CS606: Computer Graphics / Term 2 (2022-23) / Programming Assignment 1 (A1)
International Institute of Information Technology Bangalore

Announcement Date: Jan 25, 2023
Submission Deadline: 11:59 pm IST, Feb 10, 2023 (Friday)

Summary: 2D interactive planar rendering with 2D translation, rotation, scaling/zooming.
Learning Objectives:

- WebGL programming
- Creation and rendering of simple 2D geometric shapes as primitives
- Transformation of 2D objects and scene -- instance-wise implementation
- Key(board) events for changing control modes, and transformation implementation
- Mouse events for picking objects

Assignment: Pac-ManController

The goal of this assignment is to develop a WebGL program that will allow the user to pick a Pac-Man maze configuration, and allow the movement of Pac-Man, as shown in https://www.google.com/logos/2010/pacman10-i.html
This is a 2D-rendering exercise, with the assumption that the game is on the x-y plane.

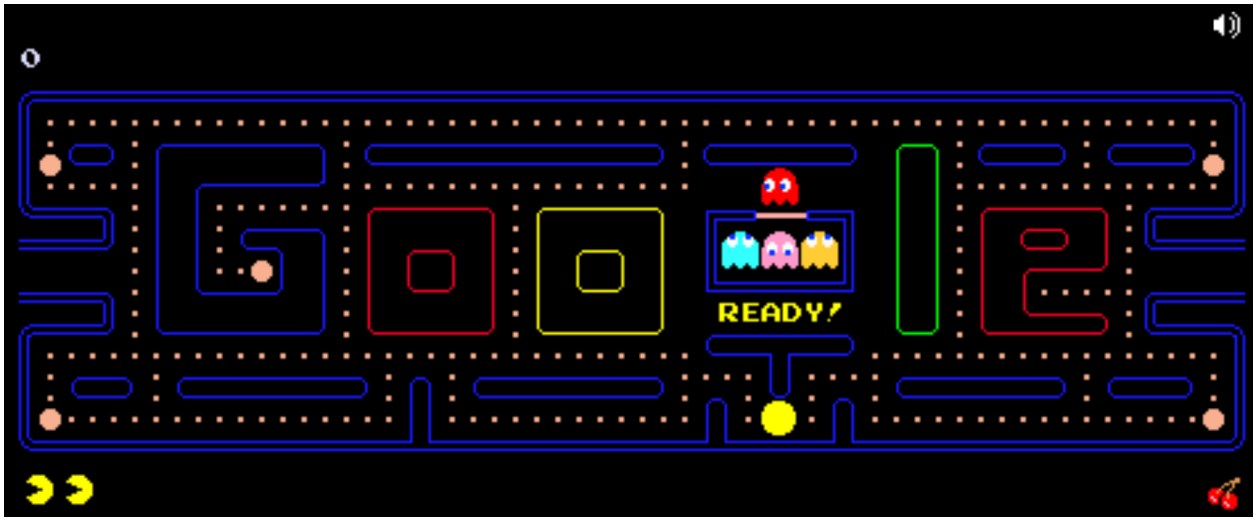

Figure 1: Pac-Man configuration as given in the Google Doodle

Rendering:

It is required to render Pac-Man and control its movements. However, it is not required to program the movement of the adversaries, i.e. the ghosts. The details of how to play the game are given at https://en.wikipedia.org/wiki/Pac-Man

> *The player controls Pac-Man, who must eat all the dots inside an enclosed maze while avoiding four colored ghosts. Eating large flashing dots called "Power Pellets" causes the ghosts to temporarily turn blue, allowing Pac-Man to eat them for bonus points.*

The ghosts may be rendered as simple triangles in their initial positions with a unique color other than blue. The color scheme used in the aforementioned Google Doodle (Figure 1) and the game description described in Wikipedia may be followed.

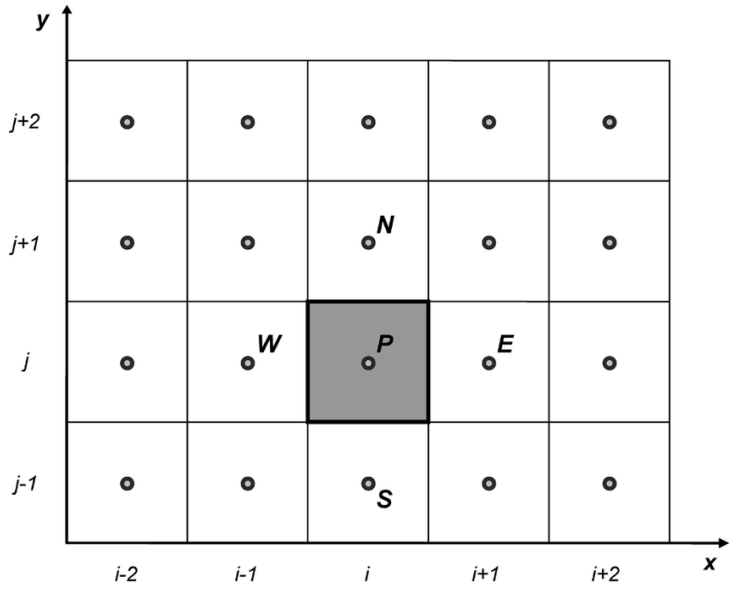

Figure 2: "Invisible" grid for rendering the points at the center of each grid cell.
(Image courtesy: Vieira et al. (2009) Modeling landscape evolution due to tillage: model development)

