Pac-Man – Animation

```
/**
 * Draw one frame of an animation.
 *
 * @param g the graphics context
 * @param w width of the drawing area
 * @param h height of the drawing area
 */

public void drawFrame(GraphicsContext g, double w, double h) {
    g.clearRect(0, 0, w, h); // clear window background

    // *** draw a frame here!

    // *** update animation variables here!
}
```

- for an animation, define the actions for one frame of the animation
  - draw everything – Pac-Man, ghosts, …
  - update everything – move Pac-Man and ghosts, handle things running into each other, eat pellets, …
  - build this up incrementally!
- the system provides the loop by continually calling `drawFrame`
  - that doesn't stop – to keep something from moving, don't update those animation variables

Pac-Man – Appearance

Your program should work like the demo, not like the real version of the game

Note that you do not have to duplicate the appearance of the demo exactly — as long as it is possible to tell the difference between the various game components and the specifications given above are met, you are free to choose exactly how things are displayed.

Pac-Man – Appearance

- does Pac-Man need to look like Pac-Man in the game?
  - the requirement is that Pac-Man look different from the ghosts and other elements
  - see the "Technical Details – Drawing" section for information on drawing an arc shape
- does Pac-Man need to be animated?
  - Pac-Man needs to move around the board and face in the direction of movement
  - the mouth doesn't need to open and close
- what should the "powered up" Pac-Man look like?
  - it should be distinguishable from not powered up
  - the demo changes the color (red instead of yellow)

Pac-Man – Appearance

- should the ghosts look like ghosts in the game?
  - the requirement is that they have different colors and look different from Pac-Man and other elements
  - a simplified version of what they look like in the real game is fine

Pac-Man – Specifications

- how big does the board need to be?
  - it should fill the window
  - the provided code makes the drawing area 800x600
    - you can change this if you want, but don't make it too big – the most common screen size is 1920x1080 so don't go bigger than that
- how many pellets do we need?
  - four power pellets
  - "a bunch" of regular pellets

Pac-Man – Behavior

- how can you ensure separation of the power pellets without hardcoding the placements?
  - you can hardcode the placements, though do so in terms of the width and height of the maze rather than assuming a particular size
- how do you get the powered up effect to last for a limited time?

The Passage of Time

The passage of time comes up in handling Pac-Man's powered-up state — the effect of eating a power pellet only lasts for a certain amount of time. This time can be measured in terms of a number of frames rather than seconds (or some other unit of actual time). To handle the limited-time powered-up state, Pac-Man has an instance variable for the power remaining. Set it to a non-zero value when Pac-Man eats a power pellet, and decrease the value by 1 as part of the "update animation variables" step in `drawFrame`.

Pac-Man – Behavior

- how do you get the ghosts to hunt the player, then run away during power-ups?
  - you don't – this is one of the simplifications compared to the original game

- Ghosts: There should be four ghosts with different colors. When a ghost hits a wall, a new direction of movement is chosen randomly. In addition, a ghost may occasionally change direction spontaneously. Running into a "powered up" Pac-Man causes the ghost to die. A dead ghost does not move. Finally, the appearance of each ghost should indicate whether it is alive or dead (e.g. by using different colors).
  - ghosts move in their current direction until a wall is hit or a new direction is spontaneously chosen
    - "spontaneously chosen" – in every frame, flip a coin with a low probability of heads and change direction if it comes up heads

```
if (Math.random() < 0.5)    // There's a 50/50 chance that this is true.
```

- when a ghost dies, can it come back?
  - no

Pac-Man – Detecting Overlaps

- how do you detect when Pac-Man or a ghost hits a wall?
- how do you detect when Pac-Man touches a ghost?
- how do you detect when Pac-Man eats a pellet?
  - the "Handling Geometry" section in the handout talks about this
- how do you tell if the pellet is facing the right way to count?
  - the pellet doesn't face in a direction, but Pac-Man does and that affects whether a pellet is eaten
  - consult the class design for how to answer a question about Pac-Man
- how do you do any of this when things are moving?
  - an animation is a series of frames rather than continuous movement
  - in the "update animation variables" step –
    - check if moving Pac-Man or a ghost would hit a wall before moving them
    - after Pac-Man and the ghosts are moved, see if their current positions overlap
    - after Pac-Man is moved, see if its current position overlaps a pellet

Pac-Man – Technical Details

- what's with `null` and the pellets?

- Use arrays to store the collections of walls, pellets, and ghosts. When a pellet is eaten, remove it from the array by storing `null` in that slot of the pellet array — don't just make it invisible by failing to paint it. (This does mean that you will need to be careful to skip over `null` values when drawing or doing other things with pellets.)

  - `null` is a special value meaning "no address" – the value `null` does not refer to an object

```
String[] words = { "hello", null, "goodbye" };
for ( int i = 0 ; i < words.length ; i++ ) {
  if ( words[i] == null ) {
    System.out.println(i+": "+words[i]);
  } else {
    System.out.println(i+": "+words[i]+" has length "+
                       words[i].length());
  }
}
```

  - at the beginning the array will be full of pellets, but as each is eaten, store `null` in that slot instead

Pac-Man – Implementation

- how do you use subroutines/functions/private helper methods to avoid repeated code?
  - avoid repeated code by defining a subroutine that does the task, and instead calling the subroutine multiple times
  - subroutine vs function depends on whether or not there is a return type
  - subroutine/function (`static`) vs private helper method (`private` and not `static`) depends on the location
    - use a subroutine/function in the class with the main program – `PacMain`
    - use a private helper method in classes which define objects