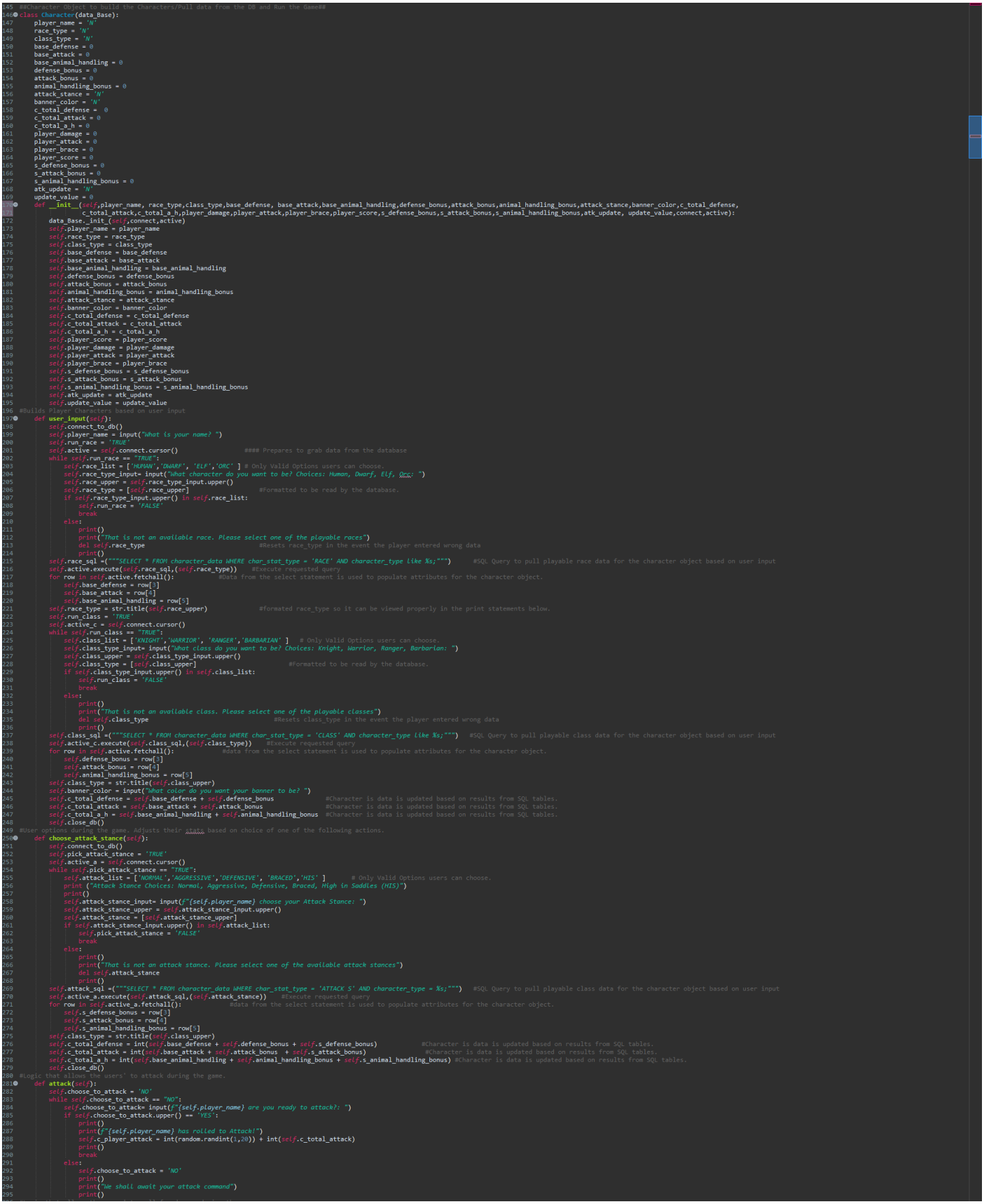
Database Object from Main Module (Main Module screen shot 1 of 5)
Selects/Updates/Inserts data from the joust_game database.

Character Object (Builds players' characters and actions for the joust) (Main Module screen shot 2 of 5)



