CS246 Final Project: Constructor
Demo

Jim Liu
Shuchen (Mason) Liu
Yuetong (Ivy) Jiang

Before Playing…

The files to be used for the game:

ctor:	Executable program to run the game
layout.txt:	The file which contains a default board used to initialize a board
backup.txt:	A valid game state saved from the previous games. Used to load a game.

Set up a board:

Setting up a game through the command line arguments below:

Command	Description
`-seed xxx`	Sets the random number generator's seed to xxx. This seed will be applied all the way through the program. xxx should be an integer.
`-load xxx`	Load an exist valid game saved in file xxx. Assume the file loaded contains valid game states.
`-board xxx`	Create a new game by the board read from file xxx. File xxx doesn't need to be assumed with any validity
`-random-board`	Create a new game with a randomly generated board.
`-customize`	Customize the game with special features

◊ The commands can be entered in any combinations, while -board and -load are contradict. When those commands appear at the same time, only the last argument will be used. -random-board is ignored when either -board or -load appeared.

```
@ubuntu2004-008% ./ctor -load backup.txt -board layout.txt
>  Generated by a new game with board given in layout.txt
```

```
@ubuntu2004-008% ./ctor -load backup.txt -board layout.txt -random-board
>  Generated by a new game with board given in layout.txt
```

◊ The commands can be entered with repetitions. However, only the last argument of the same type is applied. For example:

```
@ubuntu2004-008% ./ctor -seed 231 -seed 246 -board layout.txt -seed 230
>  Generated by a new game with board given in layout.txt
>  Seed for random number generator is 230
```

◊ If no commands were given, then the program will generate a new board with default layout, which is read from layout.txt

```
@ubuntu2004-008% ./ctor
>  Generate a new game by the default board
```

◊ All errors can be handled:

```
@ubuntu2004-008% ./ctor -load backup.txt -board
ERROR: Commands for -board should be provided!
```

-> Need file for -board

```
@ubuntu2004-008% ./ctor -seed backup.txt
Usage: Seed should be Integer!
```

-> Need a number for -seed

```
@ubuntu2004-008% ./ctor -unkonwn -command
ERROR: Invalid command line arguments!
```

-> Unknown flags

◊ **To enable extra features**, the user can enter -customize, then the customizing questions will be prompted:

```
>  Customize the Game:
>  Enter building points to win the game: (integer between 3-15)
5
>  Do you want to have geese in the game? (yes/no)
no
>  Do you want to get building suggestions in each turn? (yes/no)
yes
>  Completed!
```

1) Enter the building points to win the game.
   If you want to play a quick game, then enter a smaller value. The value should between 3 to 15, since initially everyone has 2 basements, 2 building points.
2) Do you want to have geese involved in the game?
   Geese will steal resources when dice rolled with 7. While you can remove geese from the game, and nothing will happen (include losing resources, moving geese, and steal from others) when dice rolled to 7.
3) Do you want to get building suggestions in each turn?
   If you entered yes, a building suggestion will show up after each player rolled the dice. Suggestion are determined by the recourses available to the current player.

```
>  Builder Red's turn.
roll
>  Input a dice value between 2 to 12 (inclusive):
5
>  You have rolled: 5
>  Based on your current resources, actions avaiable are:
>          build-road <edge#>: Build a Road;
>          build-res <housing#>: Build a Basement;
>          improve <housing#>: Improve a Basement to House;
>          trade <colour> <give> <take>: Trade with other players;
>          next: Pass control onto the next builder;
>  To list all commands, please enter 'help'
status
>  Builder Blue     has  2 building points,  1 BRICK,  5 ENERGY,  3 GLASS,  0 HEAT,  1 WIFI.
>  Builder Red      has  2 building points,  1 BRICK,  5 ENERGY,  2 GLASS,  5 HEAT,  1 WIFI.
>  Builder Orange   has  2 building points,  1 BRICK,  0 ENERGY,  4 GLASS,  0 HEAT,  2 WIFI.
>  Builder Yellow   has  2 building points,  3 BRICK,  3 ENERGY,  2 GLASS,  0 HEAT,  0 WIFI.
```

Start the Game!

Phase 1: Setting Up Two Initial Buildings (optional)

If you are using a new board and play a new game, all the players need to set up their first 2 basements on the board.

