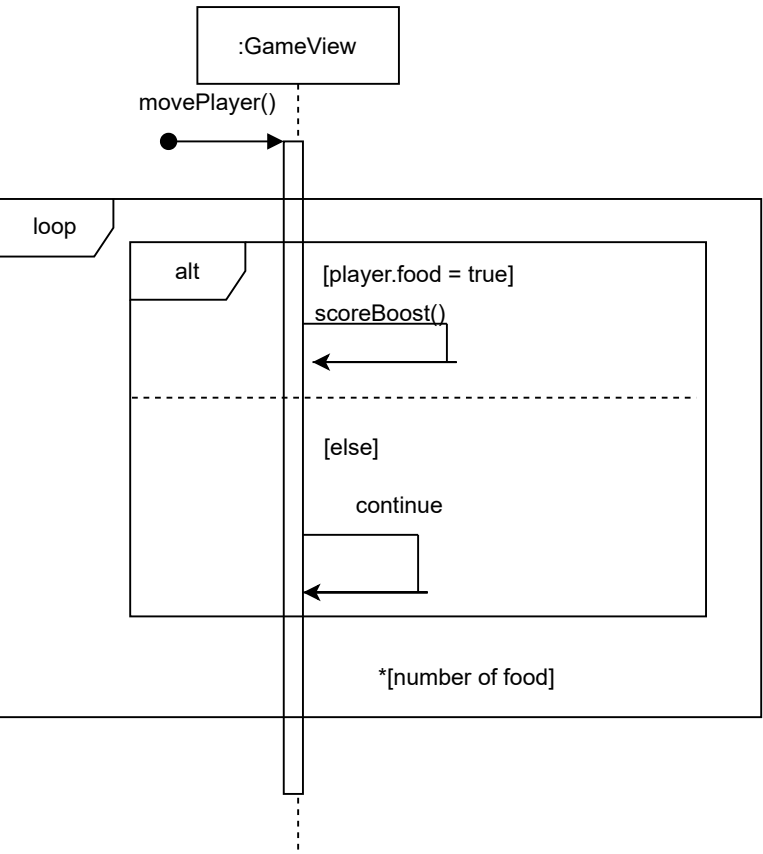


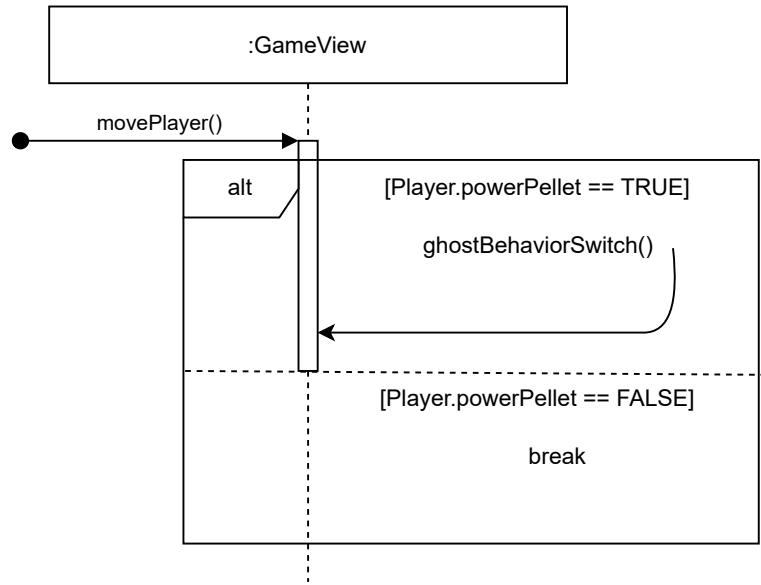
UPDATES:

