ESTR1002

Problem Solving by Programming

2020 – 21 Term 1

Course Project – Number Crush

1. Introduction

1.1 Number Crush

Figure 1: the screenshot of the Candy Crush game. Source: https://king.com/zh/game/candycrush

"Number Crash" is a single-player puzzle game, which is a simplified version of the well-known game "Candy Crush"; see Figure 1 for a screenshot of "Candy Crush." In our "Number Crush" game, a player may iteratively **swap adjacent numbers** in a game board to match rows and columns of same numbers. Once the numbers match, they are eliminated from the board and are replaced with new ones, which could potentially create further matches and eliminations. A player's goal is to collect the maximum possible score after **100 number swaps**.

1.2 Project deliverables

In this course project, you need to submit (see Section 6 in this specification for details)

1) [**basic program**] a C program to provide interactive gameplay for a human player; and
2) [**AI player**] a C function to generate swap decisions. We will test the performance of your AI player and compare its performance with the AI players of your classmates.

2. Game Rules

2.1 Setup

The game board is a **10x10 grid of 100 cells**; each cell contains a digit in { 1 , 2 , … , K }, where K is the number of unique digits in a game. Before a game starts, an initial game board of random numbers was created. See Figure 2 on the right for an example initial game board for K=4. You should implement your basic program by **setting K as 4**, but when we test your AI player, we may use other values for K (see Section 4 for more details).

1	2	3	4	3	2	3	2	3	4
2	3	4	2	3	4	1	4	1	1
4	2	3	2	1	1	4	1	4	1
2	3	4	3	3	4	1	2	1	4
4	3	3	2	3	1	2	3	2	2
2	1	4	3	4	1	4	1	2	4
2	1	1	2	1	4	2	4	3	3
4	2	4	3	3	1	2	1	1	3
2	1	1	4	4	2	1	1	4	1
1	3	4	2	1	2	4	2	1	3

Figure 2. An example of the initial game board of "Number Crush".

2.2 Basic rules

- In each step, you can swap numbers in **any two adjacent cells, either horizontally or vertically.**
- At any time, if there are three or more same numbers **along a row/column**, you get the corresponding scores and the numbers that form the row/column will be eliminated. The more numbers you can eliminate, the more scores you will get. We will give more details on score calculation later in Section 2.3.
- Note that a swap that does not eliminate any number is still an effective swap. It is because for certain game boards, a single swap may not be able to eliminate any number, so the player needs to perform multiple swaps to eliminate numbers.
- The eliminated numbers will leave **empty cells** at their original positions. As a result, the numbers above the empty cells will **fall down** to fill the empty cells. After

that, new random numbers in { 1, 2, …, K } are generated to fill the empty cells from the top to fill the game board, until there are no more empty cells.

- After existing numbers fall down and new numbers are generated, more numbers may be eliminated. Your program should **iteratively eliminate numbers and generate new ones UNTIL no numbers can be eliminated**. Then, a swap step is said to be done, and the next swap can be performed on the resulting game board.

Hint: your AI should consider this feature to get more scores in each number swap step.

2.3 How to calculate the score?

Every time when numbers are eliminated, the player gets the corresponding scores. The total score that a player receives **for each group of same-number elimination** is

(the number of "same numbers" forming a row/column)^2.

Figure 3 below shows four example groups of to-be-eliminated numbers and the associated scores. A number group may span more than one row(s) and one column(s); you should first identify consecutive same-number rows and columns and then calculate the associated score for each row/column; see Figure 3 for examples.

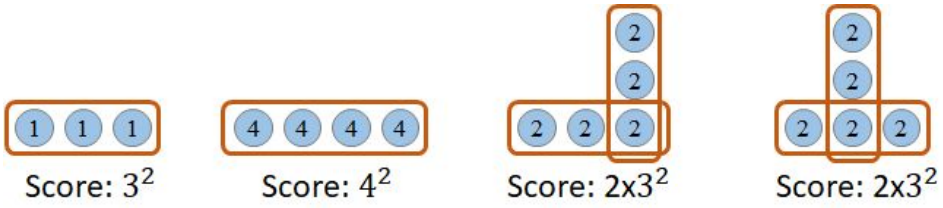

Figure 3. Numbers that form contingent row(s)/column(s) with the corresponding scores.

