): Display.errorMsg('Internal Error: server failed to update properly! Report this!');**

**if(response == '$LookupError'): Display.errorMsg('Internal Error: server-user lookup error! Report this!');**

**else:**

**actionQueue, doBreak = checkForCommands(response, actionQueue, opponent);**

**if(doBreak): break;**

**actionQueue['userdata']['remainingTurns'] = actionQueue['userdata']['remainingTurns'] - 1;**

**actionQueue['userdata']['miniTurns'] = actionQueue['userdata']['miniTurns'] + 1;**

**actionQueue['userdata']['skipped'] = False;**

**# They're up! (wait for 60[+10] seconds)**

**else:**

**Display.gameMsg('Waiting on', opponent+' (~60 seconds until timeout)...');**

**response = block\_on\_challenge('$NoData', {"u":u, "p":p, "python":1, "type":"bat4", "SID":SID, 'opponent':opponent}, at\_end=0, stall\_time=70);**

**if(is\_JSON(response)):**

**Display.gameMsg('->', (opponent+' skipped his turn?! Foolish!' if response['userdata']['answerStatus1']==None else response['userdata']['answerStatus1']));**

**evaluateQueue(response, False);**

**Display.sysMsg('Opponent\'s turn ended. Your move!');**

**else:**

**# Send the "is alive ne?" packet**

**Display.gameMsg('No response from opponent, clearing victory conditions...');**

**conn1.params = {"u":u, "p":p, "python":1, "type":"upd", "SID":SID, 'data':'$Alive:'+opponent};**

**conn1.request();**

**response = block\_on\_challenge('$NoData', {"u":u, "p":p, "python":1, "type":"bat4", "SID":SID, 'opponent':opponent, "ignore":'$Alive:'+opponent}, at\_end=0, stall\_time=10, eval\_response=False);**

**if(response == '$Yes'):**

**Display.gameMsg('Opponent\'s turn was skipped.');**

**conn1.params = {"u":u, "p":p, "python":1, "type":"upd", "SID":SID, 'data':'$Skip:'+actionQueue['username']};**

**conn1.request();**

**time.sleep(3);**

**elif(response == False or response[0] == '{'):**

**Display.gameMsg('Opponent has disconnected from the server. You win by default.');**

**actionQueue['userdata']['victory'] = True;**

**finalEvaluation(actionQueue, opponent);**

**break;**

**else:**

**actionQueue, doBreak = checkForCommands(response, actionQueue, opponent);**

**if(doBreak): break;**

**#Change up the order**

**first = not first;**

**else:**

**actionQueue['userdata']['victory'] = True;**

**finalEvaluation(actionQueue, opponent);**

**break;**

**else:**

**if(not actionQueue['userdata']['victory']): finalEvaluation(actionQueue, opponent);**

**Display.sysMsg('Cleaning up room...');**

**conn1.params = {"u":u, "p":p, "python":1, "type":"kil", "SID":SID};**

**Display.sysMsg('Room destroyed successfully.');**

**heartbeat.pauseFlag = False;**

**conn1.makeNulls = False;**

This is mini-MAR’s main head2head function. The code is commented above.

**# Begin the main python code**

**try:**

Begin the main python code

**while(True):**

**# Staying away from my complex menu object, since explaining it in a write up would be... well, complex.**

**i = Display.menu(OrderedDict([('free play', 0), ('local play', 1), ('online play', 2), ('exit', 3)]), initMsg='Select Your Gameplay Mode:', prefix='# ');**

Ask the user what they want to do.

