

Pacman Debug Report

Summary

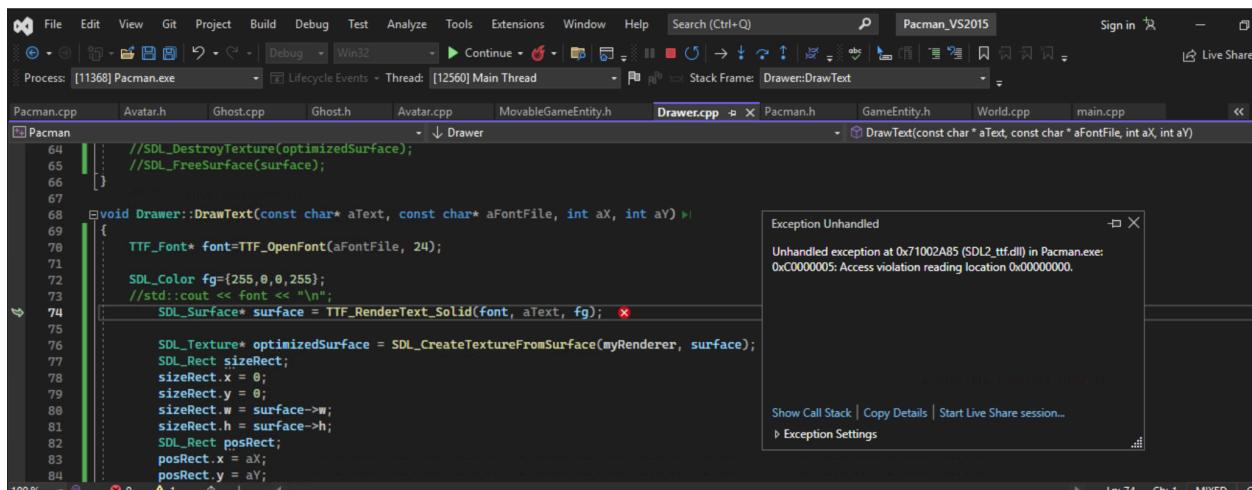
Bugs Fixed / Features added

1. Identifying and Resolving Memory leaks.
2. Added animation to the Packman character.
3. Randomized the movement of characters.
4. Added more Ghost characters.
5. Added fix for the problem of ghosts getting stuck inside the den.

Problem: The game crashes after running for 20-30 seconds.

Solution: Identifying and Resolving Memory Leak

The main challenge involved identifying and resolving a recurring game crash that occurred approximately 20 to 30 seconds after launch. Through rigorous debugging, I successfully identified an unhandled exception within the Drawtext method of the **Drawer.cpp** file. However, the issue persisted despite attempts to resolve it by implementing a try-catch block.



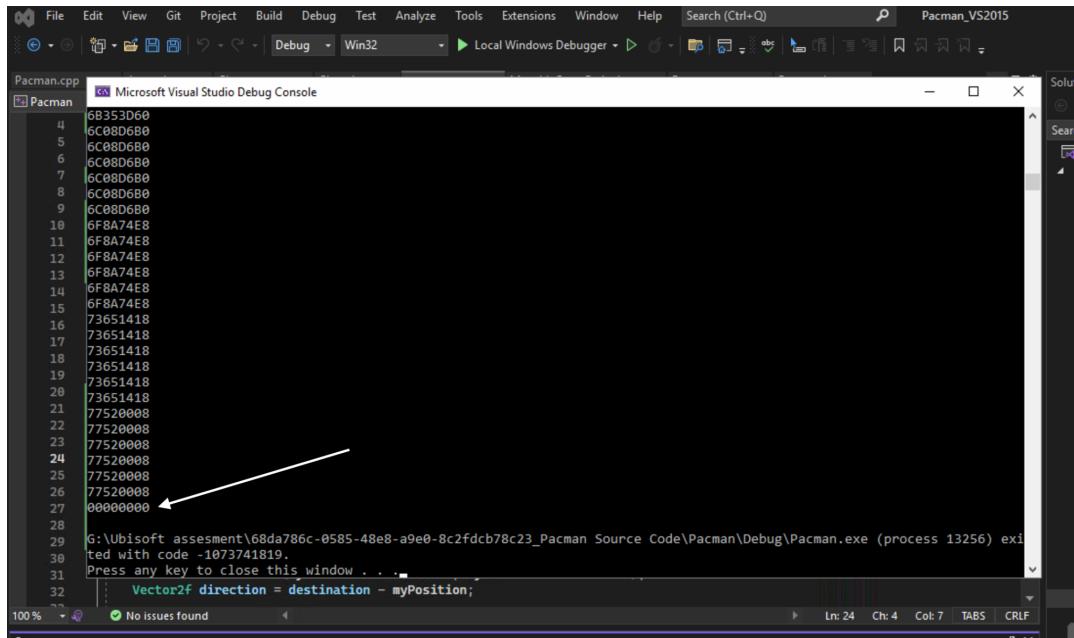
The screenshot shows a Microsoft Visual Studio interface during a debug session. The title bar says "Pacman_VS2015". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help. The toolbar has various icons for debugging and navigation. The status bar at the bottom shows "Exception Unhandled" and "Unhandled exception at 0x71002A85 (SDL2_ttf.dll) in Pacman.exe: 0xC0000005: Access violation reading location 0x00000000." The code editor displays "Drawer.cpp" with the following snippet:

```
//SDL_DestroyTexture(optimizedSurface);
//SDL_FreeSurface(surface);
}

void Drawer::DrawText(const char* aText, const char* aFontFile, int aX, int aY)
{
    TTF_Font* font=TTF_OpenFont(aFontFile, 24);
    SDL_Color fg={255,0,0,255};
    //std::cout << font << "\n";
    SDL_Surface* surface = TTF_RenderText_Solid(font, aText, fg);
    SDL_Texture* optimizedSurface = SDL_CreateTextureFromSurface(myRenderer, surface);
    SDL_Rect sizeRect;
    sizeRect.x = 0;
    sizeRect.y = 0;
    sizeRect.w = surface->w;
    sizeRect.h = surface->h;
    SDL_Rect posRect;
    posRect.x = aX;
    posRect.y = aY;
```

