Assignment 1
CS606: Computer Graphics

Monjoy Narayan Choudhury
IMT2020502
IIIT Bangalore
Monjoy.Choudhury@iiitb.ac.in

I. Introduction

The aim of this assignment is to develop an application that achieves 2D planar rendering with 2D translation, rotation , and scaling. To associate a semantic meaning to the graphics application we try to recreate the famous ”Pacman” game. The geometric shapes implemented in this project are triangles, rectangles (squares), circles, and sectors of circles which are used to create the enemies, walls of the maze, pellets/power pellets, and the pacman respectively. The pacman can translate onto pellets, power pellets, and cannot go through walls while rotation involves the rotation of pacman itself about its own axis as well as rotation of the maze and all its components based on rules mentioned in the problem statement. The application also allows the user to change modes where they can pick the pacman up and drop it in a valid location.

II. Controls to use the application

1) Arrow keys for basic movement of pacman in the permitted regions.
2) ’)’ to rotate the pacman 45°clockwise.
3) ’(’ to rotate the pacman 45°anticlockwise.
4) ’]’ to rotate the maze 90°clockwise.
5) ’[’ to rotate the maze 90°anticlockwise.
6) ’c’ to change the maze configuration
7) ’m’ to change mode. Mode involves the standard game mode and drag-and-drop mode. The game starts in normal mode.
8) Mouse Click on pacman to latch onto it and move around in drag-and-drop mode. Clicking mouse again would signal a drop command and the pacman would drop/not drop in the location based on the business rules stated later.

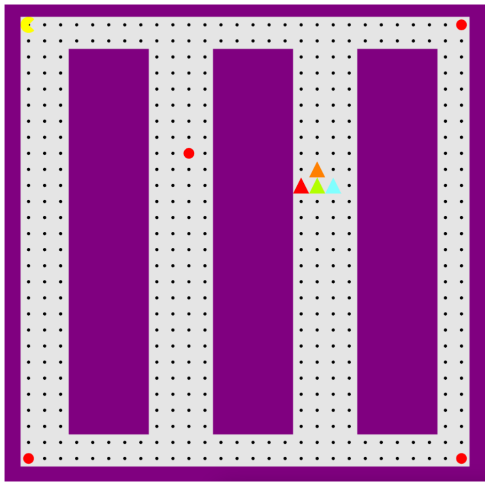


Fig. 1: Maze 1

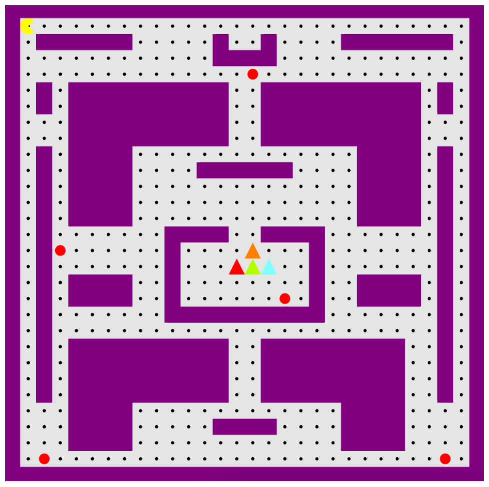


Fig. 2: Maze 2

III. Assumptions and design choices

A. Assumptions

1) There is no pacman and enemy interaction.
2) In modify mode if pacman is dropped onto a power pellet it wont trigger any change.
3) Taking the pacman in modify mode and dropping it in a power pellet won’t trigger the power pellet.
4) One can enter modify mode only when they are not on a power pellet.
5) Once the power pellet is visited by the pacman. Upon visiting the same power pellet it wont trigger to show the effect of pacman ”consuming” it.

B. Design Choices

1) Selection of space:
The entire code is written in conventions followed by webGL that is in the clip space. There are no inherent

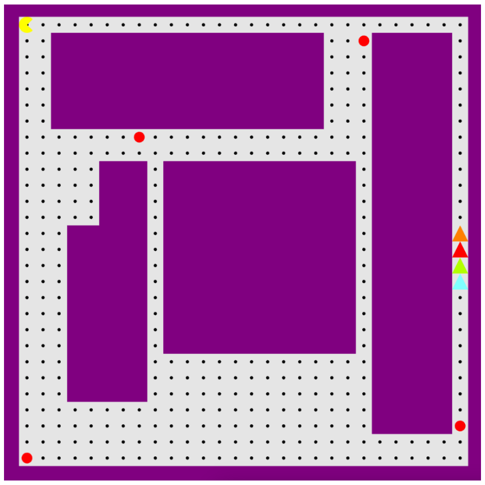


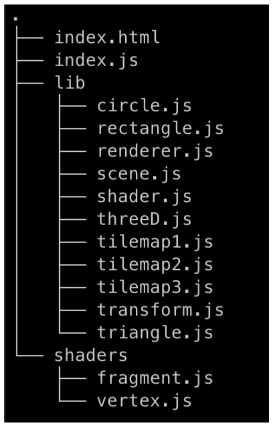
Fig. 3: Maze 3

conversions being made for converting clip space values into screen coordinates as such. Rather every value in grid coordinates is written in terms of how the clip space would interpret it.
Update: Upon enquiring the same from the instructor, it was advised to try to use the object space as it would help in the later assignments, and hence a complete migration of code to pixel space was done.

