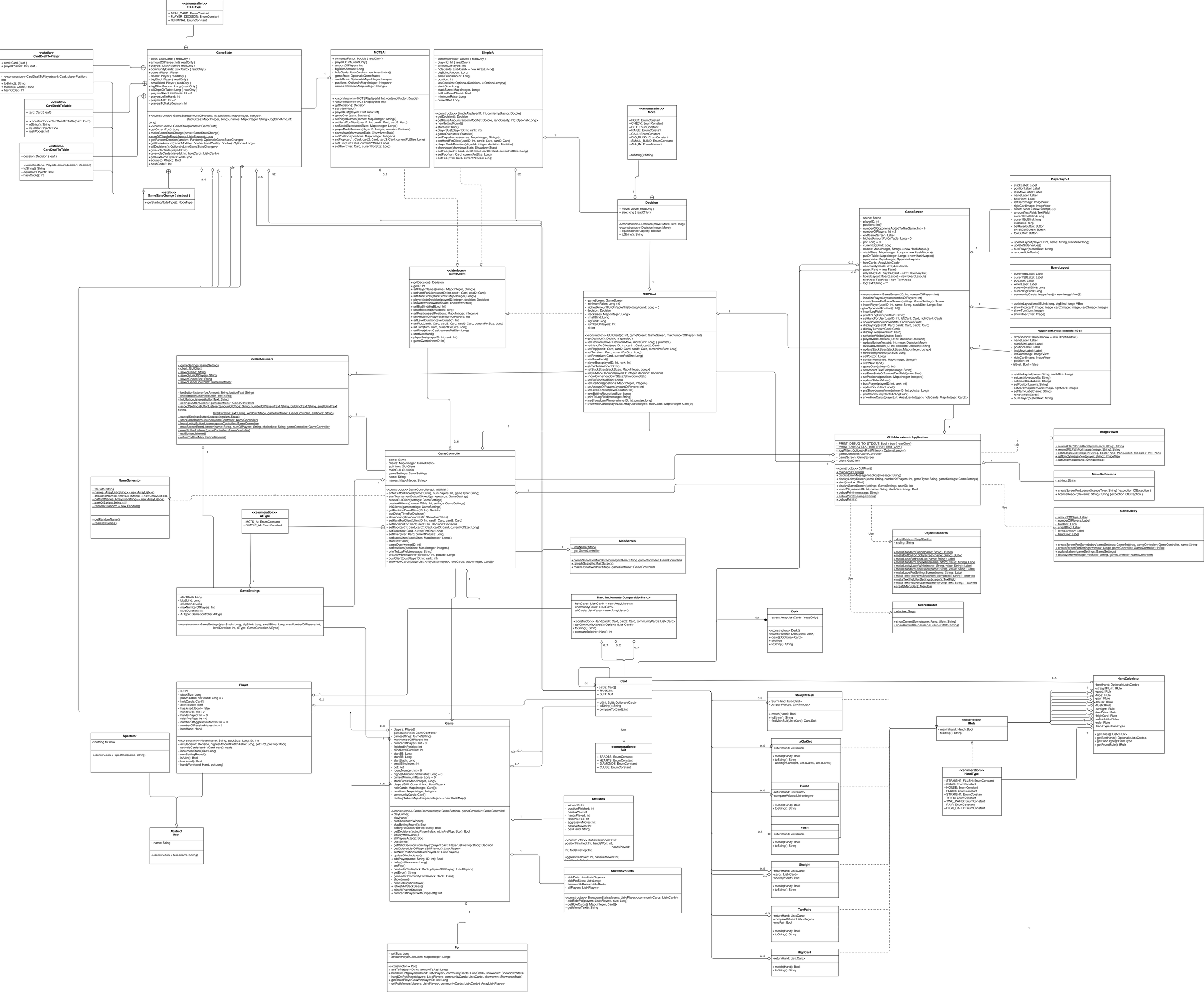
<<enumeration>>
NodeType
+ DEAL_CARD: EnumConstant
+ PLAYER_DECISION: EnumConstant
+ TERMINAL: EnumConstant
<<static>>
CardDealToPlayer
+ card: Card { leaf }
+ playerPosition: Int { leaf }
- <<constructor>> CardDealToPlayer(card: Card, playerPosition: Int)
+ toString(): String
+ equals(o: Object): Bool
+ hashCode(): Int
<<static>>
CardDealToTable
+ card: Card { leaf }
+ <<constructor>> CardDealToTable(card: Card)
+ toString(): String
+ equals(o: Object): Bool
+ hashCode(): Int
<<static>>
CardDealToTable
+ decision: Decision { leaf }
+ <<constructor>> PlayerDecision(decision: Decision)
+ toString(): String
+ equals(o: Object): Bool
+ hashCode(): Int
GameState
- deck: List<Card> { readOnly }
+ amountOfPlayers: Int { readOnly }
+ players: List<Player> { readOnly }
+ communityCards: List<Card> { readOnly }
+ currentPlayer: Player
- dealer: Player { readOnly }
- bigBlind: Player { readOnly }
- smallBlind: Player { readOnly }
+ bigBlindAmount: Long { readOnly }
+ allChipsOnTable: Long { readOnly }
- playersGivenHoleCards: Int = 0
- playersLeftInHand: Int
- playersAllIn: Int = 0
- playersToMakeDecision: Int
+ <<constructor>> GameState(amountOfPlayers: Int, positions: Map<Integer, Integer>, stackSizes: Map<Integer, Long>, names: Map<Integer, String>, bigBlindAmount: Long)
+ <<constructor>> GameState(oldState: GameState)
+ getCurrentPot(): Long
+ makeGameStateChange(move: GameStateChange)
+ isAllOfChipsInPlay(players: List<Player>): Long
+ getRandomDecision(random: Random): Optional<GameStateChange>
+ getRaiseAmount(randomModifier: Double, handQuality: Double): Optional<Long>
+ allDecisions(): Optional<List<GameStateChange>>
+ giveHoleCards(playerId: Int)
+ giveHoleCards(playerId: Int, holeCards: List<Card>)
+ getNextNodeType(): NodeType
+ equals(o: Object): Bool
+ hashCode(): Int
<<static>>
GameStateChange { abstract }
+ getStartingNodeType(): NodeType
MCTSAI
- contemptFactor: Double { readOnly }
- playerID: Int { readOnly }
- amountOfPlayers: Int
- bigBlindAmount: Long
- holeCards: List<Card> = new ArrayList<>()
- gameState: Optional<GameState>
- stackSizes: Optional<Map<Integer, Long>>
- positions: Optional<Map<Integer, Integer>>
- names: Optional<Map<Integer, String>>
+ <<constructor>> MCTSAI(playerId: Int, contemptFactor: Double)
+ <<constructor>> MCTSAI(playerId: Int)
+ getDecision(): Decision
+ startNewHand()
+ playerBust(playerID: Int, rank: Int)
+ gameOver(stats: Statistics)
+ setPlayerNames(names: Map<Integer, String>)
+ setHandForClient(userID: Int, card1: Card, card2: Card)
+ setStackSizes(stackSizes: Map<Integer, Long>)
+ playerMadeDecision(playerID: Integer, decision: Decision)
+ showdown(showdownStats: ShowdownStats)
+ setPositions(positions: Map<Integer, Integer>)
+ setFlop(card1: Card, card2: Card, card3: Card, currentPotSize: Long)
+ setTurn(turn: Card, currentPotSize: Long)
+ setRiver(river: Card, currentPotSize: Long)
SimpleAI
