Report

No generative AI tools were used in the preparation of the solution to this work.

Design

My plan currently is that to do everything up to step 4 with the added features:

1. Human player
2. Console input on how many random, passive, aggressive, and smart players can begin a game of poker hands
3. Console input on whether we want human player to play
4. Console input on what human player can be named
5. Side pots and main

This table represents the top-down design of my functions.

Function name	Input	Process	Output	Dependencies
gameLoop	None	Manages main flow of the game. It creates the game state using createGameState, and then uses the helper function gameLoop' that takes care of the core gameplay logic, which processes the game state and returns an updated state when the game concludes. The updated state is passed to endGame, responsible for handling end-of-game tasks like printing results or cleanup.	IO()	createGameState gameLoop' endGame
endGame	GameState	The endGame function is used to handle the final steps of the program after the game loop has concluded. Displays the final GameState.	IO()	TODO
gameLoop'	GameState	It will recursively call itself at the end. It will call a series of methods to simulate the gameplay of poker (e.g., shuffling the deck, dealing cards, assigning the dealer and blinds, running a betting round, and determining a winner) and will reset fields such as deck, bettingStage and gamesPlayed before recursively calling itself with this new updated game state.	IO GameState	determineWinner bettingRound assignDealerAndBlinds dealCards shuffleDeck createDeck

		However, if the number of games played so far is above 100 then it will end the recursive call and return the game state as it is. and updates the game state with new values such as the incremented game count, reset bets, pot, and community cards. The updated state is then passed back to gameLoop' for the next iteration. Once the game count reaches the limit, the final state is returned.		
determineWinner	IO GameState	It will manage the showdown stage of a game of poker, determining the winner(s) of a round after evaluating hands which is done by the evaluateHand function over players who are still in the game and have not folded during the current round, sharing the pot by calling a helper function called sharePot and updating the returned game state. It will also make sure that any player who have no chips, after the winner(s) have been declared and chips count updated then, their isOutOfGame field will be set to true.	IO GameState	evaluateHand sharePot
sharePot	GameState	It will calculate the contributions for each player and then based out figure out what players are eligible for what pot and store them in a list of tuples (so pot and then eligible players). Then will iterate over each item in this list to share the different pots to winner or tiers for each pot via the helper function sharePot'. This latter function will find the winner or tiers eligible for the specific pot and share evenly the pot. If there are any left-over chips that couldn't be shared equally they will go the player that's closest to the dealer.	IO GameState	evaluateHand

bettingRound	IO GameState	This function will be an implementation of the core betting logic for each stage of a poker game except for ShowDown and will transitions the game state from stage to stage. If it the betting stage is Pre-Flop, then it will enforce the small and big blinds by using the smallBlinderIndex and bigBlinderIndex from gameState to identify the players who are small and big blinders, and it will use the helper functions bet and raise to actually update the gameState to reflect such events. Then it will call helper function playersTakeAction sequentially over a list of players in order of their turn, which function will do all of the stuff to update the game state to show the effects of player taking actions. At the end of the preflop it will then deal community cards via the dealCards function and then return this newly updated game state with the bettingStage set to Flop. If the betting stage is Flop, Turn, or River. It will call helper function playersTakeAction sequentially over a list of players in order of their turn. After the updated game state is returned it will now set the betting stage to the next one (i.e. if its Turn now, then River) and deal 1 card to the community via dealCards, however if the current betting stage is River, then no cards will be dealt, and the betting Stage will be set to Nothing. If the betting stage is Nothing, then it will just return the gameState as it is. If at any point of the start of each condition, there is only one player left in the round then any other functions won't be executed and	IO Game State	playersTakeAction bet raise dealCards

		the gameState will be returned intact.		
assignDealerAndBlinds	GameState	This function will handle the assignment of the dealer button and blinds in a poker game. If a dealer is already in the list of players, the dealer button is moved to the player to the left of the current dealer. Otherwise, if no dealer exists yet, the last player in the list is assigned as the dealer by default. Once the dealer is determined, the small blind is assigned to the player immediately to the left of the dealer, and the big blind is assigned to the player to the left of the small blind. Note: In a two-player game, the dealer also posts the small blind, with the other player posting the big blind.	Game State	None
evaluateHand	[Card]	It will determine the hand ranking of a given list of cards by evaluating it against all possible hand rankings and selecting the greatest ranking hand ranking, which needs a custom implementation for Ord to work as intended.	Hand Ranking	greatestRoyalFlush greatestStraightFlush greatestFourOfAKind greatestFullHouse greatestFlush greatestStraight greatestThreeOfAKind greatestTwoPairs greatestOnePair greatestHighCard
playersTakeAction	GameState Int [Player]	This function will make sure that all players have taken an action and do need to take any more actions. If all players have check during the current bettingStage then we end the recursive call. If all players have matched the current bet or folded or they're out of the game completely then we end the	IO Game State	playerRandomStrategy playerPassiveStrategy playerAggressiveStrategy playerSmartStrategy

		recursive call. If all players except for one have folded or out of the game, then we end the recursive call and return an updated gameState with the bettingStage set to Nothing (signifying the program to move on to evaluating hands and determining the winner). If the player is out of the game completely or have folded, then recursively call playersTakeAction for the next player that needs to take an action. Otherwise, we then compute the possible valid actions a player can take based on the gameState. A player can bet or check if no bets have been made in the current bettingStage. A player can raise if the current bet but doubled is less than the player's chips. A player can fold only if a bet has been made. A player can go all in anytime in the game. We then collect all actions into a list of options and submit these options to the player's strategy. Then use the updated gameState taken from the player's strategy being applied to recursively call playersTakeAction, but make sure that players that this function goes through will be updated in such way that if someone raises or bets you add all the players that need to match the new bet.		playerHumanPlay
playerSmartStrategy	GameState Player [Action]	This function is one that based on the current betting stage and current bet and number of players and the handRanking of his/her cards it will then carry out appropriate actions. I will use lots of complex guards statements in this function.	IO Game State	check fold bet raise call allIn
playerHumanPlay	GameState Player	I would first print out the options and numbered and wait for the user to input a number from the	IO Game State	check fold bet

