CG Assignment 1 Report IMT2020501

Implementation and Outcomes:

The implementation consists of 3 main things, The pacman, the Pellet array and the Grid array. Then I will go by each functionality and a brief idea about how I implemented it.

The coordinate system :

I changed the coordinate system so that the top left corner of the screen is the origin and the bottom right corner is going to be the screenWidth, ScreenHeight.
This is done through multiplication of the u_resolution matrix in the vertex shader and all conversions to clipspace take place over there.

Rendering Circles in WebGL :

The circles were all rendered by drawing multiple triangles. The main point here was that since pacman was also a circle it had many similar properties like the circle. The center of the circle was the main key which was used to render the circle at the respective position on the screen.

Pacman and **Pellets** are essentially circles.

About Grid :

The grid is essentially a matrix of 0s 1s and 2s for spots that are not allowed to be accessed, allowed to be accessed and will have a pellet and allowed to be accessed and will have a super pellet.
The grid forms the backbone of the rendering of the pellets and the collision. Each time a grid gets activated either at creation or when the orientation of the grid changes or the map is swapped out for a new one.

To render the grid I make Squares with top left corner at i*chunksize, j*chunksize and width and height = chunksize.

Chunksize is the number of pixels that I want each grid square to be. Here I have set it to 50 pixels.

About the pellets :

As mentioned the rendering of the pellets happens based on the grid for each grid element if there is a 1 a normal pellet is rendered and is made of type food. If there is a 2 in that position on the grid a Super Pellet is rendered and is given the type super pellet. The other point to note here is that all the pellets are stored in a pellet array that is found in the index file for passing it in case an ‘Eat pellet’ Event occurs.

About Pacman

Implementational detail : All the positions that the pacman class stores are recorded in the posX and posY and center list. posX and posY are the “Deltas” in X and Y axis from the center of the pacman. To transform the pacman these deltas are modified according to how much the pacman has moved since the spawn.

The pacman class has several major functions explained as follows (this includes the bulk of the logic for collisions transforms and also the movement of the pacman class in general)

checkwall() :
Takes in the values of the pacman position and then returns true if there is no wall and false if there is a wall

move() :
Called from index.js when there is a key pressed event. Has functionality to enable the pacman to move up down left and right via the transform functions in the transform.js class.
Depending upon the event the corresponding position of the pacman is incremented by the chunksize and the transforms are applied. (More on that later)

rotatePacman():
This function is used to implement the manual rotation of the pacman using the ‘)’ key

teleport() :
In order to enable the transform to the position clicked by the mouse, this function will take the coordinates on the screen of the mouse click and convert them into the nearest valid block in the grid (by floor division with the chunksize and multiplying with chunksize again.). Does not allow the pacman to be placed on the red regions (Where there is a wall in the maze.)

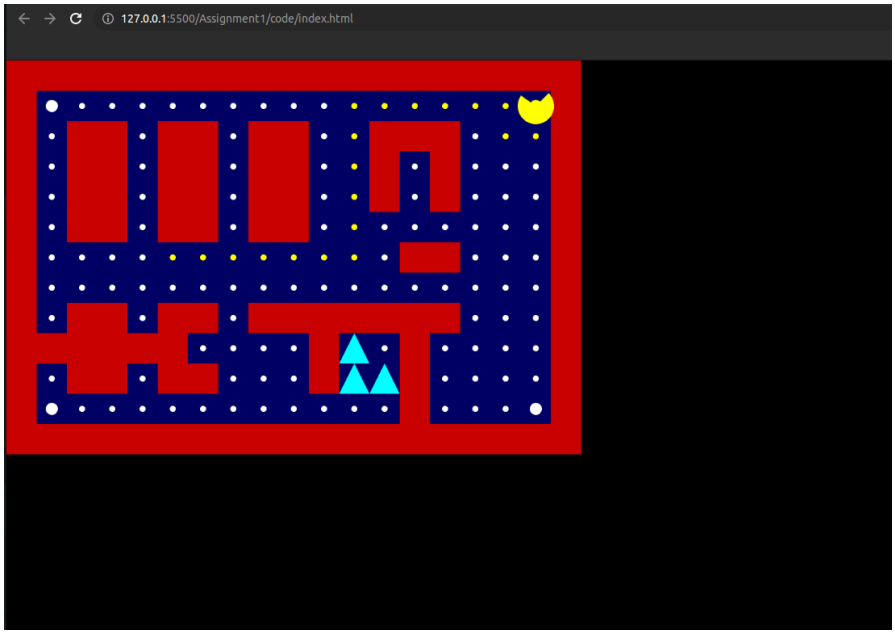
About the Transforms :

