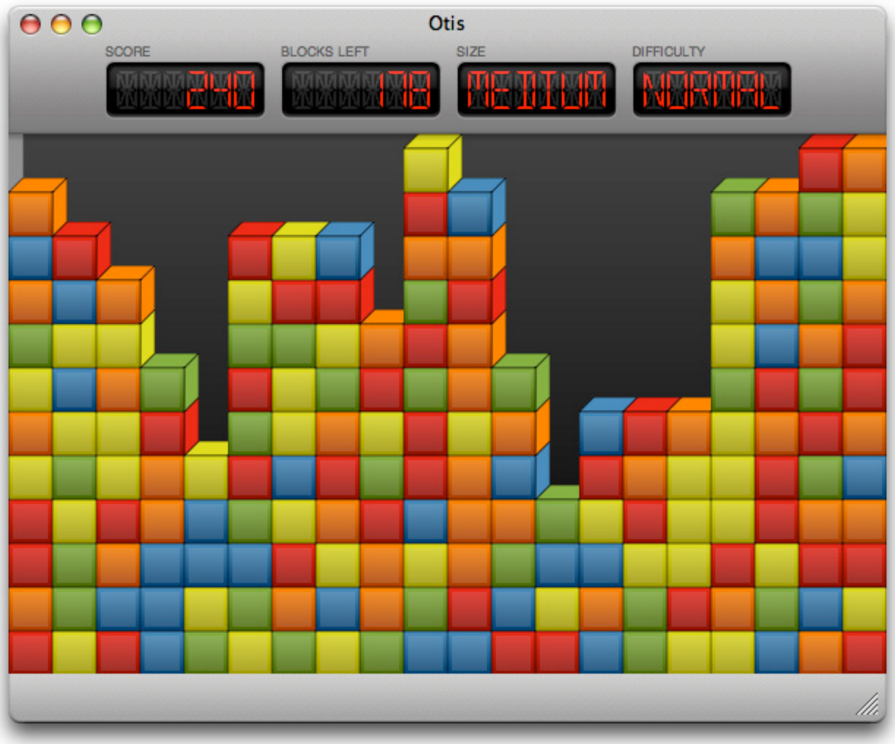
FINAL REPORT



Chao Hsuan Shen 50730113
Fall 2012

AI FINAL REPORT

The architecture of this report

1. Why I am doing this, why I take this course?
2. What I have done & how ?
3. Success and failure?
   How can I do better if started all over again?
   What I have learned and what I will learn in the future?

Why I an doing this, why I take this course?

Last summer vacation, I was intrigued by the idea of artificial intelligence. Inspired by a website : singularity.org , I wish I could know more about AI and programming. So I choose the course. I've never written any single program before, don't know any of programming language ,never taken any CS course, but I do want to know more. So here I am.

What I have done & how?

The program to play chain shot game has two mode:

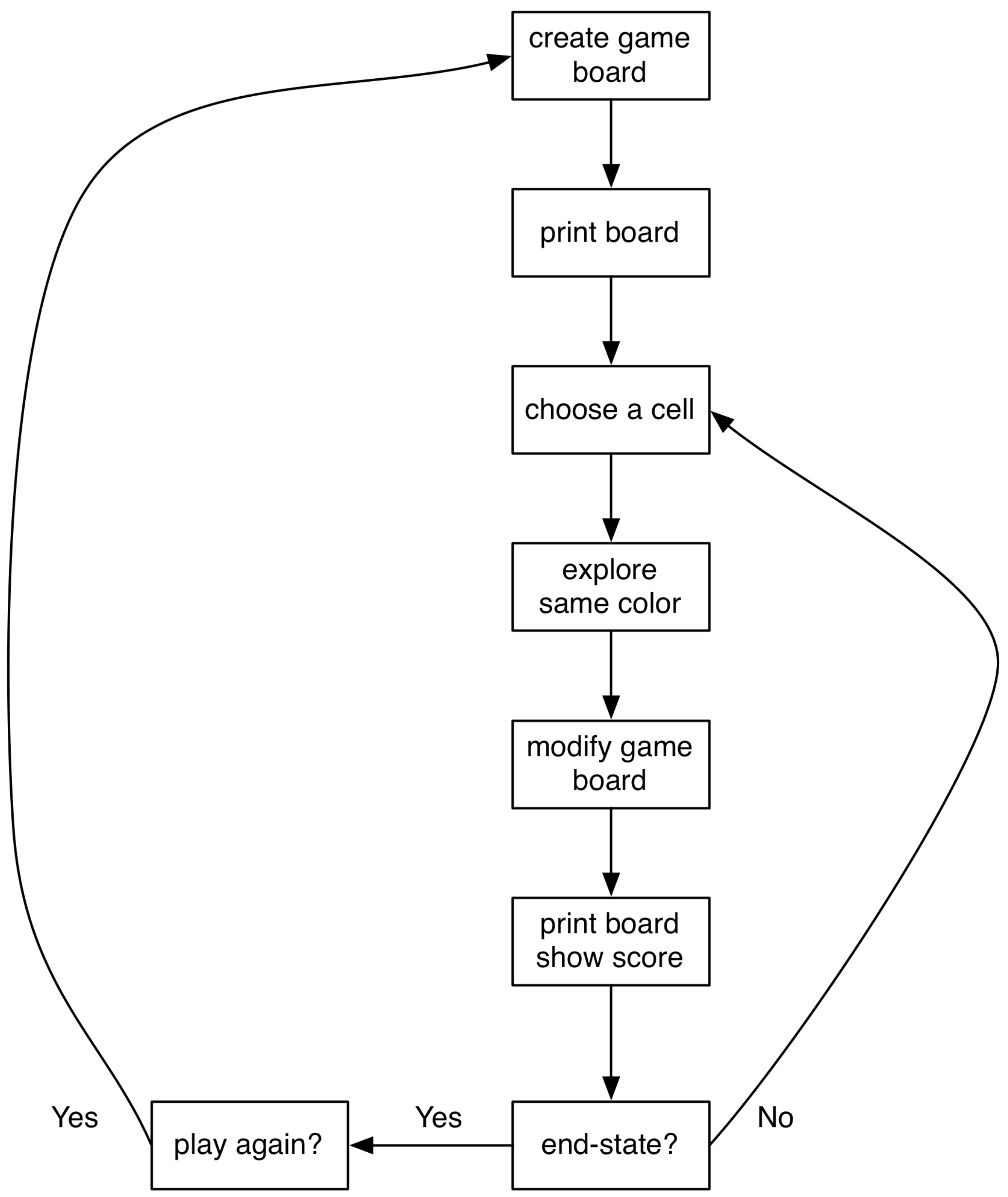
I. manual, interactive mode .

II. automatic, searching for highest score mode.

FOR MANUAL PART, THE IDEA IS VERY SIMPLE:

- create an array for game board data
- representing the board on the screen
- let user choose a point
- manipulate board data
- loop until user had no move to choose, show the score and terminate the program

Here is the flowchart.



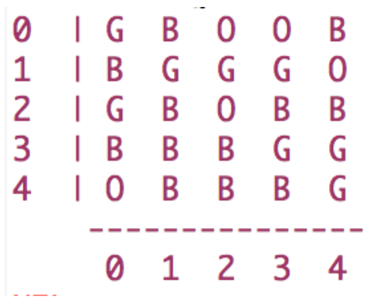
Here I have 4 key process:

1. How to store and represent the board data
2. With a given point, how to find all same connected same color cells?
3. Knowing a set of same color cells, how to manipulate board data?
4. How to decide if it is end state?

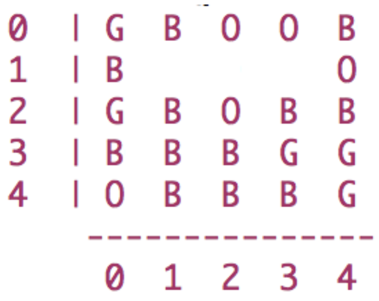
1. How to store and represent?

   At the beginning I create a global variable ,*board*, to store board data. The data structure is an 2D array. Instead of choosing a vector or pure simple list, 2D array has intrinsic coordinate system, which help me to visualize the board and thus makes it easier for me to develop means of manipulating board data.

   During the game, cells are eliminated set by set, I use "nil" to represent those eliminated cells. But those "nils" can't be printed on screen as "nil", so I create a small process, "nil-filter", to convert every nil cell to " ", empty space, so that those eliminated cells would be empty space on the screen.



