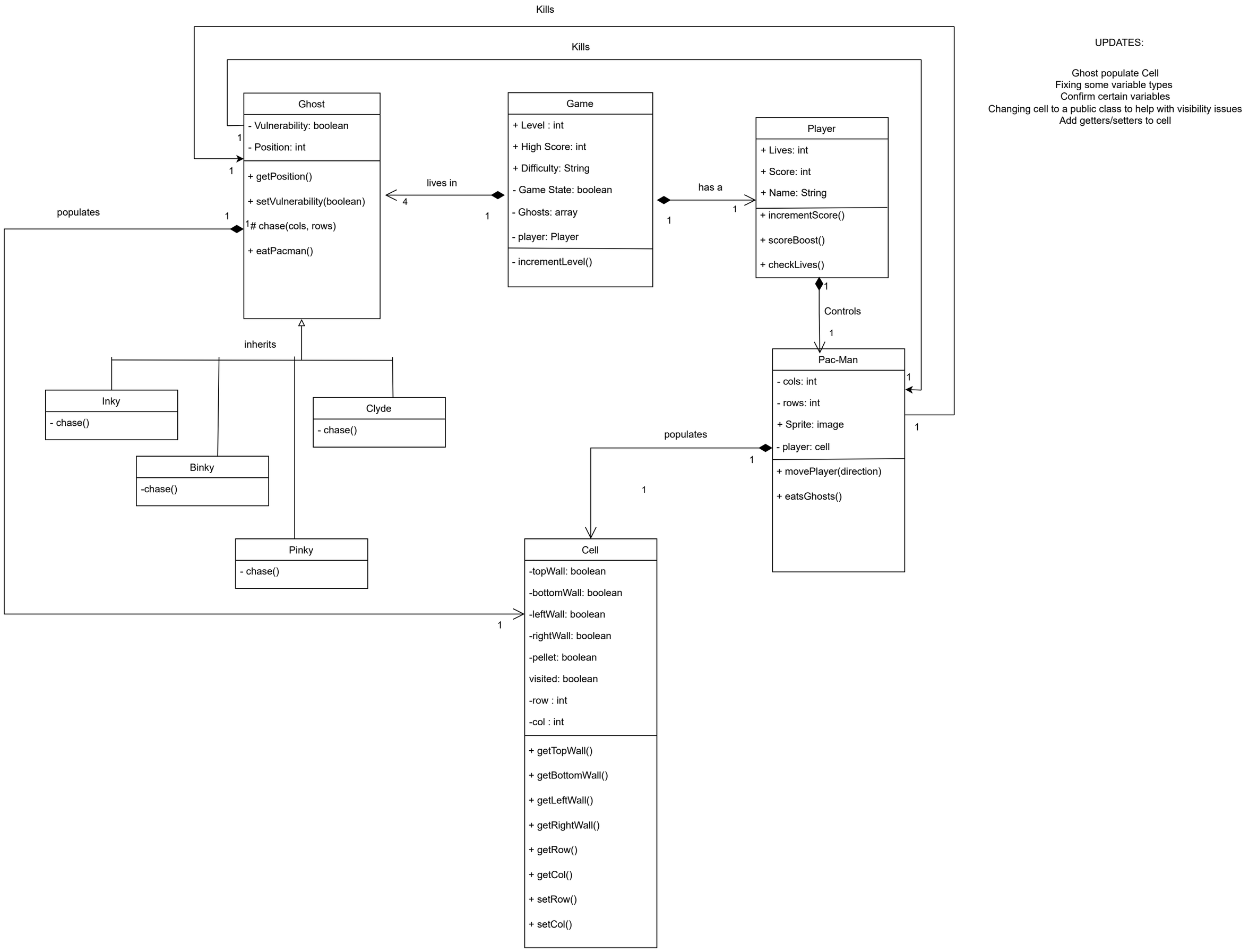
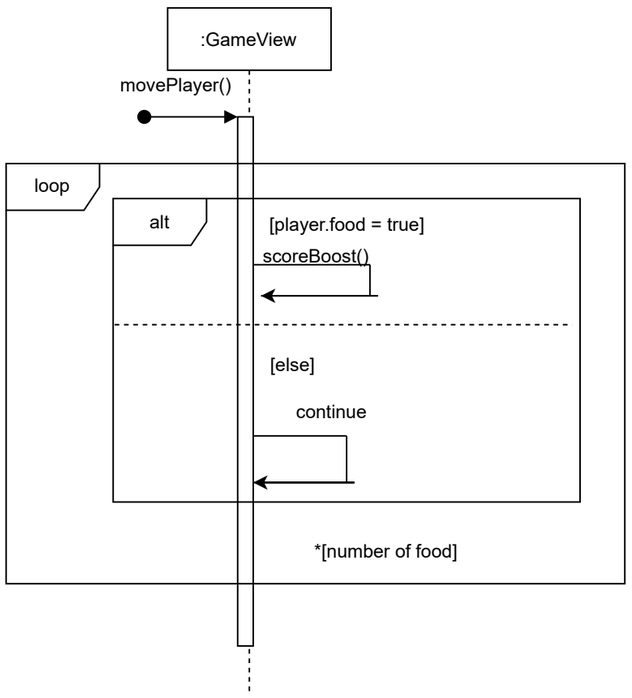
Sprint 4 DCD

