Strategic Player and the Game State

```
class GameState(object):
    def __init__(self,split,dd,player_hand,dealer_hand):
        self.hand_split = split
        self.hand_dd = dd
        self.player_hand = player_hand
        self.dealer_hand = dealer_hand
```

Strategic Player and the Game State

```
class GameState(object):
    def __init__(self,split,dd,player_hand,dealer_hand):
        self.hand_split = split
        self.hand_dd = dd
        self.player_hand = player_hand
        self.dealer_hand = dealer_hand
```

Creates an object that contains the information about the Player's cards and the card showed by the Dealer.

Strategic Player and the Game State

```
class StrategicPlayer(Player):

    def get_state(self, h):
        if self.table.output:
            print("Complete game state: (spl, dd, hand, dealer)",
                  self.hands[h].split,
                  self.hands[h].dd,
                  self.hands[h].sorted_list(),
                  self.table.dealer.hands[0].sorted_list())
        s = GameState(self.hands[h].split,
                      self.hands[h].dd,
                      self.hands[h].sorted_list(),
                      self.table.dealer.hands[0].sorted_list())

        return s.__hash__()
```

Strategic Player and the Game State

```
class StrategicPlayer(Player):

    def get_state(self, h):
        if self.table.output:
            print("Complete game state: (spl, dd, hand, dealer)",
                  self.hands[h].split,
                  self.hands[h].dd,
                  self.hands[h].sorted_list(),
                  self.table.dealer.hands[0].sorted_list())
        s = GameState(self.hands[h].split,
                      self.hands[h].dd,
                      self.hands[h].sorted_list(),
                      self.table.dealer.hands[0].sorted_list())

        return s.__hash__()
```

get_state(): Given an object of the class GameState, the function returns the hash number corresponding to this object.

Strategic Player and the Game State

```
class Strategy(object):
    def __init__(self):
        self.table = {}
```

Strategic Player and the Game State

```
class Strategy(object):
    def __init__(self):
        self.table = {}
```

Creates an object that contains the table with the hash number (GameState) and its corresponding Strategy.

Strategic Player and the action method

```
def action(self, h):
    gstmp = self.get_state(h)

    if gstmp in self.strategy.table:
        p = self.strategy.table[gstmp]
    else:
        if self.table.output:
            print("Hand no.", h, ": ", "Split status: ",
                  self.hands[h].split,
                  " -- double down status", self.hands[h].dd)
        if self.hands[h].split == 0:
            p = self.default[0] #split o/wise not
        elif self.hands[h].dd == 0:
            p = self.default[1] # doubledown o/wise not
        else:
            p = self.default[2] # hit o/wise stand
        self.strategy.table[gstmp] = p

    act = p > random.random()
    if act:
        self.game.prob.append(p)
    else:
        self.game.prob.append(1-p)

    self.game.game_state.append(self.get_state_non_hash(h))
    self.game.action.append(act)
    return act
```

Strategic Player and the action method

```
def action(self, h):
    gstmp = self.get_state(h)  (1)

    if gstmp in self.strategy.table:
        p = self.strategy.table[gstmp]  (2)
    else:
        if self.table.output:
            print("Hand no.", h, ": ", "Split status: ",
                  self.hands[h].split,
                  " -- double down status", self.hands[h].dd)
        if self.hands[h].split == 0:
            p = self.default[0] #split o/wise not
        elif self.hands[h].dd == 0:
            p = self.default[1] # doubledown o/wise not
        else:
            p = self.default[2] # hit o/wise stand
        self.strategy.table[gstmp] = p

    act = p > random.random()
    if act:
        self.game.prob.append(p)
    else:
        self.game.prob.append(1-p)

    self.game.game_state.append(self.get_state_non_hash(h))
    self.game.action.append(act)
    return act
```

The action() method is called every time that the player needs to decide to hit/stand, double down, or split the cards. This method firstly retrieves the hash number for the given combination of cards (Step 1). Then, it searches in the Strategy Table for the entry and its corresponding action (Step 2).

Strategic Player and the action method

```
class StrategicPlayer(Player):
    """ A player with different strategies. """
    def __init__(self, name, cre=0):
        self.credits = cre
        self.hands = []
        hand1 = Hand()
        self.hands.append(hand1)
        self.name = name
        self.strategy = Strategy()
        self.default = [1,0,0]
            # split o/wise not
            # doubledown o/wise not
            # hit o/wise stand
        self.game = Game()
        self.history = History()
```

Strategic Player and the action method

```
class StrategicPlayer(Player):
    """ A player with different strategies. """
    def __init__(self, name, cre=0):
        self.credits = cre
        self.hands = []
        hand1 = Hand()
        self.hands.append(hand1)
        self.name = name
        self.strategy = Strategy()
        self.default = [1,0,0]
            # split o/wise not
            # doubledown o/wise not
            # hit o/wise stand
        self.game = Game()
        self.history = History()
```

The Strategic Player has a table where default Values for the different actions are defined.