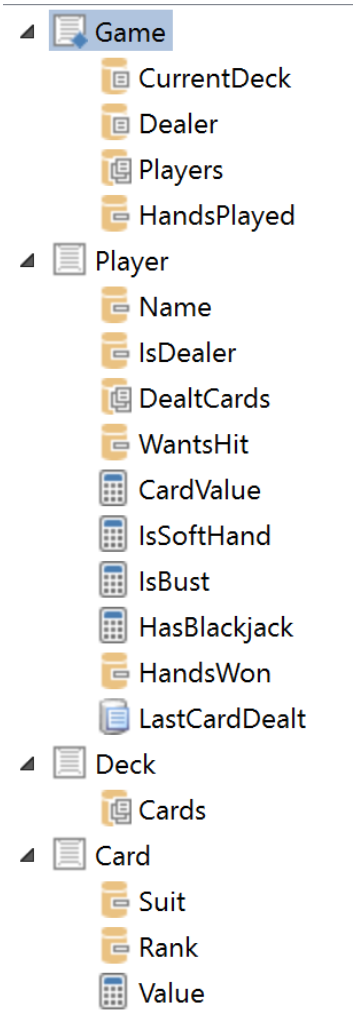
Solving Strange Challenges using Blackjack

Blackjack, Twenty-One, Pontoon, Vinget-et-Un : it goes by many names, but the concept remains consistent - get 21 points and beat the dealer. A simple game to learn and straightforward to play, Blackjack is a great game to use as a test scenario for programming gameplay logic into a Rule Application. If you're interested, give it a try - then come back here for some thoughts.

Just like when creating a business process-oriented Rules Application, while it may seem relatively straightforward at a first, there are always 'gotchas' that pop up during elaboration that require some more thought. Let's walk through some of things that made this scenario interesting.

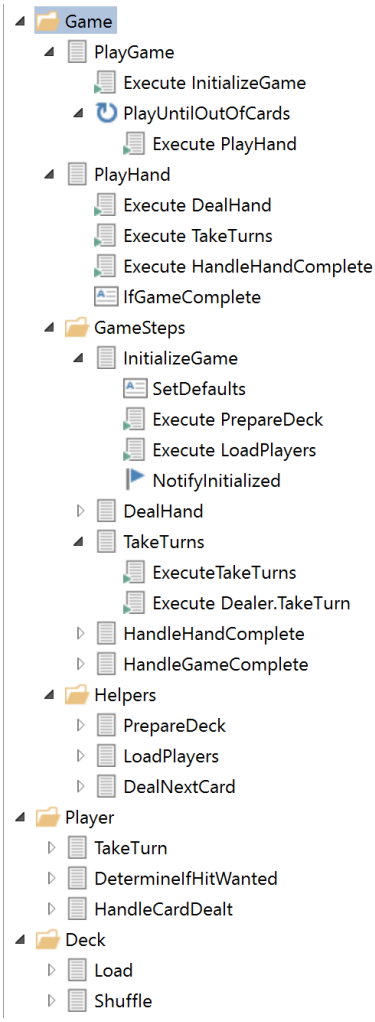


Loading the deck

Sure, you could create an inline table with 52 entries explicitly defining the deck, but where's the fun in that?! A far more compact way is to have short Value Lists for Rank and Suit, then loop through those in nested loops to programmatically build up the deck. Want to put multiple decks into your dealing shoe? No problem - add another level of looping to throw in as many as you want!

Shuffling

What good is a deck if it's perfectly in order? Shuffling is a bit of a special case in the context of a Rule App - you're not likely to want to randomize something in a production application. Fear not, though, where there's a will – there's a way! All you need to do is move your cards into a temporary collection, then loop through that collection, generating a Random number between 1 and the count of cards remaining in the copied deck each time, take the card at that index, and move it back into your main deck. By the time your temporary deck is empty, your main deck collection has been filled with a randomly ordered set of cards.



Calculating the Total Value of cards in your hand

What initially looks like simple addition takes a dark turn when you realize that not only do 4 Ranks of cards share the same point value (10/J/Q/K), but Aces can be 1 or 11 points depending on the state of the rest of the cards in your hand - Oy Vey!

For the first issue, the duplication of the point value means a Value List of key/value pairs (points/Rank) won't work. However, changing the Data element from a value list to an Inline Table allows you to populate the Value of a card separately from the Rank, which eliminates the duplicate key issue.