**# Free play mode, which is basically the game as it was originally intended**

**if(i == 0):**

**gameNum = 0;**

**state = 'y';**

**name = Display.cooked\_input("What's your name? "); # Python doesn't even have a do-while?! Wow. Really? ... Really? What a waste of code.**

**while not re.match('^[0-9a-zA-Z\_]{4}[0-9a-zA-Z\_]\*$', name) and 4 <= name <= 25: # ^[0-9a-zA-Z\_]{4,25}$ wasn't working for some odd reason >.<**

**Display.errorMsg('Invalid name! Numbers/Letters only (between 4 and 25 chars).');**

**name = Display.cooked\_input("What's your name? ");**

**Display.gameMsg('Welcome to the Letter Guessing Game,', name+'!');**

**while state == 'y':**

**# Finish setting up our environment**

**tries = 0;**

**maxTries = 7;**

**ascii\_uppercase = ascii\_uppercase[::-1];**

**answer = ascii\_uppercase[randint(0, 25)];**

**gameNum = gameNum + 1;**

**Display.gameMsg('For you, this is game #'+str(gameNum)+'.');**

**# Begin the game**

**while tries < maxTries:**

**guess = Display.cooked\_input('Guess a letter, '+name+': ');**

**if(guess in ascii\_letters):**

**if(guess.upper() == answer):**

**Display.gameMsg("You've guessed correctly! (Total", (tries + 1), "turns)");**

**answer = -1;**

**break;**

**else:**

**print Display.errorWrapper() + "Incorrect! Your guess was", distanceAlgorithm(ascii\_uppercase, guess.upper(), answer, 4.0, 40);**

**else:**

**Display.errorMsg("That's not a letter,", name+". You've lost two turns for that!");**

**maxTries = maxTries - 2;**

**report = maxTries - (tries + 1);**

**Display.gameMsg("You have", (report if report > 0 else 'no'), "tries remaining.");**

**tries = tries + 1;**

**if(answer == -1): Display.playerMsg('Good job!');**

**else:**

**Display.errorMsg("You failed", name+".");**

**Display.errorMsg("The correct answer was " + answer + ".");**

**state = Display.cooked\_input('Try again? (y/n) ');**

Begin the “free play”mode. The lines above are commented where necessary.

**# Play mode which would allow players to grab their stats from the server and play a local head2head match**

**elif(i == 1): Display.gameMsg('Unfortunately, this gameplay mode is not available yet!');**

A play mechanic that is not yet available.

**# We're in Online Play mode!**

**elif(i == 2):**

**# Tell the user that we're asking the server for a new encryption key**

**Display.sysMsg("Handshaking with server..."); # Go pre-emptive handshake!**

**# Ask for, obtain, and store the the encryption key**

**switch = False;**

**response = conn1.requestKey();**

**if(response != False):**

**switch = True;**

**KEY = response[0];**

**SID = response[2];**

**else: Display.errorMsg('Connection timed out. Please try again.');**

**while(not loggedIn and switch):**

**i = Display.menu(OrderedDict([('login', 0), ('register', 1), ('back', 2)]), initMsg='Select an option:', prefix='# ');**

