

Top-Down Design, Bottom-Up Implementation

Rose-Hulman Institute of Technology

Computer Science and Software Engineering

Early Project kickoff

- We want you to begin thinking about this project now, so that you can work hard with your team this weekend.
- Details are on Project kickoff slides.

Plan for Today

- Design, implement, and test Blackjack together
 - Lots of whiteboard work
 - I'll share code after class

Designing/Implementing a Larger Program

- One common strategy: *top-down design*
 - Break the problem into a few big pieces
 - One function for each piece
 - Break each piece into smaller pieces
 - Continue until the pieces are “bite size”

Recall: Top-level Algorithm

- Create initial card deck
- Deal initial cards
- Player plays until busted or chooses to stop
- Dealer plays until required to stop
- Report who wins

Top-level Functions Called by main

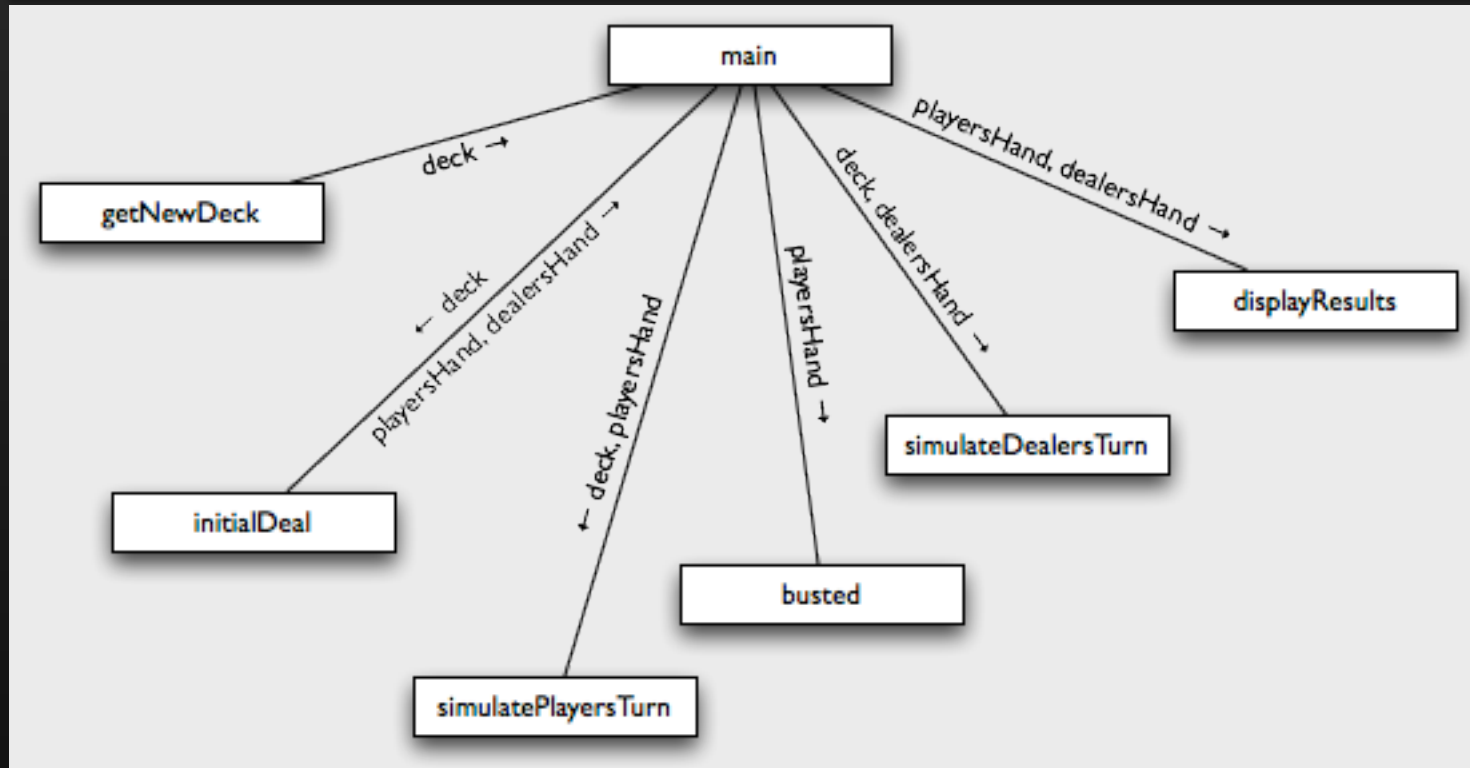
- **getNewDeck()**—Creates and returns a complete deck of cards
- **initialDeal(deck)**—Deals cards from the deck to each player, returns the hands
- **simulatePlayersTurn(deck, playersHand)**—Allows player to choose hit or stay
- **busted(playersHand)**—Checks whether the given hand is over 21
- **simulateDealersTurn(deck, dealersHand)**—Dealer hits or stays, based on the rules
- **displayResults(playersHand, dealersHand)**—Determines and displays who wins.