- contemptFactor: Double { readOnly }
- playerId: Int { readOnly }
- amountOfPlayers: Int
- holeCards: List<Card> = new ArrayList<>()
- bigBlindAmount: Long
- smallBlindAmount: Long
- position: Int
- lastDecision: Optional<Decision> = Optional.empty()
- stackSize: Long
- stackSizes: Map<Integer, Long>
- betHasBeenPlaced: Bool
- minimumRaise: Long
- currentBet: Long
<<constructor>> SimpleAI(playerID: Int, comtemptFactor: Double)
+ getDecision(): Decision
+ getRaiseAmount(randomModifier: Double, handQuality: Int): Optional<Long>
+ newBettingRound()
+ startNewHand()
+ playerBust(playerID: Int, rank: Int)
+ gameOver(stats: Statistics)
+ setPlayerNames(names: Map<Integer, String>)
+ setHandForClient(userID: Int, card1: Card, card2: Card)
+ playerMadeDecision(playerId: Integer, decision: Decision)
+ showdown(showdownStats: ShowdownStats)
+ setFlop(card1: Card, card2: Card, card3: Card, currentPotSize: Long)
+ setFlop(turn: Card, currentPotSize: Long)
+ setFlop(river: Card, currentPotSize: Long)
<<enumeration>>
Move
+ FOLD: EnumConstant
+ CHECK: EnumConstant
+ BET: EnumConstant
+ RAISE: EnumConstant
> CALL: EnumConstant
> BIG_BLIND: EnumConstant
+ SMALL_BLIND: EnumConstant
+ ALL_IN: EnumConstant
+ toString(): String
Decision
+ move: Move { readOnly }
+ size: long { readOnly }
<<constructor>> Decision(move: Move, size: long)
<<constructor>> Decision(move: Move)
+ equals(other: Object): boolean
+ toString(): String
<<interface>>
GameClient
+ getDecision(): Decision
+ getID: Int
+ setPlayerNames(names: Map<Integer, String>)
+ setHandForClient(userID: int, card1: Card, card2: Card)
+ setStackSizes(stackSizes: Map<Integer, Long>)
+ playerMadeDecision(playerID: Integer, decision: Decision)
+ showdown(showdownStats: ShowdownStats)
+ setBigBlind(bigBLind: Int)
+ setSmallBlind(smallBlind: long)
+ setPositions(setPositions: Map<Integer, Integer>)
+ setAmountOfPlayers(amountOfPlayers: Int)
+ setLevelDuration(levelDuration: Int)
+ setFlop(card1: Card, card2: Card, card3: Card, currentPotSize: Long)
+ setTurn(turn: Card, currentPotSize: Long)
+ setRiver(river: Card, currentPotSize: Long)
+ startNewHand()
+ playerBust(playerID: Int, rank: Int)
+ gameOver(winnerID: Int)
GUIClient
- gameScreen: GameScreen
- minimumRaise: Long = 0
- highestAmountPutOnTableThisBettingRound: Long = 0
- decision: Decision
- stackSizes: Map<Integer, Long>
- smallBlind: Long
- bigBlind: Long
- numberOfPlayers: Int
- id: Int
<<constructor>> GUIClient(id: Int, gameScreen: GameScreen, maxNumberOfPlayers: Int)
+ getDecision(): Decision { guarded }
+ setDecision(move: Decision.Move, moveSize: Long) { guarded }
+ setHandForClient(userID: Int, card1: Card, card2: Card)
+ setFlop(card1: Card, card2: Card, card3: Card, currentPotSize: Long)
+ setTurn(turn: Card, currentPotSize: Long)
+ setRiver(river: Card, currentPotSize: Long)
+ startNewHand()
+ playerBust(playerID: Int, rank: Int)
+ gameOver(winnerID: Int)
+ setStackSizes(stackSizes: Map<Integer, Long>)