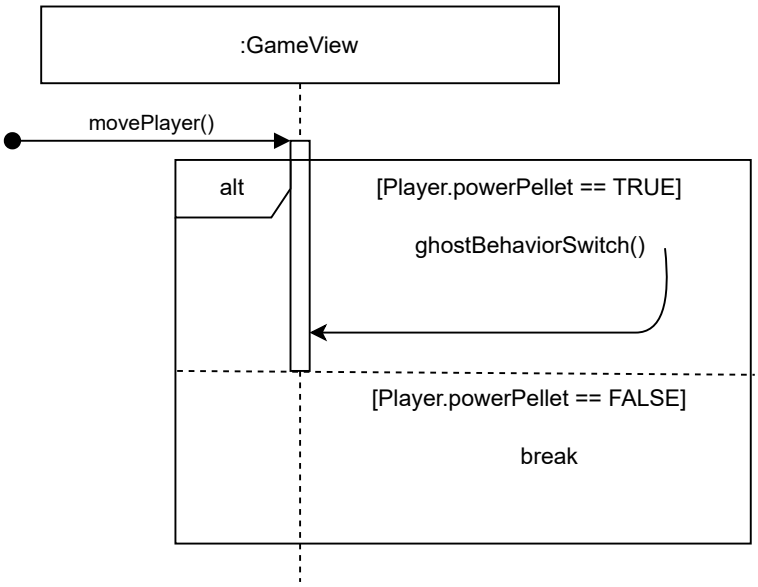


UPDATES:

Ghost populate Cell
Fixing some variable types
Confirm certain variables
Changing cell to a public class to help with visibility issues
Add getters/setters to cell