A couple of function names may be different than what we produced "on the fly" last class, but the functionality is the same.

Implementation of main

```
def main():  
    deck = getNewDeck()  
    playersHand, dealersHand = initialDeal(deck)  
    simulatePlayersTurn(deck, playersHand)  
    if(not busted(playersHand)):  
        simulateDealersTurn(deck, dealersHand)  
    else:  
        print("Busted!!!")  
    displayResults(playersHand, dealersHand)
```

Initial Blackjack Structure Diagram



Some Preliminary Data Values

Define some constants used by many functions

```
suits = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
```

```
cardNames = ['Ace', 'Deuce', '3', '4', '5',  
             '6', '7', '8', '9', '10',  
             'Jack', 'Queen', 'King']
```

```
winningScore = 21
```

```
dealerMustHoldScore = 16
```

Card is represented by a list: [cardName, suit]

Examples: ['Ace','Clubs'] or ['7','Diamonds']

A hand or a deck is a list of cards.

Q2

Bottom-up Testing

- Implement and test as we go
- Small changes, well tested make debugging easy

Class Exercise

Design, Implement, and Test

Q3-5

Designing newDeck()

- Write steps of **newDeck()** in English
- Write the code
- Refer to:
 - Data values on handout
 - Structure diagram on handout

newDeck() – returns complete deck

- start with an empty list
- for each cardName/suit pair
 - generate a card with that name and suit
 - add card to list
- Return the list

```
# Create an entire deck of cards  
def newDeck():  
    deckList = []  
    for s in suits:  
        for c in cardNames:  
            deckList.append([c, s])  
    return deckList
```

Designing initialDeal(deck)

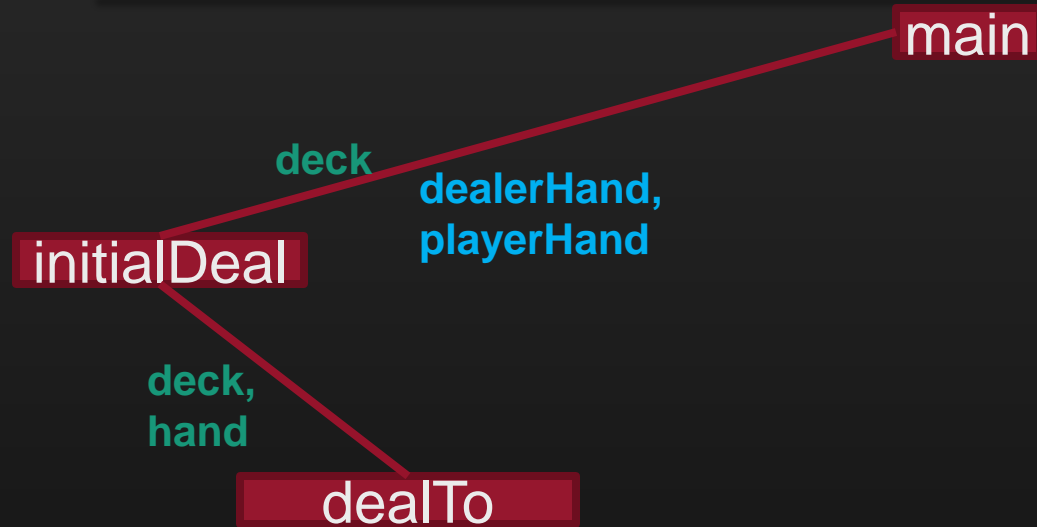
- Work in groups of 4 at a whiteboard
- Write steps for initialDeal(deck) function in English
- Write the code
- Take about 10 minutes
- Refer to:
 - Data values on handout
 - Structure diagram on handout
 - Do you need new functions? Add them to your structure chart

initialDeal(deck)

