

Pacman - A C++ Raylib Project

By: Tyson, Colten, Em and Morgan

March 6, 2024

This project will use the external C++ library known as RayLib to handle the functions to recreate the arcade game "Pacman", made by game designer Toru Iwatani. The core game loop has the player moving Pacman to collect dots in a maze shaped environment while 4 ghosts' path towards him. The player's goal is to reach a high score by collecting dots, eating ghosts with the power pellets and collecting occasional fruit.

TABLE OF CONTENTS

Introduction/Problem Definition	1
Problem description.....	1
Project objectives.....	1
Development environment	2
Operational environment	2
Requirements and Design Description.....	2
Implementation	3
Installation and Operating Instructions	7
Installation instructions.....	7
Training materials.....	8
Help	
files.....	10
Recommendations for Enhancement.....	12
Bibliography	13
 Appendix A: UML Diagrams / System	
Flowcharts.....	14
Appendix B: Mockups of the User Interface.....	17
Appendix C: Internal Documentation.....	20

Appendix D: Individual student work logs	22
Tyson.....	22
Log book	22
Source code	23
Screenshots of completed online tutorials	24
Printout of unfinished program component.....	25
Progress reports, discussions with teacher	26
 Morgan.....	 27
Log book	27
Source code	30
Screenshots of completed online tutorials	32
Printout of unfinished program component.....	33
Progress reports, discussions with teacher	34
 Colten.....	 35
Log book	35
Source code	36
Screenshots of completed online tutorials	37

Printout of unfinished program component	38
Progress reports, discussions with teacher	39
Em.....	40
Logbook.....	40
Source code	41
Screenshots of completed online tutorials	42
Printout of unfinished program component	43
Progress reports, discussions with teacher	44

1. Introduction/Problem Definition ~~ Morgan

a. Problem description and objectives:

Project Goals consist of level creation, character creation, movement, collision, game items, and an end condition. Each of these goals requires a lot of knowledge and application to make them work together. The user wants to see a project and we are making a fantastic project to fulfill it.

b. Development environment:

Hardware:

CPU: Intel(R) Pentium(R) CPU G2030 @ 3.00GHz

RAM: 8GB

Software:

OS: Windows 10

IDE: Microsoft Visual Studio

Installed Programs: Raylib

c. Operational environment:

Software:

Installed Programs: Raylib, C++

2. Requirements and Design Description (architecture, external and internal functions, interfaces)

Supplementary written description/analysis of all models (UMLs, flowcharts, user interface).



{Pacman Game Loop Flowchart}

Brief descriptions to accompany each model.

New Learning: UML class diagrams: [read](#) and/or [listen](#).

4. Implementation

The program objectives and [end user requirements](#). In other words, what your software is capable of and what the user needs to be able to achieve when using the product. A brief explanation/pseudocode to outline it. You may need separate sections for each member of your groups' work.

The UI of the software needs to print/update the user's score, generate the design of the play area, and apply the character design to the proper characters. The functionality of the software needs to have movement (up, down, left, right but not diagonal), the ghosts must track to Pacman in the most efficient route, pellets and cherries must be generated that give the player score, power pellets must be generated that allows Pacman to eliminate ghosts, collision against walls and enemy ghosts, lives, a lose screen, and generate the level's physics.

5. Installation and Operating Instructions

External User Documentation (User guide with installation instructions, training materials, Help files and FAQs, etc.)

- **Navigate to the downloadable file**
- **After installing the file open/unzip**
- **Locate the {filename}.exe inside of the installed file**
- **Open the executable to open the application**
 - **CONTROLS**
 - **W/A/S/D movement for cardinal direction**
 - **ESC to close the application**

7. Recommendations for Enhancement (What could be improved?)

- Sprite animation for the Pacman character, as well as a more seamless transition to the power pellet mode and scared ghost mode.
- The ghost pathing could be improved. It could be improved by adding something like the A* algorithm to the ghost pathing so the ghost actually paths towards Pacman instead of moving randomly.
- The fruit and pellets have strange properties that could be improved. The collision for when Pacman touches the consumable is based on one corner which could be changed to the center of Pacman. Also, sometimes the fruit's 10 second spawn timer will instantly run out and remove the fruit from the game.
- We could have more maps so that as you progress the game changes and you have to learn new strategies instead of infinitely playing the same game

8. Bibliography (Cite all references used, APA style)

Understanding pac-man ghost behavior. GameInternals. (n.d.).