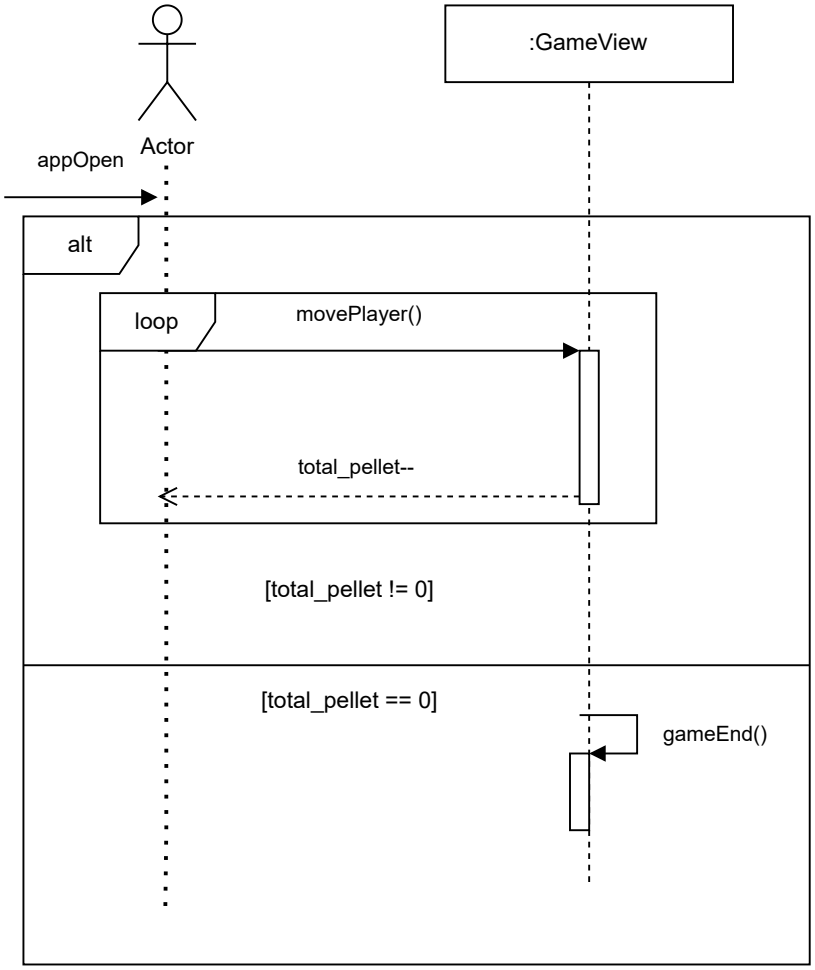
Aditi's Use Case: As a player, I want to collect food(fruit) so I can gain bonus points.



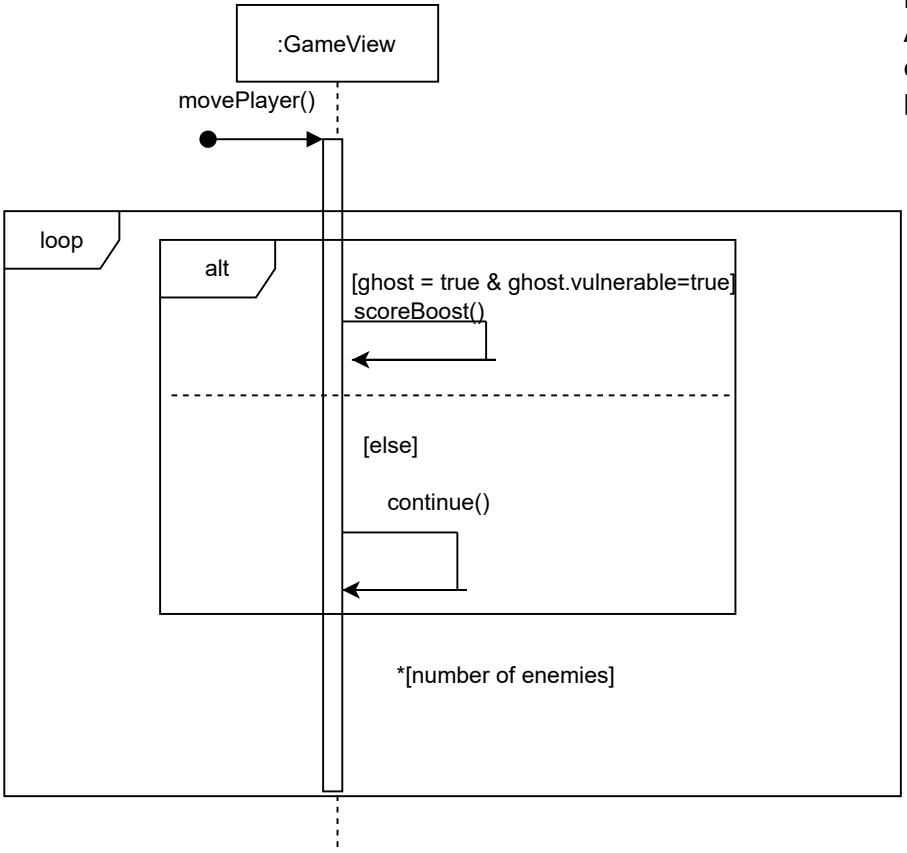
Adena's Use Case: As a player, I want to be able to eat power pellets so I can attack ghosts.

