Demo

To Start a game

Valid Command-line arguments to enter the game(we list some cases as belows)

A. ./constructor(play the game with default layout.txt imported)

B. ./constructor -random-board(player the game with a randomly generated board)

C. ./constructor -seed x -random-board(set the seed value to x, and use that seed to randomly generate a board)

D. ./constructor -board layout.txt(assume layout.txt is a valid file to read from, we stores the board information from this file and play the game)

E. ./constructor -load save.txt(assume save.txt is a valid file, then we continue the game where the game information is stored in save.txt)

F. ./constructor -random-board -random-board -seed x(we type -random-board multiple times, but this action has the same behavior of C)

……. *There are many other valid cases that can activate a game, but we choose not to specify them here because our code is sufficient for accomplish this.*

Invalid Command-line arguments that won't enter the game(list some cases as belows)

A. ./constructor -load or ./constructor -board or ./constructor -seed (without a valid parameter in each case, the game won't starts, and it will throw a exception indicating that the current commands (-load, -board,-seed) lack valid parameters

B. ./constructor -load error.txt or ./constructor -board error.txt (here we assume that error.txt is a file with invalid format)

C. ./constructor -seed x -seed x (if change -seed by -board or -load, it will inform user of multiple specified command-line arguments more than once)

D. ./constructor -load save.txt -board layout.txt(it will specify that -load and -board cannot be specified together (we can either enter -load first or -board first) since they contradicts each other)

E. ./constructor -random-board -load save.txt or -random-board -board layout.txt(either of them will inform user of the previously-defined -random-board command is ignored because -load and -board has high priority to -random-board)

…. *There are many other invalid cases that can activate a game, but we choose not to specify them here because our code is sufficient for accomplish this.*

If we start the game without using -load,

Then in order to begin the game, the builder will be asked to build two basements in the following order:

Blue >> Red >> Orange >> Yellow >> Yellow >> Orange >> Red >> Blue

After being asked:

Builder (Blue/Red/Orange/Yellow) where do you want to build a basement?

Builder is asked to input an integer between 0 and 53 to help build a basement.

i. valid integer is entered:

- a basement will try to be built at given location.
- case 1: if they builder has enough resources, the basement is successfully built, the following will be printed:

"Builder (Blue/Red/Orange/Yellow) successfully built a Basement at #."

- case 2: if there is a residence already built at the given location

or next to it, or the residence that the builder entered does not exist, the following will be printed:
"You cannot build here."

ii. invalid input is entered:
- will be asked to type again.
The following will be printed:
"Invalid command.
Please enter 'help' for a list of valid commands."

The Game Start:

After being asked:
Enter a Command:
At the beginning of the term, if the user enter:

1. *load* or *l* (in either lower case or upper case):
   - the dice of the current builder will be set to loaded dice.
   - "Builder (Blue/Red/Orange/Yellow) now has loaded Dice." will be printed.
2. *fair* or *f* (in either lower case or upper case):
   - the dice of the current builder will be set to fair dice.
   - "Builder (Blue/Red/Orange/Yellow) now has fair Dice." will be printed.
3. *roll* or *r* (in either lower case or upper case): builder will first be asked to input an integer between 2 and 12 to help set the dice if he/she has a loaded dice.
   i. valid integer is entered:
      - the dice will be set to input value.
   ii. invalid input is entered:
      - "Invalid input ERROR: Not an integer" will be printed
      - will be asked to type again if Input not between 2 and 12:"

- rolls the current builder's dice.
- if dice 7 is rolled, any builder with 10 or more resources will automatically lose half of their resources (rounded down) to the geese. The resources lost are chosen at random and the process is described below. For each builder who loses resources. The following will be printed:
"Builder <colour> loses <numResourcesLost> resources to the geese. They lose:
<numResource> <resourceName>"
For example,
"Builder Blue loses 242 resources to the geese. They lose :
45 BRICK
55 ENERGY
47 GLASS
47 HEAT
48 WIFI
Builder Red loses 15 resources to the geese. They lose :
1 BRICK
5 GLASS
9 HEAT
Builder Orange loses 250 resources to the geese. They lose :
48 BRICK
57 ENERGY
48 GLASS
50 HEAT
47 WIFI"
The current builder can choose where to place the geese after "Choose where to place the GEESE." is printed.

i. valid integer is entered:

- case 1: if that tile already has geese, or the tile that the builder entered does not exist, the following will be printed:
  "Geese can't move here."
- case 2: if that tile is a valid place to place geese, the following will be printed:
  a) "Builder (Blue/Red/Orange/Yellow) has no builders to steal from." If there is no other builder on that tile with enough resource to steal from.
  b) "Builder (Blue/Red/Orange/Yellow) can choose to steal from [builders]." If there exists at least one builder with enough resource to steal from.
  - If the builder chooses one of the listed builders, the following will be printed:
    "Builder (Blue/Red/Orange/Yellow) steals GLASS from builder (Blue/Red/Orange/Yellow)."
  - If the builder chooses anyone else,
    The following will be printed and will be asked to enter again:
    "They can't be stolen from."
  - If the builder chooses anyone else,
    The following will be printed and will be asked to enter again:
    "Invalid colour."

ii. invalid input is entered:

- will be asked to type again.
  The following will be printed:
  "Invalid command.
  Please enter 'help' for a list of valid commands."

