Can’t Stop game documentation, by file

Word wants everything capitalized. I do not capitalize my functions, variables, or file names, so ignore the capitals.

1. Main.py
   1. Imports
      1. Helper
      2. Pregame
      3. Board
      4. Turn
      5. Player\_progress\_funcs
   2. Functions
      1. Main()
         1. Calls all functions necessary to run the game.
         2. Takes no arguments.
2. Pregame
   1. Imports
      1. Helper
   2. Functions
      1. Get\_player\_name()
         1. Asks for a single string that represents one player’s name.
         2. Returns the name.
         3. Takes no arguments.
      2. Display\_players\_list(names\_list)
         1. Takes a list of player names, which are strings.
         2. prints each name in a numbered list.
      3. Add\_new\_player\_bool()
         1. Asks the user if they want to add a new player to the lineup.
         2. Returns the user’s answer as a Boolean.
         3. Takes no arguments.
      4. Remove\_player\_bool()
         1. Asks the user if they want to remove a player from the lineup.
         2. Returns the user’s answer as a Boolean.
         3. Takes no arguments.
      5. Remove\_player(names\_list)
         1. Takes a list of player names, which are strings.
         2. Displays the list of player names
         3. Asks the user for the index of the player they wish to remove.
         4. Returns a list of player names, not including the removed player.
      6. Add\_players\_loop(names\_list)
         1. Takes a list of player names, which are strings.
         2. Displays the list of player names
         3. Assumes the user has already stated they wish to add a player to the list
         4. Loops through asking for and adding players and asking if the player wants to add another name.
         5. Returns the updated list of players.
      7. Too\_few\_players\_check(names\_list)
         1. Takes a list of player names, which are strings.
         2. Checks if there are too few players.
            1. If there are, forces the user to add players until there are enough players.
         3. Returns updated list of player names.
      8. Remove\_players\_loop(names\_list)
         1. Takes a list of player names, which are strings.
         2. Assumes the user has already stated they wish to remove a player from the list.
         3. Loops through asking for the index of and removing players and asking if the player wants to remove another name.
         4. Calls too few players check.
         5. Returns the updated names list.
      9. Get\_names\_list()
         1. Asks the user for a list of players’ names.
         2. Returns the names as a list of strings.
         3. Takes no arguments.
      10. Get\_players\_names()
          1. This is the main function.
          2. This calls all the functions to get the list of players and makes sure it is correct before finalizing it.
          3. Returns the list of player names.
3. The\_round.py
   1. Imports
      1. Helper
      2. Random
   2. Functions
      1. Roll()
         1. Returns a list of 4 random integers between 1 and 6, representing the dice.
            1. E.g. [4, 2, 1, 2]
         2. Takes no arguments.
      2. Get\_combos(dice)
         1. Takes the list of 4 integers, intended to have been generated from the roll() function
            1. E.g. [2, 5, 1, 1]
         2. Returns every possible pair of sums from the 4 integers.
            1. E.g. (from the integers above) [[7, 2], [3, 6], [3, 6]]
      3. Select\_combo(combos)
         1. Takes a list of lists(pairs of sums)
            1. E.g. [[5, 7], [4, 8], [5, 7]]
         2. Asks the user to select one of the sums by its index.
         3. Returns a selected pair of sums
            1. E.g. [5, 7]
      4. Get\_allowed\_cols(progress)
         1. Takes a dictionary as an argument
            1. i.e. {col:[progress, col\_max],col:[progress,col\_max]}
            2. this is the information for the active player
         2. returns a list of columns that are not currently maxed out by white pieces or by this player in some way.
      5. Filter\_combo\_through\_white\_list(combo,white\_list)
         1. Takes a list of sums and a keys object of all the columns containing white pieces.
         2. Returns a list of sums which only includes integers within the keys object.
      6. Filter\_combos\_through\_white\_pieces(combos,white\_list)
         1. Takes a list of lists of sums and a keys object of all the columns containing white pieces.
         2. Returns a list of lists of sums which only include integers within the keys object.
      7. Run\_round()
         1. This is the main function for this file.
         2. Takes a dictionary of white pieces as an optional argument.
         3. Runs a single roll of the dice until a combo is selected.
         4. Returns the selected combo.
   3. Turn.py
      1. Imports
         1. Helper
         2. The\_round
         3. White\_pieces\_func
         4. From board import get\_board\_dict
      2. Add\_new\_white\_pieces\_loop(new\_white\_pieces,white\_pieces)
         1. Takes a list of sums and a dictionary of white pieces.
         2. , where each sum has been added to the dictionary with a value of 1
         3. Handles if there are two of the same number in the list of sums.
         4. Returns a dictionary of white pieces.
      3. Iterate\_white\_piece(num,white\_pieces)
         1. Takes an integer, representing a col, and a dictionary of white pieces.
         2. It assumes that the col is already in the dictionary.
         3. It adds one to the value of the col in the dictionary.
         4. Returns the dictionary.
      4. Select\_one\_new\_white\_pieces(new\_white\_pieces)
         1. Takes a list of integers representing new white pieces.
         2. Assumes there is only one spot left in the white pieces and that there are two distinct new white pieces.
         3. Asks the user which integer to use.
         4. Returns the selected integer.
      5. Handle\_new\_white\_pieces(spots\_left,new\_white\_pieces,white\_pieces)
         1. Takes the number of spots left for white pieces, a list of potential new white pieces, and the dictionary of white pieces.
         2. Handles what should be done with the new white pieces.
         3. Returns the updated dictionary.
      6. Add\_white\_pieces(white\_pieces,selection)
         1. Takes a dictionary of white pieces and a list of sums.
         2. This function handles all of adding new white pieces, calling the right functions, etc.
         3. Returns the updated dictionary.
      7. Bombed()
         1. Prints a message telling the user they bombed.
         2. Returns false.
         3. Calls enter\_to\_cont()
         4. Takes no arguments.
      8. Roll\_again()
         1. Asks the user if they wish to roll again.
         2. Returns the result as a Boolean.
         3. Takes no arguments.
      9. Get\_player\_progress\_with\_max(progress)
         1. Takes a dictionary
            1. Dictionary has a players progress
            2. i.e. {col:progress,col:progress}
         2. returns a dictionary of progress on every column, where no progress is indicated by -1
            1. i.e. {col:[progress, col\_max],col:[progress,col\_max]}
      10. update\_player\_progress\_with\_max(progress,white\_pieces)
          1. takes two dictionaries
             1. progress is a progress with max dictionary

