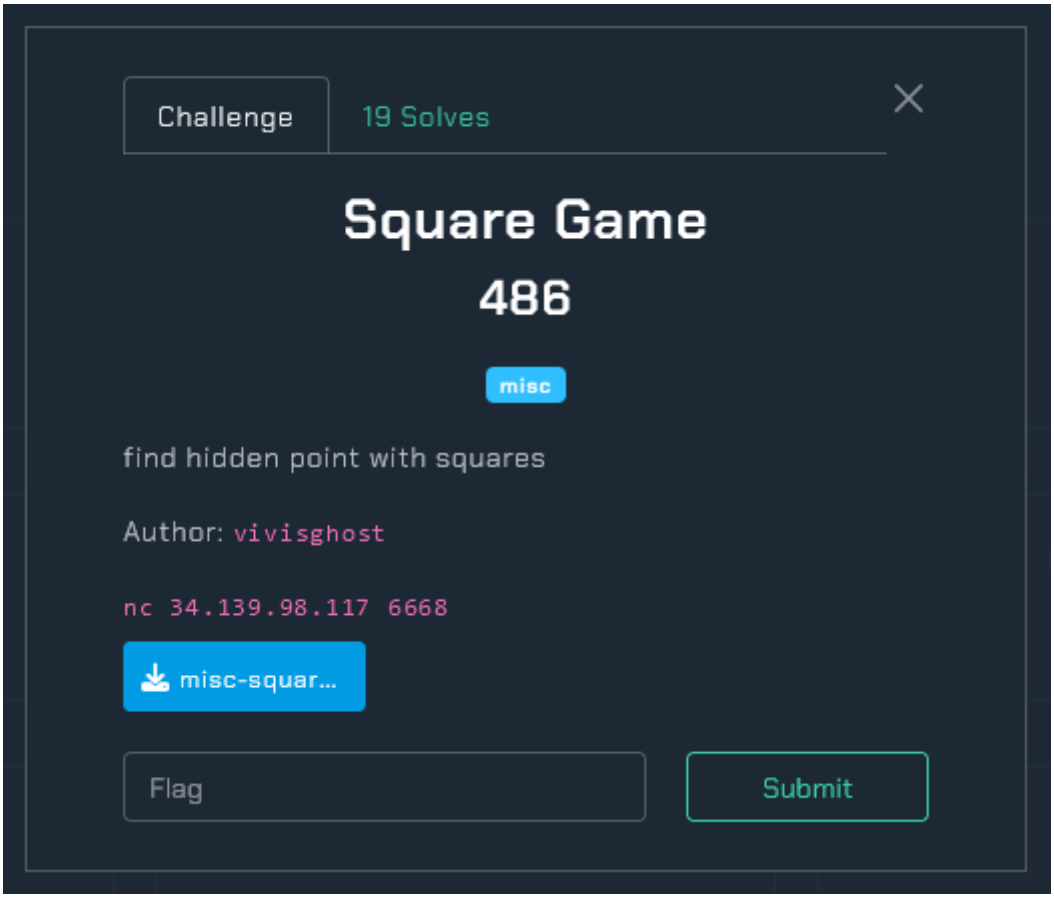
Square Game Walkthrough



How to Solve the Challenge

Essentially, this is a binary search problem. Once a hit is recorded, a bounding box can be established with the following parameters: x_min, x_max, y_min, and y_max. The strategy is to select a radius for the next guess such that the new square halves the bounding box.

Upon a hit, the bounding box can be contracted by the overlap between the new square and the bounding box.

After a miss, we can adjust the bounding box by subtracting the area covered by the last guess square.

Choosing a radius.