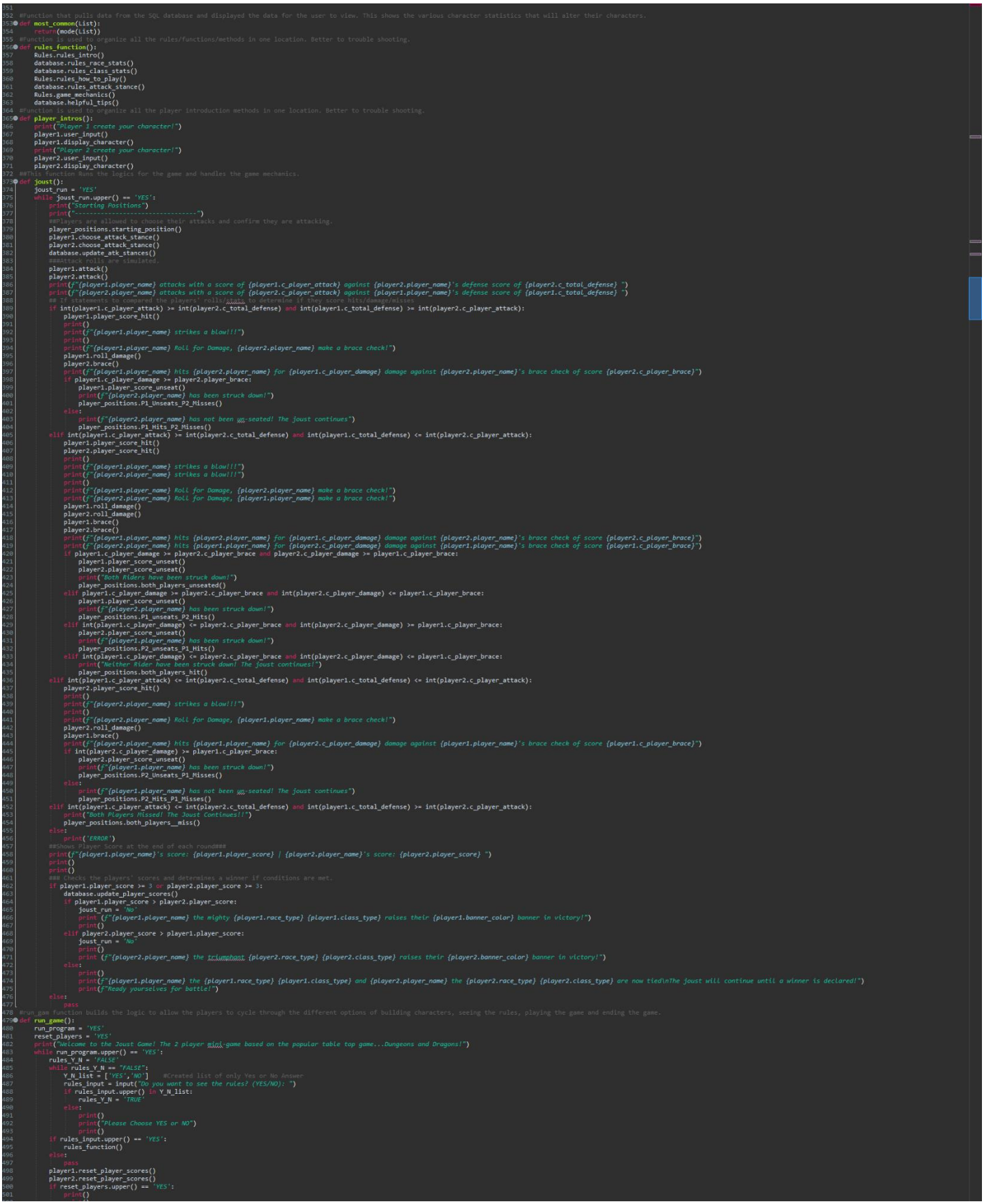
```
##Character Object to build the Characters/Pull data from the DB and Run the Game##
class Character(data_Base):
    player_name = 'N'
    race_type = 'N'
    class_type = 'N'
    base_defense = 0
    base_attack = 0
    base_animal_handling = 0
    defense_bonus = 0
    attack_bonus = 0
    animal_handling_bonus = 0
    attack_stance = 'N'
    banner_color = 'N'
    c_total_defense =  0
    c_total_attack = 0
    c_total_a_h = 0
    player_damage = 0
    player_attack = 0
    player_brace = 0
    player_score = 0
    s_defense_bonus = 0
    s_attack_bonus = 0
    s_animal_handling_bonus = 0
    atk_update = 'N'
    update_value = 0
    def __init__(self,player_name, race_type,class_type,base_defense, base_attack,base_animal_handling,defense_bonus,attack_bonus,animal_handling_bonus,attack_stance,banner_color,c_total_defense,
                 c_total_attack,c_total_a_h,player_damage,player_attack,player_brace,player_score,s_defense_bonus,s_attack_bonus,s_animal_handling_bonus,atk_update, update_value,connect,active):
        data_Base.__init__(self,connect,active)
        self.player_name = player_name
        self.race_type = race_type
        self.class_type = class_type
        self.base_defense = base_defense
        self.base_attack = base_attack
        self.base_animal_handling = base_animal_handling
        self.defense_bonus = defense_bonus
        self.attack_bonus = attack_bonus
        self.animal_handling_bonus = animal_handling_bonus
        self.attack_stance = attack_stance
        self.banner_color = banner_color
        self.c_total_defense = c_total_defense
        self.c_total_attack = c_total_attack
        self.c_total_a_h = c_total_a_h
        self.player_score = player_score
        self.player_damage = player_damage
        self.player_attack = player_attack
        self.player_brace = player_brace
        self.s_defense_bonus = s_defense_bonus
        self.s_attack_bonus = s_attack_bonus
        self.s_animal_handling_bonus = s_animal_handling_bonus
        self.atk_update = atk_update
        self.update_value = update_value
#Builds Player Characters based on user input
    def user_input(self):
        self.connect_to_db()
        self.player_name = input("What is your name? ")
        self.run_race = 'TRUE'
        self.active = self.connect.cursor()                 #### Prepares to grab data from the database
        while self.run_race == "TRUE":
            self.race_list = ['HUMAN','DWARF', 'ELF','ORC' ] # Only Valid Options users can choose.
            self.race_type_input= input("What character do you want to be? Choices: Human, Dwarf, Elf, Orc: ")
            self.race_upper = self.race_type_input.upper()
            self.race_type = [self.race_upper]                  #Formatted to be read by the database.
            if self.race_type_input.upper() in self.race_list:
                self.run_race = 'FALSE'
                break
            else:
                print()
                print("That is not an available race. Please select one of the playable races")
                del self.race_type                          #Resets race_type in the event the player entered wrong data
                print()
        self.race_sql =("""SELECT * FROM character_data WHERE char_stat_type = 'RACE' AND character_type like %s;""")      #SQL Query to pull playable race data for the character object based on user input
        self.active.execute(self.race_sql,(self.race_type))    #Execute requested query
        for row in self.active.fetchall():            #Data from the select statement is used to populate attributes for the character object.
            self.base_defense = row[3]
            self.base_attack = row[4]
            self.base_animal_handling = row[5]
        self.race_type = str.title(self.race_upper)             #formated race_type so it can be viewed properly in the print statements below.
        self.run_class = 'TRUE'
        self.active_c = self.connect.cursor()
        while self.run_class == "TRUE":
            self.class_list = ['KNIGHT','WARRIOR', 'RANGER','BARBARIAN' ]   # Only Valid Options users can choose.
            self.class_type_input= input("What class do you want to be? Choices: Knight, Warrior, Ranger, Barbarian: ")
            self.class_upper = self.class_type_input.upper()
            self.class_type = [self.class_upper]                    #Formatted to be read by the database.
            if self.class_type_input.upper() in self.class_list:
                self.run_class = 'FALSE'
                break
            else:
                print()
                print("That is not an available class. Please select one of the playable classes")
                del self.class_type                             #Resets class_type in the event the player entered wrong data
                print()
        self.class_sql =("""SELECT * FROM character_data WHERE char_stat_type = 'CLASS' AND character_type like %s;""")   #SQL Query to pull playable class data for the character object based on user input
        self.active_c.execute(self.class_sql,(self.class_type))    #Execute requested query
        for row in self.active.fetchall():            #Data from the select statement is used to populate attributes for the character object.
            self.defense_bonus = row[3]
            self.attack_bonus = row[4]
            self.animal_handling_bonus = row[5]
        self.class_type = str.title(self.class_upper)
        self.banner_color = input("What color do you want your banner to be? ")
        self.c_total_defense = self.base_defense + self.defense_bonus             #Character is data is updated based on results from SQL tables.
        self.c_total_attack = self.base_attack + self.attack_bonus                #Character is data is updated based on results from SQL tables.
        self.c_total_a_h = self.base_animal_handling + self.animal_handling_bonus  #Character is data is updated based on results from SQL tables.
        self.close_db()
#User options during the game. Adjusts their stats based on choice of one of the following actions.
    def choose_attack_stance(self):
        self.connect_to_db()
        self.pick_attack_stance = 'TRUE'
        self.active_a = self.connect.cursor()
        while self.pick_attack_stance == "TRUE":
            self.attack_list = ['NORMAL','AGGRESSIVE','DEFENSIVE', 'BRACED','HIS' ]       # Only Valid Options users can choose.
            print ("Attack Stance Choices: Normal, Aggressive, Defensive, Braced, High in Saddles (HIS)")
            print()
            self.attack_stance_input= input(f"{self.player_name} choose your Attack Stance: ")
            self.attack_stance_upper = self.attack_stance_input.upper()
            self.attack_stance = [self.attack_stance_upper]
            if self.attack_stance_input.upper() in self.attack_list:
                self.pick_attack_stance = 'FALSE'
                break
            else:
                print()
                print("That is not an attack stance. Please select one of the available attack stances")
                del self.attack_stance
                print()
        self.attack_sql =("""SELECT * FROM character_data WHERE char_stat_type = 'ATTACK S' AND character_type = %s;""")   #SQL Query to pull playable class data for the character object based on user input
        self.active_a.execute(self.attack_sql,(self.attack_stance))    #Execute requested query
        for row in self.active_a.fetchall():            #Data from the select statement is used to populate attributes for the character object.
            self.s_defense_bonus = row[3]
            self.s_attack_bonus = row[4]
            self.s_animal_handling_bonus = row[5]
        self.class_type = str.title(self.class_upper)
        self.c_total_defense = int(self.base_defense + self.defense_bonus + self.s_defense_bonus)           #Character is data is updated based on results from SQL tables.
        self.c_total_attack = int(self.base_attack + self.attack_bonus  + self.s_attack_bonus)              #Character is data is updated based on results from SQL tables.
        self.c_total_a_h = int(self.base_animal_handling + self.animal_handling_bonus + self.s_animal_handling_bonus) #Character is data is updated based on results from SQL tables.
        self.close_db()
#Logic that allows the users' to attack during the game.
    def attack(self):
        self.choose_to_attack = 'NO'
        while self.choose_to_attack == "NO":
            self.choose_to_attack= input(f"{self.player_name} are you ready to attack?: ")
            if self.choose_to_attack.upper() == 'YES':
                print()
                print(f"{self.player_name} has rolled to Attack!")
                self.c_player_attack = int(random.randint(1,20)) + int(self.c_total_attack)
                print()
                break
            else:
                self.choose_to_attack = 'NO'
                print()
                print("We shall await your attack command")
                print()
```