Caelan (Ace) Scales

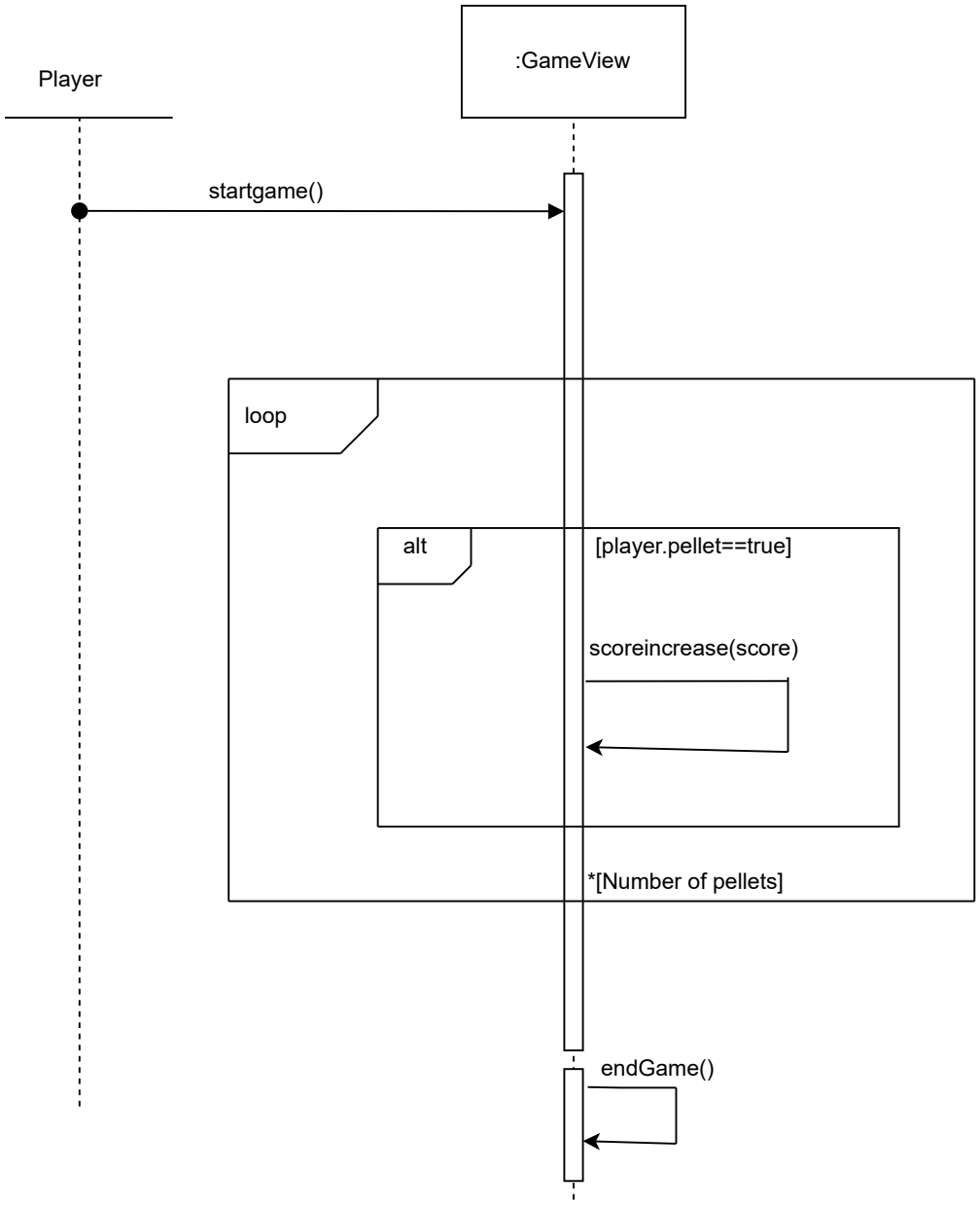
As a player, I want to eat all the pellets to win the game.



