Constructor Final Demo

Alice Li, Daniel Lu, Matthew Tam

Initializing of Game

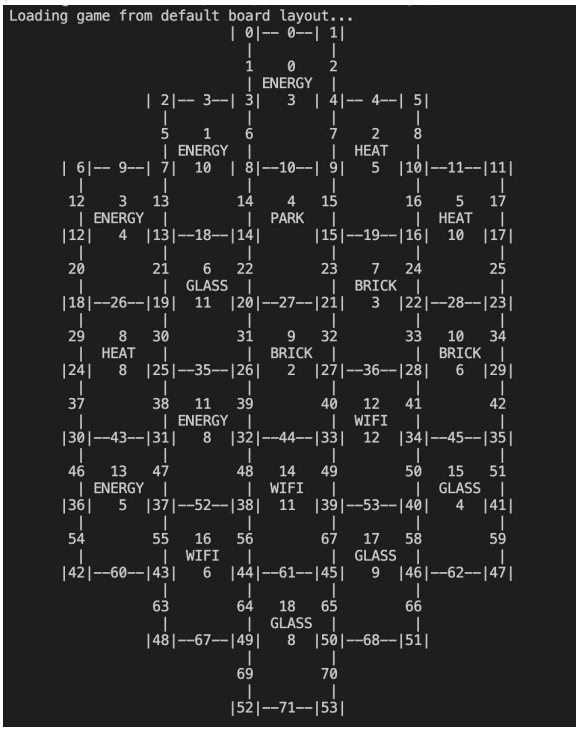
Note:

All random number generation is done based on the seed. If we are given a seed, we will get the same results. However, if no seed is given, we generate a new seed based on time. This seed is used to randomly generate the board and is used for things like rolling a fair dice and selecting what resources to steal when a goose moves to a new tile.

Initializing game with multiple command line arguments

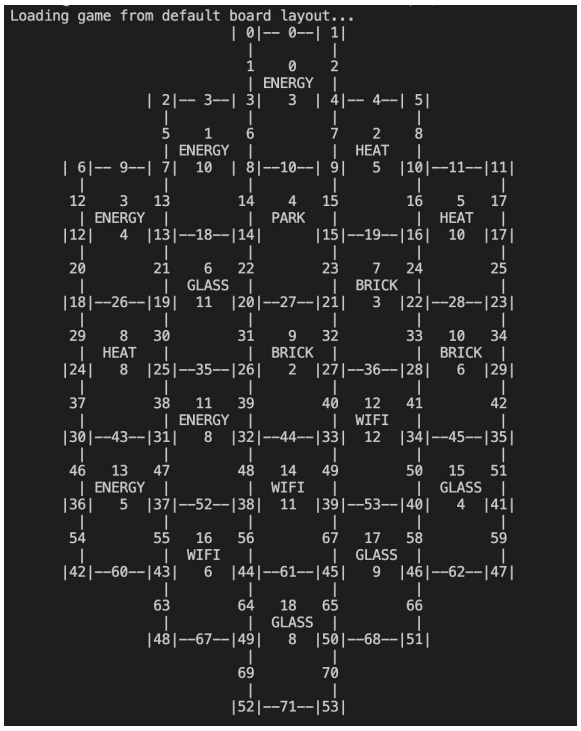
1. Produces a error message if you have both -board and -load, or if -board or -load is indicated more than once
2. Ignores -random-board if -board or -load are given as command line arguments
3. Implemented so that order of command line arguments do not matter

Initializing game with default layout

$ ./constructor 	When initializing the game, it will look for a file called 'layout.txt' in the current directory. If no file exists, the program will print the following message to std error: "Error: Unable to open file layout.txt for default board layout."

Initializing game with default layout and specific seed for PRNG

$ ./constructor -seed 100



It will initialize the board with the default layout. However, the seed that will be used in the PRNG such as for the rolling of dice will be set to what the program reads as command line arguments. The following messages are returned to std error in the cases described below:

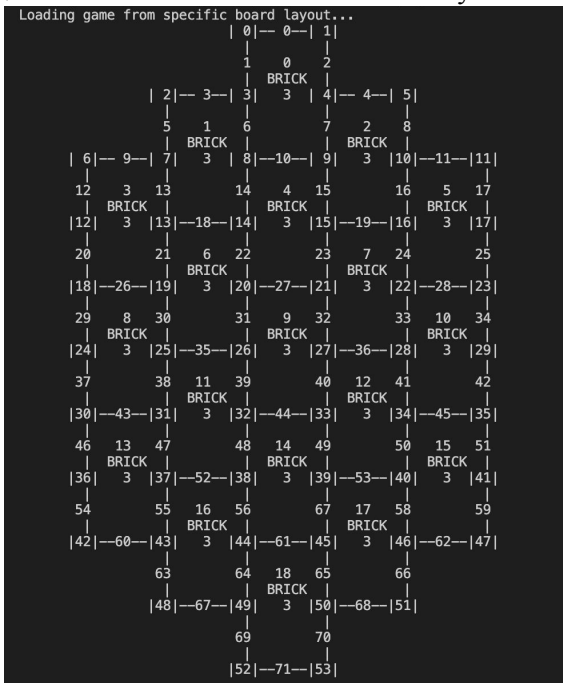
Already set -seed:
“ERROR: already specified -seed once before”

Missing value after -seed:
“ERROR: -seed missing seed argument”

Value of seed is not an integer:
“ERROR: the seed must be an integer”

Initializing game with specific board layout

$ ./constructor -board ./testSuite/testLayout1.txt



It will initialize the board with the layout provided in the file given as argument immediately following -board. The following messages are returned to std error in the cases described below:

Missing file after -board
“ERROR: -board missing filename argument”

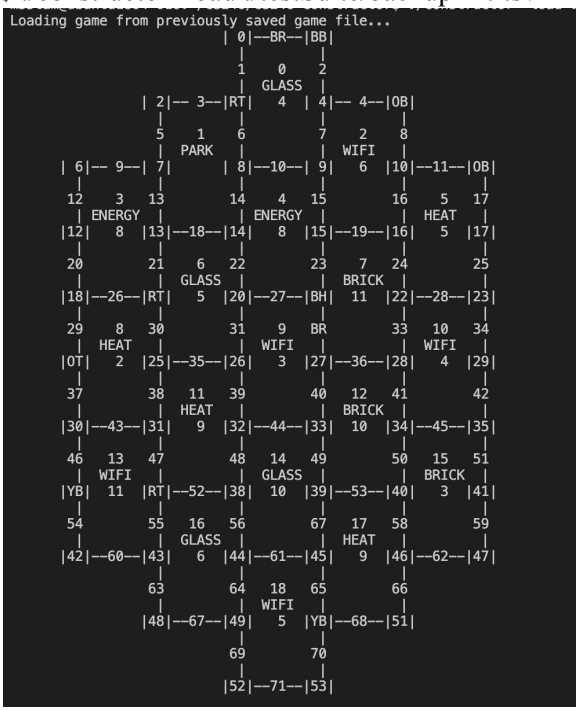
Invalid file
“ERROR: Invalid file”

Unsupported format of board layout
“ERROR: Unsupported board format in file <filename>. (... error with file ...).”