Image 1

During thorough research on StackOverflow, I discovered that the issue stemmed from a null value assignment to a variable, leading to unexpected behavior. To validate this theory, I implemented code to print the value of the "font" parameter and determine whether it indeed contained a null value.



The screenshot shows the Microsoft Visual Studio Debug Console window titled "Pacman". The console displays a series of memory dump values starting from address 4, followed by a series of 0x00000000 values. An arrow points to the first 0x00000000 value at address 27. The console output ends with a crash message: "G:\Ubisoft assesment\68da786c-0585-48e8-a9e0-8c2fdcb78c23_Pacman Source Code\Pacman\Debug\Pacman.exe (process 13256) exited with code -1073741819. Press any key to close this window . . ." The status bar at the bottom indicates "No issues Found".

Image 2

Upon examining the final value of the "font" variable, it became evident that the program was crashing due to a null pointer exception. However, since the Drawtext method functioned properly during the initial 20 seconds, it indicated a potential **memory leak** within the code. A closer inspection revealed that the Draw method lacked the necessary DestroyTexture and destroyFreesurface calls, resulting in the memory leak. Once these lines of code were added (as referenced in the image below), the game stopped crashing, resolving the issue successfully.

```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) Pacman_VS2019
Pacman.cpp Avatar.h Ghost.cpp Ghost.h Avatar.cpp MovableGameEntity.h GameEntity.h Drawer.cpp * Pacman
Pacman
39  []
40
41 void Drawer::Draw(const char* anImage, int aCellX, int aCellY)
42 {
43     SDL_Surface* surface = IMG_Load( anImage );
44
45     if (!surface)
46         return;
47
48     SDL_Texture* optimizedSurface = SDL_CreateTextureFromSurface(myRenderer, surface);
49
50     SDL_Rect sizeRect;
51     sizeRect.x = 0 ;
52     sizeRect.y = 0 ;
53     sizeRect.w = surface->w ;
54     sizeRect.h = surface->h ;
55
56     SDL_Rect posRect ;
57     posRect.x = aCellX;
58     posRect.y = aCellY;
59     posRect.w = sizeRect.w;
60     posRect.h = sizeRect.h;
61
62     SDL_RenderCopy(myRenderer, optimizedSurface, &sizeRect, &posRect);
63     //Freeing space
64     SDL_DestroyTexture(optimizedSurface);
65     SDL_FreeSurface(surface);
66 }
```