Character Object continued (Main Module screen shot 3 of 5)

```
                print()
#Logic that allows the users' to roll for damage during the game.
    def roll_damage(self):
        self.player_damage= 'NO'
        while self.player_damage == "NO":
            self.player_damage= input(f"{self.player_name} are you ready to roll for damage?: ")
            if self.player_damage.upper() == 'YES':
                print()
                print(f"{self.player_name} has rolled for damage!")
                self.c_player_damage = int(random.randint(1,12)) + 3      ### Builds a simulated damage roll based on random module.
                print()
                break
            else:
                self.player_damage = 'NO'
                print()
                print("Do not hesitate! un-seat for foe! ")
                print()
#Logic that allows the users' to roll for a brace check during the game.
    def brace(self):
        self.brace_check = 'NO'
        while self.brace_check == "NO":
            self.brace_check= input(f"{self.player_name} are you ready roll your brace check? ")
            if self.brace_check.upper() == 'YES':
                print()
                print(f"{self.player_name} has braced for incoming damage!")
                self.c_player_brace = int(random.randint(1,20)) + int(self.c_total_a_h) # Builds a simulated braced check based on random
                print()
                break
            else:
                self.brace_check = 'NO'
                print()
                print("You must defend yourself! Hurry!! ")
                print()
#Resets player scores for a new game.
    def reset_player_scores(self):
        self.player_score = 0
#The two calculations to determine how many points are scored.
    #Player scores 1 point for a hit against the opponent.
    def player_score_hit(self):
        self.player_score = int(self.player_score + 1)
    # Player scores 3 points for unseating opponent.
    def player_score_unseat(self):
        self.player_score = int(self.player_score + 3)
#Displays the statistics of the user's character after their input.
    def display_character(self):
        print(f"Welcome to the joust {self.player_name} the {self.race_type} {self.class_type} flying the {self.banner_color} banner! ")
        print("------------")
        print("Below are your character stats for the joust!")
        print("------------")
        print(f"Defense: {self.c_total_defense}")
        print(f"Attack Bonus: {self.c_total_attack}")
        print(f"Animal Handling: {self.c_total_a_h}")
        print("------------")
####PLAYER 1 and PLAYER 2 OBJECTS ########
player1 = Character('N','N','N',0,0,0,0,0,0,'N','N',0,0,0,0,0,0,0,'None','None',0,0,0,'N',0)
player2 = Character('N','N','N',0,0,0,0,0,0,'N','N',0,0,0,0,0,0,0,'None','None',0,0,0,'N',0)
```

Functions to compare actions of player 1 and player 2 objects along with logic to run the game

(Main Module screen shot 4 of 5)



