Code Review 2

1. strToLower(std::string text)

Rather than running a loop and manually iterating through each character in the text, consider using `std::transform` with a lambda utilizing `std::tolower` to convert each character to lowercase. For example:

```
std::transform(text.begin(), text.end(), text.begin(), [](unsigned char c)
{return std::tolower(c); });
```

This would improve maintainability and clarity of the code, as it becomes immediately clear what the purpose of the code is without any extraneous variables like a counter.

2. Result return type

Multiple functions return a result as a `std::string`. This results in a stringly-typed codebase, which can be hard to maintain. Consider encapsulating string results into their own class/struct, e.g.:

```
struct Result { std::string message; int resultCode; }
```

This avoids misinterpretation of returned result strings, and can also provide more information to the caller.

3. command_joinRoom(...)

In the function `command_joinRoom(...)`, the exception or error case is placed consistently before the expected 'normal' case. Consider re-ordering the function such that the normal case is placed first. For example:

```
if (existingRoom != nullptr) {
//process normal case first
} else {
//process error case
}
```

This makes the code much more readable, as you can follow the expected order while reading from the beginning.

4. command_changeName(...)

The function `command_changeName(...)` returns an empty string, which seems unintuitive considering the purpose of the function. Consider returning the new name, or perhaps a boolean to signal success or failure to change name.

5. std::map<...> commands

Commands seemed to be mapped to string literals, which leads to stringly-typed code. Consider using an enumeration to store the different command names.