For the second issue, when calculating the total card value in a player's hand, you can initially treat Aces with a value of 1 - then, if the total is 11 or less and there is an Ace in the hand, simply add 10 to the total – simple!

```
Sum(DealtCards, Value) + If(((Count(DealtCards, Value=1)>0) And (Sum(DealtCards, Value)<=11)), 10, 0)
```

Determining if a hand has Blackjack

While not overly complex, this one took a bit of thought on the simplest way to do it. Eventually, this set of logic is what I decided upon:

- The total card value must be 21
- One of the cards must be a Rank of Jack
- The first Jack's Suit must be Spades or Clubs (there can only be one, so we can safely take the first)
- There must be exactly 2 cards in the hand.

That set of 4 logic operations was the fewest number of steps I could come up with to determine BlackJack. You'll notice that nowhere in that set of logic do I check if there is an ace in the hand – as it turns out, that would be an unnecessary fifth check:

- You don't have to check if there's an Ace in the hand, because there's no other way for a Jack and one other card to sum to 21
- You cannot replace the 2-card check with a check for an Ace also being in the hand, because an Ace, black Jack, and another 10-value non-Jack card would also fulfill all logic expressions.

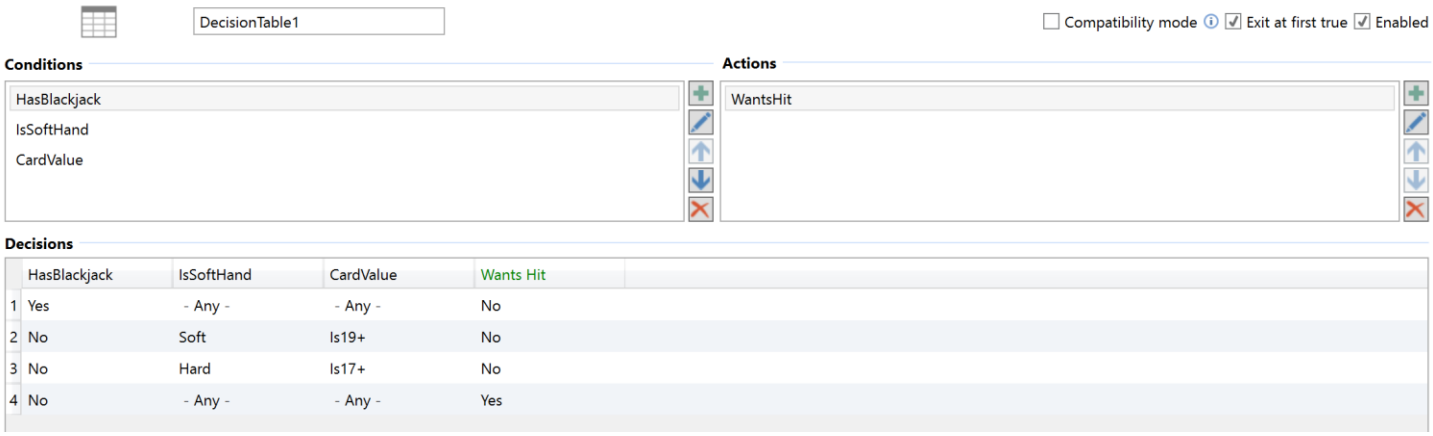
This can all be stuffed into a Calculated Field, so your business logic doesn't have to fool with anything other than coming up with the best possible strategy

```
CardValue=21 And Count(DealtCards, Value = 1)>0 And Count(DealtCards, Rank=11)>0 And (CollectionLookup(DealtCards, Suit, Rank="Jack")="Spades" Or CollectionLookup(DealtCards, Suit, Rank="Jack")="Clubs")
```

Soft/Hard Hand

This one was a new concept for me (it's possible that I'm not the most experienced gambler). The logic behind it is that you have a soft hand if you have an ace counting as 11; since it can drop to 1 if you're dealt a high card. Because of this, you can be more aggressive with taking hits than if you have either all fixed-value cards or an ace counting as one point.

When the Player is taking their turn and deciding if they want to hit or pass, my initial logic followed the HIGHLY complex strategy of taking a hit if the summed CardValue was under 17 (as I said – gambling is not my strong suit. Pun intended). By considering the current hand being hard or soft, I was able to add a bit more finesse to the hit/pass logic – which improved overall performance.



After working through these challenges, the result is a pretty slick little Rule App that is able to play Blackjack with itself. Is it useful? Highly questionable. Is it a great exercise in constructing a rule application with a wide variety of different logical and functional bits? Absolutely!