- 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

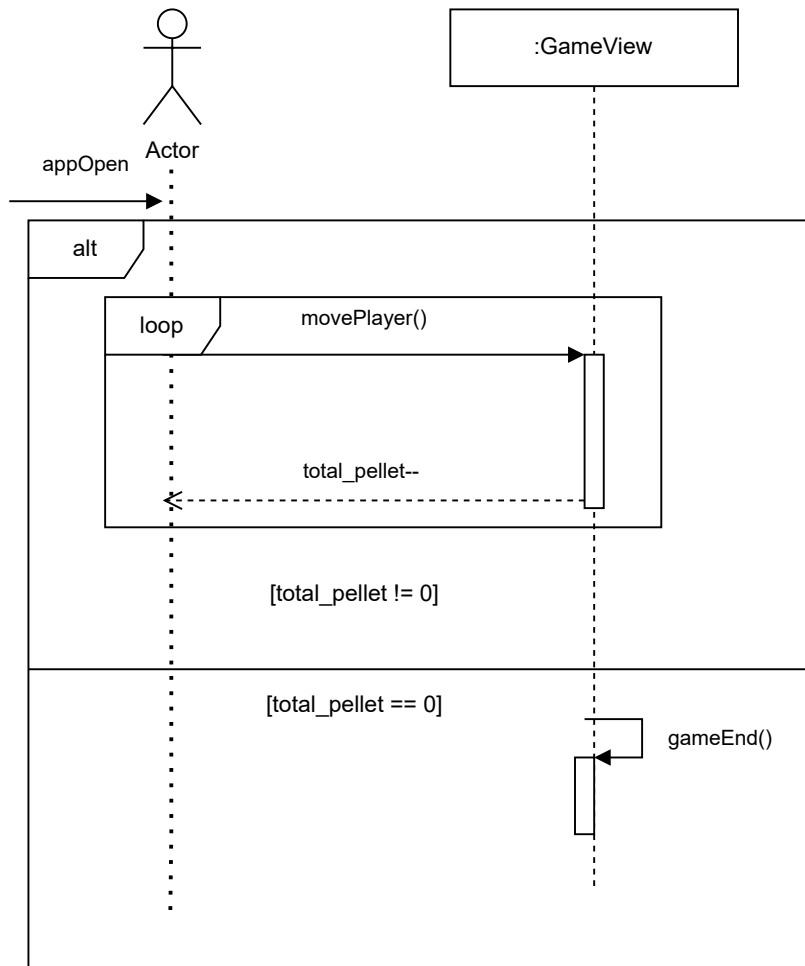


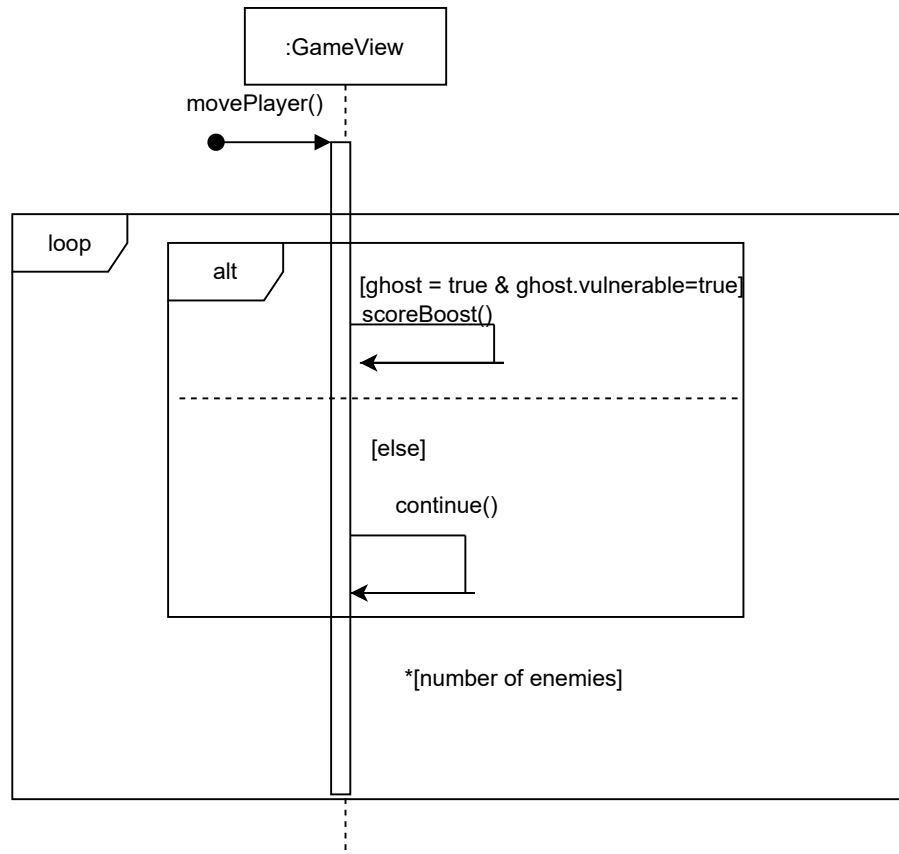
**Aditi's Use Case:** As a player, I want to collect food(fruit) so I can gain bonus points.

