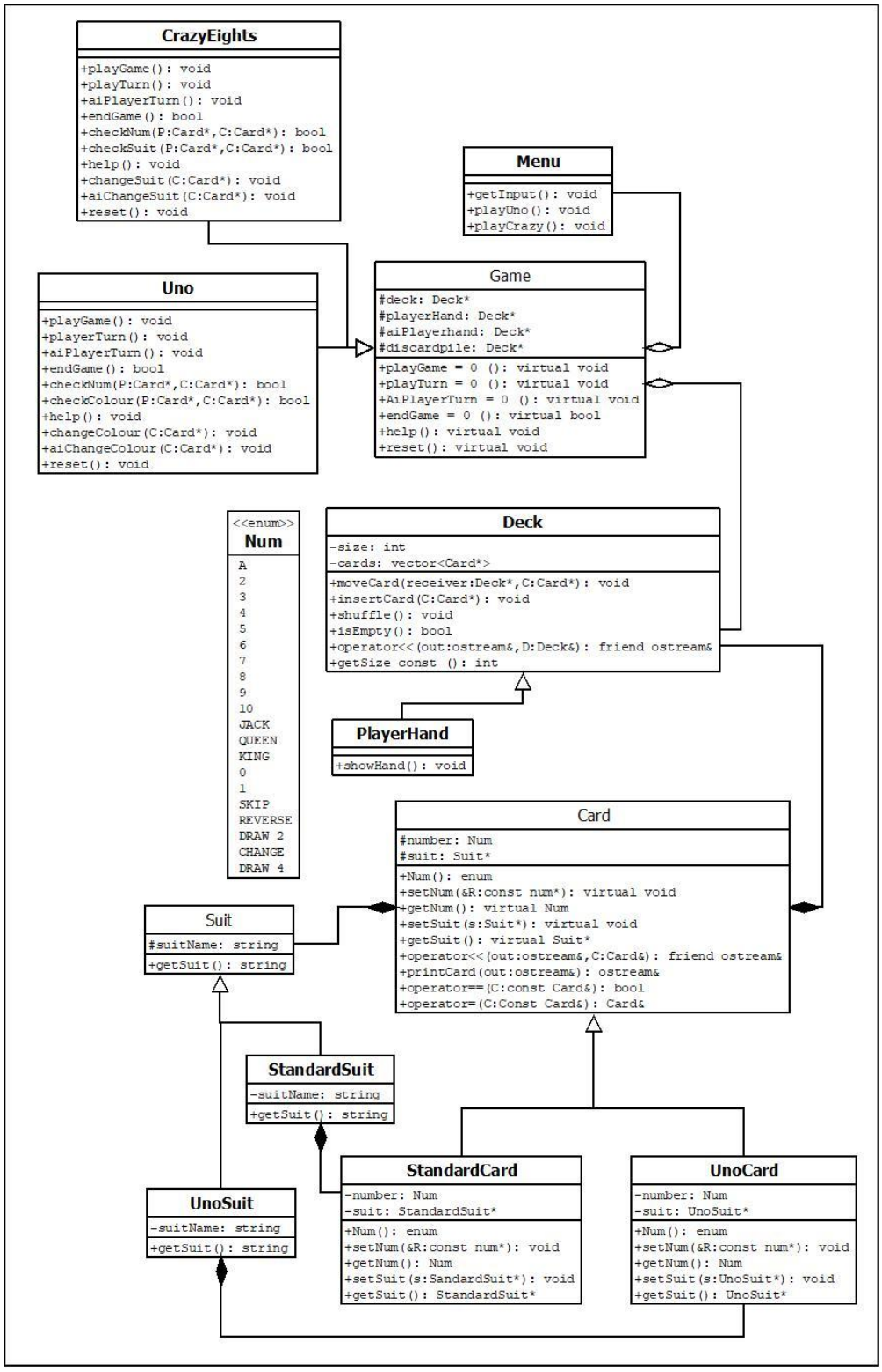
UML Diagram:


CrazyEights
+playGame(): void
+playTurn(): void
+aiPlayerTurn(): void
+endGame(): bool
+checkNum(P:Card*,C:Card*): bool
+checkSuit(P:Card*,C:Card*): bool
+help(): void
+changeSuit(C:Card*): void
+aiChangeSuit(C:Card*): void
+reset(): void
Menu
+getInput(): void
+playUno(): void
+playCrazy(): void
Uno
+playGame(): void
+playerTurn(): void
+aiPlayerTurn(): void
+endGame(): bool
+checkNum(P:Card*,C:Card*): bool
+checkColour(P:Card*,C:Card*): bool
+help(): void
+changeColour(C:Card*): void
+aiChangeColour(C:Card*): void
+reset(): void
Game
#deck: Deck*
#playerHand: Deck*
#aiPlayerhand: Deck*
#discardpile: Deck*
+playGame = 0 (): virtual void
+playTurn = 0 (): virtual void
+AiPlayerTurn = 0 (): virtual void
+endGame = 0 (): virtual bool
+help(): virtual void
+reset(): virtual void
<<enum>>
Num
A
2
3
4
5
6
7
8
9
10
JACK
QUEEN
KING
0
1
SKIP
REVERSE
DRAW 2
CHANGE
DRAW 4
Deck
-size: int
-cards: vector<Card*>
+moveCard(receiver:Deck*,C:Card*): void
+insertCard(C:Card*): void
+shuffle(): void
+isEmpty(): bool
+operator<<(out:ostream&,D:Deck&): friend ostream&
+getSize const (): int
PlayerHand
+showHand(): void
Card
#number: Num
#suit: Suit*
+Num(): enum
+setNum(&R:const num*): virtual void
+getNum(): virtual Num
+setSuit(s:Suit*): virtual void
+getSuit(): virtual Suit*
+operator<<(out:ostream&,C:Card&): friend ostream&
+printCard(out:ostream&): ostream&
+operator==(C:const Card&): bool
+operator=(C:Const Card&): Card&
Suit
#suitName: string
+getSuit(): string
StandardSuit
-suitName: string
+getSuit(): string
StandardCard
-number: Num
-suit: StandardSuit*
+Num(): enum
+setNum(&R:const num*): void
+getNum(): Num
+setSuit(s:SandardSuit*): void
+getSuit(): StandardSuit*
UnoCard
-number: Num
-suit: UnoSuit*
+Num(): enum
+setNum(&R:const num*): void
+getNum(): Num
+setSuit(s:UnoSuit*): void
+getSuit(): UnoSuit*
UnoSuit
-suitName: string
+getSuit(): string

Classes:

Card: Represents a card in the game, makes up a deck of cards

- setNum - sets the number, or value of the card
- getNum - returns the number or value on the card
- setSuit - sets the suit of the card
- getSuit - returns the suit of the cad
- operator<< - overrides the << operator in order to output the contents of the card to the screen, uses the printCard function
- printCard - reads the contents of the card and prints it to the screen
- operator== - overrides the comparison operator to compare cards
- operator= - overrides the assignment operator to assign values to cards

StandardCard: represents a standard playing card, with a suit and a standard value

UnoCard: represents an Uno card, with a colour and a number or special effect

Suit: Represents a card suit (such as a standard card suit, or a colour for an Uno card)

- getSuit - returns the name of the suit as a string

StandardSuit: represents the suit for a standard playing card (spade, diamond, heart, etc.)

UnoSuit: represents the colour of an Uno card, can be red, green, blue, yellow, or wild

Deck: Represents a collection of cards, can be used as a hand, a discard pile, or a draw pile

- moveCard - takes in the pointer for the deck that the card is moving to, as well as a pointer for the card that is being moved. Removes the card from the deck and adds it to the other deck
- insertCard - inserts a card into the deck, mostly used during initialization
- shuffle - rearranges the card vector randomly
- isEmpty - checks to see if there are any cards in the cards vector, returns true if there are none
- operator<< - overrides the output operator in order to output the contents of the deck to the screen, uses the Card's printCard function

PlayerHand: Represents the player's hand, has mostly the same functionality as a standard deck

- showHand - displays the contents of the player's hand

Game: Represents a game, abstract with pure virtual functions

Uno:

- PlayGame - Runs the logic for the game Uno, calling the necessary functions to run player turns, etc. (actual code is delegated to the called methods)
- PlayTurn - Runs the human players turn, getting input for what action to take, and calling the necessary logic to complete that task.
- aiPlayerTurn - Runs the AI Players turn, outputs the events that happened on that turn
- endGame - Ends the game and declares the winner to the player.
- checkNum - checks if the number on the 2 card arguments is the same
- checkSuit - checks if the suit on the 2 card arguments is the same
- help - displays the help function for the game, explaining the rules, how to play, etc.
- changeColour - changes the colour of the discard pile to the specified one (used for wild cards)
- aiChangeColour - changes the colour of the discard pile to the specified one (but used for the ai)
- reset - Resets the game to its initial state

CrazyEights:

- PlayGame - Runs the logic for the game crazy eights, calling the necessary functions to run player turns, etc. (actual code is delegated to the called methods)
- PlayTurn - Runs the human players turn, getting input for what action to take, and calling the necessary logic to complete that task.
- aiPlayerTurn - Runs the AI Players turn, outputs the events that happened on that turn
- endGame - Ends the game and declares the winner to the player.
- checkNum - checks if the number on the 2 card arguments is the same
- checkSuit - checks if the suit on the 2 card arguments is the same
- help - displays the help function for the game, explaining the rules, how to play, etc.
- changeSuit - changes the suit of the discard pile to the specified one (used for wild cards)
- aiChangeSuit - changes the suit of the discard pile to the specified one (but used for the ai)
- reset - Resets the game to its initial state

Menu: Provides an interface for the player to choose a game

- getInput - gets the player's input and starts a game based on their choice
- playUno - creates an Uno object and starts the game
- playCrazy - creates a Crazy Eights object and starts the game