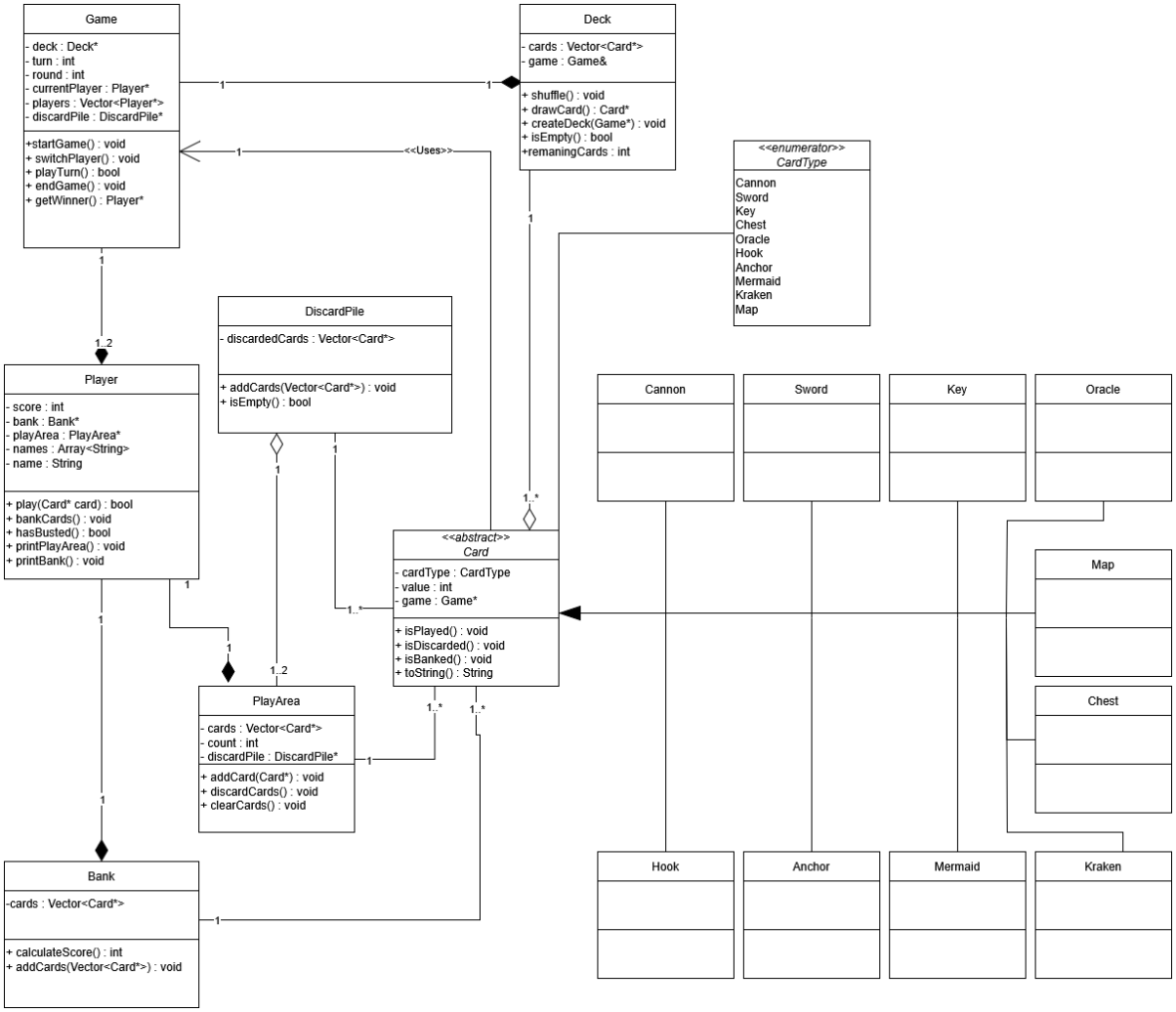
Dead Man's Draw++

Design Document

Final UML Design



Design Decisions

Core Design

The final UML design highlights the core structure of my implementation. Each class has a distinct and focused responsibility, managing only a specific aspect of the game's behaviour. By applying the separation of concerns design principle, the program is easier to understand, maintain, and extend.

My implementation makes use of the Template Method design pattern. The abstract Card class defines a common interface with three methods, isPlayed(), isBanked(), and isDiscarded(), representing the three possible activation states of a card.

Each Card subclass overrides one or more of these methods to implement specific abilities tied to that card's suit. The key advantage of this pattern is that it allows the common flow of handling cards (i.e., a card enters the PlayArea, Bank, or DiscardPile) to be defined in those classes, while delegating the specific behavior of the card to its subclass.

For example:

Dead Man's Draw++

Design Document

- When a card is added to the PlayArea, it calls card->isPlayed().
- If the card is instead sent to the Bank, the Bank class calls card->isBanked().
- Likewise, when a card is moved to the DiscardPile, card->isDiscarded() is called.

This structure ensures that all cards operate the similarly, but their abilities remain modular and easily extendable.

Using the Template Method pattern promotes code reuse, extensibility, and clear separation of responsibility between card logic and game flow logic.