- start with two empty hands
- deal two cards to each hand
- return the two hands

```
# Deal two cards to each player.  
def initialDeal(deck):  
    playerHand = []  
    dealerHand = []  
    for i in range(2):  
        dealTo(playerHand, deck)  
        dealTo(dealerHand, deck)  
    return playerHand, dealerHand
```

initialDeal Structure Diagram



Key:
formal parameters
return values

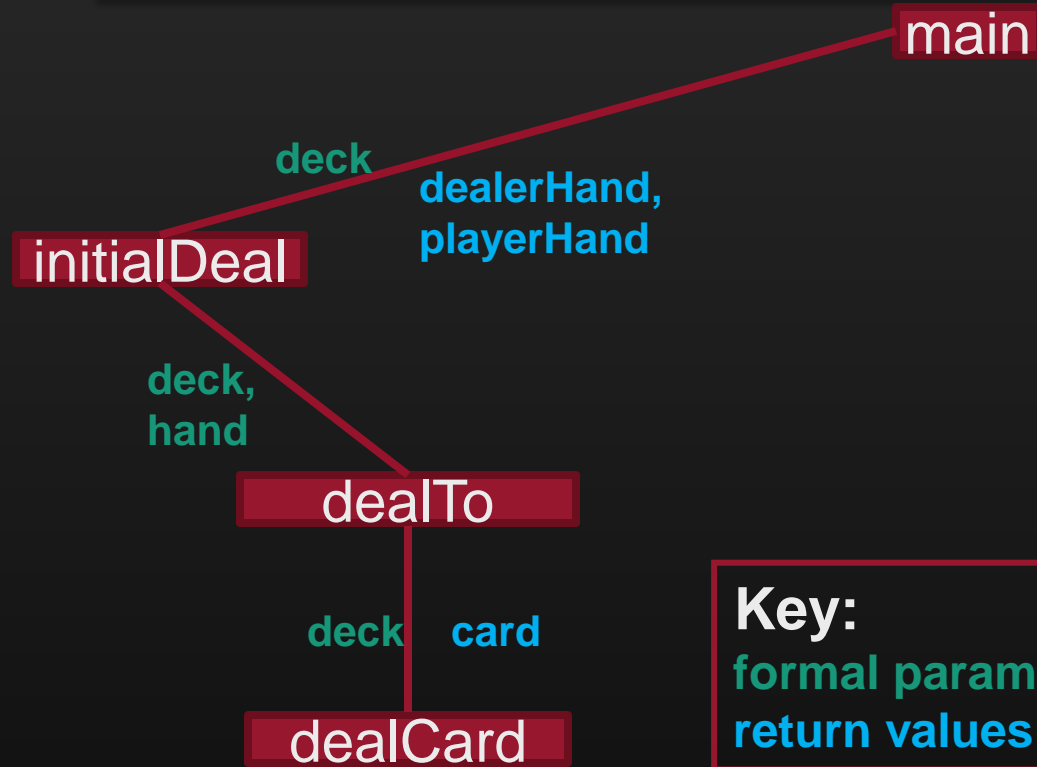
```
# Deal two cards to each player.  
def initialDeal(deck):  
    playerHand = []  
    dealerHand = []  
    for i in range(2):  
        dealTo(playerHand, deck)  
        dealTo(dealerHand, deck)  
    return playerHand, dealerHand
```

dealTo(hand, deck)