Image 3

Problem: No animation for the Packman character

Solved: Added animation to the Packman character

To enable animation for the Pacman character, it was necessary to track its orientation. To accomplish this, I added a currentOrientation variable of type string to the Avatar.h file. This variable stored values such as "left", "right", "up", and "down" to indicate the Pacman's direction. Additionally, I implemented an Animate method in the Avatar.h file. This method accepted a string parameter named position and was responsible for animating the Pacman character based on the provided position.

```
53 //METHOD TO ANIMATE PACMAN
54 void Avatar::Animate(std::string position) {
55     if (shouldAnimate) {
56         if (position == "left") {
57             if (open) {
58                 open = false;
59                 avatarImage = "closed_left_32.png";
60             }
61             else {
62                 open = true;
63                 avatarImage = "open_left_32.png";
64             }
65         }
66         else if (position == "right") {
67             if (open) {
68                 open = false;
69                 avatarImage = "closed_right_32.png";
70             }
71             else {
72                 open = true;
73                 avatarImage = "open_right_32.png";
74             }
75         }
76     }
77     else if (position == "up") {
78         if (open) {
79             open = false;
80             avatarImage = "closed_up_32.png";
81         }
82         else {
83             open = true;
84         }
85     }
86 }
```

Image 4

This method would animate the Packman using a simple boolean logic as shown above.

```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) Pacman_VS2015

Pacman.cpp -> Avatar.h Ghost.cpp Ghost.h Avatar.cpp* MovableGameEntity.h GameEntity.h Drawer.cpp*
Pacman -> Pacman
Pacman.h MoveAvatar()
```

```
343
344     return true;
345 }
346
347 void Pacman::MoveAvatar()
348 {
349     int nextTileX = myAvatar->GetCurrentTileX() + myNextMovement.myX;
350     int nextTileY = myAvatar->GetCurrentTileY() + myNextMovement.myY;
351     //std::cout << std::to_string(myAvatar->GetCurrentTileX()) + " CurrentX\n";
352     //std::cout << std::to_string(myAvatar->GetCurrentTileY()) + " CurrentY\n";
353
354
355     if (myAvatar->IsAtDestination())
356     {
357         if (myWorld->TileIsValid(nextTileX, nextTileY))
358         {
359
360             myAvatar->SetNextTile(nextTileX, nextTileY);
361             myAvatar->setAnimation(true);
362             if (myNextMovement.myX == 1.f) {
363                 myAvatar->setOrientation("right"); //Setting up the orientation of Packman based on user inputs
364                 myCurrentMovement = Vector2f(1.f, 0.f);
365             }
366             else if (myNextMovement.myX == -1.f) {
367                 myAvatar->setOrientation("left");
368                 myCurrentMovement = Vector2f(-1.f, 0.f);
369             }
370             else if (myNextMovement.myY == 1.f) {
371                 myAvatar->setOrientation("down");
372                 myCurrentMovement = Vector2f(0.f, -1.f);
373             }
374         }
375     }
376 }
```

Image 5

The orientation of the Packman is being changed based on the user input also it is important to note that the value of orientation is not changed until the next tile is valid.

Problem: Pacman Stops at Invalid NextTile Value

Solved: Added a new variable to keep check

If we press the arrow up button and the path above is invalid the Packman stops moving and gets stuck.

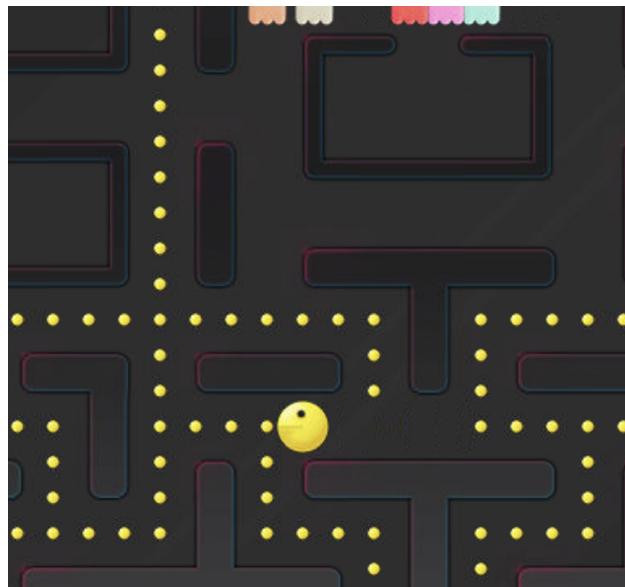


Image 6

To solve this problem I created a new **Vector2 variable myCurrentMovement** and its value gets updated only when the **nextTile** is valid. And if the tile is invalid then I assign this value to **myNextMovement** variable so that the Packman keeps on moving in the previous direction that it had.