2) Grid sizes: Input grid size is kept square in nature (30x30) in this case. This even though doesn't affect the parameters which depend on the grid size of the code written, is done for easy backtracking in case of bugs.

IV. Code structure

Please refer to the code written in pixel space in the folder named code(pixel). The given code consists of the following files:



```
.
├── index.html
├── index.js
├── lib
│   ├── circle.js
│   ├── rectangle.js
│   ├── renderer.js
│   ├── scene.js
│   ├── shader.js
│   ├── threeD.js
│   ├── tilemap1.js
│   ├── tilemap2.js
│   ├── tilemap3.js
│   ├── transform.js
│   └── triangle.js
└── shaders
    ├── fragment.js
    └── vertex.js
```

Fig. 4: Code file structure

1) The code is shaders folder is the GLSL code for vertex and fragment shader. Here the notable change from standard code is that in the vertex shader, we have first converted our screen/ pixel coordinate values into normalized (0 to 1 values) and then have scaled it to fit -1 to 1 which is the clipping space used by webGL. In the code submission written in clip space this change is not present as it is not needed. The fragment shader in both version remains same.
2) in lib:
   a) Code files circle.js, rectangle.js, triangle.js are the basic primitives which we use to build this project. They create the vertex points required by webgl to render and associate a transformation property that allows us to perform affine transformation.
   b) threeD.js is just a named export aggregation
   c) tilemap1.js,tilemap2.js,tilmap3.js are 3 maze configurations stored in form of a matrix (30x30).
   d) shader.js and renderer.js encapsulates all the webgl based code which prevents us from writing the same webgl calls again and again.
   e) scene is the abstract concept where all the primitives are stored and scene.js tries to emulate that concept.

V. Business Logic for Operations

A. *Changing of Maze*

The code involved with the creation of a maze and filling it with appropriate sprites in required places present in index.js is encapsulated into a function called addPrimitives(grid) which takes in the grid matrix stored in tilemap1.js, tilemap2.js, tilemap3.js and creates graphical elements and adds them to the scene. Once the button for changing maze configuration is invoked we simply cycle through (implemented using a modulo counter) the grid values stored in the mazes array and invoke the addPrimitives() function with the new grid and re-renders the whole level.

B. *Rotating Pacman*

It is a simple rotation about the z-axis (default set) and the rotation point is its centroid can be done using code provided in transform.js (as part of the starter code).

C. *Getting the status of the pacman*

This set involves the question of whether the current position of pacman is a valid one. To compute the status we had to create another pixelVals matrix along with a grid matrix to create a one-to-one correspondence with what grid coordinate map to which pixelValues. In this part, we were plagued with the curse of Floating point number comparison in javascript which was handled by computing the absolute difference between 2 float values and checking if it was very close to zero (using a threshold value). Using the above hack we compared pacman's current position to the pixelVals grid values and simply returned the value at the corresponding grid coordinate. This helped us check whether pacman was moving to a pellet, power pellet, or into a wall and act accordingly.

D. Rotation of the maze

For graphical rotation a simple rotation 90°clockwise or anticlockwise about the maze center coordinates along the z-axis should have got the job done easily. But since this is an interactive game, a simple graphical rotation would not have sufficed. We have to respect the fact that the walls physically have also moved to a newer location along with pellets, powerPellets, and of course the pacman. However, there is one invariant fact that the surrounding of an object should remain same i.e if the pacman was besides a power pellet then after rotation too it must be besides that specific pellet only. Our approach can be summarised in the following algorithm:

1) Rotate each primitive graphically based on the orientation required
2) For the given primitive: obtain the centroid component(in case of any shape) and also the current translation value in case of pacman.
3) Find the corresponding grid coordinates (indices i,j from pixelVals) by matching the component values in 2) and pixelVals matrix.
4) Based on the orientation there can be two outputs for the coordinates.
    a) For clockwise movement for a matrix mxn. Value i,j changes to j,m-i-1 (for 0-based indexing).
    b) For anticlockwise movement for a matrix mxn. Value i,j changes to j-n-1,i (for 0-based indexing).
5) We then need to recompute the pixel value matrix pixelVals with the new grid. This can be precomputed once before the above steps are done.
6) Once we have the new pixel value matrix and the new index for the location of an object then we can just update the respective parameter (translated difference value for pacman, translated difference value and centroid values for triangle and centroid values for the other).
7) Note for the triangles we need to perform only a translation and not a rotation but since we had applied the same rotation to all primitives we need to reverse that effect for the triangle.