If you continue to play a previous game, then we skip this phase and jump to the second phase.

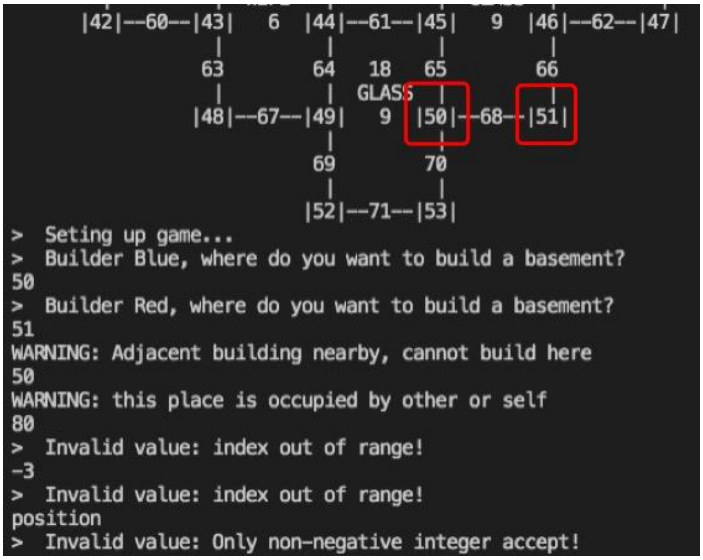
To set up:

Players are ordered in the sequence:
`Blue -> Red -> Orange -> Yellow -> Yellow -> Orange -> Red -> Blue`

Note:

- ◊ Each time only one position can be chosen.
- ◊ Based on the game rules, players cannot build a residence adjacent to the other residences, or on the position which has already been occupied.

For example, the cases below are not permitted:



50 and 51 are adjacent

-> Blue built on position 50
-> Red cannot built since adjacent
-> Red cannot build since repetition
-> Red cannot build since `pos>53`
-> Red cannot build since `pos<0`
-> Invalid inputs

Phase 2: Play the Game:

Turn Order:

◊ Start with a new game: Once the first eight residences are set, it will be Blue builder's turn, followed by Red, then Orange and then Yellow. This sequence repeats until the game ends.

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

◊ Continue with an exist game: start from the turn recorded in the state file, and then follows the same loop

```
>  Start!
>  Builder Red's turn.
```

1. Roll Dice (Mandatory)

During each turn, the player roll a dice to gain resources.

The available commands are:

commands	Description
`load`	The player begins to use a load dice
`fair`	The player begins to use a fair dice
`roll`	Roll the dice. If using load dice, the builder then need to enter a value between 2 to 12 manually;
`save <file>`	Save the game to the target file and then quit the game
`quit`	Quit the game without saving

Note:

- Each player has a own separate dice. All players are initially using loaded dice
- If using fair dice, dice value is generated randomly based on the seed. If you didn't provide such a seed in command line, then seed 0 will be applied.

Examples:

Using load dice:

```
>  Builder Red's turn.
roll
>  Input the sum of two dices, value between 2 to 12 (inclusive):
11
>  You have rolled: 11
Builder Bluegained: 1 WIFI 2 GLASS
Builder Orangegained: 1 WIFI
Builder Yellowgained: 1 GLASS
```

Using fair dice:

```
>  Builder Orange's turn.
fair
>  Player Orange uses fair dice now
roll
>  You have rolled: 3
Builder Bluegained: 5 HEAT
Builder Redgained: 1 HEAT
```

Invalid Input Examples:

```
>  Builder Red's turn.
next
>  Please first roll the Dice. Remember to enter 'fair' or 'load' when needed
load
>  Player Red uses loaded dice now
8
>  Please first roll the Dice. Remember to enter 'fair' or 'load' when needed
roll
>  Input a dice value between 2 to 12 (inclusive):
1
>  Invalid value: index out of range!
>  Please re-enter the dice value: integer between 2-12 (inclusive)
3
>  You have rolled: 3
```

Invalid command, should be either "roll" "load" "fair"

Should first enter "roll" then the number;
Should input value between 2 – 12;

Move Geese

When 7 is rolled, we have a sequence of actions related to the geese:

1) Builders who have 10 or more resources will random lose half of them (round down).

```
>  Builder Yellow's turn.
roll
>  Input a dice value between 2 to 12 (inclusive):
7
>  You have rolled: 7
>  Geese attack!
>  Builder B loses 7 resources to the geese. They lose:
>  4 HEAT
>  2 WIFI
>  1 GLASS
>  Builder O loses 7 resources to the geese. They lose:
>  4 HEAT
>  2 ENERGY
>  1 BRICK
```

2) The current builder can move the geese to <u>any other</u> tiles

```
>  Choose where to place the Geese. They are on tile 2 now.
2
>  Geese should be moved to the tile not previously on.
70
>  Invalid value: index out of range!
8
>  Builder Red has no builders to steal from.
```

Cannot move to the same tile

Out of range

3) The current builder can steal one resources from one of the builders who has residences on the target tile and have non-zero resources

   Note, you should explicitly enter the builder's name, for example, `"Red"`, `"Yellow"`, `"Blue"`, instead of inputting `"red"`, `"R"`, `"BLUE"`

```
>  Builder Yellow has no builders to steal from.
```

The position either has no residences or other players have no resources to steal.

```
>  Builder Blue can choose to steal from Red, Yellow.
>  Choose a builder to steal from.
Orange
>  Please select a builder mentioned above.
Red
>  Builder Blue steals HEAT from builder Red.
```

Can only steal from one of the builders Listed.

2. Using Resources

After rolling dice, now you can use your resources. The commands available are:

Commands	Description
`board`	Prints the current board
`status`	Prints the current status of all four players
`residences`	Prints the current building status of the current player
`build-road <road#>`	Build a road at number <road#>
`build-res <housing#>`	Build a basement at number <housing#>

`improve <housing#>`	Improve the residence at number <housing#>
`trade <colour> <give> <take>`	Trade with builder <colour>, giving one <give>, for one <take>
`next`	Pass the control onto the next builder
`save <file>`	Save the current game state to file <file>
`quit`	Quit the game without saving
`help`	List all commands
`suggestion`	List all possible commands for the current builder according to the player's status

- **Build Residence: build-res <housing#>**

  You can build a residence at <housing#> if the following conditions are all met:

  i) There are no other residences adjacent to the position you want to build

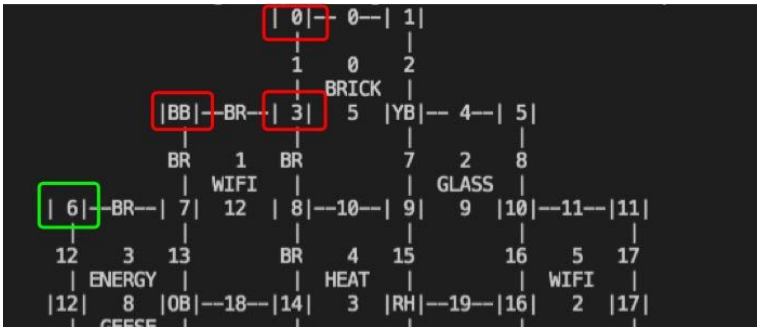
  ii) There is at least one road built by the current builder, which is connected to the position you want to build

  iii) The position hasn't been occupied

  iv) You have enough resources

  v) The index given as <housing#> is a non-negative integer within the range of the board (i.e. 0-53)

Example: Builder Blue's turn.



Position 2 is a basement built by Blue;

Position 0 1 6 are all empty;

```
build-res 0
>  No adjacent same color road nearby, cannot build here
build-res 2
WARNING: this place is occupied by other or self
build-res 3
>  Adjacent building nearby, cannot build here
build-res 70
>  Invalid value: index out of range!
build-res 6
>  Player Blue built Basement 6
```

0 has no roads connected;
2 is occupied by Blue;
3 is adjacent to 2;
70 is out of bound;

6 is valid!

```
status
>  Builder Blue     has  8 building points,  6 BRICK,  4 ENERGY,  6 GLASS,  7 HEAT,  0 WIFI.
>  Builder Red      has  5 building points,  2 BRICK,  4 ENERGY,  1 GLASS,  2 HEAT,  2 WIFI.
>  Builder Orange   has  2 building points,  1 BRICK,  5 ENERGY,  0 GLASS,  8 HEAT,  4 WIFI.
>  Builder Yellow   has  2 building points,  0 BRICK,  2 ENERGY,  6 GLASS,  0 HEAT,  0 WIFI.
build-res 0
WARNING: resource not enough for building residence
```

