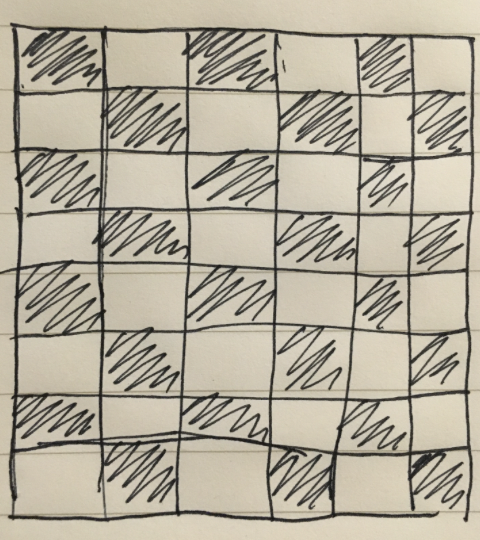
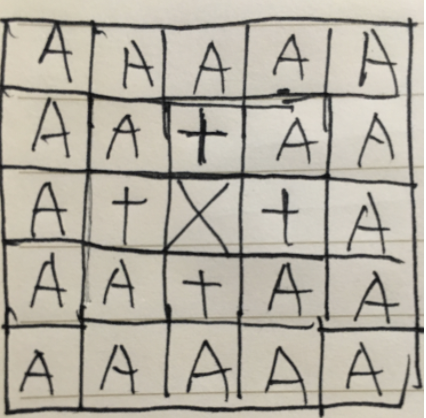
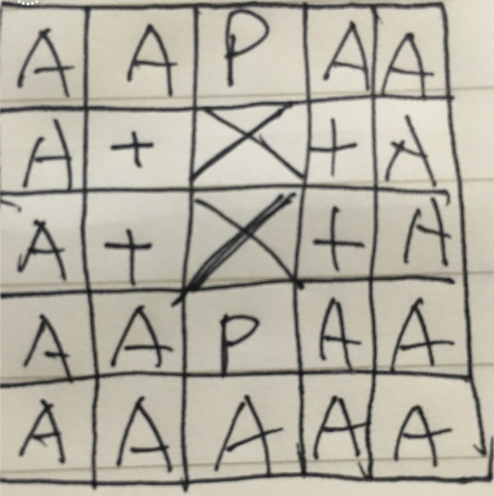
CS32 Project 3 report TianqiZhu 504291494

1. Description of data structure
   1. For project 3, the order I finish the project is following: first I implement game.cpp but I left the play function. Then I go to implement board.cpp. Then I implement the human player. Finish the human player, I go back to finish play function in game.cpp then I implement the mediocre player and then good player.
   2. In game.cpp, I use a STL vector to store all ships and ship information in a single game.
   3. In board.cpp, I use a STL vector to store all ships that has been placed on the board. Besides, I use a 2-dimensional char grid to represent the board. In detail, on the grid ‘.’ means available position to place ship or empty position, ‘#’ means blocked position, ‘X’ means shot hits a ship, ‘o’means shot missed and characters like ‘A’ ‘B’ are ships. So every time, any action happens, the cahr on the board will change to store the information.
   4. In player.cpp, I use a 2-dimensional char grid to represent different player’s attack strategy. On the grid, ‘A’ means available position to attack, ‘+’ means recommended attack position, ‘P’ means highly recommended attack position and ‘N’ means attacked position. The grid will update after each step to store information.
   5. I used several do-while loop in the display of the game. This really helps because at start, the game begins automatically, and the game stops if some conditions are met. Which is exactly do-while loop.
2. Goodplayer
   1. Goodplayer has same place ship function as mediocre player and goodplayer does not take action based on enemy’s attack(do nothing with recordopponentattack function). However, goodplayer has different attack strategy.
   2. In stage 1, the goodplayer does not randomly hit on enemy’s board, but instead it hits from upper left corner using 2 for loops and each time it attacks, it jumps thru one position as in standard game, the minimum length of ship is 2. So to quickly find a ship, goodplayer does not go thru every position but go thru positions that are separated by 1 position. Like the image follows:
   3. After each hit, good player will record the near 1 grid in 4 directions as ‘+’, recommend attack point. It will dynamically update. Shown in image follows:
   4. If it keep hitting in stage 2, it will generate high recommended attack point on the same row or col and next attack will go to these highly recommended attack point in priority. Shown in follows:
   5. Base on my own test, my good player can win 97.5% games against mediocre player. I runned the game in a total about 20000 times. But if the goodplayer does not start from upper left corner to counter the mediocre player(as mediocre player place ship from upper left corner), the win rate is about 87%. The problem for good player is: sometimes the recommended attack place ‘+’ is useless as there is probably no ship. However, I still go them in a priority to normal ‘A’ points as this prevents a lot of bugs. In a conclusion, my good player can easily beat mediocre player.
3. Psueodocode
   1. game.cpp play

Player\* GameImpl::play(Player\* p1, Player\* p2, Board& b1, Board& b2, bool shouldPause)

*check if placed ship for both player*

*check if there is any human player*

*repeatedly :*

*display player 2’s board*

*p1 to attack*

*if valid attack*

*record attack result*

*display p2 board*

*if not valid attack*

*record attack result*

*cerr*

*p2 record oppoenents attack*

*if all p2’s ship destroyed*

*end game, return*

*check wait for enter*

*display player 1’s board*

*p2 to attack*

*if valid attack*

*record attack result*

*display p1 board*

*if not valid attack*

*record attack result*

*cerr*

*p1 record oppoenents attack*

*if all p1’s ship destroyed*

*end game, return*

*check wait for enter*

*return nullptr*

* 1. mediocre player recursively placeship

bool mediocre::place(int nShips, Board& b)

*check the total number of ships to place*

*if there is no ship to place directly return true*

*from the upper left corner repeatedly go thru each position on the board*

*check if can place the first ship on that point and temporarily put ship there, yes then*

*check if can place the left totalnumber of ships -1 ships on board if yes then*

*directly return true*

*if no*

*replace the temporarily putted first ship*

*move to the next point on the board*

*return false*

* 1. good player recommendattack()

Point GoodPlayer::recommendAttack()

*if in stage 1*

*if there is any ‘+’ or ‘P’ on grid, repeately find a ‘+’ or ‘p’*

*return that random position*

*if there is no length 1 ship*

*repeatedly find and return an available point from upper left corner of grid and row and col has same remainder when divided by 2.*

*if there is length 1 ship*

*repeatedly find and return an available point from upper left corner of grid*

*if in stage 2*

*if there is any ‘P’ on grid*

*return that position*

*if there is any ‘+’ on grid, repeately fina a ‘+’*

*return that random position*

* 1. goodplayer recordattack

*if in stage 1*

*if not hit*

*stage 1*

*if hit and destroyed*

*stage 1*

*if hit not destroy*

*stage 2*

*change nearby 4 postions to ‘P’*

*if in stage 2*

*if not hit*

*stage 2*

*if hit and destroyed*

*stage 1*

*if hit not destroy*

*stage 2*

*change nearby 4 postions to ‘P’*

*check if consecutively hit*

*if yes change on the same row or col’s ‘+’ to ‘P’*