- Pick a random card from the deck and move it to the hand

```
# deal a card from this deck and place it in this hand.  
def dealTo(hand, deck):  
    hand.append(dealCard(deck))
```

initialDeal Structure Diagram

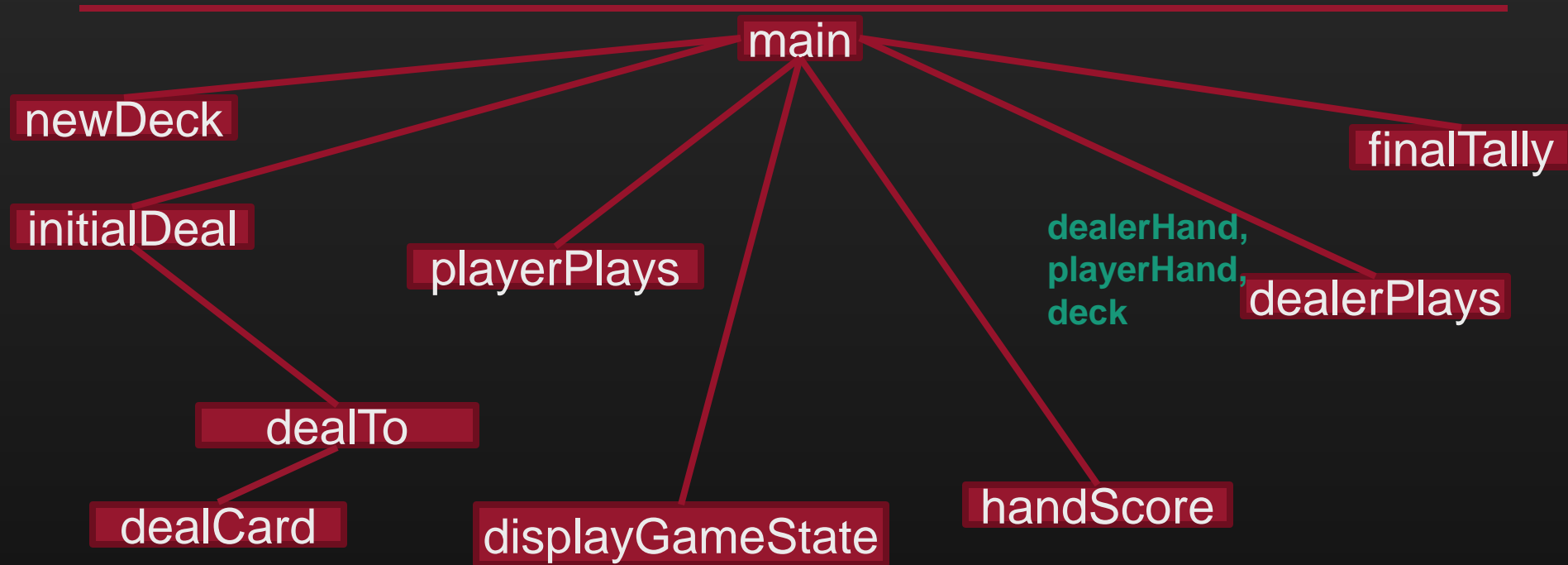


```
# Remove a random card from  
# the deck and return it
```

```
def dealCard(deck):  
    pos = randrange(len(deck))  
    card = deck[pos]  
    deck.remove(card)  
    return card
```

Key:
formal parameters
return values

Let's skip ahead to dealerPlays()



Designing dealerPlays()

- Work in groups of 4 at a whiteboard
- Write steps of dealerPlays() in English
- Write the code:
 - Do you need new functions? Add them to your structure chart
- Take about 10 minutes

dealerPlays

- while dealerMustTakeaHit
 - deal a card to Dealer's hand

Dealer takes hits until no more hits allowed.

