Final Project

Intro Computer Graphics – Fall, 2025

Lasair Servilla
University of New Mexico
lservilla@unm.edu

Kaneez Fatima
University of New Mexico
kfatima@unm.edu

December 13, 2025

1 Project Overview

For our final project we choose to create a arcade like experience, with three different game options (one game being available in both 2D and 3D). The two available games being a **Claw Machine** and **Pac-man**. Find deployed webpage at: `https://cs512-finalproject.onrender.com` It should be noted that the page is being hosted by Render.com due to the original code using python flask, and the free version of the service can take at least a minute to load if it has been inactive for too long.

2 Home Page

The home page for the project is simply a game selection. Click the button for the game you wish to play and a new page with the game will load. The home page just uses HTML for everything. The same theme mostly carries through to the game pages to have a consistent feel to them. ChatGPT was used to find the *keyframes* functionality to achieve the animated flicker and pulsing background.

3 Claw Machine

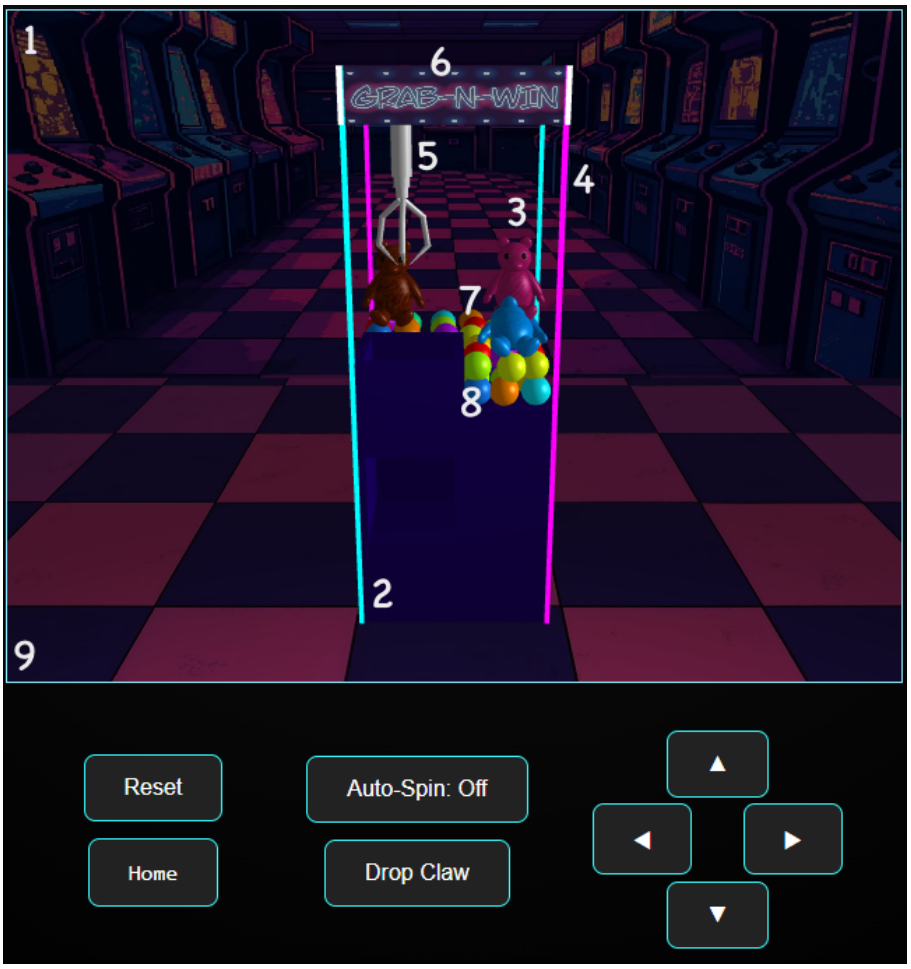

Figure 1: Claw Machine Game Including Controls

3.1 How to use

Similar to real life claw machine games, the virtual game is simple in mechanics but difficult to play successfully. To play use the arrow buttons located on the bottom right of the webpage to move the claw around the machine. When the claw is in position, use the *Drop Claw* button to release the claw and hopefully get lucky. The two sitting teddy bears are the only winnable prizes, and the claw must be perfectly centered over their heads for them to be picked up. The reset button will repose the scene, claw, and any teddy bears that have been won to their original positions. The home button will take you back to the arcade home page. The Auto-spin will rotate the whole scene. Additional scene navigation functionality is available: angle changes can be achieved through clicking and dragging on the scene, zoom in and out using W and S keys, arrow keys on keyboard can change the camera pose. To change the color and position of the balls and teddy bears refresh the page.