	[Action]	options i.e. 1 for call, 2 for fold, and so on. Then based on the number chosen that action gets done. Obviously, if a human player were to play, some of the info that gets printed out to the console such as the full gamestate attributes showing the hands of a player won't be printed out		raise call allIn
playerAggressiveStrategy	GameState Player [Action]	This function is a similar implementation of playerRandomStrategy the only difference being that the valid actions are revised so that their option to call is much higher that doing any other options by replicating the check action in the list of actions. Also, the option to raise or bet is also higher than the option to check or fold	IO Game State	check fold bet raise call allIn
playerPassiveStrategy	GameState Player [Action]	This function is a similar implementation of playerRandomStrategy the only difference being the valid actions are revised so that there is no option to raise, bet or go all in, and that there is a greater chance that the player checks over calling and folding by replicating the check action in the list	IO Game State	check fold call
playerRandomStrategy	GameState Player [Action]	This function is an implementation of what a player that always choses randomly would do. An amount of chips, in case the player randomly choses to raise or bet, will be generated to be a random chosen value between the twice the current bet and the players chips, provided the players chips is greater than twice the current bet. From the list of valid possible actions, it will randomly pick one via a randomly generated index. This chosen action will then be executed via one of the helper functions (i.e. Fold for fold, Bet for	IO Game State	check fold bet raise call allIn

		bet, Call for call, etc.). The updated game state is then returned.		
call	GameState	It will first check if this player has enough chips to call. If it doesn't then it will then pass the yet untouched game state to the function allIn. Otherwise, it will update the player's chips, bets placed and actions to reflect the changes that will occur when a player in real life calls. It will then the players list in the game state to remove the outdated info of the current player and replace with the new info. It will also check how much chips that the player needs put in extra to match the bet and ensure that the player put not a chip more than that. It will return the updated game state, so these changes are reflected.	IO Game State	None
raise	Player	It will first check if this player has enough chips to raise the bet. If it doesn't then it will then pass the yet untouched game state to the function allIn. Otherwise, it will update the player's chips, bets placed and actions to reflect the changes that will occur when a player in real life raises. It will then update the players list in the game state to remove the outdated info of the current player and replace with the new info. It will also check how much chips that the player needs put in extra to match the bet plus any chips due to raising the bet and ensure that the player not put a chip more than that. It will return the game state, so these changes are reflected.	IO Game State	None
bet	GameState	It will first check if this player has enough chips to bet. If it doesn't then it will then pass the yet untouched game state to the function allIn. Otherwise, it will update the player's chips, bets	IO Game State	None

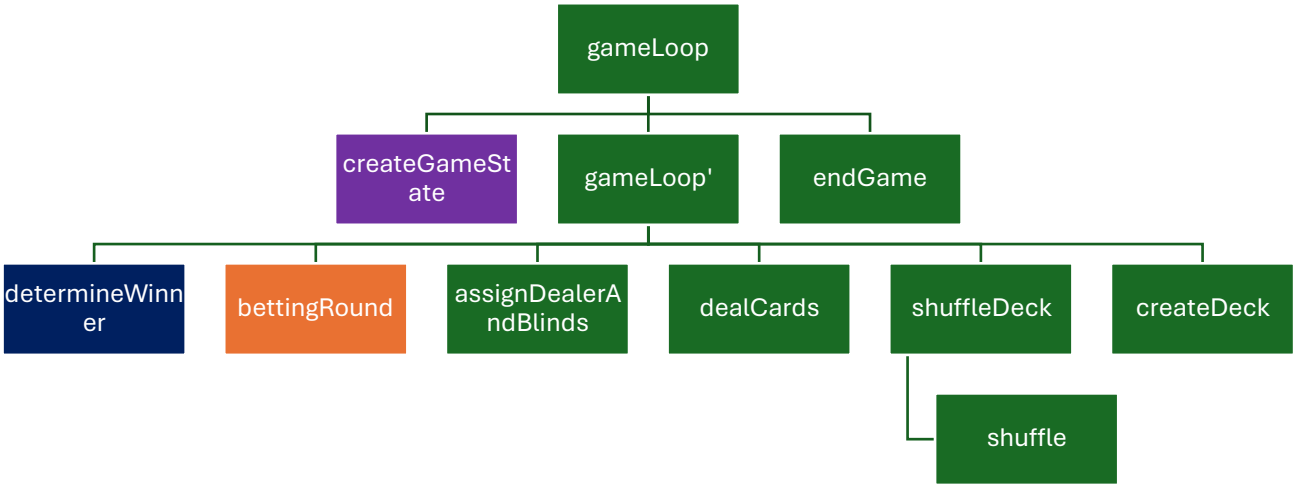
		placed and actions to reflect the changes that will occur when a player in real life bets. It will then update the players list in the game state to remove the outdated info of the current player and replace with the new info. It will return the game state, so these changes are reflected.		
allIn	Player	It will update the player's chips, bets placed and actions to reflect the changes that will occur when a player in real life goes all in. It will then update the players list in the game state to remove the outdated info of the current player and replace with the new info. It will return the game state, so these changes are reflected.	IO Game State	None
fold	GameState	It will update the player's hands and actions to reflect the changes that will occur when a player in real life folds. It will then update the players list in the game state to remove the outdated info of the current player and replace it with the new info. It will return the game state, so these changes are reflected.	IO Game State	None
check	GameState Player	It updates the actions list of the player and replace the old info of the player in the gameState with the new info	IO Game State	None
dealCards	Either (Int, GameState) GameState	This function will handle two cases for distributing cards: 1. Dealing cards to the community. Dealing cards to players (their private hands).	Game State	None
shuffleDeck	IO GameState	The function will retrieve the existing deck from the game state, shuffle it, and return the updated game state with the new shuffled deck. It uses a helper function called shuffle	IO Game State	shuffle

