OOP Final Project — Poker Game

Overview

Balatro is a critically acclaimed **roguelike deck-building poker game**. The game creatively blends **traditional poker hands** with **roguelike mechanics**, challenging players to build powerful decks that maximize points through combos and synergies.

In this project, you will implement a simplified **Balatro game** using object-oriented programming principles. The game includes core gameplay, score tracking, and optional bonus features such as an inventory/shop system. Players will log in, play rounds of poker, and accumulate scores as high as possible. The project will be graded according to the criteria outlined below.

Grading Criteria

	Component	Points
Basic	Basic Mechanics	60 pts
	Bonus Mechanics	20 pts
	Project Report	10 pts
Advanced	Additional Features	10 pts
	Total	100 pts

Basic Mechanics (60 pts)

Students are expected to implement the following core functionality:

1. Setting Stage

- ➢ Prompt player to Login (Load players data if exist in JSON file)
- ➢ Have the option to display the leaderboard showing top players and their scores
- ➢ Have the option to Logout

2. Playing Stage

- ➢ Open Games: Initially, the player would get 8 cards and have 4 rounds to play cards and 3 rounds of opportunity to discard cards.
- ➢ Play Cards: Allow the player to select and play between 1 to 5 cards from their hand in each round.
- ➢ Discard Cards: Allow the player to discard between 1 to 5 cards in each discard stage.
- ➢ Sort Hand: Provide functionality to sort the hand by suit or value.
- ➢ Score Calculation: Compute score based on the hand played.

3. Award Stage

- ➢ If four times of Play Cards or the player has no cards in hand, then display the final score for the rounds.
- ➢ Show gameplay statistics:
  - Total cards played
  - Times of each hand type (e.g., 3 Flushes, 1 Full Houses, etc.)
  - Number of discarded cards
- ➢ After displaying results, return to setting stage

Score Calculation Rules

Each hand is scored based on poker hand rankings. A multiplier is applied based on the type of hand. The score is calculated as:

score = (sum of card values with a specific card combination) × (multiplier)

Card Values is listed below

Cards	Value
A	11
K Q J	10
10 ~ 2	Face values

Here are the hand type and their multiplier

Hand Type	Multiplier
Straight Flush	×9
Four of a Kind	×8
Full House	×7
Flush	×6
Straight	×5

Three of a Kind	×4
Two Pair	×3
Pair	×2
High Card	×1

e.g. If player select & play [A, A, K, K, 5], which is a two pair, the score will be:

(11 + 11 + 10 + 10) * 3 = 126

Since card 5 is not calculated as two pairs, it will not be added to the score.

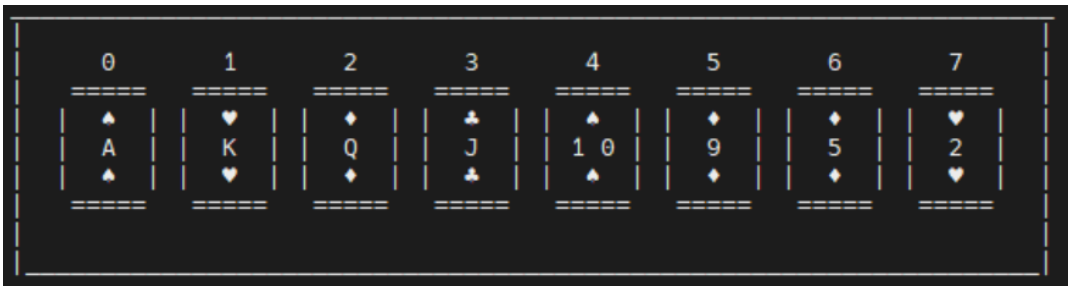
e.g. If the player selects & plays [9, 5, 4, 3], which is a high card (not belonging to any other types), only the card with maximum value should be selected. The score will be:

(9) * 1 = 9

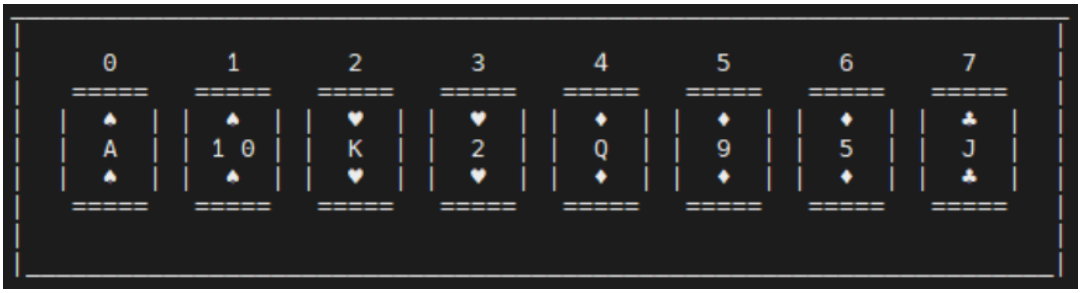
Card sorting mechanics

At the playing stage, players should be able to sort current hands in two ways.

Sort by values: (Suits are arbitrary)



Sort by suits: (Cards with the same suit are placed together)



Leader board

Before each game starts, there should be an option to check the current leader board, listing current players ranks and their score.
You can implement this in writing/reading json files to record current existing players and their scores.

Json file read/write

JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.

JSON are built from 2 structures:

1. *Object: Key-Value pair, wrapped in {}*
2. *Arrays: Ordered lists, wrapped in []*

e.g. players.json

```
[
    {
        "name": "Bob",
        "maxScore": 85
    },
    {
        "name": "Alice",
        "maxScore": 120
    }
]
```

In C++, the popular nlohmann/json library allows you to work with JSON files like native C++ data structures. You can check the github link for further usage.

To acquire the json.hpp, enter the following command inside terminal:
wget https://raw.githubusercontent.com/nlohmann/json/develop/single_include/nlohmann/json.hpp -P include/

To use this package, directly include the header file into your code. You can also declare the namespace for convenience.

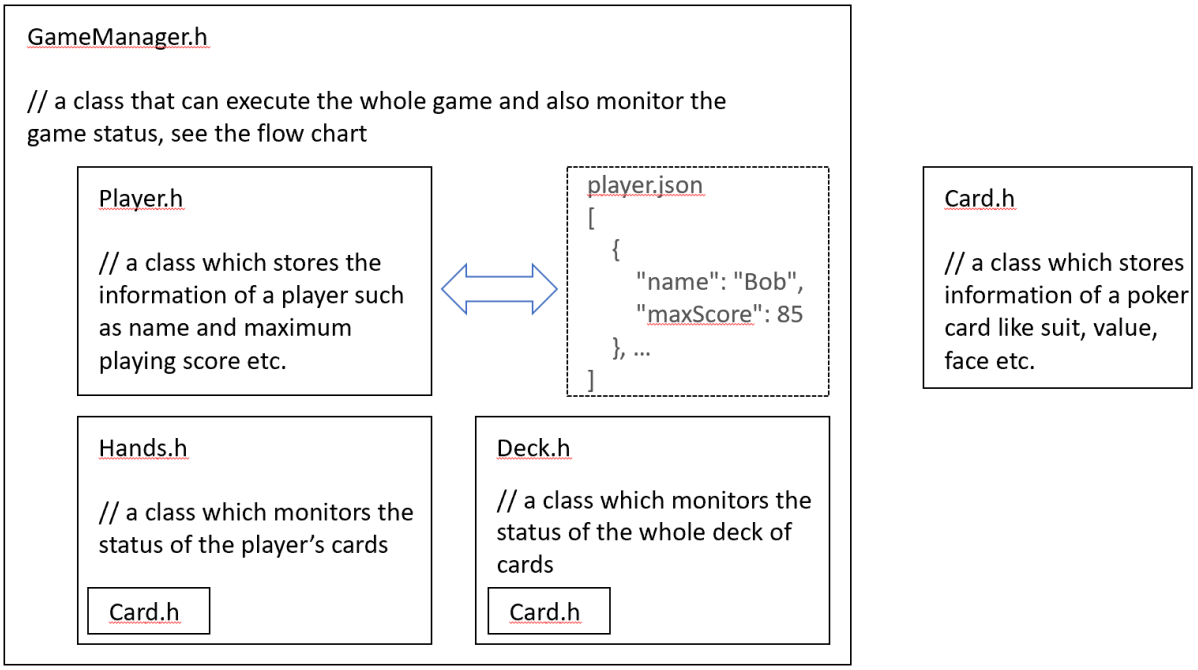
```
#include "GameManager.h"
#include "json.hpp"
#include <iostream>
#include <fstream>

using json = nlohmann::json;
```