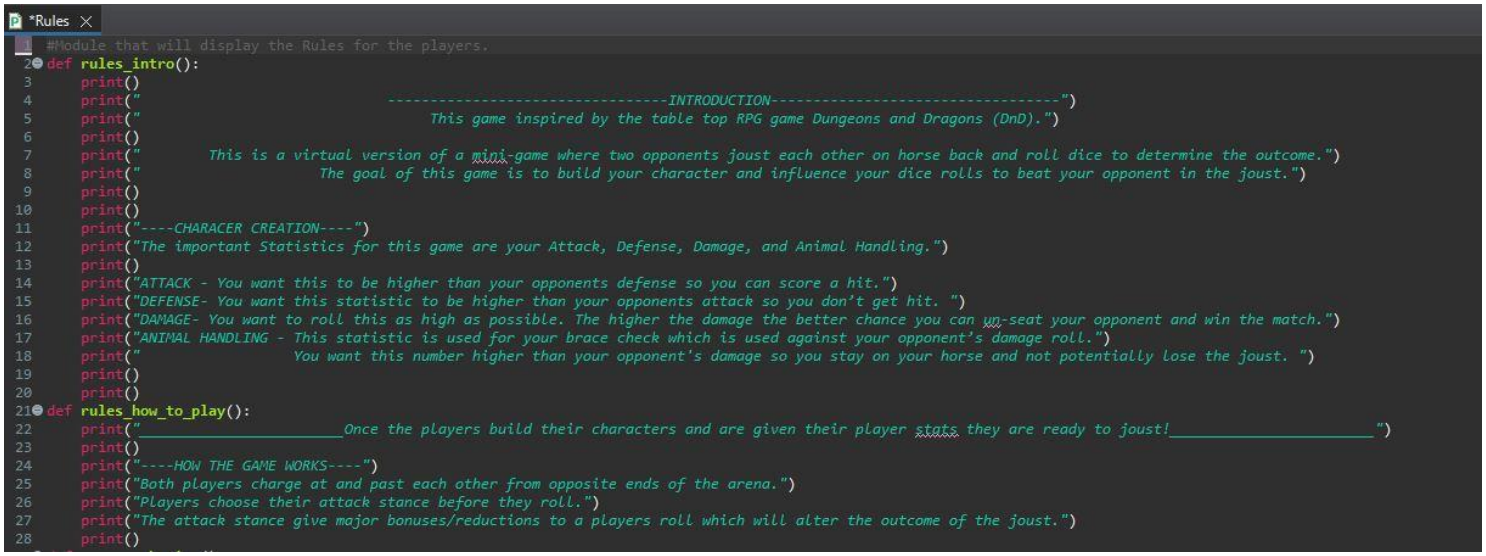
```
#Function that pulls data from the SQL database and displayed the data for the user to view. This shows the various character statistics that will alter their characters.
def most_common(List):
    return(mode(List))
#Function is used to organize all the rules/functions/methods in one location. Better to trouble shooting.
def rules_function():
    Rules.rules_intro()
    database.rules_race_stats()
    database.rules_class_stats()
    Rules.rules_how_to_play()
    database.rules_attack_stance()
    Rules.game_mechanics()
    database.helpful_tips()
#Function is used to organize all the player introduction methods in one location. Better to trouble shooting.
def player_intros():
    print("Player 1 create your character!")
    player1.user_input()
    player1.display_character()
    print("Player 2 create your character!")
    player2.user_input()
    player2.display_character()
##This function runs the logics for the game and handles the game mechanics.
def joust():
    joust_run = 'YES'
    while joust_run.upper() == 'YES':
        print("Starting Positions")
        print("-----------------------------------")
        ##Players are allowed to choose their attacks and confirm they are attacking.
        player_positions.starting_position()
        player1.choose_attack_stance()
        player2.choose_attack_stance()
        database.update_atk_stances()
        ##Actions of player are simulated.
        player1.attack()
        player2.attack()
        print(f"{player1.player_name} attacks with a score of {player1.c_player_attack} against {player2.player_name}'s defense score of {player2.c_total_defense} ")
        print(f"{player2.player_name} attacks with a score of {player2.c_player_attack} against {player1.player_name}'s defense score of {player1.c_total_defense} ")
        ## If statements to compare the player's rolls values to determine if they score hits/damage/misses
        if int(player1.c_player_attack) >= int(player2.c_total_defense) and int(player1.c_total_defense) >= int(player2.c_player_attack):
            player1.player_score_hit()
            print()
            print(f"{player1.player_name} strikes a blow!!!")
            print()
            print(f"{player1.player_name} Roll for Damage, {player2.player_name} make a brace check!")
            player1.roll_damage()
            player2.brace()
            print(f"{player1.player_name} hits {player2.player_name} for {player1.c_player_damage} damage against {player2.player_name}'s brace check of score {player2.c_player_brace}")
            if player1.c_player_damage >= player2.player_brace:
                player1.player_score_unseat()
                print(f"{player2.player_name} has been struck down!")
                player_positions.P1_Unseats_P2_Misses()
            else:
                print(f"{player2.player_name} has not been un-seated! The joust continues")
                player_positions.P1_Hits_P2_Misses()
        elif int(player1.c_player_attack) >= int(player2.c_total_defense) and int(player1.c_total_defense) <= int(player2.c_player_attack):
            player1.player_score_hit()
            player2.player_score_hit()
            print()
            print(f"{player1.player_name} strikes a blow!!!")
            print(f"{player2.player_name} strikes a blow!!!")
            print()
            print(f"{player1.player_name} Roll for Damage, {player2.player_name} make a brace check!")
            print(f"{player2.player_name} Roll for Damage, {player1.player_name} make a brace check!")
            player1.roll_damage()
            player2.roll_damage()
            player1.brace()
            player2.brace()
            print(f"{player1.player_name} hits {player2.player_name} for {player1.c_player_damage} damage against {player2.player_name}'s brace check of score {player2.c_player_brace}")
            print(f"{player2.player_name} hits {player1.player_name} for {player2.c_player_damage} damage against {player1.player_name}'s brace check of score {player1.c_player_brace}")
            if player1.c_player_damage >= player2.c_player_brace and player2.c_player_damage >= player1.c_player_brace:
                player1.player_score_unseat()
                player2.player_score_unseat()
                print("Both Riders have been struck down!")
                player_positions.both_players_unseated()
            elif player1.c_player_damage >= player2.c_player_brace and int(player2.c_player_damage) <= player1.c_player_brace:
                player1.player_score_unseat()
                print(f"{player2.player_name} has been struck down!")
                player_positions.P1_unseats_P2_Hits()
            elif int(player1.c_player_damage) <= player2.c_player_brace and int(player2.c_player_damage) >= player1.c_player_brace:
                player2.player_score_unseat()
                print(f"{player1.player_name} has been struck down!")
                player_positions.P2_unseats_P1_Hits()
            elif int(player1.c_player_damage) <= player2.c_player_brace and int(player2.c_player_damage) <= player1.c_player_brace:
                print("Neither Rider have been struck down! The joust continues!")
                player_positions.both_players_hit()
        elif int(player1.c_player_attack) <= int(player2.c_total_defense) and int(player1.c_total_defense) <= int(player2.c_player_attack):
            player2.player_score_hit()
            print()
            print(f"{player2.player_name} strikes a blow!!!")
            print()
            print(f"{player2.player_name} Roll for Damage, {player1.player_name} make a brace check!")
            player2.roll_damage()
            player1.brace()
            print(f"{player2.player_name} hits {player1.player_name} for {player2.c_player_damage} damage against {player1.player_name}'s brace check of score {player1.c_player_brace}")
            if int(player2.c_player_damage) >= player1.c_player_brace:
                player2.player_score_unseat()
                print(f"{player1.player_name} has been struck down!")
                player_positions.P2_Unseats_P1_Misses()
            else:
                print(f"{player1.player_name} has not been un-seated! The joust continues")
                player_positions.P2_Hits_P1_Misses()
        elif int(player1.c_player_attack) <= int(player2.c_total_defense) and int(player1.c_total_defense) >= int(player2.c_player_attack):
            print("Both Players Missed! The Joust Continues!!")
            player_positions.both_players__miss()
        else:
            print('ERROR')
        ##Shows Player Score at the end of each round###
        print(f"{player1.player_name}'s score: {player1.player_score} | {player2.player_name}'s score: {player2.player_score} ")
        print()
        print()
        ### Checks the players' scores and determines a winner if conditions are met.
        if player1.player_score >= 3 or player2.player_score >= 3:
            database.update_player_scores()
            if player1.player_score > player2.player_score:
                joust_run = 'No'
                print (f"{player1.player_name} the mighty {player1.race_type} {player1.class_type} raises their {player1.banner_color} banner in victory!")
                print()
            elif player2.player_score > player1.player_score:
                joust_run = 'No'
                print()
                print (f"{player2.player_name} the triumphant {player2.race_type} {player2.class_type} raises their {player2.banner_color} banner in victory!")
            else:
                print()
                print(f"{player1.player_name} the {player1.race_type} {player1.class_type} and {player2.player_name} the {player2.race_type} {player2.class_type} are now tied\nThe joust will continue until a winner is declared!")
                print(f"Ready yourselves for battle!")
        else:
            pass
#run_gam function builds the logic to allow the players to cycle through the different options of building characters, seeing the rules, playing the game and ending the game.
def run_game():
    run_program = 'YES'
    reset_players = 'YES'
    print("Welcome to the Joust Game! The 2 player mini-game based on the popular table top game...Dungeons and Dragons!")
    while run_program.upper() == 'YES':
        rules_Y_N = 'FALSE'
        while rules_Y_N == "FALSE":
            Y_N_list = ['YES','NO']   #Created list of only Yes or No Answer
            rules_input = input("Do you want to see the rules? (YES/NO): ")
            if rules_input.upper() in Y_N_list:
                rules_Y_N = 'TRUE'
            else:
                print()
                print("Please Choose YES or NO")
                print()
        if rules_input.upper() == 'YES':
            rules_function()
        else:
            pass
        player1.reset_player_scores()
        player2.reset_player_scores()
        if reset_players.upper() == 'YES':
            print()
```

Run Game Functions allows the players to view the rules, run the game, end the game, start a new game and/or reset their characters.

(Main Module screen shot 5 of 5)