<http://gameinternals.com/understanding-pac-man-ghost-behavior>

W3Schools online CPP editor. W3Schools Online Web Tutorials. (n.d.).

https://www.w3schools.com/cpp/trycpp.asp?filename=demo_files_read

GitHub. (n.d.). <https://github.com/features/copilot>

Appendix A: UML Diagrams, system flowcharts

Appendix B: Images of the user interface

Appendix C: Internal Documentation (your code with [comments](#), [docstrings](#), block/line comments etc.)

<https://www.programiz.com/python-programming/docstrings>

Appendix D: Individual student work logs.

Eg: A log book or diary of activities, work completed, etc.; Any evidence that the student has contributed positively to the project; e-mailed documents, progress reports, discussions, aha moments etc.

Tyson Scobie:

Tasks:

- Map Generation code
 - Ghost structs and behavior (W/ Colten)
- Creation of private Github Repo for file access
- General debugging and error management

Log:

March 6th - March 22nd < Created the map generation code, given a text file of numbers it parses it and uses it later to draw the map and represent key items such as the dots and fruit.>

```
// Function to read the map txt file and parse through it //
string MapParse(string mapFiletxt) { //Reads the map txt file
    ifstream map(mapFiletxt); //Opens the map txt file

    if (!map.is_open()) {
        cout << "Error: File not found" << endl; //Error message if file is not found
    }

    string currentLine; //Current line in the map txt file
    int currentLineNum = 0; //Current line number in the map txt file
    string mapChars;
    while (getline(map, currentLine)) { //While there are lines in the map txt file
        for (int i = 0; i < currentLine.length(); i++) {
            mapChars = mapChars + currentLine[i]; //Adds the current line to the mapChars string
            if (i == currentLine.length() - 1) {
                currentLineNum++; //Increments the current line number
            }
        }
    }
    return mapChars; //Returns the mapChars string
}

string map = MapParse("map.txt"); //Calls the MapParse function and assigns the returned map string to the map variable
```

{Map Parsing code} Tyson Scobie with assistance from Github Copilot and W3Schools
https://www.w3schools.com/cpp/trycpp.asp?filename=demo_files_read

April 9th-23rd

I was absent during this time as I was in Japan, no progress was made during this time for me.

May 6th - May 30th

I assisted in creating the ghosts' structures, as well working on their movements, collision and behaviors (colliding with Pacman, eaten by Pacman etc). I also created the prototype ghost instantiation using Classes rather than structs.

{See Coltens log for source code}

Morgan Hayes:

Tasks:

- Pacman collision
- Grid-based movement

- UI elements and lives

March 1st - March 30th

I was learning how to use raylib. I learned how to format code and how to draw rectangles, and circles, and learned how to manipulate them using Structs.

April 1st - April 15th

During this time, I was working on changing the movement of Pacman to a grid-based movement so it would work more seamlessly with the collision. I worked with Em to make a movement system, but it did not work with other people's code. Cole ended up using previous ideas, created by Em and I, to make a working movement

system. This is the final movement code:

```
1  // When Pacman leaves the screen, it reappears on the opposite side
2      if (pacman.x > screenWidth) {
3          pacman.x = -20;
4      }
5      if (pacman.x < - pacman.width) {
6          pacman.x = screenWidth;
7      }
8      if (pacman.y > screenHeight) {
9          pacman.y = 0;
10     }
11     if (pacman.y < 0) {
12         pacman.y = screenHeight;
13     }
14
15
16     // When Ghost leave the screen, they reappear on the opposite side
17     for (auto& ghost : ghosts) {
18         if (ghost.rect.x > screenWidth) {
19             ghost.rect.x = -20;
20         }
21         if (ghost.rect.x < -ghost.rect.width) {
22             ghost.rect.x = screenWidth;
23         }
24         if (ghost.rect.y > screenHeight) {
25             ghost.rect.y = 0;
26         }
27         if (ghost.rect.y < 0) {
28             ghost.rect.y = screenHeight;
29         }
30     }
```

