Assignment 1
ECSE321 SOFTWARE ENG.

By
Simon Foucher
260 223 197

Feb 16th 2009
McGill University

Description of the system

The system consists of a blackjack table, players, a dealer and a deck of cards. These elements interact with each other according to blackjack rules defined in the system's requirements. The design has been split up into 5 classes, described in the system's architecture.

List of Requirements

(extracted form Project proposal document)

- Table must accommodate a dealer + 7 players
- At the beginning of each round, each player's account gets deducted 10$ and receives an initial hand of two cards. The dealer receives a single card
- Cards are valued as follows: from 2 – 9 = face value, 10 or face = 10, Aces = 1 or 11
- The players goes first, able to take additional cards desired. If player busts, he can no longer take cards for the busted hand
- Then the dealer plays his hand automatically. If the dealer busts, he loses to all remaining players.
- At game Start, the number of players is decided (2 to 7) with each supplying a name (if none provided, called Player#, with # the number 1-7. Each player is given 100$ to play with.
- At round Start, Each player's bank total is subtracted 10$ if available, otherwise kicked out. Each player gets 2 cards, dealer gets 1
- Play begins with the first available player.
- The player is given up to 5 options: Hit, Stand, Double Down, Split, & Insurance
    1. Hit: The player is dealt one extra card, whose value is added to the player's current card total. Player can hit as long as hand < 22, otherwise play switches to the next available player.
    2. Stand: The player decides to keep his current card. Play switches to the next available player.
    3. Double Down: If player's account >=10, he is dealt one extra card, deducted 10$ then play is passed.
    4. Split: Only available when a player is dealt a pair of cards and account >= 10. 10$ gets deducted, one of the cards goes into a new split hand. Can only be used 1 per round, per player.
    5. Insurance: If Dealer's card = Ace and player's account >= 5, get deducted 5$ and if dealer gets blackJack, player receives 10$
- Dealer plays automatically at the end of the round. Dealer hits on 16, and stands on 17.
- Payouts: If the dealer busts, all non-busted players win their bet. If tie, all players equaling the dealer get their bet back Blackjack is a hand wherein the two initial cards add up to 21. The only way of doing this is with a K, Q, J, or 10 with an Ace. Provided the dealer does not get a blackjack as well, the payout is 3:2, versus the typical 1:1.

System's Architecture

**Had I been aware of the code volume involved with GUI, I would have separated the user interface and the game flow into 2 separate classes. Instead, they are both grouped in the principle class: Table.java

<u>Cards.java</u>
Main object abstracted: deck of cards

Physical Attributes
deckOfCards: array of 52 integers

Methods
getNextCard(): returns the next card on top of the deck, shuffles it if empty

setIndexCard(int card, int Index): Force insert the card specified in the text field Card # and inserts it at Index

shuffleDeck(): Shuffles the deck of cards using java.util.Random. Also checks for duplicates to ensure that every single card is in the deck only once.

toString(int cardValue): Inputs a card value, returns the car's name as a String

<u>Dealer.java</u>
Main object abstracted: Blackjack Dealer

Physical Attributes
int hand[11] : Array containing 11 integers (maximum number of cards before busting)

String mainHand: A strung representation of the content of the hand

Logical attributes
boolean canHit: Manages weather the dealer can hit his main hand or not

Methods
resetHand(): Deletes all the cards in the hand, set String mainHand = "Empty Hand"

getHandValue(int handToCalculate[]): Imputs the hand to calculate as a parameters (used to compute either main or split hand), computes how much the hand is worth. (Aces are taken as worth 11 unless hand value < 21; then they take 1 as a value). Returns the value of the hand.
hitHand(int card): inserts the card in the parameter card in the main Hand

convCardValue(int cardValue): Inputs a card value and converts it into a string. Returns that string (called by function stringHand)

stringHand(int handToConv[]): Inputs a hand, calls convCardValue for every card and returns the string equivalent of the hand