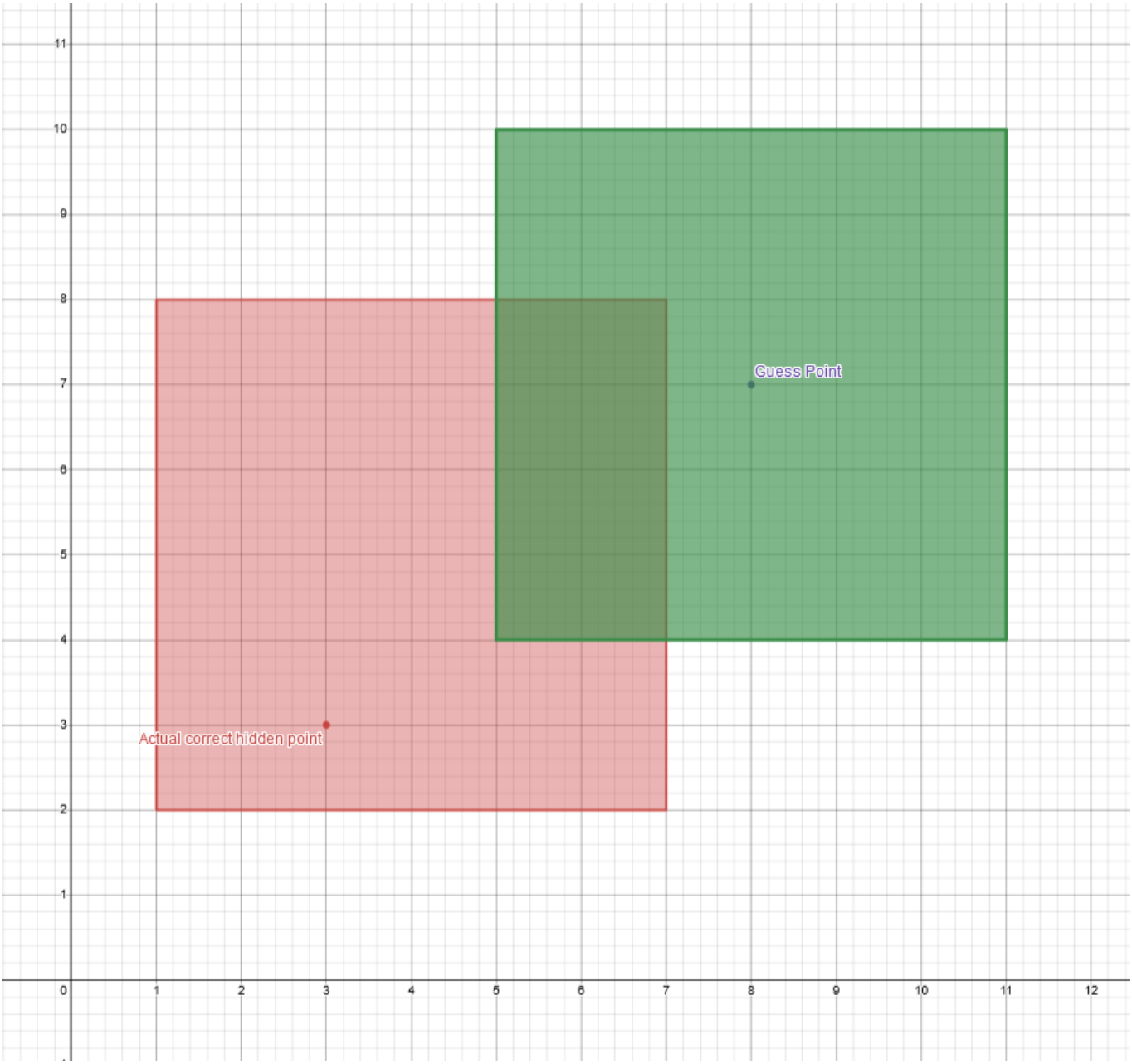
Selecting an appropriate radius is key to maintaining efficient search boundaries. To avoid irregular shapes upon a miss, it is important that at least three of the four bounds of the new square are either equal to or exceed the dimensions of the bounding box:

- The maximums must be greater than the corresponding bounding box maximums.
- The minimums must be less than the corresponding bounding box minimums.