2.4 End of game

The game ends after you finish 100 "number swap" operations. Your goal is to gain the highest possible score after the 100 steps.

3. Implementation

Below, we describe some key functions as guidelines for you. Note that we **do not** use CodeSubmit for the project, but you should still follow certain formats or requirements for consistent grading.

3.1 Encoding the game board

We store the game board using a **one-dimensional integer array of length 100.** The location of each cell in the board is represented using an integer in [00, 99]. For example, "47" means the location at the "fifth" row and the "eighth" column.

- The upper-left corner of the board is 00.
- The cell on its right is 01.
- The cell just under 01 is 11.

See example cell coordinates in Figure 4 below.

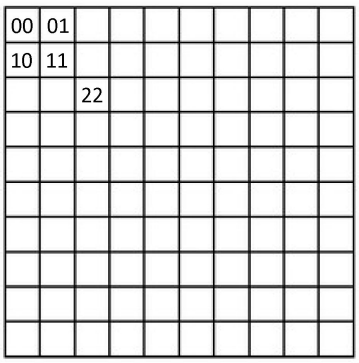

Figure 4. Encoding of the locations on the game board.

3.2 Print the board and scores

Write a function that takes a 1-D array (game board) as its input and displays the user interface, i.e., to print the game board on the terminal/command prompt. Your program must display the game board after each swap operation and print how many numbers are eliminated and the obtained score. See Figure 6 for a sample screenshot.

```
############## STEP 27 ###############
Total score: 201
    0 1 2 3 4 5 6 7 8 9
   --------------------
0 | 2 1 3 1 2 2 3 2 4 2
1 | 1 3 1 2 4 3 1 4 1 3
2 | 2 3 3 4 1 2 4 1 3 2
3 | 2 4 4 1 4 2 1 2 4 3
4 | 4 2 4 3 3 1 2 3 2 1
5 | 2 4 1 3 4 4 1 1 3 2
6 | 2 3 1 2 3 4 2 4 1 3
7 | 4 2 4 3 1 2 2 1 2 3
8 | 1 3 1 2 4 1 1 2 4 1
9 | 3 3 4 4 2 1 4 1 1 3

Please enter swapping locations:
Numbers at 78 and 77 are swapped.
3 numbers are eliminated!
3 numbers are eliminated!
3 numbers are eliminated!
3 numbers are eliminated!
Score obtained in this step: 36
```

Figure 5. An example of printing the game board, in which there are four rounds of elimination after a swap.

3.3 Reading a swap operation from the user

For interactive gameplay, write a function to ask the user for the next swap operation. We use a four-digit integer to represent a swap, i.e., the first two and last two digits represent the locations of the two numbers to be swapped. For example, 2122 means that we swap the number at 3rd row 2ed column with the number at 3rd row 3rd column.

3.4 Eliminating numbers

After each swap operation, your program should locate the numbers to be eliminated, as defined earlier in Section 2.2, and compute the related score.

3.5 Numbers falling down and generating new numbers

After the number elimination, your program should move the remaining numbers down to fill the empty (eliminated) cells, and generate new ones to fill these cells from the top.

3.6 A valid move?

Though we assume that the inputs from users are always four-digit numbers, the inputs may not be correct. The provided "two swap locations" are allowed not to eliminate any numbers, but they **must be adjacent horizontally or vertically**.

