CS106 W20 - Assignment 03

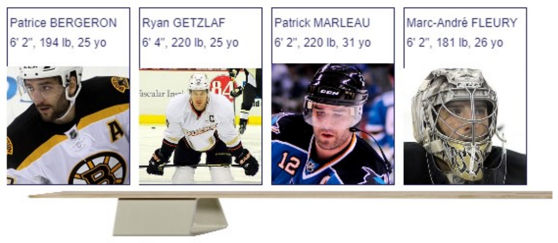
Due: Friday, January 24, 11:59 PM

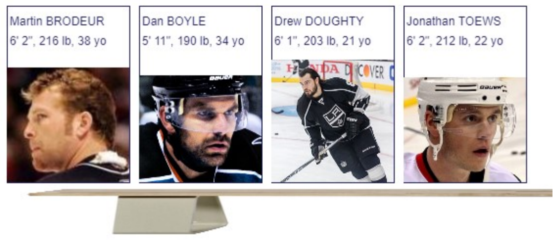
Revision Note (Jan 21): The scope of A03 Trading Canadians has been reduced, and more guidance has been provided.

Revision Note (Jan 14): The Trading Canadians hockey-card exchange was formerly Assignment 02. It is now Assignment 03.

This assignment is an application of containers, as discussed in the second week of CS106.

You are to write a JavaScript p5 program to model two generous Canadians giving away their hockey cards, as seen in https://youtu.be/m7Qk0y44O5s





Begin with this starter code: A03_StarterCode

Create a folder: CS106_A03

Save your sketch folder as A3_all, in the CS106_A03 folder.

Requirements and Grading

Starter (no marks)

First, examine the starter code in detail, to ensure you understand it as this will give you insights into how to use the starter code with the code you write. You will note the following from the starter code:

- The sketch takes up the full browser tab and resizes along with the browser window.
- You are provided with a data set of the Gold-Medal-winning Canadian Men's Ice Hockey team roster from the 2010 Vancouver Olympics. The data includes player attributes, found in the global variable *builtInPlayerList* in array-of-records form. The data also includes one image per player, as a *.jpg* file in the *data* folder, with a file name based on the player's attributes.
- The player images have different sizes and aspect ratios. The starter code includes everything that you should need to deal with this fact. See the function *resizeImageForCard*.
- The data set defines the attribute *Designation* on some, but not all, players.
- Running the starter code shows cards scattered randomly around the page.
- Running the starter code only shows an image for Sidney Crosby. Other players have a blank space where the image should be.

Container Enhancement (2 marks)

Fix the starter code so that all players get their correct images, at the right size. This probably requires modifying all of: *preload*, *startup* and *drawOneCard*. The file name format is First_Last.jpg.

Title and Caption (2 marks)

Implement this required text processing into your card design. This much is shown in the video.

- These two formats of text lines must occur exactly. Example:

```
Sidney CROSBY
5' 11", 200 lb, 23 yo
```

- Each line must appear to be one smooth run of text.
- The player's last name must be written in all uppercase.
- The player's height and weight must be given in Imperial units, formatted like the example above.
- The player's age must be calculated as-on December 31, 2010. Note that this date is chosen to make calculation easy. For example: Sidney Crosby (whom the starter-data has born in August 1987) was 23 on December 31, 2010 because he turned 23 sometime during 2010.

Card Design (4 marks)

Implement an original and visually appealing card design. No such design is shown in the example video and submitting the level of visual design seen in the video will receive little or no marks for this section. It must include these elements:

- Presenting several of the attributes in the player data set, beyond those required above for the two lines of text.
- At least one use of oversized text.
- At least one use of a non-textual element varying according to a card attribute, such as:
  - colour
  - choice of icon
  - size/position of a visual element
  - size/position of the whole card
  - visibility of the card (filtering).
- A basic overall impression that it is a hockey card (definition, examples).

Full marks for everything up to this point are achievable independently of implementing any trading functionality.

Layout and Trading functionality (6 marks)

Implement game-play functionality that models two traders exchanging hockey cards to form fantasy teams.

- Each trader has a hand of cards, suggested by a pattern of card locations on the screen. In the video's example, this is two horizontal layouts, across the top and bottom.
- Both hands start empty. When the program user presses the SPACE key, a card is dealt from a deck, first to one trader, then the other. Cards are dealt without replacement (each hockey player appears at most once) and in random order on each program run (browser page refresh). If all cards have been dealt, then pressing SPACE does nothing.
- Clicking the mouse on a card gives the card to the opposite trader. It is removed from the hand where it was clicked. Other cards must move to fill any gap left behind.

