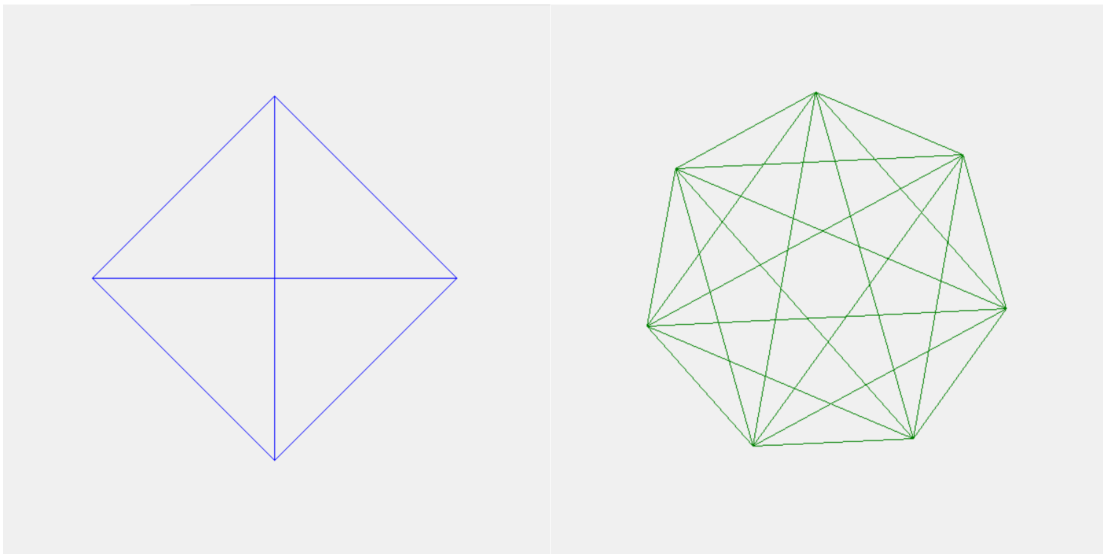
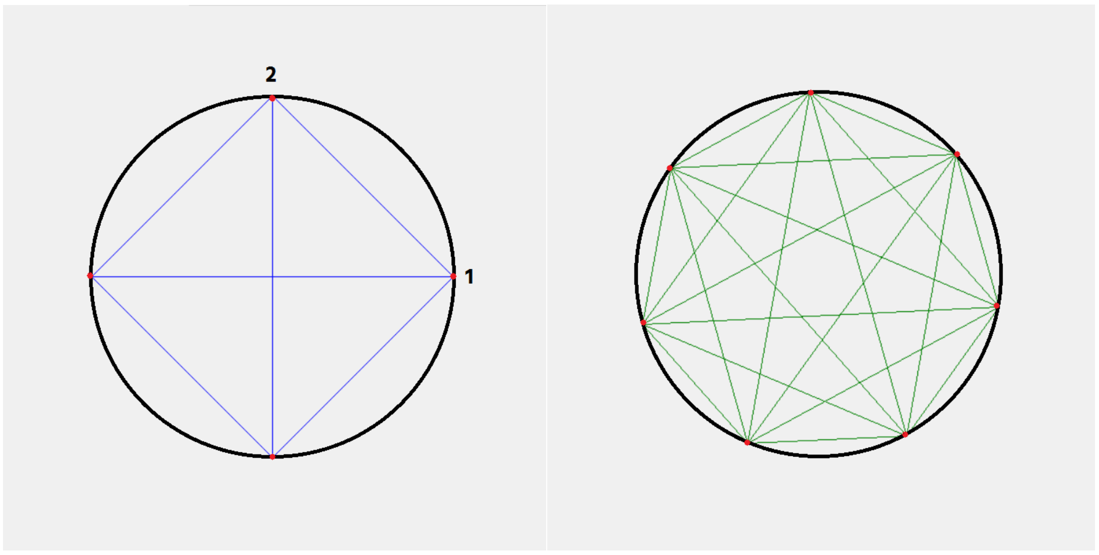
Homework 4.1

1. Fancy Wheel

These are two examples of a fancy wheel.



A fancy wheel is completely made out of lines. You can draw a fancy wheel by connecting each vertex to every other vertex. Each vertex is located on an “imaginary circle” around the wheel, as seen in the picture below.



