

Selects/Updates/Inserts data from the joust_game database.

```

mysql.connector
# Statistics report mode
import random
# Import player positions
# Import Rules
# class Data_Base():
    connect = ''
    active = ''
    def __init__(self,connect,active):
        self.connect = connect
        self.active = active
    def connect_to_db(self):
        if self.connect == 'mysql':
            self.connect = mysql.connector.connect(
                user='root',
                password='root',
                host='127.0.0.1',
                database='joust_game')
    def close_db(self):
        self.connect.close()
# # Prints out saved data from the SQL database to display helpful information for the user when picking their characters.
def rules_race_stats(self):
    self.connect_to_db()
    print("""Method to run a SQL Query that pulls all playable race data for players to view""")
    print()
    print("""PLAYABLE RACES---""")
    print()
    self.active = self.connect.cursor()
    self_race_all_sql = """SELECT character_type, defense_stat, attack_stat, animal_h_stat FROM character_data WHERE char_stat_type = 'RACE';"""
    self.active.execute(self_race_all_sql)
    for row in self.active.fetchall():
        print(f"{row[0]} / Base Attacks: '{row[1]}' / Base Defense: '{row[2]}' / Base Animal Handling Stat: '{row[3]}'")
    def rules_class_stats(self):
        print()
        print("""PLAYABLE CLASSES---""")
        print()
        self.active = self.connect.cursor()
        self_class_all_sql = """SELECT character_type, defense_stat, attack_stat, animal_h_stat FROM character_data WHERE char_stat_type = 'CLASS';"""
        self.active.execute(self_class_all_sql)
        for row in self.active.fetchall():
            print(f"{row[0]} / Defense Bonus: '{row[1]}' / Attack Bonus: '{row[2]}' / Animal Handling Bonus: '{row[3]}'")
    def rules_attack_stance(self):
        print("""Method to run a SQL query that pulls all attack stance data for players to view""")
        print()
        print("""ATTACK STANCES---""")
        print()
        self.active = self.connect.cursor()
        self_attack_stance_sql = """SELECT character_type, defense_stat, attack_stat, animal_h_stat FROM character_data WHERE char_stat_type = 'ATTACK S';"""
        self.active.execute(self_attack_stance_sql)
        for row in self.active.fetchall():
            print(f"{row[0]} / Defense Bonus: '{row[1]}' / Attack Bonus: '{row[2]}' / Animal Handling Bonus: '{row[3]}'")
        print("""HIS + High In Saddle- An Aggressive form of riding which grants the player a high attack boost at the risk of being wounded easily.""")
        print("""BRAVED- A Defensive form of riding which grants the player a high animal handling boost for their braud check at the risk of having low attack and not scoring a point.""")
        self.close_db()
# # Prints out the high score from the SQL database to show the players the high score of the game. (The player who scored the most points in a Joust.)
def high_score(self):
    self.connect_to_db()
    self.hs_active = self.connect.cursor()
    self_high_score_sql = """SELECT player1_name, player1_race, player1_class, p1_score FROM joust_instances UNION ALL SELECT player2_name, player2_race, player2_class, p2_score FROM joust_instances ORDER BY p1_score DESC LIMIT 1;"""
    for row in self.hs_active.fetchall():
        self.hs_player_name = row[0]
        self.hs_player_race = row[1]
        self.hs_player_class = row[2]
        self_hs_player_score = row[3]
    try:
        print(f"(self.hs_player_name) the {self.hs_player_race} {self.hs_player_class} currently holds the high score for the joust game with a score of {self.hs_player_score}")
    except UnboundLocalError:
        print(f"There is not player with a high score! Be the first to claim the title!")
    self.close_db()
def helpfull_tips(self):
    self.connect_to_db()
    self_race_list = []
    self_atk_list = []
    self_active = self.connect.cursor()
    self_race_sql = """SELECT player1_race FROM joust_instances UNION ALL SELECT player2_race FROM joust_instances;"""
    self_active.execute(self_race_sql)
    self_race_list.append(self_race_list)
    self_atk_sql = """SELECT player1_class FROM joust_instances UNION ALL SELECT player2_class FROM joust_instances;"""
    self_active.execute(self_atk_sql)
    self_atk_list.append(self_atk_list)
    self_atk_active = self.connect.cursor()
    self_atk_row = self_atk_active.fetchall()
    self_atk_list.append(self_atk_row)
    self_most_common_race = most_common(self_race_list)
    self_most_common_class = most_common(self_atk_list)
    print(f"-----HELPFUL TIPS-----")
    print(f"Most Used Race: {self_most_common_race}, Most Used Class: {self_most_common_class}, Most Used attack stance is {self_most_common_atk}.")
    self.close_db()
# # Inserts the current joust instance and player data to the database.
# # The data stored in the SQL database is state for the game that is preserved when the program ends. (Preserving Game results)
def insert_data(self):
    self.connect_to_db()
    self_sql = """INSERT INTO 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) VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);"""
    self_values = (0,player1.player_name,player2.player_name,player1_race.type,player2_race.type,player1.class.type,player2.class.type,int(player1.c_total.defense),player1.c_total.attack,
int(player1.c_total.h),int(player2.c_total.defense),player2.c_total.attack,int(player2.c_total.h),player1.banner_color,player2.banner_color,player1.player_score,player2.player_score)
    self_active = self.connect.cursor()
    self_active.execute(self_sql,self_values)
    self.connect.commit();
    print(f"Active Rowcount,{self_active.rowcount},We are ready to Joust!")
    self.close_db()
# # Updates the joust_instances SQL table based on player 1 and 2 Input
def update_atk_stance(self):
    self.connect_to_db()
    self_p1_atk_stance_sql = """UPDATE joust_instances SET p1_atk_stance = %s ORDER BY joustID DESC LIMIT 1;"""
    self_p1_atk_active = self.connect.cursor()
    self_p1_atk_active.execute(self_p1_atk_stance_sql,player1.attack_stance)
    self_p2_atk_active = self.connect.cursor()
    self_p2_atk_active.execute(self_p2_atk_stance_sql,player2.attack_stance)
    self.connect.commit();
    self_p1_atk_active.execute(self_p1_atk_stance_sql,player2.attack_stance)
    self_connect.commit();
    self_close_db()
# # Updates the joust_instances SQL table based on player 1 and 2 Input
def update_player_scores(self):
    self.connect_to_db()
    self_p1_score_update_sql = """UPDATE joust_instances SET p1_score = %s ORDER BY joustID DESC LIMIT 1;"""
    self_p1_active = self.connect.cursor()
    self_p1_active.execute(self_p1_score_update_sql,player1.player_score)
    self_p2_active = self.connect.cursor()
    self_p2_active.execute(self_p2_score_update_sql,player2.player_score)
    self_connect.commit();
    self_close_db()
# Database object creation
database = data_base('','')
# This will find the Characters/Pull data from the DB and Run the Game
class Character(data_base):