Blue need one more WIFI to build a basement

- **Build Roads: build-road <road#>**

You can build a road at <road#> if the following conditions are all met:

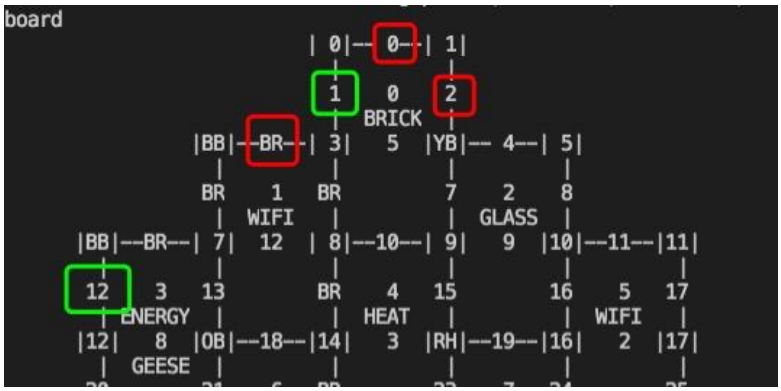
i) There is at least one road or residence built by the current builder, which is connected to the position you want to build

ii) The position hasn't been occupied

iii) You have enough resources

iv) The index given as <road#> is a non-negative integer within the range of the board (i.e. 0-71)

Demonstration: Builder Blue's turn.



Road 0 is not connected to any other roads or residences;

Road 1 connected to road 3 built by Blue;

Road 2 connected to basement 4 built by Yellow;

Road 3 is built by Blue;

Road 12 is connected to basement 6 built by Blue;

```
build-road 100
>  Invalid value: index out of range!
build-road 0
>  No adjacent same color road or vertice nearby, cannot build here
build-road 3
WARNING: this road is occupied by other or self
build-road 1
>  Player Blue built road 1
build-road 12
>  Player Blue built road 12
status
>  Builder Blue     has  9 building points,  4 BRICK,  3 ENERGY,  5 GLASS,  3 HEAT,  0 WIFI.
>  Builder Red      has  5 building points,  2 BRICK,  4 ENERGY,  1 GLASS,  2 HEAT,  2 WIFI.
>  Builder Orange   has  2 building points,  2 BRICK,  5 ENERGY,  0 GLASS, 10 HEAT,  1 WIFI.
>  Builder Yellow   has  2 building points,  0 BRICK,  2 ENERGY,  6 GLASS,  0 HEAT,  0 WIFI.
build-road 2
WARNING: resource not enough for building road
```

To build road 100: 100 is out of bound;

To build road 0: 0 is not connected to any other roads or residences;

To build road 3: 3 is already built by Blue;

To build Road 1: Valid! Since it connected to road 3, which built by Blue;

To build Road 12: Valid! Since connected to house 6, which built by Blue;

To build Road 2: Lack of resources WIFI to build a road;

- **Improve Residence: improve <housing#>**

  You can improve a residence at <housing#> if the following conditions are all met:

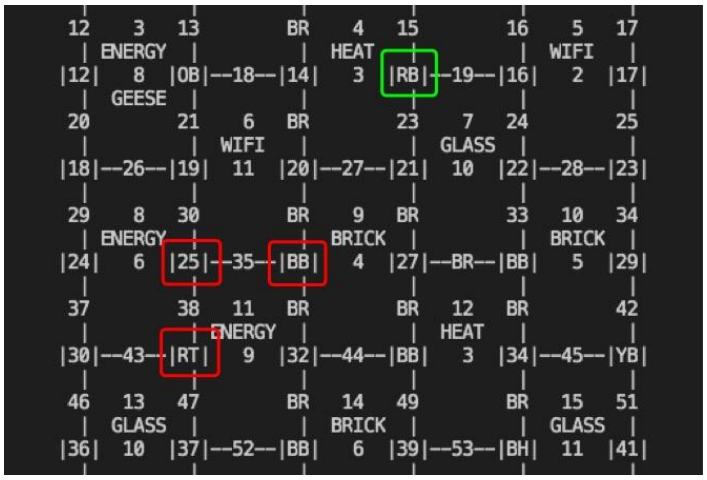
  i) The position has a residence built by the current player, and it's not a Tower

  ii) You have enough resources

  iii) The index given as <housing#> is a non-negative integer within the range of the board (i.e. 0-53)

  If the player has improved successfully, then one building point will be assigned.