**# We're attempting to log someone in**

**if(i == 0):**

**u = Display.cooked\_input("> Username: ");**

**p = Display.getpass("> Password:");**

**# Tell the user that we're doing things**

**Display.sysMsg("\n Conversing with server...");**

**# Package the new data**

**conn1.params = {"u":u, "python":1, "type":"", "SID":SID};**

**# 1. Split the key in half**

**# 2. Place chunks at both the front and the end (double salt)**

**# 3. SHA-1 the whole thing again**

**eKeyPiece1 = KEY[0:20];**

**eKeyPiece2 = KEY[20:40];**

**p = str(hashlib.sha1(eKeyPiece1 + hashlib.sha1(hashlib.md5(p).hexdigest()).hexdigest() + eKeyPiece2).hexdigest());**

**u = str(hashlib.sha1(eKeyPiece1 + u + eKeyPiece2).hexdigest());**

**conn1.params["u"] = u;**

**conn1.params["p"] = p;**

**conn1.params["type"] = "lin";**

**# Authenticate the user's information**

**response = conn1.request();**

**# Alert the user of the result, and display the proper options accordingly**

**if response != False:**

**if response == "Approved":**

**Display.playerMsg("Login Successful.");**

**# Grab the user's player data and store it as a dictionary**

**conn1.params = {"u":u, "p":p, "python":1, "type":"pul", "SID":SID};**

**conn1.makeNulls = True;**

**response = conn1.request();**

**if response != False:**

**userdict = json\_decode(response);**

**userpowers = userdict['powers'];**

**temp\_arr = {};**

**for power in userpowers:**

**temp\_arr[power['name']] = {'desc':power['desc'], 'json':power['json'], 'sp':power['sp']};**

**userpowers = temp\_arr;**

**userdict = userdict['userdata'];**

**loggedIn = True;**

**conn1.makeNulls = False;**

**# Welcome the user**

**Display.playerMsg("Welcome to Online Mini-MAR,", userdict["username"]+"!");**

**# Report any bugs you may find in this program to trefighter2334@aol.com ASAP, thanks!**

**Display.adminMsg("Please report any bugs you may find to trefighter2334@aol.com");**

**# Start polling the server for challenge requests**

**heartbeat = Challenger(conn1.baseSite, conn1.target, {"u":u, "p":p, "python":1, "type":"bat", "SID":SID});**

**while(loggedIn):**

**i = Display.menu(OrderedDict([('challenge', 0), ('whois', 1), ('powers', 4), ('settings', 5), ('logout', 2), ('exit', 3)]), initMsg='Available Options:', prefix='# ');**