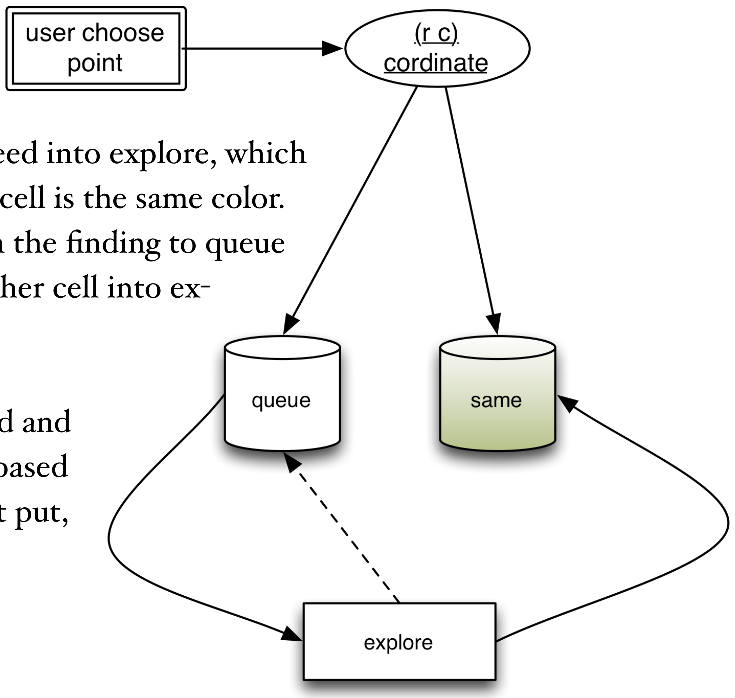
```
0  | G  B  O  O  B
1  | B  G  G  G  O
2  | G  B  O  B  B
3  | B  B  B  G  G
4  | O  B  B  B  G
     ---------------
     0  1  2  3  4
```



```
0  | G  B  O  O  B
1  | B           O
2  | G  B  O  B  B
3  | B  B  B  G  G
4  | O  B  B  B  G
     ---------------
     0  1  2  3  4
```

2. How to find same color cells? User choice will be push into queue and same. Cells in queue feed into explore, which will check if up, down, left, right cell is the same color. If it find the same color cell, push the finding to queue and same. Then queue feeds another cell into explore and do it again.

   At the end, queue will be depleted and same has all the same color cells based on user choice. So same is the out put, that is a list of coordinates.



3. How to manipulate board data?

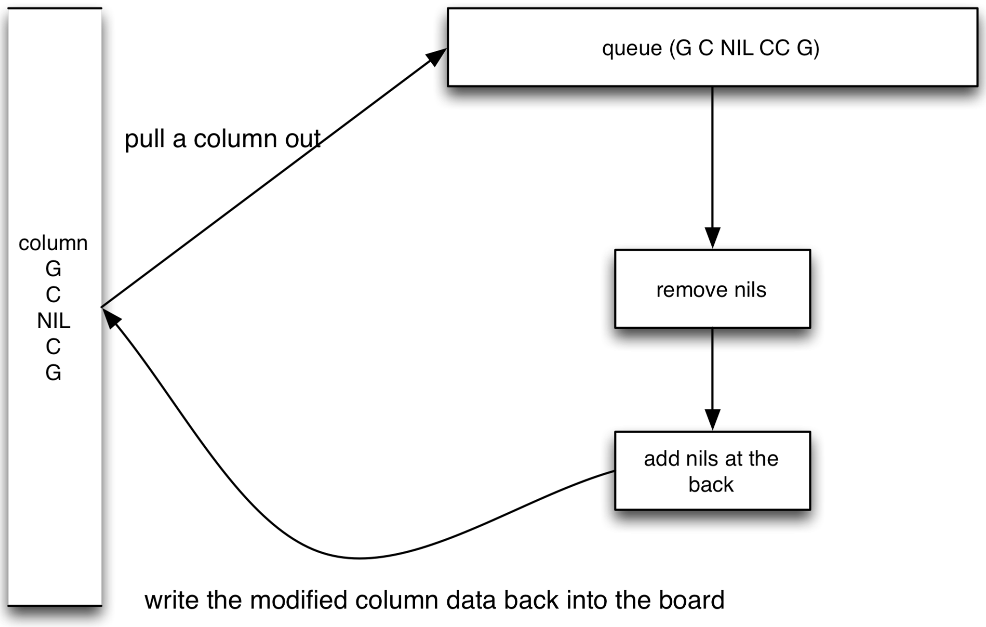
Now I have a set of coordinates of same color cells. Manipulation has 3 steps:

1. Nilize

    - I have to write those same color cell off. Write "nil" to the place that indicated by "same" list.