April 16th - May 9th

I worked on collision with Em. It was difficult to code because we needed to create a system to allow Pacman to change directions at certain spots as well as make it easy to change directions. We experimented with making the corners circles and having the player move diagonally in the space but it did not work. We also changed numbers for a lot of different objects like Pacman and the walls. We tried to make a buffer system between Pacman and the walls but it did not fix anything and resulted in the same issue.

```

1 // Check for collision with walls
2 for (int i = 0; i < map.length(); i++) {
3     if (map[i] == '1' || map[i] == '9') {
4         Rectangle wall = { static_cast<int>(i % 28 * 20), static_cast<int>(i / 28 * 20), 20, 20 };
5         if (CheckCollisionRecs(pacman, wall)) {
6             //cout << "Collision with wall at x: " << wall.x << " y: " << wall.y << endl;    (this is for debugging)
7             if (pacman.x < wall.x) {
8                 pacman.x -= speed;
9             }
10            if (pacman.x > wall.x) {
11                pacman.x += speed;
12            }
13            if (pacman.y < wall.y) {
14                pacman.y -= speed;
15            }
16            if (pacman.y > wall.y) {
17                pacman.y += speed;
18            }
19        }
20    }
21 }
22 }
23 }

```

May 10th - May 27th

I learned how to add different fonts to the program and apply them to text. I used this font to write the title and used a special font to draw a 'c' as the Pacman symbol in the lives counter. I created rectangle structures to draw a bordered end screen and buttons with text that would restart the game or exit the program. I reset the game by setting the variables to zero, resetting Pacman's position, and recalling the map.

```

85 // Mouse position
86 Vector2 mousePoint = { 0.0f, 0.0f };
87
88 // Define end screen
89 Rectangle screen = { 40, screenHeight / 2 - 200, screenWidth - 80, screenHeight / 2 + 40 };
90 //Rectangle screenBorder = { 20, 20, screenWidth - 40, screenHeight - 40 };
91 Rectangle screenText = { screen.width, screen.height, screen.x - 40, screen.y - 40 };
92
93 Rectangle screenButtonText1 = { screen.width, screen.height, screen.x - 40, screen.y - 40 };
94 Rectangle screenButton1 = { screenButtonText1.x / 4 - 43, screenButtonText1.y - 8, screen.x * 4 + 7, screen.y / 5 };
95
96 Rectangle screenButtonText2 = { screen.width, screen.height, screen.x - 40, screen.y - 40 };
97 Rectangle screenButton2 = { screenButtonText1.x / 2 + 60, screenButtonText1.y - 8, screen.x * 4 + 19, screen.y / 5 };
98
99 // Fonts
100 Font titleFont = LoadFont("fonts/titleFont.ttf");
101 Font livesFont = LoadFont("fonts/livesFont.ttf");
102

```

```

303 // Get mouse position
304 mousePoint = GetMousePosition();
305
306 // Draw UI
307 //void DrawTextEx(Font font, const char* text, Vector2 position, float fontSize, float spacing, Color tint);
308 DrawTextEx(titleFont, "Pacman", { screenWidth / 2 - 60, screenHeight / 60 }, 20, 3, WHITE);
309 DrawTextEx(titleFont, TextFormat("Score: %i", score), { screenWidth / 2 - 100, screenHeight / 20 }, 20, 3, WHITE);
310
311 // Draw lives
312 DrawTextEx(titleFont, TextFormat("Lives: "), { screenWidth / 40, screenHeight - 38 }, 35, 3, WHITE);
313 if (lives == 3) {
314     DrawTextEx(livesFont, "ccc", { screenWidth / 40 + 180, screenHeight - 32 }, 35, 3, YELLOW);
315 }
316 else if (lives == 2) {
317     DrawTextEx(livesFont, "cc", { screenWidth / 40 + 180, screenHeight - 32 }, 35, 3, YELLOW);
318 }
319 else if (lives == 1) {
320     DrawTextEx(livesFont, "c", { screenWidth / 40 + 180, screenHeight - 32 }, 35, 3, YELLOW);
321 }
322 else if (lives == 0) {
323     //Stop pacman
324     velY = 0;
325     velX = 0;
326
327     DrawTextEx(livesFont, "", { screenWidth / 40 + 180, screenHeight - 32 }, 35, 3, YELLOW);
328     DrawRectangleRec(screen, BLACK);
329     DrawRectangleLinesEx(screen, 10, BLUE);
330
331     // Draw the end screen text and buttons
332     DrawTextEx(titleFont, "Game Over", { screenText.x / 2 - 45, screenText.y / 2 }, 20, 3, WHITE);
333
334     DrawRectangleRec(screenButton1, PINK);
335     DrawRectangleLinesEx(screenButton1, 4, BLUE);
336     DrawTextEx(titleFont, "Play Again", { screenButtonText1.x / 4 - 35, screenButtonText1.y }, 15, 3, WHITE);
337
338     DrawRectangleRec(screenButton2, RED);
339     DrawRectangleLinesEx(screenButton2, 4, BLUE);
340     DrawTextEx(titleFont, "End Session", { screenButtonText2.x - 170, screenButtonText2.y }, 15, 3, WHITE);
341