Rotate Transform : updates the model transform matrix with a rotation matrix made using the fromRotation() gl matrix function.

Translate Transfrom : updates the model transform matrix with a translational matrix made using the fromTranslation() gl matrix function.

TranslateRotateScaleTransform : takes the initial position of the object and then translates the origin to that point, applies the translation and scaling effects, then is multiplied by the matrix that takes the origin back to the initial position. And then finally I multiply the translation matrix that takes the object where it needs to translate to post being scaled and rotated appropriately.

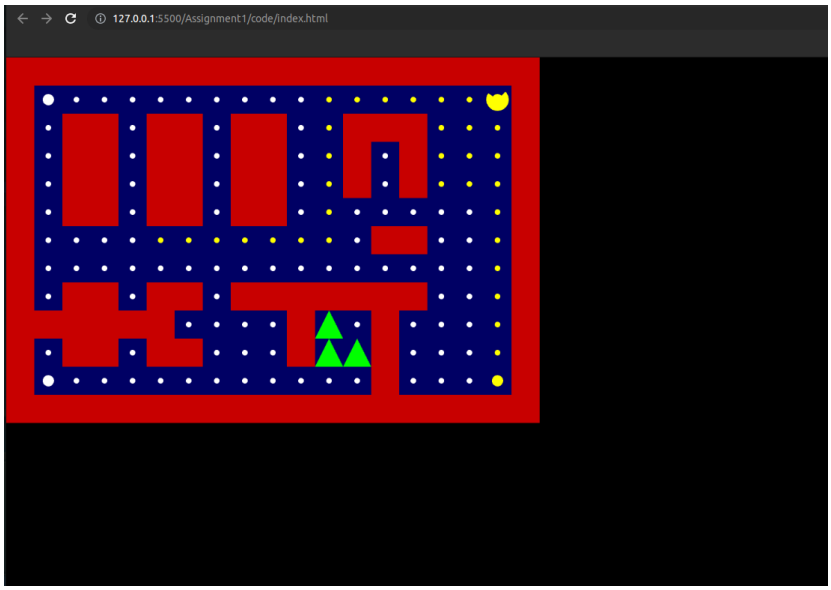
Snapshots of Pacman moving based on above description