Also note that it is valid to have a board full of only one resource. We do not check for ‘valid boards’ (except we make sure that rolls of 7 must be a park resource).

Initializing game from saved game

$ ./constructor -load ./testSuite/backupInit-.sv



Produces the board from the saved game file and skips the part to set up initial buildings (basements). It will also automatically start at the correct player's turn. The following messages are returned to std error in the cases described below:

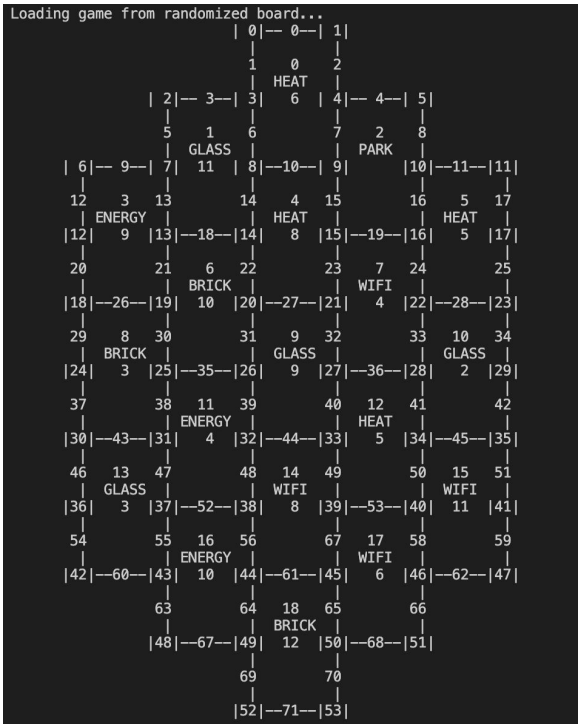
Missing file after -load
"ERROR: -load missing filename argument"

Invalid file
"ERROR: Invalid file"

Unsupported format of board layout
"ERROR: Unsupported game format in file <filename>. (... error with file ...)."

Initializing game with random board layout

$ ./constructor -random-board -seed 420024



If you produce a randomized board with the same seed (420024), you will get the same board displayed on the left. When you do not provide a seed, the generated board will look different each time.

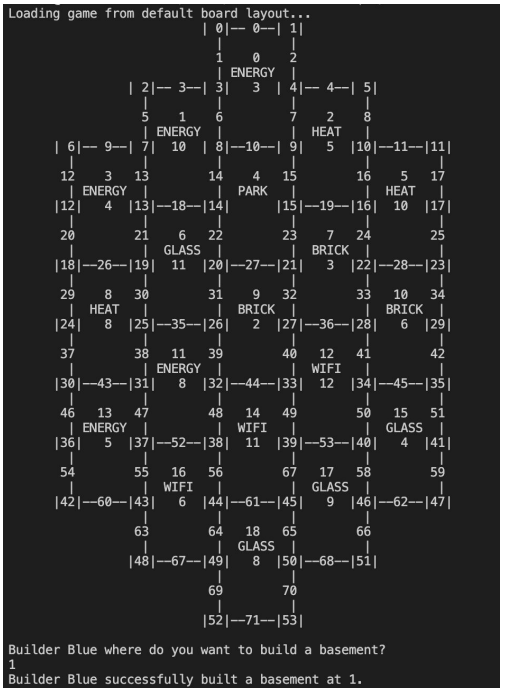
When randomly generated, the board consists of the following resources: 3 WIFI, 3 HEAT, 4 BRICK, 4 ENERGY, 4 GLASS, and 1 PARK. The values on the board will have the distribution of one tile with value 2, two tiles with values between 3 and 6, two tiles with values between 8 and 11 and one tile with the value 12.

Gameplay Walkthrough:

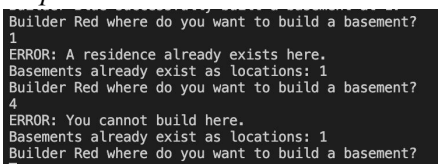
Gameplay begins (when not loading from a saved game) with each player being asked to build 2 basements. It starts with the first player (Blue) and once everyone places 1 basement, it goes in the reverse order starting from the last player (Yellow).

Consider the following example where we just loaded in game from default layout. It starts off by printing the board and then repeatedly asks players where they would want to place basements.

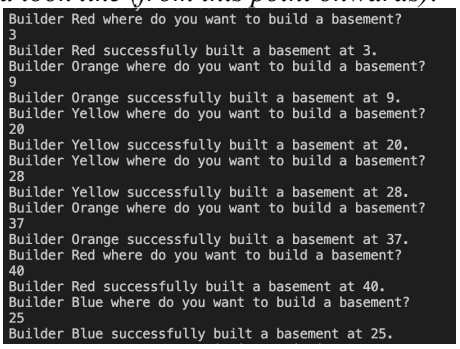
An example of successful basement placement.



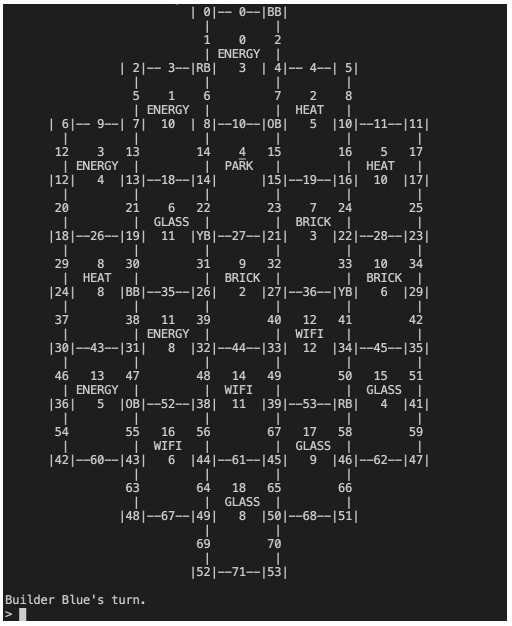
This is what happens when you try to place a basement on a tile that already has a basement or if you place it on a invalid vertex.



