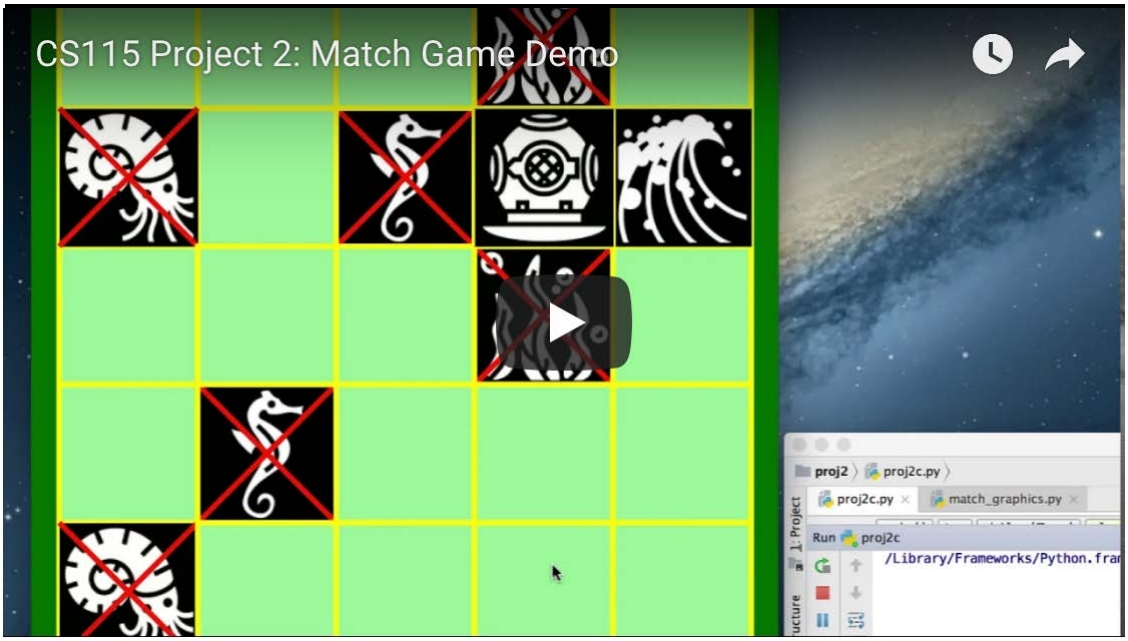
CS 115 Project 2, Spring 2017: Match Game

[Back to CS 115 schedule]

Summary: In this project, you will make a simple matching game using the graphics package. The idea is that users will click on cards one at a time, to flip them over to find pairs. The below video shows the final game and a demonstration of gameplay.



Due dates:

- *Checkpoint A:* Due as a demo in workshop, lab, or tutoring hours before Thursday Mar. 23 at 7PM
- *Checkpoint B:* Due as a demo in workshop, lab, or tutoring hours before Thursday Apr. 6 at 7PM
- *Final code:* Due via Moodle on Thursday, Apr. 13 at 11:55 PM

Getting Started

Download the following files into your PyCharm directory for Project 02:

- yourlastnameP2.py, a blank template for Project 2
- graphics.py
- match_graphics.py, a support library for this project.
- icons.zip (sealife set) or icons.zip (adventuring set) or build your own set (see Extra Credit).

Do not modify `match_graphics.py`.

The file `icons.zip` should be unzipped to create the directory `icons/` which holds a series of .gif files. Do not rename these graphics or their parent directory, since the file names are listed in `match_graphics.py`. The directory `icons/` needs to be in the same directory as `graphics.py`, as `match_graphics.py` and as your code.

