SE 3XA3: Module Interface Specification (MIS)
BigTwo

Team 06, Team Name: Aplus^3
Senni Tan, tans28
Manyi Cheng, chengm33
Jiaxin Tang, tangj63

April 12, 2021

Contents

List of Tables

Table 1: **Revision History**

Date	**Version**	**Notes**
Mar 16, 2021	0.0	Initial Draft
Apr 10, 2021	1.0	Revision 1.0

1 Introduction

This document provides the overview and also the details of the module design. The code files that implement the modules in this document have been generated in doxygen files in the same folder.

2 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module: Hardware-Hiding Module

M2: Behaviour-Hiding Module: App Module

M3: Behaviour-Hiding Module: Card Module

M4: Behaviour-Hiding Module: Player Module

M5: Behaviour-Hiding Module: PlayerBot Module

M6: Behaviour-Hiding Module: Rules Module

M7: Behaviour-Hiding Module: Game Module

M8: Behaviour-Hiding Module: GameplayField Module

Level 1	**Level 2**
Hardware-Hiding Module	Hardware-Hiding Module
Behaviour-Hiding Module	App Module Card Module Player Module PlayerBot Module Game Module GameplayField Module
Software Decision Module	Rules Module

Table 2: Module Hierarchy

3 MIS of App Module

3.1 Uses

Game

3.2 Interface Syntax

3.2.1 Exported Types

App = ?

3.2.2 Exported Access Programs

Name	In	Out	Exceptions
App	Game	App	InvalidInput
render	-	Screen	-

3.3 Interface Semantics

3.3.1 State Variables

game: Game

3.3.2 Environmental Variables

Screen

3.3.3 Assumptions

The constructor App is called for the object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

3.3.4 Access Program Semantics

App(*game*):

- transition: $game := game$
- output: $out := self$
- exception := exc := ((typeof(game) $\neq$ Game) $\Rightarrow$ InvalidInput)

render():

- output := output each component in the Game module with the expected image in the image folder to the screen.
- exception := None

4 MIS of Card Module

4.1 Interface Syntax

4.1.1 Exported Types

Card = ?

4.1.2 Constants

type = [””, ”A”, ”2”, ”3”, ”4”, ”5”, ”6”, ”7”, ”8”, ”9”, ”10”, ”J”, ”Q”, ”K”]
suits = [”D”, ”C”, ”H”, ”S”]
SuiteVal = [1, 2, 3, 4]
suitsPath = [”Diamonds”, ”Clubs”, ”Hearts”, ”Spades”]
valuesPath = [””, ”Ace”, ”2”, ”3”, ”4”, ”5”, ”6”, ”7”, ”8”, ”9”, ”10”, ”Jack”, ”Queen”, ”King”]

4.1.3 Exported Access Programs

Name	In	Out	Exceptions
Card	Z, Z	Card	InvalidInput ∨ FileNotFound
image	-	Screen	-

4.2 Interface Semantics

4.2.1 State Variables

type:String
suit:String
suitVal:int
value:int
image:String A string contains the location of the image file for the card

4.2.2 Environmental Variables

Screen

4.2.3 Assumptions

The constructor Card is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

4.2.4 Access Program Semantics

Card($type\_num, suit\_num$):

- transition: $type, suit, suitVal, value, image :=$
  $type[type\_num], suits[suit\_num], suitVal[suit\_num],$
  $((type\_num == 1) \Rightarrow 14)|((type\_num == 2) \Rightarrow 15)|type\_num),$
  $"NAP - 01\_" + suitsPath[suit\_num] + "\_" + valuesPath[type\_val] + ".png"$
- output: $out := self$
- exception := $exc :=$ ((typeof(suite) $\neq$ SuiteT) $\vee$ (typeof(num) $\neq$ NumT) *lor* (typeof(image) $\neq$ String) $\Rightarrow$ InvalidInput) $\wedge$ (can not find file at image location $\Rightarrow$ FileNotFound)

image()