```

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

```
143 #Character Object will hold the Characters/Pull data from the DB and run the games
144 class Character(data_Base):
145     player_name = 'N'
146     race_type = 'W'
147     class_type = 'N'
148     base_defense = 0
149     base_attack = 0
150     base_animal_handling = 0
151     defense_bonus = 0
152     attack_bonus = 0
153     animal_handling_bonus = 0
154     attack_stance = 'N'
155     banner_color = 'N'
156     c_total_defense = 0
157     c_total_attack = 0
158     c_total_a_h = 0
159     player_damage = 0
160     player_attack = 0
161     player_brace = 0
162     player_score = 0
163     s_defense_bonus = 0
164     s_attack_bonus = 0
165     s_animal_handling_bonus = 0
166     atk_update = 'N'
167     update_value = 0
168
169     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,
170                 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):
171         data_Base.__init__(self, connect, active)
172         self.player_name = player_name
173         self.race_type = race_type
174         self.class_type = class_type
175         self.base_defense = base_defense
176         self.base_attack = base_attack
177         self.base_animal_handling = base_animal_handling
178         self.defense_bonus = defense_bonus
179         self.attack_bonus = attack_bonus
180         self.animal_handling_bonus = animal_handling_bonus
181         self.attack_stance = attack_stance
182         self.banner_color = banner_color
183         self.c_total_defense = c_total_defense
184         self.c_total_attack = c_total_attack
185         self.c_total_a_h = c_total_a_h
186         self.player_score = player_score
187         self.player_damage = player_damage
188         self.player_attack = player_attack
189         self.player_brace = player_brace
190         self.s_defense_bonus = s_defense_bonus
191         self.s_attack_bonus = s_attack_bonus
192         self.s_animal_handling_bonus = s_animal_handling_bonus
193         self.atk_update = atk_update
194         self.update_value = update_value
195
196     #Builds player character based on user input
197     def user_input(self):
198         self.connect_to_db()
199         self.player_name = input("What is your name? ")
200         self.run_race = 'TRUE'
201         self.active = self.connect.cursor()
202
203         while self.run_race == 'TRUE':
204             self.race_list = ['HUMAN', 'DWARF', 'ELF', 'ORC'] # Only Valid Options users can choose.
205             self.race_type = input("What character do you want to be? Choices: Human, Dwarf, Elf, Orc: ")
206             self.race_upper = self.race_type.upper()
207             self.race_type = [self.race_upper] #Formatted to be read by the database.
208             if self.race_type[0] in self.race_list:
209                 self.run_race = 'FALSE'
210             else:
211                 print()
212                 print("That is not an available race. Please select one of the playable races")
213                 del self.race_type
214                 print()
215                 self.race_sql = ("SELECT * FROM character_data WHERE char_stat_type = 'RACE' AND character_type like %s;" % self.race_type) #SQL Query to pull playable race data for the character object based on user input
216                 self.active.execute(self.race_sql, (self.race_type)) #Execute requested query
217                 for row in self.active.fetchall(): #data from the select statement is used to populate attributes for the character object.
218                     self.base_defense = row[1]
219                     self.base_attack = row[2]
220                     self.base_animal_handling = row[3]
221                 self.race_type = str.title(self.race_upper) #formatted race_type so it can be viewed properly in the print statements below.
222                 self.run_race = 'TRUE'
223                 self.active_c = self.connect.cursor()
224                 while self.run_race == 'TRUE':
225                     self.class_list = ['KNIGHT', 'RANGER', 'WARRIOR', 'BARBARIAN'] # Only Valid Options users can choose.
226                     self.class_type = input("What class do you want to be? Choices: Knight, Warrior, Ranger, Barbarian: ")
227                     self.class_upper = self.class_type.upper()
228                     self.class_type = [self.class_upper] #Formatted to be read by the database.
229                     if self.class_type[0] in self.class_list:
230                         self.run_race = 'FALSE'
231                     else:
232                         print()
233                         print("That is not an available class. Please select one of the playable classes")
234                         del self.class_type
235                         print()
236                         self.class_sql = ("SELECT * FROM character_data WHERE char_stat_type = 'CLASS' AND character_type like %s;" % self.class_type) #SQL Query to pull playable class data for the character object based on user input
237                         self.active_c.execute(self.class_sql, (self.class_type)) #Execute requested query
238                         for row in self.active_c.fetchall(): #data from the select statement is used to populate attributes for the character object.
239                             self.defense_bonus = row[1]
240                             self.attack_bonus = row[2]
241                             self.animal_handling_bonus = row[3]
242                             self.class_type = str.title(self.class_upper)
243                             self.banner_color = input("What color do you want your banner to be? ")
244                             self.c_total_defense = self.base_defense + self.defense_bonus #Character is data is updated based on results from SQL tables.
245                             self.c_total_attack = self.base_attack + self.attack_bonus #Character is data is updated based on results from SQL tables.
246                             self.c_total_a_h = self.base_animal_handling + self.animal_handling_bonus #Character is data is updated based on results from SQL tables.
247                             self.close_db()
248
249     #User options during the game. Adjusts their stats based on choice of one of the following actions.
250     def choose_attack_stance(self):
251         self.connect_to_db()
252         self.pick_attack_stance = 'TRUE'
253         self.active_a = self.connect.cursor()
254         while self.pick_attack_stance == 'TRUE':
255             self.attack_list = ['NORMAL', 'AGGRESSIVE', 'DEFENSIVE', 'BRACED', 'HIS'] # Only Valid Options users can choose.
256             print("Attack Stance Choices: Normal, Aggressive, Defensive, Braced, High in Saddle (HIS)")
257             print()
258             self.attack_stance = input(f"{self.player_name} choose your Attack Stance: ")
259             self.attack_stance_upper = self.attack_stance.upper()
260             self.attack_stance = [self.attack_stance_upper]
261             if self.attack_stance[0] in self.attack_list:
262                 self.pick_attack_stance = 'FALSE'
263             else:
264                 print()
265                 print("That is not an attack stance. Please select one of the available attack stances")
266                 del self.attack_stance
267                 print()
268                 self.attack_sql = ("SELECT * FROM character_data WHERE char_stat_type = 'ATTACK S' AND character_type = %s;" % self.class_type) #SQL Query to pull playable class data for the character object based on user input
269                 self.active_a.execute(self.attack_sql, (self.attack_stance)) #Execute requested query
270                 for row in self.active_a.fetchall(): #data from the select statement is used to populate attributes for the character object.
271                     self.s_defense_bonus = row[1]
272                     self.s_attack_bonus = row[2]
273                     self.s_animal_handling_bonus = row[3]
274                     self.class_type = str.title(self.class_upper)
275                     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.
276                     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.
277                     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.
278                     self.close_db()
279
280     #Logic that allows users' to attack during the game.
281     def attack(self):
282         self.choose_to_attack = 'NO'
283         while self.choose_to_attack == 'NO':
284             self.choose_to_attack = input(f"{self.player_name} are you ready to attack?: ")
285             if self.choose_to_attack.upper() == 'YES':
286                 print()
287                 print(f"{self.player_name} has rolled to Attack!")
288                 self.c_player_attack = int(random.randint(1,20)) + int(self.c_total_attack)
289                 print()
290                 break
291             else:
292                 self.choose_to_attack = 'NO'
293                 print()
294                 print("We shall await your attack command")
295                 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 module.
            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)