Class responsibilities

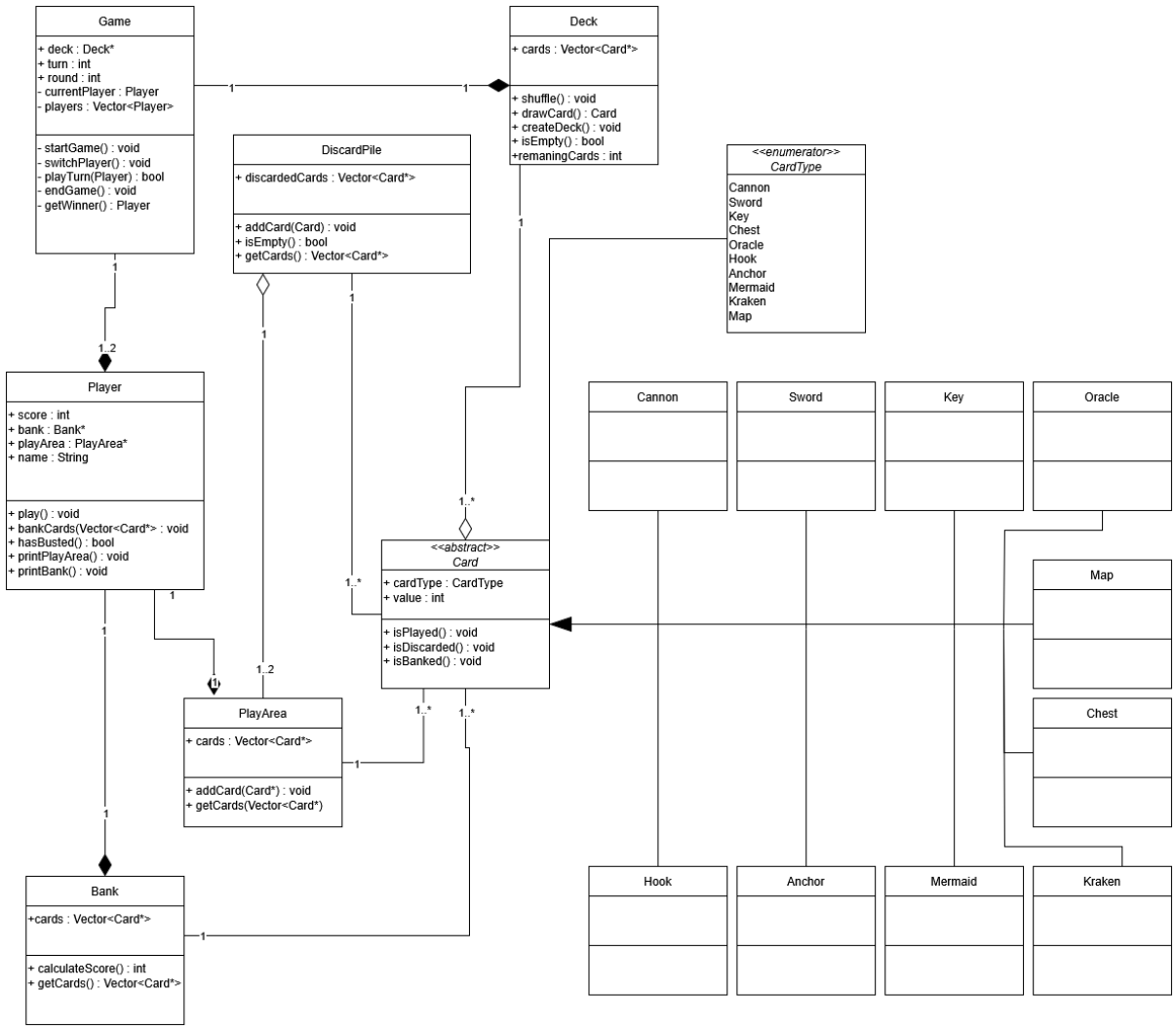
- Game
  - The Game class manages the overall game flow, including turns, rounds, and player interactions. It is responsible for coordinating the sequence of play and calling the necessary functions to initiate and process each player's turn.
- Player
  - The Player class manages the Bank and PlayArea classes and initiates all actions required after a card is draw and player into the play area. It has access to the score and is responsible for displaying information to the player.
- Deck
  - Is owned by the game and controls the creating, shuffling and drawing of cards.
- PlayArea
  - Is owned by an instance of the player class and it manages a vector of cards received by the player. The PlayArea is responsible for discarding cards.
- Bank
  - Is owned by an instance of the player and stores that player banked cards. It is responsible for calculating and retrieving the score for the player.
- DiscardPile
  - The DiscardPile is a container class that stores the discarded cards for all players. Providing access for the Card subclasses to read, write and retrieve from according to their ability.
- Card
  - The Card class is an abstract class that implements 3 methods for the 3 states of a card. These methods contain the abilities for each Card subclass.

Connor Freebairn / FRECD002 / ID:110375069

Dead Man's Draw++

Design Document

Initial UML design



Differences Between Initial and Final class diagram

The initial design adheres almost identically to the final implementation of the class structure with a difference in relationships, declarations and variable access.

The larges deviation from the initial design was the relationship between the card class and the game class. Originally, I intended for cards to only interact with the Player class and the different container classes. This idea wasn't fully fleshed out in my original design and proved difficult and overly complex to implement due to the required handling of multiple players, class instances and vectors. In the final implementation it was much simpler for the card to have reference to the Game object and work its way down through the class structure to access and modify the data required for the different card abilities. To facilitate this each class, contain getters for all variables that would be required by a cards ability to function.