The console log providing the positions for the two sitting teddy bears is still available in the submitted version. This can be used to calculate the exact claw movement needed to win a teddy bear prize. From the (x,z) position provided, the claw needs to be in position (x+5, z-5). The claw starts at (0,0) and moves in 0.5 increments. So a teddy pose of (0,-4) would require the claw to be at (5,-9), which would require the back arrow button to be clicked 18 times and the right arrow button to be clicked 10 times.

3.2 Underlying Structure & Code

All sections can be referenced in Figure 1 to see their exact location within the game.

1. **Background**
   The background is a scaled cube with UVs manually added in order to wrap a texture across the faces. The base image was found online and then recreated with ChatGPT (I only did this because the original was already listed as being AI generated and I wanted a better quality). The animated lighting was then hand drawn.

2. **Machine Cabinet**
   The machine cabinet is built up of a bunch of scaled cube primitives. There is nothing too special about them.

3. **Sitting Teddy**
   The sitting teddy is a simple hierarchical model made up of a bunch of sphere primitives that I sized and positioned by hand. The spheres have UV's calculated in order to wrap a fur texture and eye texture on them. The teddy model is just a singular JavaScript object as there was no need to create a entire tree structure when the whole thing moves as a single unit. The drawing function for the teddy simply binds a different texture to the sphere depending on what part is currently getting drawn. Due to scaling needs the teddy has been completely redesigned from the homework it first appeared in. The addition of multiple textures, including different colors of fur is new as well.

4. **Neon Rope Lights**
   The neon rope lights on the side of the game cabinet are just Phong shaded scaled cubes, with adjustments made to make them appear to be a glowing neon. The base of the lights are a single brightly colored cube with a smaller white cube traveling down them to match the background. To achieve the glowing affect, just while the lights were being drawn I increased the ambient light and increased an emissive factor that was set to 0 everywhere else.

5. **Claw**
   The claw is by far the most complex part of the game, being a more complex hierarchical

model employing a state machine to animate the movement. The model itself was completely redesigned for the final project as well as the states. Complexities include having to make movements from pivot points rather than centered on the primitive, and a tree structure with multiple levels and differing branches to correctly set up movement. The state machine for the claw includes movements for the teddy bears as their movement needed to line up with that of the claw.

6. **Machine Front Text**
   The front text is a completely hand drawn video clip that uses the same construction method as the background. Due to there being multiple videos, their frames are all updated in a loop within the *render* function.

7. **Laying Down Teddy**
   The laying down teddy bear is identical to the sitting teddy except for the rotations and translations used to orient the scaled spheres in the correct position.

8. **Colorful Balls**
   The balls are the same from prior homework, I just make an array of ball buffers initiated in all the different colors I wanted them to appear in. The color ball array is then referenced when drawing the balls so as they are all a random color. Working on other things I have realized there is a better way to change the colors so as all the buffers that stay the same are stored for each color, but I already had it set up in such a way and didn't have time to make it better.

9. **Floor**
   The floor is a still video so I can use the same drawing function as the other textured cubes. The original image was created by ChatGPT to match the background and I turned it into a video clip.

3.3 AI Use Statement

Specific areas that AI was directly used is in computing the buffer information for the primitives such as the normals and indices. It was also directly used to help compute the UVs for the spheres. The cube UVs were originally found using AI but then I messed with them to be what I wanted. The transformation functions that were used were generally gotten from AI as well, however some of them were provided to us earlier in the semester and I am no longer sure where each one came from. (The question from the presentation about how I change the shape of the sphere used the *mat4Scale* function, I assume I figured it out using AI but it could have been a provided function that I have just been using throughout the semester, which is the case for all the functions in transformations.js).