+ playerMadeDecision(playerID: Integer, decision: Decision)
+ showdown(showdownStats: ShowdownStats)
+ setBigBlind(bigBlind: Long)
+ setPositions(positions: Map<Integer, Integer>)
+ setAmountOfPlayers(amountOfPlayers: Int)
+ setLevelDuration(levelDuration: Int)
+ newBettingRound(potSize: Long)
+ printToLogField(message: String)
+ preShowdownWinner(winnerID: Int, potsize: long)
+ showHoleCards(playerList: ArrayList<Integer>, holeCards: Map<Integer, Card[]>)
GameScreen
- scene: Scene
- playerID: Int
- positions: Int[*]
- numberOfOpponentsAddedToTheGame: Int = 0
- numberOfPlayers: Int = 2
- endGameScreen: Label
- highestAmountPutOnTable: Long = 0
- pott: Long = 0
- currentBigBlind: Long
- names: Map<Integer, String> = new HashMap<>()
- stackSizes: Map<Integer, Long> = new HashMap<>()
- putOnTable: Map<Integer, Long> = new HashMap<>()
- opponents: Map<Integer, OpponentLayout>
- holeCards: ArrayList<Card>
- communityCards: ArrayList<Card>
- pane: Pane = new Pane()
- playerLayout: PlayerLayout = new PlayerLayout()
- boardLayout: BoardLayout = new BoardLayout()
- textArea: TextArea = new TextArea()
- logText: String = ""
<<constructor>> GameScreen(ID: Int, numberOfPlayers: Int)
- initializePlayerLayouts(numberOfPlayers: Int)
+ createSceneForGameScreen(settings: GameSettings): Scene
+ insertPlayer(userID: Int, name: String, stackSize: Long): Bool
- giveOpponentPosition(): Int[]
+ insertLogField()
+ printToLogField(printInfo: String)
+ setHandForUser(userID: Int, leftCard: Card, rightCard: Card)
+ showdown(showdownStats: ShowdownStats)
+ displayFlop(card1: Card, card2: Card, card3: Card)
+ displayTurn(turnCard: Card)
+ displayRiver(riverCard: Card)
+ setActionVisible(visible: Bool)
+ playerMadeDecision(ID: Int, decision: Decision)
- updateButtonTexts(id: Int, move: Decision.Move)
- evaluateDecision(ID: Int, decision: Decision): String
+ updateStackSizes(stackSizes: Map<Integer, Long>)
+ newBettingRound(potSize: Long)
+ setPot(pot: Long)
+ setNames(names: Map<Integer, String>)
+ startNewHand()
+ gameOver(winnerID: Int)
+ setAmountTextField(message: String)
+ setErrorStateOfAmountTextField(error: Bool)
+ setPositions(positions: Map<Integer, Integer>)
+ updateSliderValues()
+ bustPlayer(playerID: Int, rank: Int)
- updateYourHandLabel()
+ preShowdownWinner(winnerID: Int, potsize: Long)
- printCommunityCardsToLogField()
+ showHoleCards(playerList: ArrayList<Integer>, holeCards: Map<Integer, Card[]>)
PlayerLayout
- stackLabel: Label
- positionLabel: Label
- lastMoveLabel: Label
- nameLabel: Label
- bestHand: Label
- leftCardImage: ImageView
- rightCardImage: ImageView
- slider: Slider = new Slider(0,0,0)
- amountTextField: TextField
- currentSmallBlind: long
- currentBigBlind: long
- stackSize: long
- betRaiseButton: Button
- checkCallButton: Button
- foldButton: Button
+ updateLayout(playerID: int, name: String, stackSize: long)
+ updateSliderValues()
+ bustPlayer(bustedText: String)