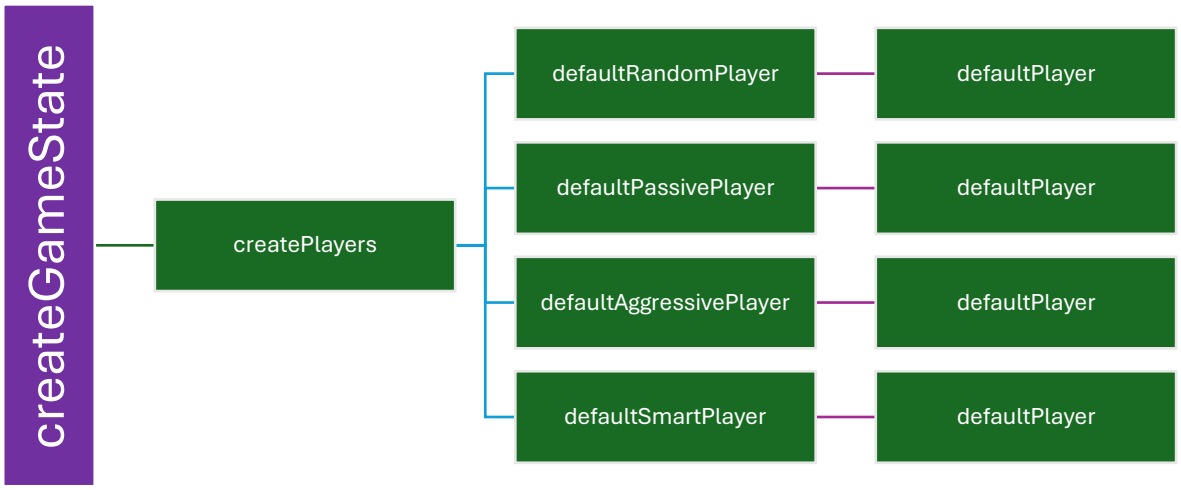
createDeck	IO GameState	Will update the game state with a newly generated deck of cards. The deck is created using all possible combinations of 'Rank' and 'Suit'.	IO Game State	
createGameState	None	Creates a new game state with an empty deck and no dealer, small blinder, or big blinder indexes with an attribute that stores the number of games played	Game State	createPlayers
createPlayers	Just String, (Int, Int, Int, Int)	It will create a list of players with different behaviours and names based on the integers in the tuple with the option to add a custom human player name	[Player]	defaultRandomPlayer defaultPassivePlayer defaultAggressivePlayer defaultSmartPlayer defaultHumanPlayer

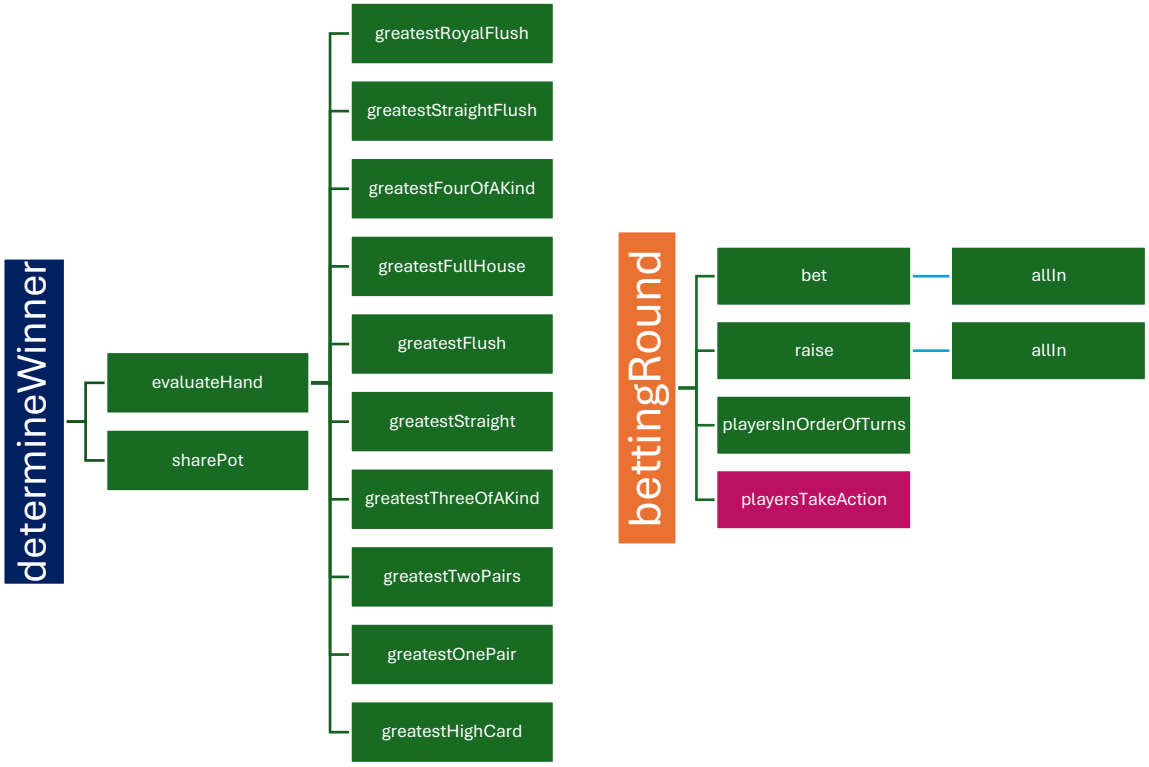
This table does not contain any data, type, new type constructors.

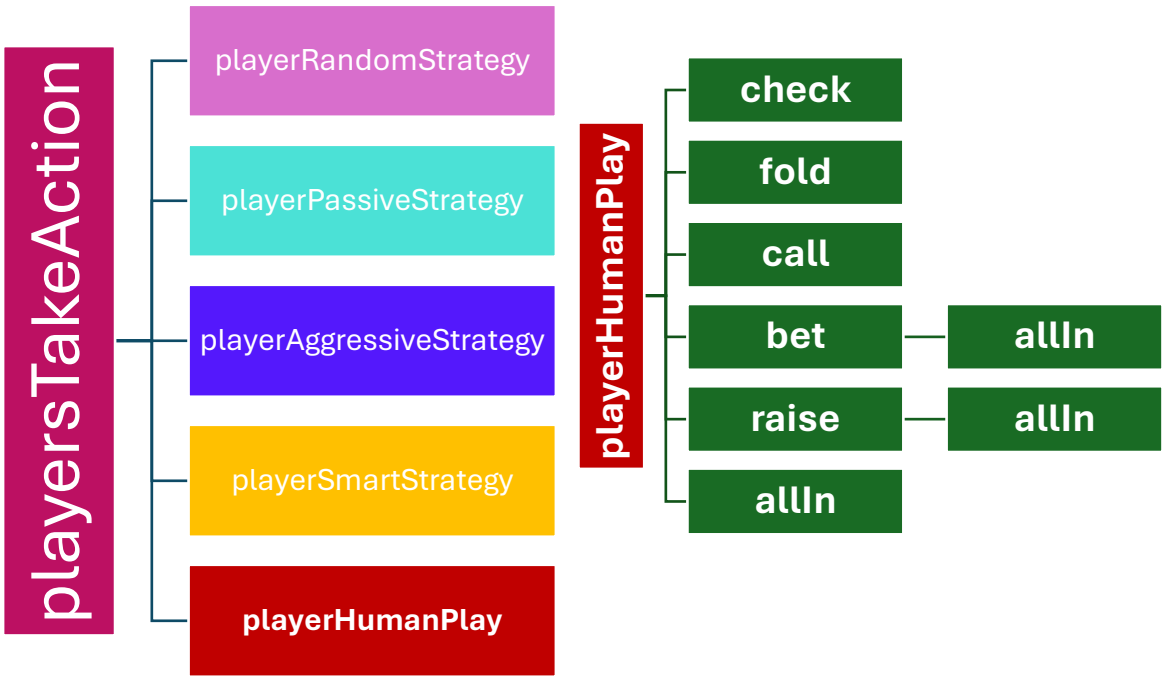
Note: Also, there is structure to support other poker variants due to the way I compare different handRankings, which is something that can be implemented later in the development process.