Overall requirements

General considerations for your solution include:

- Do not worry about overrunning available space.
    - For the dealt-out hands of cards, it's okay to assume infinitely much space is available for continuing your layout pattern. Do not add logic for scrolling, wrapping, etc. Your visual design should allow for at least six cards per trader when the window size is 800 px wide by 600 px tall. If a user creates a larger hand of cards, then it is okay to let the later ones run off the screen.
    - For text in a card, it's okay to assume text lengths given in the starter-code data set. Do not add logic for text wrapping, nor handle unexpectedly long names, players weighing billions of pounds, etc. While this will not be tested, it is okay if, for example, a 100-character first name implied that text would be painted outside of a card's intended border. Your visual aesthetic marks will be based only on the player data provided.
- You are not permitted to modify *getCards* and you are not permitted to code-in or load any additional or re-stated player information.

General Correctness

- One mark will be deducted for files or directories named incorrectly (the zip file, etc.)
- One or more marks will be deducted if the program crashes (depending on the severity).

Assignments that do not run may receive a grade of 0. Even if you don't complete the entire assignment, don't leave it in a broken state. Make sure it runs so we can find ways to give you part marks.

General Style and Efficiency

- One or more marks will be deducted if your code does not adhere to the style and efficiency requirements given in Assignment 1.

- You cannot get part marks if the TA cannot understand what you are trying to do.

Restrictions

In general, you may not use any functions, libraries, or statements not covered in lecture or labs unless not specifically exempted below or in a post by a TA or instructor on this assignment discussion board. For example:

- NO translate(), rotate(), or scale() functions
- No classes
- You MAY use bezier, arc, and other standard drawing functions

If in doubt, make a post to ask about using a specific statement or function. Functionality marks will be deducted for using forbidden functions/statements.

Hints

You may use the following hints to help you design and implement a solution. Your solution may vary somewhat from the hints.

- Treat *builtInPlayerList* as an invisible deck of cards that have not been dealt yet. Shuffle it once, during startup.

- Add an attribute called *image* to each card record. It stores the player's loaded photo. Set this attribute in *preLoad* while looping over all the cards. Because you won't have dealt any cards yet, this means modifying what you find in *builtInPlayerList*. An example of doing this is in the starter code's *setup*, where it adds the properties *TEMP_x* and *TEMP_y*.

- Define two arrays, *trader1Cards* and *trader2Cards*. These start out empty. Dealing a card means removing from the end of *builtInPlayerList* and adding to the end of one of *trader1Cards* or *trader2Cards*. Use *push* and *pop*.

- Loop over all the cards again in *setup*, just like in the starter code. Instead of picking random *TEMP_x* and *TEMP_y* values, do these preprocessing steps:
    - Call *resizeImageForCard* to get the image scaled to the size you need.
    - Do the player's age calculation and store the result in a new attribute called *ageAt2010Close*.
- Define and calculate these helper numbers, which you will need repeatedly, when drawing and hit testing.
    - Global variables *trader1CardRowY* and *trader2CardRowY*. Calculate their values in *setup*.
    - A function *cardX*(*cardNum*). It returns the starting x-coordinate of any card that is at position *cardNum* in its row.
- Draw *trader1Cards* across the top, and *trader2Cards* across the bottom, by using two loops. This means no cards will get drawn until you deal them. Find or make a fixed background, like the benches in the video, so that the screen doesn't start empty. For example, you could create the benches using the functions rect() and ellipse(), or alternatively you could create a graphic of the benches in Photoshop and load that image.
- Your draw helper calls should go *draw* → *drawCards* → *drawOneCard*, with appropriate looping.

```
// handOfCards is one of trader1Cards or trader2Cards
// cardRowY is one of trader1CardRowY or trader2CardRowY
function drawCards(handOfCards, cardRowY)

// Draws card at position (x, y).  The parameter card is a record, with properties
// like FirstName and image.
function drawOneCard(card, x, y)
```

- Your *mouseClicked* helpers should go *mouseClicked* → *hitQuery_Mouse_Cards* → *hitTest_Mouse_Card*, with appropriate looping. Depending on the outcome, they may also go *mouseClicked* → *doGiveaway*.

```
// Returns a value n (0 <= n < handOfCardsLegth), if the current mouse position is over
// card number n, from a row of cards aligned with y = cardRowY.  Returns -1 otherwise.
function hitQuery_Mouse_Cards(handOfCardsLength, cardRowY)

// Returns true if the current mouse position is over card number cardNum, from a row
// of cards aligned with y = cardRowY.  Returns false otherwise.
function hitTest_Mouse_Card(cardNum, cardRowY)
```

```
// Removes the card at position cardNum from fromHand and adds it at the end of toHand;
// fromHand is one of trader1Cards or trader2Cards; toHand is the opposite one.
function doGiveaway(cardNum, fromHand, toHand)
```

- A nice-looking card is possible by creating a whole-card background image manually (Photoshop, Paint, …) and having your code draw the player-specific elements on top of it.

Submitting

Add a file README.txt to your sketch folder. Include attributions for any materials like graphics that are included in your submission. If you designed it yourself, say so.

Zip your assignment sketch folder (A3_all) into one zip file CS106_A03. Submit it to the correct assignment dropbox. Consult "How to Submit" on Learn for more information on how to create a ZIP.

It is your responsibility to submit to the correct dropbox with the correct files before the deadline. Otherwise you will receive a mark of 0.