+ removeHoleCards()
BoardLayout
- currentBBLabel: Label
- currentSBLabel: Label
- potLabel: Label
- winerLabel: Label
- currentSmallBlind: Long
- currentBigBlind: Long
- communityCards: ImageView[] = new ImageView[5]
+ updateLayout(smallBLind: long, bigBlind: long): VBox
+ showFlop(card1Image: Image, card2Image: Image, card3Image: Image)
+ showTurn(turn: Image)
+ showRiver(river: Image)
OpponentLayout extends HBox
- dropShadow: DropShadow = new DropShadow()
- nameLabel: Label
- stackSizeLabel: Label
- positionLabel: Label
- lastMoveLabel: Label
- leftCardImage: ImageView
- rightCardImage: ImageView
- position: Int
- isBust: Bool = false
+ updateLayout(name: String, stackSize: Long)
+ setLastMoveLabel(s: String)
+ setStackSizeLabel(s: String)
+ setPositionLabel(s: String)
+ setCardImage(leftCard: Image, rightCard: Image)
+ setNameLabel(name: String)
+ removeHoleCards()
+ bustPlayer(bustedText: String)
ButtonListeners
- gameSettings: GameSettings
- client: GUIClient
- savedName: String
- savedNumOfPlayers: String
- savedChoiceBox: String
- savedGameController: GameController
+ betButtonListener(betAmount: String, buttonText: String)
+ checkBUttonListener(buttonText: String)
+ foldButtonListener(buttonText: String)
+ settingsButtonListener(gameController: GameController)
+ acceptSettingsButtonListener(amountOfChips: String, numberOfPlayersText: String, bigBlindText: String, smallBlindText: String, levelDurationText: String, window: Stage, gameController: GameController, aIChoice: String)
+ cancelSettingsButtonListener(window: Stage)
+ startGameButtonListener(gameController: GameController)
+ leaveLobbyButtonListener(gameController: GameController)
+ mainScreenEnterListener(name: String, numOfPlayers: String, choiceBox: String, gameController: GameController)
+ errorButtonListener(gameController: GameController)
+ exitButtonListener()
+ returnToMainMenuButtonListener()
GameController
- game: Game
- clients: Map<Integer, GameClient>
- guiClient: GUIClient
- mainGUI: GUIMain
- gameSettings: GameSettings
- name: String
- names: Map<Integer, String>
<<constructor>> GameController(gui: GUIMain)
+ enterButtonClicked(name: String, numPlayers: Int, gameType: String)
+ startTournamentButtonClicked(gamesettings: GameSettings)
- createGUIClient(settings: GameSettings)
- createAIClients(numberOfAIs: Int, settings: GameSettings)
- initClients(gamesettings: GameSettings)
+ getDecisionFromClient(ID: Int): Decision
- addDelayTimeForDecision()
+ showdown(showdownStats: ShowdownStats)
+ setHandForClient(clientID: Int, card1: Card, card2: Card)
+ setDecisionForClient(userID: Int, decision: Decision)
+ setFlop(card1: Card, card2: Card, card3: Card, currentPotSize: Long)
+ setTurn(turn: Card, currentPotSize: Long)
+ setRiver(river: Card, currentPotSize: Long)
+ setStackSizes(stackSizes: Map<Integer, Long>)
+ startNewHand()
+ gameOver(winnerID: Int)
+ setPositions(positions: Map<Integer, Integer>)