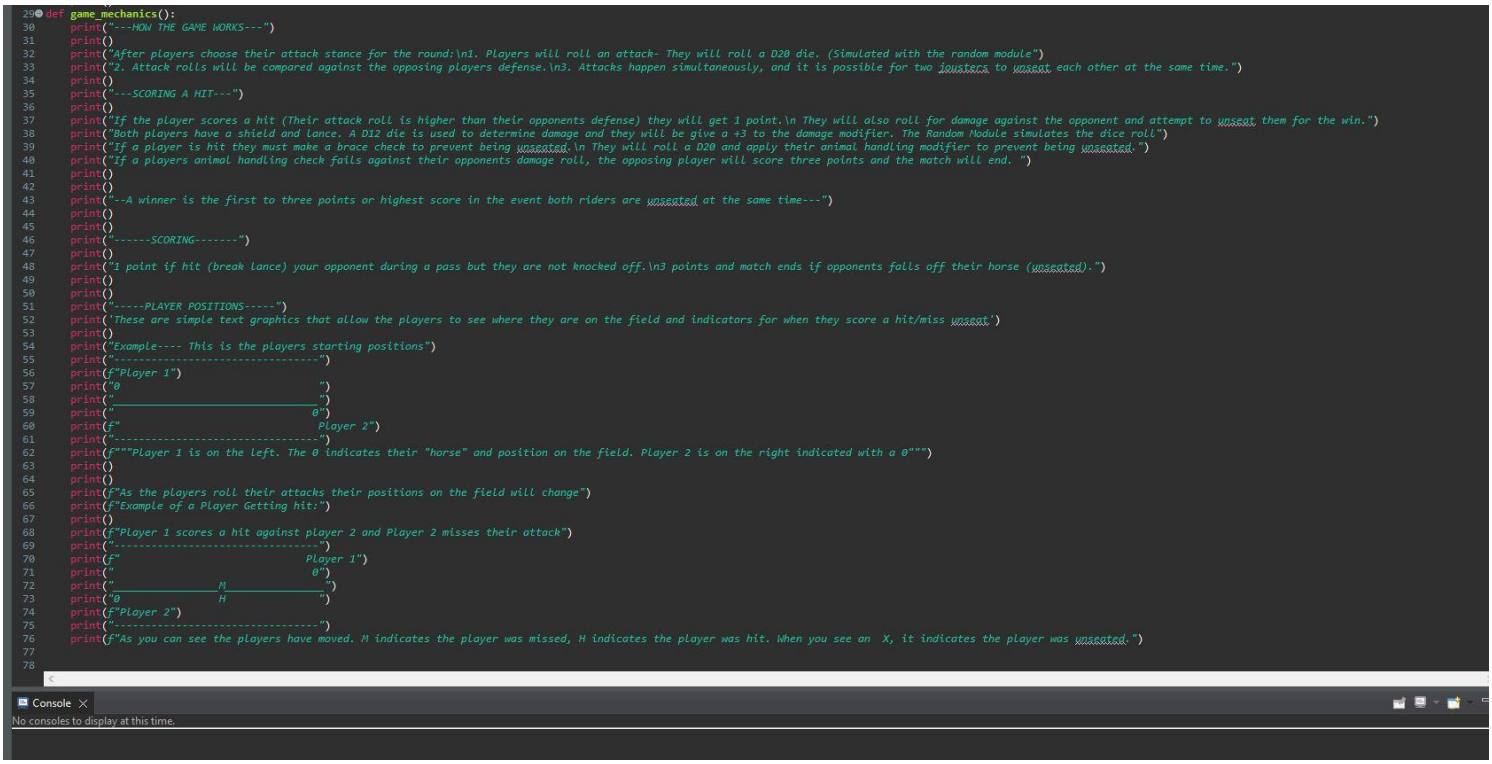
```
#run_gam function builds the logic to allow the players to cycle through the different options of building characters,
  # seeing the rules, playing the game and ending the game.
def run_game():
    run_program = 'YES'
    reset_players = 'YES'
    print("Welcome to the Joust Game! The 2 player mini-game based on the popular table top game...Dungeons and Dragons!")
    while run_program.upper() == 'YES':
        rules_Y_N = 'FALSE'
        while rules_Y_N == "FALSE":
            Y_N_list = ['YES','NO']    #Created list of only Yes or No Answer
            rules_input = input("Do you want to see the rules? (YES/NO): ")
            if rules_input.upper() in Y_N_list:
                rules_Y_N = 'TRUE'
            else:
                print()
                print("Please Choose YES or NO")
                print()
        if rules_input.upper() == 'YES':
            rules_function()
        else:
            pass
        player1.reset_player_scores()
        player2.reset_player_scores()
        if reset_players.upper() == 'YES':
            print()
            print()
            database.high_score()
            print()
            print()
            player_intros()
            database.insert_data()
            joust()
        else:
            database.high_score()
            player1.reset_player_scores()
            player2.reset_player_scores()
            database.insert_data()
            joust()
        run_program_Y_N = 'FALSE'
        while run_program_Y_N == "FALSE":
            run_program = input("Do you want to play again? (YES/NO): ")
            if run_program.upper() in Y_N_list:
                run_program_Y_N = 'TRUE'
            else:
                print()
                print("Please Choose YES or NO")
                print()
        if run_program.upper() == 'YES':
            reset_Y_N = 'FALSE'
            while reset_Y_N == 'FALSE':
                reset_players = input("Do you want to create new characters? (YES/NO): ")
                if reset_players.upper() in Y_N_list:
                    reset_Y_N = 'TRUE'
                else:
                    print()
                    print("Please Choose YES or NO")
                    print()
        else:
            print("Thank you for playing!")
            print("Program Ended")
            database.close_db()

```

Player Rules Module Screenshot 1 of 2



```python
#Module that will display the Rules for the players.
def rules_intro():
    print()
    print("                                ----------------------------------INTRODUCTION----------------------------------")
    print("                                     This game inspired by the table top RPG game Dungeons and Dragons (DnD).")
    print()
    print("        This is a virtual version of a mini-game where two opponents joust each other on horse back and roll dice to determine the outcome.")
    print("                        The goal of this game is to build your character and influence your dice rolls to beat your opponent in the joust.")
    print()
    print()
    print("----CHARACER CREATION----")
    print("The important Statistics for this game are your Attack, Defense, Damage, and Animal Handling.")
    print()
    print("ATTACK - You want this to be higher than your opponents defense so you can score a hit.")
    print("DEFENSE- You want this statistic to be higher than your opponents attack so you don't get hit. ")
    print("DAMAGE- You want to roll this as high as possible. The higher the damage the better chance you can un-seat your opponent and win the match.")
    print("ANIMAL HANDLING - This statistic is used for your brace check which is used against your opponent's damage roll.")
    print("                    You want this number higher than your opponent's damage so you stay on your horse and not potentially lose the joust. ")
    print()
    print()
def rules_how_to_play():
    print("_____________________________Once the players build their characters and are given their player stats they are ready to joust!_____________________________")
    print()
    print("----HOW THE GAME WORKS----")
    print("Both players charge at and past each other from opposite ends of the arena.")
    print("Players choose their attack stance before they roll.")
    print("The attack stance give major bonuses/reductions to a players roll which will alter the outcome of the joust.")
    print()
```

Player Rules Module Screenshot 1 of 2



```python
def game_mechanics():
    print("---HOW THE GAME WORKS---")
    print()
    print("After players choose their attack stance for the round:\n1. Players will roll an attack- They will roll a D20 die. (Simulated with the random module")
    print("2. Attack rolls will be compared against the opposing players defense.\n3. Attacks happen simultaneously, and it is possible for two jousters to unseat each other at the same time.")
    print()
    print("---SCORING A HIT---")
    print()
    print("If the player scores a hit (Their attack roll is higher than their opponents defense) they will get 1 point.\n They will also roll for damage against the opponent and attempt to unseat them for the win.")
    print("Both players have a shield and lance. A D12 die is used to determine damage and they will be give a +3 to the damage modifier. The Random Module simulates the dice roll")
    print("If a player is hit they must make a brace check to prevent being unseated.\n They will roll a D20 and apply their animal handling modifier to prevent being unseated.")
    print("If a players animal handling check fails against their opponents damage roll, the opposing player will score three points and the match will end. ")
    print()
    print()
    print("--A winner is the first to three points or highest score in the event both riders are unseated at the same time---")
    print()
    print()
    print("------SCORING-------")
    print()
    print("1 point if hit (break Lance) your opponent during a pass but they are not knocked off.\n3 points and match ends if opponents falls off their horse (unseated).")
    print()
    print()
    print("-----PLAYER POSITIONS-----")
    print('These are simple text graphics that allow the players to see where they are on the field and indicators for when they score a hit/miss unseat')
    print()
    print("Example---- This is the players starting positions")
    print("---------------------------------")
    print(f"Player 1")
    print("0                               ")
    print("_________________________________")
    print('                                0')
    print(f"                                Player 2")
    print("---------------------------------")
    print(f"""Player 1 is on the Left. The 0 indicates their "horse" and position on the field. Player 2 is on the right indicated with a 0""")
    print()
    print()
    print(f"As the players roll their attacks their positions on the field will change")
    print(f"Example of a Player Getting hit:")
    print()
    print(f"Player 1 scores a hit against player 2 and Player 2 misses their attack")
    print("---------------------------------")
    print(f"                              Player 1")
    print("                               0")
    print("_____________M_________________")
    print("0              H                ")
    print(f"Player 2")
    print("---------------------------------")
    print(f"As you can see the players have moved. M indicates the player was missed, H indicates the player was hit. When you see an  X, it indicates the player was unseated.")
```

