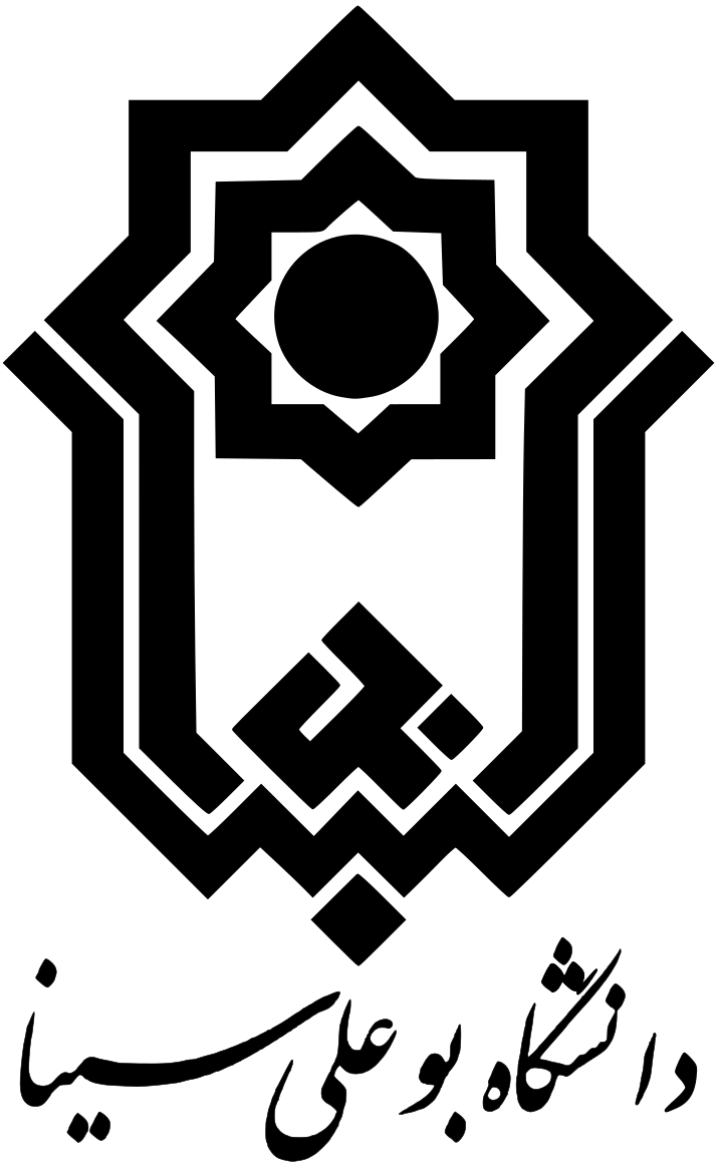

Condottiere

Advanced Programming
Final Project
Repository on GitHub

Professor
Mehdi Sakhaeinia, PhD

Students
Seyed Amir mohammad Fazeli - 40212358030
Amirabbas Fazelinia - 40212358031

Table of Contents

Terminology

Term	**Meaning**
move	Refers to move semantics in C++
own	X owns Y when X is responsible for managing Y's resources/object
is a	X is a Y when X inherits Y
has a	Refers to aggregation relationship in OOP
is a part of	Refers to composition relationship in OOP

Introduction

Condottiere is a strategic card game that simulates the political and military conflicts of the Italian Renaissance. Players take on the roles of mercenary leaders, known as condottieri, and compete to control territories and gain power. The game involves a combination of tactical card play, negotiation, and bluffing, making it a challenging and engaging experience for players. With its unique mechanics and historical theme, Condottiere offers an immersive gaming experience that will appeal to fans of strategy games.

Classification

This game contains multiple classes which will be explained in the incoming pages.

Game

A **Game** object holds the state of the application and contains the core logic. When created, it instantiates a **Map** object, *moves* the given **Player**s to one of its fields, and determines who starts the battle.
It includes different events which are subscribed within the constructor and it will do a specified function upon notifying.
Also consists of the **Render** function which is a main core of the **UI** interface,
And the **Update** function which updates the game based on player's interaction.

Relations

- **AssetManager** *is a part of* **Game**.
- **State** *is a part of* **Game**.
- Each **Event** *is a part of* **Game**.
- Each **Timer** *is a part of* **Game**.
- **MainMenu** *is a part of* **Game**.
- **CustomizationMenu** *is a part of* **Game**.
- **PauseMenu** *is a part of* **Game**.
- **Map** *is a part of* **Game**.
- Each **Player** *is a part of* **Game**.
- Each **Card** *is a part of* **Game**.
- **StatusBar** *is a part of* **Game**.

Map

A **Map** object *owns* the adjacency matrix of the game map as well as all the regions within it. When created, it *moves* the given regions and adjacency matrix to its fields.
It provides **GetRegions** and **GetRegion** methods which return a reference to the entire regions list and a single region respectively.
Method **FindWinners** looks into the map and figures out who wins the game at the current state. Note that there may be more than one winner if there is a tie when the regions are all conquered.