Object-Oriented Requirements

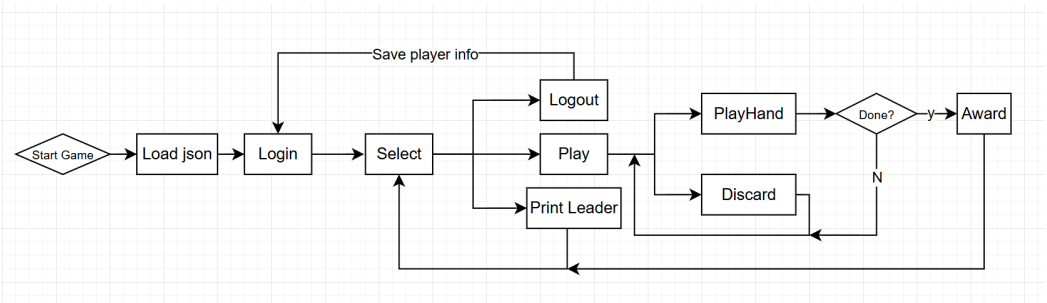
Your implementation must demonstrate proper use of OOP concepts:

- ➢ Encapsulation: Separate data from behaviors using classes appropriately.
- ➢ Abstraction: Use interfaces or base classes to generalize behavior.
- ➢ Inheritance/Polymorphism: Utilize where appropriate (e.g., for different item types).
- ➢ Modularity: Keep code clean and separated into logical modules.

OOP structure example:



GameManager class flowchart example (for basic mechanics):



You can modify the overall structure as you wish, but be sure to implement the overall project in OOP-manners.

Bonus Mechanics (20 pts) *Optional, but required for full bonus points.*

Implement an in-game shop/inventory/banking system as described below.

1. Currency System

After each round, players are awarded in-game money, and each player's balance must be stored and preserved across sessions.

money_awarded = final_score / 10

2. Shop System

At the start of each game, allow the player to visit a shop and buy items using their balance.

Required Items:

Item Name	Description	Price
Score ×2 Ticket	Doubles the score of the next played hand	10
Spade Ticket	Changes 3 random cards in hand to Spades	5
Heart Ticket	Changes 3 random cards in hand to Hearts	5
Diamond Ticket	Changes 3 random cards in hand to Diamonds	5
Club Ticket	Changes 3 random cards in hand to Clubs	5
Copy Ticket	Copies a randomly selected card in hand (adds a duplicate)	8