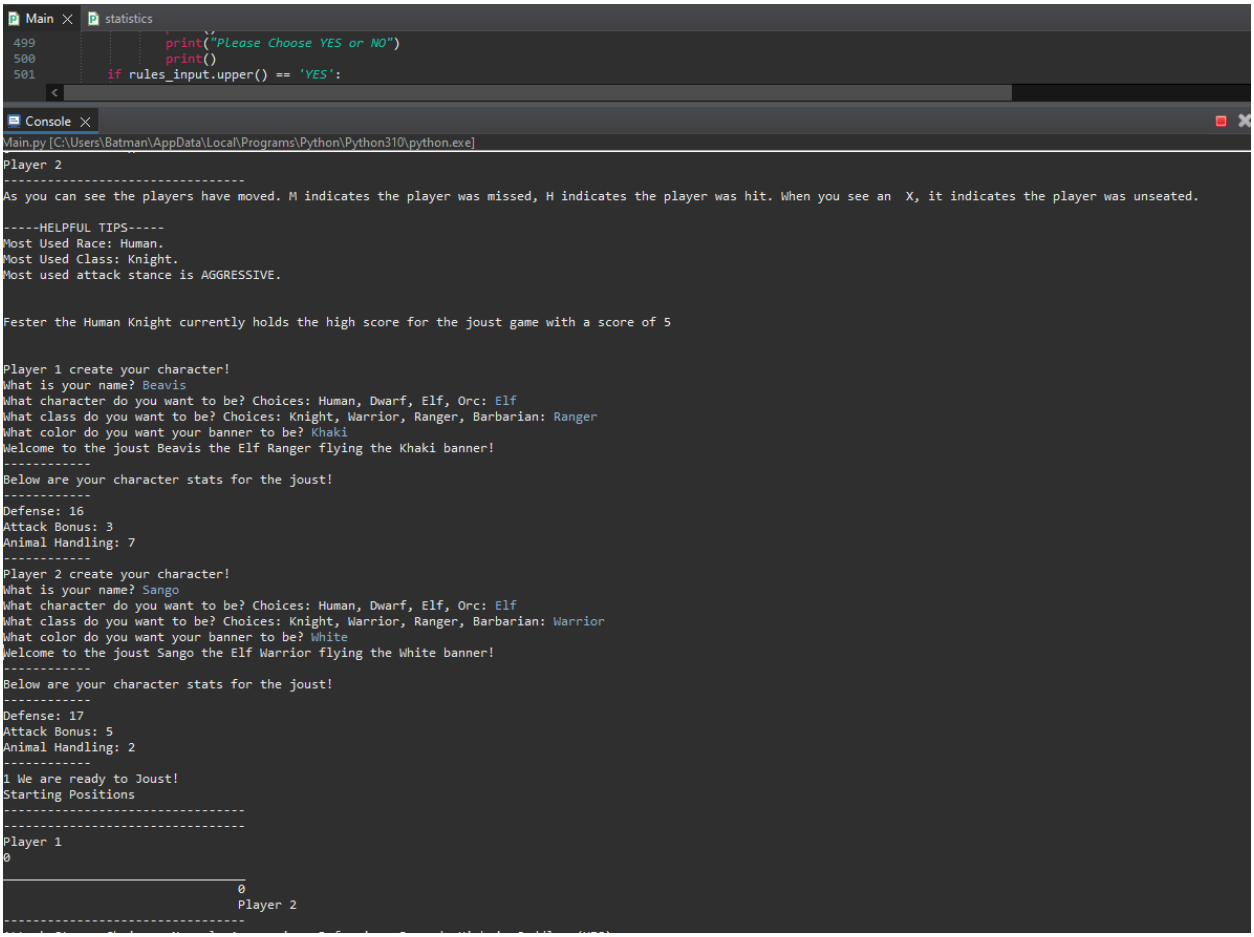
```
Console
No consoles to display at this time.
```

# Player Positions Module

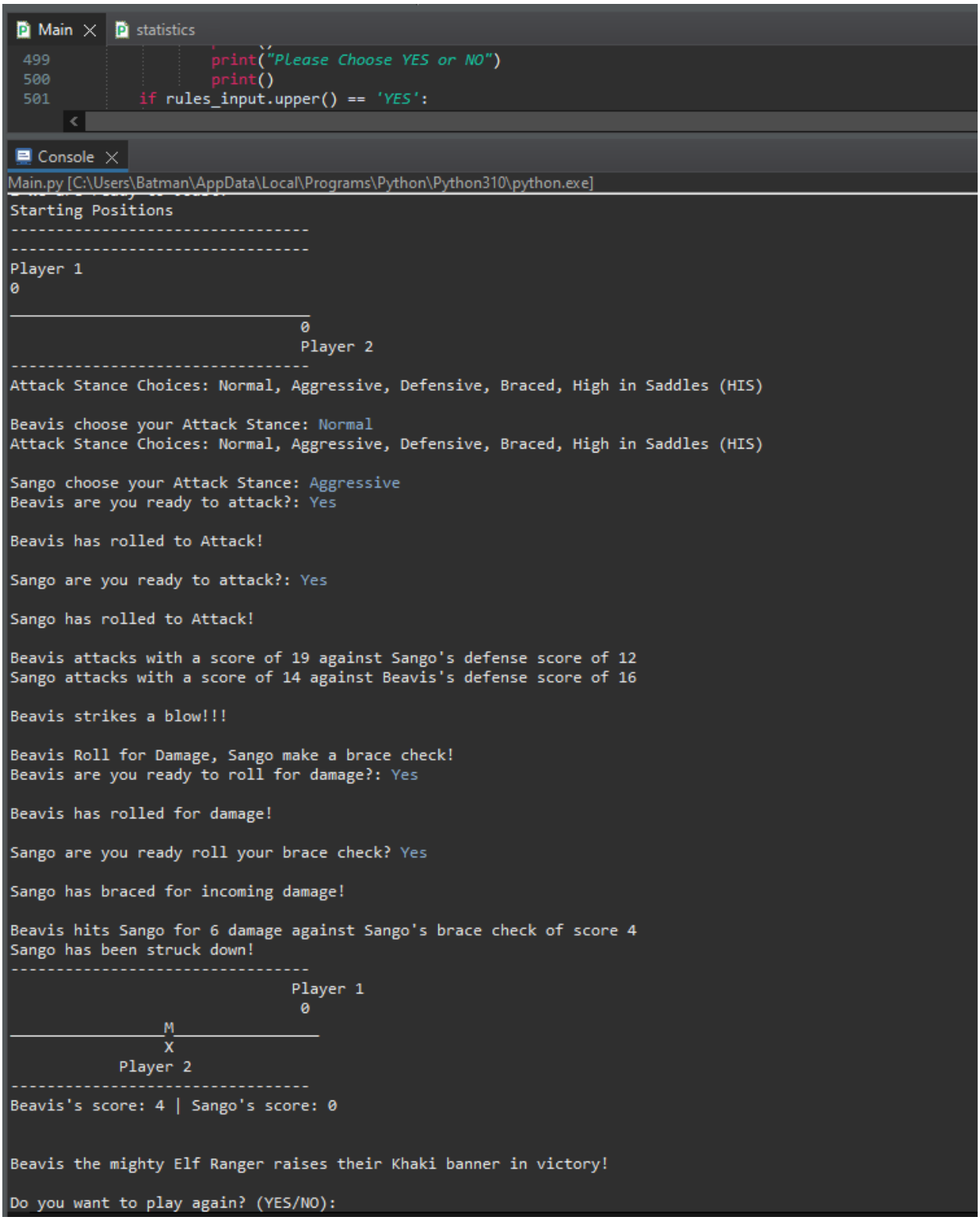
```python
##The modules is used to store the "graphics" (Player positions) that is displayed after the players attack each other.
#Starting Position is the default position for the players before they attack each other.
def starting_position():
    print("-----------------------------------")
    print(f"Player 1")
    print("0                                 ")
    print("_____________________________________")
    print("                                  0")
    print(f"                                  Player 2")
    print("-----------------------------------")
def P1_Hits_P2_Misses():
    print("-----------------------------------")
    print(f"                                Player 1")
    print("                                  0")
    print("_____________M_____________________")
    print("0            H                    ")
    print(f"Player 2")
    print("-----------------------------------")
def P2_Hits_P1_Misses():
    print("-----------------------------------")
    print(f"                                  Player 1")
    print("                                  0")
    print("_____________H_____________________")
    print("0            M                    ")
    print(f"Player 2")
    print("-----------------------------------")
# Both Hit 3
def both_players_hit():
    print("-----------------------------------")
    print(f"                                Player 1")
    print("                                  0")
    print("_____________H_____________________")
    print("0            H                    ")
    print(f"Player 2")
    print("-----------------------------------")
def both_players__miss():
    print("-----------------------------------")
    print(f"                                  Player 1")
    print("                                  0")
    print("_____________M_____________________")
    print("0            M                    ")
    print(f"Player 2")
    print("-----------------------------------")
def both_players_unseated():
    print("-----------------------------------")
    print(f"             Player 1")
    print("                                  ")
    print("_____________X_____________________")
    print("             X                    ")
    print(f"             Player 2")
    print("-----------------------------------")
def P1_unseats_P2_Hits():
    print("-----------------------------------")
    print(f"                                  Player 1")
    print("                                  0")
    print("_____________H_____________________")
    print("             X                    ")
    print(f"           Player 2")
    print("-----------------------------------")
def P2_unseats_P1_Hits():
    print("-----------------------------------")
    print(f"             Player 1")
    print("                                  ")
    print("_____________X_____________________")
    print("0            H                    ")
    print(f"Player 2")
    print("-----------------------------------")
def P1_Unseats_P2_Misses():
    print("-----------------------------------")
    print(f"                                Player 1")
    print("                                  0")
    print("_____________M_____________________")
    print("             X                    ")
    print(f"           Player 2")
    print("-----------------------------------")
def P2_Unseats_P1_Misses():
    print("-----------------------------------")
    print(f"             Player 1")
    print("                                 ")
    print("_____________X_____________________")
    print("0            M                    ")
    print(f"Player 2")
    print("-----------------------------------")