```
351
352 #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.
353 def most_common(List):
354     return(mode(List))
355 #function is used to organize all the rules/functions/methods in one location. Better to trouble shooting.
356 def rules_function():
357     Rules.rules_intro()
358     database.rules_race_stats()
359     database.rules_class_stats()
360     Rules.rules_how_to_play()
361     database.rules_attack_stance()
362     Rules.game_mechanics()
363     database.helpful_tips()
364 #function is used to organize all the player introduction methods in one location. Better to trouble shooting.
365 def player_intros():
366     print("Player 1 create your character!")
367     player1.user_input()
368     player1.display_character()
369     print("Player 2 create your character!")
370     player2.user_input()
371     player2.display_character()
372 #this function have the logics for the game and handles the game mechanics.
373 def joust():
374     Joust_run = "YES"
375     while Joust_run.upper() == "YES":
376         print("Starting Positions")
377         print("-----")
378         #player 1 and player 2 choose their attacks and confirm they are attacking.
379         player_positions.starting_position()
380         player1.choose_attack_stance()
381         player2.choose_attack_stance()
382         database.update_atk_stances()
383         #set attack rolls to 0 simulated.
384         player1.attack()
385         player2.attack()
386         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}")
387         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}")
388         #if statements to compare the players' rolls/attacks to determine if they score hits/damage/misses
389         if int(player1.c_player_attack) >= int(player2.c_total_defense) and int(player1.c_total_defense) >= int(player2.c_player_attack):
390             player1.player_score_hit()
391             print()
392             print(f"{player1.player_name} strikes a blow!!!")
393             print()
394             print(f"{player1.player_name} Roll for Damage, {player2.player_name} make a brace check!")
395             player1.roll_damage()
396             player2.brace()
397             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}")
398             if player1.c_player_damage >= player2.player_brace:
399                 player1.player_score_unseat()
400                 print(f"{player2.player_name} has been struck down!")
401                 player_positions.P1_unseats_P2_Misses()
402             else:
403                 print(f"{player2.player_name} has not been unseated! The joust continues")
404                 player_positions.P1_hits_P2_Misses()
405             elif int(player1.c_player_attack) >= int(player2.c_total_defense) and int(player1.c_total_defense) <= int(player2.c_player_attack):
406                 player1.player_score_hit()
407                 player2.player_score_hit()
408                 print()
409                 print(f"{player1.player_name} strikes a blow!!!")
410                 print(f"{player2.player_name} strikes a blow!!!")
411                 print()
412                 print(f"{player1.player_name} Roll for Damage, {player2.player_name} make a brace check!")
413                 print(f"{player2.player_name} Roll for Damage, {player1.player_name} make a brace check!")
414                 player1.roll_damage()
415                 player2.roll_damage()
416                 player1.brace()
417                 player2.brace()
418                 print(f"{player1.player_name} hits {player2.player_name} for {player1.c_player_damage} damage against {player1.player_name}'s brace check of score {player2.c_player_brace}")
419                 print(f"{player1.player_name} hits {player2.player_name} for {player2.c_player_damage} damage against {player1.player_name}'s brace check of score {player1.c_player_brace}")
420                 if player1.c_player_damage >= player2.c_player_brace and player2.c_player_damage >= player1.c_player_brace:
421                     player1.player_score_unseat()
422                     player2.player_score_unseat()
423                     print("Both Riders Have been struck down!")
424                     player_positions.both_players_unseated()
425                 elif player1.c_player_damage >= player2.c_player_brace and int(player2.c_player_damage) <= player1.c_player_brace:
426                     player1.player_score_unseat()
427                     print(f"{player2.player_name} has been struck down!")
428                     player_positions.P1_unseats_P2_Hits()
429                 elif int(player1.c_player_damage) <= player2.c_player_brace and int(player2.c_player_damage) >= player1.c_player_brace:
430                     player2.player_score_unseat()
431                     print(f"{player1.player_name} has been struck down!")
432                     player_positions.P2_unseats_P1_Hits()
433                 elif int(player1.c_player_damage) <= player2.c_player_brace and int(player2.c_player_damage) <= player1.c_player_brace:
434                     print("Neither Rider have been struck down! The joust continues!")
435                     player_positions.both_players_hit()
436                 elif int(player1.c_player_attack) <= int(player2.c_total_defense) and int(player1.c_total_defense) <= int(player2.c_player_attack):
437                     player2.player_score_hit()
438                     print()
439                     print(f"{player2.player_name} strikes a blow!!!")
440                     print()
441                     print(f"{player2.player_name} Roll for Damage, {player1.player_name} make a brace check!")
442                     player2.roll_damage()
443                     player1.brace()
444                     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}")
445                     if int(player2.c_player_damage) >= player1.c_player_brace:
446                         player2.player_score_unseat()
447                         print(f"{player1.player_name} has been struck down!")
448                         player_positions.P2_unseats_P1_Misses()
449                     else:
450                         print(f"{player1.player_name} has not been unseated! The joust continues")
451                         player_positions.P2_hits_P1_Misses()
452                 elif int(player1.c_player_attack) <= int(player2.c_total_defense) and int(player1.c_total_defense) >= int(player2.c_player_attack):
453                     print("Both Players Missed! The Joust Continues!")
454                     player_positions.both_players_miss()
455                 else:
456                     print("ERROR")
457             ##Shows Player Score at the end of each round##
458             print(f"{player1.player_name}'s score: {player1.player_score} | {player2.player_name}'s score: {player2.player_score}")
459             print()
460             print()
461             ## Checks the players' scores and determines a winner if conditions are met.
462             if player1.player_score >= 3 or player2.player_score >= 3:
463                 database.update_player_scores()
464                 if player1.player_score > player2.player_score:
465                     Joust_run = "No"
466                     print(f"{player1.player_name} the mighty {player1.race_type} {player1.class_type} raises their {player1.banner_color} banner in victory!")
467                     print()
468                 elif player2.player_score > player1.player_score:
469                     Joust_run = "No"
470                     print(f"{player2.player_name} the {player2.race_type} {player2.class_type} raises their {player2.banner_color} banner in victory!")
471                     print()
472                 else:
473                     print()
474                     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! The joust will continue until a winner is declared!")
475                     print(f"Ready yourselves for battle!")
476             else:
477                 pass
478 #run game 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.
479 def run_game():
480     run_program = "YES"
481     reset_players = "YES"
482     print("Welcome to the Joust Game! The 2 player joust game based on the popular table top game...Dungeons and Dragons!")
483     while run_program.upper() == "YES":
484         rules_Y_N = "FALSE"
485         while rules_Y_N == "FALSE":
486             rules_Y_N_list = ["YES", "NO"] #created list of only Yes or No Answer
487             rules_input = input("Do you want to see the rules? (YES/NO): ")
488             if rules_input.upper() in rules_Y_N_list:
489                 rules_Y_N = "TRUE"
490             else:
491                 print()
492                 print("Please Choose YES or NO")
493                 print()
494             if rules_input.upper() == "YES":
495                 rules_function()
496             else:
497                 pass
498             player1.reset_player_scores()
499             player2.reset_player_scores()
500             if reset_players.upper() == "YES":
501                 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)

```
478 #run_gam function builds the logic to allow the players to cycle through the different options of building characters,  
479 # seeing the rules, playing the game and ending the game.  
480 def run_game():  
481     run_program = 'YES'  
482     reset_players = 'YES'  
483     print("Welcome to the Joust Game! The 2 player mini-game based on the popular table top game...Dungeons and Dragons!")  
484     while run_program.upper() == 'YES':  
485         rules_Y_N = 'FALSE'  
486         while rules_Y_N == "FALSE":  
487             Y_N_list = ['YES', 'NO'] #Created list of only Yes or No Answer  
488             rules_input = input("Do you want to see the rules? (YES/NO): ")  
489             if rules_input.upper() in Y_N_list:  
490                 rules_Y_N = 'TRUE'  
491             else:  
492                 print()  
493                 print("Please Choose YES or NO")  
494                 print()  
495         if rules_input.upper() == 'YES':  
496             rules_function()  
497         else:  
498             pass  
499         player1.reset_player_scores()  
500         player2.reset_player_scores()  
501         if reset_players.upper() == 'YES':  
502             print()  
503             print()  
504             database.high_score()  
505             print()  
506             print()  
507             player_intros()  
508             database.insert_data()  
509             joust()  
510         else:  
511             database.high_score()  
512             player1.reset_player_scores()  
513             player2.reset_player_scores()  
514             database.insert_data()  
515             joust()  
516         run_program_Y_N = 'FALSE'  
517         while run_program_Y_N == "FALSE":  
518             run_program = input("Do you want to play again? (YES/NO): ")  
519             if run_program.upper() in Y_N_list:  
520                 run_program_Y_N = 'TRUE'  
521             else:  
522                 print()  
523                 print("Please Choose YES or NO")  
524                 print()  
525         if run_program.upper() == 'YES':  
526             reset_Y_N = 'FALSE'  
527             while reset_Y_N == 'FALSE':  
528                 reset_players = input("Do you want to create new characters? (YES/NO): ")  
529                 if reset_players.upper() in Y_N_list:  
530                     reset_Y_N = 'TRUE'  
531                 else:  
532                     print()  
533                     print("Please Choose YES or NO")  
534                     print()  
535         else:  
536             print("Thank you for playing!")  
537             print("Program Ended")  
538             database.close_db()  
539
```