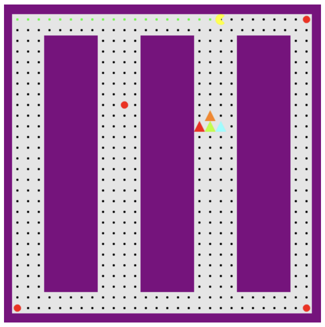


Fig. 5: Maze 1 - Default

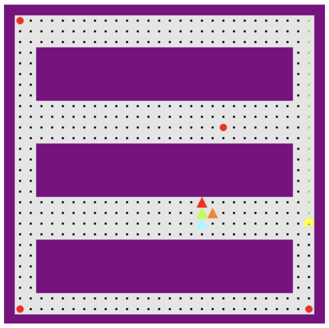


Fig. 6: Maze 1 - 90°clockwise

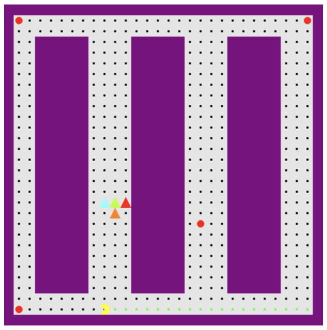


Fig. 7: Maze 1 - 180°clockwise

E. Mode changing and onClicks

We convert the mouse coordinates to pixel coordinates and then compare the converted mouse values with the value of the pacman to lock onto it. After that, we keep track of the movement of the mouse and apply the respective amount of translation required on the pacman graphic. We added an additional flag to make sure this movement of pacman is not confused with the standard movements and that no interaction with power pellets occurs in this mode.

VI. Answer to Questions

A. What did you do to ensure that the objects (Pac-Man, ghosts) do not change orientation or location when the maze rotates?

1) For pacman upon rotation of the maze we need to make sure that it stays in the same position. For this, we make sure that the point about which pacman rotates is its own and not the maze center. This makes sure that the pacman rotates about its own center but not around the maze. Now to make sure it is in the right place we use the approach mentioned in Section IV part D where we map the current grid coordinates to the newer grid coordinates after transformation.
2) For the ghosts rendered as triangle one design choice was taken that its centroid would necessarily correspond to the next grid point on the pixel space (similar to how a circle or a square's center is processed here) and corresponding values in pixel space for uniformity in the pixelVals. We don't apply the angle rotation on

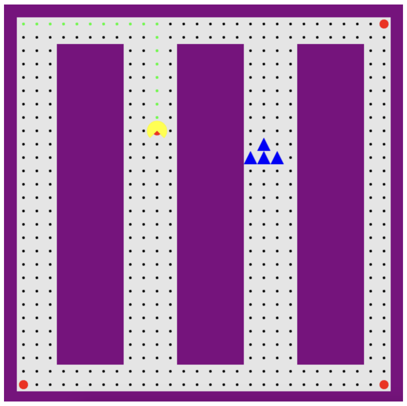


Fig. 8: Pacman consuming a power pellet

the triangles and directly apply rotateItem() on it which translates it to the respective position in the rotated grid.

B. *What did you do differently when rotating the maze vs Pac-Man?*

Rotating the maze is pretty simple if we don't count in the pacman. For each element in the scene, you apply the rotation about the point in maze center. While rotating the pacman we have to be careful about the point we are planning to rotate it. We rotate it about its own point and not the maze center. After that, the same approach mentioned in Section IV part D is applied by using the right values for the coordinates of the triangle.

C. *If you were asked to scale the maze, what would be the steps you would implement?*

This depends on the exact definition of scale. From my understanding, we want to increase all the maze components by some size, basically, making the maze larger without changing the grid size. This can be easily done in the current code as we just need to apply scaling using matrix transformation (similar to the pacman upon consumption of a power pellet) to all primitives part of the scene or the maze based on what you want to grow in size and what not.

VII. Conclusion

This assignment helped us learn the basics of WebGL and programming in JavaScript. It also allowed us to implement matrix transformation as the model transformation matrix as discussed in class. The event handling using mouse and keyboard calls helped us understand the thought process which goes behind making a graphics application interactive. We can now hope to extend these concepts and build more complex 3D objects and render them using WebGL leveraging the fact that 3D space is extension of the 2D space. Please consider going through the video which runs through the entire code to understand more.