- output := $out :=$ image
- exception := None

5 MIS of Player Module

5.1 Uses

Card

5.2 Interface Syntax

5.2.1 Exported Types

None

5.2.2 Exported Access Programs

Name	In	Out	Exceptions
Player	seq of Card	Player	-
selectCard	Card	-	-
handlePlayClick	mouse click on screen	-	InvalidCombination
handlePassTurnClick	mouse click on screen	-	-
handleTypeSort	-	-	-
handleTypeSort	-	-	-

5.3 Interface Semantics

5.3.1 State Variables

prop: seq of Card the cards that the player has
selectedCards: seq of Card

5.3.2 Environmental Variables

Mouse, Screen

5.3.3 Assumptions

The constructor Player is called for the object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

5.3.4 Access Program Semantics

Player(*props*):

- transition: $props, selectedCards := props, []$
- output: $out := self$

selectCard(e):

- transition: $selectedCards :=$ the card(s) that is(are) clicked on the screen
- exception: None

handlePlayClick(e):

- transition: remove the card(s) selected by mouse clicking from the selectedCards and props, and deal out the selected cards on screen
- exception: $exc :=$ if the selecetd cards are not valid $\Rightarrow$ InvalidCombination

handlePassTurnClick(e):

- transition: pass the turn for the player in the game
- exception: None

handleTypeSort():

- transition: sort *props* in the number type value order, from smallest to largest.
- exception: None

handleSuitSort():

- transition: sort *props* in the Suite value order, from smallest to largest.
- exception: None

6 MIS of PlayerBot Module

6.1 Uses

Rules

6.2 Interface Syntax

6.2.1 Exported Types

6.2.2 Exported Access Programs

Name	In	Out	Exceptions
BotPlayCards	Sequence of Card, Sequence of Card	Sequence of Card	
BotStartingTurn	Sequence of Card	Card	
BotFreeTurn	Sequence of Card	Sequence of Card	
BotSelectSingle	Sequence of Card, Sequence of Card	Card	
BotSelectPair	Sequence of Card, Sequence of Card	Sequence of Card	
BotSelectFive	Sequence of Card , Sequence of Card	Sequence of Card	

6.3 Interface Semantics

6.3.1 Access Program Semantics

BotPlayCards(s, l):
Input: A list of cards owned by the current playerBot. A list of cards played by the last player.
Transition:None
Output: out = cards where If l.length == 1, cards = checkSingle(s, l).
If l.length == 2, cards = checkPair(s, l).
If l.length == 5, cards = checkFive(s, l).
Exceptions: None

BotStartingTurn(s):
Input: None
Transition: None
Output: out := Card representing Diamond 3 if s contains Diamond 3
Exceptions: None

BotFreeTurn(s):
Input: A sequence of Card
Transition: None
Output: out = Valid cards to be played
Exceptions: None

BotSelectSingle(s, l):
Input: A list of cards owned by the current playerBot. A list of cards played by the last player.
Transition: None
Output: Valid Card to be played.
Exceptions: None

BotSelectPair(s, l):
Input: A list of cards owned by the current playerBot. A list of cards played by the last player.
Transition: None
Output: Valid pair of Cards to be played.
Exceptions: None

BotSelectFive(s, l):
Input: A list of cards owned by the current playerBot. A list of cards played by the last player.
Transition: None
Output: Valid combination of Cards to be played.
Exceptions: None

7 MIS of Rules Module

7.1 Uses

7.2 Interface Syntax

7.2.1 Exported Access Programs

Name	In	Out	Exceptions
newDeck		Sequence of Card	
shuffle	Sequence of Card	Sequence of Card	
isValidStartingPlay	Sequence of Card	boolean	
isValidPlay	Sequence of Card	boolean	
isValidSingle	Sequence of Card	boolean	
isVaildPair	Sequence of Card	boolean	
isValidFiveCardPlay	Sequence of Card	boolean	
isVaildStraight	Sequence of Card	boolean	
isVaildFlush	Sequence of Card	boolean	
isValidFullHouse	Sequence of Card	boolean	
isValidFourOfaKind	Sequence of Card	boolean	
isStrongerPlay	Sequence of Card	boolean	
isStrongerSingle	Card	boolean	
isStrongerPair	Sequence of Card	boolean	
isStrongerFive	Sequence of Card	boolean	
setUserCards	Sequence of Card	Sequence of Card	
setFirstTurn	Sequence of Card, Sequence of Card Sequence of Card, Sequence of Card	boolean	
getSuitValue	char	int	
sortCardsValue	Sequence of Card		
sortCardsSuite	Sequence of Card		