```

Program displaying the rules for the players. Uses SQL to display data from the character table in the joust game database.

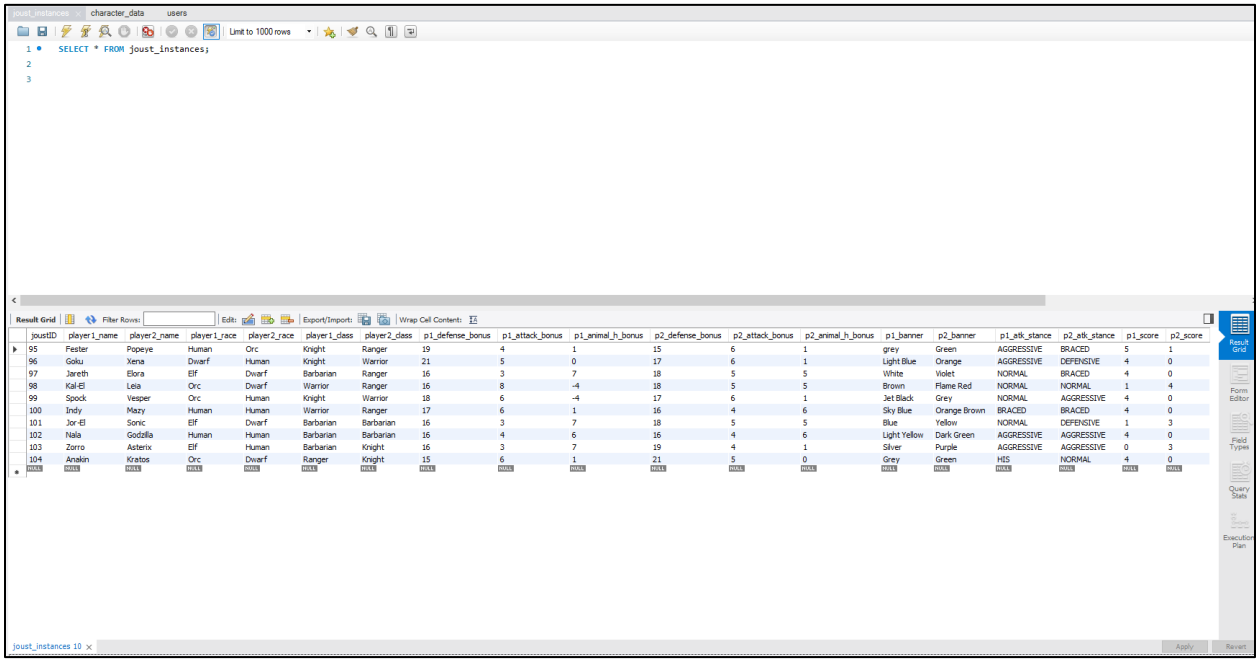
Programing building player characters based on user input.



Players taking turns deciding their attack stance and rolling their virtual dice. After a series of decisions. Player 1 (Beavis) is the winner and unseated their opponent.



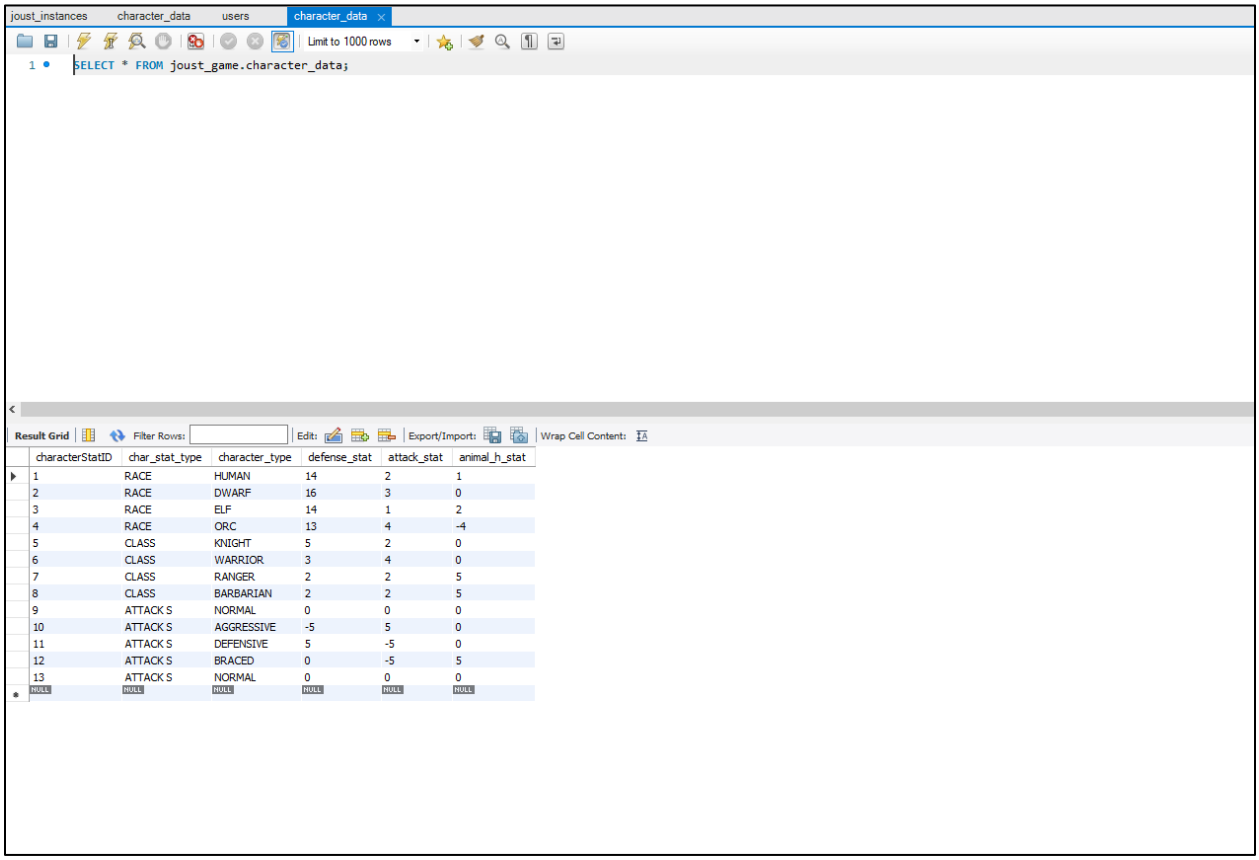
Database to hold each game instance (joust_instances). joustID 104 is the current row before insert.



```
SELECT * FROM joust_instances;
```