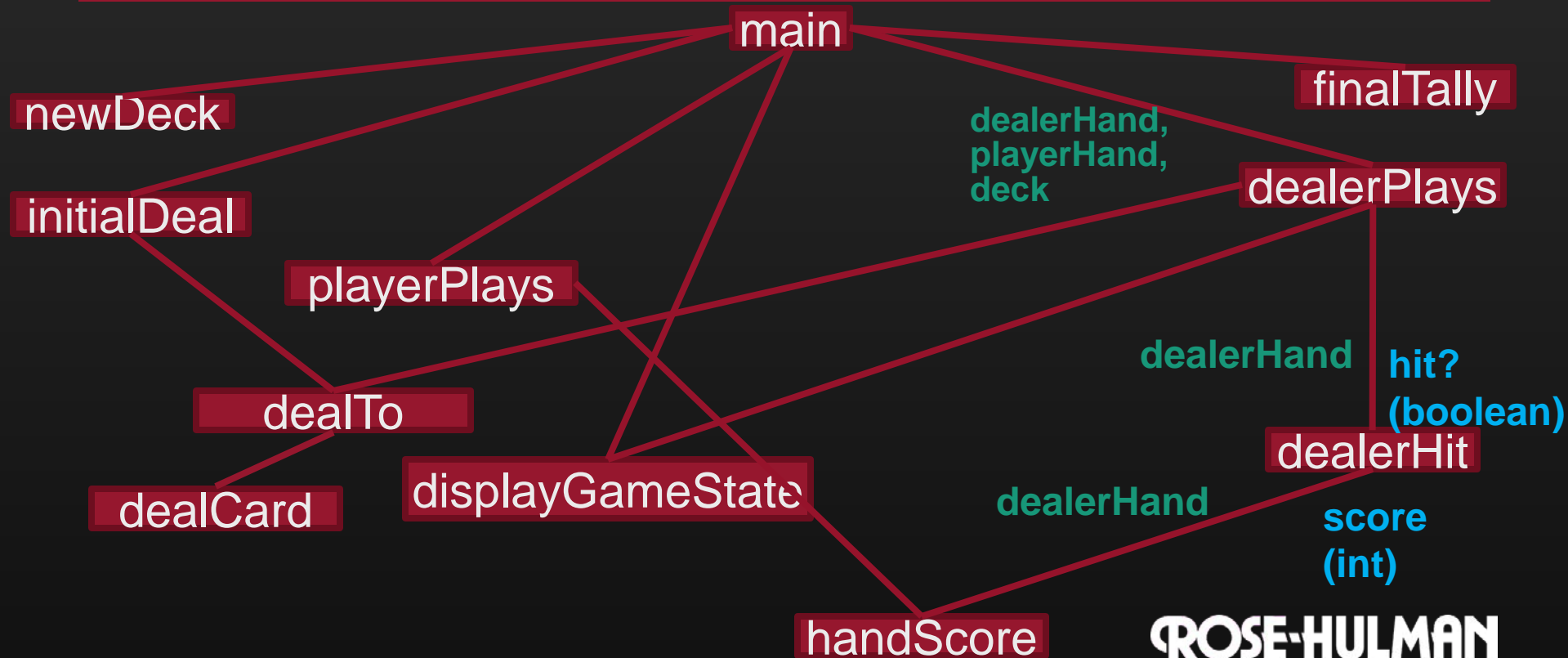
```
def dealerPlays(player, dealer, deck):  
    displayGameState(player, dealer, True)  
    while dealerHit(dealer):  
        sleep(3)  
        print("Dealer takes a hit")  
        dealTo(dealer, deck)  
    displayGameState (player, dealer, True)
```

**# Determine whether dealer
"takes a hit" (gets another card).**

```
def dealerHit(dealerHand):  
    dealerScore = handScore(dealerHand)  
    return dealerScore < dealerMustHoldScore
```

Design so far

Key:
formal parameters
return values



Code for handScore()

