ECE 651-Homework 2: Sally's Stash

For this assignment you will be creating the different versions of a children's game designed to encourage logic, strategy, and critical thinking as well as teach them the difference between rows and columns. The game, Sally's Stash, has two players hiding different colored stacks of gold for the squirrel and then each tries to find the stacks the other has hidden. Interestingly, it is believed that real tree squirrels hide their nuts by size, type, nutritional value and taste using a spatial chunking technique so they can find them later.

Before we delve into the Sally's Stash game requirements, the purpose of this assignment is about dealing with changing requirements. Although this won't happen in the real world or our projects moving forward, we will give you the requirements for the first **two** versions so you can look and plan ahead. Therefore, as you are designing Version 1, you can think about how you can accommodate changes to requirements in general, but especially for Version 2 that is ahead.

Version 1 Requirements

Sally is a special squirrel who, in addition to burying acorns for the winter, likes to bury gold. In this game, two players will hide gold stacks that come in different sizes in a grid-like layout as shown below. As a result, gold can be "buried" using the coordinates in the grid (e.g., [B1-B3], or [D5-G5]) For this version, you only need a text-based grid (Note: extra credit will include a graphical user interface).

Sally's stashes come in stacks specific sizes and colors. Both players (player A and player B) have the same stash to hide at the beginning of the game. Each player is using the same device to play (See extra credit for implementing an extra credit networked version).

0	1	2	3	4	5	6	7	8	9
A									
B	P	P	P						
C								G	
D					R			G	
E					R			G	
F					R			G	
G				B	R			G	
H				B				G	
I									
J	G								
K	G	R						B	
L	G	R						B	
M	G	R							
N	G	R							
O	G				P	P	P		
P									
Q					R				
R					R				P
S					R				P
T					R				P

Figure 1. Potential configuration of stacks on either player's "board". The color in the figure is just to demonstrate and is not necessary in Version 1.

2	Green stacks that are	1x2
3	Purple stacks that are	1x3
3	Red stacks that are	1x4
2	Blue stacks that are	1x6

Figure 2. Stacks available to each player

GAMEPLAY I: Placing Stacks

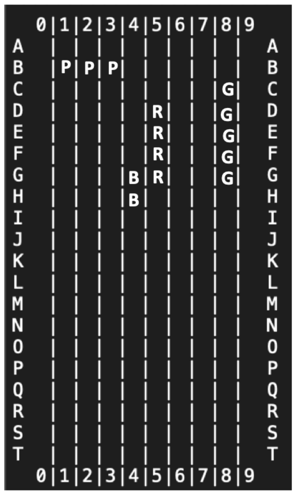
- The first phase of the game asks the players to place their stashes.
- The game should display a blank "board" (i.e., the just grid)
- Player A places his stacks first, with the following instructions:

  Player A, you are going place Sally's stash on the board. Make sure Player B isn't looking! For each stack, type the coordinate of the upper left side of the stash, followed by either H (for horizontal) or V (for vertical). For example, M4H would place a stack horizontally starting at M4 and going to the right. You have

2	Green stacks that are	1x2
3	Purple stacks that are	1x3
3	Red stacks that are	1x4
2	Blue stacks that are	1x6

- Next, the player should be prompted to place his/her first stack by displaying:

  Player A, where do you want to place the first Green stack?
- If the location is invalid (collides with another stack, results in a stack going off the grid, etc.), you should explain the problem to player, and ask them to place again. You should ignore case (i.e., M4H m4h m4H M4h are all the same).
- If placement is successful, you should display the board with the stack on it. You will only need to display a G, P, R, B for the stack color respectively (i.e., no colors are necessary). And the number of letters (Gs, Ps, Rs, or Bs) should match the dimensions of the stack.
  - For example, your board would look as shown below, after a Green stack at C8 (player code: C8V), Purple stack at B1 (player code: B1H), Red stack at D5 (player code: D5V), and a Blue stack at G4 (player code: G4V)