Otherwise AI was used like a quick reference for WebGL functions to learn what I needed to correctly set specific things up. For example loading an image uses *gl.texImage2D* and *gl.pixelStorei*, I found those and how to properly use them through ChatGPT. There were also a couple instances where I was having issues with the code that I had written working correctly and I dropped it into ChatGPT to quickly find what was wrong. I know I got help on getting the movement correct for the side claws by having it suggest that I split the angle between planes, so I could still use the same setup as the front claw. It was also very helpful on where I missed something in the state machine, as it's made up of a lot of very similar looking code that only has minor changes (I had to use it for this multiple times). Partway through the project I also had it clean up and organize what I had, as it was becoming a mess and hard to find the sections I was looking for. Since then I have changed a lot, and it is back to being a mess, but it's kinda grouped.

4 PAC-MAN

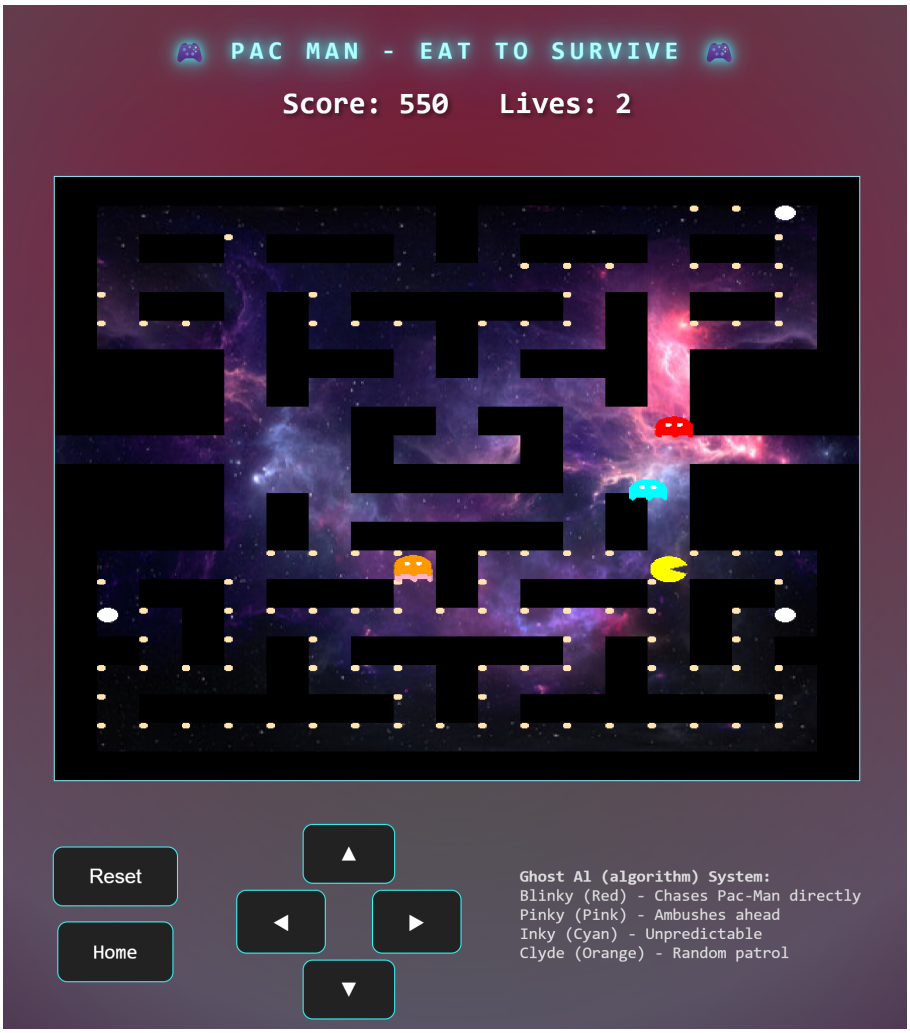

Figure 2: PAC-MAN Game Including Controls

4.1 How to use

The Pac-Man game features intuitive controls designed for accessibility with different ways. Player can control Pac-Man's movement using the arrow keys on their keyboard: the up arrow moves Pac-Man upward, down arrow moves downward, left arrow moves left, and right arrow moves right. For touch-screen users, on-screen directional buttons are provided as an alternative control method. The objective is simple: navigate Pac-Man through the maze to collect all yellow dots (worth 10 points each) while avoiding the four colored ghosts. Special power pellets appear as larger glowing dots in the corners of the maze; consuming these temporarily turns ghosts blue and vulnerable, allowing Pac-Man to eat them for bonus points (200 points per ghost). The game displays real-time score and remaining lives at the top of the screen. Players start with three lives, losing one each time a ghost catches Pac-Man. The game ends when all lives are depleted or when all dots and pellets are collected. In the letter case the player will win. A "Reset" button is available to restart the game at any time. For making it more visually appealing, a galaxy image background is applied.

4.2 Underlying Structure & Code

1. **Background**
   The game implements classic Pac-Man mechanics including maze navigation, ghost algorithm, collision detection, and score tracking.

2. **Game State Structure**
   The game maintains a central state object containing:

   Pac-Man state: Position (x, y), direction, mouth animation

   Ghost array: 4 ghosts with positions, colors, speeds, current state

   Collectibles/pac-man food: Dots and power pellets

   Game metrics: Score, lives, frightened mode timer

   Game status: Game over flag