Problem: Animation Continues Even After Pacman Reaches a Dead End

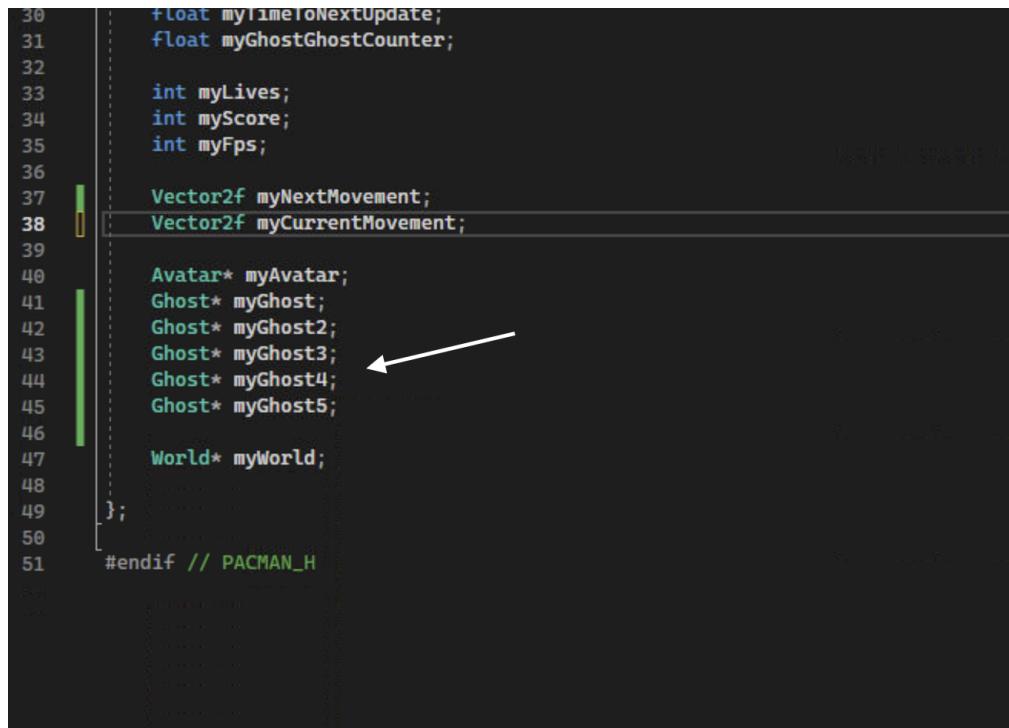
Solved: Added a boolean check

To achieve this feature I simply put a boolean check **shouldAnimate** inside the **Avatar.cpp** file. The value of this variable is changed from **MoveAvatar()** function if the next tile is valid.

Problem: The game had only one Ghost character

Solved: Added 5 Ghost characters for a better gameplay

To add the Ghost characters I declared the characters in the **Packman.h** file. Since all the ghost characters had an almost similar function I just copied the ghost action code for all the other ghosts.



```
30     float myTimeToNextUpdate;
31     float myGhostGhostCounter;
32
33     int myLives;
34     int myScore;
35     int myFps;
36
37     Vector2f myNextMovement;
38     Vector2f myCurrentMovement; ←
39
40     Avatar* myAvatar;
41     Ghost* myGhost;
42     Ghost* myGhost2;
43     Ghost* myGhost3;
44     Ghost* myGhost4;
45     Ghost* myGhost5;
46
47     World* myWorld;
48
49 };
50
51 #endif // PACMAN_H
```

Image 7

Problem: Abnormal Ghost Behavior After the Game Reset

The ghost character, as well as the Packman character's **next tile**, was not being reset so as soon as Pacman dies the ghost directly moves to the next tile which it had set before death.

```

if (myGhostGhostCounter <= 0.f)
{
    myLives--;

    myAvatar->SetPosition(Vector2f(13 * 22, 22 * 22));
    // reset direction and destination values
    myAvatar->setOrientation("left");
    myAvatar->setAnimation(true);
    //myAvatar->SetNextTile()
    myAvatar->SetNextTile(myAvatar->GetPosition().myX / 22, myAvatar->GetPosition().myY / 22);
    myCurrentMovement = Vector2f(-1.f, 0.f);
    myGhost->SetPosition(Vector2f(13 * 22, 13 * 22));
    myGhost->SetNextTile(myGhost->GetPosition().myX / 22, myGhost->GetPosition().myY / 22);

    myGhost2->SetPosition(Vector2f(13 * 22, 13 * 22));
    myGhost2->SetNextTile(myGhost->GetPosition().myX / 22, myGhost->GetPosition().myY / 22);

    myGhost3->SetPosition(Vector2f(13 * 22, 13 * 22));
    myGhost3->SetNextTile(myGhost->GetPosition().myX / 22, myGhost->GetPosition().myY / 22);

    myGhost4->SetPosition(Vector2f(13 * 22, 13 * 22));
    myGhost4->SetNextTile(myGhost->GetPosition().myX / 22, myGhost->GetPosition().myY / 22);

    myGhost5->SetPosition(Vector2f(13 * 22, 13 * 22));
    myGhost5->SetNextTile(myGhost->GetPosition().myX / 22, myGhost->GetPosition().myY / 22);

}
else if (myGhost4->myIsClaimableFlag && !myGhost4->myIsDeadFlag)
{

```

Image 8

To counter this problem, I changed the value of the **next tile** for all the elements once the Packman dies.