7.3 Interface Semantics

7.3.1 State Constants

suitsPath = [”Diamonds”, ”Clubs”, ”Hearts”, ”Spades”]
valuesPath = [””, ”Ace”, ”2”, ”3”, ”4”, ”5”, ”6”, ”7”, ”8”, ”9”, ”10”, ”Jack”, ”Queen”, ”King”]
suits = [”D”, ”C”, ”H”, ”S”]
SuiteVal = [1, 2, 3, 4]
type = [””, ”A”, ”2”, ”3”, ”4”, ”5”, ”6”, ”7”, ”8”, ”9”, ”10”, ”J”, ”Q”, ”K”]

7.3.2 Assumptions

shuffle(deck) must be called before setUserCards().

7.3.3 Access Program Semantics

newDeck():
Input: None
Transition : None
Ouput : out := ($\forall$ i, j : **N**, c : Card | i $\in$ [0..3], j $\in$ [0..12] : Deck.add(c) where c.suite := Suite[i] $\wedge$ c.num := Num[j])
Exceptions: None

shuffle(deck):
Input: A sequence of Card
Transition : None
Output: output := deck by rearrange the order of Cards in deck randomly.
Exceptions: None

isValidStartingPlay(s):
Input: A list of the Cards owned by the current player.
Transition: None
Output : out := ($\exists$ i : **N**| i $\in$ [0..s.length] : s[i].suite == 'Diamond' $\wedge$ s[i].num == '3') $\wedge$ isValidPlay(s) $\Rightarrow$ true Exceptions: None

isValidPlay(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := isValidSingle(s) $\vee$ isValidPair(s) $\vee$ isValidFiveCardPlay(s) $\Rightarrow$ true
Exceptions: None

isValidSingle(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := s.length == 1
Exceptions: None

isValidPair(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := s.length == 2 $\wedge$ s[0].num == s[1].num
Exceptions: None

isValidFiveCardPlay(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := s.length == 5 $\wedge$ (isValidStraight(s) $\vee$ isValidFlush(s) $\vee$ isValidFullHouse(s) $\vee$ isValidFourOfaKind(s)) $\Rightarrow$ true
Exceptions: None

isValidStraight(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := s.length == 5 $\wedge$ ($\forall$ i : $\mathbf{N}$| i $\in$ [0..3] : NumSort(s)[i+1].num == NumSort(s)[i].num + 1)
Exceptions: None

isValidFlush(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := s.length == 5 $\wedge$ ($\forall$ i : $\mathbf{N}$| i $\in$ [1..4] : s[i].suite == s[0].suite)
Exceptions: None

isValidFullHouse(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := s.length == 5 $\wedge$ (NumSort(s)[0] == NumSort(s)[1] $\wedge$ NumSort(s)[3] == NumSort(s)[4] $\wedge$ (NumSort(s)[2] == NumSort(s)[1] $\vee$ NumSort(s)[2] == NumSort(s)[3]))
Exceptions: None

isValidFourOfaKind(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := s.length == 5 $\wedge$ ((NumSort(s)[0] == NumSort(s)[1] $\wedge$ NumSort(s)[0] == NumSort(s)[2] $\wedge$ NumSort(s)[0] == NumSort(s)[3]) $\vee$ (NumSort(s)[4] == NumSort(s)[1] $\wedge$ NumSort(s)[4] == NumSort(s)[2] $\wedge$ NumSort(s)[4] == NumSort(s)[3]))
Exceptions: None

isStrongerPlay(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : $\neg$ (s.length == last.length) $\Rightarrow$ false
s.length == 1 $\wedge$ isStrongerSingle(s[0], last[0]) $\Rightarrow$ true
s.length == 2 $\wedge$ isStrongerPair(s, last) $\Rightarrow$ true
s.length == 5 $\wedge$ isStrongerFive(s, last) $\Rightarrow$ true

