Electrical & Computer Engineering Department, University of Maryland, College Park

Fall 2019 **ENEE140** Dr. Gang Qu

Project 3: A Simple Monopoly Game

Posted: Monday, November 4, 2019
Due: 11:59 pm Monday, November 25, 2019

Project Objective:

1. Master the use of command line argument and file I/O.
2. Practice to use and manipulate two dimensional arrays.
3. Learn how to generate random numbers and symbols in C.
4. Learn how to develop an interactive game with system time.
5. Improve skills in problem solving and efficient programming.

Project Description:

In this project, you will design and implement a simplified version of *Monopoly*, a two-player game between a user and the computer. You will have a 20x20 board containing a closed path. Properties are located along this path that you may purchase and upgrade. If you land on the other player's property, you have to pay rent. There are also locations on the path, called Chance, which if you land on you must follow certain instruction. The goal of the game is to earn as much money as possible by investing your starting money. The game ends when one player runs out of cash and goes bankrupt.

General Requirements:

1) Your program should run with the following command line arguments:

```
a.out path_length path_file property_file chance_file
```

   Here, `a.out` is the compiled executable; `path_length` is the length of the path on the board; the others are names of the input files that will be described later. These file names can be any legal file names in the same directory as `a.out`.

   You should check that the user has input the command line arguments and files correctly.

   a. If the number of arguments is incorrect, print the following error message and exit:

```
Incorrect number of arguments provided.
Usage: a.out path_length path_file property_file chance_file
```

   b. If any of the files cannot be opened, print the following error message and exit:

```
XXX.txt could not be opened properly.
```

   Here XXX.txt is the name of the file that caused an error. If multiple files cannot be opened properly, printing the error message for any one of them is sufficient.

2) You will need to design data structures to store the board path, property, and Chance information. You may also need to give these data structures appropriate initial values.

3) Turn Structure: On every turn, you must:
   a. Print the turn number (starting at 1), whose turn it is (P1: user; P2: computer), and their balance
   b. Print the board
   c. Have the active player (the one who has the turn) roll the dice, and print the result
   d. Move the active player along the path according to the value rolled on the dice, and print the coordinates of the space they landed on in `(y x)` format
   e. Give the active player $200 if they pass or land on GO
   f. Print the property information of the space that the active player lands on a property
   g. Have the active player purchase or upgrade the property they land on, if the property is unowned or is owned by them (P2 will automatically purchase/upgrade the property)
   h. Charge them for rent, if the property they land on is owned by their opponent
   i. Draw a Chance card and execute it if the space they land on is a Chance space
   j. If the player is P1, the user, prompt them to end their turn.

4) Your code should be documented properly. This includes using proper style, naming variables in a logical way, and including comments (especially about program flow and any part of your code that isn't intuitively obvious).

Detailed Project Information:

1) Input files:

   `path_file`: each line of this file contains two sets of coordinates specifying a step in the path, the next space after `(y1 x1)` is `(y2 x2)`, in the following format:

```
(y1 x1) -> (y2 x2)
```

   `(y1 x1)` on the first line will always be (0 0), the starting point of the path. Each line's `(y2 x2)` will be the `(y1 x1)` for the next line so the path will be connected. `(y2 x2)` on the last line will also be (0 0) to close the path.

   `property_file`: each line of this file contains a set of coordinates followed by exactly 6 numbers representing the sale prices and rental costs, in the following format:

```
(y x) p1 p2 p3 | r1 r2 r3
```

   `(y x)` are the coordinates at which the specified property is located. `p1`, `p2`, and `p3` are the prices to purchase and upgrade the property to stages 1, 2, and 3, respectively; while `r1`, `r2`, and `r3` are the rental costs for the property at those upgrade stages.

   `chance_file`: each line of this file contains the coordinates of a space, `(y x),` where a random Chance card action should be selected and performed when landed on. The first line in this file is always JAIL, the space to which players are sent when GO TO JAIL is picked as the Chance action.

2) Game Board: The Monopoly board should be a 20x20 grid, displaying path information and player location. The following is an example of a game board:

```
   0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9
0  >  >  >A v
1  ^        >  >  >  >  >  >  >  >  >  >  >  >  >  >  >  v
2  ^                                                     v
3  ^  <                                                  v
4     ^                                                  v
5     ^                                                  v
6     ^                                                  v
7     ^                                                  v
8     ^                                                  v
9     ^                                                  v
0     ^                                                  v
1     ^                                                  v
2     ^                                                  v
3     ^                                                  v
4     ^                                                  v
5     ^                                                  v
6     ^                                                  v
7     ^                                                  v
8     ^  <  <  <  <  <  <  <  <B <  <  <  <  <  <  <  <  <
9
```

Axis Labels: The first/top row in the game board are the column indices. There are 3 spaces before the first 0, and 2 spaces between each subsequent index. Starting from the second row, the first space in each row is the index for that row.

Path Information: A path starts and loops back to (0 0). Each space in the path contains a character pointing to the direction of the next space in the path: > right, < left, ^ up, and v down. For example, there is a 'v' at (0 3), indicating that the path will go down to the next space (1 3). The direction characters are aligned in the same columns as the column indices.

Player Position: The position of the two players, user and computer, are represented by A and B, respectively, on the game board. 'A' or 'B' should be displayed to the immediate right of the path direction character. If both players land on the same space, use 'E' instead.