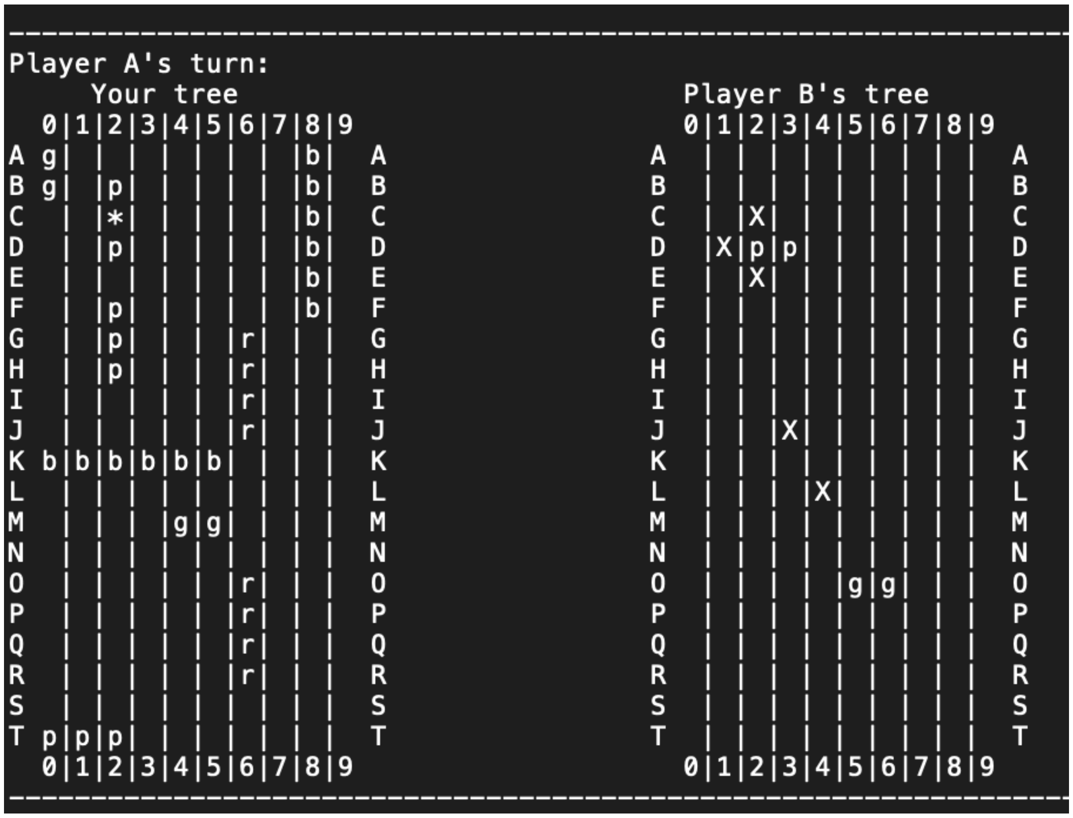
```
  0|1|2|3|4|5|6|7|8|9
A  | | | | | | | | |  A
B  |P|P|P| | | | | |  B
C  | | | | | | | |G|  C
D  | | | | |R| | |G|  D
E  | | | | |R| | |G|  E
F  | | | | |R| | |G|  F
G  | | | |B|R| | |G|  G
H  | | | |B| | | | |  H
I  | | | | | | | | |  I
J  | | | | | | | | |  J
K  | | | | | | | | |  K
L  | | | | | | | | |  L
M  | | | | | | | | |  M
N  | | | | | | | | |  N
O  | | | | | | | | |  O
P  | | | | | | | | |  P
Q  | | | | | | | | |  Q
R  | | | | | | | | |  R
S  | | | | | | | | |  S
T  | | | | | | | | |  T
  0|1|2|3|4|5|6|7|8|9
```

- Once a stack is placed, and the updated board is displayed, the Player should be prompted for the next stack. Repeat this until all stacks are placed for Player A
- Once all stacks have been buried for Player A, repeat the process for Player B beginning with the instructions (e.g., **Player B, you are going place Sally's stash on the board...)**
- Note that player A's stacks and player B's stacks are in different grounds close to Sally's trees, so player B will start with a blank board, and can place her stacks anywhere (*i.e.*, even though Player A has a stack at A0, player B can place a stack at A0 also, since they are "different" A0s).

