**The detail of algorithm and GUI**

**The part of Algorithm**

Recommend uses the state Tree's greedy search. Here, a state is a method of composing teams, where any two students are swapped, and the state of the system changes from one state to another. The status tree is to root the current state and exchange all "reasonable" states of two students as children nodes (this is also called expand a form). Then find the best one in The Children nodes and do the same operation.

Since swapping any two students gives a new state, the children of each expansion are the 20 choose two levels of complexity, so we used a strategy to reduce the complexity by swapping the best overall team for the worst, with all students in both teams changing once. In each swap, we only try to switch one student to another team, so that the maximum number of nodes to be expanded each time is 4\*4 = 16.

Also, when searching, we stipulate that the state tree should be expanded at most five layers, which significantly reduces the complexity of the algorithm.

The algorithm refers to Best First Search, USES priority Queue to sort the expanded status nodes, and selects the best node to extend each time. And remember that the best status node so far is returned after the algorithm is finished.

To judge the quality of a state, we use a combined cost function, 0.7 \* STD dev skill fall + 0.2 \* STD dev Average Skill + 0.1 \* STD dev preference percentage. Because we think the balance of skill fall is the most important.

**The part of GUI**

I used Scene Builder to design parts of the GUI. Each designed window will have a Class to manage, and there are related methods to call the data. If the data content increases, the results appear at a slower rate, but the GUI part of the design is not affected.