Rakab

(2nd phase)

Reza Namvaran 40212358042

Mahdi Sadeghi   40212358026

Advanced Programming Spring 2024

Dr. Sakhaei-nia

Introduction:

this work report details the development of the second phase of

advanced programming final project, Rakab.

In this phase, we focused on designing a graphical user interface (GUI) for the game, utilizing the Raylib library to enhance the user experience.

The project has been successfully compiled using GCC, MinGW, and Clang compilers, ensuring compatibility and performance across different platforms. we developed this project using CMake build system.

All of the development process is available on our GitHub repository:

https://github.com/Reza-namvaran/Rakab

Challenges:

1. Graphics: Designing a developing a GUI with C++ was the main challenge of our project, but we were able to resolve this by using State design Pattern in our project.

2. Save: Saving and loading one of the previous games was a challenge but we solve it by developing a system for our game Storage, therefore we was able to save and load the game easily.

Tools:

1. Git & Github: we used Git as our version control system in order to improve our development process.
2. Raylib: This project GUI is developed with Raylib. Raylib is a C base library for developing graphics with different programming languages, it uses functions to develop a program or a game .e.g for controlling inputs from keyboard or mouse, rendering textures and images, etc.
3. Gimp: We used Gimp to edit our picture assets.
4. Figma: We used Figma to design our game GUI.
5. CMake: Used as build system.

New Cards:

In this version of the game new cards were added.

1. Bishop
2. Spy
3. Turncoat

Peace Sign:

A new feature of this version is peace sign. The player who played Bishop card owns this sign. If peace sign is placed on a land no battle can happen on that land, therefore we cannot place war sign on it.

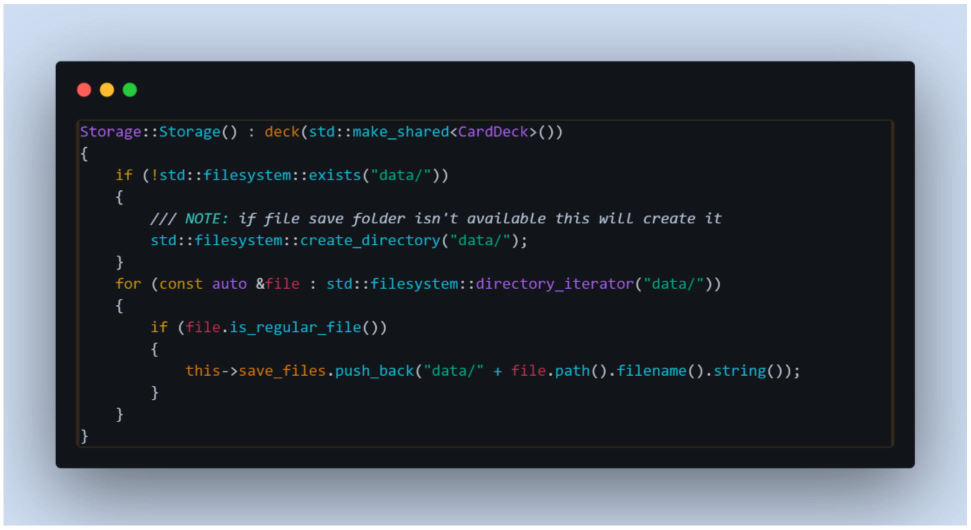
Save and Load System:

One of the biggest new features is saving a match and loading it whenever that we want. Now players can save up to five games and load each of them any time without any data loss.

Note: If more than five games are saved, the oldest one will be automatically deleted.