Assuming the rest of the players place basement in valid locations this is an example of what it would look like (from this point onwards):



Once the basements are set up, the game begins with player Blue's turn unless we are loading from a saved file in which the correct player turn is determined. At the beginning of each player's turn we will print out the state of the board along with whose player turn it is.



The following are valid moves at the beginning of each turn:

Help:

```
> help
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.
```

These are the only valid commands during the 'beginning of turn' phase. The next set of valid commands become available after the player enters 'roll', as they proceed to moves available in the 'during the turn' phase.

Status:

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

Notice that all players all 0 resources and 2 building points at the start of the game.

Load:

```
> load
Dice changed to loaded dice.
```

Fair:

```
> fair
Dice changed to fair dice.
```

Roll:

To test rolling of a loaded dice, exit the existing game and run the following:

$ *./constructor -load ./testSuite/backupRoll.sv -seed 420*

Change the player dice to loaded and enter 'roll'. The program will ask for user input and distribute the resources according to the board and location of player basements. If the roll is not valid, it will re-prompt the user for new input. The following example distributes resources in both tile 0 & 7.

```
Builder Blue's turn.
> status
Blue has 4 building points, 6 BRICK, 5 ENERGY, 4 GLASS, 2 HEAT , and 5 WIFI.
Red has 9 building points, 0 BRICK, 1 ENERGY, 2 GLASS, 2 HEAT , and 3 WIFI.
Orange has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 10 WIFI.
Yellow has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 1 WIFI.
> load
Dice changed to loaded dice.
> roll
Input a roll between 2 and 12:
3
3 is rolled.
Builder Blue got:
1 BRICK
Builder Red got:
3 WIFI
Builder Orange got:
3 WIFI
Enter a command:
> status
Blue has 4 building points, 7 BRICK, 5 ENERGY, 4 GLASS, 2 HEAT , and 5 WIFI.
Red has 9 building points, 0 BRICK, 1 ENERGY, 2 GLASS, 2 HEAT , and 6 WIFI.
Orange has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 13 WIFI.
Yellow has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 1 WIFI.
```

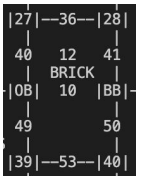
We use status before and after the roll to ensure that the correct players received the correct amount of resources. Also notice that the number of resources earned are dependent on how many residences and the type of residence the player owns on the tile with resources to be distributed.

To test rolling of a fair dice, continue the game from the step above by entering next. This will begin the next players 'beginning of turn' phase. Change the player dice to fair and enter roll which will output the generated dice roll and distribute subsequent resources.

```
> next
```

[… printing of board at beginning of Builder Red's turn …]

```
Builder Red's turn.
> fair
Dice changed to fair dice.
> roll
7 is rolled.
Builder Blue loses 11 resources to the goose. They lose:
5 Brick
1 Energy
3 Glass
2 Wifi
Builder Red loses 5 resources to the goose. They lose:
1 Heat
4 Wifi
Builder Orange loses 26 resources to the goose. They lose:
4 Brick
6 Energy
2 Glass
8 Heat
6 Wifi
Choose where to place the GOOSE.
12
Builder Red can choose to steal from Blue Orange
Blue
Builder Red steals Brick from builder Blue
Enter a command:
> status
Blue has 4 building points, 1 BRICK, 4 ENERGY, 1 GLASS, 2 HEAT , and 3 WIFI.
Red has 9 building points, 1 BRICK, 1 ENERGY, 2 GLASS, 1 HEAT , and 2 WIFI.
Orange has 7 building points, 6 BRICK, 4 ENERGY, 8 GLASS, 2 HEAT , and 7 WIFI.
Yellow has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 1 WIFI.
```



Here the randomly generated roll returned 7. Referring to the status of each player after we rolled the loaded dice builder blue red and orange had more than 10 resources and will lose half of their resources (rounded down). After choosing to move the geese to tile 12, they had the choice to steal from orange and blue. After deciding on stealing from blue, the board randomly generated a resource to steal. We use status after the roll to show that the correct players received the correct amount of resources.

If a player has no resources, but has a residence on the tile where the geese is to be placed, they are not returned in the list of users who you can steal from. This can be tested by exiting the current game and starting the new game with the following command:

$ *./constructor -load ./testSuite/backupGeeseEdge.sv -seed 420*

After loading in the game, change the dice to fair and roll (which once again returns 7). In the example the board is set-up the same except player yellow owns the basement on vertex 34.

```
Builder Blue's turn.
> status
Blue has 4 building points, 6 BRICK, 5 ENERGY, 4 GLASS, 2 HEAT , and 5 WIFI.
Red has 9 building points, 0 BRICK, 1 ENERGY, 2 GLASS, 2 HEAT , and 3 WIFI.
Orange has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 10 WIFI.
Yellow has 2 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT , and 0 WIFI.
> fair
Dice changed to fair dice.
> roll
7 is rolled.
Builder Blue loses 11 resources to the goose. They lose:
3 Brick
3 Energy
2 Glass
1 Heat
2 Wifi
Builder Orange loses 25 resources to the goose. They lose:
5 Brick
5 Energy
5 Glass
6 Heat
4 Wifi
Choose where to place the GOOSE.
12
Builder Blue can choose to steal from Orange
orange
Builder Blue steals Heat from builder Orange
```

Notice the use of status was used to show that builder yellow has 0 resources. Finally we also implemented the input to be case friendly. In the situation where builder blue tries to place geese on invalid tile then we get the following:

```
Choose where to place the GOOSE.
19
ERROR: Tile selection is invalid.
```

If builder blue tries to steal a resource from an invalid player we get the following:

```
Builder Blue can choose to steal from Orange
yellow
ERROR: Choose a valid player that you can steal from.
red
ERROR: Choose a valid player that you can steal from.
blue
ERROR: Choose a valid player that you can steal from.
```