3.7 The program flow

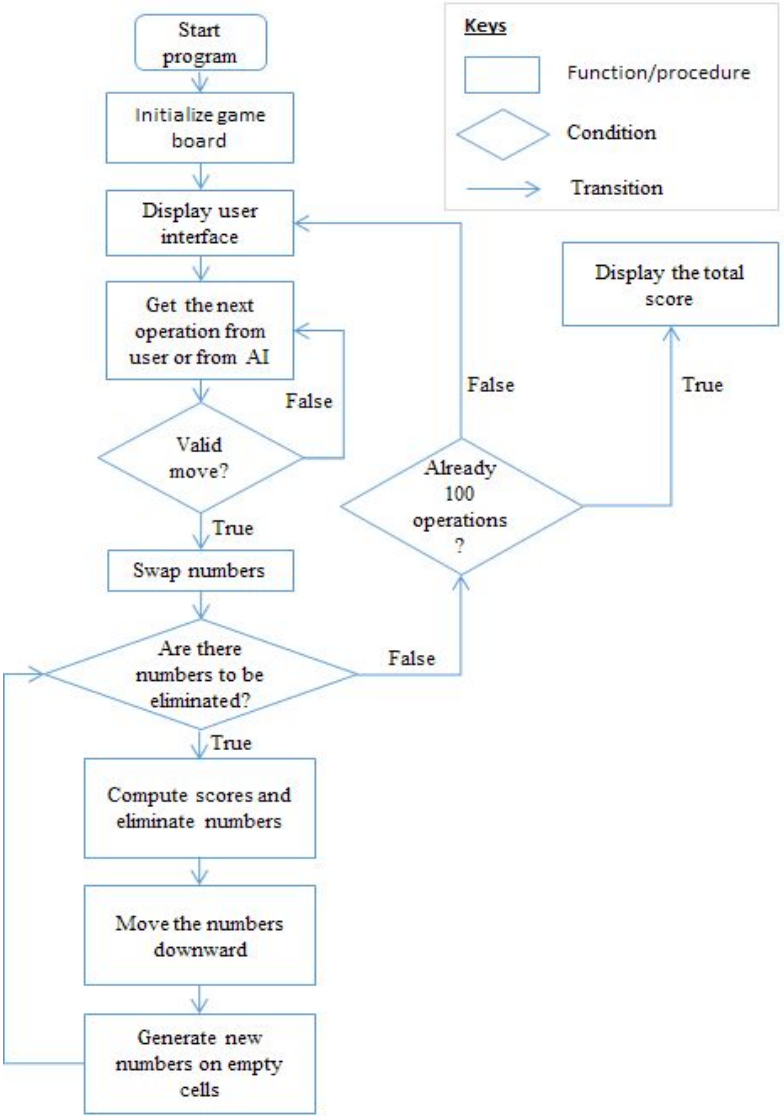

Figure 6. The program flow.

Note: you may create additional functions in your code to make your program more modular and easier to debug. In this way, you can learn and try the **divide-and-conquer concept** we discussed in class.

Figure 7 on next page shows two example (K=4 and K=6) swap operations to illustrate the game flow: (column a) an initial game board; (column b) the game board JUST after the two picked numbers are swapped; (column c) we highlight the same-number rows

and columns to be eliminated; and (column d) after existing numbers move down and new numbers are generated to fill the empty space. Note the bottom of (d), after moving numbers downward and generating new numbers, more numbers could be eliminated, so your program needs to be able to iteratively check for more number elimination, etc.

Figure 7. Two example gameplays (top row and bottom row) for a single swap operation. We use green background to highlight newly generated numbers.

Note: you MUST eliminate all same-number rows & columns (different rows/columns could have different numbers) in a game board BEFORE moving the remaining numbers down to fill the empty cells and randomly generating new numbers from the top.

4. Implement Your Computer Player

After you finish the basic program, which supports human gameplay, you will come to the most challenging and exciting part of the project, where we replace the function in "*Section 3.3 Reading a swap operation from the user*" **by an AI player function**.

In each round, your AI player will be given the current game board and the value of K (see Section 2.1 for the meaning of K). Given this information, your AI player should output a valid (and the best possible) swap operation for the current game board.

To implement a computer player, you mainly need to finish one function. Its prototype is defined below (**You MUST use your own student ID** as "*studentID*" below):

```
int ai_player_studentID( int K , const int * board );
```

This function takes the following two inputs:

- ***int K***: It is an integer **within set {4, 5, 6, 7}**. This means that we will test how "smart" your AI player is by playing multiple games under different Ks. In general, the difficulty of a game increases for a larger K.
- ***const int * board***: It is a one-dimensional array (*const* means constant, i.e., your function should not modify the contents of the input array) that represents the numbers in the game board, same at what is defined in Section 3.2.

The output has the following format:

- **The return value (int)**: a four-digit integer that represents a swap; see Section "3.3 Reading a swapping operation from the user" for its meaning.

