# Description

Convert your C version of the hoarding game into a C++ version

# Changes

Input and output are for the C and C++ versions are the same **EXCEPT** for the following

- Users will enter their names at the beginning of the program. These will be displayed instead of 0, 1, 2, 3, ...
- Displaying the gamestate has been changed and follows this pattern
    - Space Number | Space Name | Owner | Players
        - If no one owns the property the Owner is None
        - Players are displayed as [player_name : $cash_held]
        - When multiple players are on the same space they are displayed based on the order they were created in
- Users can only roll dice and leave the game now as the player details are now constantly visible

# Requirements

1. Your code must have at **LEAST** the following classes. You may have more but you must have at least these
    - Gamestate
    - Board
    - Player
    - Space
    - Property
    - DiceRoller
    - Rules

2. All of your classes and free standing functions must be inside of a namespace named Monopoly. For some reason CLion has issues autocompleting the members when inside the namespace but the members are still there. You just have to type them out manually.
3. Any dynamic space you allocate must be managed by a smart pointer.

# Restrictions

You may not use the following in your code

- Global variables
    - Class static variables are ok
- Arrays either dynamically or statically allocated except for argv

- C strings except for those located in argv
    - String literals strings in your code that appear inside “” are still ok
- Any C functions except for printf, scanf, fopen, fclose, fscanf, and sscanf
    - If you are looking for an equivalent to `atoi` see `stoi`
- `using namespace std;`
- `new` or `delete`
    - Your dynamic memory allocation is being managed by smart pointers so there is no need to call these function

# Hints and Advice

## Think about your design before coding

A little bit of thought about how you want to structure your solution goes a long way in producing neat, understandable code. Think about how you want to structure your classes. What is each made up of? What do you want each to be able to do?

That being said you'll encounter problems that you didn't consider when you start coding so your design will need to change to handle them. That's ok and a natural part of programming just update your design accordingly but don't forget the big picture when doing so.

## Break your problems down

Break your problems down. Break your problems down. Break your problems down. You really want to be doing this as it makes solving this problem and any others much much easier. Start with our big steps. For each one of those big steps, break it into slightly smaller but still big steps.Each of these steps will eventually become a function call. Keep doing this until you finally get down to a step that can be solved in a few lines of code (around 1 - 20 lines) and then put your solution there.

Each of your functions should probably be nore more than 25 lines of code. There might be some exceptions but if you are going over 25 then you really should start breaking down the parts of your problem into more functions. Trust me when I say this as I say it from a place of experience, that it is much easier to read, understand, maintain, and update small functions then it is with large functions. This is true even if you create hundreds of small functions as each of those functions on their own will be really easy to understand because they are small. If you've been writing really long functions use this project as place to break those bad habits and start reaping the benefits of small functions.

## Avoid Circular Inclusions

It's very likely that your code will have circular references in it. For example a player owns Properties but your Property could have a Player* to reference who their owner is. This is

completely fine, normal, and a good design. What you need to be careful with in our design is not to have circular inclusions, either direct or indirect. A circular inclusion is when a .h file, say A.h, includes another file, say B.h, that includes A.h. This could either be direct, B.h has a `#include "A.h"` in it or it includes another file that includes A.h. For example B.h could include C.h which includes D.h which includes A.h.

Code with circular includes like this will not compile correctly. To get this to work you should define a strict order of when you will include files in your header and when you will use a forward declaration of the class. For me I said that if two classes were mutually dependent on each other then if one class is made of the other it would include that class. If the other class only referenced the first class it would have only a forward reference. For example a Player has Properties but a Property only refers to a player, so Player would include Property and Property would have a forward reference to Player. Then since I needed to use Player methods in Property.cpp I would include Player.h in Property.cpp to get access to them.

What rule you pick to break circular inclusions doesn't really matter, the important part is that you are consistent following.

And as a reminder forward declarations of classes only needed when circular referencing starts happening in your code.

# Consider using a vector of Player* instead of just Player

When storing your players it might be easier to have a vector <unique_ptr<Player> > instead of just a vector<Player> as under this setup you know that your players won't move locations in memory if you have to remove elements from your array and you get to control precisely when the elements are deleted. To add a unique pointer to a vector don't forget to use std::move. For example if I have a unique pointer `up` and and want to add it to the end of a vector, `v`, I would do `v.push_back(std::move(up))`.

## Examples

User input is underlined to help differentiate between what is output and input

```
Enter how many players will be playing: 3
Enter the name of player 1: Zack
Enter the name of player 2: Bob
Enter the name of player 3: William
Space Number | Space Name | Owner   | Players
0            | Start      | None    | [Zack : $220], [Bob : $220],
[William : $220]
1            | A          | None    |
```

```
2             | B           | None    |
3             | C           | None    |
4             | D           | None    |
5             | E           | None    |
6             | F           | None    |
7             | G           | None    |
8             | H           | None    |
Zack please enter your move
1 to roll dice
2 to leave the game
Your move: 1
Zack, you rolled a 6
Would you like to buy F for $100?
Enter y for yes or n for no: y
Zack bought F for $100
Space Number | Space Name | Owner   | Players
0             | Start      | None    | [Bob : $220], [William : $220]
1             | A          | None    |
2             | B          | None    |
3             | C          | None    |
4             | D          | None    |
5             | E          | None    |
6             | F          | Zack    | [Zack : $120]
7             | G          | None    |
8             | H          | None    |
Bob please enter your move
1 to roll dice
2 to leave the game
Your move: 1
Bob, you rolled a 10
Would you like to buy A for $100?
Enter y for yes or n for no: y
Bob bought A for $100
Space Number | Space Name | Owner   | Players
0             | Start      | None    | [William : $220]
1             | A          | Bob     | [Bob : $120]
2             | B          | None    |
3             | C          | None    |
4             | D          | None    |
5             | E          | None    |
6             | F          | Zack    | [Zack : $120]
7             | G          | None    |
8             | H          | None    |
```

```
William please enter your move
1 to roll dice
2 to leave the game
Your move: 1
William, you rolled a 7
Would you like to buy G for $100?
Enter y for yes or n for no: y
William bought G for $100
Space Number | Space Name | Owner   | Players
0            | Start      | None    |
1            | A          | Bob     | [Bob : $120]
2            | B          | None    |
3            | C          | None    |
4            | D          | None    |
5            | E          | None    |
6            | F          | Zack    | [Zack : $120]
7            | G          | William | [William : $120]
8            | H          | None    |
Zack please enter your move
1 to roll dice
2 to leave the game
Your move: 2
Space Number | Space Name | Owner   | Players
0            | Start      | None    |
1            | A          | Bob     | [Bob : $120]
2            | B          | None    |
3            | C          | None    |
4            | D          | None    |
5            | E          | None    |
6            | F          | Bob     |
7            | G          | William | [William : $120]
8            | H          | None    |
Bob please enter your move
1 to roll dice
2 to leave the game
Your move: 2
Space Number | Space Name | Owner   | Players
0            | Start      | None    |
1            | A          | None    |
2            | B          | None    |
3            | C          | None    |
4            | D          | None    |
5            | E          | None    |
```

```
6            | F          | William |
7            | G          | William | [William : $120]
8            | H          | None    |
The winners are
William
```