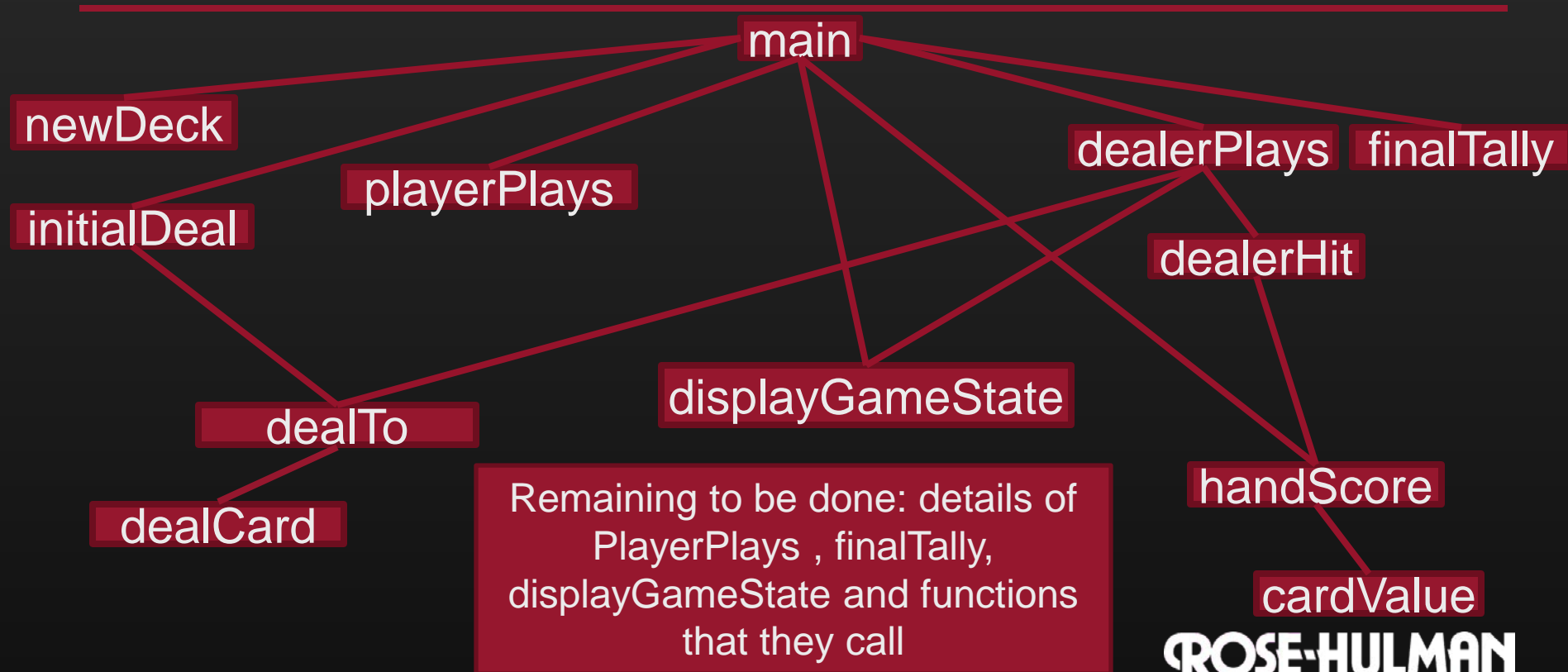
```
# Calculate the score for the whole hand.
def handScore(hand):
    score = 0
    hasAce = False
    for card in hand:
        val = cardValue(card)
        score += val
        if val == 1:
            hasAce = True
    if score <= winningScore - 10 and hasAce:
        score = score + 10
    return score
```

What if they have
two or more aces?

