Problem statements with their corresponding **Solutions.**

1. Problem: Ball does not bounce off the walls correctly.

Solution:
Ensure collision detection for the top and bottom walls is implemented correctly.

```
if (ball.y <= 0 || ball.y >= GAME_HEIGHT - BALL_DIAMETER) {
    ball.setYDirection(-ball.yVelocity);
}
```

2. Problem: Paddle movement is not smooth.

Solution:
Update the paddle's position in the `move()` method continuously using `yVelocity` instead of moving only when keys are pressed.

```
public void move() {
    y += yVelocity;
}
```

3. Problem: Ball passes through the paddle.

Solution:
Check for collision using `Rectangle` intersection and reverse the ball's X-direction when a collision occurs.

```
if (ball.intersects(paddle1) || ball.intersects(paddle2)) {
    ball.setXDirection(-ball.xVelocity);
}
```

4. Problem: Score does not update correctly.

Solution:
Check if the ball has crossed the left or right boundary and increment the corresponding player's score.

```
if (ball.x <= 0) {
   score.player2++;
   newBall();
}
if (ball.x >= GAME_WIDTH - BALL_DIAMETER) {
   score.player1++;
   newBall();
}
```

5. Problem: Game does not end when a player reaches the maximum score.

Solution:
Add a condition to check if either player has reached the `MAX_SCORE` and stop the game.

```
if (score.player1 >= MAX_SCORE || score.player2 >= MAX_SCORE) {
   gameRun = false;
   displayGameOver();
}
```

6. Problem: Paddle moves out of bounds.

Solution:
Limit paddle movement within the game screen boundaries.

```
if (paddle1.y <= 0) paddle1.y = 0;

if (paddle1.y >= GAME_HEIGHT - PADDLE_HEIGHT) paddle1.y = GAME_HEIGHT -
PADDLE_HEIGHT;
```

7. Problem: Ball moves too fast, making it difficult to play.

Solution:
Adjust the initial ball velocity and increment it gradually on paddle collision.

```
ball.xVelocity = 2;

ball.yVelocity = 2;
```

8. Problem: Game window is not centered on the screen.

Solution:
Use `setLocationRelativeTo(null)` to center the window.

```
this.setLocationRelativeTo(null);
```

9. Problem: Game restarts immediately after it ends.

Solution:
Show a "Game Over" screen with a button to restart the game instead of restarting automatically.

```
JButton restartButton = new JButton("Restart");

restartButton.addMouseListener(new MouseAdapter() {

    @Override

    public void mouseClicked(MouseEvent e) {

        new MainMenu();

    }

});
```

10. Problem: Players do not know which keys to use.

Solution:
Add key control instructions on the main menu screen.

```
JLabel controlsLabel = new JLabel("Player 1: W/S | Player 2: Up/Down");

controlsLabel.setForeground(Color.white);

controlsLabel.setBounds(100, 150, 200, 50);

panel.add(controlsLabel);
```

11. Problem: Ball always starts at the same position.

Solution:
Randomize the ball's initial position and direction.

```
random = new Random();

ball = new Ball(GAME_WIDTH / 2, random.nextInt(GAME_HEIGHT - BALL_DIAMETER), BALL_DIAMETER, BALL_DIAMETER);

ball.setXDirection(random.nextBoolean() ? -1 : 1);

ball.setYDirection(random.nextBoolean() ? -1 : 1);
```

12. Problem: Game runs too slowly.

Solution:
Increase the frame update rate by adjusting the `amountOfTicks` variable.

```
double amountOfTicks = 120.0;  // Increase from 60 to 120 for smoother gameplay.
```

13. Problem: No pause functionality.

Solution:
Add a pause feature using a key press.

```
boolean isPaused = false;

if (e.getKeyCode() == KeyEvent.VK_P) {

    isPaused = !isPaused;

}
```

14. Problem: Ball movement is not diagonal.

Solution:
Ensure both `xVelocity` and `yVelocity` are non-zero when initializing the ball.

```
ball.setXDirection(random.nextInt(2) == 0 ? 1 : -1);

ball.setYDirection(random.nextInt(2) == 0 ? 1 : -1);
```

15. Problem: Game frame is resizable, causing display issues.

Solution:
Disable resizing of the game window.

```
this.setResizable(false);
```

16. Problem: Paddles are not visually distinct.

Solution:
Assign different colors to each paddle.

```
g.setColor(id == 1 ? Color.blue : Color.red);

g.fillRect(x, y, width, height);
```

17. Problem: Game Over screen is not visually appealing.

Solution:
Improve layout with better alignment and fonts.

```
gameOverLabel.setFont(new Font("Arial", Font.BOLD, 40));
```

18. Problem: Background is too plain.

Solution:
Add a simple background color or image.

```
this.setBackground(Color.darkGray);
```

19. Problem: Players cannot quit the game mid-match.

Solution:
Add a "Quit" button in the main game panel.

```
JButton quitButton = new JButton("Quit");

quitButton.addMouseListener(new MouseAdapter() {

    @Override

    public void mouseClicked(MouseEvent e) {

        System.exit(0);

    }

});
```

20. Problem: Paddle speed is too high.

Solution:
Reduce the `speed` variable of the paddle.

```
int speed = 5;  // Lower from 10 to 5.
```

21. Problem: Ball does not speed up after a collision.

Solution:
Increase the ball's velocity after each paddle collision.

```
ball.xVelocity += ball.xVelocity > 0 ? 1 : -1;

ball.yVelocity += ball.yVelocity > 0 ? 1 : -1;
```

22. Problem: No sound effects.

Solution:
Add basic sound effects on paddle and wall collisions

```
AudioClip clip = Applet.newAudioClip(new URL("paddle_hit.wav"));

clip.play();
```

.23. Problem: No score reset after restarting the game.

Solution:
Reset scores when the game restarts.

```
score.player1 = 0;

score.player2 = 0;
```

24. Problem: Main menu is not shown after quitting a game.

Solution:
Display the main menu after the Game Over screen.

```
new MainMenu();
```

25. Problem: Game feels too static.

Solution:
Add animations or particle effects when scoring a point.

```
g.setColor(Color.yellow);

g.fillOval(ball.x, ball.y, BALL_DIAMETER, BALL_DIAMETER);
```