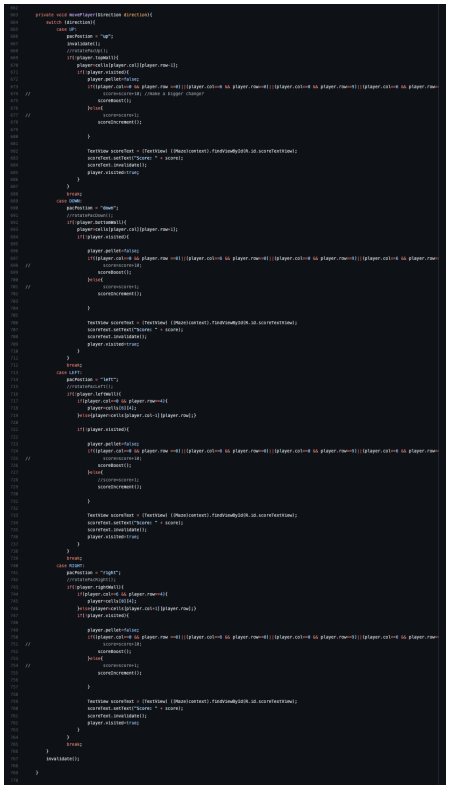
Devanshi's Use Case:
As a player, I want to kill enemies so I can earn more points.



Vishal Use Case: As a player, I want to collect pellets to increase my score



Aditi: <u>Code Smell- Bloaters: Long Method</u>



Problem: It exhibits the code smell long method, because as you can see in the screenshot above movePlayer(Direction direction) is a really long method. And as we learned in class, long methods are harder to understand and classes with short methods lives the longest.

Solution: To fix this code smell, we can use the possible heuristic taught in lectures- that whenever we feel the need to comment, make that into a new method. Although there are no comments in the method above, we can definitely break it into 5 smaller methods- the original movePlayer(Direction direction), which can check the direction of the player and call one of other four methods accordingly, like- moveRight(), moveLeft(), moveUp(), moveDown().

Devanshi: SOLID Principle: Single Responsibility Principle(SRP)

```
    private void rotatePacLeft() {
        switch(configure.getPacRes()) {
            case 1:
                pac=BitmapFactory.decodeResource(getResources(),R.drawable.mrpacleft);
                break;
            case 2:
                pac=BitmapFactory.decodeResource(getResources(),R.drawable.mspacleft);
                break;
            case 3:
                pac=BitmapFactory.decodeResource(getResources(),R.drawable.awarepacleft);
                break;
        }
//        gameCanvas.drawBitmap(pac,(player.col*cellSize)+(cellSize/4),(player.row*cellSize)+(cellSize/4),null);
    }

    private void rotatePacDown() {
        switch(configure.getPacRes()) {
            case 1:
                pac=BitmapFactory.decodeResource(getResources(),R.drawable.mrpacdown);
                break;
            case 2:
                pac=BitmapFactory.decodeResource(getResources(),R.drawable.mspacdown);
                break;
            case 3:
                pac=BitmapFactory.decodeResource(getResources(),R.drawable.awarepacdown);
                break;
        }
//        gameCanvas.drawBitmap(pac,(player.col*cellSize)+(cellSize/4),(player.row*cellSize)+(cellSize/4),null);

    }

    private void rotatePacUp() {
        switch(configure.getPacRes()) {
            case 1:
                pac=BitmapFactory.decodeResource(getResources(),R.drawable.mrpacup);
                break;
            case 2:
                pac=BitmapFactory.decodeResource(getResources(),R.drawable.mspacup);
                break;
            case 3:
                pac=BitmapFactory.decodeResource(getResources(),R.drawable.awarepacup);
                break;
        }
//        gameCanvas.drawBitmap(pac,(player.col*cellSize)+(cellSize/4),(player.row*cellSize)+(cellSize/4),null);

   private void rotatePacRight() {
       switch(configure.getPacRes()) {
           case 1:
               pac=BitmapFactory.decodeResource(getResources(),R.drawable.mrpacright);
               break;
           case 2:
               pac=BitmapFactory.decodeResource(getResources(),R.drawable.mspacright);
               break;
           case 3:
               pac=BitmapFactory.decodeResource(getResources(),R.drawable.awarepacright);
               break;
       }
//       canvas.drawBitmap(pac,(player.col*cellSize)+(cellSize/4),(player.row*cellSize)+(cellSize/4),null);

   }
```