Players.java (extends Dealer.java)
Main object abstracted: Blackjack players

Added Physical attributes
int splitHand[11]: Array containing 11 integers (maximum number of cards before busting)

String splitHandString: A string representation of the split hand
int account: To record account balance
String playerName: TO record player's name

Added Logical attributes
boolean canSplit
boolean didSplit
boolean canHitSplit
boolean bankrupt
boolean doubleDown
boolean canInsure
boolean didInsure

Added Methods
getPaid(int amount): Adds the amount to the player's account balance
split(): If the player can split this method performs the split
hitSplitHand(int card): Hits the split hand with the card provided
resetHand(): Overrides dealer's reserHand(); this method also resets the splitHand

HighScoreRecord.java
Main object abstracted: Data Record for high scores

Physical Attributes
int scorePtr: A pointer to the Data Record arrays
int score[10] : 10 deep integer array to record the scores
String name[10] : 10 deep String array to record player's names

Methods
insertScore(String newName, int newScore): Inserts new name and score in the Data structure if needed using insertion sort algorithm. The score is used as a sorting index and the name array follows the score array's behavior such that the scores are always sorted, and that names and scores at any given index match.

String stringNames(): Reads the name[10] array content and exports it in a one name per row single String

stringScores(): Reads the score[10] array content and exports it in a one score record per row single String

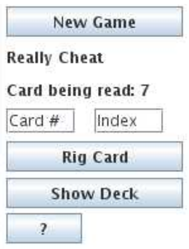
*When displaying high scores, the names are displayed in one text field and the scored are displayed in a second one immediately to the right of the names.

Table.java (extends Jframe, imports swing graphics libraries and awt colors)

Main object abstracted: Blackjack table (game logic and GUI)

Physical attributes

Game attributes:



New Game button: When pressed, activates a flag that is read in every while loops of the game. Triggers a new game to start

Card Being read: Shows the value of card index (the next card to be dealt)

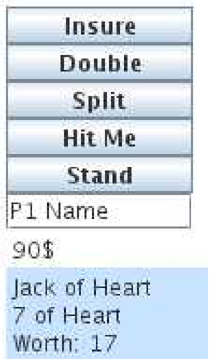
Card # and Index text fields: let user input a card number and an index of where to insert it.

Rig Card button: Inserts card value entered in 'card #' at 'Index' in the deck

Show Deck button: Displays content of deck of cards

'?' Button: Gives information on how to cheat

Player attributes:



Insurance, Double, Split, Hit Me and Pass buttons: behave as required by the system requirements when it is that player's play

P1 Name: Text field to enter player's names. Will be read to record High scores

There are also an account balance field, and one field to display the content of each hand (regular and split if need be)

Logical attributes

boolean newGame: notifies when new game button is pressed

boolean gameOver: traps main in a while loop displaying high scores

Methods

updateGui(): Updates all the GUI output elements. Displays account balances or 'BANKRUPT!', transform player's hand content into strings and displays them.

initiatePlayer(Player newPlayer): Called at the start o every rounds. Resets the player's hand, deduct 10$ from his account, hit main hand with 2 cards, reset game flow boolean variables (canHit = true, didSplit = false, canHitSplit = false, doubleDown = false, canInsure = false, didInsure = true) then calls updateGui();

doMainHand(Player player): Handles the game for the player's main hand by listening to the appropriate buttons. First detect if player is in condition to split and insure, and if so, activates both those buttons via boolean flags. Afterwards, player.canHit is set to true and the function enters a while(canHit) loop. In this loop, it looks at the newGame flag (and exits if true), calls updateGui() so the player is notified of his actions and then the thread is put to sleep for 200mS (to avoid eating up CPU resources, short enough that human player does not notice). Finally, the loop checks if the player busted and exits of so. The loop will also exit if player hits "stand" button, which will reset canHit = false.

doSplitHand(Player player): Very similar to doMainHand, except that button actions are applied to the player's split hand. Also, the 'insurance', 'split' and 'double' buttons are ignored.

doAccounting(Player player): Called at the end of every round, this function determines the payouts to be made. It starts by setting doneHand = false, doneSplit = true and doubleBet = 1. If player did double, doubleBet is set to 2. Whenever paying out, the payout amount is always multiplied by doubleBet. Then the function looks at weather the player didSplit, and if so, doneSplit os set to false.