Player Rules Module Screenshot 1 of 2

```
1 #Module that will display the Rules for the players.
2 def rules_intro():
3     print()
4     print("-----INTRODUCTION-----")
5     print("This game inspired by the table top RPG game Dungeons and Dragons (DnD).")
6     print()
7     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.")
8     print("The goal of this game is to build your character and influence your dice rolls to beat your opponent in the joust.")
9     print()
10    print()
11    print("----CHARACER CREATION----")
12    print("The important Statistics for this game are your Attack, Defense, Damage, and Animal Handling.")
13    print()
14    print("ATTACK - You want this to be higher than your opponents defense so you can score a hit.")
15    print("DEFENSE- You want this statistic to be higher than your opponents attack so you don't get hit. ")
16    print("DAMAGE- You want to roll this as high as possible. The higher the damage the better chance you can win-seat your opponent and win the match.")
17    print("ANIMAL HANDLING - This statistic is used for your brace check which is used against your opponent's damage roll.")
18    print("You want this number higher than your opponent's damage so you stay on your horse and not potentially lose the joust. ")
19    print()
20    print()
21 def rules_how_to_play():
22     print("-----HOW THE GAME WORKS-----")
23     print()
24     print("Both players charge at and past each other from opposite ends of the arena.")
25     print("Players choose their attack stance before they roll.")
26     print("The attack stance give major bonuses/reductions to a players roll which will alter the outcome of the joust.")
27     print()
```