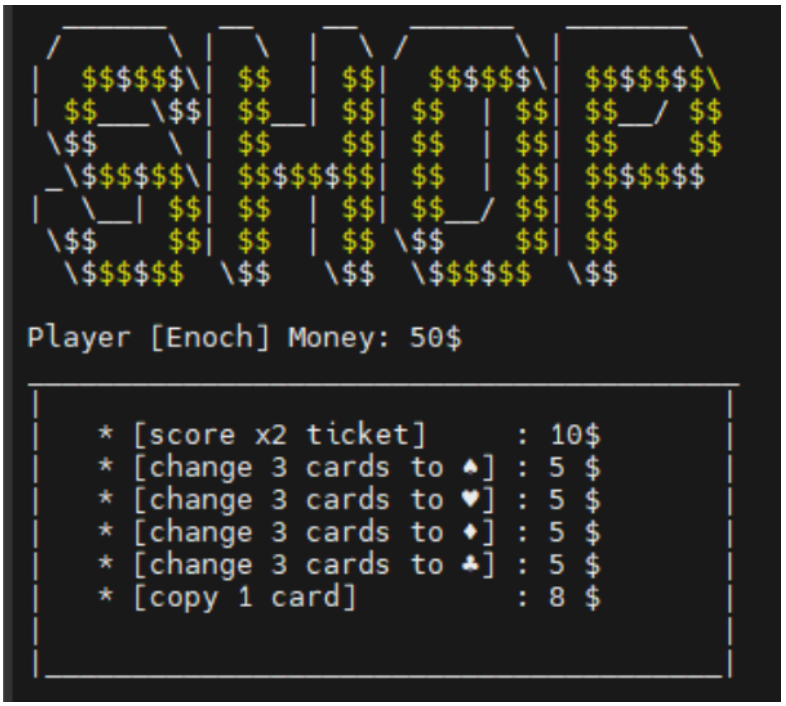
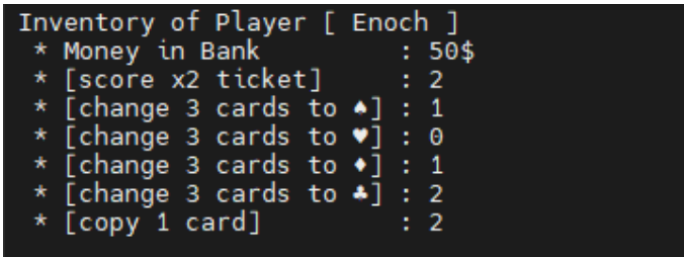


Fig. Visualization of the Shop System

3. Inventory System

- ➢ Purchased items should be stored in the player's inventory.
- ➢ Provide a user interface to view current inventory.
- ➢ When an item is used, it should be removed from inventory.



Additional Features (10 pts)

Points will be awarded for features beyond the scope of the basic and bonus mechanics. You can use your creativity to add more features. For example:

- ➢ Enhanced user interface/UX
- ➢ Multiplayer support (turn-based)
- ➢ AI opponents (even simple rule-based)
- ➢ More items in the shop

Additional features' score will be ranked among students. The best will get 10 pts, etc. Rank will be determined by TA.

Project Report (10 pts)

Your report must clearly demonstrate your **understanding of object-oriented design** as applied to the development of your simplified Balatro card strategy game. Please include the following sections:

- Class design overview (UML or equivalent diagram)
  - Class names
  - Inheritance relationships
  - Major attributes and methods
  - Associations
  - ***Highlight your design decisions and how they reflect OOP principles.***
- Explanation of main modules and their roles
  - For each core class or module explain:
  - Its **primary responsibilities**

    - Why it was separated as its own class
    - How it interacts with other modules
    - ***Highlight your design decisions and how they reflect OOP principles.***
- Gameplay flow description
    - Describe the full **gameplay loop**, including:
    - Start of game → Round setup → Sort Hands → Play/Discard Cards → Scoring → Next round
- Summary of challenges and how they were overcome
- List of additional/bonus features implemented and how you implemented them.