GAMEPLAY II: Play begins

- Player A will begin the game with a first guess. Then, they will alternate turns between each player.
- First, display the board to player A. You will display two things:

- Player A's stacks
- Player A’s hits/misses.
  - In this game, “hit” means that Player A has guessed the location of a part of one of Player B’s stacks. Hits are denoted with *
  - In this game “miss” means that Player A has guessed a location, but there was no portion of the stack found. Misses are denoted with X



```
-----------------------------------------------------------------
Player A's turn:
     Your tree                           Player B's tree
  0|1|2|3|4|5|6|7|8|9                    0|1|2|3|4|5|6|7|8|9
A g| | | | | | | |b|  A                A  | | | | | | | | |  A
B g| |p| | | | | |b|  B                B  | | | | | | | | |  B
C  | |*| | | | | |b|  C                C  | |X| | | | | | |  C
D  | |p| | | | | |b|  D                D  |X|p|p| | | | | |  D
E  | | | | | | | |b|  E                E  | |X| | | | | | |  E
F  | |p| | | | | |b|  F                F  | | | | | | | | |  F
G  | |p| | | |r| | |  G                G  | | | | | | | | |  G
H  | |p| | | |r| | |  H                H  | | | | | | | | |  H
I  | | | | | |r| | |  I                I  | | | | | | | | |  I
J  | | | | | |r| | |  J                J  | | |X| | | | | |  J
K b|b|b|b|b|b| | | |  K                K  | | | | | | | | |  K
L  | | | | | | | | |  L                L  | | | |X| | | | |  L
M  | | | |g|g| | | |  M                M  | | | | | | | | |  M
N  | | | | | | | | |  N                N  | | | | | | | | |  N
O  | | | | | |r| | |  O                O  | | | | |g|g| | |  O
P  | | | | | |r| | |  P                P  | | | | | | | | |  P
Q  | | | | | |r| | |  Q                Q  | | | | | | | | |  Q
R  | | | | | |r| | |  R                R  | | | | | | | | |  R
S  | | | | | | | | |  S                S  | | | | | | | | |  S
T p|p|p| | | | | | |  T                T  | | | | | | | | |  T
  0|1|2|3|4|5|6|7|8|9                    0|1|2|3|4|5|6|7|8|9
-----------------------------------------------------------------
```

In the above example, player A has successfully hit player B's stacks at four locations:

O5: a green stack
O6: a green stack
D2: a purple stack
D3: a purple stack

and missed player B's stacks at D1, C2, E2, J3, and L4 (meaning no stacks are located there)

Player A's own stacks have been hit once, at C2.

- Player A is then prompted for the coordinate to look for buried gold.
- If the coordinates are invalid, the game should prompt player A to enter a valid choice.
- The game will then report the result, such as

```
--------------------------------------------------------------------------------
You found a stack!
--------------------------------------------------------------------------------
```

or

```
--------------------------------------------------------------------------------
You missed!
--------------------------------------------------------------------------------
```

- Play then proceeds with Player B's turn. The boards are displayed for Player B
- Player B's state is shown on the left with all of B's stacks known, and Player A's tree is shown on the right with only hits/misses known).
- Play continues to alternate until one player has no more stacks that can be found.
- When one player has no more stacks, the other player wins.
- Your game should print a message stating who won, and then exit.