Player Rules Module Screenshot 1 of 2

```
29 def game_mechanics():
30     print("----HOW THE GAME WORKS----")
31     print()
32     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)")
33     print("2. Attack rolls will be compared against the opposing players defense.\n3. Attacks happen simultaneously, and it is possible for two jousters to win-seat each other at the same time.")
34     print()
35     print("----SCORING A HIT----")
36     print()
37     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 win-seat them for the win.")
38     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.")
39     print("If a player is hit they must make a brace check to prevent being win-seated.\n They will roll a D20 and apply their animal handling modifier to prevent being win-seated.")
40     print("If a players animal handling check falls against their opponents damage roll, the opposing player will score three points and the match will end. ")
41     print()
42     print()
43     print("----A winner is the first to three points or highest score in the event both riders are win-seated at the same time----")
44     print()
45     print()
46     print("-----SCORING-----")
47     print()
48     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 (win-seated).")
49     print()
50     print()
51     print("-----PLAYER POSITIONS-----")
52     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 win-seat.")
53     print()
54     print("Example---- This is the players starting positions")
55     print("-----")
56     print(f"Player 1")
57     print(" ")
58     print(" ")
59     print(" ")
60     print(f"Player 2")
61     print(" ")
62     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")
63     print()
64     print()
65     print(f"As the players roll their attacks their positions on the field will change")
66     print(f"Example of a Player Getting Hit:")
67     print()
68     print(f"Player 1 scores a hit against player 2 and Player 2 misses their attack")
69     print("-----")
70     print(f"Player 1")
71     print(" ")
72     print(" ")
73     print(" ")
74     print(f"Player 2")
75     print("-----")
76     print(f"As you can see the players have moved. H indicates the player was missed, H indicates the player was hit. When you see an X, it indicates the player was win-seated.")
77     print()
78     print()
```

Player Positions Module

```
10 ##The module is used to store the "graphics" (Player positions) that is displayed after the players attack each other.  
11 # Starting Position is the default position for the players before they attack each other.
```

```
12 def starting_position():  
13     print("-----")  
14     print(f"Player 1")  
15     print("0")  
16     print("0")  
17     print(f"Player 2")  
18     print("-----")  
19 def P1_Hits_P2_Misses():  
20     print("-----")  
21     print(f"Player 1")  
22     print("0")  
23     print("H")  
24     print(f"Player 2")  
25     print("-----")  
26 def P2_Hits_P1_Misses():  
27     print("-----")  
28     print(f"Player 1")  
29     print("0")  
30     print("H")  
31     print(f"Player 2")  
32     print("-----")  
33 def both_players_hit():  
34     print("-----")  
35     print(f"Player 1")  
36     print("0")  
37     print("H")  
38     print(f"Player 2")  
39     print("-----")  
40 def both_players_miss():  
41     print("-----")  
42     print(f"Player 1")  
43     print("0")  
44     print("H")  
45     print(f"Player 2")  
46     print("-----")  
47 def both_players_unseated():  
48     print("-----")  
49     print(f"Player 1")  
50     print("X")  
51     print(f"Player 2")  
52     print("-----")  
53 def P1_unseats_P2_Hits():  
54     print("-----")  
55     print(f"Player 1")  
56     print("0")  
57     print("X")  
58     print(f"Player 2")  
59     print("-----")  
60 def P2_unseats_P1_Hits():  
61     print("-----")  
62     print(f"Player 1")  
63     print("X")  
64     print(f"Player 2")  
65     print("-----")  
66 def P1_unseats_P2_Misses():  
67     print("-----")  
68     print(f"Player 1")  
69     print("0")  
70     print("X")  
71     print(f"Player 2")  
72     print("-----")  
73 def P2_unseats_P1_Misses():  
74     print("-----")  
75     print(f"Player 1")  
76     print("X")  
77     print(f"Player 2")  
78     print("-----")
```