- The corresponding resources that the builder gained will be printed.
- This ends the "Beginning of the turn" phase and moves the builder to "During the turn".

4. *status* or *s* (in other lower case or upper case):
   - prints the current status of all builders in order from builder 0 to 3.
   - The following will be printed:
     "Builder Builder Blue has # building points, # BRICK, # ENERGY, # GLASS, # HEAT, # WIFI.
     Builder Builder Red has # building points, # BRICK, # ENERGY, # GLASS, # HEAT, # WIFI.
     Builder Builder Orange has # building points, # BRICK, # ENERGY, # GLASS, # HEAT, # WIFI.
     Builder Builder Yellow has # building points, # BRICK, # ENERGY, # GLASS, # HEAT, # WIFI."
5. *help* or *h* (in other lower case or upper case):
   - prints out the list of commands that can be used at the Beginning of the turn.
   - The following will be printed:
     "Valid commands:
     ~ load : changes current builder's dice type to 'loaded'
     ~ fair : changes current builder's dice type to 'fair'
     ~ roll : rolls the dice and distributes resources.
     ~ status : prints the current status of all builders in order from builder 0 to 3.
     ~ help : prints out the list of commands."
6. Anything else
   - will be considered as invalid input and will be asked to type again.

- The following will be printed:
  "Invalid command.
  Please enter 'help' for a list of valid commands."

During the turn, if the user enter:

1. *status* or *s* (in either lower case or upper case):
   - prints the current status of all builders in order from builder 0 to 3.
   - The following will be printed:
     "Builder Builder Blue has # building points, # BRICK, # ENERGY, # GLASS, # HEAT, # WIFI.
     Builder Builder Red has # building points, # BRICK, # ENERGY, # GLASS, # HEAT, # WIFI.
     Builder Builder Orange has # building points, # BRICK, # ENERGY, # GLASS, # HEAT, # WIFI.
     Builder Builder Yellow has # building points, # BRICK, # ENERGY, # GLASS, # HEAT, # WIFI."

2. *help* or *h* (in other lower case or upper case):
   - prints out the list of commands that can be used During of the turn.
   - The following will be printed:
     "Valid commands:
     ~ board : prints the current board.'
     ~ status : prints the current status of all builders in order from builder 0 to 3.'
     ~ build - road <road#> : attempts to builds the road at <road#>.
     ~ build - res <housing#> : attempts to builds a basement at <housing#>.
     ~ improve <housing#> : attempts to improve the residence at <housing#>.
     ~ trade <colour> <give> <take> : attempts to trade with builder <colour>, giving one resource of type <give> and receiving one resource of type <take>.
     ~ market <sell> <buy> : attempts to sell resources on the market, giving four resource of type <sell> and receiving one resource of type <buy>.
     ~ next : passes control onto the next builder in the game.
     ~ save <file> : saves the current game state to <file>.
     ~ help : prints out the list of commands."
3. *board* or *b* (in other lower case or upper case):
   - prints the current board.
4. *residences* or *r* (in other lower case or upper case):
   - prints the residences the current builder has currently completed.
   - For example:
     "Builder Blue gained:
     1 ENERGY
     Builder Orange gained:
     1 ENERGY
     Enter a command:
     > residences
     Red has built:
     6 B
     40 B"
5. *build-res* or *buildres* (in other lower case or upper case): builder will then be asked to input an integer between 0 and 53 to help build a basement.
   i. valid integer is entered:
   - a basement will try to be built at given location.
     - case 1: if they builder has enough resources, the basement is successfully built, the following will be printed:
       "Builder (Blue/Red/Orange/Yellow) successfully built a Basement at #."
     - case 2: if there is a residence already built at the given location or next to it, or the current builder do not have a road built to that

vertex, or the residence that the builder entered does not exist, the following will be printed:
"You cannot build here."

- - case 3: if the builder does not have enough resources, the following will be printed:
"You do not have enough resources.
The cost of a basement is one Brick, one Energy, one Glass, and one Wifi resource."

ii. invalid input is entered:
- will be asked to type again.
The following will be printed:
"Invalid command.
Please enter 'help' for a list of valid commands."