```

```

341
342 if (CheckCollisionPointRec(mousePoint, screenButton1))
343 {
344     if (IsMouseButtonDown(MOUSE_BUTTON_LEFT))
345     {
346
347         score = 0;
348         lives = 3;
349         pacman.x = screenWidth / 2;
350         pacman.y = 26 * 20;
351         dotCounter = 0;
352         fruitTimer = 0;
353
354
355         map = MapParse("map.txt");
356     }
357 }
358
359 if (CheckCollisionPointRec(mousePoint, screenButton2))
360 {
361     if (IsMouseButtonDown(MOUSE_BUTTON_LEFT))
362     {
363         return 0;
364     }
365
366
367 }

```

Colten Lamb:

Tasks:

- Pacman Fruit/Pellets
- Pacman map drawing
- Ghost structs and behavior (W/ Tyson)
 - Pacman collision and movement
 - Texture importing and loading
- Merging all the code together into one file

Log:

April 19th - May 9th

I began working on the Fruit and Pellet systems, as well as finding ways to implement texture into a RayLib project. Originally these systems were made separately from the map and functioned as standalone features, but I thought this was inefficient and recreated the systems to smoothly fit with our projects map parser.

```
1 // Fruit spawning (first fruit spawns after 70 dots are eaten on '2' and despawns after 10 seconds)
2 if (dotCounter >= 70 && map[i] == '2') {
3     //DrawCircle(i % 28 * 20 + 10, i / 28 * 20 + 10, 5, RED);
4     DrawTextureEx(cherryTexture, { static_cast<float>(i % 28 * 20), static_cast<float>(i / 28 * 20) }, 0, 1, WHITE);
5     fruitTimer++;
6     if (fruitTimer == fps * 10) {
7         map[i] = '0';
8         fruitTimer = 0;
9     }
10    // When pacman collides with a fruit, the fruit is removed from the map and checks with score counter
11    if (pacman.x == i % 28 * 20 && pacman.y == i / 28 * 20) {
12        map[i] = '0';
13        score += 100;
14        cout << "Score: " << score << endl;
15    }
16 }
17
18 // Second fruit spawn on '3' after 170 dots are eaten
19 if (dotCounter >= 170 && map[i] == '3') {
20     //DrawCircle(i % 28 * 20 + 10, i / 28 * 20 + 10, 5, PINK);
21     DrawTextureEx(strawberryTexture, { static_cast<float>(i % 28 * 20), static_cast<float>(i / 28 * 20) }, 0, 1, WHITE);
22     fruitTimer++;
23     if (fruitTimer == fps * 10) {
24         map[i] = '0';
25         fruitTimer = 0;
26     }
27    // When pacman collides with a fruit, the fruit is removed from the map and checks with score counter
28    if (pacman.x == i % 28 * 20 && pacman.y == i / 28 * 20) {
29        map[i] = '0';
30        score += 200;
31        cout << "Score: " << score << endl;
32    }
33 }
34 }
```



```
1  // Ghost image loading and resizing
2      Image Inky = LoadImage("inky.png");
3      ImageResize(&Inky, 20, 20);
4      Texture2D inkyTexture = LoadTextureFromImage(Inky);
5
6      Image Blinky = LoadImage("blinky.png");
7      ImageResize(&Blinky, 20, 20);
8      Texture2D blinkyTexture = LoadTextureFromImage(Blinky);
9
10     Image Pinky = LoadImage("pinky.png");
11     ImageResize(&Pinky, 20, 20);
12     Texture2D pinkyTexture = LoadTextureFromImage(Pinky);
13
14     Image DaveChapelle = LoadImage("Dave.png");
15     ImageResize(&DaveChapelle, 20, 20);
16     Texture2D daveChapelleTexture = LoadTextureFromImage(DaveChapelle);
17
18     // Scared Ghost image loading and resizing
19     Image ScaredGhost = LoadImage("ScaredGhost.png");
20     ImageResize(&ScaredGhost, 20, 20);
21     Texture2D scaredGhostTexture = LoadTextureFromImage(ScaredGhost);
22
23     // Fruit image loading and resizing
24     Image Cherry = LoadImage("cherry.png");
25     ImageResize(&Cherry, 25, 25);
26     Texture2D cherryTexture = LoadTextureFromImage(Cherry);
27
28     Image Strawberry = LoadImage("strawberry.png");
29     ImageResize(&Strawberry, 18, 18);
30     Texture2D strawberryTexture = LoadTextureFromImage(Strawberry);
```



```

1 // When pacman collides with a power pellet, the power pellet is removed from the map and checks with score counter
2     if (pacman.x == i % 28 * 20 && pacman.y == i / 28 * 20 && map[i] == '8') {
3         map[i] = '0';
4         score += 50;
5         cout << "Score: " << score << endl;
6         powerPellet = true;
7         frameOffset = frameCounter + fps * 10;
8         dotCounter++;
9         cout << dotCounter << endl;
10
11 // When 244 dots are eaten reset everything but the score
12 if (dotCounter >= 244) {
13     lives = 3;
14     pacman.x = screenWidth / 2;
15     pacman.y = 26 * 20;
16     dotCounter = 0;
17     fruitTimer = 0;
18
19 // Reset ghosts
20 InitGhosts();
21
22 // Reset map
23 map = MapParse("map.txt");
24
25 }
26
27 }