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

```

[Console] X
Mainpy[Cs\Users\Eatman\AppData\Local\Programs\Python\Python310\python.exe]
Welcome to the Joust Game! The mini-game based on the popular table top game...Dungeons and Dragons!
Do you want to see the rules? (YES/NO): Yes

-----INTRODUCTION-----
This game inspired by the table top RPG game Dungeons and Dragons (DnD).

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.
The goal of this game is to build your character and influence your dice rolls to beat your opponent in the joust.

----CHARACTER CREATION----
The Important Statistics for this game are your Attack, Defense, Damage, and Animal Handling.

ATTACK - You want this to be higher than your opponents defense so you can score a hit.
DEFENSE- You want this statistic to be higher than your opponents attack so you don't get hit.
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.
ANIMAL HANDLING - This statistic is used for your brace check which is used against your opponent's damage roll.
You want this number higher than your opponent's damage so you stay on your horse and not potentially lose the joust.

Players choose their playable race and class.
These choices give you bonuses or reductions that will affect your dice rolls.

---PLAYABLE RACES---
Race Type: HUMAN | Base Defense: 14 | Base Attack: 2 | Base Animal Handling Stat: 1
Race Type: DWARF | Base Defense: 16 | Base Attack: 3 | Base Animal Handling Stat: 0
Race Type: ELF | Base Defense: 14 | Base Attack: 1 | Base Animal Handling Stat: 2
Race Type: ORC | Base Defense: 13 | Base Attack: 4 | Base Animal Handling Stat: -4

---PLAYABLE CLASSES---
Class Type: KNIGHT | Defense Bonus: 5 | Attack Bonus: 2 | Animal Handling Bonus: 0
Class Type: WARRIOR | Defense Bonus: 3 | Attack Bonus: 4 | Animal Handling Bonus: 0
Class Type: RANGER | Defense Bonus: 2 | Attack Bonus: 2 | Animal Handling Bonus: 5
Class Type: BARBARIAN | Defense Bonus: 2 | Attack Bonus: 2 | Animal Handling Bonus: 5
Once the players build their characters and are given their player stats they are ready to joust!

----HOW THE GAME WORKS----
Both players charge at and past each other from opposite ends of the arena.
Players choose their attack stance before they roll.
The attack stance give major bonuses/reductions to a players roll which will alter the outcome of the joust.

The Various Attack Stances and their statistics are:

---ATTACK STANCES---
Attack Stance: NORMAL | Defense Bonus: 0 | Attack Bonus: 0 | Animal Handling Bonus: 0
Attack Stance: AGGRESSIVE | Defense Bonus: -5 | Attack Bonus: 5 | Animal Handling Bonus: 0
Attack Stance: DEFENSIVE | Defense Bonus: 5 | Attack Bonus: -5 | Animal Handling Bonus: 0
Attack Stance: BRACED | Defense Bonus: 0 | Attack Bonus: -5 | Animal Handling Bonus: 5
Attack Stance: NORMAL | Defense Bonus: 0 | Attack Bonus: 0 | Animal Handling Bonus: 0

Note * HIS = High in Saddle- An Aggressive form of riding which grants the player a high attack boost at the risk of being unseated easily.
Braced- A Defensive form of riding which grants the player a high animal handling boost for their braced check at the risk of having low attack and not scoring a point.

---HOW THE GAME WORKS---
After players choose their attack stance for the round:
1. Players will roll an attack- They will roll a D20 die. (Simulated with the random module
2. Attack rolls will be compared against the opposing players defense.
3. Attacks happen simultaneously, and it is possible for two jousters to unseat each other at the same time.

---SCORING A HIT---
If the player scores a hit (Their attack roll is higher than their opponents defense) they will get 1 point.
They will also roll for damage against the opponent and attempt to unseat them for the win.
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
If a player is hit they must make a brace check to prevent being unseated.
They will roll a D20 and apply their animal handling modifier to prevent being unseated.
If a players animal handling check fails against their opponents damage roll, the opposing player will score three points and the match will end.

--A winner is the first to three points or highest score in the event both riders are unseated at the same time--

-----SCORING-----
1 point if hit (break lance) your opponent during a pass but they are not knocked off.
3 points and match ends if opponents falls off their horse (unseated).

----PLAYER POSITIONS----
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

Example--- This is the players starting positions
-----
Player 1
0
0
Player 2
-----
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

As the players roll their attacks their positions on the field will change
Example of a Player Getting hit:
Player 1 scores a hit against player 2 and Player 2 misses their attack
-----
Player 1
0
M
H
Player 2
-----
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.

-----HELPFUL TIPS-----
Most Used Race: Human.
Most Used Class: Knight.
Most used attack stance is AGGRESSIVE.

Fester the Human Knight currently holds the high score for the joust game with a score of 5

Player 1 create your character!
What is your name?
```