6. *build-road* or *build road* (in other lower case or upper case): builder will then be asked to input an integer between 0 and 71 to help build a road.

i. valid integer is entered:
- a road will try to be built at given location.
- case 1: if there is a road already built at the given location or the road that the builder entered does not exist, the following will be printed:
"You cannot build here."
- case 2: if they builder has enough resources, the road is successfully built, the following will be printed:
"Builder (Blue/Red/Orange/Yellow) successfully built a Road at #."
- case 3: if the builder does not have enough resources, the following will be printed:
"You do not have enough resources.
The cost of a Road is one Heat and one WIFI resource."

ii. invalid input is entered:
- will be asked to type again. The following will be printed:
"ERROR: isn't a valid integer."

7. *improve* (in other lower case or upper case): builder will then be asked to input an integer between 0 and 53 to help improve the residence.

i. valid integer is entered:
- the residence will try to be improved at given location.
- Case 1: if the builder improves others' residences or improve an empty location or the road that the builder entered does not exist, "Invalid residence." will be printed.
- Case 2: if the improved residence is Tower, "You can't improve that building." will be printed.
- Case 3: if there is no enough resources, the following will be printed:
"You do not have enough resources.
The cost to improve a Basement to a House is two GLASS and three HEAT resource.
The cost to improve a House to a Tower is three BRICK, two ENERGY, two GLASS, one WIFI, and two HEAT."
- Case 4: if the residence can be improved, the following can be printed:
"Builder (Blue/Red/Orange/Yellow) successfully built a (House/Tower) at #."

ii. invalid input is entered:
- will be asked to type again. "ERROR: isn't a valid integer." will be printed.

8. *trade* (in other lower case or upper case): builder will then be asked to input three times for <colour>(either a number between 0-3, full name or the first letter of its name), <give>(either a number between 0-4, full name or the first letter of its name), <take> (either a number between 0-3, full name or the first letter of its name).

i. valid integer or string is entered:
- the current builder will try to trade with builder <colour>, giving one resource of type <give> and receiving one resource of type <take>.
- Case 1: when the builder trade with himself. "Why are you trading for

the same resource...” will be printed.
- Case 2: when the builder has no <give> resource, “You don't have enough <give>.” will be printed.
- Case 3: when the <colour> has no <take> resource, “<colour> doesn't have enough <take>" will be printed.
- Case 4: when <colour>, <give> and <take> are all valid, “<current_builder > offers <colour> one <give> for one <take>. Does <take> accept this offer? accept this offer?” will be asked. Then the <colour> needs to enter“yes” or “no”
- If “yes” is entered, “<current_builder > gains one <take> and loses one <give>
, <colour> gains one <give> and loses one <take>.” will be printed.
- If “no” is entered, “<colour> declined the trade.” will be printed.
- If an invalid command is entered, “Invalid command.” will be printed.

ii. invalid input is entered:
- will be asked to type again. Print “Invalid colour.” or “Invalid item.” depends on whether the given colour or resource name is valid or not.

9. *next* or *n* (in other lower case or upper case):
   - A board will be printed. Passes control onto the next builder in the game. This ends the “During the Turn” phase.
   - If any builder gains at least 10 building points, ends the game.
10. *save* (in other lower case or upper case): builder will then be asked to input an filename to help save the game.
    - the game will then be saved to the given file. “Saving to <file_name>” will be printed.
11. *market* (in other lower case or upper case): builder will then be asked to input two times for <sell>(either a number between 0-4, full name or the first letter of its name), <buy> (either a number between 0-3, full name or the first letter of its name).

i. valid integer or string is entered:
- the current palyer will try to trade with market, giving four resource of type <sell> and receiving one resource of type <buy>.
- Case 1: when the builder exchange with same resource. “Why are you buying for the same resource?” will be printed.
- Case 2: when the builder has no <give> resource, “You don't have enough <give>.” will be printed.
- Case 3: when <sell> and <buy> are all valid, “<current_builder > gains one <buy> and loses four <sell>.” will be printed.

ii. invalid input is entered:
- will be asked to type again. Print “Invalid colour.” or “Invalid item.” depends on whether the given colour or resource name is valid or not.

11. Anything else
    - will be considered as invalid input and will be asked to type again.
    - The following will be printed”
      “Invalid command.
      Please enter 'help' for a list of valid commands.”

End of the Game

After being asked:
Would you like to play again?
Enter *yes* or *y* to starts over from the beginning.
Enter *no* or *n*, the game exits.
Anything else will be considered as invalid input and will be asked to type again.