2. Downward

    - First I create a sub-routine to modify one column

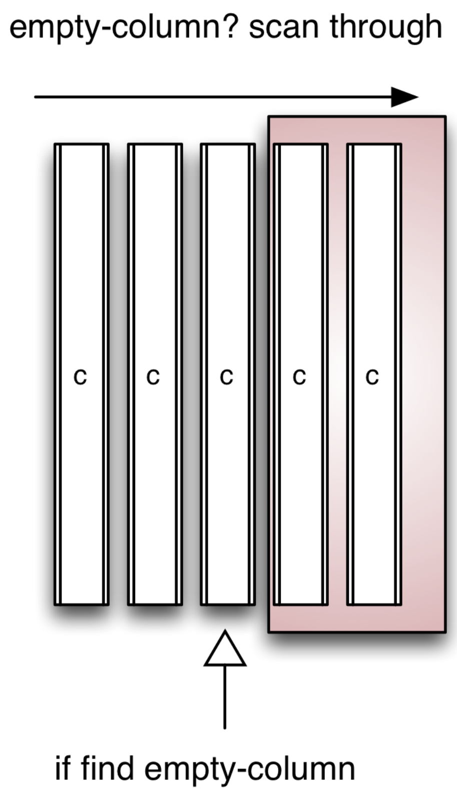


    - Then apply this routine to every column of the board.

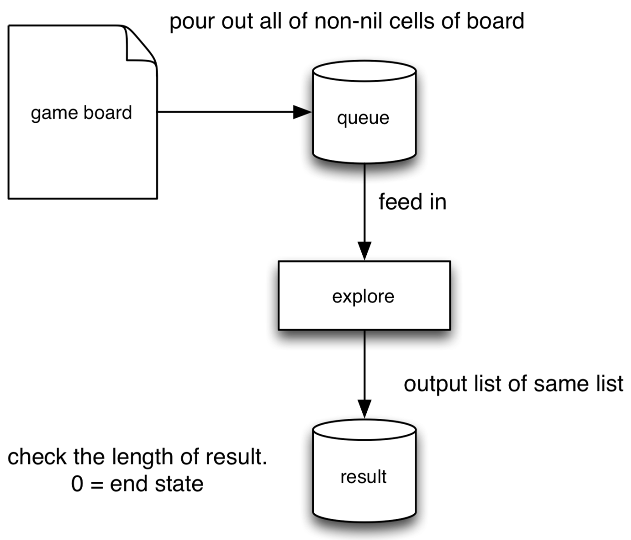
3. Left-shift

    - First I create a sub-routine to detect if the board has empty columns

- Then I scan through the board, if find one empty column, all cells right to that column have to move one left unit, and make the last column an empty column.

