Command-line Interface

-seed x	it must be followed by a number to set the seed for the whole game
-load xxx	it must be followed by a string to specify the file name used to load the game(file format has to be as described later, ie "savefile.txt")
-board xxx	it must be followed by a string to specify the file name used to load the board. (file format has to be as described later, ie "layout.txt")
-random-board	starts a game with a randomly generated board. (use seed)
-computer	*extra feature plays the game with **one** human player and **three** computer players. (default four humans)

- If the user does not use -seed to specify a seed, the game will use **random seed**(system_clock)
- The game checks the command in the above order to determine how to load the game(check -load first, -board next, then -random-board).
- If **no command** is provided, the game loads the board from the "layout.txt" file. (if the game can not find layout.txt, the game can not start since there is not enough information).
- We design two types of players: **human players** who reads input from cin and **computer players** who choose from the available commands randomly with the objective of winning the game.

After loading the board with one specified strategy stated above, the actual game starts.

File format

There are three situations that need to read/write files

1. -load xxx command. Load the game from xxx file
2. when the game received eof from input at any point, the current game state is saved to back.sv
3. -board xxx command. Load the board from xxx file

For **(3)** situation, the file format is described as section 3.8 in project pdf
For the **(1), (2)** situations, the file format is described as section 3.7
<curTurn> -> an integer that indicates which builder's turn
<builder0Data>
<builder1 Data>
<builder2Data>
<builder3Data>
<board>
<geese>

Note:
For example, if builder 0 doesn't have any resources, road, residences and have two basements at 0, 19. The builder0 data line in the file is as follows, the r h characters appear but no T, H characters.
0 0 0 0 0 r h 19 B 0 B

Beginning of Game:

At the beginning of the game, each builder is prompted with the statement:
"Builder <colour>, Where do you want to build a basement?".

- **For human players**, the program needs to read in an integer from cin which represents a valid vertice.
    - **Normal Case**: The player **responds with an integer representing a valid vertex that has not been occupied**. The program adds a basement at the location chosen for the current player.
    - **Wrong Input(1):** The player's response is not an integer, the program waits for another input from cin.
    - **Wrong Input(2):** The player responds with an invalid integer and waits for another input from cin.
- **For computer players,** the process of choosing commands is much more intelligent, the computer will choose a valid vertex randomly.

The basements will be chosen by builders in the order Blue, Red, Orange, Yellow, Yellow, Orange, Red, Blue. Each builder will choose two valid basements at the beginning of the game.

Once all builders have placed their two basements, the program will print the updated board and begin the game.

Note: If the player quits during this phase, the data won't be saved

Beginning of Turn:

The builder can enter any of the following three commands:
By default, the player will have **loaded dice** at the beginning of the game

Command	Description
load	Sets the dice of the current builder to be loaded dice
fair	Sets the dice of the current builder to be fair dice
roll	Rolls the current builder's dice

	• If the current dice is loaded, then the program prints “Input a roll between 2 and 12.” and waits for the input. • If the current dice is fair dice, then the program rolls the fair dice twice randomly.

Once the player inputs “**roll**”, the program ends the “Beginning of the turn” phase and moves the builder to “During the turn”.

During the Turn:

During the turn, a builder can input any of the following commands:

Command	Description
board	prints the current board to standard output
status	prints the current status of all builders in order from builder 0 to 3
residences	prints the residences the current builder has currently completed
build-road <road#>	attempts to build the road at <road#>
build-res <housing#>	attempts to build the basement at <housing#>
improve <housing#>	attempts to improve the residence at <housing#>
trade <colour> <give> <take>	The program reads three strings for color, give and take. • Valid inputs: color is not himself, in the player range, give and take are available resources type • attempts to trade with builder <colour> giving one resource type <give> and receiving one resource of type <take> • If color1 has enough give resources and color2 has enough take resources, the program waits for a “yes” or “no” from player <colour2> •
next	passes control onto the next builder in the game. This ends the “During the Turn” phase

save <file>	saves the current game state to <file> The player needs to provide the filename string enclosed in <>. The format of this file is described in 3.7.
help	prints out the list of commands

- For commands that need the players to input an integer or string
  Invalid Inputs situations:
  (a): If the player does not input valid type
  (b): If the player does not input in range and invalid
  If any of the above wrong input situations arise, the program outputs the corresponding messages to warn the players and waits for the player to input another valid input.
  Otherwise, the program implement the command successfully

If the current player inputs “next”, the current player’s turn ends, and passes control onto the next player in the game.

End of Game

If a builder wins, the builders are prompted “ builder x wins!!”
The builders are prompted with the question: “Would you like to play again?”
If the answer is “yes”, then start a new game.
If the answer is not yes, then the game exits