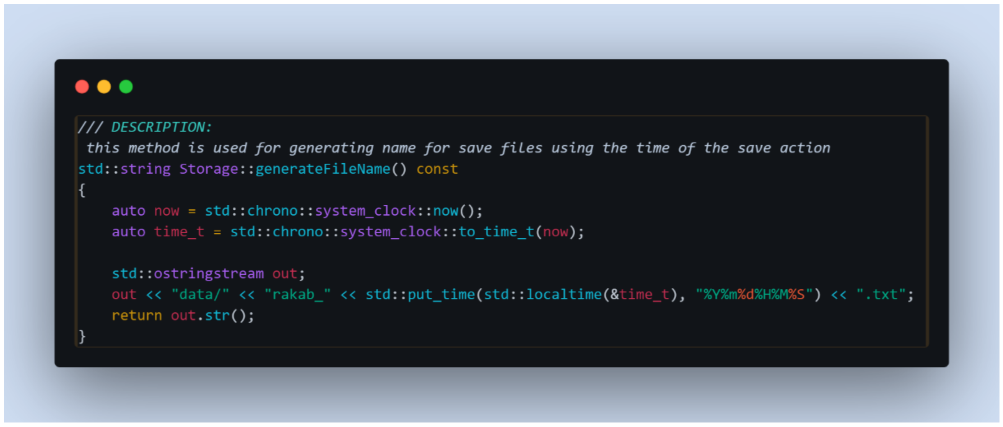
How do save system works:

We have developed a class called Storage for separating saving/loading data from the rest of the game. This class have some methods to save data that are necessary for playing a match. It saves data in txt files. This class has control fellows for different situations (mostly by using filesystem library) like if data folder which is the place that our save files are located in wasn't available in the folder it will create it.



```cpp
Storage::Storage() : deck(std::make_shared<CardDeck>())
{
    if (!std::filesystem::exists("data/"))
    {
        /// NOTE: if file save folder isn't available this will create it
        std::filesystem::create_directory("data/");
    }
    for (const auto &file : std::filesystem::directory_iterator("data/"))
    {
        if (file.is_regular_file())
        {
            this->save_files.push_back("data/" + file.path().filename().string());
        }
    }
}
```

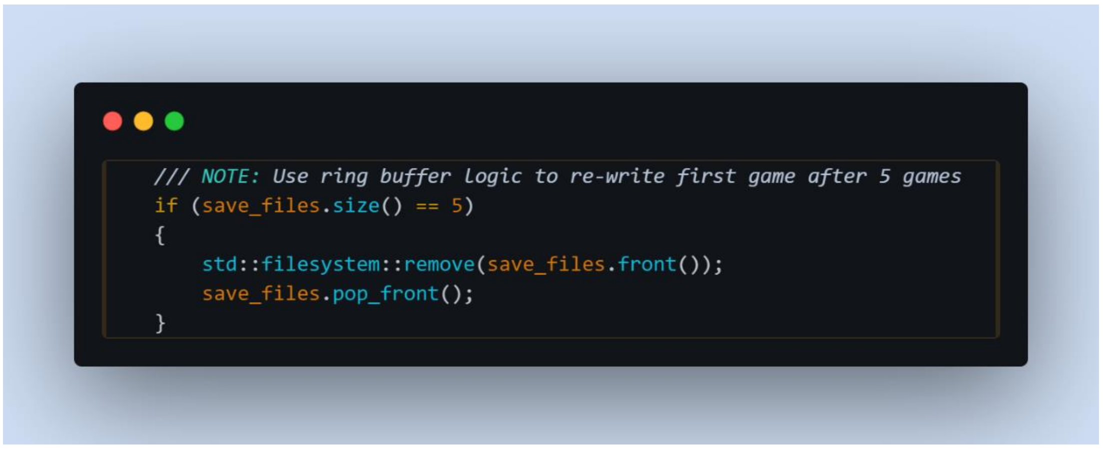
Also this class uses a system to create save files. It will create a file with a specific name format. Each file stores in data folder and will be saved in this format "rakab_Date.txt" Date is the exact date of that game (base on secs).



```cpp
/// DESCRIPTION:
 this method is used for generating name for save files using the time of the save action
std::string Storage::generateFileName() const
{
    auto now = std::chrono::system_clock::now();
    auto time_t = std::chrono::system_clock::to_time_t(now);

    std::ostringstream out;
    out << "data/" << "rakab_" << std::put_time(std::localtime(&time_t), "%Y%m%d%H%M%S") << ".txt";
    return out.str();
}
```

This class uses a structure called Ring Buffer for handling different save files count in the game. In this structure we store our file names into a deque and whenever player wants to save more than five games it will delete the oldest game from data the directory.



```
/// NOTE: Use ring buffer logic to re-write first game after 5 games
if (save_files.size() == 5)
{
    std::filesystem::remove(save_files.front());
    save_files.pop_front();
}
```

GUI Development:

State design pattern:

To handle game graphics more effectively we used State design pattern.