¬ (s.length ∈ [1, 2, 5]) ⇒ false
Exceptions: None

isStrongerSingle(s):
Input: A Card selected by the current player.
Transition: None
Output : out := s.num > last[0].num ∨ (s.suite > last[0].suite ∧ s.num == last[0].num)
Exceptions: None

isStrongerPair(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := isValidPair(s) ∧ s[0].num ¿ last[0].num ∨ (SuiteSort(s)[1].suite > Suite-Sort(last)[1].suite ∧ s[0].num == last[0].num) Exceptions: None

isStrongerFive(s):
Input: A list of Cards selected by the current player.
Transition: None
Output : out := isValidFourOfaKind(s) ∧ isValidFullhouse(last) ⇒ true
isValidFourOfaKind(s) ∧ isValidFlush(last) ⇒ true
isValidFourOfaKind(s) ∧ isValidStraight(last) ⇒ true
isValidFullHouse(s) ∧ isValidStraight(last) ⇒ true
isValidFullHouse(s) ∧ isValidFlush(last) ⇒ true
isValidFullFlush(s) ∧ isValidStraight(last) ⇒ true
isValidStraight(s) ∧ isValidStraight(last) ∧ NumSort(s)[4].num > NumSort(last)[4].num ⇒ true
isValidFlush(s) ∧ isValidFlush(last) ∧ s[0].suite > last[0].suite ⇒ true
isValidFullHouse(s) ∧ isValidFullHouse(last) ∧ (NumSort(s)[3].num > NumSort(last)[3].num) ⇒ true
isValidFourOfaKind(s) ∧ isValidFourOfaKind(last) ∧ (NumSort(s)[4].num > NumSort(last)[4].num ∨ NumSort(s)[0].num > NumSort(last)[0].num) ⇒ true
Exceptions: None

setPalyerCards(deck):
Input: Shuffled deck
Transition:None
Output: out := s where (∀ i : **N** | i ∈ [0..12]: s.push(deck.pop()))
Exceptions: None

setFirstTurn(player, left, top, right):
Input: A sequence of Card owned by each of the four player
Transition:None

Output: out:= turn where turn = 'player' if Diamond 3 is in player
turn = 'left' if Diamond 3 is in left
turn = 'top' if Diamond 3 is in top
turn = 'right' if Diamond 3 is in right
Exceptions: None

getSuitValue(suit):
Input: a char representing suit type
Output: out := 1 if suit == 'D'
2 if suit == 'C'
3 if suit == 'H'
4 if suit == 'S'
Exceptions: None

sortCardsValue(s):
Input: A list of Cards
Transition: None
Output : out := a list of Cards from s sorted in the number rank order
Exceptions: None

sortCardsSuit(s):
Input: A list of Cards
Transition: None
Output : out := a list of Cards from s sorted in the suit rank order
Exceptions: None

8 MIS of Game Module

8.1 Interface Syntax

8.1.1 Exported Types

Game = ?

8.1.2 Exported Access Programs

Name	In	Out	Exceptions
Game			
startGame			
resetGame			
handleTimer			
handlePlayerDeal	sequence of Card	Boolean	
AIplayCards			
getCardsforTurn		sequence of Card	
updateNextTurnCards	sequence of Card		
updateField	sequence of Card		
updateNextTurn			
handlePlayPass			
typeSort			
suitSort			
isGameOver			
displayPass			
computePlayerScore			

8.2 Interface Semantics

8.2.1 State Variables

rules: boolean
playerCards: sequence of Cards
leftCards: sequence of Cards
topCards: sequence of Cards
rightCards: sequence of Cards
playerField: sequence of Cards
leftField: sequence of Cards
topField:sequence of Cards
rightField: sequence of Cards
startingTurn: boolean
playerScore: int
turn: String
minutes: int
seconds: int
cardsPlayed: sequence of Cards
lastMove: sequence of Cards
playerLastMove: sequence of Cards
freeMove: boolean
gameOver: boolean

8.2.2 Environmental Variables

Screen

8.2.3 Assumptions

The constructor of Game is called only as a React component. The access routines cannot be called outside of the scope.

