Object-Oriented Project Part 2 - GROUP

Team: Nolawee Mengist
Shawn Polson
Theerarun Tubnonghee (Steve)
Jose Escobar

Title: Blackjack!

Project Summary: A replication of the Blackjack card game that a user will play against a computer dealer and (optionally) computer players. Our goal is to create a program that looks and feels like an exact digitization of the real game; our code will faithfully represent Blackjack and our computer dealers/players will play autonomously.

Project Requirements:

Business Requirements: *We have omitted the “Business Requirements” table here because we do not have any.*

User Requirements				
ID	**Requirement**	**Topic Area**	**Actor**	**Priority**
UR-001	As a player, I can view the rules of Blackjack.	(optional)	(optional)	(optional)
UR-002	As a player, I can pick how many computer players to play with.			
UR-003	As a player, I can be dealt cards			
UR-004	As a player, I can place bets using chips			
UR-005	As a player, I can know the total amount of “money” I own.			
UR-006	As a player, I play against a CPU dealer			
UR-007	As a player, I can “hit” during my turn.			
UR-008	As a player, I can “stand” during my turn.			

UR-009	As a player, I can “double” during my turn.			
UR-010	As a player, I can “split” during my turn if my first two cards are of the same denomination.			
UR-011	As a player, I can win the bet if I get “21”			
UR-012	As a player, I lose the bet if I “bust” over 21.			
UR-013	As a player, I can play through as many rounds as I want.			
UR-014	As a player, I can still lose to the dealer even if I get 21			
UR-015	As a player, I can drop out and end the game with my winnings any time			
UR-016	As a player, I can save the current state of the game and continue to play.			
UR-017	As a player, I can continue my saved state of the game			
UR-018	As a player, I can start a new game and play with some amount of chips.			

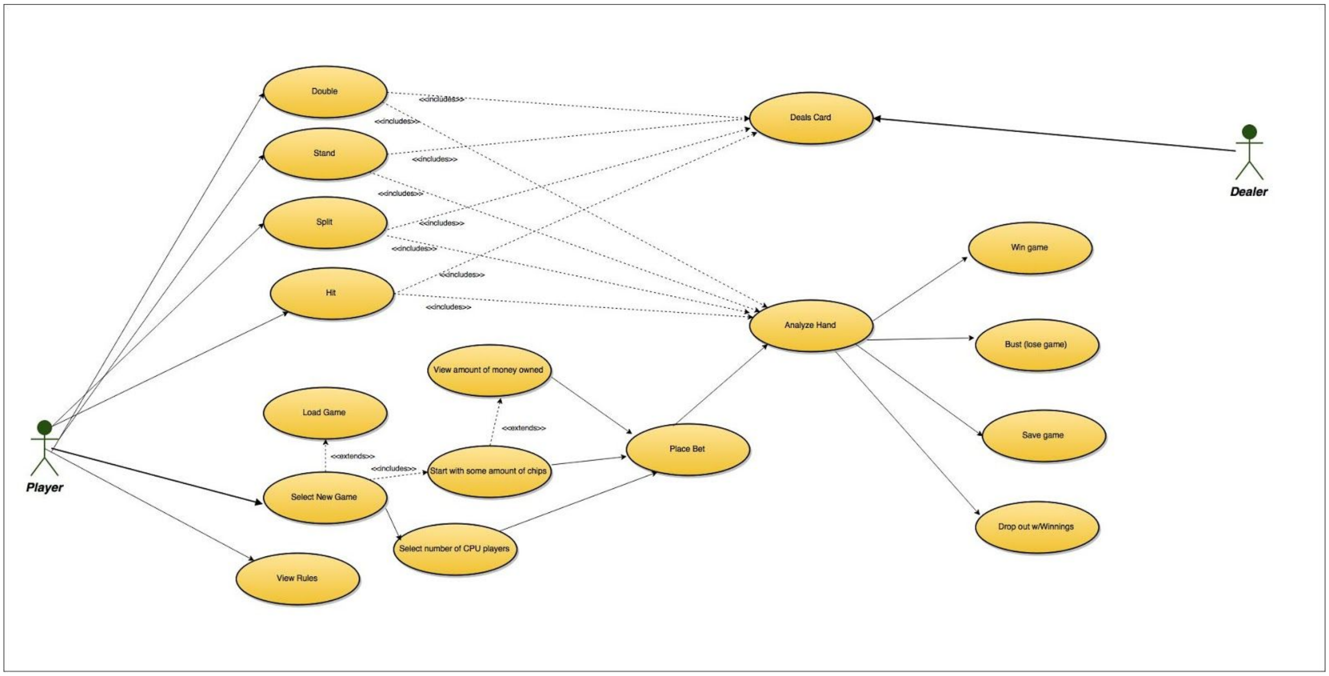
Functional Requirements				
ID	**Requirement**	**Topic Area**	**Actor**	**Priority**
FR-001	As a player, I lose if I run out of chips.	(optional)	(optional)	(optional)
FR-002	Card deck should be shuffled before play or when deck runs out			
FR-003	Dealer and computer players must			

	run through their turns when needed			
FR-004	Computer players must track their own total money and not bet past 0			
FR-005	Dealer and computer players must not hit when their currentTotal>=16			
FR-006	When player saves the game and exits, all totals and current cards in deck must be saved for reloading			
FR-007	User can navigate through the game.			