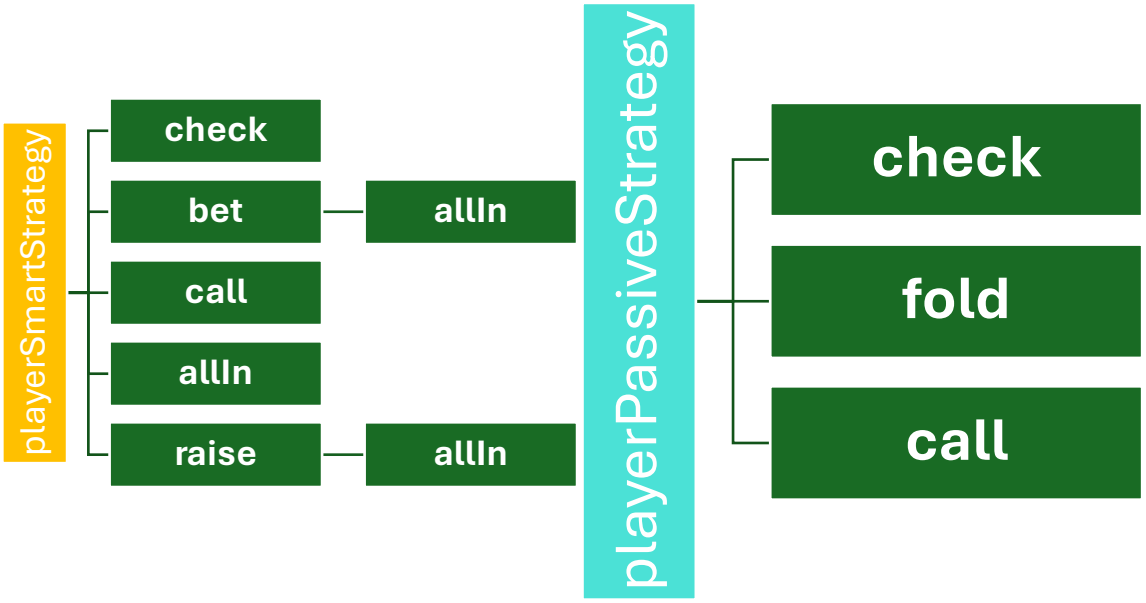
Diagrams

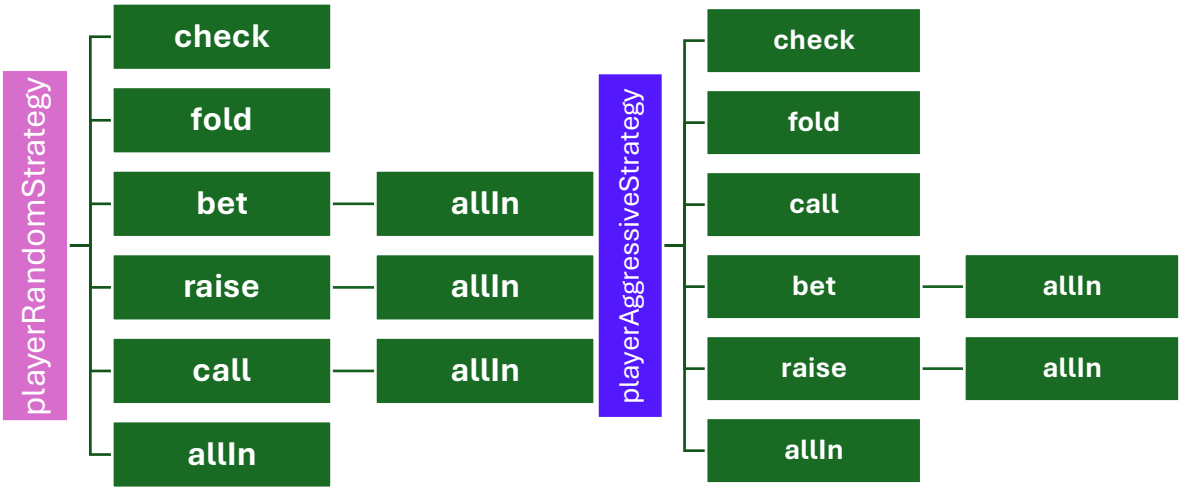

gameLoop
createGameSt ate
gameLoop'
endGame
determineWinn er
bettingRound
assignDealerA ndBlinds
dealCards
shuffleDeck
createDeck
shuffle


createGameState
createPlayers
defaultRandomPlayer
defaultPassivePlayer
defaultAggressivePlayer
defaultSmartPlayer
defaultPlayer
defaultPlayer
defaultPlayer
defaultPlayer


determineWinner
evaluateHand
sharePot
greatestRoyalFlush
greatestStraightFlush
greatestFourOfAKind
greatestFullHouse
greatestFlush
greatestStraight
greatestThreeOfAKind
greatestTwoPairs
greatestOnePair
greatestHighCard
bettingRound
bet
raise
playersInOrderOfTurns
playersTakeAction
allIn
allIn


playersTakeAction
playerRandomStrategy
playerPassiveStrategy
playerAggressiveStrategy
playerSmartStrategy
playerHumanPlay
playerHumanPlay
check
fold
call
bet
raise
allIn
allIn
allIn





These diagrams don't include any data, type or new type constructors.