| joustID | player1_name | player2_name | player1_race | player2_race | player1_class | player2_class | p1_defense_bonus | p1_attack_bonus | p1_animal_h_bonus | p2_defense_bonus | p2_attack_bonus | p2_animal_h_bonus | p1_banner | p2_banner | p1_atk_stance | p2_atk_stance | p1_score | p2_score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 95 | Fester | Popeye | Human | Orc | Knight | Ranger | 19 | 4 | 1 | 15 | 6 | 1 | grey | Green | AGGRESSIVE | BRACED | 5 | 1 |
| 96 | Goku | Xena | Dwarf | Human | Knight | Warrior | 21 | 5 | 0 | 17 | 6 | 1 | Light Blue | Orange | AGGRESSIVE | DEFENSIVE | 4 | 0 |
| 97 | Jareth | Elora | Elf | Dwarf | Barbarian | Ranger | 16 | 3 | 7 | 18 | 5 | 5 | White | Violet | NORMAL | BRACED | 4 | 0 |
| 98 | Kal-El | Leia | Orc | Dwarf | Warrior | Ranger | 16 | 8 | -4 | 18 | 5 | 5 | Brown | Flame Red | NORMAL | NORMAL | 1 | 4 |
| 99 | Spock | Vesper | Orc | Human | Knight | Warrior | 18 | 6 | -4 | 17 | 6 | 1 | Jet Black | Grey | NORMAL | AGGRESSIVE | 4 | 0 |
| 100 | Indy | Mazy | Human | Human | Warrior | Ranger | 17 | 6 | 1 | 16 | 4 | 6 | Sky Blue | Orange Brown | BRACED | BRACED | 4 | 0 |
| 101 | Jor-El | Sonic | Elf | Dwarf | Barbarian | Barbarian | 16 | 3 | 7 | 18 | 5 | 5 | Blue | Yellow | NORMAL | DEFENSIVE | 1 | 3 |
| 102 | Nala | Godzilla | Human | Human | Barbarian | Barbarian | 16 | 4 | 6 | 16 | 4 | 6 | Light Yellow | Dark Green | AGGRESSIVE | AGGRESSIVE | 4 | 0 |
| 103 | Zorro | Asterix | Elf | Human | Barbarian | Knight | 16 | 3 | 7 | 19 | 4 | 1 | Silver | Purple | AGGRESSIVE | AGGRESSIVE | 0 | 3 |
| 104 | Anakin | Kratos | Orc | Dwarf | Ranger | Knight | 15 | 6 | 1 | 21 | 5 | 0 | Grey | Green | HIS | NORMAL | 4 | 0 |

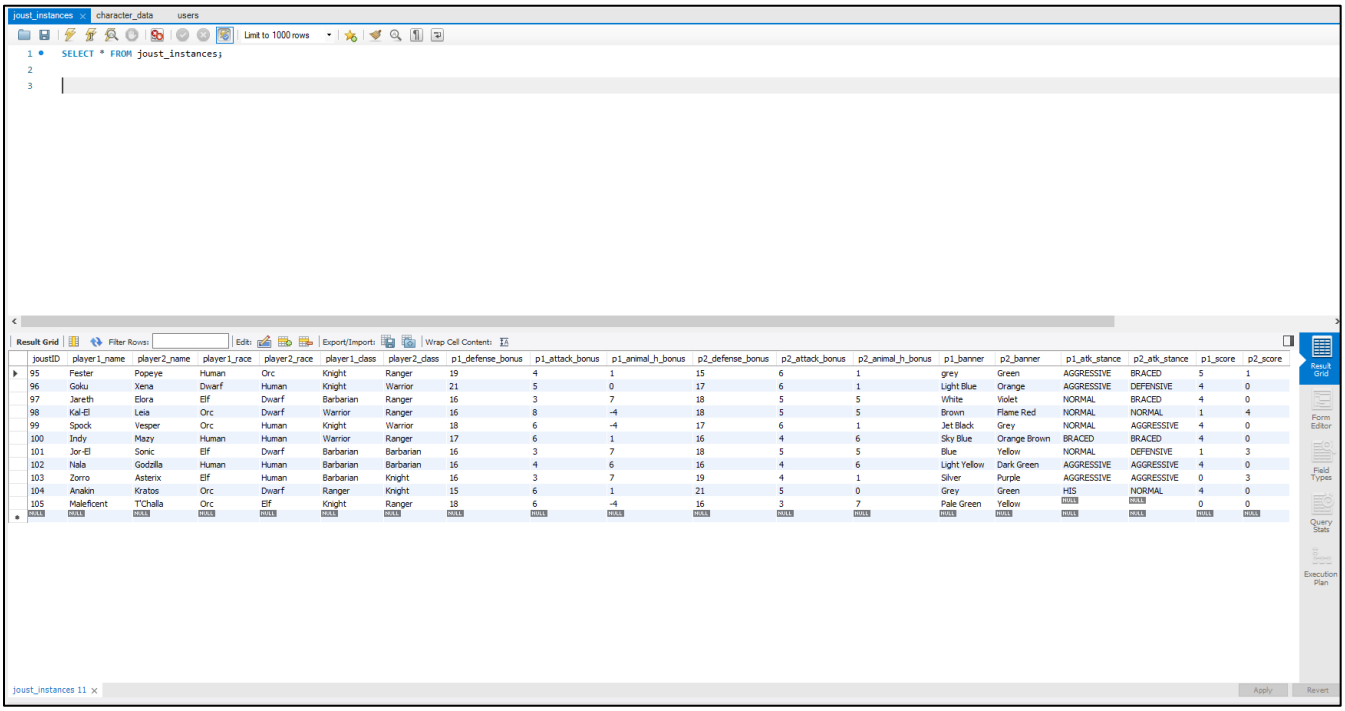
Character Data Table (Used to store data that helps the players build their characters).



```
SELECT * FROM joust_game.character_data;
```

| characterStatID | char_stat_type | character_type | defense_stat | attack_stat | animal_h_stat |
|---|---|---|---|---|---|
| 1 | RACE | HUMAN | 14 | 2 | 1 |
| 2 | RACE | DWARF | 16 | 3 | 0 |
| 3 | RACE | ELF | 14 | 1 | 2 |
| 4 | RACE | ORC | 13 | 4 | -4 |
| 5 | CLASS | KNIGHT | 5 | 2 | 0 |
| 6 | CLASS | WARRIOR | 3 | 4 | 0 |
| 7 | CLASS | RANGER | 2 | 2 | 5 |
| 8 | CLASS | BARBARIAN | 2 | 2 | 5 |
| 9 | ATTACK S | NORMAL | 0 | 0 | 0 |
| 10 | ATTACK S | AGGRESSIVE | -5 | 5 | 0 |
| 11 | ATTACK S | DEFENSIVE | 5 | -5 | 0 |
| 12 | ATTACK S | BRACED | 0 | -5 | 5 |
| 13 | ATTACK S | NORMAL | 0 | 0 | 0 |

Inserting Data into the Database (Row with joustID 105)




Updating data based on player choices and joust outcome.