CS 246 Final Project – Constructor

DEMO

Trevor Chen (h586chen)

Eric Liu (e44liu)

Constructor is a game which you are a housing developer in Waterloo. You want to build as many housings as possible for Waterloo students. The first player that gained 10 building points win the game.

To start a game, there are several options to choose to start a game:

Command	Description
-seed xxx	Set up a new game using the seed xxx.
-load xxx	Set up a game from the file xxx. This file should contain valid game content.
-board xxx	Set up a new game with a file in the board xxx. The board do not have restriction of random generation.
-random-board	Set up a new game with random board.

- If the program starts without any command line argument, the program will read the board from layout.txt as default.
- Use a combination of "-seed xxx" and "-random-board" to generate a game with board according to seed.
- **[*Additional Feature*]** The program can handle any combination of command line options.

```
e44liu@ubuntu1804-008:~/cs246/eric/project/CS246-Project$ ./constructor -seed 12 -load Data.txt -random-board
ERROR: -random-board being ignored in favour of previous -load
```

```
e44liu@ubuntu1804-008:~/cs246/eric/project/CS246-Project$ ./constructor -seed 12 -seed 12 -seed 12 -seed 12
ERROR: already specified -seed once before
```

>>> Initializing Game Stage <<<

[*New Game Starts Here*]

You are now at the initializing game stage, where the board is generated, and each player get turns to build their first two residence on the board.

```
> Builder Blue's turn.
>
```

[*Game Rule*] Residences cannot be built adjacent to each other and during this stage, the residences cannot experience building upgrade.

```
Builder Red where do you want to build a basement?
> 1
You cannot build here.
Basements already exist at locations: 0
```

```
Builder Yellow where do you want to build a basement?
> 23
Builder Yellow where do you want to build a basement?
> 23
You cannot build here.
Basements already exist at locations: 0 7 23 28
```

After the board has set up, the board will display automatically to you and wait for the first player (Blue) to input the Basement.

[*Game Rule*] The player turns: *Blue* > *Red* > *Orange* > *Yellow* > *Yellow* > *Orange* > *Red* > *Blue*

After all players have build their initial two basements on the board, the game starts.

>>> Game Stage <<<

> Beginning of Turn <

[*Loaded Game Starts Here*]

The players take turns to play, with the order *Blue* > *Red* > *Orange* > *Yellow* and repeat.

At the Beginning of Turn, players can enter the following commands:

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 and end "Beginning of the turn".
status	Print out all player's resources and building points.
help	See what command can be input.

Demonstration of Command "load":

```
> load
Builder Blue now has loaded Dice.
>
```

Demonstration of Command "fair":

```
> fair
Builder Blue now has fair Dice.
>
```

Demonstration of Command "roll" of a fair dice:

```
> roll
The number you rolled is 9
Builder Yellow gained:
1 ENERGY
Enter a command:
>
```

Demonstration of Command "roll of a load dice:

```
> roll
Input a roll between 2 and 12:
> 5
```

[*Game Rule*] – Roll (fair & load dice)

Roll command consume a load dice by default.

Give resources to the builder who build buildings on the tile which tile value matches the dice roll point.

Print information about who gain what resources.

Enter the [During of Turn] stage.

Demonstration of Command "status": [**Note**] "status" has similar output in [During] and [Beginning].

```
> status
> Builder Blue     has 2 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT, 0 WIFI.
> Builder Red      has 2 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT, 0 WIFI.
> Builder Orange   has 2 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT, 0 WIFI.
> Builder Yellow   has 2 building points, 0 BRICK, 1 ENERGY, 0 GLASS, 0 HEAT, 0 WIFI.
>
```

Demonstration of Command "help": [**Note**] "help" has different output in [During] and [Beginning].

```
> help
~ 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.
>
```

> **During of Turn** <

At the During of Turn, players can enter the following commands:

Command	Description
board	Prints the current board.
status	Prints the current status of all builders in order from Blue>Red>Orange>Yellow
residences	Prints the residences the current builder has currently completed.
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>
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>
help	Prints out the list of commands
market <sell> <buy>	**[*Additional Feature*]** Sell 4 resources of the same type and gain one resources of any type

Demonstration of Command "status":

```
> status
Builder Blue    has 2 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT, 0 WIFI.
Builder Red     has 2 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT, 0 WIFI.
Builder Orange  has 2 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT, 0 WIFI.
Builder Yellow  has 2 building points, 0 BRICK, 1 ENERGY, 0 GLASS, 0 HEAT, 0 WIFI.
```

Demonstration of Command "residences":

```
> residences
Blue has built:
23 T
35 H
46 T
50 H
52 B
```

[Note] The information is printed in the order of position from least to greatest.