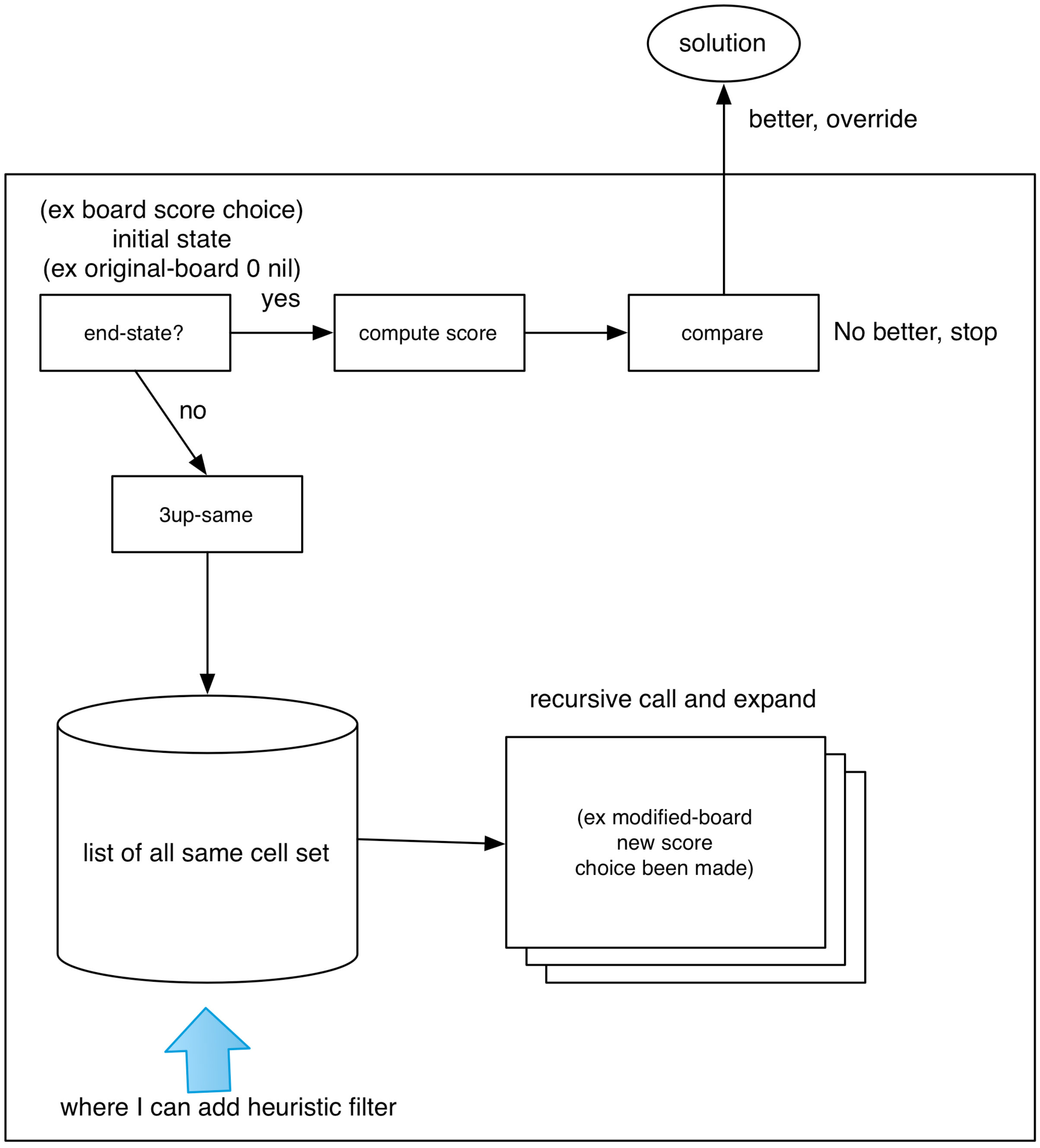


4. End state?
   The idea is simple, if the board has no any cell set that has more than 3 same color cells, the board is in the end state.



For automatic mode:

My idea is to write an exhaustive search as back bone structure, then use heuristic function to filter out those paths that are not worth to explore further, so that I don't have to explore whole problem space.



This is a DFS exhaustive search. I create an variable, solution, with bigger scope to store the best result the program found. So when DFS is running and the board hit the end state, the temporary result is compared with the "result" and if it is better one, the "result" will be overrode.

All of the heuristic function will be represented as a filter to the list of all same cell sets in order to reduce the searching space.

Success and failure?
How can I do better if started all over again?
What I have learned and what I will learn in the future?

The biggest success is that I complete a software project mostly by myself. I say so because I deliberately avoid asking other people questions, trying to solve whatever problem in my way. If I can't solve it at the moment, I study and search, trying to find a new perspective to the problem and do it again. The major disadvantages of this approach are slow progress and usually end up into local maximal, by which I mean local maximal is that every solution I made is accordant to my ability, since this is the first programming experience, the result of the program is inevitably limited by myself, inevitably not so good.

However, those disadvantages are exactly what I need for the first time learning experience. Slow progress gives me time. Time to think a problem again and again, time to face my blind spot. That is the time makes me suffer and also the time for me to think and grow. The limited, compromised result reflects what I can do now, and reminds me what I need to learn in the future, for all those needs come from the time I spent on fighting uncertainty and the suffering due to unable to deliver ideal results. That is the biggest success, motivation and interest to moving forward.