   Pac-Man is positioned at coordinate (1, 1) in the maze, starting with three lives and a score of zero. Four ghosts are positioned at the center of the maze, each assigned unique colors (red, pink, cyan, orange).

3. **Rendering System**
   Fragment Shader: Renders different shapes (circles, squares, Pac-Man, ghosts) using procedural generation

   Shape Types:

   Shape 0: Circles (dots, pellets)

   Shape 1: Pac-Man with animated mouth

   Shape 2: Squares (maze walls)

   Shape 3: Ghosts with eyes and wavy bottoms

   Full-screen quad rendering is implemented for background images

4. **Matrix for Maze and Collision Detection**
   The maze layout is loaded from the buildMaze() function, which returns a 2D array representing the game maze. The system scans through this array to identify and catalog.

   - 0 = Wall

   - 1 = Dot (pac-man food)

   - 2 = Empty space

   - 3 = Power pellet (activates frightened mode of ghosts)

5. **Main Game Loop Algorithm**
   Pac-man Movement: Pac-Man then moves in his current direction at a speed of 3 units per delta time. The new position is calculated by adding a directional vector (speed multiplied by the direction's unit vector). Before accepting this new position, the canMove() function validates it by converting the coordinates to grid coordinates from the central game state object and checking if that maze cell is passable. If valid, Pac-Man's position updates; otherwise, he stops at the obstacle/wall.

   Ghost Movement: Each ghost's algorithm operates on a decision cycle. Every ghost maintains a timer that increments with each frame's delta time. The algorithm first checks whether the ghost can continue moving forward in its current direction by testing

if the next position ahead is passable (empty space). If the ghost is blocked OR the timer exceeds 0.3 seconds, the ghost must choose a new direction. The system generates a list of all valid directions by testing each of the four cardinal directions (up, down, left, right) to see if movement in that direction would be blocked by a wall.

Collision detection: After Pac-Man moves, its coordinates are calculated to determine which maze cell he occupies. It then iterates through the dots array, checking if any dot's grid position matches Pac-Man's cell. When a match is found, that dot is removed from the array, the score increases by 10 points, and the display updates. The same process applies to power pellets, but eating one triggers additional effects: the score increases by 50 points, the frightened mode activates, and a 6-second timer begins counting down. During frightened mode, all ghosts turn blue and become vulnerable i.e can be eaten by pac-man.

Win Condition Check: After processing pellets, the algorithm checks if both the dots array and pellets array are empty. If all collectibles have been consumed, the game over flag flag pop up, and a victory message displays with the final score and a "Play Again" button.