  Demonstration: Builder Red's turn.



Position 15 is a basement built by Red

Position 25 is empty, has no owner

Position 26 is a basement built by Blue

Position 31 is a Tower built by Red

```
status
>  Builder Blue     has  8 building points,  0 BRICK,  1 ENERGY,  3 GLASS,  7 HEAT,  2 WIFI.
>  Builder Red      has  4 building points,  2 BRICK,  3 ENERGY,  2 GLASS,  4 HEAT,  2 WIFI.
>  Builder Orange   has  2 building points,  2 BRICK,  4 ENERGY,  1 GLASS,  7 HEAT,  0 WIFI.
>  Builder Yellow   has  2 building points,  0 BRICK,  1 ENERGY,  6 GLASS,  0 HEAT,  0 WIFI.
residences
>  Red has built:
15 B
31 T
improve 70
>  Invalid value: index out of range!
improve 31
WARNING: cannot improve a top level building
improve 26
WARNING: this place is not occupied by current player
improve 25
WARNING: this place is not occupied by current player
improve 15
>  Player Red improved residence 15
status
>  Builder Blue     has  8 building points,  0 BRICK,  1 ENERGY,  3 GLASS,  7 HEAT,  2 WIFI.
>  Builder Red      has  5 building points,  2 BRICK,  3 ENERGY,  0 GLASS,  1 HEAT,  2 WIFI.
>  Builder Orange   has  2 building points,  2 BRICK,  4 ENERGY,  1 GLASS,  7 HEAT,  0 WIFI.
>  Builder Yellow   has  2 building points,  0 BRICK,  1 ENERGY,  6 GLASS,  0 HEAT,  0 WIFI.
improve 15
WARNING: resource not enough for improving residence
residences
>  Red has built:
15 H
31 T
```

-> Red has 4 points

-> Red has two buildings

-> 70 is out of bound

-> 31 is already a Tower

-> 26 is built by Blue

-> 25 has no buildings

-> 15 is a valid!

-> Red has 5 points now

-> not enough resources to improve 15 to a Tower

Finally Red has one House, one Tower.

- **Trade with Other Players: trade <colour> <give> <take>**

  You can trade with other players if you need any specific resources.

  Input that player's colour, the recourse you provide, and the resources you want.

  Note:

  - Make sure you have enough <give> type resources, and builder <colour> has enough <take> type resources.
  - Each time only one resources can be traded.
  - The request will be prompted to the builder <colour>, who should either input "yes" or "no" (accept any case)

  Demonstration: Builder Red's turn:

```
status
>  Builder Blue     has  8 building points,  5 BRICK,  1 ENERGY,  4 GLASS,  9 HEAT,  2 WIFI.
>  Builder Red      has  5 building points,  2 BRICK,  3 ENERGY,  5 GLASS,  0 HEAT,  2 WIFI.
>  Builder Orange   has  2 building points,  2 BRICK,  4 ENERGY,  0 GLASS, 10 HEAT,  0 WIFI.
>  Builder Yellow   has  2 building points,  2 BRICK,  1 ENERGY,  6 GLASS,  0 HEAT,  0 WIFI.
trade Orange Heat Brick
>  You do not have enough resources.
trade Orange Energy Glass
>  Orange do not have enough resources.
trade Purple Glass Heat
>  Invalid player's name color!
trade Orange Tree Glass
>  Invalid resources type!
```

-> Red doesn't have Heat
-> Orange doesn't have Glass
-> Purple is not a valid player
-> Tree is not a valid resource

```
trade Blue Energy Heat
>  Red offers Blue one ENERGY for one HEAT.
>  Does Blue accept this offer?   yes/no (accept any case)
yes
>  Successfully traded!
```

-> Successfully traded

Extra Feature:

- It's a user friendly recipe that you only need to make sure the first letter of <colour>, <give>, and <take> is correct, either in upper case or the lower case. Any typos will be automatically corrected. For example, "blue", "Re", "Gless", "Hee" will be corrected as "Blue", "Red", "Glass", "Heat".
- Also, the response from builder <colour> can be any case of "yes" or "no".

```
trade orange Glasses Energ
>  Red offers Orange one GLASS for one ENERGY.
>  Does Orange accept this offer?   yes/no (accept any case)
Yes
>  Successfully traded!
```

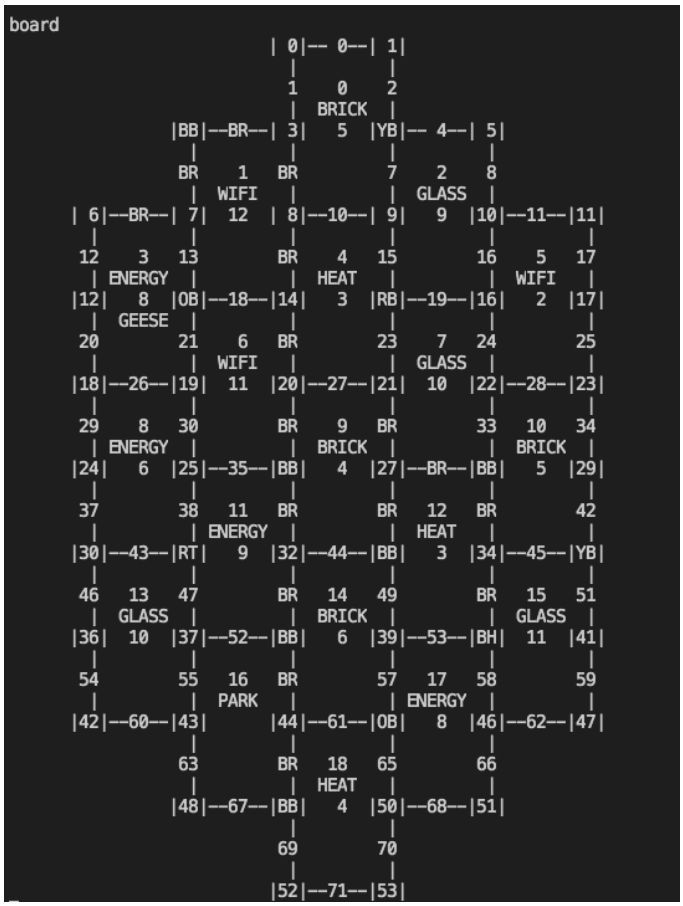
```
trade Yep Energy Glass
>  Red offers Yellow one ENERGY for one GLASS.
>  Does Yellow accept this offer?   yes/no (accept any case)
NO
>  Trade request was refused!
```

```
trade O G H
>  Red offers Orange one GLASS for one HEAT.
>  Does Orange accept this offer?   yes/no (accept any case)
YES
>  Successfully traded!
```

3. Print Information

- Print Board: board

You can check the latest board by this command



- Print Status: status

You can check the current status of all players by this command

```
status
>  Builder Blue     has  8 building points,  5 BRICK,  5 ENERGY,  4 GLASS, 12 HEAT,  2 WIFI.
>  Builder Red      has  4 building points,  2 BRICK,  8 ENERGY,  8 GLASS,  5 HEAT,  2 WIFI.
>  Builder Orange   has  2 building points,  3 BRICK,  6 ENERGY,  0 GLASS,  8 HEAT,  0 WIFI.
>  Builder Yellow   has  2 building points,  2 BRICK,  1 ENERGY,  7 GLASS,  0 HEAT,  0 WIFI.
```

- Print residences: residences

You can check the current player's building status by this command.

Note, it will only print out the buildings you built, no roads.

```
>  Blue has built:
 2 B
26 B
28 B
33 B
38 B
40 H
49 B
```

- **Next: next**

  You can pass the control to the next player, then prompt out the next player's turn.