Testing

Simulation of a large number of games of 4 players of different behaviours (Random, Passive, Aggressive, Smart)

When it came to the simulation of this case, I decided to run 1000 times the gameLoop function in another method called main, and then run that method 10 times and report back the number of times each type of player won over the 1000 games. I have found these remarks:

- Run 1:
```
[("Random",516),("Smart",94),("Aggressive",293),("Passive",97)]
```
- Run 2:
```
[("Random",524),("Smart",90),("Aggressive",298),("Passive",88)]
```
- Run 3:
```
[("Random",511),("Smart",86),("Aggressive",295),("Passive",108)]
```
- Run 4:
```
[("Random",502),("Smart",98),("Aggressive",315),("Passive",85)]
```
- Run 5:
```
[("Random",502),("Smart",98),("Aggressive",315),("Passive",85)]
```
- Run 6:
```
[("Random",511),("Smart",96),("Aggressive",293),("Passive",100)]
```
- Run 7:
```
[("Random",505),("Smart",98),("Aggressive",294),("Passive",103)]
```
- Run 8:
```
[("Random",496),("Smart",95),("Aggressive",302),("Passive",107)]
```
- Run 9:
```
[("Random",525),("Smart",86),("Aggressive",293),("Passive",96)]
```
- Run 10:
```
[("Random",499),("Smart",96),("Aggressive",305),("Passive",100)]
```

These results tell me that my implementation for a smart player is a poor one, due to the, in average, having the lowest win rate (less than 10%), which was very unexpected. Also, I expected the aggressive player to win a lot more, so I am very surprised about it winning about 30% of the time. Perhaps, I need to do more runs to see if my results so far are anomalies, but due to time constraints I will leave it as it is, for someone else or

me to deal in the future. However, after further reflection, I have noticed that I haven't really used the values of the cards to determine what the smart player should do neither did I really fully used the past actions of players that are still in the game.

Test cases

Note: The test cases used to compare hand rankings are chosen in a way to choose the ones where any bug might be, however, I have tested all of them , just decided to report distinct examples.

Test name	Inputs	Outputs	Outcomes	Reflections
Suit can be printed	Hearts Diamonds Clubs Spades	H D C S	Passed Passed Passed Passed	None
Suit Eq works	Hearts == Hearts	True	Passed	No need to check for other suits
Rank can be printed	Two Three Four Five Six Seven Eight Nine Ten Jack Queen King Ace	2 3 4 5 6 7 8 9 10 J Q K A	Passed Passed Passed Passed Passed Passed Passed Passed Passed Passed Passed Passed Passed	None

Rank Ord and Eq works	Two == Two Ace > Two Ace < King Five < Jack	True True False True	Passed Passed Passed Passed	No need to check for every possible combination. Just checking with options that may have bugs, but don't in this case
Card can be printed	Card Three Spades Card Jack Diamonds	3S JD	Passed Passed	None
Card comparison works	Card Three Diamonds == Card Three Spades Card Jack Clubs > Card Nine Diamonds Card Jack Hearts > Card Eight Hearts	False True True	Passed Passed Passed	None
Player	---	----	---	Assume it works, given identical logic being used from previous code blocks such as data Card
Behaviour	---	----	---	Assume it works, given identical logic being used from previous code blocks such as data Card
Action	---	----	---	Assume it works, given identical logic being used from previous code blocks such as data Card
BettingStage	---	----	---	Assume it works, given identical logic being used from previous code blocks such as data Card
GameState	---	----	---	Assume it works, given identical

				logic being used from previous code blocks such as data Card
defaultPlayer works and returns a player	defaultPlayer "test" Aggressive	Player {name = "test", hand = [], chips = 1000, betsPlaced = [0,0,0,0], isDealer = False, behaviour = Aggressive, actions = [], isOutOfTheGame = False}	Passed	None
defaultRandomPlayer works	defaultRandomPlayer "test"	Player {name = "test", hand = [], chips = 1000, betsPlaced = [0,0,0,0], isDealer = False, behaviour = Random, actions = [], isOutOfTheGame = False}	Passed	None
defaultPassivePlayer	---	----	---	Assume it works, given identical logic being used from previous code blocks such as defaultRandomPlayer
defaultAggressivePlayer	---	----	---	Assume it works, given identical logic being used from previous code blocks such as defaultRandomPlayer
defaultSmartPlayer	---	----	---	Assume it works, given identical logic being used from previous code blocks such as defaultRandomPlayer

defaultHumanPlayer	---	----	---	Assume it works, given identical logic being used from previous code blocks such as defaultRandomPlayer
createPlayers	createPlayers (Just "test") (2, 1, 1, 1) createPlayers Nothing (2, 1, 1, 1)	.... 	Passed Passed	As expected, though, I had to adjust so that if the tuple contains negative numbers, it returns an empty list of players for what that negative number represents. Also had to make sure that if the sum of all types of players are not enough then it will default to return a game stage with 2 random players and that if the sum of the integers in the tuple contained is more than 10 it will just return the first 10 players so that it doesn't leads to problems when dealing cards
Deck is created	createDeck $ pure createGameState	....	Passed	None
Deck is shuffled	shuffleDeck $ createDeck $ pure createGameState	...	Passed	I forgot to update the newly shuffled deck at one point, but now it works.
Shuffled deck has no missing elements	sort (shuffleDeck $ createDeck $ pure createGameState) == (createDeck $	True	Passed	None

	pure createGameState)			
dealCards works when dealing to the community	dealCards but Left option and numberOfCards set to 3	A gameState with 3 community cards	Passed	I had to go back to the lectures as I forgot the syntax
dealCards works when dealing to players	dealCards Right option	A gameState where all players have 2 cards	Passed	I was struggling with how to deal cards to players but I mistakenly found a way by playing around with all the list operations on terminal
check works	check gameState player, where player has to be a player in gameState	A gameState where the player last actions is a check depending on what betting stage we currently in	Passed	It was quite easy I just had to restructure the way I thought about checking and checking whether a player has checked. So instead of having just check I changed it to CheckPreFlop, CheckFlop, CheckTurn, CheckRiver.
fold works	Check gameState player, where player has to be a player in gameState	A gameState where the player last action is a fold depending on what betting stage, we currently in	Passed	It was the easiest one to do and the most straight forward
allIn works	allIn gameState player, where player has to be a player in gameState	A gameState where the player last action is a allIn depending on what betting stage we currently in.	Passed	The hardest so far had to also go back to my player and gameState constructors and change and add fields to them so to make the function