Single Responsibility Principle(SRP) is displayed here as we see each method is responsible for just one thing that is rotating the pacman in one direction. There could have been one method that does all the rotations but here we implement the SRP and hence have an individual method for each rotation. Also, the names of the methods are really precise, like rotatePacRight indicates how it will rotate the pacman to the right, which is also a characteristic of SRP.

Adena Code Smell: Bloaters- Long Class

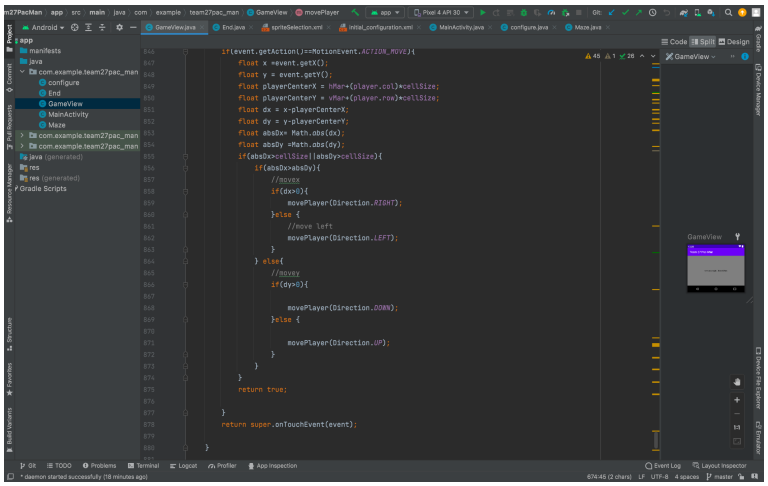
```java
package com.example.team27pac_man;

import ...

public class GameView extends View {
    Context context;
    private static int lives =configure.getLives();
    private static int score=0;
    private enum Direction{UP,DOWN,LEFT,RIGHT}
    //TODO
    //generate 3 to 4 different layoutIds for each difficulty level
    //assign "layout" to a specific layoutIds later depending on the level
    // make separate methods scoreBoost(), and incrementScore()
    private int [][] layout;
//    private Canvas gameCanvas;
    private String pacPostion = "right";

    Queue<int[]> redq = new LinkedList<>(Arrays.asList(new int[][]{{2,5},{3,5},{3,4},{3,3}}));
    Queue<int[]> yellowq = new LinkedList<>(Arrays.asList(new int[][]{{4,5},{3,5},{3,4},{3,3}}));
    Queue<int[]> blueq = new LinkedList<>(Arrays.asList(new int[][]{{4,4},{3,4},{3,3}}));
    Queue<int[]> pinkq = new LinkedList<>(Arrays.asList(new int[][]{{2,4},{3,4},{3,3}}));
    int redcounter=0;
    int yellowcounter=0;
    int bluecounter=0;
    int pinkcounter=0;

    private cell[][] cells;
    private cell player;
    private cell blue_ghost;
    private cell pink_ghost;
    private cell red_ghost;
    private cell yellow_ghost;
    private static final int COLS=7,ROWS=10;
    private float cellSize,hMar,vMar;
    private final float thick = 15;
```



Problem: This class is trying to accomplish too much. As you can see in the screenshots, large classes often have large amounts of instance variables. You can see ours is nearly 900 lines long.

Solution: One thing we can do is delete comments and have readable and complete code. Additionally, we should follow the SRP and have this class do just one thing. Right now, this class controls maze behaviors, ghost behaviors, and player movement. Each of these can and should have dedicated classes of their own.