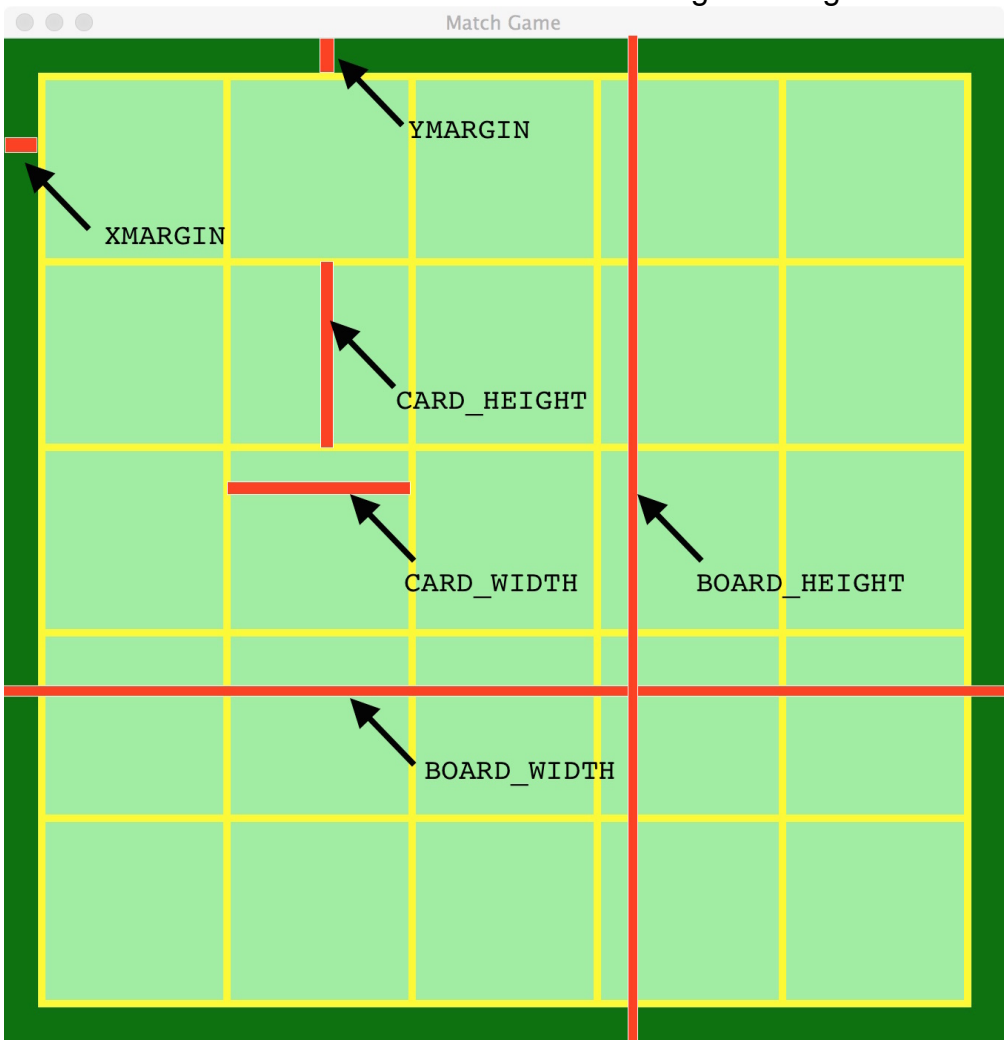
Checkpoint A (5 points)

For Checkpoint A, you will need to demonstrate a program that draws the game board with all the cards shown. You need to be able to display the following window to complete the checkpoint:

Specifically:

- Complete the `shuffle_cards()` and `show_card()` functions (read their docstrings for guidance).
- The board is comprised of 25 rectangles with a 'Yellow' border of width 5-pixels. A card image is centered in each rectangle.
- The cards are randomly shuffled and placed each time. There are 24 pairs of cards and one extra card.
- Each time you start your program, the extra card should be randomly selected and the placement of the cards within the grid should be random.
- Use the constants defined in `match_graphics.py` for the window dimensions, etc. You can

see how these constants relate to the board using the image below:



Hints:

- Review all the code in `match_graphics.py`.
  This holds important constants, data structures and functions that you are expected to use in your program. For example, the names of the card images (in the `icons/` directory) are written in a list, called `images`.
- The 'hidden cards' in our game are rectangles filled 'LightGreen' with yellow, 5-pixel borders. If you have a `Rectangle()` called `r1`, you can give it a border using `r1.setOutline('Yellow')`. To make its border 5 pixels wide, use `r1.setWidth(5)`. See our past labs if you need help recalling how to fill a shape with a color using the graphics library.
- Complete the `shuffle_cards()` function.
  The goal of the `shuffle_cards()` function is to generate a two-dimensional `cards` list, where `cards[i][j]` holds the name of the card that should appear at position (i,j) in the board grid. The function `shuffle()` from the library `random` can be used to randomly permute the list, modifying the list in place. Thus, `shuffle(images)` will shuffle all the images.
- To display an image, use the `Image()` object from `graphics.py`. The documentation is here: http://mcsp.wartburg.edu/zelle/python/graphics/graphics/node13.html.
  Here is some sample code that displays "sample.gif" centered at the point (400, 400) in a window called `win`:

```
card = Image(Point(400, 400), "sample.gif")
card.draw(win)
```