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



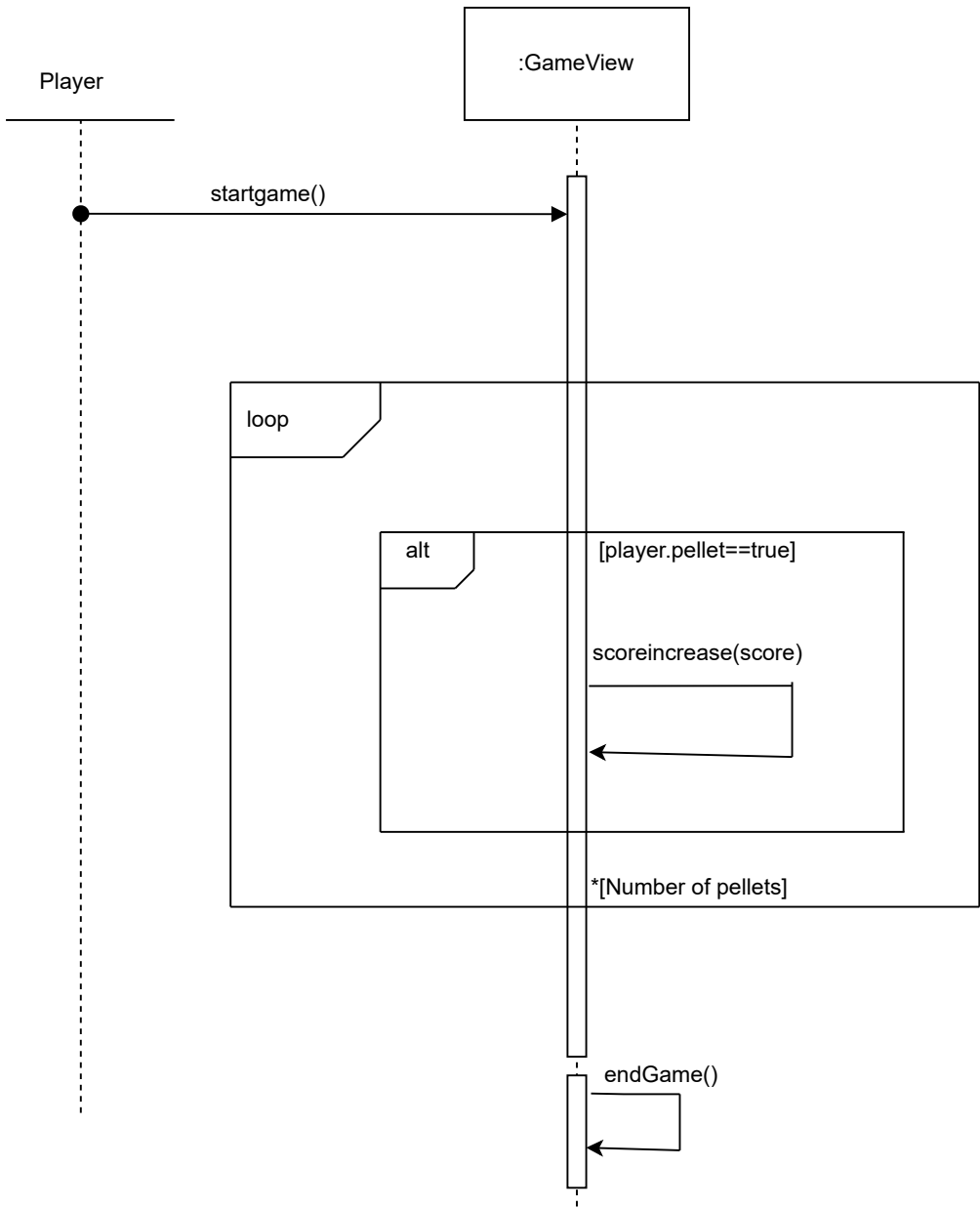
Caelan (Ace) Scales  
As a player, I want to eat all the pellets to win the game.