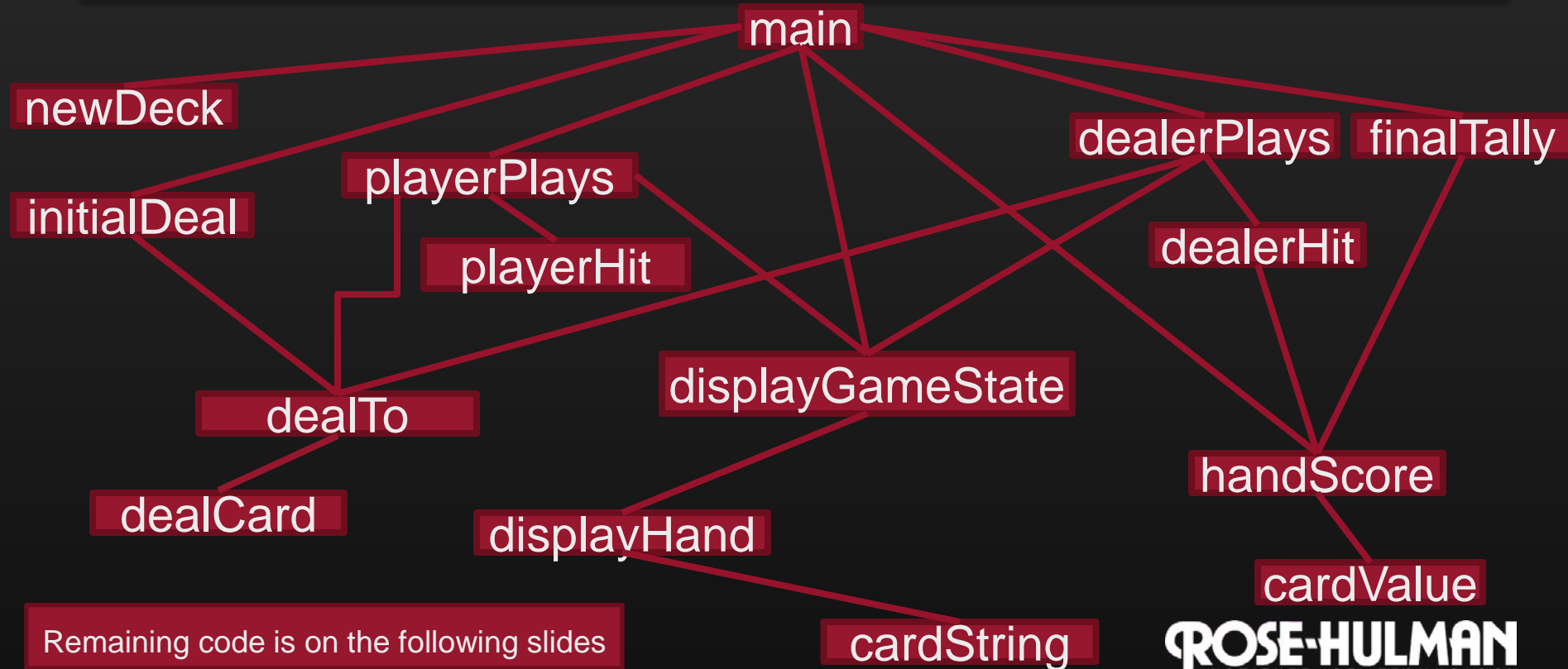
Code for cardValue()

```
# calculate how many points this card is worth.  
# Face cards count 10.  
# Ace Counts 1 (or 11, but that adjustment is  
#      made at the handScore level).  
def cardValue(card):  
    name = card[0]  
    pos = cardNames.index(name)  
    if pos < 10: # if not a face card.  
        return pos + 1  
    return 10
```

What we have developed so far



Complete Structure Diagram



Remaining code is on the following slides

Display Functions

Show the contents of both players' hands.

```
def displayGameState(playerHand, dealerHand, gameOver):  
    displayHand('Dealer', dealerHand, gameOver)  
    displayHand('Player', playerHand, True)
```

print the contents of the hand. showAll false implies not showing dealer's hole card

```
def displayHand(name, hand, showAll):  
    print(name + "'s hand:",)  
    if showAll:  
        print("(score is %d)" % (handScore(hand)))  
        print(cardString(hand[0]))  
    else:  
        print()  
        print(' Face Down' )  
    # print the rest of the hand.  
    for i in range(1, len(hand)):  
        print(cardString(hand[i]))
```

return a string that represents the given card.

```
def cardString(card):  
    return ' ' + card[0] + " of " + card[1]
```

playerPlays and PlayerHit

Player takes hits until Busted or stops requesting hits.

```
def playerPlays(player, dealer, deck):  
    while playerHit(handScore(player)):  
        dealTo(player, deck)  
        displayGameState(player, dealer, False)
```

Ask player whether she wants another card.

```
def playerHit(playerScore):  
    if playerScore > winningScore:  
        return False  
    answer = win_raw_input("Hit? (Y/N) ")  
    firstChar = answer[0]  
    return firstChar == 'y' or firstChar == 'Y'
```

finalTally function

```
# Figure out who won.
def finalTally(player, dealer):
    playerScore = handScore(player)
    dealerScore = handScore(dealer)
    if dealerScore > winningScore:
        print("DEALER IS BUSTED, YOU WIN")
    elif dealerScore > playerScore:
        print("DEALER WINS" )
    else:
        print("YOU WIN!" )
```