Demonstration of Command "build-road <road#>": [Note] Player will lose resources and board will be updated if the road is built successfully.

```
> build-road 25
Builder Blue successfully built a road at 25.
```

Demonstration of Command "build-res < housing#>": [Note] Player will lose resources and board will be updated if the Res is built successfully.

```
> build-res 11
Builder Blue successfully built a Basement at 11.
```

Demonstration of Command "improve <housing#>":

```
> improve 11
Builder Blue successfully built a House at 11.
```

```
> improve 11
Builder Blue successfully built a Tower at 11.
```

[Note] Player will lose resources and board will be updated if the Res is improved successfully.

Demonstration of Command “trade <colour> <give> <take>”:

```
> trade red brick energy
> > Blue offers Red one BRICK for one ENERGY.
Does Red accept this offer?
y
Blue gains one ENERGY and loses one BRICK,
Red gains one BRICK and loses one ENERGY.
```

[**Note**] Player may lose resources and gain resources.

```
> trade blue brick brick
> > Can't trade with yourself.
> > Why are you trading for the same resource...
Enter a command:
> trade orange wifi brick
> > Blue offers Orange one WIFI for one BRICK.
Does Orange accept this offer?
n
Orange declined the trade.
```

Demonstration of Command “save <file>”:

```
> save saveIT.sv
Saving to saveIT.sv...
```

Demonstration of Command “help”:

```
> help
~ board : prints the current board.
~ 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 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.
```

[***Additional Feature***] Demonstration of Command “market <sell> <buy>”:

```
Enter a command:
> market brick wifi
Welcome to the Market! : )
Which resources do you want to sell ?
> Which resources would you like?
> You got one [WIFI] and sold four [BRICK] !
Thanks for visiting market, Bye~
```

For Command “next”:

The [During of Turn] ends.

The game loops back to Beginning of Turn and the next player plays the game.

The game will not end (keep looping in this stage) only if EOF is reached or one player wins (reached 10 building points).

```
Enter a command:
> End of file reached.

Saving to backup.sv...
e44liu@ubuntu1804-008:~/cs246/eric/project/CS246-Project$
```

EOF is Read

```
Enter a command:
> next
Congratulations!! Blue wins!!
Would you like to play again?
>
```

One Player Wins

>>> Ending Stage <<<

Condition 1:

When EOF reached, the program is popped from the [**Game Stage**] and automatically save the game to backup.sv

Condition 2:

When a player wins, the program will be popped from the [**Game Stage**] and ask if the user want to play again.

If received any combination of case of “no” or “n”, the game ends and program exist.

If received any combination of case of “yes” or “y”, the game cleans the current board and players’ data. Then the game regenerated the same empty board from last game and the program entered the [**Initializing Game Stage**].