5 PAC-MAN 3D

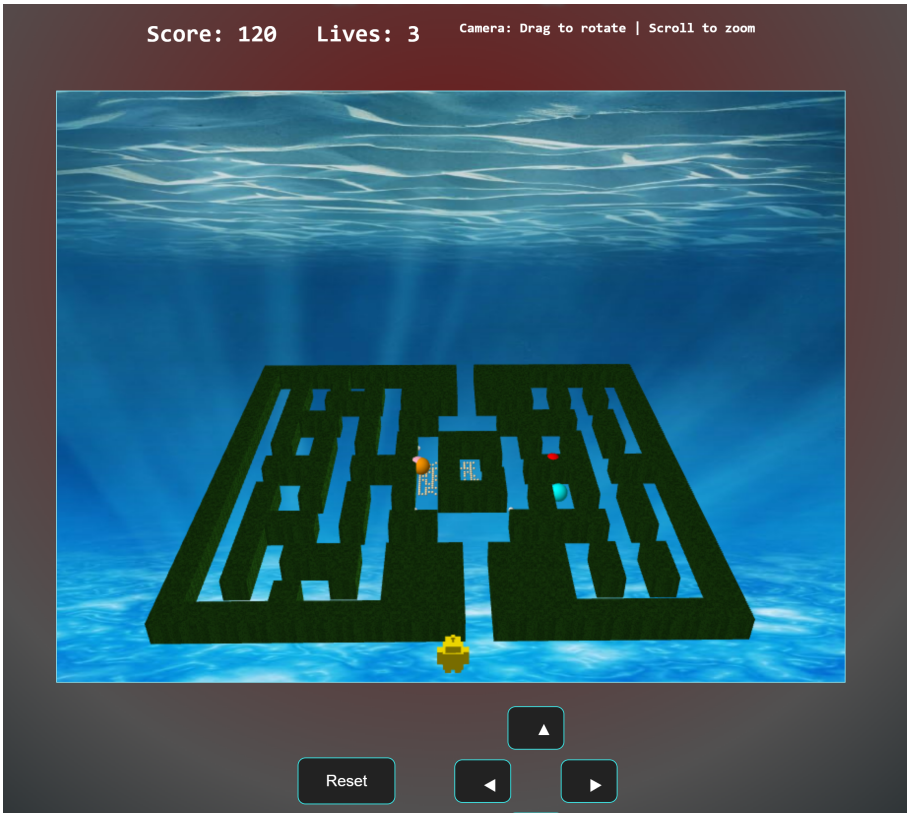

Figure 3: PAC-MAN 3D Game

5.1 Overview

As in the pac-man 2D game, there was not much to add visually appealing features. Therefore I coded pac-man 3D to add more visually appealing features by adding grass texture to the maze walls, changing shape of pac-man into a robot and ghosts shape into circular balls. All the

shapes in this game are primitive defined in primitives.js Due to time constraints, the proper functionality of pac-man game is not implemented.

5.2 Underlying Structure & Code

1. **Background**
   The general concept and overall architechture is same as pac-man 2d, such as maintaining the central state object for tracking the current location coordinates, dot array, pallets array, ghost's position etc. expect I have added camera positions in it to incorporate the 3d view.

   Pac-Man starts at grid coordinate (1, 1) with three lives and zero score. Four ghosts positioned at the maze center with unique colors: red (1,0,0), pink (1,0.7,0.8), cyan (0,1,1), and orange (1,0.6,0).

2. **Dual shader program**

   One vertex shader renders a full screen quad to cover entire view port, and another vertext shader handles the transformation of 3D vertices from model space → world space → view space → clip space and Passes interpolated normals and texture coordinates to fragment shader.

   fragment shader implements Phong lighting model with diffuse and ambient components. It calculates per-pixel lighting based on surface normals

5.3 AI Usage/help

AI was used specifically to the webgl shader's syntax, Used AI to draw the shapes using procedural generation method inside the fragment shader for various types of shapes, i.e. ghosts shape. Got AI's help to get an overview of ghosts working behavior, how to chase the pac-man. Other than that AI was used to get the basic code for background image rendering.

6 Project Breakup

Lasair Servilla:

I created the basic structure for both the code and report, as well as set up the website URL. Otherwise I worked on the Claw Machine and it's associated sections of the report.

Kaneez Fatima:

I worked on pac-man and pac-man 3D, worked on both the game's complete code and logic.