i.e. {col:[progress, col\_max],col:[progress,col\_max]}

* + - * 1. white\_pieces is a normal white pieces dictionary
      1. returns a progress with max dictionary with positions updated from the white pieces.
    1. Run\_turn():
       1. handles the rounds of rolling for user and handles the white pieces.
       2. if the player ends their turn willingly, returns white pieces.
       3. if the player bombs, returns false.

1. White\_pieces\_funcs.py
   1. Functions
      1. Display\_white\_pieces(white\_pieces)
         1. Takes a dictionary of white pieces.
         2. Prints the info in the dictionary for the user.
2. Board.py
   1. Imports
      1. Helper
   2. Functions
      1. Get\_board\_dict()
         1. Returns the board
            1. {2: 3, 3: 5, 4: 7, 5: 9, 6: 11, 7: 13, 8: 11, 9: 9, 10: 7, 11: 5, 12: 3}
         2. Takes no arguments
      2. If\_progress\_col\_place\_empty(col,place,progress)
         1. Takes two integers and a dictionary
            1. Col is the column
            2. Place is the place in that column
            3. Progress is a dictionary of progress

i.e. {col:{place:[],place:[]},col:{place:[name,name]}}

* + - 1. returns an updated progress dictionary that makes space for a certain col and place if those spaces don’t yet exist.
    1. Format\_one\_player\_progress\_col\_first(name,entry)
       1. Takes a string and a dictionary
          1. The dictionary is formatted {col:place,col:place}
       2. Returns a progress dictionary for one player’s progress with the col first
          1. i.e. {col:{place:name},col:{place:name}}
    2. format\_all\_progress\_col\_first(col,col\_progress,all\_progress}
       1. takes an integer and two dictionaries
          1. the integer is a col number
          2. the first dictionary is formatted:

i.e {place:name,place:name}

* + - * 1. the second dictionary is formatted:

{col:{place:[name],place:[name,name,name]},col:{place:[name,name]}}

* + - 1. This function add the col\_progress info to the correct col in the all\_progress dictionary
      2. Returns the updated all\_progress dictionary
    1. Add\_entry\_to\_player\_progress\_col\_first(player\_progress,all\_progress)
       1. Takes two progress dictionaries
          1. The first dictionary is formatted:

{col:{place:name,place:name},col{place:name}}

* + - * 1. The second dictionary is formatted:

{col:{place:[name],place:[name,name,name]},col:{place:[name,name]}}

* + - 1. Adds the info from player\_progress into all\_progress
      2. Returns updated all\_progress
    1. Format\_player\_progress\_col\_first(progress)
       1. Takes a dictionary
          1. i.e. {name: {col:place,col:place},name: {col:place,col:place}}
       2. returns a progress dictionary with the column first
          1. i.e {col:{place:[name],place:[name,name,name]},col:{place:[name,name]}}
    2. get\_progress\_str(progress)
       1. takes a dictionary
          1. i.e {place:[name,name,name],place:[name]}
       2. returns a string which tells the progress of all players in a specific column
    3. col\_to\_str(col\_max,progress,col)
       1. takes an integer, a dictionary, and another integer
          1. the dictionary is formatted:

i.e. {place:[name,name,name],place:[name]}

* + - * 1. col\_max is the length of a column
      1. returns a string that represents a column for the display of the board
    1. display\_board(board\_dict,progress)
       1. takes two dictionaries
          1. the first dictionary is:

{2: 3, 3: 5, 4: 7, 5: 9, 6: 11, 7: 13, 8: 11, 9: 9, 10: 7, 11: 5, 12: 3}

* + - * 1. The second dictionary is formatted:

i.e. {name: {col:place,col:place},name: {col:place,col:place}}

* + - 1. prints the board, column by column

1. White\_pieces\_funcs.py
   1. Imports
      1. Helper
      2. From board import get\_board\_dict
   2. Functions
      1. Add\_new\_white\_pieces\_loop(new\_white\_pieces, white\_pieces)
         1. Takes two dictionaries
            1. The first dictionary is for the new white pieces to add to the white pieces dictionary

i.e. {col: place, col: place}

always has no more than two entries

* + - * 1. the second dictionary is for white pieces

i.e. {col: place, col: place}

* + - 1. this function adds the new white pieces to the white pieces list
      2. returns the updated white\_pieces list
    1. iterate\_whtie\_pieces(col,white\_pieces)
       1. take an int and a white pieces dictionary
          1. {col:place,col:place,col:place}
       2. Adds one to the place in the given col
       3. Returns the updated white pieces dictionary
    2. Select\_one\_new\_white\_pieces(new\_white\_pieces)
       1. Takes a list of exactly two integers
       2. Asks the user to select one of the integers to play, since there is only one white piece left
       3. Returns a list containing only the selected integer
    3. Handle\_new\_white\_pieces(spots\_left,new\_white\_pieces,white\_pieces)
       1. Takes an integer and two dictionaries
          1. Spots left is the number of white pieces that have not been placed
          2. The new white pieces is formatted:

{col:place,col:place}

Always has exactly two entries

* + - * 1. The white pieces is formatted:

{col:place,col:place}

Can have between 0 and 3 entries

* + - 1. Adds the new white pieces to the white pieces dictionary, and handles if there isn’t enough space
      2. Returns updated white pieces
    1. Get\_progress\_for\_col(progress,col)
       1. Takes a dictionary and an integer
          1. Dictionary is formatted:

{col:place,col:place}

Has an entry for every col

* + - 1. Returns the place in the dictionary for that column
    1. Add\_white\_pieces(white\_pieces,selection,progress)
       1. Takes a dictionary, a list, and a dictionary
          1. White pieces is formatted

{col:place,col:place}

* + - * 1. Selection is formatted:

[col,col]

* + - * 1. Progress is formatted

{col:place,col:place}

Has an entry for every col

Represents one player’s progress in every col

* + - 1. Adds the selection to the white pieces, considering the player’s progress
      2. Returns the updated white pieces
    1. Display\_white\_pieces(white\_pieces)
       1. Takes a white pieces dictionary
       2. Prints the white pieces info in a user-readable way

1. Player\_progress\_funcs.py
   1. Imports
      1. Helper
   2. Functions
      1. Create\_players\_progress\_dict(names\_list)
         1. Takes a list of strings which represents the players names
         2. Returns a dictionary for players progress
            1. Formatted:

{name:{},name:{}}

* + 1. Display\_active\_player(name)
       1. Takes a name as a string
       2. Prints whose turn it is
    2. Iterate\_active\_player\_index(i,players\_len):
       1. Takes an integer, which is the index of the active player and the number of players
       2. Returns the index of the next active player
    3. Update\_player\_progress\_dict(progress\_dict,white\_pieces)
       1. Takes a progress dictionary and a white\_pieces dictionary
          1. Both are formatted:

{col:place,col:place}

* + - 1. Adds the white pieces info to the progress dictionary
      2. Returns the updated progress dictionary

1. Helper.py
   1. functions
      1. can\_be\_int(string):
         1. takes a string as an argument.
         2. returns whether that string can be an integer as a Boolean.
      2. Yes\_no\_question(string)
         1. Takes a string, which should be a yes or no question to ask the user
         2. Asks the user this question along with “ (Y or N) “
         3. Returns Boolean True for an affirmative response or False for a negative response.
         4. The response from the user is not case-sensitive.
         5. It takes y, yes, or yeah as affirmative.
         6. It takes n, no, or nope as negative.
         7. Any other response will tell the user “Please select Y or N ” and take a new response from the user until an acceptable response is obtained.
      3. End\_section()
         1. Prints two blank lines to end a section.
         2. Takes no arguments.
      4. Enter\_to\_cont()
         1. This function requires the user to press enter before the script continues, allowing the user to pause and acknowledge information.
         2. Calls End\_section()
      5. Print\_numbered\_list(the\_list)
         1. Takes a list, usually of strings.
         2. Prints each element in the list on its own line in a numbered list, where the numbers are the indexes of the element.
         3. That is, the first element will be numbered “0”
      6. Print\_unnumbered\_list(the\_list)
         1. Takes a list, usually of strings.
         2. Prints each element in the list, on its own line with no numbers.
      7. Remove\_duplicates\_from\_list(the\_list)
         1. Takes a list
         2. Returns a list without duplicates.
      8. First\_char(string)
         1. Takes a string.
         2. Returns the first character of the string.