		And the betsPlaced have been updated if needed And that the bets in game state have been updated And that the player's chips have been updated		work and reduce the complexity of it
bet works	bet gameState player, where player has to be a player in gameState	A gameState where the player last action is a bet depending on what betting stage we currently in. And the betsPlaced have been updated And that the bets in game state have been updated And that the player's chips have been updated	Passed	Even though the shouldICallAllIn guard is not needed given what I do after in the other methods, I still kept it there because I couldn't be bored.
raise works	raise gameState player, where player has to be a player in gameState	A gameState where the player last actions is a raise depending on what betting stage we currently in And the betsPlaced have been updated And that the bets in game state have been updated And that the player's chips have been updated	Passed	I had a bug that allowed the player to raise negative amount of chips but I have got rid off it but things to do to improve this function is to ensure that the argument entered is not negative to begin with
call works	call gameState player, where player has to be a player in gameState	A gameState where the player last actions is a call depending on what betting stage we currently in And the betsPlaced have been updated	Passed	Things to do to improve this function is to ensure that the player passed in is an actual player of the gameState before performing any operations

		And that the bets in game state have been updated And that the player's chips have been updated		
playerRandomStrategy works	A gameState with a deck and some players, with at least one of them to have random behaviour and that the player passed in is random, and a list of actions that can be carried out. Run this a few times so it runs through every action	Everything should work given that we tested the allIn, raise, bet, etc. functions the only to check is that after it is ran multiple times the number of actions carried out should be stabilising	Passed	Things to do to improve this function is to ensure that the player passed in is an actual player of the gameState before performing any operations And to ensure the player behaviour is random indeed else throw an error
playerPassiveStrategy works	A gameState with a deck and some players, with at least one of them to have passive behaviour and that the player passed in is passive, and a list of actions that can be carried out. Run this a few times so it runs through every action	Everything should work given that we tested the allIn, raise, bet, etc. functions the only to check is that after it is ran multiple times the number of actions carried out should be stabilising and that this player never raises or bets no matter what	Passed	Things to do to improve this function is to ensure that the player passed in is an actual player of the gameState before performing any operations And to ensure the player behaviour is random indeed else throw an error
playerAggressiveStrategy works	A gameState with a deck and some players, with at least one of them to have aggressive behaviour and that the player passed in is aggressive, and a list of actions that can be carried out. Run this a few times so it runs	Everything should work given that we tested the allIn, raise, bet, etc. functions the only to check is that after it is ran multiple times the number of actions carried out should show that the aggressive player	Passed	Things to do to improve this function is to ensure that the player passed in is an actual player of the gameState before performing any operations And to ensure the player behaviour is random indeed else throw an error

	through every action	rarely folds and checks		
playerHumanPlay works	----	----	----	I have not implemented this function yet, due to time constraint, something one can do easily based on the foundation I laid so far
playerSmartStrategy	The best way to run it is once human play is implemented to run it against 2 smart players and see how easy it is to win	The smart player should have fairly good winning rate	----	Due to not having implemented human play yet I can really test how good is smart player but from the looks of it when against random players and aggressive players it performs terribly
playersTakeAction: when all players fold	Pass in a gameState where all players last actions is fold	Game state should remain unchanged	Passed	None
playersTakeAction: when a players is marked as out of game it should be skipped	Pass in a game state where the player about to act is marked out of game	This player should be skipped	Passed	Rather than create a list of active players I preferred using a field that determine if a player is out of game or not
playersTakeAction: when all players check	Pass in a gameState where all players checked	The gameState should be returned unchanged and		
playersTakeAction: when a player matches a bet	---	----	Passed	Assume obvious as the component that deal with the update of such action works and has been tested

playersTakeAction: when a players goes all in	---	---	Passed	As the components that deal with that process have been tested and do work
playersTakeAction: when a player raises all other players must match the new bet	A game state with a human player that raises	The playersInOrderOfTurns is updated to include all of the players from the next player to act to the player before this player that raised at the end of the list.	Passed	It took me a long time to figure how to do this one, and it is very complex so I am sure there is a way to simplify things further. Also, the duplicates of players can be removed and also there is no need to pass in the index of the playersInOrderOfTurns as I can just keep taking and passing in the tail of such list, but then again having the list gives you clear perfect view on the logic flow of the program .
playersTakeAction: when a player raises it cant raise again unless another player raised	A game state with a human player that raises	The playersInOrderOfTurns is updated to include all of the players from the next player to act to the player before this player that raised at the end of the list.	Passed	It took me a long time to figure how to do this one, and it is very complex, so I am sure there is a way to simplify things further. Also, the duplicates of players can be removed and also there is no need to pass in the index of the playersInOrderOfTurns as I can just keep taking and passing in the tail of such list, but then again having

				the list gives you clear perfect view on the logic flow of the program. Also the way the playersInOrderOfTurns is updated ensures that the player that raised is not in it so no chance of it raising again
playersTakeAction: when all players call or match the bet	A gameState where all players last action is call and have matched the bet	The gameState should return unchanged	Passed	It is in one of the first few guards, and was straight forward to implement which is good.
Instance HandRanking compare bit: One Pair	compare (HighCard Ace [King, Queen, Jack, Ten]) $ HighCard Ace [Queen, Jack, Ten, Nine]	GT	Passed	None
Instance HandRanking compare bit: One Pair	compare (OnePair Three [Ace, King, Queen]) (OnePair Three [Ace, King, Jack])	GT	Passed	None
Instance HandRanking compare bit: Two Pairs	compare (TwoPair (King, Queen) [Jack]) (TwoPair (King, Queen) [Ten])	GT	Passed	None
Instance HandRanking compare bit: Three of a kind	compare (ThreeOfAKind Five [Ace, King]) (ThreeOfAKind Five [Ace, Queen])	GT	Passed	None
Instance HandRanking compare	compare (Straight Ace [King, Queen, Jack, Ten]) (Straight	LT	Passed	None

bit: Straight	King [Queen, Jack, Ten, Nine])			
Instance HandRanking compare bit: Flush	compare (Flush Spades (Ace, King, Queen, Jack, Ten) []) (Flush Spades (Ace, King, Queen, Jack, Nine) [])	GT	Passed	None
Instance HandRanking compare bit: Full House	compare (FullHouse (Three, Two) []) (FullHouse (Three, Ace) [])	LT	Passed	None
Instance HandRanking compare bit: StraightFlush	compare (StraightFlush Ace []) (StraightFlush King [])	LT	Passed	None
Instance HandRanking compare bit: Royal Flush	compare (RoyalFlush [Ace, King, Queen, Jack, Ten]) (RoyalFlush [Ace, King, Queen, Jack, Ten])	EQ	Passed	None
Instance HandRanking compare bit: One Pair	compare (FullHouse (Three, Two) []) (Flush Hearts (Ace, King, Queen, Jack, Ten) [])	GT	Passed	It didn't work at first but that's because I forgot to add the guard to compare when they are different hand ranking are compared.
Instance HandRanking compare bit: One Pair	compare (HighCard Ace [King, Queen, Jack, Ten]) (HighCard Ace [King, Queen, Jack, Ten])	EQ	Passed	None
greatestHighCard	greatestHighCard [Card Ace Spades, Card King Hearts, Card Queen Diamonds, Card Jack Clubs]	Just (HighCard Ace [King, Queen, Jack])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex,