Programing building player characters based on user input.

```

Main X statistics
499         print("Please Choose YES or NO")
500         print()
501         if rules_input.upper() == 'YES':

Console X
Main.py [C:\Users\Batman\AppData\Local\Programs\Python\Python310\python.exe]
Player 2
-----
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.

-----HELPFUL TIPS-----
Most Used Race: Human.
Most Used Class: Knight.
Most used attack stance is AGGRESSIVE.

Fester the Human Knight currently holds the high score for the joust game with a score of 5

Player 1 create your character!
What is your name? Beavis
What character do you want to be? Choices: Human, Dwarf, Elf, Orc: Elf
What class do you want to be? Choices: Knight, Warrior, Ranger, Barbarian: Ranger
What color do you want your banner to be? Khaki
Welcome to the joust Beavis the Elf Ranger flying the Khaki banner!
-----
Below are your character stats for the joust!
-----
Defense: 16
Attack Bonus: 3
Animal Handling: 7
-----
Player 2 create your character!
What is your name? Sango
What character do you want to be? Choices: Human, Dwarf, Elf, Orc: Elf
What class do you want to be? Choices: Knight, Warrior, Ranger, Barbarian: Warrior
What color do you want your banner to be? White
Welcome to the joust Sango the Elf Warrior flying the White banner!
-----
Below are your character stats for the joust!
-----
Defense: 17
Attack Bonus: 5
Animal Handling: 2
-----
1 We are ready to Joust!
Starting Positions
-----
Player 1
0
-----
0
-----
Player 2
-----
```

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.

```

Main Main X statistics
499         print("Please Choose YES or NO")
500         print()
501         if rules_input.upper() == 'YES':
    <

Console X
Main.py [C:\Users\Batman\AppData\Local\Programs\Python\Python310\python.exe]
Starting Positions
-----
-----
Player 1
0
-----
0
Player 2
-----
Attack Stance Choices: Normal, Aggressive, Defensive, Braced, High in Saddles (HIS)
Beavis choose your Attack Stance: Normal
Attack Stance Choices: Normal, Aggressive, Defensive, Braced, High in Saddles (HIS)
Sango choose your Attack Stance: Aggressive
Beavis are you ready to attack?: Yes
Beavis has rolled to Attack!
Sango are you ready to attack?: Yes
Sango has rolled to Attack!
Beavis attacks with a score of 19 against Sango's defense score of 12
Sango attacks with a score of 14 against Beavis's defense score of 16
Beavis strikes a blow!!!
Beavis Roll for Damage, Sango make a brace check!
Beavis are you ready to roll for damage?: Yes
Beavis has rolled for damage!
Sango are you ready roll your brace check? Yes
Sango has braced for incoming damage!
Beavis hits Sango for 6 damage against Sango's brace check of score 4
Sango has been struck down!
-----
Player 1
0
-----
M
X
Player 2
-----
Beavis's score: 4 | Sango's score: 0
Beavis the mighty Elf Ranger raises their Khaki banner in victory!
Do you want to play again? (YES/NO):
```

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

1 • SELECT * FROM joust_instances;

2

3

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	Elena	Elf	Dwarf	Barbarian	Ranger	16	3	7	18	5	5	White	Violet	NORMAL	BRACED	4	0
98	Kol-EI	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	Goldil	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).

1 • 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)

Limit to 1000 rows

```
1 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
105	Maleficent	T'Challa	Orc	Elf	Knight	Ranger	18	6	-4	16	3	7	Pale Green	Yellow	AGGRESSIVE	NORMAL	0	0

Apply Revert

Updating data based on player choices and joust outcome.

Limit to 1000 rows

```
1 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
105	Maleficent	T'Challa	Orc	Elf	Knight	Ranger	18	6	-4	16	3	7	Pale Green	Yellow	AGGRESSIVE	NORMAL	-4	0