**CSP Function**

General Architecture:

*Keep in mind tables are assigned to kids, not kids to tables*

Algorithm analyzes and chooses kid from list of remaining kids who need to tables

List of possible table assignments are found

If there are some possible assignments, value algorithm evaluates (with some random chance) which one should be assigned and then assigns it (includes altering all sorts of variables)

If there are no possible assignments, smart backtracking algorithm begins

Backtracks to what is deemed the right place

Goes forward from there with different assignment and remembers that the assignment it has is impossible with the preceding kids assigned as they were

Arguments:

* Kid Array
  + JSON Array of kidIds
  + kidId.name (String)
  + kidId.isBoy (Boolean)
  + kidId.isRowdy (Boolean)
* Tables Array
  + Array of ints
  + Each int represents one table and the value of the int is the maximum amount of kids allowed at that table
* Blacklist Hashmap
  + Key: Kid ID
  + Value: Array of kids who are blacklisted from being with the key Kid ID

Goal:

Write a function that outputs an array of names that represents the tables

* Guys and girls evenly spaced
* Rowdy kids evenly spaced
* All blacklist constraints satisfied

General function goals

* Impossible to get in endless loop
* Has to make progress on the problem over time (things need to be ruled out)
* Has a random element to it

Function:

Starts by creating some variables for later use using the arguments given

* Table Data Hashmap
  + Key: Table ID
  + Value: Array of ints of length 4
    - Array[0] – How many kids can be added to the table
    - Array[1] – How many more girls or boys there are (2 means there are two more boys than girls, -1 means there is one more girl than boy)
    - Array[2] – How many rowdy girls are at the table
    - Array[3] – How many rowdy boys are at the table
* Possibilities Hashmap
  + Key: Kid ID
  + Value: Array of possible tables for the kid to be assigned

Also creates some empty variables to be filled and used once the function starts functioning

* Documentation Hashmap
  + Empty Hashmap to be filled with documentation on assignments and results and what not, to be used in effective backtracking system.
* Assignment Hashmap
  + Empty hashmap to be filled with assignments when they are made, used to create output when function is done
* Memoization Hashmap
  + Empty hashmap to be filled with things that didn’t work so that the function doesn’t get into any loops doing the same thing over
* xDepth Int
  + Set to zero at beginning, changed and used to know current sweep depth
* Order Array
  + Used to hold the order that kids need to be assigned, stays the same even if backtracking (this is so memoization can work and function can gradually rule stuff out)

**Function can be separated into four main functional parts**

1. Choosing of Kid to be Swept
   1. Swept – I’m using this to mean the rest of the stuff that follows
2. Choosing of Tables/Possible Assignments
3. Analyzation of possibilities/Assignment
4. Backtracking if necessary

1. Choosing Kid

* Chooses kid out of remaining possibilities
* Choice based first off of number of constraints that the kid is involved in (One with most is chosen/random choice out of ones with equal numbers that are above zero)
* If none of the remaining kids are involved in any constraints, decision is based off of amount of remaining possibilities. This is stored in the possibilities array (one with least is chosen/random choice out of all the ones tied for least)
* If they are all the same, the choice it totally random
* Adds kid to order array (he is in there permanently now)

2. Choosing of Tables/Possible Assignments

* Once kid is picked, gets all the possible assignments for that kid
* Goes through each one and sees if it is valid
  + Might not be valid because such an assignment would remove the last possible assignment from one of the kids who has yet to be assigned
  + Causes
    - Fills up table that is last possible assignment of someone
    - Joins last possible assignment of someone blacklisted
    - Makes the |gender average| > some number (calculate this number early)
    - Makes number of rowdy people too high (calculate this number early)
* Gathers a list of possible assignments (tables)

3. Analyzation of possibilities/Assignment

* Assesses the possible assignments for value
  + Lowest value is most number of possibilities it takes away from remaining kids
  + Assigns the one that is of highest value
* What does assignment entail?
  + Changes tableData
  + Changes Possibilities
    - Blacklist
    - Filled up
    - Gender Average (Might have to put possibilities back)
    - Rowdy number
  + Documents assignment and what the assignment caused in documentation array
  + Puts assignment inside assignments array

4. Backtracking

* If there are no possible assignments
* Creates list of assignments (toBeEval) that took away possibilities from those kids whose last possibilities would have been taken away by an assignment to the current kid and also of assignments that took away possibilities from the current kid that would have been valid (wouldn’t have taken away any of the other kids last possibility). Basically backtracks to last thing that fucked with all the current kids in question, including the assignment which removed valid possibilities from the current kid. (if something removed a possibility from the current kid that would have been invalid: add to the list a tuple of )
* Evaluates last thing that fucked with shit. If it finds another valid possibility that wouldn’t fuck with shit, it backtracks to that and puts in the memo that previous possibility that is had is no longer a possibility with the same preceding assignments. It then goes forward from there.
* If no other valid possibilities are available there, it goes on to the next thing in the list and keeps going until it finds a valid change and goes from there
* sees what possibilities that aren’t available would have hypothetically been valid and sees what fucked with that shit. Then puts the things that fucked with that shit (in this case it’s just the assignment that took away the possibility that would be valid/one that took away an assignment that is invalid and why its invalid) in toBeEval which is sorted by which xDepth the assignment that fucked with shit is.

What happens if the hypothet assignment fucked with two kids, or more?

Get all the assignments that fucked with any of the kids and goes to the latest one that fucked with all the kids. (has to be smart)

List includes

We need some type of recursive shit

*Assignments can only take away the possibility of itself*

toBeEval order of shit

First all the direct stuff is added:

Assignments that took away possibilities from those kids whose last possibilities would have been taken away by an assignment to the current kid

Assignments that took away possibilities from the current kid that would have been valid (wouldn’t have taken away any of the other kids last possibility)

If its multiple shits, then we just try the one that occurred at the lowest xDepth, smart though