The failure is obvious, I fail to apply any heuristics in the program. All I can do is to reduce the size of searching space so that the program can deal with 25*25 board size within 30 seconds. The reason is not that I don't know any heuristic to apply but is all about a simple question: "How to represent an idea?"

To my understanding, programming is all about representing how-to ideas by programming language to achieve certain kind of purposes. In this project, the purpose is to find the best score of the game, the idea is all kinds of procedures about how to find it, and the programming language is Common Lisp. This is global scope concept of the overall program. In the program, it has many sub-routines. All of them represent this concept in lexical scope. Every sub-routine has its own small purpose, procedures to achieve it.

My failure to apply any heuristic in the program is the counterpart of overall failure to answer the question, “how to represent an idea?” For here I mean overall is because the failure fit into any scope, from the whole program point of view to sub-routines point of view. Then I can see the deep-rooted reason of such a failure has two parts: my idea is so limited and it is so hard for me to represent any ideas in Lisp. My idea is so limited because I see problems in such a boring, uncreative, and dull way so that I cannot come out a different perspective to catch specific problem. Under this situation, I can’t easily pivot when the current idea doesn’t work or I can’t represent such idea in Lisp. The costs are time and confidence. The reason why it is so hard to represent any idea in Lisp is that I don’t know how to communicate with computer at all. For me, to ask a computer to achieve some tasks is like I am the novice sorcerer trying to cast a spell to turn a frog into a princess, but what I get is a pig. The ideal reaction is seldom the same as the real reaction from Lisp interpreter. I may just not proficient enough to handle spells of spirits in computer. The cost is the same, confidence and time. Then time is running out, the code I hand in is the best I can do until now.

If I can do it all over again, will I do better? And how?

Sure I can do better. Actually form the first draft code to the final code, I make lots of progresses of handling Lisp. But to know how to get a better result is to know the following three question:

1. How to solve time problem?
2. How to get ideas? Different perspectives?
3. How to cast right spells to have the right reaction without any other side effects?

In order to know how to do better, I must to address what I have learned so far and what I will learn in the future. The comparison is based on the difference between the first draft code and the final code.

When I got the idea of modularity and black box abstraction, I can build simple idea from small primitives, and use abstractions of simple ideas to built more complex ideas. The code I wrote is more reusable than ever. I start to see the commonality between two ideas and extract that commonality to form a higher oder procedure. Be able to see the commonality of idea is the ability to refine ideas. To program in such way, procedures are more and more controllable. The result of the procedure is becoming expectable and side effects are less troubling me. Recursion is a brand new idea for me. I’d never see the world via recursive point of view before. So all draft codes are written in applicative style. When I get the idea of recursion, it seems to turn my mind from 2D to 3D. I wonder that ideas can have depth, likes dreams within dream. When I start to see the recursive part of problem, I can solve the problem in more succinct style, which I apply some recursive style in the later codes. Finally, during the search for better IDE of Lisp, now I use emacs with slime to write Lisp code. Because emacs is such a powerful editor, now I write C code, Scheme, and Python under emacs.

Though progresses I made between two code submission are limited, the pattern is a miniature of overall advance in computer programming. To me, all three problems are interconnected rather than independent to each other. Time problem is pervasive. Codes I wrote is so slow because I don't know how to make it faster. To my knowledge now, make it work is hard enough, not to mention how to make it fast. So when I finished the first time DFS implementation on Lisp, the code is far from optimal. It contains too many unnecessary computing, using way too inefficient algorithm to process. The code is like chunky shacky robot that is going to fall apart at any moment. Because it is the spaghetti of inefficient, unrefined ideas, when I try to add heuristics on it, the whole program becomes more slower. The back bone of the code can not endure and accommodate further tweaks.

So in this winter vacation, I will study introduction of algorithm to know how to measure the speed and how to make program faster generally. Also, I will study software engineering to know how to structure a program so I can deal with side effects that the scope of disaster can be controlled and to build a back bone architecture that is more capable to evolve without affecting existing functions and efficiency.

I learned a lot during this course. The losing confidence and disappointment do not diminish my interest to know more about artificial intelligence and the art of programming.