Problem: The ghosts in the game were getting stuck inside the den.

Solved: Added 2 points inside the den area. When the ghosts intercept these points their movement direction changes in an upward direction.

When the game starts some of the ghosts fail to move outside the den, to counter this problem I copied the tile number of the circled tiles and whenever a ghost's next tile is this tile I change the direction of the ghost to move upwards, this way they are able to move out of the den.

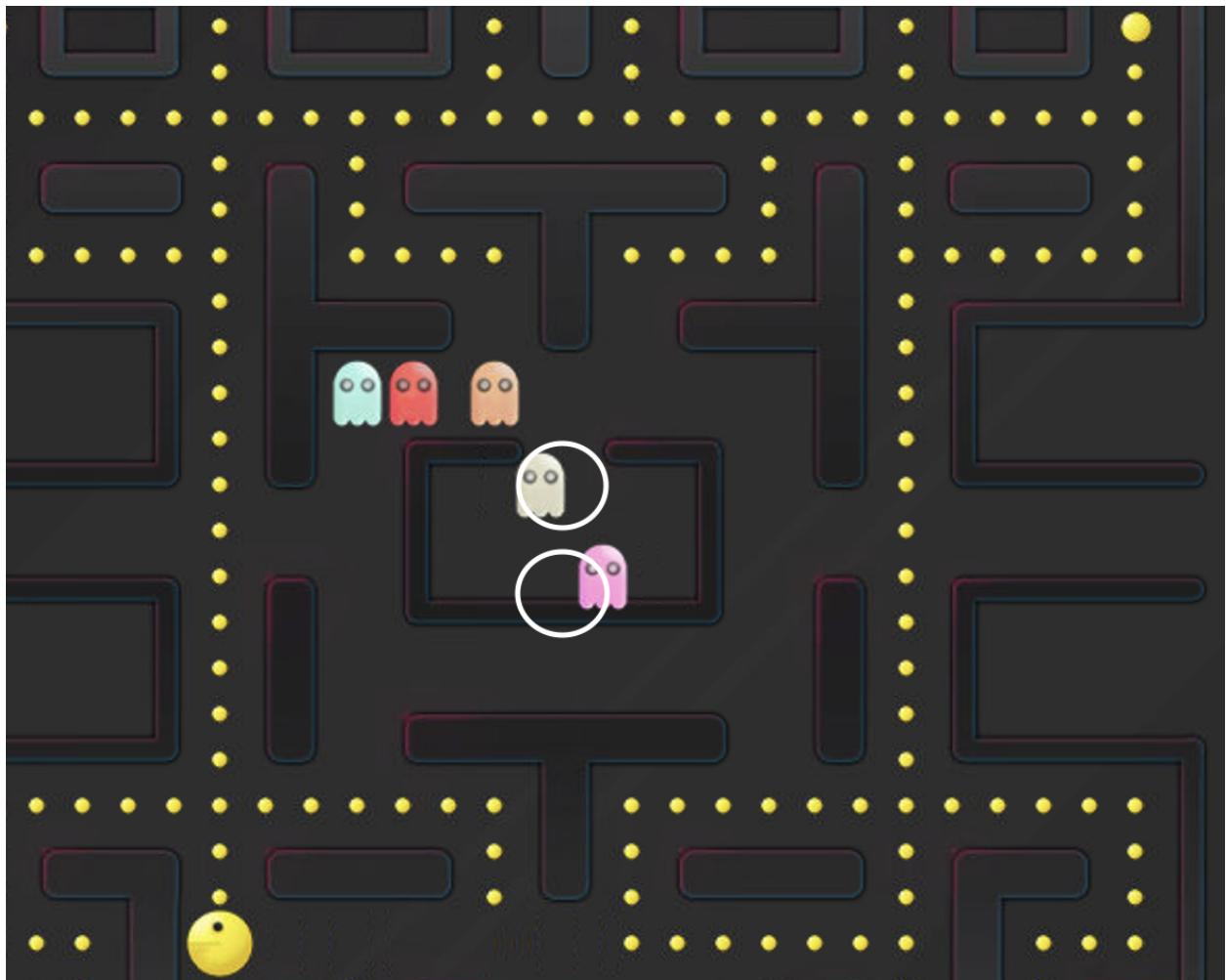
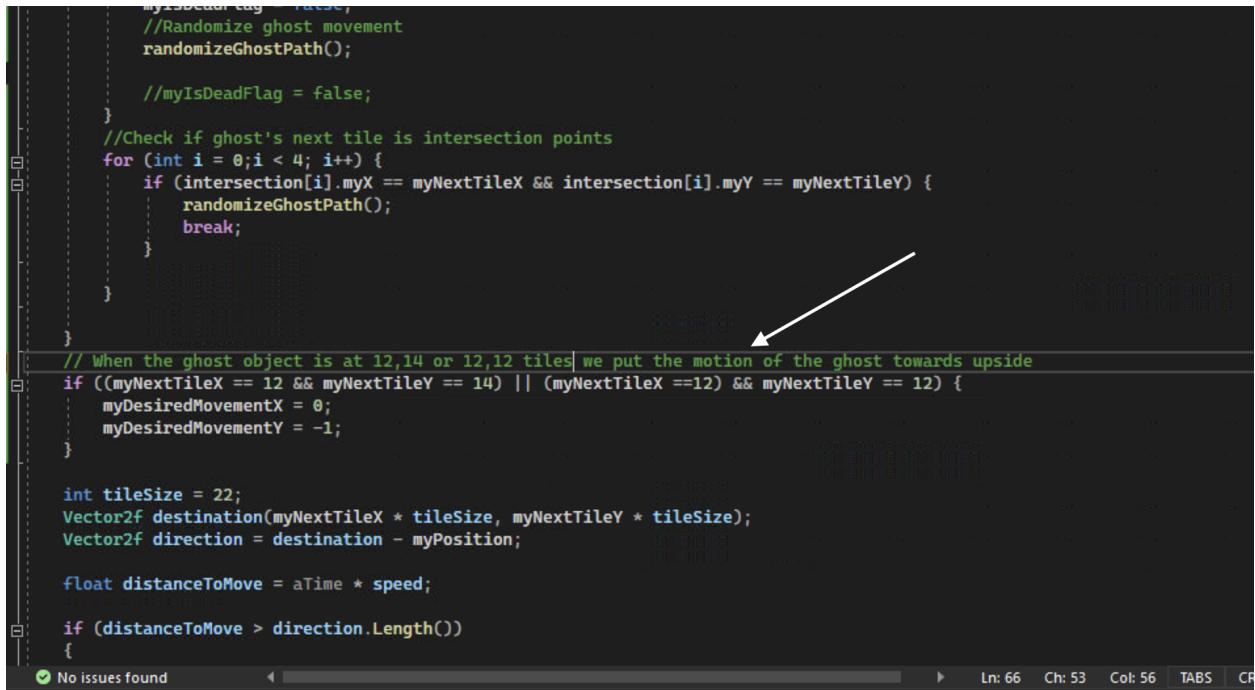