We divided our game into different states like main menu state and Player Setup state, this separation helped us handle GUI much easier.

In our system class we have a while loop that runs until the window is closed. This class also has a data member called state_manager which is from the class State Manager.

In the while loop we process, update and render graphics for each state.

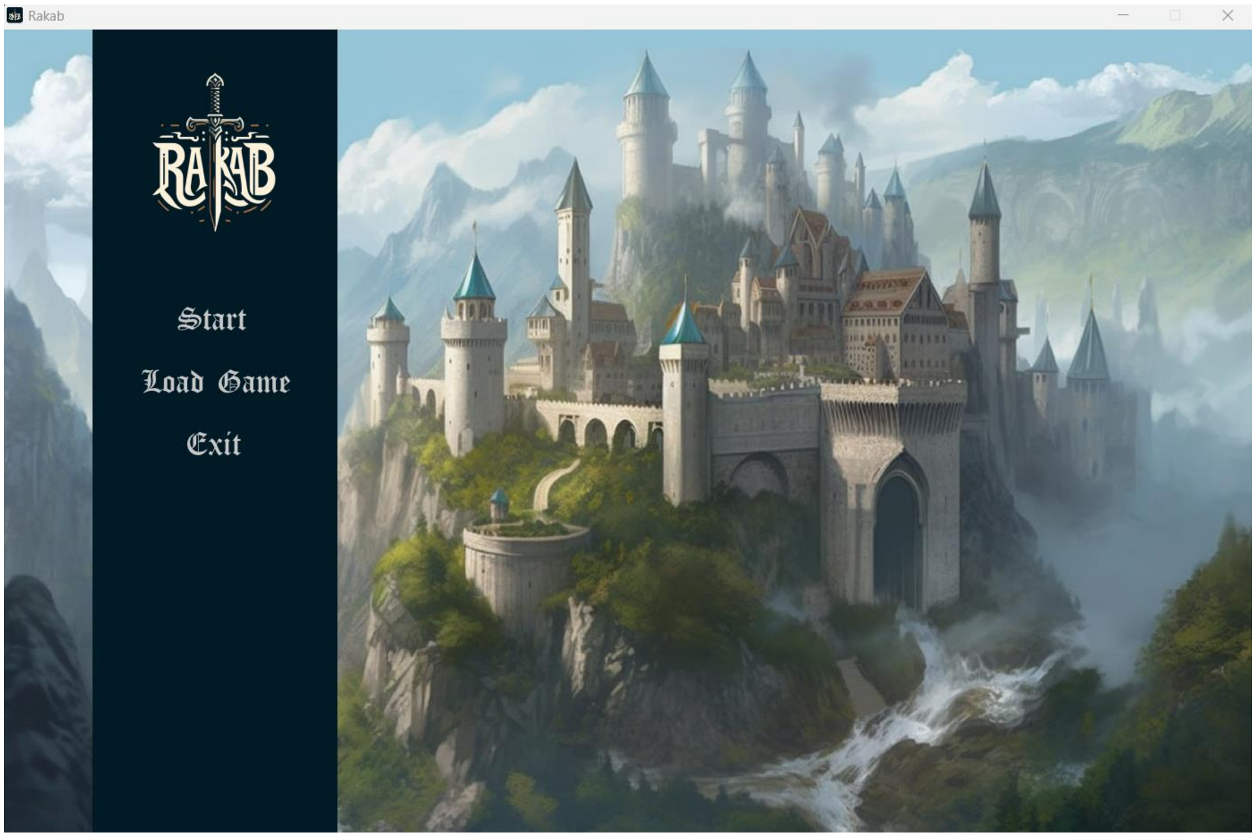
Each State has 3 main methods named as above, process will do the processing (sort of back-end code) for the background work of each state.