After the roll, enter next and you will see the following board with GEESE on tile 12 when the board gets printed at the beginning of the next builders turn:

```
> next
                        | 0|--BR--|BB|
                          |       |
                          1   0   2
                          | GLASS |
                | 2|-- 3--|RT|  4  | 4|-- 4--|OB|
                  |       |       |       |
                  5   1   6       7   2   8
                  | PARK  |       | WIFI  |
        | 6|-- 9--| 7|     | 8|--10--| 9|  6  |10|--11--|BH|
          |       |       |       |       |       |
         12   3   13       14   4   15       16   5   17
          | ENERGY |       | ENERGY |       | HEAT  |
        |12|  8  |13|--18--|14|  8  |15|--19--|16|  5  |17|
          |       |       |       |       |       |
         20       21   6   22       23   7   24       25
          |       | GLASS |       | BRICK |       |
        |18|--26--|19|  5  |20|--27--|OH| 11  |22|--28--|23|
          |       |       |       |       |       |
         29   8   30       31   9   BR       33   10   34
          | HEAT  |       | WIFI  |       | WIFI  |
        |OT|  2  |25|--35--|RT|  3  |27|--36--|28|  4  |29|
          |       |       |       |       |       |
         37       38   11  39       40   12  41       42
          |       | HEAT  |       | BRICK |       |
        |30|--43--|31|  9  |32|--44--|OB| 10  |YB|--45--|35|
          |       |       |       | GOOSE |       |
         46   13  47       48   14  49       50   15  51
          | WIFI  |       | GLASS |       | BRICK |
        |BB| 11  |RT|--52--|38| 10  |39|--53--|40|  3  |41|
          |       |       |       |       |       |
         54       55   16  56       67   17  58       59
          |       | GLASS |       | HEAT  |       |
        |42|--60--|43|  6  |44|--61--|45|  9  |46|--62--|47|
                  |       |       |       |
                 63       64   18  65       66
                  |       | WIFI  |       |
                |48|--67--|49|  5  |YB|--68--|51|
                          |       |
                         69       70
                          |       |
                        |52|--71--|53|
Builder Red's turn.
```

If command is unrecognized:

```
> buy
Invalid command.
```

The builders can continue to input the commands listed above, but once they roll the dice, they move to the next phase of the builder's turn. Similar to the 'beginning of turn' phase, they can run the following valid commands as frequently as they choose as long as they do not go 'next':

Help

```
Enter a command:
> help
Valid Commands:
~ board
~ status : prints the current status of all builders in order from builder 0 to 3.
~ residences
~ build-road <road#> : attempts to builds the road at <road#>.
~ build-res <housing#> : attempts to builds a basement at <housing#>.
~ improve-res <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>.
~ multi-trade <colour> <give> <#given> <take> <#taken> : attempts to trade with builder <colour>, giving <#given> resource
s of type <give> and taking <#taken> resources of type <take>.
~ market-trade <give> <take> : attempts to trade with the market, giving four resources of type <give> and receiving one r
esource of type <take>.
~ 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.
```

Residences

To test printing of residences owned by current builder, exit the game and run the following:

$ *./constructor -load ./testSuite/backupMainBasic.sv -seed 690*

```
Builder Orange's turn.
> roll
Input a roll between 2 and 12:
2
2 is rolled.
Builder Orange got:
3 HEAT
Enter a command:
> residences
Builder Orange has 7 building points.
Builder Orange has built:
5: Basement
21: House
24: Tower
33: Basement
```

It prints the current builder's points and their residences from lowest to highest in location (vertex).

Board

Continuing from the game which was used to test residences, enter board as the command to print the current state of the board.

```
> board
                        | 0|--BR--|BB|
                          |      |
                          1   0   2
                          | GLASS |
                | 2|-- 3--|RB|   4  | 4|-- 4--|OB|
                  |      |      |      |
                  5   1   6      7   2   8
                  | PARK |      | WIFI |
        | 6|-- 9--| 7|      | 8|--10--| 9|   6  |10|--11--|BB|
          |      |      |      |      |      |      |      |
          12   3  13      14   4  15      16   5  17
          | ENERGY |      | ENERGY |      | HEAT |
        |12|   8  |13|--18--|14|   8  |15|--19--|16|   5  |17|
          |      |      |      |      |      |      |      |
          20      21   6  22      23   7  24      25
          |       | GLASS |      |      | BRICK |      |
        |18|--26--|19|   5  |20|--27--|OH|  11  |22|--28--|23|
          |      |      |      |      |      |      |      |
          29   8  30      31   9  BR      33  10  34
          | HEAT |      | WIFI |      | WIFI |
        |OT|   2  |25|--35--|RH|   3  |27|--36--|28|   4  |29|
          |      |      |      |      |      |      |      |
          37      38  11  39      40  12  41      42
          |       | HEAT |      |      | BRICK |      |
        |30|--43--|31|   9  |32|--44--|OB|  10  |YB|--45--|35|
          |      |      | GOOSE |      |      |      |      |
          46  13  47      48  14  49      50  15  51
          | WIFI |      | GLASS |      |      | BRICK |
        |BH|  11  |RT|--52--|38|  10  |39|--53--|40|   3  |41|
          |      |      |      |      |      |      |      |
          54      55  16  56      67  17  58      59
          |       | GLASS |      |      | HEAT |      |
        |42|--60--|43|   6  |44|--61--|45|   9  |46|--62--|47|
                  |      |      |      |      |
                  63      64  18  65      66
                  |       | WIFI |      |
                |48|--67--|49|   5  |YB|--68--|51|
                          |      |
                          69      70
                          |       |
                        |52|--71--|53|
```

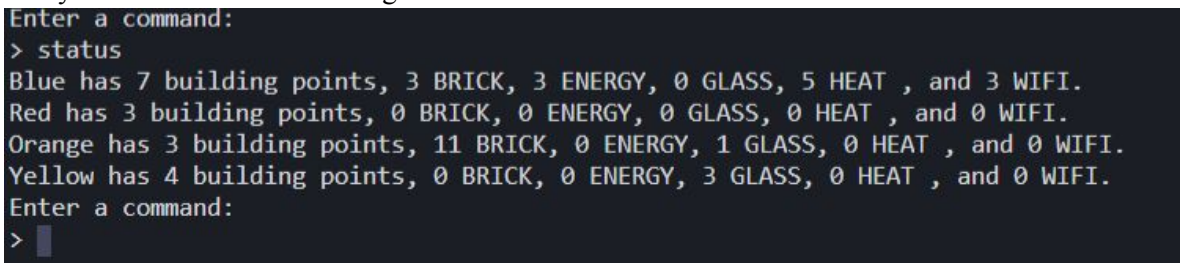
Status
[performs exact same task as the status command in the beginning of turn phase]

Trade
To test all the features of printing of board, exit the existing game and run the following:

$ *./constructor -load testSuite/trade.sv -seed 420*

You should see this.