Checkpoint B (5 points)

Your Checkpoint B code needs demonstate the following:

- The game should start with all cards hidden, like this picture
- The game should cycle through pairs of rounds, where the user is either making a 'first pick' (round n) or a 'second pick' (round n+1).
    - On a 'first pick,' the card they click is shown.
    - On a 'second pick,' the second card they click is also shown.
    - After the 'second pick' is shown, pause for 1 second and then decide if the 'first pick' and the 'second pick' are a match.
    - If they are a match, leave both showing. If they are not a match, hide both.
- If the 'second pick' is the same exact card as was selected in the 'first pick,' then the game should ignore the most recent 'second pick' and let the user make a different 'second pick' instead.
- If a 'pick' is off the game board, then the game should ignore the click and let the user make a different 'pick.'

Hints:

- To accomplish the above tasks, you need to implement the `hide_card()`, `get_col()` and `get_row()` functions.
- To pause the game for 1 second, call the function `game_delay(1)`.
- Implementing the above, you may find the `continue` or `break` control-flow keywords useful. These are demonstrated in this PythonTutor code.

Final Code

In your final code, you will extend your Checkpoint B code. At this point, the game will mark matches, remember prior matches and detect when we have enough matches to declare victory. It should look just like the opening video in behavior.

- When two cards are 'matched,' then draw a red 'X' across each card. Implement the `mark_card()` function.
- When a 'pick' is made that is a card that is already shown because its part of a match, ignore this click and let the user 'pick' again.
- When all pairs of cards are matched, detect that the game has been won and call the `you_won()` function.
- If the user closes the graphical window at any point, your program should simply exit cleanly. In other words, encorporate exception-handling logic to deal with the `GraphicsError` exception.

Hints:

- For the final code, you will need to implement the `mark_card()` function.
- To remember that the card at row i and column j has been matched with some other card successfully, consider adding the pair (i,j) into a list to track this data. To add something to the end of the list, use its `append()` member function:

```
matches = []
...
pair = (i,j)
matches.append(pair)
```

- To see if (i,j) is in a list, you can use the following conditional logic, demonstrated here in some PythonTutor code too:

```
if (i,j) in matches:
    print("Hey, (i,j) is in the list 'matches'")
```

There is no demo for your final code. See the end of this specification for submission instructions.

Grading

Correctness [65%]

The most important part of your grade is the correctness of your final program. Your program will be tested numerous times, using different inputs, to be sure that it meets the specification. A detailed grading sheet for this project is posted here.

Programming Design and Style [25%]

In addition to being correct, your program should be easy to understand and well documented. For details, see the grading sheet above.

Checkpoints [10%]

Your checkpoints are each worth 5 points, as described above. The checkpoints cannot be submitted late.

Extra credit [up to 5%]

There are lots of possibilities to extend this project for extra credit. For example, you may select alternative game icons from http://game-icons.net (I converted these to 125x125 pixel GIFs to make the two icon sets provided above). You may also use different color schemes to create your own themed game. If you have an idea you'd like to implement, then (*at least a day before final code submission*) you need to propose it to the instructor in writing on Piazza and get approval for the idea. You are responsible to making sure it is possible to satisfy the whole project grading rubric while also encorporating your extra credit: you can't add functionality if it means removing some graded functionality

---

Submitting your final code

You should submit your final code on Moodle by the deadline. As a backup, I strongly encourage you to upload a copy to your *blue* directory in case something goes wrong with your submission to Moodle.

Please name your file *yourlastname*P2.py, substituting your actual last name (in lowercase) as indicated.

Collaboration policy

Programming projects must be your own work, and academic misconduct is taken very seriously. You may discuss ideas and approaches with other students and the course staff, but you should work out all details and write up all solutions on your own. **The following actions will be penalized as academic dishonesty:**

- Copying part or all of another student's assignment
- Copying old or published solutions
- Looking at another student's code or discussing it in great detail. You will be penalized if your program matches another student's program too closely.
- Showing your code or describing your code in great detail to anyone other than the course staff or tutor.

Late policy

There is a 48-hour grace period associated with the final project deadline. This grace period is designed to **only** cover small personal emergencies and other unexpected events. No other consideration will be given for these small emergencies.