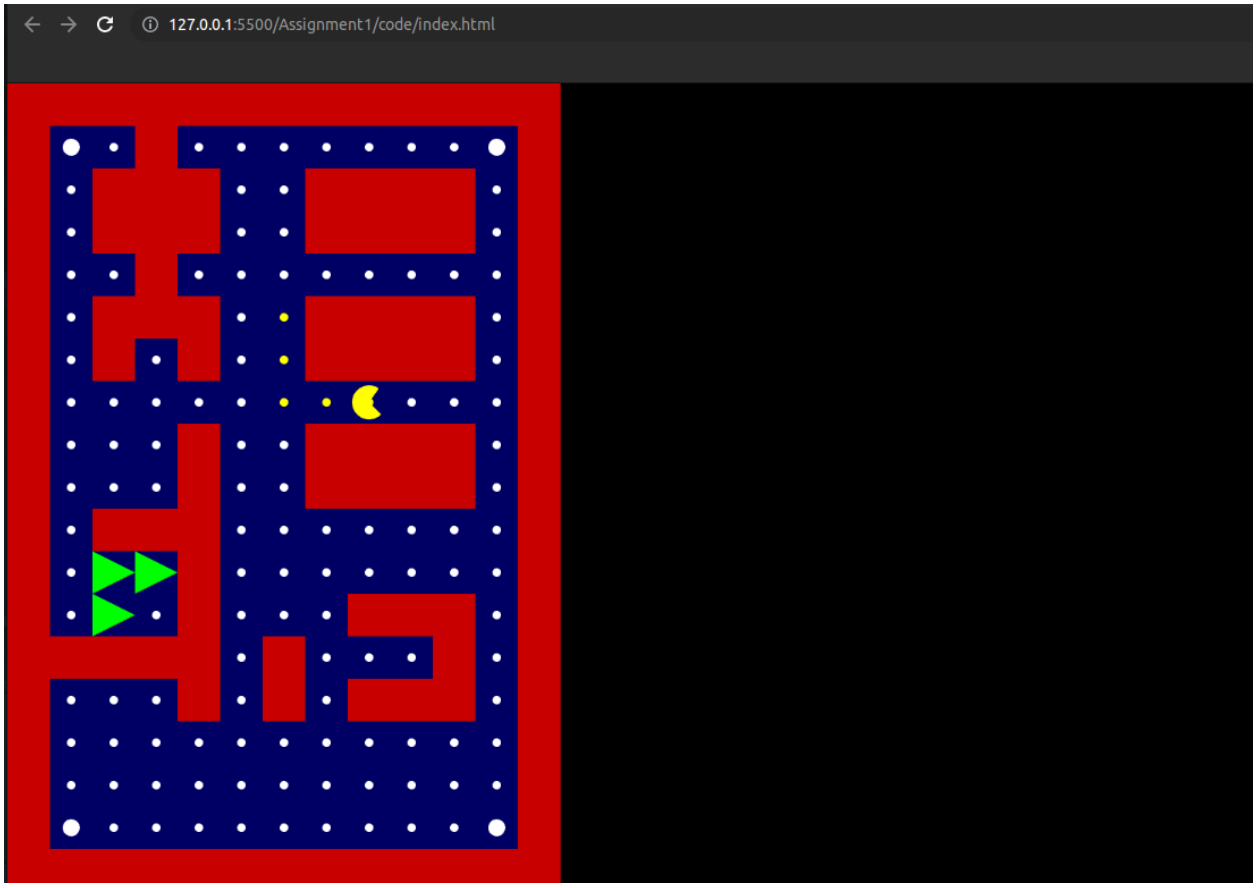
Notice in the above photo how the pellets eaten by pacman are colored yellow. Also the ghosts have all become cyan coloured and the pacman is 1.5 times in size

Whenever a pellet is eaten, handlePelletCol() is called which takes the pellet array and the current position in terms of the grid index of the pacman and it removes the pellet corresponding to that location from both the pellet array and the ‘scene’ then it creates a yellow pellet and replaces that pellet that was removed with this one.

A similar approach is taken with the ghosts.



As seen above, once the Super pellet is eaten, the pacman is going to stop increasing in size, The pellet is marked as eaten and all special effects are disabled.



The orientations of the matrix can be changed and for the most part, The pacman remains where it is. Also the pellets that were eaten remain eaten and the grid collisions also work.