3) Game Rules:
   i. On every board, (0 0) corresponds to the GO space on a traditional monopoly board. This is where all players start. When players pass GO, they collect $200.
   ii. Each player begins the game with a balance of $1500.
   iii. Player 1, the user, goes first; followed by Player 2, the computer. Then they take turns.
   iv. If a player has $0 or less at the end of a turn, the game ends and the opponent wins. The program should then print who the winner is and exit. For example,

```
Player 2 has gone bankrupt. Player 1 wins!
```

   v. Property purchase and upgrade. If Player 2 lands on a property that is not owned by any player, the computer will purchase the property if she has enough money. If the property is owned by the computer, she will upgrade the property if she has enough money. For Player 1, the user will make the decision on purchasing/upgrading the property.

vi. Rent. When a player lands on a space owned by the other player, he/she must pay rent. If his/her does not have enough money to pay the rent, he/she goes bankrupt and game over.

vii. Property Information. If there is a property located on a space that the player lands on, the following information must be printed below the board:
Coordinates of the space in `(y x)`; owner (`P1`, `P2`, or `None`); stage of the property (`1, 2, 3)`; upgrade price (`N/A` if already upgraded to stage 3); and rental cost. For example,

```
Space    Owner   Upgrade Stage   Upgrade Cost    Rental Cost
(5 8)    P2      2               $1500           $300
```

viii. Chance. When a player lands on a Chance space, a Chance card will be drawn randomly from the following options:
Receive money; Lose money; Lose a turn; Receive an extra turn; Move forward; Move backward; Go to JAIL.

You need to print a message describing the cause for the action before executing the action. In classic Monopoly an example of option (i) is "Holiday fund matures. Receive $100." You only need one cause message per option, and it is totally up to you. Be creative! Your TAs like to be entertained while grading. However, be sure to keep the content appropriate and respectful.

For the options of receiving/losing money, you decide the amount. It can be a fixed amount, a random amount, or something related to the player's current balance (for example, give half of your money to your opponent). For the options of moving forward/backward, a random number between 1 and 3 should be selected.

ix. JAIL. When a player is sent to JAIL, he/she goes directly without passing GO (and will not collect the $200). You must print out the following message

```
Player X is visiting JAIL!
```

The player in JAIL still collects rent if the other player lands on his/her property. He/she will not lose his/her turn either. When he/she rolls doubles (that is, the two dice have the same amount such as two 3's, two 5's, etc), he/she gets out of JAIL and advance an amount of spaces equal to the sum of the two dice.

4) **Bonus** (5 points, no partial credits). In the path input file, each line represents one step in the path. If there is a straight line with multiple steps, for example from position (5 8) to (5 13), we can represent this by one line `(5 8) -> (5 13)`, instead of 5 lines. If your code can handle such input format correctly, you will receive the 5 points for bonus.

Special Notes:

- (0 0) (aka GO) will never have a property on it and will never be a Chance space.
- No space will have both property and Chance information.
- You may assume that the files given to you are in the correct format.

- In both the regular and bonus input path files, all path segments will be straight horizontal or vertical lines. You will never encounter syntax like `(7 2) -> (11 5)`. The coordinates are also numbers between 0 and 19.
- There is a sample program template at the end of this document. You do not have to use it for your project, but it may give you some hints on how to complete the project.
- A sample executable and some input files will be uploaded to the public folder on GLUE.

Project Requirements:

1. You must program using C under GLUE UNIX system and name your program **p3.c**.
2. Your program must be properly documented.
3. submit the project (**p3.c only**) by (replace the **? in 010?** by your own section number):
   submit 2019 fall enee 140 010? 3 p3.c

Grading Criteria:

Correctness:	90%
Good coding style and documentation:	10%
Bonus:	+5%

Late submission penalty: -40% for the first 24 hours
No submission will be accepted after the first 24 hours.

Program that does not compile under GLUE UNIX:	-100%
Wrong file name (other than p3.c):	-100%

```
----------------------------------------------------------------- -
Below is a programming template/outline just for your reference (also
posted in the GLUE public folder). It by no means is a complete
program or defines all the necessary variables. You do not have to
follow it.

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 20
#define START_BALANCE 1500
#define GO_RATE 200

int main(int argc, char *argv[]) {
  int path_length; // length of the path

  int p1_balance = START_BALANCE, p2_balance = START_BALANCE;

  int turn; // turn number

  int die1, die2; // values for the two dice
  int y, x; // loop variables, you might need more

  if (argc != 5) {
    // print error message and use exit(0)
  }

  path_length = atoi(argv[1]);
  int player_[path_length]; // this array's size depends on the path
                            // length, so it must be initialized here

  FILE *path_file, *property_file, *chance_file; // 3 input files

  // open the files and do safety checks

  // define data structures for reading files
  char path_board[N][N+1];  // need the '\0' for each row in the board
                            // store the direction character < > v ^

  int step[path_length][2]; // the coordinate pair for each space along
                            // the path.

  int property_data[path_length][8]; // store property information

  int chance_data[path_length];      // store chance information

  // read in the data files into the data structures you have defined

  while (p1_balance > 0 && p2_balance > 0) // while game not over
  {   // 1. roll the die twice to get die1 and die2
      // 2. move the current player along the path by die1+die2 steps
      //    (note the possible exception when the player is in JAIL)
      // 3. make the corresponding actions based on the landing position
      //    and who is the player (user and computer will be treated
      //    differently in certain occasions such as purchasing property)
      // This phase is one of the most difficult parts of the project.
      // Many things (such as balance of both players, ownership and
      // stage of the property, etc.) need to be updated in this phase.
      // Read the project requirements carefully and make plan before
      //    start writing the code.
      // 4. Update the turn information (note that there are cases that
      //    the current player will keep the turn)
  }

  // check and print the winner

  return 0;
}
```