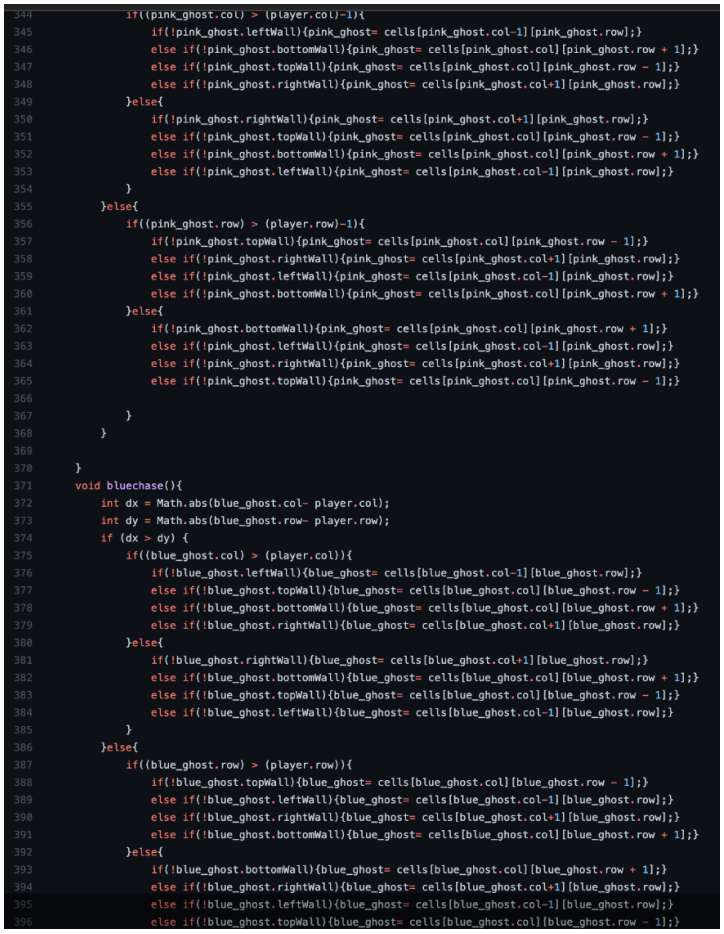
Ace SOLID Principle: SRP

```
    private class cell{
        boolean topWall = false;
        boolean bottomWall = false;
        boolean rightWall = false;
        boolean leftWall =false;
        boolean pellet =true;
        boolean visited =false;
        int col,row;

        public cell(int col, int row) {
            this.col = col;
            this.row = row;
        }
    }
}
```

SRP is being upheld here as this class exists solely to represent space in the 2D matrix. The class is used in the Maze file to create the individual elements of the maze(such as the enemies, pellets, walls and Pac-Man) and is used in a move specific context throughout the file. Therefore, it asks as a container class to help with the aspects of the game that the player interacts with.