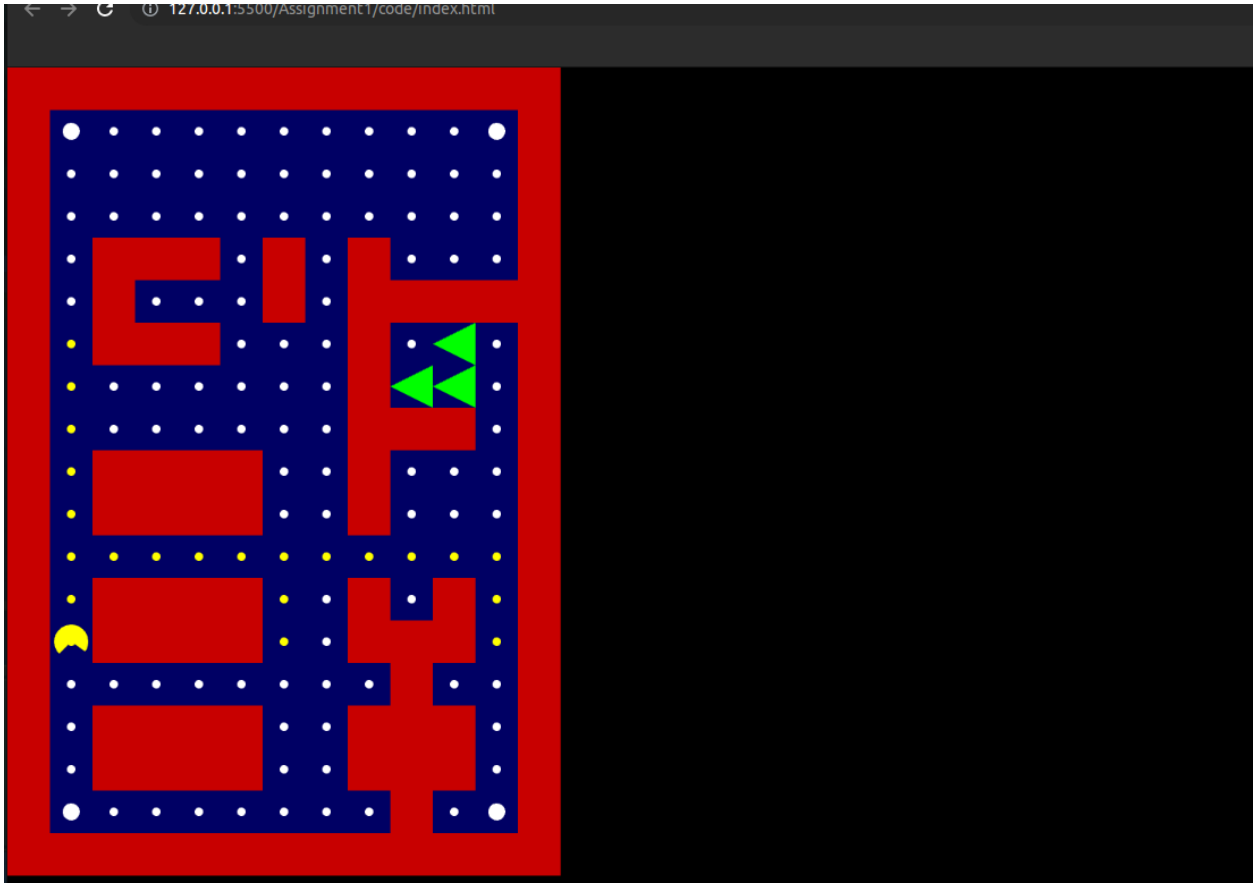
Explaining the maze rotation :

This feat was accomplished by rotating each primitive in the scene by 90 degrees about the origin my coordinate system, that being the top left corner of the screen. So all the primitives once rotated 90 degrees must also be translated to the left by the number of rows in the original grid * chunksize.

Similarly if I were to rotate the grid by 90 degrees about the top left corner, it would be flipped outside the screen. So a maze orientation and angle variable has been maintained in the index.js file and depending upon the current orientation then the grid is rotated by CURRENTANGLE +- 90 depending on direction of rotation based on the input. Then say the grid is rotated by 180, I move all primitives in the x by translating

them in x by the number of columns in the grid * chunksize and in +Y (downwards as explained in the first point about my coordinate system) by the number of rows in the grid times the chunksize.

