Convention: the chessboard’s index is as follows

To win the game, one should try to place their chesses in an array with length 3.

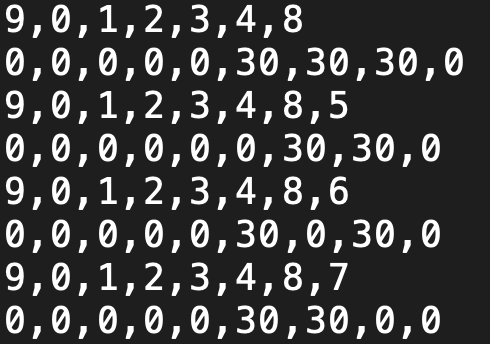
This project’s goal is to create a map, all\_learning. The key should be a vector<int>, which starts with a “9” (the starting point of the game), follows by an array of numbers, such as “9,1,3,6,2,8”. The value should also be a vector<int>, whose size=9. As the game goes on, a recorder records the steps that players play, and store them into a vector “allmoves”. In order to decide where should place the chess next, the computer program search through all\_learning, and find the key which coincides to allmoves. The value of that key is the weight of the place that computer program should place its next key. For instance, if the current allmoves is {9,1,4,6,7}, then the program looks for the key {9,1,4,6,7} in all\_learning. Suppose the value is {25,0,30,50,0,35,0,0,15}, the program then chooses a number randomly according to weight shown in the list, i.e., weight 25 for 0, weight for 1, etc. There are three sub-programs to accomplish this, initializing.cpp, training.cpp, and playing.cpp.

1. initialing.cpp:

First, add a key-value pair to all\_learning, {{9}:{30,30,30,30,30,30,30,30}}, where 30 is “constexpr int initial”, which is tunable.

Then, create longer keys iteratively. To be more precise, given the presence of key {9}, we add key-value pairs to map\_temp with key length 2, such as {{9,1},{30,0,30,30,30,30,30,30}}. Append map\_temp to all\_learning so that all\_learning contains key-value pairs with key length 1 and 2. Iterate the process to make the keys in all\_learning longer.

Finally, record all\_learning into a csv file. There are 260650 key-value pairs in all\_learning, which is expected, since . The format of the file looks like

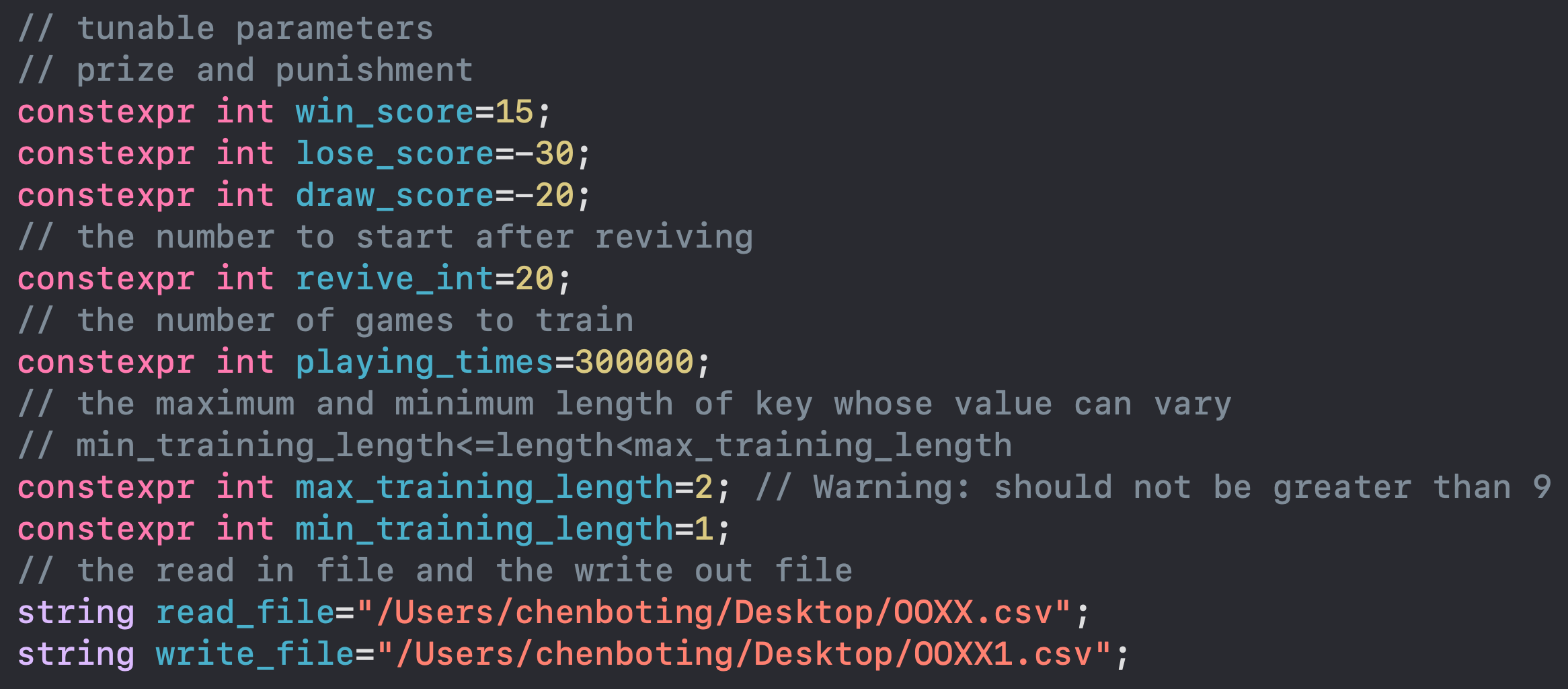


According to the file, there are key-value pairs {9,0,1,2,3,4,8}:{0,0,0,0,0,30,30,30,0} in all\_learning, etc.

1. training.cpp

This is perhaps the most difficult part of the project. I make the program play with itself, and let them learn through try add error. Initially, it has no idea what the rules are, so they place the chess randomly. The only thing the program would know is when the winner of game appears, as well is who he is. Then the program will cut allmoves into pieces, e.g., cut {9,4,0,2,6,1,3} into {9}, {9,4}, {9,4,0}, etc. The winner of this game is player2, who drops his chess second, aka the defensive player. As a result, we punish key {9} by reducing the 4th element of its value, award key {9,4} by increasing the 0th element of its value, etc. After playing lots of games, we sufficiently award and punish the keys, and record our new all\_learning into a new csv file.

There are several tunable parameters in training.cpp.



win\_score: the award I give to the winners.

lose\_score: the punishment I give to the loser.

draw\_score: the award/punishment I give to both players in a draw game.

revive\_int: if all the weights are punished to 0, revive it with all possible moves’ weights equal to revive\_int.

playing\_times: the number of gameplays I ask them to play.

max\_training\_length: the maximum length of keys I want to train in this run.

min\_training\_length: the minimum length of keys I want to train in this run.

read\_file: the file I want to import all\_learning from.

write\_file: the file I want to export the trained all\_learning to.

Of course, the toughest part of this training program is to find the most suitable win\_score, draw\_score, lose\_score to each length of the key.

1. playing.cpp

In this project, convention is changed to , which is more user friendly.

This program imports the trained all\_learning map, and players can play against it.