**# Challenge someone! Woooo!**

**if(i == 0):**

**heartbeat.pauseFlag = True;**

**conn1.params = {"u":u, "p":p, "python":1, "type":"cha", "SID":SID};**

**conn1.params['chal'] = Display.cooked\_input('Enter the username of the player you\'d like to challenge: ');**

**Display.sysMsg('Conversing with the server...');**

**response = conn1.request();**

**# Waiting for Challengee to accept/deny**

**if(response == "$ChallengerWaiting"):**

**Display.gameMsg('Waiting for a response from', conn1.params['chal']+'...');**

**response = block\_on\_challenge('$NoDice', {"u":u, "p":p, "python":1, "type":"bat2", "SID":SID});**

**if(response != False):**

**first = (False if response['opponent'] == response['first'] else True);**

**Display.evnMsg('Challenge Accepted! '+(response['opponent']+' is' if not first else 'You\'re') + ' up first.');**

**Display.sysMsg('Preparing battlefield...');**

**time.sleep(5);**

**MAR\_battle(response['opponent'], first);**

**else: Display.errorMsg("Connection to server was dropped. Please try again.");**

**# Waiting for challenger's client to acknowledge our acceptance of their initial challenge**

**elif(response == "$ChallengedWaiting"):**

**Display.sysMsg('Waiting for acknowlegement...');**

**response = block\_on\_challenge('$NoDice', {"u":u, "p":p, "python":1, "type":"bat3", "SID":SID});**

**if(response != False):**

**first = (False if response['opponent'] == response['first'] else True);**

**Display.evnMsg('Challenge Acknowledged! '+(response['opponent']+' is' if not first else 'You\'re') + ' up first.');**

**Display.sysMsg('Preparing battlefield...');**

**time.sleep(5);**

**MAR\_battle(response['opponent'], first);**

**else: Display.errorMsg("Connection to server was dropped. Please try again.");**

**else:**

**if(response == "$NoMatch"): Display.errorMsg('Invalid or bad username.');**

**elif(response == "$Busy"): Display.errorMsg('That user is already in a game!');**

**elif(response == "$Illegal"): Display.errorMsg('You cannot challenge yourself!');**

**elif(response == "$AlreadyWaiting"): Display.errorMsg('You already have a challenge pending. Try again later.');**

**else:**

**if(not response): Display.errorMsg("Connection to server was dropped. Please try again.");**

**else:**

**Display.errorMsg("Error. The response received from server was unrecognizable.");**

**Display.errorMsg(">>", response);**

**heartbeat.pauseFlag = False;**

**# Request information on another user**

**elif(i == 1):**

**conn1.params = {"u":u, "p":p, "python":1, "type":"req", "SID":SID};**

**conn1.params['stat'] = Display.cooked\_input('Enter the username of the player you\'d like information on: ');**

**Display.sysMsg('Conversing with the server...');**

**response = conn1.request();**

**if(response != False):**

**if(response == "BadUser"): Display.errorMsg('Invalid or bad username.');**

**else: Display.gameMsg(Display.newline+' '+response, '\n');**

**else: Display.errorMsg('Your request failed. Please try again.');**

**# Logout of the program and return to the main options screen**

**elif(i == 2):**

**# Di~e!**

**itera = 0;**

**conn1.params = {"u":u, "p":p, "python":1, "type":"out", "SID":SID};**

**Display.sysMsg('Attempting to log you out of the system...');**

**while loggedIn:**

**try:**

**if itera > 10:**

**Display.errorMsg("Logout attempt limit exceeded.");**

**loggedIn = False;**

**switch = False;**

**break;**

**else:**

**response = None;**

**response = conn1.request();**

**if response == "Goodbye":**

**loggedIn = False;**

**Display.playerMsg("Logged out successfully!");**

**switch = False;**

**break;**

**except BadStatusLine: pass;**

**except Exception:**

**Display.errorMsg("Error. The system was not able to log you out at this time.");**

**Display.errorMsg("Please try again next time.");**

**switch = False;**

**break;**

**itera = itera + 1;**

**# The user wishes to exit the program**

**elif(i == 3): MAR\_exit(0);**

**# The user wishes to modify their player's power tree**

**elif(i == 4): Display.gameMsg('This mechanic has not been enabled yet!');**

**# The user wishes to tinker with the game's internal settings**

**elif(i == 5): Display.gameMsg('This mechanic has not been enabled yet!');**

**elif response == "Denied":**

**Display.errorMsg("Invalid username/password combination.");**

**Display.errorMsg("Remember your username/password is case sensitive!");**

**elif response == "Malformed":**

**Display.errorMsg("You have illegal characters in your username/password.");**

**Display.errorMsg("(Usernames are only allowed letters and numbers!)");**

**Display.errorMsg("Contact Dark Gray for assistance or create a new account.");**

**else:**

**if(not response): Display.errorMsg("Connection to server was dropped. Please try again.");**

**else:**

**Display.errorMsg("Error. The response received from server was unrecognizable.");**

**Display.errorMsg(">>", response);**

**else: Display.errorMsg("Connection to server was dropped. Please try again.");**

**#We're attempting to register someone**

**elif(i == 1):**

**# Ask the user for his/her information**

**u = Display.cooked\_input("> Desired Username: ");**

**p = Display.getpass("> Desired Password:");**

**# Display all the available class choices**

**Display.gameMsg('Choose a class from those listed below (press enter to proceed):\n');**

**Display.indent();**

**Display.gameMsg(' -- Assassin -- ');**

**Display.gameMsg("As an Assassin you'll become the ultimate man-slayer, delivering deadly");**

**Display.gameMsg("blows to your enemy before melting back into the crowd... all in the");**

**Display.gameMsg("blink of an eye.");**

**Display.gameMsg("Nothing is True; Everything is permitted. A creed.