Project 2

Hearts (v6)

CIS-17A

Stephanie Peacock

December 17, 2023

Introduction

Title : Hearts

The game of Hearts is usually a four player game. Each player is dealt 13 cards (dividing the deck equally). The player's intent is to score as low as possible. Each hand consists of thirteen tricks, in which each player plays one card. The lead player sets the suit for the trick, and remaining players must match the suit, if possible, else they can play the card of their choice.

The player with the two of clubs plays the first trick of each hand, usually point cards are not allowed during that trick, nor are hearts allowed to lead unless a heart has already been played. The winner of the trick (player with the highest card of the lead card suit) plays the first card of the next trick. This continues until all cards have been played.

Players score their hands as follows:

Any Heart card is worth one point. Queen of Spades is worth 13 points

If a player manages to collect all 14 point cards (for a total of 26 points) they score 0 points for the hand, and all remaining players are scored 26 points. This is called "Shooting the Moon."

Players play until an agreed upon total point count is reached, usually 50 or 100. Once one player reaches that point, the game is over, and the player with the lowest point tally is considered the winner.

Summary

Project size: ~1000 lines

<u>Classes:</u>

- Card
  - Overloads: < , > , != , == (int) , == (bool)
- Deck
  - Aggregates Card
- AbsPlayer (Abstract)
  - Pure virtual: play(Player&, Stooge **)
- Player (child of AbsPlayer)
  - Aggregates Card
  - Overloads: play(Player &, Stooge **)
- Stooge (child of Player)
  - Overloads: play(Player &, Stooge **)

My Modifications from Project 1 v3:

I converted all structures to classes.

I substantially trimmed down by player validation section with better structuring. Previously I was validating for each suit's value range. With Classes I was able to simply find the stooge that was first and compare the player's chosen card suit to the stooge's card suit using the overloaded ==.

The new version is ~ 90 lines of code, Project 1's player input validation was ~ 172 lines of code.

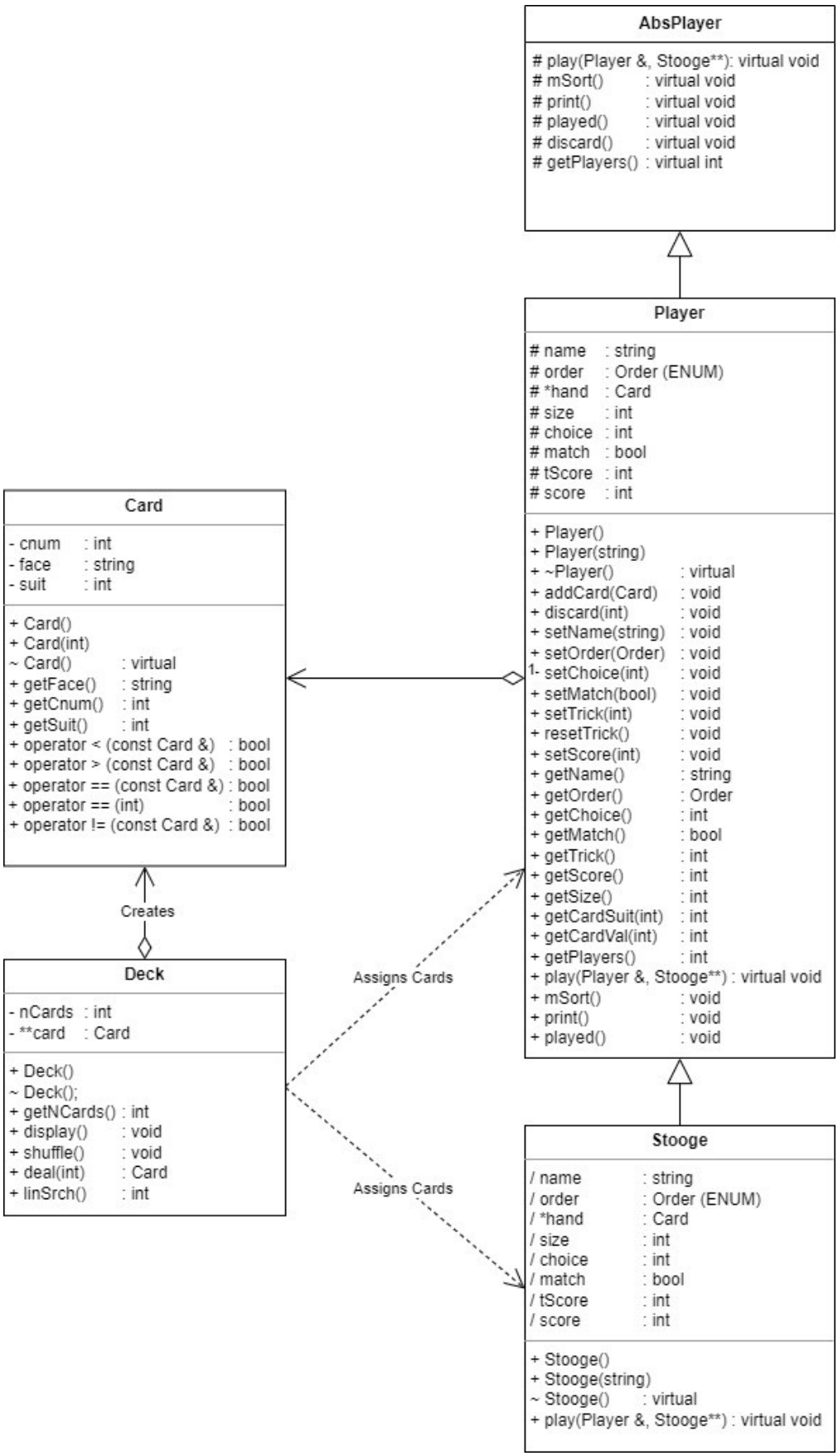
I also added a push/pop function to the player's hand – inspired by the SimpleVector lab assignment, which eliminated several display functions that removed card visibility only, while keeping the elements in place.