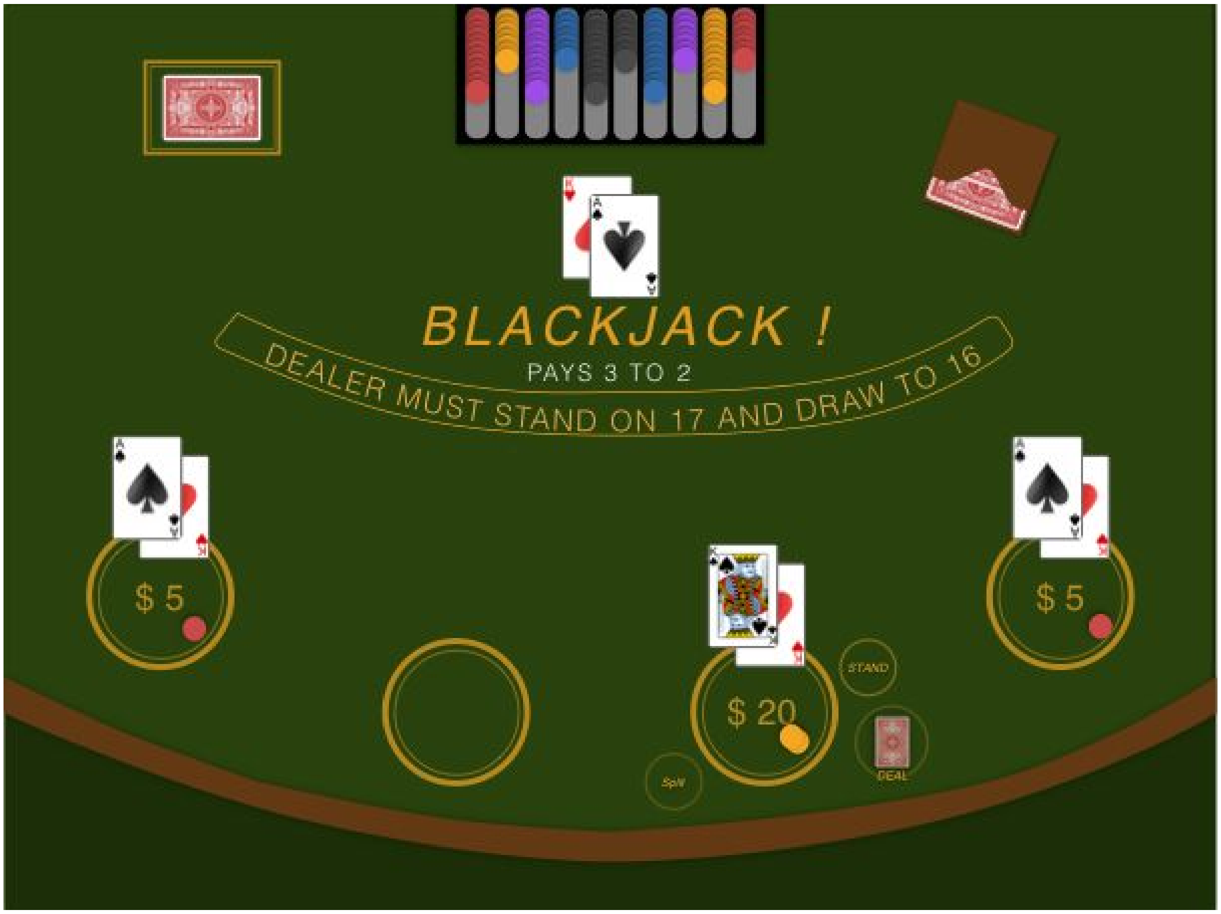
Non-Functional Requirements				
ID	**Requirement**	**Topic Area**	**Actor**	**Priority**
NFR-001	The game can be played on the command line before our GUI is implemented.	(optional)	(optional)	(optional)
NFR-002	The game should work properly each time it’s opened			
NFR-003				

Use Cases:

- **Use Case Overview:**

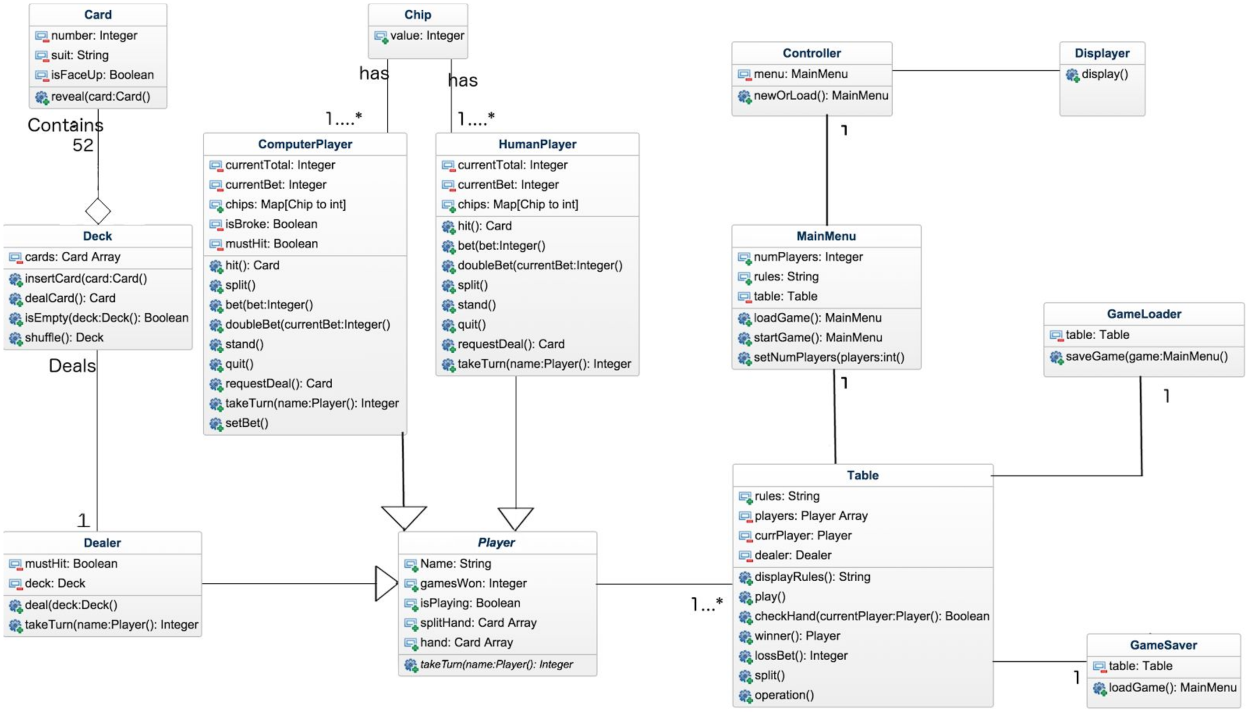

Double
<<includes>>
<<includes>>
Deals Card
Dealer
Stand
<<includes>>
<<includes>>
Split
<<includes>>
<<includes>>
Hit
<<includes>>
<<includes>>
Win game
Analyze Hand
Bust (lose game)
View amount of money owned
Load Game
<<extends>>
Place Bet
Save game
<<extends>>
<<includes>>
Start with some amount of chips
Player
Select New Game
Drop out w/Winnings
Select number of CPU players
View Rules

UI Mockups:



Data Storage: *We do not have plans for using a database. We plan on implementing "save game" by saving/loading our model (table) object; to have that be the object that persists).*

Class Diagram:


Card
number: Integer
suit: String
isFaceUp: Boolean
reveal(card:Card()
Contains
52
Deck
cards: Card Array
insertCard(card:Card()
dealCard(): Card
isEmpty(deck:Deck(): Boolean
shuffle(): Deck
Deals
1
Dealer
mustHit: Boolean
deck: Deck
deal(deck:Deck()
takeTurn(name:Player(): Integer
Chip
value: Integer
has
1....*
has
1....*
ComputerPlayer
currentTotal: Integer
currentBet: Integer
chips: Map[Chip to int]
isBroke: Boolean
mustHit: Boolean
hit(): Card
split()
bet(bet:Integer()
doubleBet(currentBet:Integer()
stand()
quit()
requestDeal(): Card
takeTurn(name:Player(): Integer
setBet()
HumanPlayer
currentTotal: Integer
currentBet: Integer
chips: Map[Chip to int]
hit(): Card
bet(bet:Integer()
doubleBet(currentBet:Integer()
split()
stand()
quit()
requestDeal(): Card
takeTurn(name:Player(): Integer
Player
Name: String
gamesWon: Integer
isPlaying: Boolean
splitHand: Card Array
hand: Card Array
takeTurn(name:Player(): Integer
1...*
Controller
menu: MainMenu
newOrLoad(): MainMenu
Displayer
display()
1
MainMenu
numPlayers: Integer
rules: String
table: Table
loadGame(): MainMenu
startGame(): MainMenu
setNumPlayers(players:int()
1
GameLoader
table: Table
saveGame(game:MainMenu()
1
Table
rules: String
players: Player Array
currPlayer: Player
dealer: Dealer
displayRules(): String
play()
checkHand(currentPlayer:Player(): Boolean
winner(): Player
lossBet(): Integer
split()
operation()
1
GameSaver
table: Table
loadGame(): MainMenu