Afterwards, the thread looks at different payout scenarios based on the requirements by running various if(condition AND doneHand is False) statements. As soon as a payout id pame, doneHand/doneSplit is set to true and every other payout cases are skipped because of this flag. After the payout, the thread is paused for 100ms. Since the background color of account Balance text field got set to green prior to entering the function, this draws player's attention to the accounts being updated.

Finally, the function checks if the player is bankrupt, and flags player.bankrupt = true if it is the case.

main(String[] args): First it created a Table object, then enters a while(true loop). Inside this loop, it starts by pausing the thread to reduce CPU load, the looks to see if newGame button got pressed since last iteration.

If so, a text is displayed on the screen to advise user and the thread is paused for 900ms to let the player read it. All bank accounts are set back to 100$ and bankrupt flags are removed. If the deck has not been rigged by the user, the thread shuffles the cards
the remove the RigDeck and newGame flags. Weather the deck got rigged or not, the card index is set to 0. This allows the user to simply rig the first card of the deck. Since the getNextCard function ignores 'empty' cards, a user can rig the first 2 cards of the deck, have the next 20 empty and still play a regular game.

After the newGame overheads, the regular round thread starts. The hands of the players and dealer are reset, and their background colors set to white. Functions initiatePlayer is called for every player that is not bankrupt, the dealer gets his initial deal and the GUI is updated.

Now the human interactive game flow starts and is identically repeated for every player who is not bankrupt. The player's hand's background is set to light gray and function doMainHand is called.
It will exit if the player busts or hits 'Stand' and the background is reset to white. Then we detect a split condition and if so, doSplitHand() is called and the splitHand's background color is set to light grey, to indicate to user that he is now in his split hand. This is repeater for all non bankrupt players.

Afterwards, if newGame did not get pressed and if the flag GameOver if false, the dealer's hand is dealt according to requirements and the accounting is performed on all non bankrupt players by calling doAccounting

Then we check for a game Over condition by looking to see if there is only a single player not Bankrupt. If so, gameOver is set to true and that player's name and score are fed to insertScore function, which will determines if player made it to high scores, and if so records his data.

Finally there is a while(gameOver & newGame == false) loop to catch the main thread in a game over state. In this state, every player's hands are reset, high scores are displayed as well as a test saying "Game Over! Player x Wins". This thread is also paused to avoid overworking the CPU and can only exit when the user presses the new Game button.

Outline of how requirements are fulfilled

Every requirements have been fulfilled. There were some ambiguities on payout rules when mixing special cases. For example if player gets a blackjack, dealer gets a blackjack and insurance was purchased, how much does the player get back? Or if a player can purchase insurance after he has hit, or weather or not he can purchase insurance on a split hand, etc... These issues have been resolved based on subjective interpretation of the requirements and are subject for change after first release based on user feedback. Since these conditions are rare, incorrect results shouldn't affect the regular game flow most of the time.

Since this software is in beta release and might still contain bugs or require graphics modifications, the option of choosing the number of players has been left for the next release. The default fixed number of player is 3 for the time being. It would be simple enough to implement a chose player option by simply copying and pasting the code for players and adding a prompt on new games for the number of players, but since the player's specs are prompt to change, it would be simpler to receive feed back, modify 3 players then afterwards copy the code, rather than copy the code, then modify it 7 times based on more specific requirements.

Besides a few ambiguities briefly described earlier, as well as the absence of a choosing the number of players option, every other requirements have been followed to the best of my knowledge (as a non blackjack player).