Relations

- **Map** *has a* **State**.
- Each **Region** *is a part of* **Map**.
- **Map** *has all* **Player**s.

Region

A **Region** object contains a name and has a ruler of type **Player**.
Also a **coordinate** which will be checked upon user's interaction.

Relations

- **Region** *has a* **PlayerInfo**.

Player

A **Player** object contains a name, a color, position and an age along with some other internal state that we will get into in a bit.
It has **Render** and **update** methods alongside **RenderRows** and **RenderCards** which will render different canvases within the player's class.
PlayCard is a method which checks collisions with a player's interaction , first deleting the cards and then does it's ability relative to the chosen card.
It also holds the **Mercenary** cards and provides **PickCard** for adding a card to this collection of cards.
Also tracks special cards in different methods and attributes.
It keeps track of whether the player has passed as well.

Relations

- **Player** *has a* **State**.
- **Player** *has a* **Season**.
- **Player** *has each* **Event**.
- Each **Card** *is a part of* **Player**.
- Each **Button** *is a part of* **Player**.

PlayerInfo

A light-weight representation of **Player**'s data.

Card

A **Card** is a object containing a enumerator which hold different type of cards within itself,
Provides **GetType** method which returns type of the card,
Also **GetAsset** which provides chosen card texture to render.

Mercenary

A **mercenary** object has a power which will be provided upon construction.
Has two key methods **GetAsset** and **GetPower** which returns texture and power.

AssetManager

An object of this class type will provide every canva which will be used throughout the game, and also has a destructor which frees the given space associated with canvas.

State

State has an enumerator which includes various states that affect the game flow.
It will track the current state and previous state, using method **Set.**
The current and previous state can be retrieved by **Get** and **GetPrev** methods.
It has an overridden operator which is used for proper logging.

Event

Every object with this type has a special ability throughout the game.
It is initiated within the **Game** class, either it can be initiated directly or using functions, and then passed to other classes.
It can send a value which can be casted to any favorable type and uses an observer class which validates the incoming event.

Season

An enum class that holds the current season.
Seasons have an impact on game flow.

Button

A **Button** object has a text and **Rectangle** object which holds coordinates in 2D space.
It has **Hovered** and **Pressed** methods which checks if a button has been clicked on.
Enable and **Disable** method to enable and disable button functionality.

Entity

A pure class with **Render** and **Update** methods to be inherited.

MainMenu

A Main Menu object renders menu aspects.

Relations

- **Main Menu** has **Buttons.**

- **Main Menu** has **Events.**
- **Main Menu** has **State.**

CustomizationMenu

An object of this class handles inputs and number of players.
It checks if the number of players are within range and validates inputs that user enters,
And shows proper error messages if inputs are incorrect.

Relations

- Customization Menu has **Buttons.**
- Customization Menu has **Events.**
- Customization Menu has **State.**
- Customization Menu has **Input**.

PauseMenu

Pause Menu objects is a menu within the game which pauses the current game.
It has 3 buttons ,
Continue button to continues the game,
Help button to show proper help menu to user.
Save & Exit button to exit the game with proper saving.

Relations

- Pause Menu has an **Event.**
- Pause Menu has **State.**
- **Help Menu** is a part of **PauseMenu.**

HelpMenu

Help Menu object shows a proper help menu to the users who are unfamiliar with the game flow.
It has instructions for all of the cards and their usage.

Relations

- Help Menu has **Buttons.**

Input

Input object is place holder for user input .
It has a placeholder as a string and a coordinate as a rectangle for rendering.

StreamWriter/StreamReader

These classes abstract away common operations among streams, such as writing strings or vectors of objects. It only requires the user to implement a few essential operations, namely reading raw binary data or jumping to another position in the stream, in order to provide the needed functionalities.

FileStream

Class **FileStream** inherits both **StreamWriter** and **StreamReader** in order to ease the process of writing to/reading from a binary file.

Relations

- **FileStream** *is a* **StreamWriter**.
- **FileStream** *is a* **StreamReader**.

Challenges

Project Management

We used Git and GitHub for efficient collaboration on our C++ project. These tools helped us manage changes, work on different parts simultaneously, and track progress effectively.

Memory Leaks

We solved memory leaks in our C++ project by using smart pointers for better memory management. This improved performance and stability by preventing memory leaks and ensuring effective resource management.

Segmentation Fault

During the course of our C++ project, we encountered several segmentation faults attributed to null pointer dereferences. We addressed this issue by ensuring that all constructors and functions were provided with non-null pointers.

References

- What is a Makefile and how does it work? | Opensource.com
- Move Semantics in C++
- Conventional Commits
- Serialization in Hazel - My Game Engine
- Lambdas in C++
- State · Design Patterns Revisited
- SMART POINTERS in C++ (std::unique_ptr, std::shared_ptr, std::weak_ptr)