Example Visuals on terminal

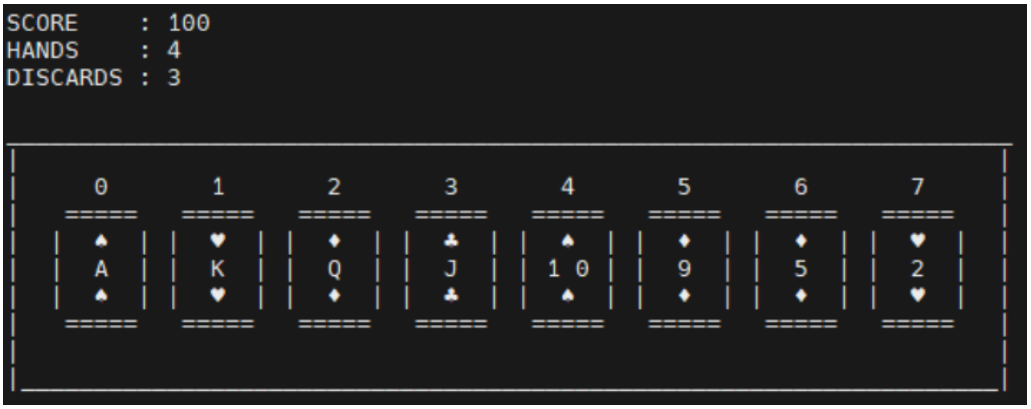


Fig. Visualization of the interface

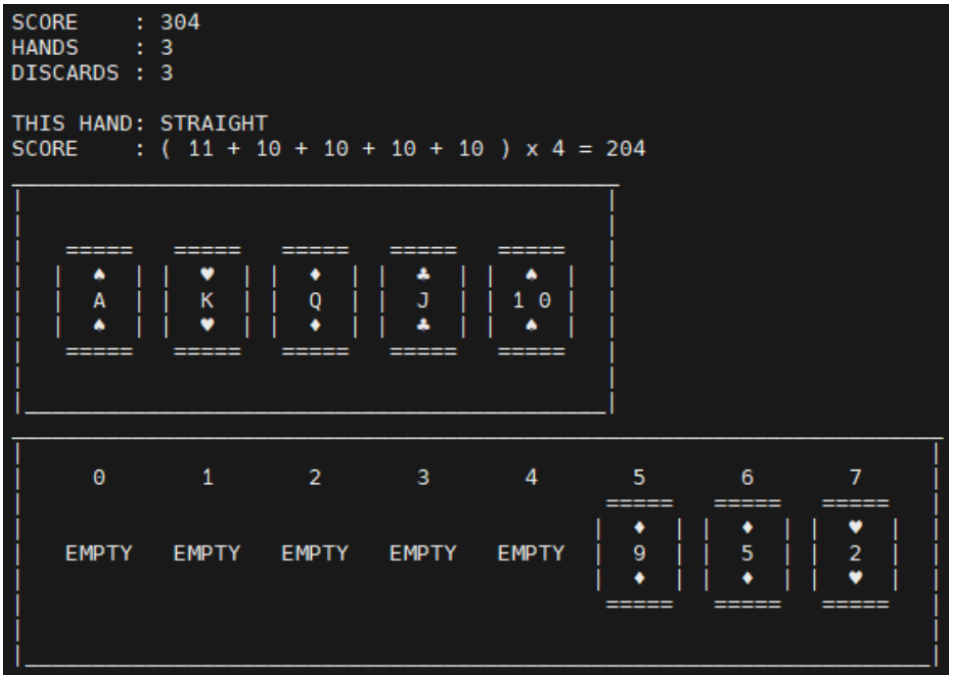


Fig. Visualization of the playing cards processing

Submission

1. Source files:

- Include all CPP files with detailed comments and proper indentation.
- Provide executable files.
- Provide ReadMe.txt to tell TA how to compile & run your code.
- Ensure all the necessary files for running the project are included.

2. Report (Student_ID.pdf)

3. Please compress the files above as Student_ID.tar and upload to E3.

4. Naming error: -10 points

Important Requirement :

- This project must be implemented using OOP principles. You are expected to design your program using classes, encapsulation, inheritance, and polymorphism where appropriate.
- Procedural or purely functional implementations will receive a grade of zero. Make sure your class design is reflected in both your code and your final report.

TA

1. 郭家均 **(kuochiayung@gmail.com)**
2. 趙泓瑞 **(raychao10301@gmail.com)**
3. 鄭律恒 **(mike.ee13@nycu.edu.tw)**

Plagiarism is strictly prohibited.

Code from the Internet cannot be used directly.

If found, the score will be 0 and we will report to the professor and handle the matter according to the relevant regulations.

REFERENCE:

If you have any problem regarding the gameplay mechanics, you can check the referred link.

肉鴿卡牌策略遊戲《小丑牌》公開基本玩法及卡牌介紹