Command-line Interface

		Example uses
-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 **i.e. use the provided files in zip: “savefile.txt” or “backup.sv”** (file format is as described later)	**./constructor -seed 10 -load savefile.txt** The game saved in savefile.txt is after the beginning phase
-board xxx	it must be followed by a string to specify the file name used to load the IDboard, **i.e. use the provided file in zip:“layout.txt”**. (file format is as described later)	**./constructor -seed 10 -board layout.txt** start the game from the very beginning
-random-board	starts a game with a randomly generated board. (use seed)	**./constructor -seed 1 -random-board** start the game from the very beginning(choosing two basements for each player)
-computer	*extra bonus feature plays the game with **one** human player and **three** computer players. (default four humans)	**./constructor -seed 10 -board layout.txt -computer**

- 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).
- -load and -board are contradicting commands.
- 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.

Notes for testing:
There are some input files provided for testing, instructions are at the end of this file

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 except the beginning of the game, 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: 0 represents Blue, 1 is Red…..
<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 character appears.
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 and prints nothing.
    - **Wrong Input(1):** The player's response is not an integer, then the program prints a warning message and waits for another input from cin.
    - **Wrong Input(2):** The player responds with an invalid vertex, then the program prints a warning message 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. Thus, we don't worry about the wrong inputs for computer players.

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(reaches eof) during this phase, the data won't be saved into backup.sv.

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 a valid integer from input. ● If the current dice is fair dice, then the program rolls the fair dice twice randomly.

Sample inputs for testing in the Beginning of Turn:

(1) If the player inputs **"load"**, then the program outputs a message telling the player that the current dice has been set to loaded dice. This is used to test command load.
(2) If the player inputs **"fair"**, then the program outputs a message telling the player that the current dice has been set to fair dice. This is used to test command fair.
(3) If the player inputs **"roll"**:
  (a) If the current dice for the player is **loaded dice**, then the program prints out **"Input a roll between 2 and 12: "**
  (i) **Normal Case**: The player inputs an integer between 2 and 12, then the program prints out a message telling the player what number he chooses.
  (ii) **Invalid Case**: The player does not input a valid integer between 2 and 12, then the program prints out a warning message and lets the player input again.

(b) If the current dice for the player is **fair dice**, then the program will generate two numbers between 1 and 6 randomly and their sum will be used for this turn.

After the dice is rolled:

(a) If the number rolled **is not 7**, then any builder who has a residence on a tile with the same value as the roll receives the same value as the roll receives resources associated with the tile and the residence present. The program prints out a message of the resource distribution.

(b) If the number rolled **is 7**(We can use loaded dice to explicitly roll a 7 for testing), any builder with 10 or more resources will automatically lose half of their total resources to the geese and the program will output messages of the amounts of the losing resources. Then the program prints out **"Choose where to place the GEESE"**.

(i) **Normal Case**: If the player responds with the number of any tile except where the GEESE is currently placed. Then the current builder steals a random resource from a builder who has a residence on the tile where the geese are moved.Then the builder chooses with the colour from whom they want to steal a random resource. **The input color should be exactly in the list.** Then the program chooses one random resource for the current player and outputs a corresponding message to tell the players which kind of resources he has stolen from another player.

(ii) Invalid Case: If the player does not respond to a valid tile number, the program outputs corresponding message to warn the player and waits until a valid input.

The above process is used to test moving geese functionality.

After 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#> **Note: <road#> should be an integer**

build-res <housing#>	attempts to build the basement at <housing#> **Note: <housing#> should be an integer**
improve <housing#>	attempts to improve the residence at <housing#> **Note: <housing#> should be an integer**
trade <colour> <give> <take> For colour, only the first letter is capital(Red, Blue.etc.) For give and take(resources), letters are all capital(HEAT, GLASS. etc.)	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 command "trade", if the player does not input another valid player's colour and/or the resources do not appear in the game and/or the player does not have enough resources for trading and/or the chosen player does not have enough resources for trading, our program outputs corresponding message to warn the player.

- For commands that need the players to input an integer or string, our program could handle some invalid input situations:
  (a): If the player does not input valid type
  (b): If the player does not input in valid range
  If any of the above wrong input situations arises, the program outputs the corresponding messages to warn the players.
  Otherwise, the program implements the command successfully and outputs nothing.

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.

Sample inputs

We provide two sample saved inputs for testing.

- To test beginning of the game phase

Using commands **./constructor -seed 10 -board layout.txt** or **./constructor -seed 10 -random-board**
we can input the integer in **savedinputs1**.txt:

```
12
23
23   // 23 has been occupied
55   // 55 is not in range
32
33
39
51
52
-1   // -1 is not in range
s   // s is not an integer
0
35
```

- For other parts of the game, we could use **savedinputs2.txt** in the zip file for testing. We comment below to tell what commands we are testing.

Using the command line **./constructor -seed 10 -load savefile.txt**
then input the commands given in **savedinputs2.txt** to test other commands during the game.

```
fair      // testing command “fair” in beginning of turn phase
roll      // testing “rolling fair dice”
next      // testing command “next” in during of turn phase
load      // testing command “load”
roll      // testing “rolling loaded dice”
9
next
roll
```

```
9
build-road 28
build-res 23
improve 23
improve 23
board   // testing command “board” in during of turn phase
status   // testing command “status” in during of turn phase
residences   // testing command “residences in during of turn phase
build-road 31   // successful, since there is a adjacent residence owned
build-road 25   // fail since no adjacent road or residence built by same owner
build-road 39  // successfully since there is a adjacent road we just built
build-road 44   // fail since we shouldn’t bypass others’ residences
build-road 40
build-road 44   // successful since we detour from another direction and doesn't bypass
next
roll
2
board
improve 30    // testing command “improve” in during of turn phase
Build-road
help      // testing command “help” in during of turn phase
trade Red BRICK ENERGY   // testing command “trade” in during of turn phase
yes
save temp.txt    // testing command “save” in during of turn phase
```