From here, we'll roll to move to the action phase:
> roll
> 4
We should see Builder Orange get 1 Glass.
Try running
> status
And you should see the following statuses:



So here let's try some trades. Blue has no glass, so let's try … and fail, at trading glass for something else.
> trade orange glass brick

```
Enter a command:
> trade orange glass brick
You don't have enough GLASS.
Enter a command:
>
```

Ahhh, well that sucks. Let's try trading for something that someone else doesn't have! We have heat, so let's trade them that.
> trade orange heat energy

```
Enter a command:
> trade orange heat energy
Orange doesn't have enough ENERGY.
Enter a command:
>
```

Aww he had no energy. We did have heat though. Ok enough errors … let's make a trade happen! Try
> trade yellow heat glass

```
Enter a command:
> trade yellow heat glass
Does YELLOW accept the trade?
```

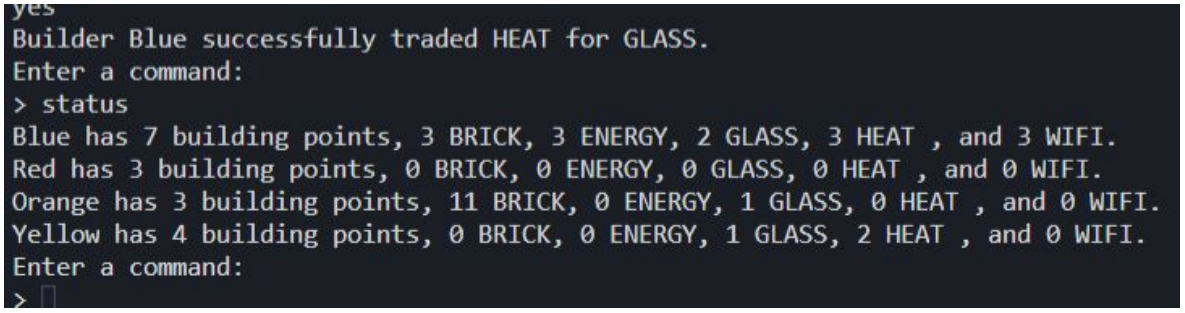
Yellow gets to choose to accept or decline. So let's accept
> yes

```
Does YELLOW accept the trade?
yes
Builder Blue successfully traded HEAT for GLASS.
Enter a command:
>
```

And if we check status, the trade succeeded.
> status

```
> status
Blue has 7 building points, 3 BRICK, 3 ENERGY, 1 GLASS, 4 HEAT , and 3 WIFI.
Red has 3 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT , and 0 WIFI.
Orange has 3 building points, 11 BRICK, 0 ENERGY, 1 GLASS, 0 HEAT , and 0 WIFI.
Yellow has 4 building points, 0 BRICK, 0 ENERGY, 2 GLASS, 1 HEAT , and 0 WIFI.
Enter a command:
>
```

Anywhere along the way, if an input was invalid after giving the trade command, our I/O will request another input until you give a valid one. If you reach end of file, it simply exits and goes to the standard auto save!
If you'd like to try it, try
> trade
Followed by any incorrect inputs along the way, or even ctrl-d to exit and auto save.



And check the status again to see the trade succeed! You can also decline by simply saying no after correct inputs are given.

```
Enter a command:
> status
Blue has 7 building points, 3 BRICK, 3 ENERGY, 2 GLASS, 3 HEAT , and 3 WIFI.
Red has 3 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT , and 0 WIFI.
Orange has 3 building points, 11 BRICK, 0 ENERGY, 1 GLASS, 0 HEAT , and 0 WIFI.
Yellow has 4 building points, 0 BRICK, 0 ENERGY, 1 GLASS, 2 HEAT , and 0 WIFI.
Enter a command:
> trade orange brick glass
Does ORANGE accept the trade?
no
ORANGE has declined the trade.
Enter a command:
> status
Blue has 7 building points, 3 BRICK, 3 ENERGY, 2 GLASS, 3 HEAT , and 3 WIFI.
Red has 3 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT , and 0 WIFI.
Orange has 3 building points, 11 BRICK, 0 ENERGY, 1 GLASS, 0 HEAT , and 0 WIFI.
Yellow has 4 building points, 0 BRICK, 0 ENERGY, 1 GLASS, 2 HEAT , and 0 WIFI.
Enter a command:
>
```

Multi Trade

To test all the features of printing of board, exit the existing game and run the following:

$ *./constructor -load ./testSuite/multiTrade.sv*

This is on Orange's turn. After we roll and print the status, we see that Orange attempts to trade 2 bricks for 3 glass from Yellow. If Yellow declines (oh no rejected), we see that the status does not change

```
Enter a command:
> status
Blue has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 1 HEAT , and 10 WIFI.
Red has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 1 WIFI.
Orange has 9 building points, 2 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 0 WIFI.
Yellow has 4 building points, 3 BRICK, 2 ENERGY, 5 GLASS, 1 HEAT , and 7 WIFI.
Enter a command:
> multi-trade yellow brick 2 glass 3
Does YELLOW accept the trade?
no
YELLOW has declined the trade.
Enter a command:
> status
Blue has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 1 HEAT , and 10 WIFI.
Red has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 1 WIFI.
Orange has 9 building points, 2 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 0 WIFI.
Yellow has 4 building points, 3 BRICK, 2 ENERGY, 5 GLASS, 1 HEAT , and 7 WIFI.
Enter a command:
```

Way to go Orange, let's try again. This time Yellow has accepted the offer, and we see that the corresponding number of resources for Orange and Yellow has changed.

```
Enter a command:
> multi-trade yellow brick 2 glass 3
Does YELLOW accept the trade?
yes
Builder Orange successfully traded 2 BRICK for 3 GLASS.
Enter a command:
> status
Blue has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 1 HEAT , and 10 WIFI.
Red has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 1 WIFI.
Orange has 9 building points, 0 BRICK, 10 ENERGY, 13 GLASS, 10 HEAT , and 0 WIFI.
Yellow has 4 building points, 5 BRICK, 2 ENERGY, 2 GLASS, 1 HEAT , and 7 WIFI.
Enter a command:
>
```

When would the trade fail? Here, Orange has 0 bricks but offers 2 bricks for 1 glass, so an error is produced saying not enough resources. Also following, Orange tries to get 2 heat from Yellow but Yellow only has 1 heat. Here an error command is also produced. No trade happens

```
Enter a command:
> multi-trade yellow brick 2 glass 1
You don't have enough BRICK.
Enter a command:
> multi-trade yellow energy 4 heat 2
Yellow doesn't have enough HEAT.
Enter a command:
>
```

Market Trade

To test all the features of printing of board, exit the existing game and run the following:

$ *./constructor -load ./testSuite/marketTrade.sv*

This is now on Orange's turn. If we roll and view status, we see Orange's resources. If we try to market trade bricks for a wifi, we see that this fails because Orange does not have 4 bricks

```
Enter a command:
> status
Blue has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 10 WIFI.
Red has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 1 WIFI.
Orange has 9 building points, 2 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 0 WIFI.
Yellow has 4 building points, 3 BRICK, 2 ENERGY, 5 GLASS, 1 HEAT , and 7 WIFI.
Enter a command:
> market-trade brick wifi
You do not have enough resources.
Enter a command:
```

Now we try to market trade energy for wifi and glass for wifi (because we all need wifi:)) we see that they succeed, Orange lost 4 energy and for glass, in exchange for 2 wifi

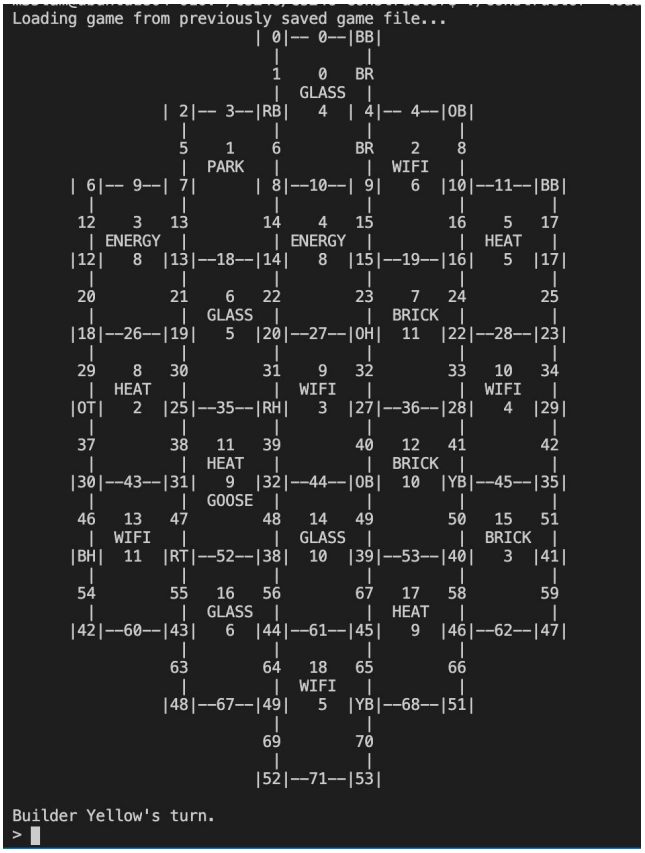
```
Enter a command:
> market-trade energy wifi
Builder Orange successfully traded 4 ENERGY for 1 WIFI.
Enter a command:
> market-trade glass wifi
Builder Orange successfully traded 4 GLASS for 1 WIFI.
Enter a command:
> status
Blue has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 10 WIFI.
Red has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 1 WIFI.
Orange has 9 building points, 2 BRICK, 6 ENERGY, 6 GLASS, 10 HEAT , and 2 WIFI.
Yellow has 4 building points, 3 BRICK, 2 ENERGY, 5 GLASS, 1 HEAT , and 7 WIFI.
Enter a command:
>
```

Build-res

To test all the features of printing of board, exit the existing game and run the following:

$ *./constructor -load testSuite/buildResidence.sv*

You should get the following output:



From here we roll (on loaded dice) and enter 2 in order to move to the action phase:
If we enter status, we notice that the current builder Yellow has no resources. When we try building a basement we get the following:

```
Enter a command:
> build-res 46
You do not have enough resources.
```

Since builder yellow can't do anything due to his lack of resources, enter next and move to the next builder's turn (blue). Once again we will roll (on loaded dice) and enter 2 in order to move to the action phase: When we try building a basement on vertex 43 we get the following:

```
Enter a command:
> build-res 43
ERROR: You cannot build here.
```

We must pick a vertex which is connected to a road owned by the current builder with all immediately adjacent vertices empty. Let's try building a basement on vertex 9.

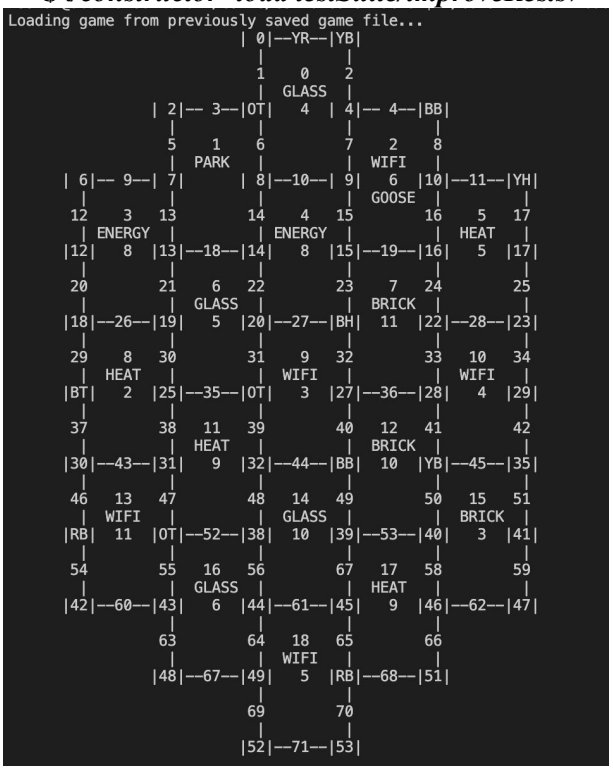
```
Enter a command:
> status
Blue has 4 building points, 1 BRICK, 2 ENERGY, 3 GLASS, 4 HEAT , and 5 WIFI.
Red has 6 building points, 0 BRICK, 1 ENERGY, 2 GLASS, 2 HEAT , and 3 WIFI.
Orange has 7 building points, 10 BRICK, 9 ENERGY, 8 GLASS, 13 HEAT , and 6 WIFI.
Yellow has 2 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT , and 11 WIFI.
Enter a command:
> build-res 9
what happened
Builder Blue successfully built a Basement at 9.
Enter a command:
> status
Blue has 5 building points, 0 BRICK, 1 ENERGY, 2 GLASS, 4 HEAT , and 4 WIFI.
Red has 6 building points, 0 BRICK, 1 ENERGY, 2 GLASS, 2 HEAT , and 3 WIFI.
Orange has 7 building points, 10 BRICK, 9 ENERGY, 8 GLASS, 13 HEAT , and 6 WIFI.
Yellow has 2 building points, 0 BRICK, 0 ENERGY, 0 GLASS, 0 HEAT , and 11 WIFI.
```

Notice that the use of status was used to show that the correct resources were taken away from player Blue when building his new basement. Furthermore you can build multiple residences per turn (as long as all the previous conditions mentioned prior are satisfied).