+ printToLogField(message: String)
+ preShowdownWinner(winnerID: Int, potSize: Long)
+ bustClient(bustPlayerID: Int, rank: Int)
+ showHoleCards(playerList: ArrayList<Integer>, holeCards: Map<Integer, Card[]>)
GUIMain extends Application
- PRINT_DEBUG_TO_STDOUT: Bool = true { readOnly }
- PRINT_DEBUG_LOG: Bool = true { read_Only }
- logWriter: Optional<PrintWriter> = Optional.empty()
- gameController: GameController
- gameScreen: GameScreen
- client: GUIClient
<<constructor>> GUIMain()
+ main(args: String[])
+ displayErrorMessageToLobby(message: String)
+ displayLobbyScreen(name: String, numberOfPlayers: Int, gameType: String, gameSettings: GameSettings)
+ start(window: Stat)
+ displayGameScreen(settings: GameSettings, userID: Int)
+ insertPlayer(userID: Int, name: String, stackSize: Long): Bool
+ debugPrintln(message: String)
+ debugPrint(message: String)
+ debugPrintln()
ImageViewer
+ returnURLPathForCardSprites(card: String): String
+ return(URLPathFor(images(Image: String): String
+ setBackgroundImage(in: String, borderPane: Pane, sizeX: Int, sizeY: Int): Pane
+ getEmptyImageView(player: String): ImageView
+ getChipImage(name: String): Image
MenuBarScreens
- styling: String
+ createScreenForLicence(licenceType: String) { exception IOException }
+ licenceReader(fileName: String): String { exception IOException }
GameLobby
- amountOfChips: Label
- numberOfPlayers: Label
- bigBlind: Label
- smallBlind: Label
- levelDuration: Label
- headLine: Label
+ createScreenForGameLobby(gameSettings: GameSettings, gameController: GameController, name:String)
+ createScreenForSettings(window: Stage, gameController: GameController): HBox
+ updateLabels(gameSettings: GameSettings)
+ displayErrorMessage(message: String, gameController: GameController)
NameGenerator
- filePath: String
+ names: ArrayList<String> = new ArrayList<>()
+ characterNames: ArrayList<String> = new ArrayList<>()
+ pathOfSeries: ArrayList<String> = new ArrayList<>()
+ pathOfSeries: String = ""
+ random: Random = new Random()
+ getRandomName()
+ readNewSeries()
<<enumeration>>
AIType
+ MCTS_AI: EnumConstant
+ SIMPLE_AI: EnumConstant
GameSettings
- startStack: Long
- bigBLind: Long
- smallBlind: Long
- maxNumberOfPlayers: Int
- levelDuration: Int
- AIType: GameController.AIType
<<constructor>> GameSettings(startStack: Long, bigBlind: Long, smallBlind: Long, maxNumberOfPlayers: Int, levelDuration: Int, aiType: GameController.AIType)
MainScreen
- imgName: String
- gc: GameController
+ createSceneForMainScreen(imageNAme: String, gameController: GameController)
+ refreshSceneForMainScreen()
+ makeLayout(window: Stage, gameController: GameController)
ObjectStandards
- dropShadow: DropShadow
- styling: String
+ makeStandardButton(name: String): Button
+ makeButtonForLobbyScreen(name: String): Button
+ makeLabelFotHeadLine(name: String): Label
+ makeStandardLabelWhite(name: String, value: String): Label