				however I believe there is still room for improvement
greatestHighCard	greatestHighCard [Card Ace Spades, Card Ace Hearts, Card King Diamonds, Card Queen Clubs]	Just (HighCard Ace [King, Queen, Ace])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestHighCard	greatestHighCard []	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestOnePair	greatestOnePair [Card Ace Spades, Card Ace Hearts, Card King Diamonds, Card Queen Clubs]	Just (OnePair Ace [King, Queen])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestOnePair	greatestOnePair [Card Ace Spades, Card King Hearts, Card Queen Diamonds, Card Jack Clubs]	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestOnePair	greatestOnePair [Card Ace Spades, Card Ace Hearts, Card King Diamonds, Card King Clubs]	Just (OnePair Ace [King, King])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe

				there is still room for improvement
greatestTwoPairs	greatestTwoPairs [Card Ace Spades, Card Ace Hearts, Card King Diamonds, Card King Clubs, Card Queen Diamonds]	Just (TwoPair (Ace, King) [Queen])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestTwoPairs	greatestTwoPairs [Card Ace Spades, Card Ace Hearts, Card King Diamonds, Card King Clubs, Card Queen Diamonds, Card Queen Hearts]	Just (TwoPair (Ace, King) [Queen, Queen])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestTwoPairs	greatestTwoPairs [Card Ace Spades, Card King Hearts, Card Queen Diamonds, Card Jack Clubs]	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestTwoPairs	greatestTwoPairs [Card Ace Spades, Card Ace Hearts, Card King Diamonds, Card Queen Clubs]	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestThreeOfAKind	greatestThreeOfAKind [Card Ace Spades, Card Ace Hearts, Card Ace Diamonds, Card King Clubs, Card Queen Diamonds]	Just (ThreeOfAKind Ace [King, Queen])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement

greatestThreeOfAKind	greatestThreeOfAKind [Card Ace Spades, Card King Hearts, Card Queen Diamonds, Card Jack Clubs, Card Ten Spades]	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestThreeOfAKind	greatestThreeOfAKind [Card Ace Spades, Card Ace Hearts, Card Ace Diamonds, Card King Spades, Card King Clubs, Card King Diamonds]	Just (ThreeOfAKind Ace [King, King, King])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestThreeOfAKind	greatestThreeOfAKind [Card Ace Spades, Card Ace Hearts, Card Ace Diamonds, Card Two Clubs, Card Three Diamonds]	Just (ThreeOfAKind Ace [Three, Two])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestStraight	greatestStraight [Card Ace Spades, Card Two Hearts, Card Three Diamonds, Card Four Clubs, Card Five Spades, Card Six Diamonds]	Just (Straight Ace [Six])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestStraight	greatestStraight [Card Ten Spades, Card Jack Hearts, Card Queen Diamonds, Card King Clubs, Card Ace Spades]	Just (Straight Ten [])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestStraight	greatestStraight [Card Ace Spades,	Nothing	Passed	I had to redo this function 4 different

	Card King Hearts, Card Queen Diamonds, Card Jack Clubs, Card Nine Spades]			times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestStraight	greatestStraight [Card Ace Spades, Card Two Hearts, Card Three Diamonds, Card Four Clubs, Card Five Spades, Card Six Diamonds, Card Seven Spades]	Just (Straight Three [Ace, Seven])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestStraight	greatestStraight [Card Ace Spades, Card Two Hearts, Card Three Diamonds, Card Four Clubs, Card Five Spades]	Just (Straight Ace [])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestFlush	greatestFlush [Card Ace Hearts, Card King Hearts, Card Queen Hearts, Card Jack Hearts, Card Ten Hearts, Card Two Diamonds]	Just (Flush Hearts (Ace, King, Queen, Jack, Ten) [Two])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestFlush	greatestFlush [Card Ace Hearts, Card King Diamonds, Card Queen Spades, Card Jack Clubs, Card Ten Hearts]	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestFlush	greatestFlush [Card Ace Hearts, Card King Hearts, Card Queen Hearts,	Just (Flush Hearts (Ace, King, Queen, Jack, Ten) [Nine, Eight])	Passed	I had to redo this function 4 different times from scratch due to the first few

	Card Jack Hearts, Card Ten Hearts, Card Nine Hearts, Card Eight Spades]			instances being too complex, however I believe there is still room for improvement
greatestFullHouse	greatestFullHouse [Card Ace Hearts, Card Ace Diamonds, Card Ace Spades, Card King Clubs, Card King Hearts]	Just (FullHouse (Ace, King) [])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestFullHouse	greatestFullHouse [Card Ace Hearts, Card Ace Diamonds, Card King Spades, Card Queen Clubs, Card Jack Hearts]	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestFullHouse	greatestFullHouse [Card Ace Hearts, Card Ace Diamonds, Card Ace Spades, Card King Clubs, Card King Hearts, Card Queen Diamonds, Card Queen Hearts]	Just (FullHouse (Ace, King) [Queen, Queen])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestFullHouse	greatestFullHouse [Card Ace Hearts, Card Ace Diamonds, Card Ace Spades, Card King Hearts, Card King Diamonds]	Just (FullHouse (Ace, King) [])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestFourOfAKind	greatestFourOfAKind [Card Ace Hearts, Card Ace Diamonds, Card Ace Spades, Card	Just (FourOfAKind Ace [King])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being

	Ace Clubs, Card King Hearts]			too complex, however I believe there is still room for improvement
greatestFourOfAKind	greatestFourOfAKind [Card Ace Hearts, Card Ace Diamonds, Card King Spades, Card Queen Clubs, Card Jack Hearts]	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestFourOfAKind	greatestFourOfAKind [Card Ace Hearts, Card Ace Diamonds, Card Ace Spades, Card Ace Clubs, Card King Hearts, Card King Diamonds, Card King Spades, Card King Clubs]	Just (FourOfAKind Ace [King, King, King])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestFourOfAKind	greatestFourOfAKind [Card Ace Hearts, Card Ace Diamonds, Card Ace Spades, Card Ace Clubs, Card Two Hearts, Card Three Diamonds]	Just (FourOfAKind Ace [Three, Two])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestStraightFlush	greatestStraightFlush [Card Ten Hearts, Card Jack Hearts, Card Queen Hearts, Card King Hearts, Card Ace Hearts, Card Two Diamonds]	Just (StraightFlush Ten [Two])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestStraightFlush	greatestStraightFlush [Card Ten Hearts, Card Jack Hearts, Card Queen Diamonds,	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex,