8.2.4 Access Program Semantics

Game():
Input: None.
Transition: Initialize the state variables for object Game:

- rules:= true
- playerCards, leftCards, topCards, rightCards := []
- playerField, leftField, topField, rightField := []
- startingTurn:= true
- turn:= null
- playerScore:= 0
- cardsPlayed:= []
- lastMove:= []
- gameOver:= false
- freeMove:= false
- minutes:= 10
- seconds:= 0
- lastMovePlayer:= null

Output: None
Exceptions: None

startGame()
Input: None

Transition: Sets rules to false, start the game.
Output: None
Exceptions: None

resetGame()
Input: None
Transition: Resets the game to their initial states. Set seach player's deck to the randomly generated sequence of card.
Output: None
Exceptions: None

handleTimer()
Input:
Transition: Sets gameOver to be true.
Output: None
Exceptions: None

handlePlayerDeal()
Input: cards
Transition: playerFieldText:= "". $\neg validPlay(cards) \Rightarrow$ playerFieldText = "starting turn must be valid and contain 3 of diamonds"
Output: None
Exceptions: None

AIplayCards()
Input: None
Transition: Computes playableCards and update next turn.
Output: None
Exceptions: None

getCardsforTurn()
Input: None
Transition: None
Output: out := $((turn \equiv "left" \Rightarrow$ leftCards) $\cup$ $(turn \equiv "top" \Rightarrow$ topCards) $\cup$ $(turn \equiv "right" \Rightarrow$ rightCards) $\cup$ $(turn \equiv "player" \Rightarrow$ playerCards))
Exceptions: None

updateNextTurnCards(cards)
Input: cards: Sequence of cards.
Transition:

- cardsPlayed := cardsPlayed
- lastMove := cards

- lastMovePlayer := turn
- freeMove :=

Output: None
Exceptions: None

updateField(cards)
Input: cards: Sequence of cards.
Transition: ($turn \equiv$ "$opponentLeft$" $\Rightarrow$ opponentLeftField := cards) $\cup$ ($turn \equiv$ "$opponentTop$" $\Rightarrow$ opponentTopField := cards) $\cup$ ($turn \equiv$ "$opponentRight$" $\Rightarrow$ opponentRightField := cards) $\cup$ ($turn \equiv$ "$player$" $\Rightarrow$ playerField := cards)

Output: None
Exceptions: None

updateNextTurn()
Input: None
Transition: ($turn \equiv$ "$opponentLeft$" $\Rightarrow$ turn := "player") $\cup$ ($turn \equiv$ "$opponentTop$" $\Rightarrow$ turn := "opponentLeft") $\cup$ ($turn \equiv$ "$opponentRight$" $\Rightarrow$ turn := "opponentRight") $\cup$ ($turn \equiv$ "$player$" $\Rightarrow$ turn := "opponentRight")
Output: None
Exceptions: None

handlePlayerPass()
Input: None
Transition: ($startingTurn \Rightarrow$ playerFieldText := "You cannot pass the first turn") $\cup$ ($\neg startingTurn \Rightarrow$ playerFieldText := "")
Output: None
Exceptions: None

numberSort()
Input: None
Transition: playerCards := playerCards.sortCardsValue()
Output: None
Exceptions: None

suitSort()
Input: None
Transition:playerCards := playerCards.sortCardsSuit()
Output: None
Exceptions: None

isGameOver()
Input: None
Transition: $(len(currentPlayerCards) \equiv 0 \Rightarrow$ gameOver := true
Output: out := $(len(currentPlayerCards) \equiv 0 \Rightarrow$ true)
Exceptions: None

displayPass()
Input: None
Transition: Display a text to the user to indicate pass turn.
Output: None
Exceptions: None

computePlayerScore()
Input: None
Transition: None
Output: ceil((13 - playerCards.length) * (100 / 13))
Exceptions: None

9 MIS of GameplayField Module

9.1 Uses

Game

9.2 Interface Syntax

9.2.1 Exported Types

9.2.2 Exported Access Programs

Name	In	Out	Exceptions
render		HTML Card	

9.3 Interface Semantics

9.3.1 State Variables

9.3.2 Environmental Variables

Screen

9.3.3 Assumptions

9.3.4 Access Program Semantics

render()
Input: None
Transition: None
Output: Arrangement of players in gameplay field.
Exceptions: None