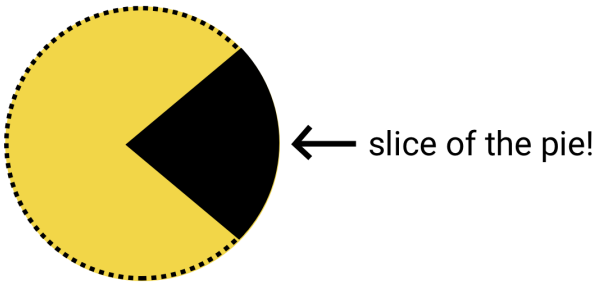

Figure 3: How to render Pac-Man? The gap of 90o marked as a "slice of the pie" is the mouth of Pac-Man. (Image courtesy: CSS-Tricks)

Specifications:

1. For the maze, imagine an “invisible” grid where a dot is rendered at the center of each grid cell, as shown in Figure 2.
    a. The grid data can be programmatically stored as a matrix of 1’s and 0’s where the matrix size is the maze resolution. 1 implies a maze point that can be accessed, and 0, otherwise.
        i. The maze point is rendered as a point of appropriate size.
        ii. We refer to locations and geometry as “points,” and rendered objects as “dots.”
    b. The dots must be rendered in orange (as shown in the Google Doodle) as an initialization. Once Pac-Man visits the dot, its color changes to green to indicate that it cannot be used for scoring any further.
        i. The Google Doodle is implemented differently in this regard, as the dot vanishes. Here, we distinguish between a non-maze/inaccessible point and an accessible point that has already been accessed.
        ii. *The scoring process does not have to be implemented here, as the focus is only on the graphics implementation that will enable the scoring process as a future goal.*
2. Pac-Man must be rendered as a circle with a gap of 90^{o} sector (Figure 3), just like its iconic symbol. It should be colored yellow. The gap for the mouth also indicates the orientation of the Pac-Man.
    a. Ideally, the Pac-Man “object” must be symmetric about the x-axis when walking along a path parallel to the x-axis, and similarly, symmetry in the y-axis when traversing parallel to the y-axis.
    b. The mouth should be closest to the next accessible point and aligned to its motion.
3. The ghosts can not move from their initial position and hence can be rendered as colored triangles or squares, following the same color scheme as in Figure 1.
4. Some of the accessible points in the maze may be rendered as a larger dot, indicating the “power pellet”.
    a. The size of the point rendered as a power pellet is higher than that of the “normal” dot.