5. Requirements

5.1 Requirements of the "Basic part"

Your program starts by repeatedly reading four-digit integers (e.g., 1188) from keyboard input as successive moves until the game ends. Though we do not use codeSubmit, you **MUST follow** the requirements below for consistent and fair grading:

- To ensure that the programs of all students produce the **same game board** after the **same series of swaps**, you must:
  - Use the game board shown in Figure 2 of the main project specification as the initial game board. To do this you need to manually type each number by hand.
  - Put the following code **at the beginning of** your main() function, such that the pseudo random number generator can be correctly initialized:
    - uint32_t pesudo_seed = 0;
    - tinymt32_init(&state, pesudo_seed);
  - Also, you need to ensure both "tinymt32.h" and "tinymt32.c" files (provided by TAs) are **in the same folder of the main program**.
  - Lastly, in the file where your program produces a random number, you need to make sure the following things:
    - Include the file "tinymt32.h"
    - Set up a global variable "state" as follows:
      - tinymt32_t state;
    - Produce random number using the following code segment:
      - tinymt32_generate_uint32(&state) % 4 + 1
  - Once there are numbers being eliminated, you need to move all remaining numbers downward first. Then, all empty cells are on top of the game board. Then, you **must** fill the empty cells row by row from top to down and from left to right within each row. Your code of generating new numbers should look like the code in the following image.

```
// candies falls down and to fill the empty cells below
fall_candies(board)

// to generate new random numbers
for (int i = 0; i < 100; i++)
    if (board[i] == 0) // an empty cell
        board[i] = tinymt32_generate_uint32(&state) % 4 + 1;
```

- Your program must read inputs from the keyboard (see examples shown in Section 3.3), so that we can test your code consistently, i.e., we will input a **sequence of four-digit integers** and see if your program generates the **expected result**. We've provided you with two sample test files on the course webpage (Blackboard) for you to download and test your human player gameplay. A typical case is like this:

  2232
  1213
  ……

  Note that we will try some invalid moves and see if your program can find them.
- Your program must print the results to the terminal / command prompt.
- During each round of the game, your program must report:
  - the current status of the game board;
  - the scores obtained in this round; and

- for illegal swap, print a warning message, and ask the user to input again.
- At the end of the game, you must report the total score that the player obtained, and show the final game board.

5.2 Requirements of the "AI part"

- You must follow the **function prototype** defined in Section "4. Computer Player" to implement your AI player.
- It should not produce **valid moves**, and it should not modify the input game board.
- Your "submitted" AI function shouldn't contain any **print statement**; otherwise, the system may wrongly judge your results. Note, you may have print statements when you debug your code but remember to comment them out before submission.
- For a fair comparison, your AI function must produce an output within a reasonable amount of time, i.e., **within 5 seconds on a desktop computer**.
- You need to carefully test your code to make sure your code won't crash, e.g., the "**array out of bounds**" error!!!
- **No "main() function"** in your submitted files for the AI player part.

Please note that if your program does not follow any one of the above requirements, you may get **zero points** for the whole project.

6. Submission

We have created two submission boxes on Blackboard:

- one for you to submit the **Basic program** that supports human player and
- another one for you to submit your code **ONLY for the Computer player**.

6.1 Submission Format for Basic Part

Your submission for the basic program should be a single **zip file** named as

basic_<studentID>.zip (e.g., basic_1155123456.zip)

which **contains and should only contain** the **.h** and **.c** files in your project. The TA will download your submission files from Blackboard and compile them during the project demo to form an executable game using gcc, CodeBlocks, or Visual Studio for testing.

6.2 Submission Format for AI Player (AI function only)

Your submission for the computer player should contain **only two files**:

- aiplayer_<studentID>.h (e.g., aiplayer_1155123456.h)

  In this file, you **must** define the function elaborated in Section 4.

- aiplayer_<studentID>.c (e.g., aiplayer_1155123456.c)

where <studentID> is your own student ID. In this file, you may have more functions for your main AI function to call.

In the submitted zip file for the AI player, you need to include the above .h file and all the necessary functions inside the above .c file for your AI player to work. To implement a stronger computer player, you may define additional functions in your .c file. However, you **must** follow the above naming convention (i.e. add your own "*_studentID*" at the end of all your function names), so that your TA can take your .c file and **compile it with the TA's code without** having any **function name collision**. We may deduct your project marks if you fail to follow this convention. Also, use .c rather than .cpp or .C as the extension of your source code file.