```

```

1 // When pacman collides with a pellet, the pellet is removed from the map and checks with score counter
2     if (pacman.x == i % 28 * 20 && pacman.y == i / 28 * 20 && map[i] == '7') {
3         map[i] = '0';
4         score += 10;
5         dotCounter++;
6         cout << "Score: " << score << endl;
7         cout << "Dots eaten: " << dotCounter << endl;
8
9 // When 244 dots are eaten reset everything but the score
10 if (dotCounter >= 244) {
11     lives = 3;
12     pacman.x = screenWidth / 2;
13     pacman.y = 26 * 20;
14     dotCounter = 0;
15     fruitTimer = 0;
16
17 // Reset ghosts
18 InitGhosts();
19
20 // Reset map
21 map = MapParse("map.txt");
22
23 }
24
25 }

```

May 10th - 17th

During this time, I recreated our game map along with how it was being drawn from the text file being used. After drawing our map, I found a way to properly add collision while allowing for smooth turning at intersections in our map, this was solved by adding a buffer type system to the collision code in the game. I also implemented the constant movement for Pacman even on key release.

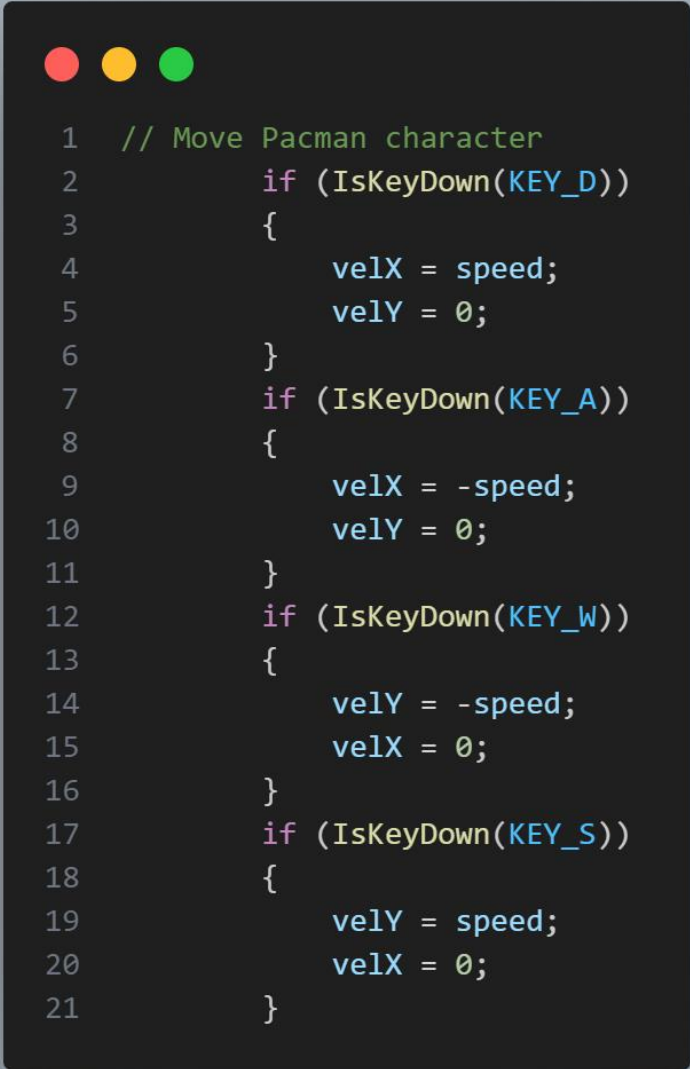
```
1 // When Pacman leaves the screen, it reappears on the opposite side
2     if (pacman.x > screenWidth) {
3         pacman.x = -20;
4     }
5     if (pacman.x < - pacman.width) {
6         pacman.x = screenWidth;
7     }
8     if (pacman.y > screenHeight) {
9         pacman.y = 0;
10    }
11    if (pacman.y < 0) {
12        pacman.y = screenHeight;
13    }
14
15
16    // When Ghost leave the screen, they reappear on the opposite side
17    for (auto& ghost : ghosts) {
18        if (ghost.rect.x > screenWidth) {
19            ghost.rect.x = -20;
20        }
21        if (ghost.rect.x < -ghost.rect.width) {
22            ghost.rect.x = screenWidth;
23        }
24        if (ghost.rect.y > screenHeight) {
25            ghost.rect.y = 0;
26        }
27        if (ghost.rect.y < 0) {
28            ghost.rect.y = screenHeight;
29        }
30    }
```



```
1 // Draw map from map.txt file
2     for (int i = 0; i < map.length(); i++) {
3         if (map[i] == '1') {
4             DrawRectangle(i % 28 * 20, i / 28 * 20, 20, 20, DARKBLUE);
5         }
6
7         // Draw pellets at each '7' in the map and centre them
8         if (map[i] == '7') {
9             DrawCircle(i % 28 * 20 + 10, i / 28 * 20 + 10, 3, WHITE);
10        }
11
12        // Draw Power Pellets at each '8' in the map and centre them
13        if (map[i] == '8') {
14            DrawCircle(i % 28 * 20 + 10, i / 28 * 20 + 10, 5, PINK);
15        }
16
17        // Draw blue squares at each '9' in the map and centre them
18        if (map[i] == '9') {
19            DrawRectangle(i % 28 * 20, i / 28 * 20, 20, 20, BLUE);
20        }
21    }
22 }
23 }
```



```
1 // Check for collision with walls
2     for (int i = 0; i < map.length(); i++) {
3         if (map[i] == '1' || map[i] == '9') {
4             Rectangle wall = { static_cast<int>(i % 28 * 20), static_cast<int>(i / 28 * 20), 20, 20 };
5             if (CheckCollisionRecs(pacman, wall)) {
6                 //cout << "Collision with wall at x: " << wall.x << " y: " << wall.y << endl;    (this is for debugging)
7                 if (pacman.x < wall.x) {
8                     pacman.x -= speed;
9                 }
10                if (pacman.x > wall.x) {
11                    pacman.x += speed;
12                }
13                if (pacman.y < wall.y) {
14                    pacman.y -= speed;
15                }
16                if (pacman.y > wall.y) {
17                    pacman.y += speed;
18                }
19            }
20        }
21    }
22 }
23 }
```



```
1  // Move Pacman character
2      if (IsKeyDown(KEY_D))
3      {
4          velX = speed;
5          velY = 0;
6      }
7      if (IsKeyDown(KEY_A))
8      {
9          velX = -speed;
10         velY = 0;
11     }
12     if (IsKeyDown(KEY_W))
13     {
14         velY = -speed;
15         velX = 0;
16     }
17     if (IsKeyDown(KEY_S))
18     {
19         velY = speed;
20         velX = 0;
21     }
```