  Demonstration:

```
next
>  Builder Orange's turn.
```

4. **Helps**
   - **Help: help**

     You can check all possible commands by it.

```
help
>  Valid commands:
>    board
>    status
>    residences
>    build-road <edge#>    e.g. build-road 7
>    build-res <housing#>    e.g. build-res 25
>    improve <housing#>    e.g. improve 25
>    trade <colour> <give> <take>    Note: make sure the first letter is capitalized: trade Orange Brick HEAT
>    next
>    save <file>    e.g. save backup.sv
>    help
```

   - **{Extra Feature} Suggestion: suggestion**

     You can check all available actions based on your current status (resources).

```
suggestion
>  Based on your current resources, actions avaiable are:
>          build-road <edge#>: Build a Road;
>          build-res <housing#>: Build a Basement;
>          improve <housing#>: Improve a Basement to House;
>          trade <colour> <give> <take>: Trade with other players;
>          next: Pass control onto the next builder;
>  To list all commands, please enter 'help'
```

Phase 3: Save/Quit/Won the Game

Save the Game

You can save the game easily by calling command: `save <file>` whenever before rolling the dice or after. `<file>` can be any valid strings.

Demonstration:

After rolling:

```
>  You have rolled: 3
Builder Blue gained: 3 HEAT
Builder Red gained: 1 HEAT
save backup.sv
>  Saved in backup.sv !
@ubuntu2004-008%
```

Before rolling:

```
>  Builder Yellow's turn.
save backup.sv
>  Saved in backup.sv !
```

Quit the Game

You can quit the game whenever you want (both before or after rolling), by either enter `quit`, or EOF.

The difference is, command quit will quit the game directly without making backups or saving.

Before rolling

```
>  Builder Yellow's turn.
quit
>  Quit the game.
@ubuntu2004-008%
```

After rolling:

```
>  You have rolled: 3
Builder Blue gained: 3 HEAT
Builder Red gained: 1 HEAT
quit
>  Quit the game.
@ubuntu2004-008%
```

But at any point when the program ask for a input, if you press ctrl-D, then a backup will be automatically saved in backup.sv:

```
>  Saved in backup.sv !
```

Won the Game

Any player who gets more than 10 building points (can be customized) will win the game.

Then a message will be prompted to ask if you would like to play again.

You can either enter “yes” or “no” (or EOF).

- ◊ yes: start with the next game
- ◊ no: quit the game without saving the game
- ◊ EOF: quit the game, while save it in backup.sv

```
status
>  Builder Blue     has  9 building points,  0 BRICK,  3 ENERGY,  3 GLASS,  3 HEAT,  2 WIFI.
>  Builder Red      has  2 building points,  0 BRICK,  2 ENERGY,  3 GLASS,  3 HEAT,  0 WIFI.
>  Builder Orange   has  2 building points,  2 BRICK,  6 ENERGY,  0 GLASS,  7 HEAT,  0 WIFI.
>  Builder Yellow   has  2 building points,  0 BRICK,  1 ENERGY,  6 GLASS,  0 HEAT,  0 WIFI.
improve 49
>  Player Blue improved residence 49
>  Builder Blue won the game!
>  Would you like to play again? (yes/no)
no
@ubuntu2004-002%
```

Note that, if you were playing with a loaded game, and entered “yes”, then the next game will begin with the state received from the given file. In other words, same game state as the beginning of your first game.

For example:

We load a game “yellowWon.txt”, which starts from Builder Yellow, and Yellow has 9 building points. Then after anyone won this game, the next game will be created from “yellowWon.txt” as well, and starts from Yellow, who have 9 building points already.

Extra Features

Extra feature which have been mentioned in previous steps are coloured in red.
This section will explain all extra features thoroughly.

To enable the extra features by command line flag: `-customize`

1. Smart Pointers

We have applied smart pointers whenever we want to define a heap allocated object, which successfully avoided memory leaks.
For example in class Board, we used shared pointers within vector of `Tile`, vector of `Vertex`, vector of `Edge`, vector of `Player`.

```
class Board {
    int posGeese = 7 ;
    int seed = -1;
    std::vector<std::shared_ptr<Tile>> tiles;
    std::vector<std::shared_ptr<Vertex>> vertices;
    std::vector<std::shared_ptr<Edge>> edges;
    std::vector<std::shared_ptr<Player>> players;
    int curTurn = 0;
    int winPoints = 10;
```