	Card King Hearts, Card Ace Clubs]			however I believe there is still room for improvement
greatestStraightFlush	greatestStraightFlush [Card Nine Hearts, Card Ten Hearts, Card Jack Hearts, Card Queen Hearts, Card King Hearts, Card Eight Hearts]	Just (StraightFlush Nine [])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestStraightFlush	greatestStraightFlush [Card Ace Hearts, Card Two Hearts, Card Three Hearts, Card Four Hearts, Card Five Hearts, Card Six Diamonds]	Just (StraightFlush Two [Six])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestRoyalFlush	greatestRoyalFlush [Card Ten Hearts, Card Jack Hearts, Card Queen Hearts, Card King Hearts, Card Ace Hearts, Card Nine Diamonds]	Just (RoyalFlush [Nine])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestRoyalFlush	greatestRoyalFlush [Card Ten Hearts, Card Jack Hearts, Card Queen Diamonds, Card King Hearts, Card Ace Clubs]	Nothing	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
greatestRoyalFlush	greatestRoyalFlush [Card Ten Hearts, Card Jack Hearts, Card Queen Hearts, Card King Hearts, Card Ace Hearts, Card Nine Hearts]	Just (RoyalFlush [Nine])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe

				there is still room for improvement
greatestRoyalFlush	greatestRoyalFlush [Card Ten Hearts, Card Jack Hearts, Card Queen Hearts, Card King Hearts, Card Ace Hearts, Card Two Diamonds, Card Three Clubs]	Just (RoyalFlush [Two, Three])	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
evaluateHand	Using the same tests cases that I have used so far for the functions above	It gives me the same results	Passed	I had to redo this function 4 different times from scratch due to the first few instances being too complex, however I believe there is still room for improvement
assignDealerAndBlinds	Use a gameState where the dealer is an index valid in the players list	It correctly transposes the dealerIndex from right to left by 1, and correctly updates the isDealer field	Passed	I had to re do it a few times as I decide to leave players that are out of the game still in the list, so it had some change in logic, but works as normally as if the players that are out of game weren't there.
assignDealerAndBlinds	Use a gameState where the index is Nothing	It correctly assigns the dealer to the last player in the list	Passed	I had to re do it a few times as I decide to leave players that are out of the game still in the list, so it had some change in logic, but works as normally as if the players that are out of game weren't there.
bettingRound	Use a gameState where the	Should return the gameState unchanged	Passed	None

	bettingStage is Nothing			
bettingRound	Use a gameState where the bettingStage is PreFlop	Should return the gameState with bettingStage is Nothing and players chips, pot, bets placed, and bets and players list actions, etc. changed	Passed	None
bettingRound	Use a gameState where the bettingStage is Flop	Should return the gameState with bettingStage is Nothing and players chips, pot, bets placed, and bets and players list actions, etc. changed. However the shouldn't be a chance of it printing to the console it is preflop	Passed	None
bettingRound	Use a gameState where the bettingStage is turn	Should return the gameState with bettingStage is Nothing and players chips, pot, bets placed, and bets and players list actions, etc. changed. However, there shouldn't be a chance of it printing to the console it is turn	Passed	None
bettingRound	Use a gameState where the bettingStage is river	Should return the gameState with bettingStage is Nothing and players chips, pot, bets placed, and bets and players list actions, etc. changed. However, there shouldn't be	Passed	None

		a chance of it printing to the console it is river		
bettingRound	Use a gameState where the bettingStage is Nothing	Should return the gameState unchanged	Passed	None
sharePot	Use a game state that reflects a single main pot with a clear winner	The winner should win all of the pot	Passed	None
sharePot	Use a game state that reflects a main pot and side pots with multiple winners	The winner of each pot is determined, and the pot should go to them. If there are any tiers the pot will be shared between them	Passed	This was a nice challenging problem that I have managed to solve quite efficiently I believe for my first try, I will try to include comments to aid in the understanding of the method
sharePot	A game state that represents a main pot with a number of multiple winners, where the number of winners is not a factor of the amount of chips in the pot	The pot should be split evenly across winners and any left-over chips goes to the player closest to the dealer going from left to right	Passed	This was also a nice challenging problem that I have managed to solve quite efficiently I believe for my first try; I will try to include comments to aid in the understanding of the method.
sharePot	A game state that represents a main pot with a number of multiple winners, where the number of winners is a factor of the amount of chips in the pot	The pot should be split evenly across winners.	Passed	None
Check that determine Winner changes the	A gameState with the betting stage set to Nothing with only one player with	A gameState where the gamesPlayed is changed to 1000	Passed	None

gamesPlayed to 1000 if only one player is still in the game	2 cards and at least 3 community cards.			
Check that players who have no chips after the pots has been shared are marked out of game after determine Winner is execute	A gameState with at least 3 community cards, where some players have no chips and 2 cards, while others do have chips and 2 cards. Ensure that the players who have no chips and 2 cards are not eligible to win any of the pots	A gameState where the players with no chips isOutOfGame field is set to True	Passed	None
gameLoop' and gameLoop exits the loop correctly after gamesPlayed field is more than 100`	Any gameState with the gamesPlayed field sets to less than 100.	A gameState where the gamesPlayed is either 100 or 1000	Passed	None

Extra Critical Reflection

I started this assignment from scratch 5 times due to not only the poor quantity and quality of code presented in the lectures but also due to the poor quality and quality of the brief explaining what needed to be done. This taught me that in future I need to make sure that I fully understood the requirements and the game before start coding.

Furthermore, this project taught me and showed me the benefit of coding bottom up which now that I have managed to get to end (albeit skipping some features) I see how easier it is to code and test and locate bugs.

Also, this project challenged me to find new and creative ways to solve different types of problems and taught me to be more conscious of code representation and repetitions.

Moreover, this project forced me to learn a functional language that I would have never delve into, given my strong background in object-oriented languages such as Java, and taught me a new way of thinking, which almost opposes to what I have learned so far during my coding journey.

This project taught me also many other lessons, but the ones I have mentioned so far are the ones that I am going to keep close to heart and try and abide by them and implement them in my future endeavours.