Before proceeding to Version 2, please check that

- You have met all requirements for Version 1
- You have rigorously and thoroughly tested your code
- You have commented your code well
- **You have made a git commit which contains your version 1 code (i.e. "delivered" version 1 to your client)**

```
=================
|  Version 2    |
=================
```

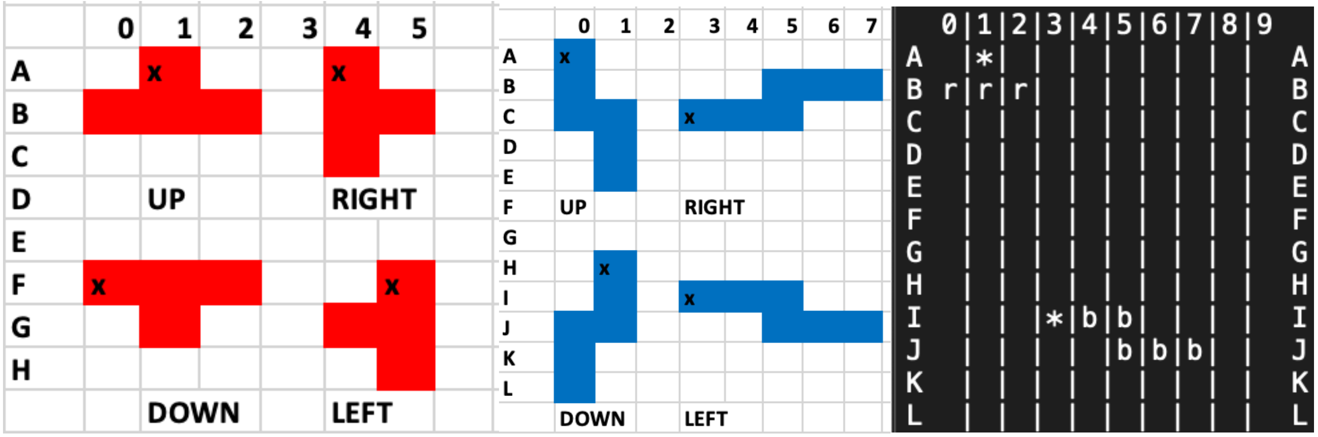
After completing version 1 of Sally's Stack, your customer changes the requirements slightly, and you now need to make Version 2. There are three changes to the requirements: changes to stack design, changes to moves, and a mode to play against the computer.

(1) Changes to stack design.

To spice the game up a bit, the customer decided that there should be some different stack shapes that are not just rectangular. specifically, each player now has the following stack types:

2	Green stacks that are	1x2
3	Purple stacks that are	1x3
3	Red "Superstacks" that can be shaped as shown below	
3	Blue "Crazystacks" that can be shaped as shown below	

The orientations for Green and Purple stacks are unchanged – Horizontal (H), Vertical (V). The Red Superstacks and Blue Crazystacks have four different orientations – Up (U), Right (R), Down (D), and left (L) as shown in Figure 4. The starting coordinate for the Superstacks and Crazystacks will be as indicated in the figure marked with an "x". In other words, a Superstack starting at A1 that's up would be A1U, and a Crazystack starting at I3 that's Left, would be I3L.



```
   0|1|2|3|4|5|6|7|8|9
A   |*| | | | | | | |   A
B  r|r|r| | | | | | |   B
C   | | | | | | | | |   C
D   | | | | | | | | |   D
E   | | | | | | | | |   E
F   | | | | | | | | |   F
G   | | | | | | | | |   G
H   | | | | | | | | |   H
I   | | |*|b|b| | | |   I
J   | | | | |b|b|b| |   J
K   | | | | | | | | |   K
L   | | | | | | | | |   L
```

Figure 4. Visual demonstration of Superstacks (Left) and Crazystacks (Middle). Note that in your version, this would all be text based as shown in the figure on the right.

(2) Changes to moves.