+ makeLobbyLabelWhite(name: String, value: String): Label
+ makeStandardLabelBlack(name: String, value: String): Label
+ makeLabelForSettingsScreen(name: String): Label
+ makeTextFieldForMainScreen(promptText: String): TextField
+ makeTextFieldForSettingsScreen(): TextField
+ makeTextFieldForGameScreen(promptText: String): TextField
+ createMenuBar(): MenuBar
SceneBuilder
- window: Stage
+ showCurrentScene(pane: Pane, title: String)
+ showCurrentScene(scene: Scene, title: String)
Hand implements Comparable<Hand>
- holeCards: List<Card> = new ArrayList<>(2)
- communityCards: List<Card>
- allCards: List<Card> = new ArrayList<>()
<<constructor>> Hand(card1: Card, card2: Card, communityCards: List<Card>)
+ getCommunityCards(): Optional<List<Card>>
+ toString(): String
+ compareTo(other: Hand): Int
Deck
- cards: ArrayList<Card> { readOnly }
<<constructor>> Deck()
<<constructor>> Deck(deck: Deck)
+ draw(): Optional<Card>
+ shuffle()
+ toString(): String
Card
- cards: Card[]
+ RANK: int
+ SUIT: Suit
+ ofint, Suit): Optional<Card>
+ toString(): String
+ compareToCard): int
<<enumeration>>
Suit
+ SPADES: EnumConstant
+ HEARTS: EnumConstant
+ DIAMONDS: EnumConstant
+ CLUBS: EnumConstant
Player
- ID: Int
- stackSize: Long
- putOnTableThisRound: Long = 0
- holeCards: Card[]
- allIn: Bool = false
- hasActed: Bool = false
- handsWon: Int = 0
- handsPlayed: Int = 0
- foldsPreFlop: Int = 0
- numberOfAggressiveMoves: Int = 0
- numberOfPassiveMoves: Int = 0
- bestHand: Hand
<<constructor>> Player(name: String, stackSize: Long, ID: Int)
+ act(decision: Decision, highestAmountPutOnTable: Long, pot: Pot, preFlop: Bool)
+ setHoleCards(card1: Card, card2: card)
+ incrementStack(size: Long)
+ newBettingRound()
+ isAllIn(): Bool
+ hasActed(): Bool
+ handWon(hand: Hand, pot:Long)
Spectator
// nothing for now
<<constructor>> Spectator(name: String)
Abstract User
- name: String
<<constructor>> User(name: String)
Game
- players: Player[]
- gameController: GameController
- gamesettings: GameSettings
- maxNumberOfPlayers: Int
- numberOfPlayers: Int = 0
- finishedInPosition: Int
- blindLevelDuration: Int
- startSB: Long
- startBB: Long
- startStack: Long
- smallBlindIndex: Int
- pot: Pot
- roundNumber: Int = 0
- highestAmountPutOnTable: Long = 0
- currentMinimumRaise: Long = 0
- stackSizes: Map<Integer, Long>
- playersStillinCurrentHand: List<Player>
- holeCards: Map<Integer, Card[]>
- positions: Map<Integer, Integer>
- communityCards: Card[]
- rankingTable: Map<Integer, Integer> = new HashMap()
<<constructor>> Game(gamesettings: GameSettings, gameController: GameController)
+ playGame()
- playHand()
- preShowdownWinner()
- skipBettingRound(): Bool
- bettingRoundisPreFlop: Bool): Bool
- getDecisions(actingPlayerIndex: Int, isPreFlop: Bool): Bool
- displayHoleCards()
- allPlayersActed(): Bool
- postBlinds()
- getValidDecisionFromPlayer(playerToAct: Player, isPreFlop: Bool): Decision
- getOrderedListOfPlayersStillPlaying(): List<Player>