Image 9



```
myIsDeadFlag = false;
//Randomize ghost movement
randomizeGhostPath();

//myIsDeadFlag = false;
}

//Check if ghost's next tile is intersection points
for (int i = 0; i < 4; i++) {
    if (intersection[i].myX == myNextTileX && intersection[i].myY == myNextTileY) {
        randomizeGhostPath();
        break;
    }
}

}

// When the ghost object is at 12,14 or 12,12 tiles we put the motion of the ghost towards upside
if ((myNextTileX == 12 && myNextTileY == 14) || (myNextTileX == 12) && myNextTileY == 12) {
    myDesiredMovementX = 0;
    myDesiredMovementY = -1;
}

int tileSize = 22;
Vector2f destination(myNextTileX * tileSize, myNextTileY * tileSize);
Vector2f direction = destination - myPosition;

float distanceToMove = aTime * speed;

if (distanceToMove > direction.Length())
{
```

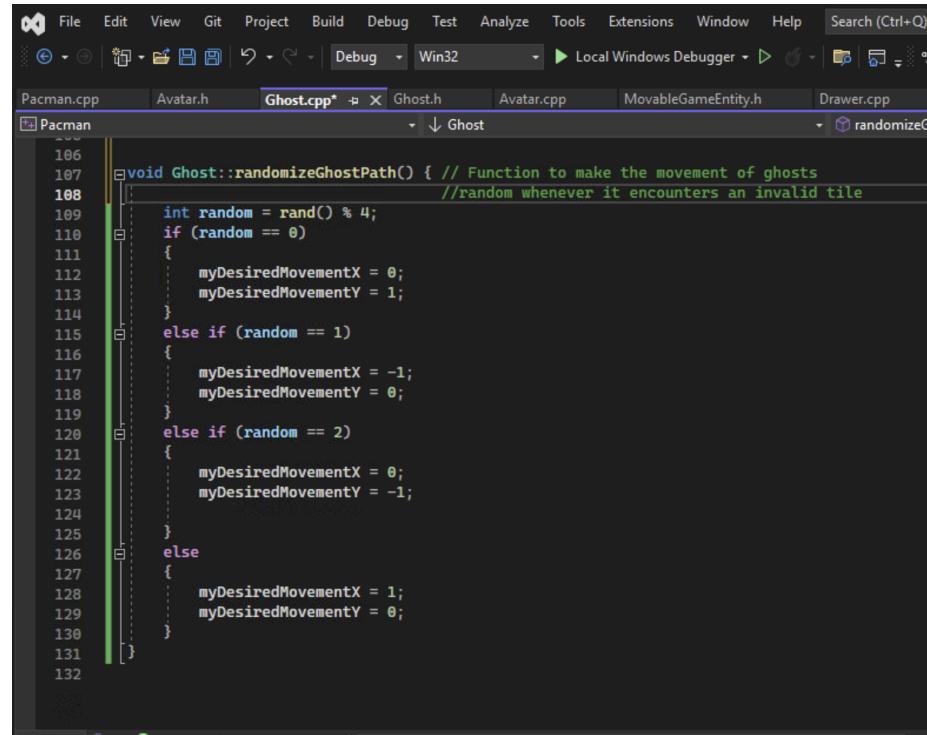
Image 10

These changes are made inside the update function of Ghost.cpp file.

Problem: The movement pattern of the ghosts in the game became predictable and repetitive, leading to a less engaging gameplay experience.

Solution: To enhance the gameplay and add more variety to the ghost behavior, the ghost movement was randomized. By introducing randomization in their movement decisions, the ghosts exhibited less predictable patterns, making it more challenging for players to anticipate their movements.

Earlier the game had a similar pattern of movement for all the ghost's characters. I changed the code to make it feel more randomized and make the gameplay fun. So I implemented a way to make sure that whenever any ghosts are encountered with a blocking invalid tile, I **randomize the outcome of this event**.



```
106 void Ghost::randomizeGhostPath() { // Function to make the movement of ghosts
107                                         //random whenever it encounters an invalid tile
108     int random = rand() % 4;
109     if (random == 0)
110     {
111         myDesiredMovementX = 0;
112         myDesiredMovementY = 1;
113     }
114     else if (random == 1)
115     {
116         myDesiredMovementX = -1;
117         myDesiredMovementY = 0;
118     }
119     else if (random == 2)
120     {
121         myDesiredMovementX = 0;
122         myDesiredMovementY = -1;
123     }
124     else
125     {
126         myDesiredMovementX = 1;
127         myDesiredMovementY = 0;
128     }
129 }
130 }
131 }
132 }
```

Image 11

This method is called from the [Update function of Ghost.cpp](#) whenever the next tile is invalid.

Problem: Ghost movement doesn't change at the intersection points.

Solved: Making the Ghost movement more randomized by making them change direction at intersection points