May 21st - May 30th

I combined my code with the UI code, and then I began looking into solutions for our ghost pathing as this was our final step in the project. For this I created two functions, one to move the ghost, and the other to initialize the ghost in the main function. The movement function handled the ghost's constant movement and turned when it hit a wall or was at an intersection in the map. To implement this movement into the game without any issues or having to rework the code completely, the turning is handled with a random value (0, 3) to choose what direction it will travel when it reaches certain points on the Ghost map. The Ghost

map is a second game map I created just for the ghost movement. I also added the ability to eat the ghost when a power pellet is collected.

```
1 // When power pellet is true draw the scared ghost texture on the ghosts
2     if (powerPellet == true) {
3         for (const auto& ghost : ghosts) {
4             DrawTexture(ghost.texture, ghost.rect.x, ghost.rect.y, WHITE);
5             DrawTexture(scaredGhostTexture, ghost.rect.x, ghost.rect.y, WHITE);
6         }
7     }
8     else {
9         for (const auto& ghost : ghosts) {
10             DrawTexture(ghost.texture, ghost.rect.x, ghost.rect.y, WHITE);
11         }
12     }
13
```

```
1 // Initialize ghosts
2     InitGhosts();
3     // Random start movement for ghosts
4     for (auto& ghost : ghosts) {
5         int direction = GetRandomValue(0, 3);
6         switch (direction) {
7             case 0: ghost.velX = 2; ghost.velY = 0; break; // Right
8             case 1: ghost.velX = -2; ghost.velY = 0; break; // Left
9         }
10     }
```



```
1  // Pacman collision with ghosts
2      for (const auto& ghost : ghosts) {
3          if (CheckCollisionRecs(pacman, ghost.rect)) {
4              if (powerPellet == true) {
5
6                  // Move only the ghost that collides with pacman
7                  for (auto& ghost : ghosts) {
8                      if (CheckCollisionRecs(pacman, ghost.rect)) {
9                          ghostTimer = 0;
10                         ghost.rect.x = 240;
11                         ghost.rect.y = 360;
12
13                         score += 200;
14                     }
15                 }
16             }
17         }
18     else {
19         lives--;
20         pacman.x = screenWidth / 2;
21         pacman.y = 26 * 20;
22         for (auto& ghost : ghosts) {
23             ghost.rect.x = 240;
24             ghost.rect.y = 360;
25         }
26     }
27 }
28
29 }
```



```
1  // Move and draw ghosts
2      MoveGhosts();
3      for (const auto& ghost : ghosts) {
4          DrawTexture(ghost.texture, ghost.rect.x, ghost.rect.y, WHITE);
5      }
6
7      // Draw scared ghost texture on ghosts when power pellet is active
8      if (powerPellet == true) {
9          for (const auto& ghost : ghosts) {
10             DrawTexture(ghost.texture, ghost.rect.x, ghost.rect.y, WHITE);
11             DrawTexture(scaredGhostTexture, ghost.rect.x, ghost.rect.y, WHITE);
12         }
13     }
14     else {
15         for (const auto& ghost : ghosts) {
16             DrawTexture(ghost.texture, ghost.rect.x, ghost.rect.y, WHITE);
17         }
18     }
```



```
1  // Map for ghosts
2      for (int i = 0; i < mapGhost.length(); i++) {
3          if (mapGhost[i] == '4') {
4              }
5          }
6
7      // Map for ghost box collision (where ghosts spawn)