More screen shots on maze rotation :

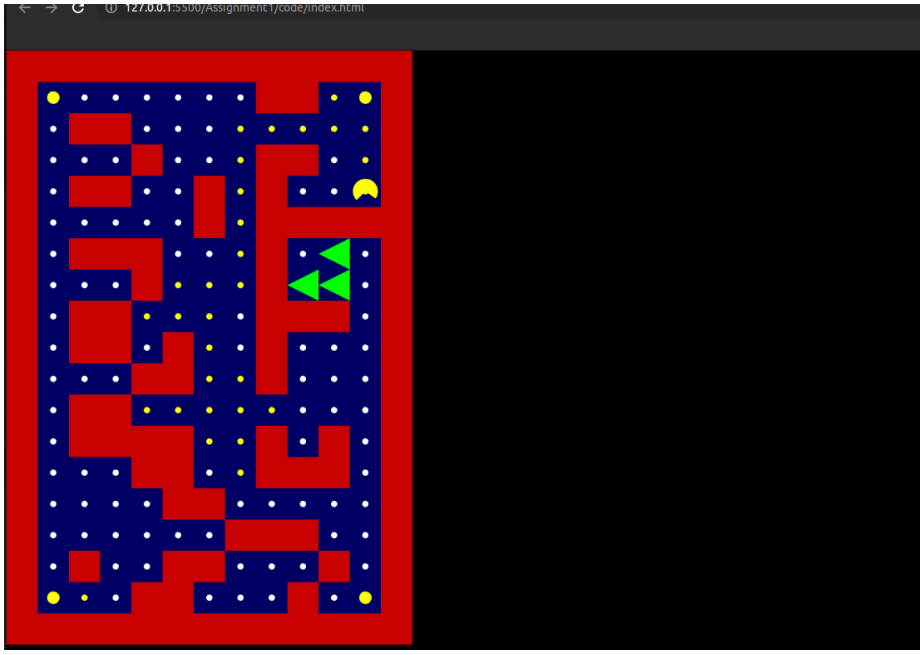


Map Changing :

To change the map, the reinit function is called and the current scene is removed completely and the new scene with the new grid indices is initialised and everything happens based on the grid as already explained so the only major parameter that needs to be changed is the grid that is being passed to the functions.


127.0.0.1:5500/Assignment1/code/index.html


127.0.0.1:5500/Assignment1/code/index.html



Mouse click teleportation of the pacman object :

If you notice the pellets eaten you will see that they are completely separate and not connected by a line of pellets eaten. This was just to demonstrate the mouse click placement of the pacman. This has been done as explained in the teleport() function of the pacman class.

Questions at the bottom of the assignment :

1) Even if the grid rotates, the pacman has it's own separate angles that it is always rotated by. So consider that the pacman is facing right side and the grid is shifted to the downward direction, the pacman still remains oriented in the same direction. Since it has it's own angle property and keeps getting rendered per it's own local information about it's orientation.
2) Although already explained for completeness : To rotate pacman, I had to shift the origin to pacman's center and then do the scale and rotate and then translate the origin back to where it was. But as explained : "Maze rotation was accomplished by rotating each primitive in the scene by 90 degrees about the origin my coordinate system, that being the top left corner of the screen. So all the primitives once rotated 90 degrees must also be translated to the left by the number of rows in the original grid * chunksize. Similarly if I were to rotate the grid by 90 degrees about the top left corner, it would be flipped outside the screen. So a maze orientation and angle variable has been maintained in the index.js file and depending upon the current orientation then the grid is rotated by CURRENTANGLE +- 90 depending on direction of rotation based on the input. Then say the grid is rotated by 180, I move all primitives in the x by translating them in x by the number of columns in the grid * chunksize and in +Y (downwards as explained in the first point about my coordinate system) by the number of rows in the grid times the chunksize."
3) This can be done using 2 ways, one is to just change the chunksize parameter in my code the alternative is that we do a similar approach to the rotategird function. Simply call the scale transform on each primitive and this is done by again translating the origin to the center of each primitive and then scale it and put the origin back. Like each object was rotated similarly we scale each object and then translate it as needed.