- setNewPositions(orderedPlayerList: List<Player>)
- updateBlindIndexes()
+ addPlayer(name: String, ID: Int): Bool
- delay(milliseconds: Long)
- setFlop()
- dealHoleCards(deck: Deck, playersStillPlaying: List<Player>)
- getError(): String
- generateCommunityCards(deck: Deck): Card[]
- showdown()
- printDebugShowdown()
- refreshAllStackSizes()
+ printAllPlayerStacks()
+ numberOfPlayersWithChipsLeft(): Int
Statistics
- winnerID: Int
- positionFinished: Int
- handsWon: Int
- handsPlayed: Int
- foldsPreFlop: Int
- aggressiveMoves: Int
- passiveMoves: Int
- bestHand: String
<<constructor>> Statistics(winnerID: Int, positionFinished: Int, handsWon: Int, handsPlayed: Int, foldsPreFlop: Int, aggressiveMoved: Int, passiveMoved: Int,
ShowdownStats
- sidePots: List<List<Player>>
- sidePotSizes: List<Long>
- communityCards: List<Card>
- allPlayers: List<Player>
<<constructor>> ShowdownStats(players: List<Player>, communityCards: List<Card>)
+ addSidePot(players: List<Player>, size: Long)
+ getHoleCards(): Map<Integer, Card[]>
+ getWinnerText(): String
Pot
- potSize: Long
- amountPlayerCanClaim: Map<Integer, Long>
<<constructor>> Pot()
+ addToPot(userID: Int, amountToAdd: Long)
+ handOutPotPlayersInHand: List<Player>, communityCards: List<Card>, showdown: ShowdownStats)
- handOutPotShare(players: List<Player>, communityCards: List<Card>, showdown: ShowdownStats)
+ getSharePlayerCanWin(playerID: Int): Long
- getPotWinners(players: List<Player>, communityCards: List<Card>): ArrayList<Player>
StraightFlush
- returnHand: List<Card>
- compareValues: List<Integer>
+ match(Hand): Bool
+ toString(): String
- findMainSuit(List<Card>): Card.Suit
xOfaKind
- returnHand: List<Card>
+ match(Hand): Bool
+ toString(): String
- addHighCards(int, List<Card>, List<Card>)
House
- returnHand: List<Card>
- compareValues: List<Integer>
+ match(Hand): Bool
+ toString(): String
Flush
- returnHand: List<Card>
+ match(Hand): Bool
+ toString(): String
Straight
- returnHand: List<Card>
- cards: List<Card>
- lookingForSF: Bool
+ match(Hand): Bool
+ toString(): String
TwoPairs
- returnHand: List<Card>
- compareValues: List<Integer>
- onePair: Bool
+ match(Hand): Bool
+ toString(): String
HighCard
- returnHand: List<Card>
+ match(Hand): Bool
+ toString(): String
<<interface>>
IRule
+ match(hand: Hand): Bool
+ toString(): String
<<enumeration>>
HandType
+ STRAIGHT_FLUSH: EnumConstant
+ QUAD: EnumConstant
+ HOUSE: EnumConstant
+ FLUSH: EnumConstant
+ STRAIGHT: EnumConstant
+ TRIPS: EnumConstant
+ TWO_PAIRS: EnumConstant
+ PAIR: EnumConstant
+ HIGH_CARD: EnumConstant
HandCalculator
- bestHand: Optional<List<Card>>
- straightFlush: IRule
- quad: IRule
- trips: IRule
- pair: IRule
- house: IRule
- flush: IRule
- straight: IRule
- twoPairs: IRule
- highCard: IRule
- rules: List<IRule>
- rule: IRule
- handType: HandType
+ getRules(): List<IRule>
+ getBestHand(): Optional<List<Card>>
+ getHandType(): HandType
+ getFoundRule(): IRule
Use