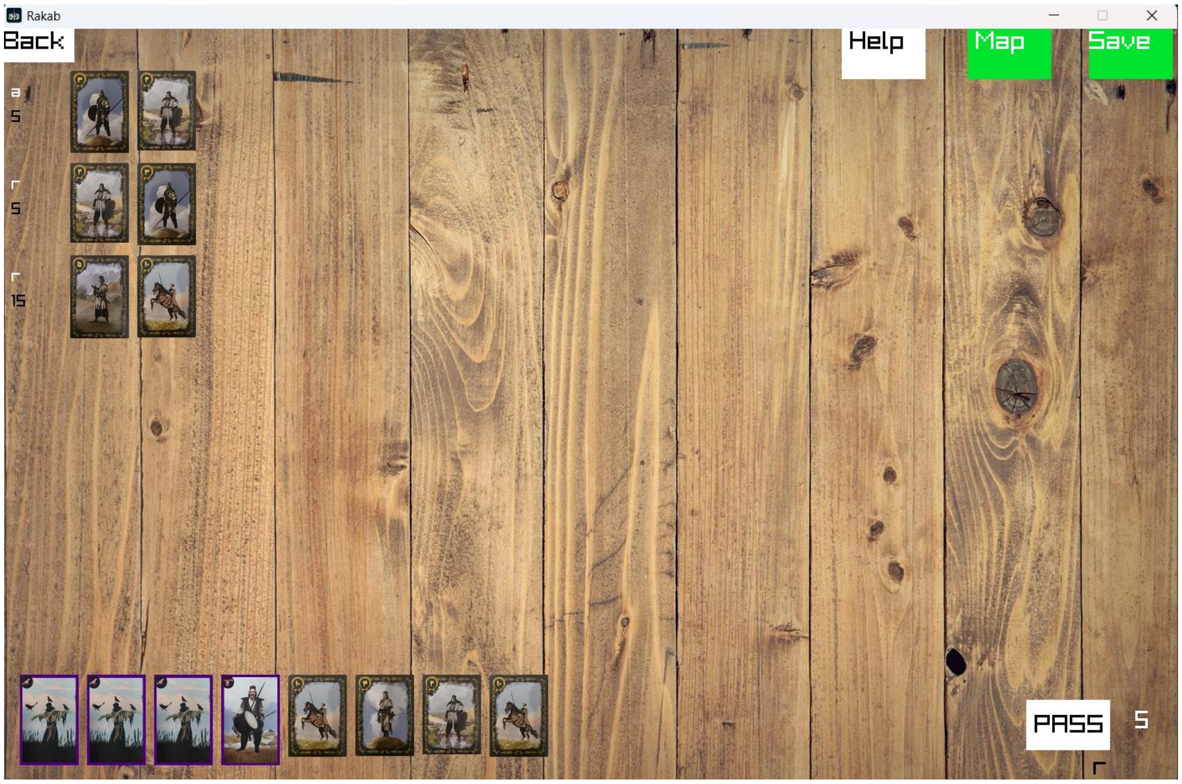
Update will update the variables and different situations in that state.

Render will render all graphics of that state.

Each state also has a status code that state manager receives it with event listener method in order to know when to move to another state.

Some States of the game (Main Menu and Match):


Rakab
RAKAB
Start
Load Game
Exit


Rakab
Back
Help
Map
Save
5
5
15
PASS
5

Game States:

1. Intro: Raylib intro.
2. Main Menu: Main menu of the game.
3. Player Setup: This state is used for getting players data e.g. name, age, color.
4. Load Menu: Here players can choose one of their previous games.
5. Match: The game state.

References:

Game development and Design Patters:

Game Programming Patterns by Robert Nystrom
https://gameprogrammingpatterns.com/