- A player now has 2 new actions: moving (M) a stack and sonar (S) scanning a stack
- Each new move is limited to 3 uses per game.
- At the start of player's turn, the game should display the board, prompt for which action type the player wants to use, and then prompt for any information needed for that action type. For example:

```
--------------------------------------------------------------------------------
Possible actions for Player A:

 D Dig in a square
 M Move a stack to another square (2 remaining)
 S Sonar scan (1 remaining)

Player A, what would you like to do?
--------------------------------------------------------------------------------
```

- Once a player is out of their 3 uses of the new moves you MAY (but do not have to) omit that item from the menu entirely.

(2a) Move stack. (M)

- When using this move type, a player selects one of his/her own stacks (the game should prompt for which stack, and the player should be able to type any coordinate which is a part of the stack they want). The player then is prompted for a new placement (location + orientation). The stack is then moved to that position.
- Any parts of the stack that have already been found should remain the same position(s) of the stack.
- If the player selects an invalid location, the player is re-prompted for their actions. They can either choose to move to a valid location, choose to dig, or choose to sonar scan
- The other player's display does NOT gain any information that the stack moved. If they previously found a part of that stack, the hit markers remain. If they previously missed the new location, those markers also remain.

(2b) Sonar scan. (M)

- When the player selects this move type, the game prompts for the center coordinates of a sonar scan.

Any coordinate on the game board is valid, even if part of the scan will go off the edges of the board. The game then considers the following pattern around (and including) the center (C) and reports on the number of squares occupied by each type of stack in that region.

```
   *
  ***
 *****
***C***
 *****
  ***
   *
```

- For example, after a scan, the game would display:

```
--------------------------------------------------------------------------------
Green stacks occupy 2 squares
Purple stacks occupy 0 squares
Red stacks occupy 5 squares
Blue stacks occupy 1 square
```

- No information about the exact position of any stack in that region is report. There is also no information about how many of each stack---only how many squares are occupied. In other words, we do not know if 2 squares occupied by a green stack is one green stack entirely inside the region, or 2 green stacks with one square in the region and one square outside of the region.

(3) Play against computer.

- When the game starts, it should prompt the user for whether each of Player A and/or Player B is a human player or to be played by the computer (Note that any combination is valid: human vs human, human vs computer, computer vs human, or computer vs computer).
- Your computer play does NOT need to be very smart---you are not expected to develop any sophisticated AI.
- Your computer player can just randomly place its stacks (as long as they are valid locations) and can just randomly dig in any square on its turn. (Note: see extra credit for building more intelligent computer interactions)
- When the computer plays its turn, no board state should be displayed, and no prompts should be printed. The game should only print the outcome of the action (e.g., Player B found your Blue stack at D4!).
- If the computer plays a special action, the game should state "Player B used a special action" but not state which one nor any details of where or its outcome.

```
=================
| Extra Credit   |
=================
```

****
***** Note that the extra credit is NOT due with the assignment, *****
***** but rather may be done after the deadline. *****
***** You may turn in any extra credit by April 4/2 *****
****

The above "game" is a nice activity in exploring changes to software requirements/design, but a pretty terrible game. We provide you with 3 opportunities for extra credit. You may do any combination of these (one, two, or three of them). These are rather open-ended, so the number of points gained will be decided based on the quality of the work.

(1) Graphical User Interface.
Replace the text-based interface with something graphical. A more polished, more intuitive UI will gain more points here.

(2) Networked game.
Make the game so that the two players can be on different computers. You can explore a variety of features in this area from simply using game play, but across a network, to adding any number of features: user accounts/authentication, match making, etc.

(3) Intelligent computer play.
Instead of just having the computer dig randomly, write an intelligent AI. This could be small/simple improvements (making use of special moves, using information from the outcome of each move to inform your next decisions) for a few points, or it could be something very clever and sophisticated for many.

For any of these, what you add, how you add it, and anything else you want to improve about the game is up to you!