\n");**

**Display.gameMsg(" -- Quote: \"That's thirty minutes away. I'll be there in ten.\"");**

**Display.gameMsg(" -- Specialty: Turn Slayer");**

**Display.pause();**

**Display.gameMsg(Display.newline+' -- Doctor -- ');**

**Display.gameMsg("As the Doctor you'll hold the lives of thousands in your hands. If you");**

**Display.gameMsg("decide to cure or crush those lives, however, is up to you.\n");**

**Display.gameMsg(" -- Quote: \"ha... Maha... Muaha... Muahaha!... BWHAHAHAHAHHAHAA!\"");**

**Display.gameMsg(" -- Specialty: Turn Gainer");**

**Display.pause();**

**Display.gameMsg(Display.newline+' -- Priest -- ');**

**Display.gameMsg("The path of the righteous man is beset on all sides by the iniquities");**

**Display.gameMsg("of the selfish and the tyranny of evil men. Blessed is he, who in the");**

**Display.gameMsg("name of charity and good will, shepherds the weak through the valley");**

**Display.gameMsg("of darkness, for he is truly his brother's keeper and the finder of ");**

**Display.gameMsg("lost children. And I will strike down upon thee with great vengeance");**

**Display.gameMsg("and furious anger those who would attempt to poison and destroy my");**

**Display.gameMsg("brothers. And you will know my name is the Lord when I lay my");**

**Display.gameMsg("vengeance upon thee.");**

**Display.gameMsg(" -- Quote: [Chasing guy down the street] \"You will pay for your sins!\"");**

**Display.gameMsg(" -- Specialty: Special Intervention");**

**Display.pause();**

**Display.gameMsg(Display.newline+' -- Courtesan -- ');**

**Display.gameMsg("Courtesan: [About to rob a diner] I love you, Nik.");**

**Display.gameMsg("Nik: I love you too, (one of your usual fake names).");**

**Display.gameMsg("Nik: [Stands with a knife] All right, everybody be cool, this is a");**

**Display.gameMsg("robbery!");**

**Display.gameMsg("Courtesan: [Stands with a gun] Any of you f%$king pr\*&ks move, and I'll");**

**Display.gameMsg("execute every m^$#erf!#king last one of ya!");**

**Display.gameMsg(" -- Specialty: Vertigo");**

**Display.gameMsg("NOTE: RECOMMENDED FOR EXPERIENCED PLAYERS ONLY!\n");**

**Display.pause();**

**Display.outdent();**

**user\_class = Display.menu(OrderedDict([('assassin', 0), ('doctor', 1), ('priest', 2), ('courtesan', 3)]), initMsg='\n Select Your Class:', prefix='-> ');**

**# Tell the user that we're doing things**

**Display.sysMsg("Conversing with server...");**

**# Package the new data**

**conn1.params = {"u":u, "python":1, "type":"", "SID":SID};**

**# Simple validation**

**if(len(u) > 25 or len(p) > 100 or len(u) < 4 or len(p) < 4):**

**Display.errorMsg("Your username/password must be between 4 and 25 characters in length.");**

**Display.errorMsg("Please try again.");**

**continue;**

**# Finish packaging the data**

**p = str(hashlib.sha1(hashlib.md5(p).hexdigest()).hexdigest());**

**conn1.params["p"] = p;**

**conn1.params["class"] = user\_class;**

**conn1.params["type"] = "reg";**

**# Send the data off for processing**

**response = conn1.request();**

**# Alert the user of the result, and display the proper options accordingly**

**if response != False:**

**if response == "Approved":**

**Display.playerMsg("The username", u, "has been registered successfully!");**

**Display.playerMsg("You may now log in with the password you provided.");**

**elif response == "Malformed":**

**Display.errorMsg("Error: The selected username is illegal.");**

**Display.errorMsg("Note that usernames should only contain letters & numbers.");**

**Display.errorMsg("Feel free to contact DG should you require assistance.");**

**elif response == "Denied":**

**Display.errorMsg("The username", u, "is not available for use.");**

**Display.errorMsg("Please select a different one.");**

**else:**

**if(not response): Display.errorMsg("Connection to server was dropped. Please try again.");**

**else:**

**Display.errorMsg("Error. The response received from server was unrecognizable.");**

**Display.errorMsg(">>", response);**

**else: Display.errorMsg("Connection to server was dropped. Please try again.");**

**# Go back a page**

**elif(i == 2): break;**

Online play mode. The lines are commented as necessary.

**# The user wishes to exit the program**

**elif(i == 3): MAR\_exit();**

Exit the program

**# Someone or something tried to interrupt the program!**

**except KeyboardInterrupt:**

**choice = Display.cooked\_input("Are you sure you wish to exit MAR? (y/n) ");**

**if(choice == 'y'): MAR\_exit();**

**else: pass;**

This code is used to handle any interrupts that may occur in the program.