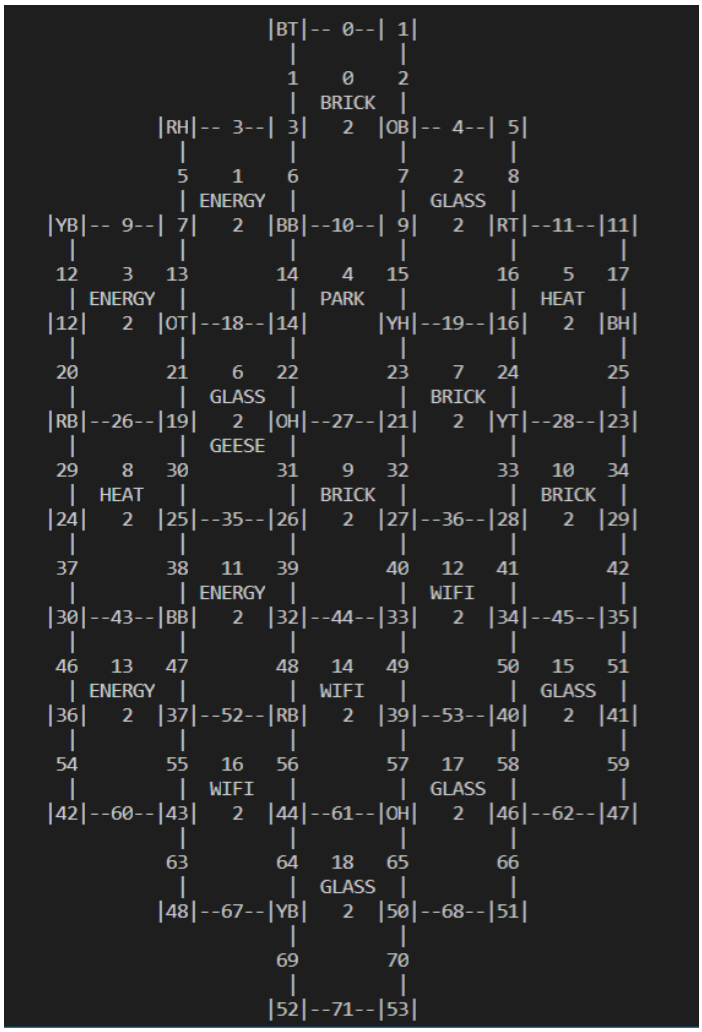
Cases During the Game

[Roll Dice and Give Resources to Builder**] (**Using [./constructor -load give.test] to check**)**

[**Note**] All tiles’ values are 2.

Player gains resources, the information is recorded and print out together after organization.

```
> Builder Blue's turn.
> roll 2
Input a roll between 2 and 12:
> Builder Blue gained:
4 BRICK
3 ENERGY
3 HEAT
Builder Red gained:
4 ENERGY
3 GLASS
4 HEAT
2 WIFI
Builder Orange gained:
3 BRICK
6 ENERGY
5 GLASS
2 WIFI
Builder Yellow gained:
8 BRICK
1 ENERGY
3 GLASS
3 HEAT
1 WIFI
```



[Roll 7 and Stole Cards from Players] (Using [./constructor -load stole.test] to check)

Initial Resources:

```
> Builder Blue's turn.
> status
> Builder Blue     has 11 building points, 90 BRICK, 10 ENERGY, 30 GLASS, 80 HEAT, 150 WIFI.
> Builder Red      has 2 building points, 50 BRICK, 100 ENERGY, 40 GLASS, 100 HEAT, 20 WIFI.
> Builder Orange   has 2 building points, 100 BRICK, 100 ENERGY, 100 GLASS, 100 HEAT, 100 WIFI.
> Builder Yellow   has 2 building points, 100 BRICK, 0 ENERGY, 100 GLASS, 5 HEAT, 100 WIFI.
```

Blue roll 7:

```
> roll 7
Input a roll between 2 and 12:
> > Builder Blue loses 180 resources to the geese. They lose :
51 BRICK
7 ENERGY
16 GLASS
36 HEAT
70 WIFI
Builder Red loses 155 resources to the geese. They lose :
26 BRICK
51 ENERGY
21 GLASS
48 HEAT
9 WIFI
Builder Orange loses 250 resources to the geese. They lose :
55 BRICK
52 ENERGY
50 GLASS
48 HEAT
45 WIFI
Builder Yellow loses 152 resources to the geese. They lose :
54 BRICK
0 ENERGY
49 GLASS
3 HEAT
46 WIFI
```

Place the geese onto a new tile and steal other player's resources:

[**Note**] Place on the same tile will print error message and let user re-input.

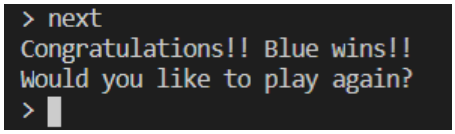
```
Choose where to place the GEESE.
> 14
Builder Blue can choose to steal from
Red, Orange.
Choose a builder to steal from.
> r
Builer Blue steals HEAT from builder Red.
```

Final status:

```
> status
Builder Blue     has 11 building points, 39 BRICK, 3 ENERGY, 14 GLASS, 45 HEAT, 80 WIFI.
Builder Red      has 2 building points, 24 BRICK, 49 ENERGY, 19 GLASS, 51 HEAT, 11 WIFI.
Builder Orange   has 2 building points, 45 BRICK, 48 ENERGY, 50 GLASS, 52 HEAT, 55 WIFI.
Builder Yellow   has 2 building points, 46 BRICK, 0 ENERGY, 51 GLASS, 2 HEAT, 54 WIFI.
```

[End Game / Replay Game**]** **(**Using [./constructor -load stole.test] to check**)**

When “next” command is read, automatically check if there is a winner:



If enter any of cases of “y” or “yes”, a new game started using the same board as last game, all previous game data is cleaned. Game stage jump back to **[Initializing Game Stage]**.

If enter any of cases of “n” or “no”, the game quit without saving the game, the program exists.

Other input will be considered as [Invalid command] and the game will let the user re-input answer.