Cross Reference for Project 2

You are to fill-in with where located in code

Chapter	Section	Topic	Where Line #"s	Pts	Notes
13		Classes			
	1 to 3	Instance of a Class		4	AbsPlayer,Player,Stooge,Card,Deck
	4	Private Data Members		4	Never Public
	5	Specification vs. Implementation		4	.h vs. .cpp files Always split
	6	Inline	All .h files	4	return stuff (all getters)
	7, 8, 10	Constructors	Stooges = default Player = defined.	4	Overloading - Player
	9	Destructors	Blow up all the things	4	Destroy cards, deck, players
	12	Arrays of Objects		4	Deck, hand* stooge array
	16	UML		4	See Next Page
14		More about Classes			
	1	Static	Defined in Player.cpp	5	static int players (Player.h)
	2	Friends		2	Did not use
	4	Copy Constructors		5	Did not use
	5	Operator Overloading	Trick function, play Functions	8	Card.h – overloaded < > == and !=
	7	Aggregation	Player.h & Deck.h	6	cards inside deck & cards inside player
15		Inheritance			
	1	Protected members	Player.h	6	Player vars protected – stooge vars private
	2 to 5	Base Class to Derived	Stooge.h	6	Stooge is a Player
	6	Polymorphic associations	Stooge & Player cpp files	6	Virtual play() for player & stooge
	7	Abstract Classes	AbsPlayer.h	6	AbsPlayer - > Player - > Stooge
16		Advanced Classes			
	1	Exceptions	~127 - 137	6	Player.cpp – verify input is in range of hand
	2 to 4	Templates		6	Did not include
	5	STL	Main.cpp ~53-59	6	<vector> for the names
		Sum		100	

Class UML


AbsPlayer
play(Player &, Stooge**): virtual void
mSort() : virtual void
print() : virtual void
played() : virtual void
discard() : virtual void
getPlayers() : virtual int
Player
name : string
order : Order (ENUM)
*hand : Card
size : int
choice : int
match : bool
tScore : int
score : int
+ Player()
+ Player(string)
+ ~Player() : virtual
+ addCard(Card) : void
+ discard(int) : void
+ setName(string) : void
+ setOrder(Order) : void
+ setChoice(int) : void
+ setMatch(bool) : void
+ setTrick(int) : void
+ resetTrick() : void
+ setScore(int) : void
+ getName() : string
+ getOrder() : Order
+ getChoice() : int
+ getMatch() : bool
+ getTrick() : int
+ getScore() : int
+ getSize() : int
+ getCardSuit(int) : int
+ getCardVal(int) : int
+ getPlayers() : int
+ play(Player &, Stooge**) : virtual void
+ mSort() : void
+ print() : void
+ played() : void
Card
- cnum : int
- face : string
- suit : int
+ Card()
+ Card(int)
~ Card() : virtual
+ getFace() : string
+ getCnum() : int
+ getSuit() : int
+ operator < (const Card &) : bool
+ operator > (const Card &) : bool
+ operator == (const Card &) : bool
+ operator == (int) : bool
+ operator != (const Card &) : bool
1
Creates
Deck
- nCards : int
- **card : Card
+ Deck()
~ Deck();
+ getNCards() : int
+ display() : void
+ shuffle() : void
+ deal(int) : Card
+ linSrch() : int
Assigns Cards
Assigns Cards
Stooge
/ name : string
/ order : Order (ENUM)
/ *hand : Card
/ size : int
/ choice : int
/ match : bool
/ tScore : int
/ score : int
+ Stooge()
+ Stooge(string)
~ Stooge() : virtual
+ play(Player &, Stooge**) : virtual void