8      for (int i = 0; i < mapGhost.length(); i++) {
9          if (mapGhost[i] == '5') {
10             }
11     }
```

```

1 void MoveGhosts() {
2     for (auto& ghost : ghosts) {
3         ghost.rect.x += ghost.velX;
4         ghost.rect.y += ghost.velY;
5
6         // Ghost collision with walls
7         for (int i = 0; i < map.length(); i++) {
8             if (map[i] == '1') {
9                 Rectangle wall = { static_cast<int>(i % 28 * 20), static_cast<int>(i / 28 * 20), 20, 20 };
10                if (CheckCollisionRecs(ghost.rect, wall)) {
11                    ghost.rect.x -= ghost.velX;
12                    ghost.rect.y -= ghost.velY;
13
14                    random_device rd;
15
16                    // Use the Mersenne Twister 19937 generator
17                    mt19937 gen(rd());
18
19                    // Define the range [1, 4]
20                    uniform_int_distribution<> dis(0, 3);
21
22                    // Change direction randomly upon collision
23                    int direction = dis(gen);
24                    switch (direction) {
25                        case 0: ghost.velX = 2; ghost.velY = 0; break; // Right
26                        case 1: ghost.velX = -2; ghost.velY = 0; break; // Left
27                        case 2: ghost.velX = 0; ghost.velY = 2; break; // Down
28                        case 3: ghost.velX = 0; ghost.velY = -2; break; // Up
29                    }
30                }
31            }
32        }
33        // Ghost direction change when on mapGhost '4'
34        for (int i = 0; i < mapGhost.length(); i++) {
35            if (mapGhost[i] == '4') {
36                //Rectangle wall = { static_cast<int>(i % 28 * 20), static_cast<int>(i / 28 * 20), 20, 20 };
37                if (ghost.rect.x == i % 28 * 20 && ghost.rect.y == i / 28 * 20 && mapGhost[i] == '4'){
38                    //if (CheckCollisionRecs(ghost.rect, wall))
39
40                    random_device rd;
41
42                    // Use the Mersenne Twister 19937 generator
43                    mt19937 gen(rd());
44
45                    // Define the range [1, 4]
46                    uniform_int_distribution<> dis(0, 3);
47
48                    // Change direction randomly upon collision
49                    int direction = dis(gen);
50                    switch (direction) {
51                        case 0: ghost.velX = 2; ghost.velY = 0; break; // Right
52                        case 1: ghost.velX = -2; ghost.velY = 0; break; // Left
53                        case 2: ghost.velX = 0; ghost.velY = 2; break; // Down
54                        case 3: ghost.velX = 0; ghost.velY = -2; break; // Up
55                    }
56                    mapGhost[i] = '0'
57                }
58                mapGhost[i] = '4';
59            }
60        }
61
62        // When ghost collide with 5 go up
63        for (int i = 0; i < mapGhost.length(); i++) {
64            if (mapGhost[i] == '5') {
65                //Rectangle wall = { static_cast<int>(i % 28 * 20), static_cast<int>(i / 28 * 20), 20, 20 };
66                if (ghost.rect.x == i % 28 * 20 && ghost.rect.y == i / 28 * 20 && mapGhost[i] == '5') {
67                    //if (CheckCollisionRecs(ghost.rect, wall))
68
69                    ghost.velX = 0; ghost.velY = -2; // Up
70                    mapGhost[i] = '0';
71                }
72                mapGhost[i] = '5';
73            }
74        }
75    }
76 }