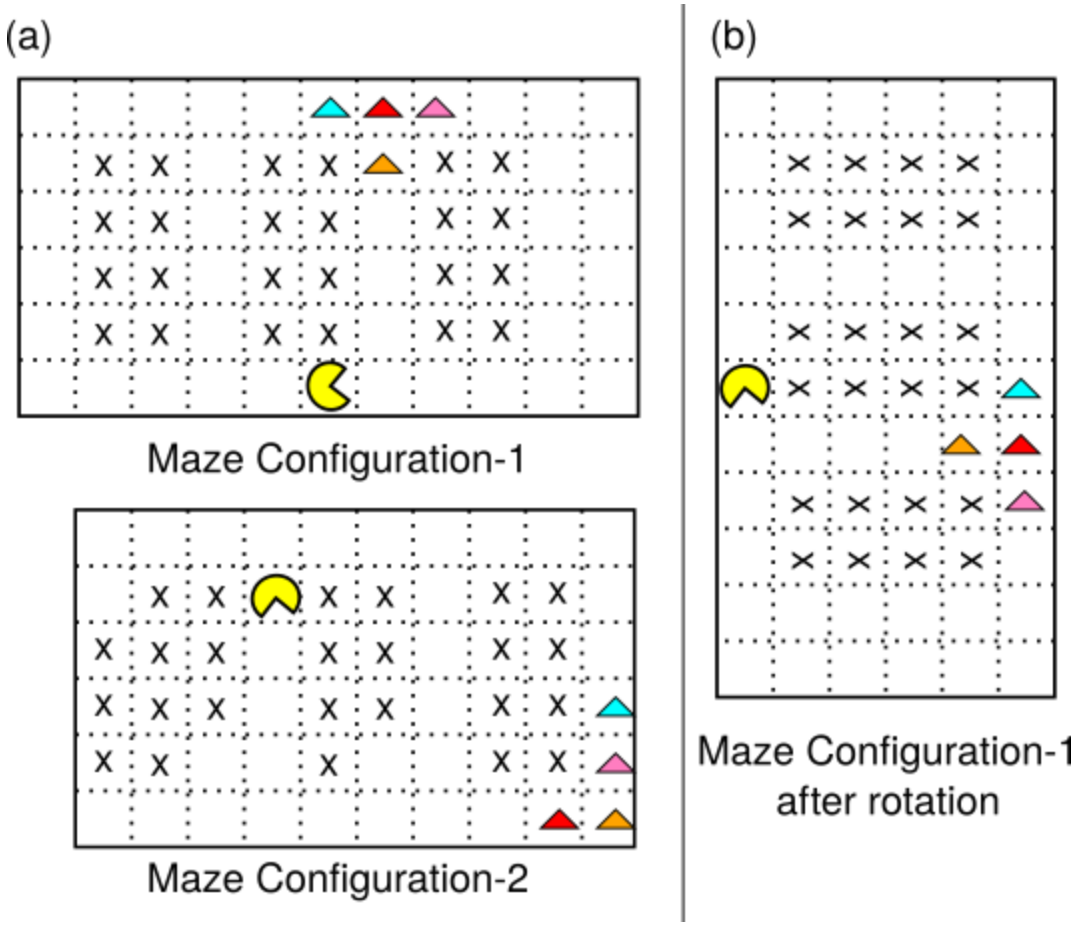

Figure 4: Maze configurations – (a) variants, and (b) after rotation. Note the locations and orientations of Pac-Man and the ghosts (colored triangles) after rotation. The x indicates inaccessible points. The dots for the accessible points or power pellets have not been rendered here, but the empty grid cells may be assumed to contain the dots. The dotted lines show the imaginary grid to enable the maze configuration.

Transformations:

1. Pac-Man should be able to rotate by 45° at a time and rotate sufficiently to orient itself to follow the maze. The rotations can be clockwise or anti-clockwise about the z-axis. Use the grid cell as the space to contain the rotations.
2. Pac-Man can "walk", i.e. translate by one dot at a time, thus moving from one accessible point to a neighboring accessible point. Once the Pac-Man "visits" an accessible point or the power pellet, its color changes to green after the Pac-Man passes, thus making it still an accessible point.
3. The maze itself can be rotated by 90° at a time.
    a. When the maze rotates, the objects (Pac-Man and ghosts) should retain their location in the maze, as shown if Figure 4 (a) and (b). The ghosts must retain their orientation, but Pac-Man may change its orientation as per the maze.

4. When Pac-Man accesses a power pellet, there should be special effects to indicate the power gain.
    a. The ghosts change to blue color.
    b. The Pac-Man grows to 1.5x its size (i.e., with respect to its radius), holds the new (larger) size in the current location, and returns to its original size once it moves to the next point. The special effect using the change in the size must be perceptible.
        i. While the scaling transformation is occurring Pac-Man cannot change its location or orientation. Any keyboard actions have to be retained invalid.

User interactions (keyboard controls):

There is a requirement that the Pac-Man must face the neighboring point it is going to access in the next step. This helps in orienting the Pac-Man as shown in the Google Doodle.

1. Arrow keys for x- and y-translation.
2. Rotations about the z-axis:
    a. Parentheses (/) for clockwise/anti-clockwise rotations of Pac-Man.
    b. Square brackets [/] for clockwise/anti-clockwise rotations of the maze itself.
3. Any specific key (say ‘c’) to iterate through the maze configurations, as shown in Figure 4 (a).
    a. Each maze configuration must specify an initial/starting position for Pac-Man.
    b. When the maze configuration is changed, Pac-Man is placed in the starting position of the new maze.
    c. Use at least 3 maze configurations.
4. User mode: there can be 2 modes – ‘normal’ or ‘modify’. The modes can be toggled by clicking on a specific key (say ‘m’).
    a. The ‘normal’ mode is when Pac-Man moves through the maze.
    b. The ‘modify’ mode is when all activities in ‘normal’ mode are disabled, and only picking and moving Pac-Man is enabled, as described in “Picking Objects.”

Picking objects:

1. The user can alter the position of Pac-Man in the ‘modify’ mode. This can be done by picking the Pac-Man and dropping it on a grid cell containing an accessible point in the maze.
    a. If the drop is on an inaccessible point, then the move is invalidated and the Pac-Man retains its current position.
    b. The state of the dots does not change when the current location of Pac-Man is changed in the ‘modify’ mode.

Questions to be answered in the report:

1. What did you do to ensure that the objects (Pac-Man, ghosts) do not change orientation or location when the maze rotates?
2. What did you do differently when rotating the maze vs Pac-Man?
3. If you were asked to scale the maze, what would be the steps you would implement? (Hint: indicate how the objects in the maze would be affected.)