You can determine the position of each vertex around the circle by calculating its angle from the horizontal, which increases by 2π/(# of vertices) from the previous vertex. For instance, vertex 2 on the 4-vertex fancy wheel differs by 2π/4 = 1/2π from vertex 1.

Keep in mind that the position of a point from the center (cx, cy) given the angle can be calculated by (x, y) = $(cx + r \cos \theta, cy - r \sin \theta)$, where r is the radius.

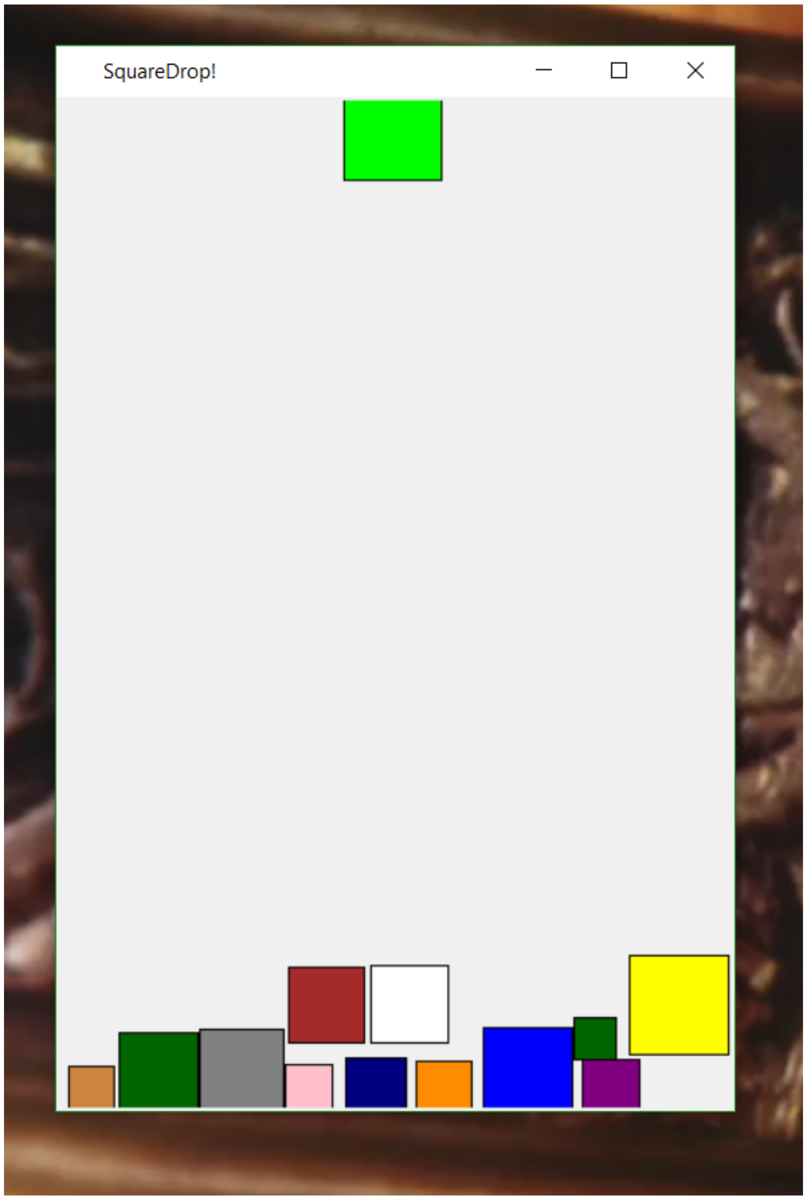
Create an animated fancy wheel program with the following specifications:

- The window size is 600x600.
- The fancy wheel is positioned at the center of the window, and has a radius of 200.
- The fancy wheel has N vertices, where N is initially 4.
    - If the user presses the Right arrow or the Up arrow, N should increase by 1.
    - If the user presses the Left arrow or the Down arrow, N should decrease by 1. N must be at least 2—so it will be a line at minimum.
- The fancy wheel can be colored red, green, or blue. It is initially blue.
    - If the user presses r, the fancy wheel's color changes to red;
    - If the user presses g, the fancy wheel's color changes to green;
    - If the user presses b, the fancy wheel's color changes to blue.
- The fancy wheel must spin, either clockwise or anti-clockwise when the timer is fired. It initially spins clockwise. It turns for 10 degrees for every 100 milliseconds.
    - If the user clicks the mouse within the wheel (or "imaginary circle"), then the fancy wheel changes its spinning direction.
    - If the click falls outside the imaginary circle, the click is ignored.