For example in class Dice, we used shared pointers for a heap allocated `Strategy`

```
class Dice {
    std::shared_ptr<Strategy> strategy;
    int dicePoint;
    int seed;
    public:
    Dice(std::shared_ptr<Strategy> strategy, int dicePoint = 0, int seed = 0);
    void setStrategy(std::shared_ptr<Strategy> strategy);
    void roll();
    void setPoint(int value);
    int getPoint();
};
```

2. Customized Game

```
>  Customize the Game:
>  Enter building points to win the game: (integer between 3-15)
5
>  Do you want to have geese in the game? (yes/no)
no
>  Do you want to get building suggestions in each turn? (yes/no)
yes
>  Completed!
```

As mentioned before, the user can customize their own game, by setting:

- Building points to win the game. (3-15)
  If you want to play a quick game, then you can choose a smaller number like 5.
  If you want to play a longer game, then you can choose a larger number like 12.
  Since when we set up the game, each player will build two basements, then 2 building points. Thus the lower bound is 3. Because the board only have 54 vertices and we have the rule that residences cannot be built adjacently. Then 15 is a proper upper bound.

- Remove geese or not
  When player rolled 7, geese will attack and people who have 10 or more resources will lose a half. Due to this feature, it will take relatively a long time to win a game. We enable the user to remove geese from the game, which kinds of provides a "Easy Mode".

- Require building suggestions
  When testing the program, we sometimes forget how many resources required to build a basement or upgrade the buildings. To avoid checking the game rules again and again, we provide a Suggestion to players. If you customize this feature by "yes", then you can receive suggestions in each turn. Suggestions are made based on how many resources the current player has.
  A sample:

```
>  Builder Red's turn.
roll
>  Input a dice value between 2 to 12 (inclusive):
5
>  You have rolled: 5
>  Based on your current resources, actions avaiable are:
>          build-road <edge#>: Build a Road;
>          build-res <housing#>: Build a Basement;
>          improve <housing#>: Improve a Basement to House;
>          trade <colour> <give> <take>: Trade with other players;
>          next: Pass control onto the next builder;
>  To list all commands, please enter 'help'
status
>  Builder Blue     has  2 building points,  1 BRICK,  5 ENERGY,  3 GLASS,  0 HEAT,  1 WIFI.
>  Builder Red      has  2 building points,  1 BRICK,  5 ENERGY,  2 GLASS,  5 HEAT,  1 WIFI.
>  Builder Orange   has  2 building points,  1 BRICK,  0 ENERGY,  4 GLASS,  0 HEAT,  2 WIFI.
>  Builder Yellow   has  2 building points,  3 BRICK,  3 ENERGY,  2 GLASS,  0 HEAT,  0 WIFI.
```

  Since builder Red has more than one HEAT and WIFI, then Red can build a road; Similarly, Red can build a basement, improve a basement. Trade with others, or just simply pass the control to the next player.

3. Suggestions During the Game

If you don't want suggestions to be shown in each turn and didn't set it in customize step. You can just simply enter the command `suggestion`, when you need, after rolling the dice. This command will give exactly the same suggestion as talked in the previous section, but only for the current player, current round.

4. Auto-Corrections When Trade

When you want to trade with other players, you need to input that player's colour, the recourse you provide, and the resources you want. We provide a auto-Correct tool that you only need to make sure the first letter of each type is correct.

For example:

`colour`:	"b" -> "Blue"
	"B" -> "Blue"
	"Read" -> "Red"
`give`:	"Glue" -> "Glass"
	"bricks" -> "Brick"
`take`:	"En" -> "Energy"
	"what" -> "Wifi"

This saves the time and you don't need to worry about the spelling problem.

Similarly, when asking the response from the other player, you can enter any case of "yes" and "no". Such as "YES", "No", "yEs", "no"

```
trade orange Glasses Energ
>  Red offers Orange one GLASS for one ENERGY.
>  Does Orange accept this offer?   yes/no (accept any case)
Yes
>  Successfully traded!
```

```
trade Yep Energy Glass
>  Red offers Yellow one ENERGY for one GLASS.
>  Does Yellow accept this offer?   yes/no (accept any case)
NO
>  Trade request was refused!
```

```
trade O G H
>  Red offers Orange one GLASS for one HEAT.
>  Does Orange accept this offer?   yes/no (accept any case)
YES
>  Successfully traded!
```