**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: Code Smell- Bloaters: Long Method

```
101 public void movePlayer(Direction direction)
102 {
103     // Move
104     direction = "up";
105     moveUp();
106     // Move
107     direction = "down";
108     moveDown();
109     // Move
110     direction = "left";
111     moveLeft();
112     // Move
113     direction = "right";
114     moveRight();
115 }
116
117 // Move
118 public void moveUp()
119 {
120     // Move
121     direction = "up";
122     moveUp();
123     // Move
124     direction = "down";
125     moveDown();
126     // Move
127     direction = "left";
128     moveLeft();
129     // Move
130     direction = "right";
131     moveRight();
132 }
133
134 // Move
135 public void moveDown()
136 {
137     // Move
138     direction = "down";
139     moveDown();
140     // Move
141     direction = "up";
142     moveUp();
143     // Move
144     direction = "left";
145     moveLeft();
146     // Move
147     direction = "right";
148     moveRight();
149 }
150
151 // Move
152 public void moveLeft()
153 {
154     // Move
155     direction = "left";
156     moveLeft();
157     // Move
158     direction = "right";
159     moveRight();
160     // Move
161     direction = "up";
162     moveUp();
163     // Move
164     direction = "down";
165     moveDown();
166 }
167
168 // Move
169 public void moveRight()
170 {
171     // Move
172     direction = "right";
173     moveRight();
174     // Move
175     direction = "left";
176     moveLeft();
177     // Move
178     direction = "up";
179     moveUp();
180     // Move
181     direction = "down";
182     moveDown();
183 }
184
185 // Move
186 public void move()
187 {
188     // Move
189     direction = "up";
190     moveUp();
191     // Move
192     direction = "down";
193     moveDown();
194     // Move
195     direction = "left";
196     moveLeft();
197     // Move
198     direction = "right";
199     moveRight();
200 }
```

**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

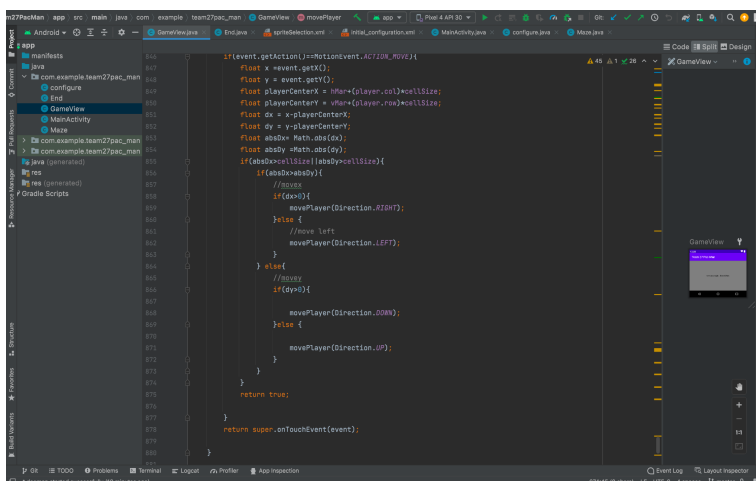
```
package com.example.team2.pac_man;

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 layouts for each difficulty level
    //assign "layout" to a specific layouts later depending on the level
    // make separate methods scoreBoost(), and incrementScore()
    private int [][] layout;
    // private Canvas gameCanvas;
    private String pacPosition = "right";

    Queue<int[]> redqs = new LinkedList<>(Arrays.asList(new int[][]{{2,5},{3,5},{3,6},{3,3}}));
    Queue<int[]> yellowqs = new LinkedList<>(Arrays.asList(new int[][]{{4,5},{3,5},{3,6},{3,3}}));
    Queue<int[]> blueqs = new LinkedList<>(Arrays.asList(new int[][]{{4,6},{3,4},{3,3}}));
    Queue<int[]> pinkqs = 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, hMap, vMap;
    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