We encourage you to approach this question one property at a time: work on generating a static 4-vertex blue fancy wheel at the correct location first; then, allow incrementing and decrementing number of vertices, followed by allowing color changes. Finally, add wheel rotation and mouse click controller.

Here is an example of an animated fancy wheel:
https://www.dropbox.com/s/5yzapvocs4jm8cb/Fancy%20Wheel.gif?dl=0

2. SquareDrop!



The SquareDrop! game aims to fit as many randomly-sized squares as possible within the game window until no more moves can be made. The concept is similar to Tetris: a square starts dropping from the center-top of the window, and you will need to position it strategically, by moving it left or right, so that you can place as many blocks as possible. The game is over when a newly-generated square immediately overlaps with an existing square. There will not be any "line elimination" (as in Tetris), and it's okay for the squares to not touch each other perfectly.

Create SquareDrop!, which has the following specifications:

- The window size is 400x600.
- A side of a square is between 25 and 75, and there are 4 options for its color: red, green, blue, yellow.
- The game starts by spawning a square on the center-top of the screen (as seen in the picture above). Each square must have a random size and color within given bounds.
- The square moves down by 10 for every 100 microseconds.
- When a square cannot go further down (either due to another square or that it reaches the bottom of the window), it stays at its position until the end of the game, and a new square is spawned. Therefore, each created square will be preserved until the end of the game.
- No two squares can overlap with each other.
- The squares may not go out of bounds (i.e. go “outside” the window border).
- When possible, pressing the left arrow key will move the square by 10 to the left; the right arrow key will move it right by 10; the down arrow key will move it down by 10.
- As written above, the game is over when a newly-generated square immediately overlaps with an existing square.
    - When the game is over, you will need to stop spawning new blocks, display “Game Over”, and display the total number of squares on screen as the score.
- Pressing the r key will reset the game.

It is acceptable for two “completed” squares to have a vertical gap of less than 10 between each other (see the dark blue and white squares in the picture above).

You may find writing the helper function intersect(square1, square2), which checks whether two squares are intersecting, to be useful in writing SquareDrop!. You may also find the examples on lecture notes 4.1 (one box falling, raining circles) to be useful.

We encourage you to approach this problem incrementally—that is, start with one block with static size and static color, ensuring that the necessary properties hold for one block. Then, proceed with two blocks, and when the game works with two blocks, add the remaining features of the game.

- Step 1—one block:
  - Start by using fixed square size and color.
  - The square starts at the center-top of the screen, where the top-half is cut (i.e. out of bounds).
  - The square moves down by 10 for every 100 microseconds.
  - When the square hits the bottom, it must stop moving down and stay there until the end of the game.
  - The square is able to move left or right by pressing the left arrow key and the right arrow key respectively. Each key press corresponds to a moving the square by 10.
    - The square must not be able to go out of bounds: the entirety of the square must always be within the window.
  - The square is able to move down faster by pressing the down arrow key. Each key press corresponds to moving the square by 10.
- Step 2—two blocks:
  - The first square (and subsequent squares, when they cannot move any further down) is preserved until the end of the game.
  - The second square starts at the correct place.
  - Write the intersect function. Ensure that it is working as desired by testing it with the two blocks.
  - Make sure that the second block "stacks" properly on the first block.
- Step 3—completing the game:
  - Ensure that all completed blocks are preserved, and no blocks may intersect throughout the game.
  - When the game is over, the game stops spawning new blocks, and displays a "Game Over" text with the game score, which is the total number of squares in the window.
  - Fulfill remaining specifications: r resets the game, and square size and colors are randomized within the given bounds.
  - Verify with the formal specifications on the previous page.

3. Fun Part!

We're giving away a large bag of Vanilla Caramel (or Caramel) Pittsburgh Popcorn on Friday's lecture, which you may keep all to yourself or share it for the class! The winner will be anyone who gets the highest score on SquareDrop! within the given specifications (400x600 window, blocks randomly sized between 25-75). You can participate as follows:

- Complete Homework 4.1 (i.e. uploaded to Autolab) by Thursday at 5pm
- Send an email to Matthew at msalim@cmu.edu by Thursday at 5:30pm, containing the following:
    - A screenshot of your game when it's over, with your score (which you think will be the highest in the class).
    - Your funniest joke, pun, or anything Game of Thrones.

Good luck!