Vishal Code Smell Object-Orientation Abusers: Switch Statements



```
            if((pink_ghost.col) > (player.col)-1){
                if(!pink_ghost.leftWall){pink_ghost= cells[pink_ghost.col-1][pink_ghost.row];}
                else if(!pink_ghost.bottomWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row + 1];}
                else if(!pink_ghost.topWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row - 1];}
                else if(!pink_ghost.rightWall){pink_ghost= cells[pink_ghost.col+1][pink_ghost.row];}
            }else{
                if(!pink_ghost.rightWall){pink_ghost= cells[pink_ghost.col+1][pink_ghost.row];}
                else if(!pink_ghost.topWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row - 1];}
                else if(!pink_ghost.bottomWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row + 1];}
                else if(!pink_ghost.leftWall){pink_ghost= cells[pink_ghost.col-1][pink_ghost.row];}
            }
        }else{
            if((pink_ghost.row) > (player.row)-1){
                if(!pink_ghost.topWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row - 1];}
                else if(!pink_ghost.rightWall){pink_ghost= cells[pink_ghost.col+1][pink_ghost.row];}
                else if(!pink_ghost.leftWall){pink_ghost= cells[pink_ghost.col-1][pink_ghost.row];}
                else if(!pink_ghost.bottomWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row + 1];}
            }else{
                if(!pink_ghost.bottomWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row + 1];}
                else if(!pink_ghost.leftWall){pink_ghost= cells[pink_ghost.col-1][pink_ghost.row];}
                else if(!pink_ghost.rightWall){pink_ghost= cells[pink_ghost.col+1][pink_ghost.row];}
                else if(!pink_ghost.topWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row - 1];}

            }
        }

    }
    void bluechase(){
        int dx = Math.abs(blue_ghost.col- player.col);
        int dy = Math.abs(blue_ghost.row- player.row);
        if (dx > dy) {
            if((blue_ghost.col) > (player.col)){
                if(!blue_ghost.leftWall){blue_ghost= cells[blue_ghost.col-1][blue_ghost.row];}
                else if(!blue_ghost.topWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row - 1];}
                else if(!blue_ghost.bottomWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row + 1];}
                else if(!blue_ghost.rightWall){blue_ghost= cells[blue_ghost.col+1][blue_ghost.row];}
            }else{
                if(!blue_ghost.rightWall){blue_ghost= cells[blue_ghost.col+1][blue_ghost.row];}
                else if(!blue_ghost.bottomWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row + 1];}
                else if(!blue_ghost.topWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row - 1];}
                else if(!blue_ghost.leftWall){blue_ghost= cells[blue_ghost.col-1][blue_ghost.row];}
            }
        }else{
            if((blue_ghost.row) > (player.row)){
                if(!blue_ghost.topWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row - 1];}
                else if(!blue_ghost.leftWall){blue_ghost= cells[blue_ghost.col-1][blue_ghost.row];}
                else if(!blue_ghost.rightWall){blue_ghost= cells[blue_ghost.col+1][blue_ghost.row];}
                else if(!blue_ghost.bottomWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row + 1];}
            }else{
                if(!blue_ghost.bottomWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row + 1];}
                else if(!blue_ghost.rightWall){blue_ghost= cells[blue_ghost.col+1][blue_ghost.row];}
                else if(!blue_ghost.leftWall){blue_ghost= cells[blue_ghost.col-1][blue_ghost.row];}
                else if(!blue_ghost.topWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row - 1];}
```

```
    private void createMaze(){
        cells=new cell[COLS][ROWS];
        for(int x = 0;x<COLS;x++){
            for(int y =0;y<ROWS;y++){

                cells[x][y]=new cell(x,y);
                if(layout[x][y]==1){
                    cells[x][y].topWall=true;
                }
                else if (layout[x][y]==2){
                    cells[x][y].bottomWall=true;

                }
                else if (layout[x][y]==3){
                    cells[x][y].leftWall=true;

                }
                else if (layout[x][y]==4){
                    cells[x][y].rightWall=true;

                }
                else if (layout[x][y]==5){
                    cells[x][y].topWall=true;
                    cells[x][y].leftWall=true;

                }
                else if (layout[x][y]==6){
                    cells[x][y].rightWall=true;
                    cells[x][y].topWall=true;

                }
                else if (layout[x][y]==7){
                    cells[x][y].bottomWall=true;
                    cells[x][y].leftWall=true;
                }
                else if (layout[x][y]==8){
                    cells[x][y].rightWall=true;
                    cells[x][y].bottomWall=true;

                }
                else if (layout[x][y]==10){
                    cells[x][y].topWall=true;
                    cells[x][y].bottomWall=true;

                }
                else if (layout[x][y]==11){
                    cells[x][y].rightWall=true;
                    cells[x][y].leftWall=true;

                }
                else if (layout[x][y]==12){
                    cells[x][y].rightWall=true;
                    cells[x][y].leftWall=true;
```

Problem: This class has many complex switch statements and many sequences of if and else-if statements, which makes the code unorganized and hard to understand.When a new condition is added, it requires us to find all the if/switch statements in the code and modify them.

Solution: In order to reduce the if statements , the use of polymorphism would be helpful. Putting complex switch statements into an extract method and puting them in their own subclasses and then replacing the conditional statement with the relevant method call would make it a polymorphic solution.