Improve-res

To test all the features of printing of board, exit the existing game and run the following to load the saved game:

$ *./constructor -load testSuite/improveRes.sv*



(my terminal is small so not the whole board fits but it should be all there)

Now it's Orange's turn. We see that orange has sufficient resources to improve any type of residence. We roll and try to build at 3 where there is already a tower, at whre Orange has no building, and at 5 where the building is not Orange's. In all these cases, we see an error message.

```
Enter a command:
> status
Blue has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 13 HEAT , and 10 WIFI.
Red has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 1 WIFI.
Orange has 9 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 10 WIFI.
Yellow has 4 building points, 3 BRICK, 2 ENERGY, 5 GLASS, 1 HEAT , and 7 WIFI.
Enter a command:
> improve-res 3
ERROR: This residence is already a tower.
Enter a command:
> improve-res 4
ERROR: You do not own this residence.
Enter a command:
> improve-res 5
ERROR: You do not own this residence.
```

Now on Yellow's turn after the roll, when we check status, we see that Yellow does not have enough resources to upgrade. Hence we try to upgrade at a valid location and an error message is produced as follows

```
Enter a command:
> status
Blue has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 13 HEAT , and 10 WIFI.
Red has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 1 HEAT , and 1 WIFI.
Orange has 9 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 16 HEAT , and 10 WIFI.
Yellow has 4 building points, 3 BRICK, 2 ENERGY, 5 GLASS, 1 HEAT , and 7 WIFI.
Enter a command:
> improve-res 1
You do not have enough resources.
Enter a command:
```

Now onto Blue's turn. We see that Blue has enough resources. After the roll, Blue enters two locations it wants to improve (5 and 21 where Blue has a residence and they are now towers) and they successfully improved. Blue can also improve the same residence again in the same round (5 again)

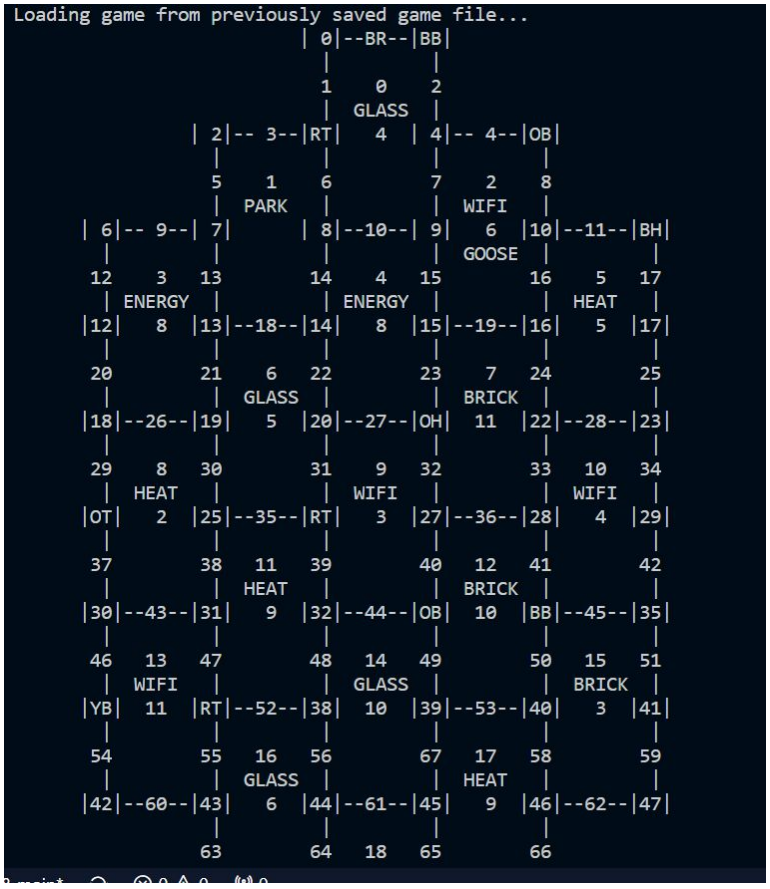
```
Enter a command:
> improve-res 5
Builder Blue successfully built a House at 5.
Enter a command:
> improve-res 21
Builder Blue successfully built a Tower at 21.
Enter a command:
> improve-res 5
Builder Blue successfully built a Tower at 5.
Enter a command:
> improve-res 33
```

Build-road

To test all the features of printing of board, exit the existing game and run the following:

$ *./constructor -load testSuite/buildRoad.sv*

You should get the following output:



Now it is blue's turn, and if we roll 5 and check everyone's status,

```
> roll 5
Input a roll between 2 and 12:
5 is rolled.
Builder Blue got:
3 GLASS
Builder Red got:
2 HEAT
Builder Yellow got:
1 WIFI
Enter a command:
> status
Blue has 9 building points, 0 BRICK, 1 ENERGY, 5 GLASS, 2 HEAT , and 0 WIFI.
Red has 4 building points, 6 BRICK, 5 ENERGY, 4 GLASS, 4 HEAT , and 5 WIFI.
Orange has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 10 WIFI.
Yellow has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 2 WIFI.
Enter a command:
```

Blue has 2 heat and 0 wifi. If we try to build a road at either a valid or invalid location, an error is shown (not enough resources)

```
Enter a command:
> build-road 3
ERROR: You do not have enough resources.
Enter a command:
> build-road 38
ERROR: You do not have enough resources.
Enter a command:
>
```

Moving onto Red's turn. Red has enough resources and tries to build somewhere (4) where no road or building connects to. An error message is shown

```
Builder Red's turn.
> roll 4
Input a roll between 2 and 12:
4 is rolled.
Builder Blue got:
3 GLASS
Builder Red got:
1 GLASS
1 WIFI
Enter a command:
> build-road 4
ERROR: Cannot build road here. None of your road or residence connect to here.
Enter a command:
>
```