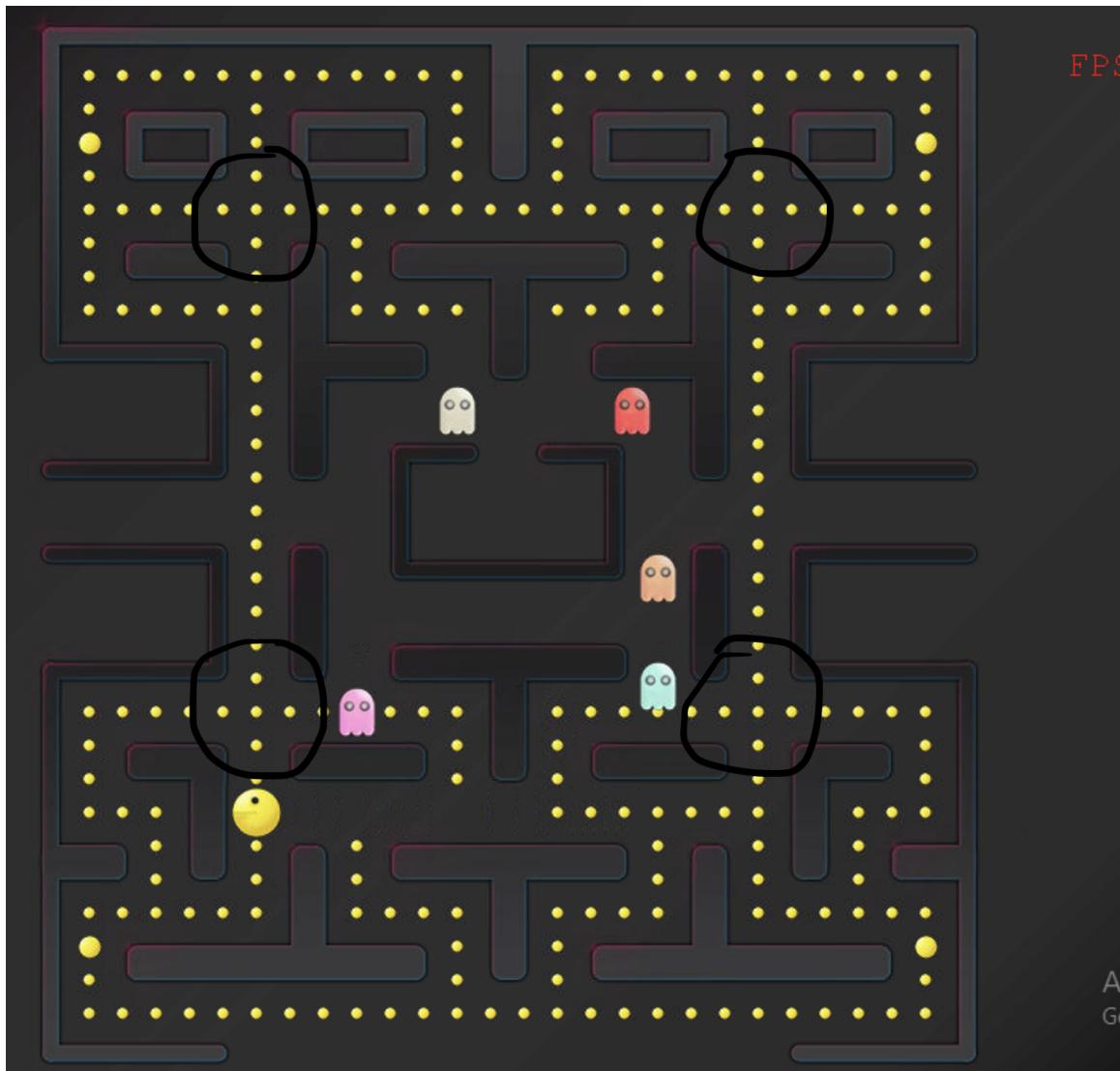


Image 12

Earlier the Ghost character kept their movement the same in the above-mentioned 4 points. Firstly I tried to find the tile value of these four intersection points by printing the value of **Pacman's currentTileX and currentTileY** value as I could move the Pacman easily to extract the tile values of these points. Once I found out the tile value of these four intersection points, I created a constant **Vector2 array in Ghost.h file** and assigned these four values.

```

20
21     void SetImage(const char* anImage);
22
23     void Die(World* aWorld);
24
25     void Draw(Drawer* aDrawer);
26     void setDesiredMovementX(int xVal) { myDesiredMovementX = xVal; }
27     void setDesiredMovementY(int yVal) { myDesiredMovementY = yVal; }
28     void randomizeGhostPath();
29
30 protected:
31
32     int myDesiredMovementX;
33     int myDesiredMovementY;
34
35     std::list<PathmapTile*> myPath;
36     //Adding the four intersection points in a constant vector array
37     const Vector2f intersection[4] = {Vector2f(5,19), Vector2f(5,4), Vector2f(20,4), Vector2f(20,19)};
38
39 };
40
41 #endif // GHOST_H

```

Image 13

Inside the **update** function of **Ghost.cpp** I initially checked if **currentTile** was equal to the values on the array or not but once I ran the code it seemed like putting a check on the **currentTile** would make the code not work because by the time the code would run the ghost element would have already crossed the intersection, so instead I put a check on **myNextTile** and the code started working properly.

```

48
49     else
50     {
51         myIsDeadFlag = false;
52         //Randomize ghost movement
53         randomizeGhostPath();
54
55         //myIsDeadFlag = false;
56     }
57     //Check if ghost's next tile is intersection points
58     for (int i = 0; i < 4; i++) {
59         if (intersection[i].myX == myNextTileX && intersection[i].myY == myNextTileY) {
60             randomizeGhostPath();
61             break;
62         }
63     }
64
65 }
66 // When the ghost object is at 13,13 tile we put the motion of the ghost towards upside
67 if (myCurrentTileX == 12 && myCurrentTileY == 14) {
68     myDesiredMovementX = 0;
69     myDesiredMovementY = -1;

```

Image 14

