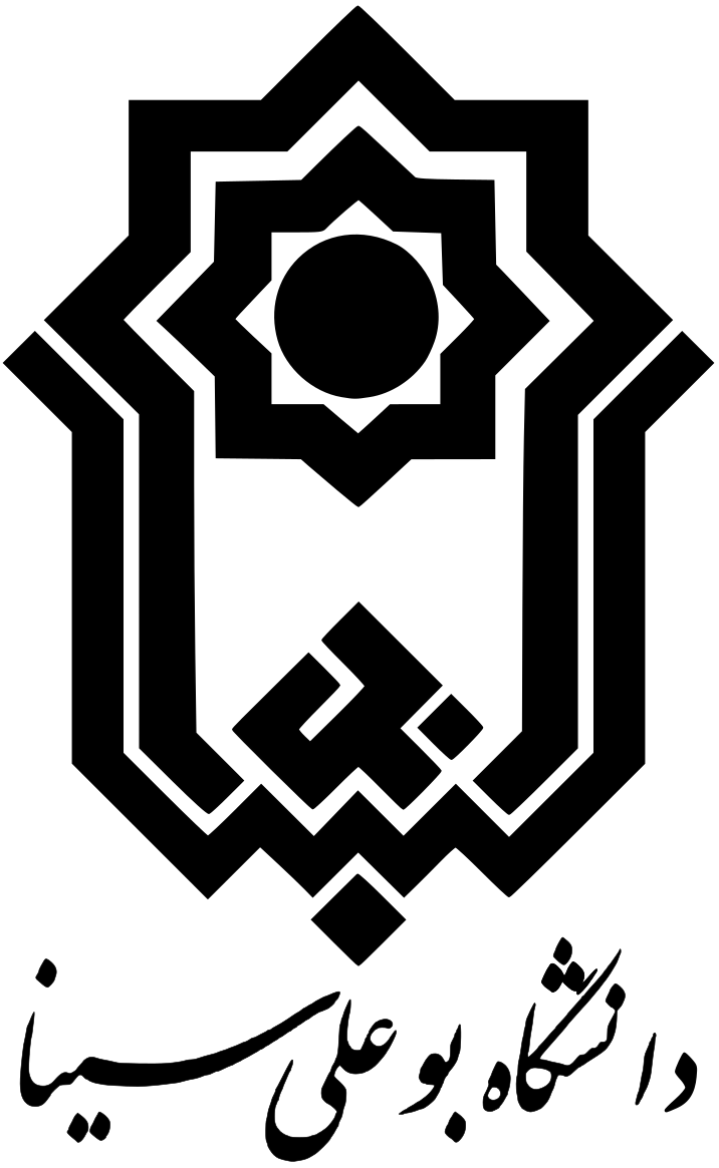


# Condottiere

**Advanced Programming**
**Final Project**

**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 provides a public method, **Start**, that deals cards among players (**DealCards** method) and also starts *the game loop* in which the battle marker is placed by the current player (**PlaceBattleMarker** method). Players are then prompted to draw a card (**PlayCard** method) one by one as the turn rotates.

### Relations

- **Map** *is a part of* **Game**.
- **Game** *has a* **Region** through the battle marker.
- Each **Player** *is a part of* **Game**.
- Each **Card** *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

- Each **Region** *is a part of* **Map**.

## Region

A **Region** object contains a name and has a ruler of type **Player**.

### Relations

- **Region** *has a* **Player** named ruler.

## Player

A **Player** object contains a name, a color, and an age along with some other internal state that we will get into in a bit.
It holds cards in the hand of the player and provides methods **TakeCard** and **AddCard** that take out a card by name and add a card respectively.
It also holds the drawn normal cards and provides **AddDrawnNormalCard** for adding a card to this collection of cards.
It keeps track of whether the player has passed as well.

### Relations

- Each **Card** *is a part of* **Player** through held cards and drawn cards.

## Card

A **Card** is a polymorphic object that contains a single method, namely **GetName**, which should simply return the name for searching purposes.

## NormalCard

A **NormalCard** object has a power which is given upon construction.
It inherits the **Card** class and implements its **GetName** method to return the power of card as string.

### Relations

- **NormalCard** *is a* **Card**.

# 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.

# References


- GNU make: https://opensource.com/article/18/8/what-how-makefile
- C++ move semantics: https://youtu.be/ehMg6zvXuMY
- Git Conventional Commits: https://conventionalcommits.org