```
344         if((pink_ghost.col) > (player.col-1)){
345             if(!pink_ghost.leftWall){pink_ghost= cells[pink_ghost.col-1][pink_ghost.row];}
346             else if(!pink_ghost.bottomWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row + 1];}
347             else if(!pink_ghost.topWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row - 1];}
348             else if(!pink_ghost.rightWall){pink_ghost= cells[pink_ghost.col+1][pink_ghost.row];}
349         }else{
350             if(!pink_ghost.rightWall){pink_ghost= cells[pink_ghost.col+1][pink_ghost.row];}
351             else if(!pink_ghost.topWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row - 1];}
352             else if(!pink_ghost.bottomWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row + 1];}
353             else if(!pink_ghost.leftWall){pink_ghost= cells[pink_ghost.col-1][pink_ghost.row];}
354         }
355     }else{
356         if((pink_ghost.row) > (player.row-1)){
357             if(!pink_ghost.topWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row - 1];}
358             else if(!pink_ghost.rightWall){pink_ghost= cells[pink_ghost.col+1][pink_ghost.row];}
359             else if(!pink_ghost.leftWall){pink_ghost= cells[pink_ghost.col-1][pink_ghost.row];}
360             else if(!pink_ghost.bottomWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row + 1];}
361         }else{
362             if(!pink_ghost.bottomWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row + 1];}
363             else if(!pink_ghost.leftWall){pink_ghost= cells[pink_ghost.col-1][pink_ghost.row];}
364             else if(!pink_ghost.rightWall){pink_ghost= cells[pink_ghost.col+1][pink_ghost.row];}
365             else if(!pink_ghost.topWall){pink_ghost= cells[pink_ghost.col][pink_ghost.row - 1];}
366         }
367     }
368 }
369 }
370 }
371 void bluechase(){
372     int dx = Math.abs(blue_ghost.col- player.col);
373     int dy = Math.abs(blue_ghost.row- player.row);
374     if (dx > dy) {
375         if((blue_ghost.col) > (player.col)){
376             if(!blue_ghost.leftWall){blue_ghost= cells[blue_ghost.col-1][blue_ghost.row];}
377             else if(!blue_ghost.topWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row - 1];}
378             else if(!blue_ghost.bottomWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row + 1];}
379             else if(!blue_ghost.rightWall){blue_ghost= cells[blue_ghost.col+1][blue_ghost.row];}
380         }else{
381             if(!blue_ghost.rightWall){blue_ghost= cells[blue_ghost.col+1][blue_ghost.row];}
382             else if(!blue_ghost.bottomWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row + 1];}
383             else if(!blue_ghost.topWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row - 1];}
384             else if(!blue_ghost.leftWall){blue_ghost= cells[blue_ghost.col-1][blue_ghost.row];}
385         }
386     }else{
387         if((blue_ghost.row) > (player.row)){
388             if(!blue_ghost.topWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row - 1];}
389             else if(!blue_ghost.leftWall){blue_ghost= cells[blue_ghost.col-1][blue_ghost.row];}
390             else if(!blue_ghost.rightWall){blue_ghost= cells[blue_ghost.col+1][blue_ghost.row];}
391             else if(!blue_ghost.bottomWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row + 1];}
392         }else{
393             if(!blue_ghost.bottomWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row + 1];}
394             else if(!blue_ghost.rightWall){blue_ghost= cells[blue_ghost.col+1][blue_ghost.row];}
395             else if(!blue_ghost.leftWall){blue_ghost= cells[blue_ghost.col-1][blue_ghost.row];}
396             else if(!blue_ghost.topWall){blue_ghost= cells[blue_ghost.col][blue_ghost.row - 1];}
397         }
398     }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
```

**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 putting them in their own subclasses and then replacing the conditional statement with the relevant method call would make it a polymorphic solution.