```




```
1 void InitGhosts() {
2     Image ghostImages[5] = {
3         LoadImage("inky.png"),
4         LoadImage("blinky.png"),
5         LoadImage("pinky.png"),
6         LoadImage("Dave.png"),
7         LoadImage("ScaredGhost.png")
8     };
9
10    for (auto& img : ghostImages) {
11        ImageResize(&img, 20, 20);
12    }
13
14    Texture2D ghostTextures[4];
15    for (int i = 0; i < 4; i++) {
16        ghostTextures[i] = LoadTextureFromImage(ghostImages[i]);
17    }
18
19    // Initialize ghosts with starting positions
20    ghosts = {
21        { { 240, 360, 20, 20 }, 2, 0, ghostTextures[0] },
22        { { 260, 360, 20, 20 }, 2, 0, ghostTextures[1] },
23        { { 280, 360, 20, 20 }, 2, 0, ghostTextures[2] },
24        { { 300, 360, 20, 20 }, 2, 0, ghostTextures[3] }
25    };
26 }
```



```
1  // Struct for the ghosts
2  struct Ghost {
3      Rectangle rect;
4      int velX, velY;
5      Texture2D texture;
6  };
7
8  vector<Ghost> ghosts;
```

Em Cockburn:

Tasks:

- Pacman Movement
- Pacman map drawing
 - Pacman collision
- Map loading and design

Log:

March

Most of my time was spent creating the basic movement and not much else, most of my time was spent on other tasks like my train crash assignment and other quizzes handed out

April & May

I started building the map with Tysons help, I created a text file that could be parsed to represent the map and Tyson made the code that reads the file and turns it into a text string, I also created the code that turns the text string into the map seen on screen and that piece of code is the only part of my code to make it into the final build. After getting the map on the screen we needed a way to interact with it and Morgan was my main help when it came to creating collision. The main issue with collision was navigating the intersections because you had to be in a pixel perfect position to change direction while in one. We tried changing the order of the code and changing the collision to circles but in the end, Cole asked Co-Pilot to create new collision and it worked without issue