Sample Preliminary Software Design

This is a sample preliminary software design for the card game Hearts. Since it's not a grid-based game, this isn't a game that you can do for your project, but this page should give you a good idea of what your preliminary software design should look like.

If you're not familiar with the game of Hearts, you can read about it here:
http://boardgames.about.com/od/hearts/a/hearts_rules.htm

Model

Constants

MIN_PLAYERS the minimum number of players in the game (3)
MAX_PLAYERS the maximum number of players in the game (7)

Suits	
HEARTS	0
DIAMONDS	1
CLUBS	2
SPADES	3

Even though numeric values are specified for suits, nothing in the program should depend on these specific values.

Ranks from 2 to 10 will be represented by integer values. The ranks defined below should have the numeric values specified below so that they will work with the other ranks.

Ranks	
JACK	11
QUEEN	12
KING	13
ACE	14

The following types of objects will be used in the game:

Card

Each Card object will have a **suit** property and a **rank** property.

- **suit** one of the following values: HEARTS, DIAMONDS, CLUBS, SPADES
- **rank** a number in the range 2 to 14. The variables JACK, QUEEN, KING, and ACE can be used for ranks in the range of 11 to 14.

Card object examples:

```
{"rank": 2, "suit": CLUBS}
{"rank": QUEEN, "suit": SPADES}
{"rank": ACE, "suit": HEARTS}
```

PlayedCard
Each PlayedCard object will have a **suit** property and a **rank** property.

- **card** a Card object
- **playerNumber** a player number (index into the players property of Game objects)

PlayedCard object example
```
{"card": {"rank": 2, "suit": CLUBS}, "playerNumber": 2}
```
(Assumes there are at least three players in the game.)

Player
Each Player object will have the following properties:

- **number**: integer
- **name**: string
- **hand**: array of Card objects (usually 17 or fewer cards)
- **discard**: array of PlayedCard objects (zero to 52 cards)
- **score**: a non-negative integer

Game
Each Game object will have the following properties:

- **players** an array of Player objects (no fewer than MIN_PLAYERS and not more than MAX_PLAYERS). Player numbers are indexes into this array.
- **deck** an array of Card objects (zero to 52 cards).
- **trick** an array of Card objects (not more than MAX_PLAYERS).
- **nextToLead**: a Player number
- **nextToPlay**: a Player number
- **endScore**: a positive integer that is the score a player needs to win the game

Functions/Methods

The following functions could be methods of Game, Player, or Card objects, but will be described as standalone functions here.

deal()
Creates a full deck, shuffles the deck, and puts cards in the **hand** array property of each player. (To ensure that all players have the same number of cards, some cards will be removed from the deck if there are 3, 5, 6, or 7 players.)

playCard(player, card)
Takes a player object and a card object as parameters. Determines whether the card can be legally played. If the specified card can be legally played, the card and the specified player's number are put into

PlayedCard object which is added to the **trick** property of the current Game object. The card is removed from the hand of the player.

endTrick()
Determines which player won the trick according to the rules of the game. (The player who played the card of the highest rank of the lead suit wins the trick.) The PlayedCard objects from the **trick** property of the current Game object are appended to the **discard** array of the player who won the trick. The **nextToLead** property of the Game object is set to the number of the player who won the trick. All PlayedCard objects are removed from the **trick** array.

endHand()
Calculates the score for the hand for each player based on the discard property of the player. Adds the hand score to the player's total score. game. (The player who played the card of the highest rank of the lead suit wins the trick.) The PlayedCard objects from the **trick** property of the current Game object are appended to the **discard** array of the player who won the trick. The **nextToLead** property of the Game object is set to the number of the player who won the trick.

NOTES

--There is *no information* in the model about how things will look in the browser. This same model could be used in a game that had a text interface or in a game with animated 3D models.

--I haven't described all of the variables and functions that will be used in the model, but I have described enough of them to decide that this approach will probably work well.

--JavaScript doesn't have constants, so the "constants" will actually be defined as variables but the program won't change their values.

--All of the names (of variables and functions) defined for this game could be defined in a single global object to avoid namespace collisions with libraries and frameworks.