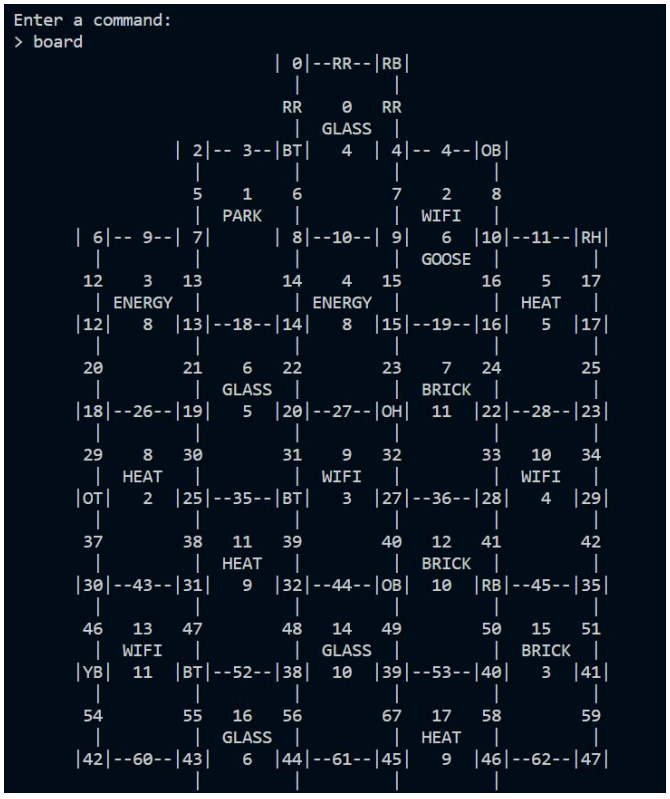
Now on Red's turn, we check the resources and build on a road where there is a building of Red connected to it. Here the road is successfully build and the cost (one heat and one wifi) is subtracted from Red's resources.

```
Enter a command:
> status
Blue has 9 building points, 0 BRICK, 1 ENERGY, 8 GLASS, 2 HEAT , and 0 WIFI.
Red has 4 building points, 6 BRICK, 5 ENERGY, 5 GLASS, 4 HEAT , and 6 WIFI.
Orange has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 10 WIFI.
Yellow has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 2 WIFI.
Enter a command:
> build-road 2
Builder Red successfully built a road at 2.
Enter a command:
> status
Blue has 9 building points, 0 BRICK, 1 ENERGY, 8 GLASS, 2 HEAT , and 0 WIFI.
Red has 4 building points, 6 BRICK, 5 ENERGY, 5 GLASS, 3 HEAT , and 5 WIFI.
Orange has 7 building points, 10 BRICK, 10 ENERGY, 10 GLASS, 10 HEAT , and 10 WIFI.
Yellow has 2 building points, 3 BRICK, 2 ENERGY, 1 GLASS, 0 HEAT , and 2 WIFI.
Enter a command:
>
```

Let's try again. We build where there is a road of Red connected to it. We build successfully again

```
Enter a command:
> build-road 1
Builder Red successfully built a road at 1.
Enter a command:
```

When we view the board again, we see that roads 1 and 2 are now belonging to Red. We also see that roads 3 and 38 does not belong to Blue and 4 did not belong to Red because the build-road failed from the errors



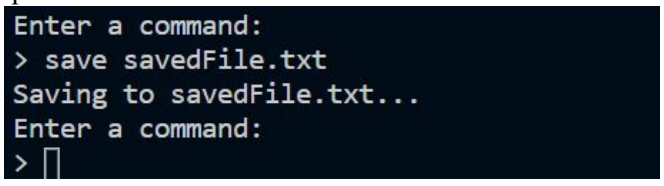
Next

Moves onto the next Builder/Player/Color's turn. Next can only be a command after the person has rolled, and if rolled the geese, after appropriate geese stealing/placement has been read. The board is printed at this command to allow the players see the status of the board.

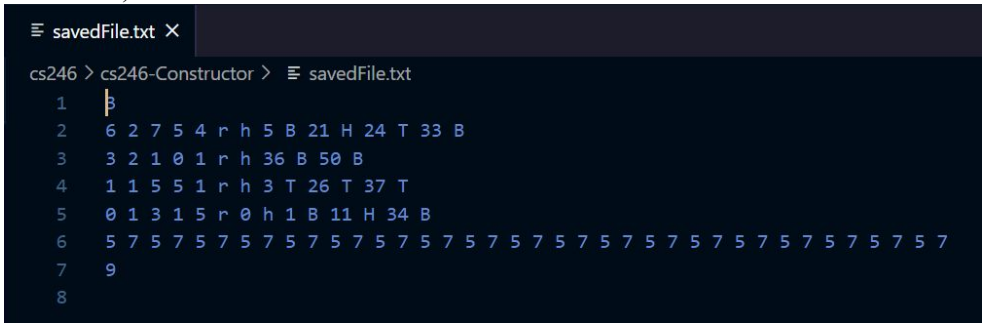
For examples to show that 'next' functions, refer to the examples above which used this command to test other features on our game.

Save

Saves the game status (turn information, player info, board info, geese info) into the file at the specified file name.



If we open the saved file this should agree with the current status of the game we were just on (from build-road)



Save with no command line argument will just save to the default file which is backup.sv

Similarly, if ever you reach end of file, the game automatically saves to backup.sv

```
Builder Yellow's turn.
> roll
Input a roll between 2 and 12:
End of file reached.
Saving to backup.sv...
```

Regardless of what action you might be doing at the time.

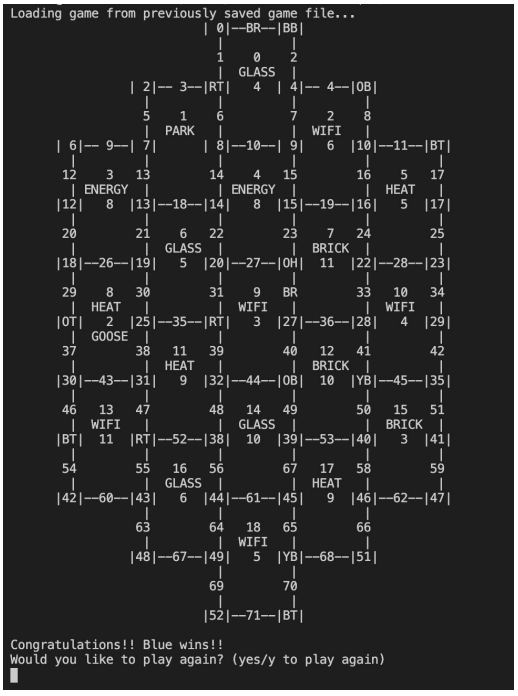
If command is unrecognized:

```
> buy
Invalid command.
```

A few extra edge cases that our code considers:

1) What happens if we try loading in a game that already has a winner?

Run the following command: $ ./constructor -load ./testSuite/backupWon.sv



In this situation, the game found that a winner already existed and prompts the user if they want to play a new game.