Submission Note:

- After preparing your submission file in the above format, you need to upload one zip file (see above) to **each of the two submission boxes** on Blackboard. Please see the project page on our course webpage in **Blackboard** for the details.
- You may submit multiple times for each submission box, and we only take the last submission before the deadline as your submission for grading.
- Note that the deadline is ***STRICT*** and the project is a **BIG COMPONENT** in the overall grading scheme of this course. Please start early and prepare early.

7. Grading

This project has two parts for grading.

- The basic "human player" part takes up **16% of the whole course**.
- The **AI part takes up 4% plus extra bonus points**. The bonus points are judged based on your AI player's performance relative to the scores of your classmates.

Part 1: the basic program (16%)

During the project demo (on the demo day during the last lecture period), the TAs will compile your program, and then test your program in the following ways:

(i) interactively try your program to play games; and

(ii) pipe some sample text files as inputs to your game executable and see if the results are correct, as expected (you will learn the meaning of "pipe" in class later).

Part 2: the AI player (4% + bonus)

The evaluation of your AI player contains two sub-parts:

- **[Part 2.1, taking up 4%] The basic performance of your AI player**.
  This includes (i) if it works well without program crash and (ii) if there is a swap in the game board that can lead to number elimination, your AI program should find it.
- **[Part 2.2, up to 3 extra bonus points] How smart your computer player is**.
  We will create a game environment, run it with the AI player of each student, and record how much score your AI player can obtain in each game:
  - We will consider four different values of K, i.e., {4, 5, 6, 7}.
  - For each K, we will initialize a random game board at the beginning (not using the one that shows in Figure 2), run four games using your AI player with 100 swaps in each game, and take the two highest scores out of the four games.
  - The final score for your AI player is the sum of the scores of the eight highest scores (two per each K).

After finishing all the games, we will rank all your AI players based on the final scores.

Then, the top 15 students in class will obtain **extra bonus points** (**beyond the basic scores in the course**) based on his/her ranking; see the table below.

Ranking	**Bonus Credit (of the whole course)**
1	3.0 bonus pts
2-3	2.2 bonus pts
4-7	1.4 bonus pts
8-15	0.6 bonus pts

8. Project Schedule

Week	**Date**	**Tasks Expected to be done**
9	Nov. 4	The project will be released on Blackboard course homepage (and will be discussed in class on Nov 4) Start to write the basic game: ● Function: Initialize game board ● Function: Display user interface ● Start to think over – how to find numbers to be eliminated? ● Further think over – how to move the numbers down? Etc. At home: implement the rest of the game for human player..
11	Nov. 18	● Around (or before) this date, you should have completed the basic game without the computer player, so that you can focus on your computer player and make it stronger using the remaining time.
12-13	Before Submission Deadline	Finish the basic program, finish a working computer player (valid moves ONLY) & perfect your computer player (with better strategies).
13	Pre-test Nov 28 11:59pm	● If you submit your computer player before this pre-deadline, we will try your code with the TA's system and will let you know your results and also whether your code has any problem.

		• This pre-tournament test is volunteer-based, and has no contributions to the final score. However, it can allow you to know your program's capability and whether it has any compilation problem, so that you can work out a better program before the deadline.
13	Dec 1 11:59pm	**Deadline** of submitting (part 1) the basic program & (part 2) the computer player.
13	Dec. 2 3:15pm-6:15pm	**DEMO DAY:** 1) TA will evaluate the basic program of all the students one by one in the lab or through zoom, depending on whether we can meet face to face. 2) TA will test your AI-Player.

9. Project Extension

Due to COVID-19 this year, we may not be able to have face-to-face final exams. If it happens, the final exam component in the course assessment will have to be removed; see "00. Course Outline and Syllabus.pdf" on Blackboard, and we will have the following two changes to the project:

(1) The total project weight in this course will increase from 20% to 25%, as mentioned before; and

(2) On the other hand, we will extend the project with some extra requirements that will contribute 10% of the total course marks. In that case, we will release the "project extension" specification in early Dec. for you to work on the project extension during the winter break. The deadline of the project extension will be arranged around early January.

Please take note of these and complete at least the "basic part" of the project.