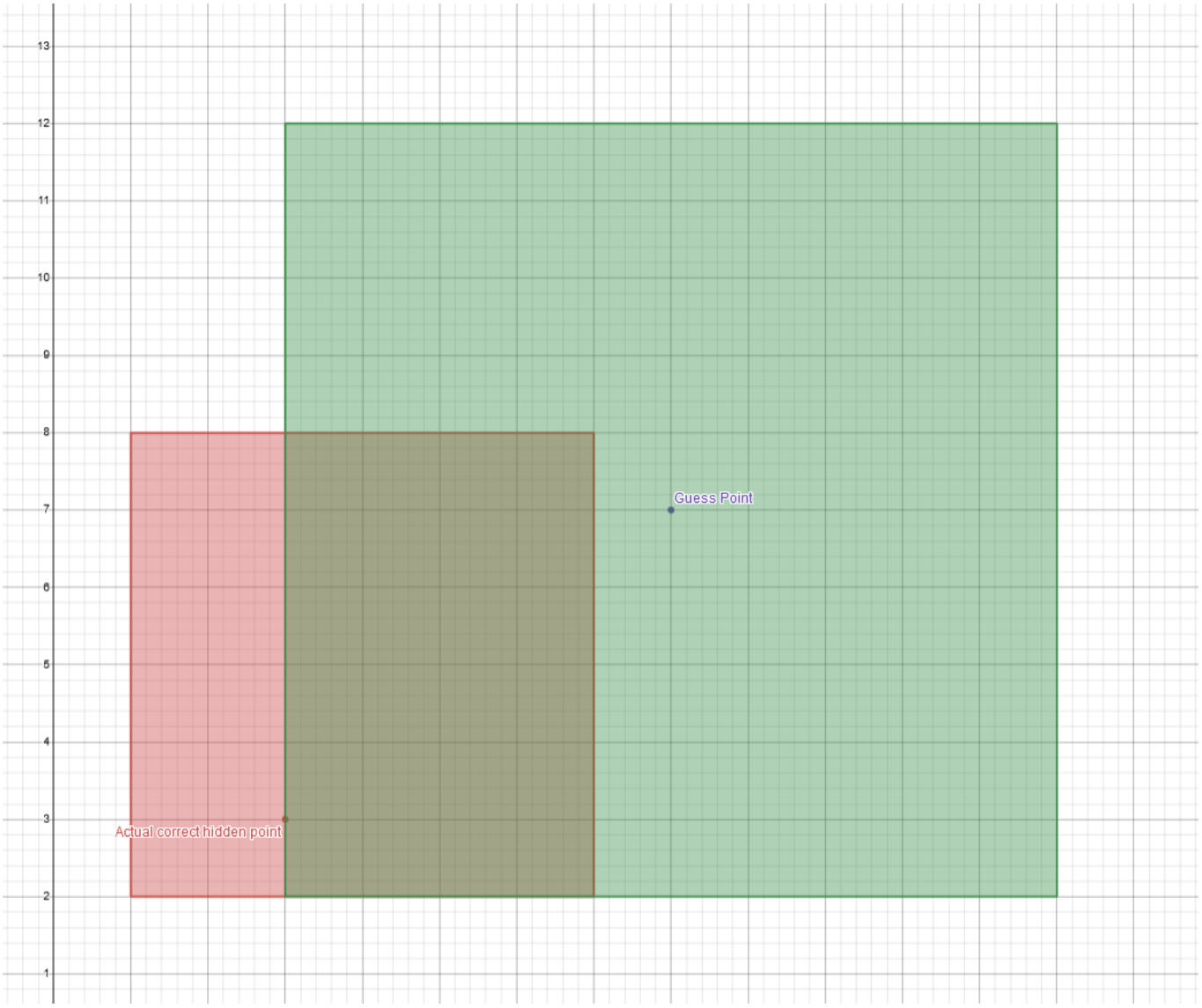
For example, a miss resulting in an 'L' shaped bounding box indicates that the y_max and x_max exceed their counterparts by a total of 2, necessitating at least one min (y_min or x_min) to be less.

It is also critical to note that if all four bounds are larger, it guarantees a hit, and no new information is gained since the bounding box remains unchanged in size.

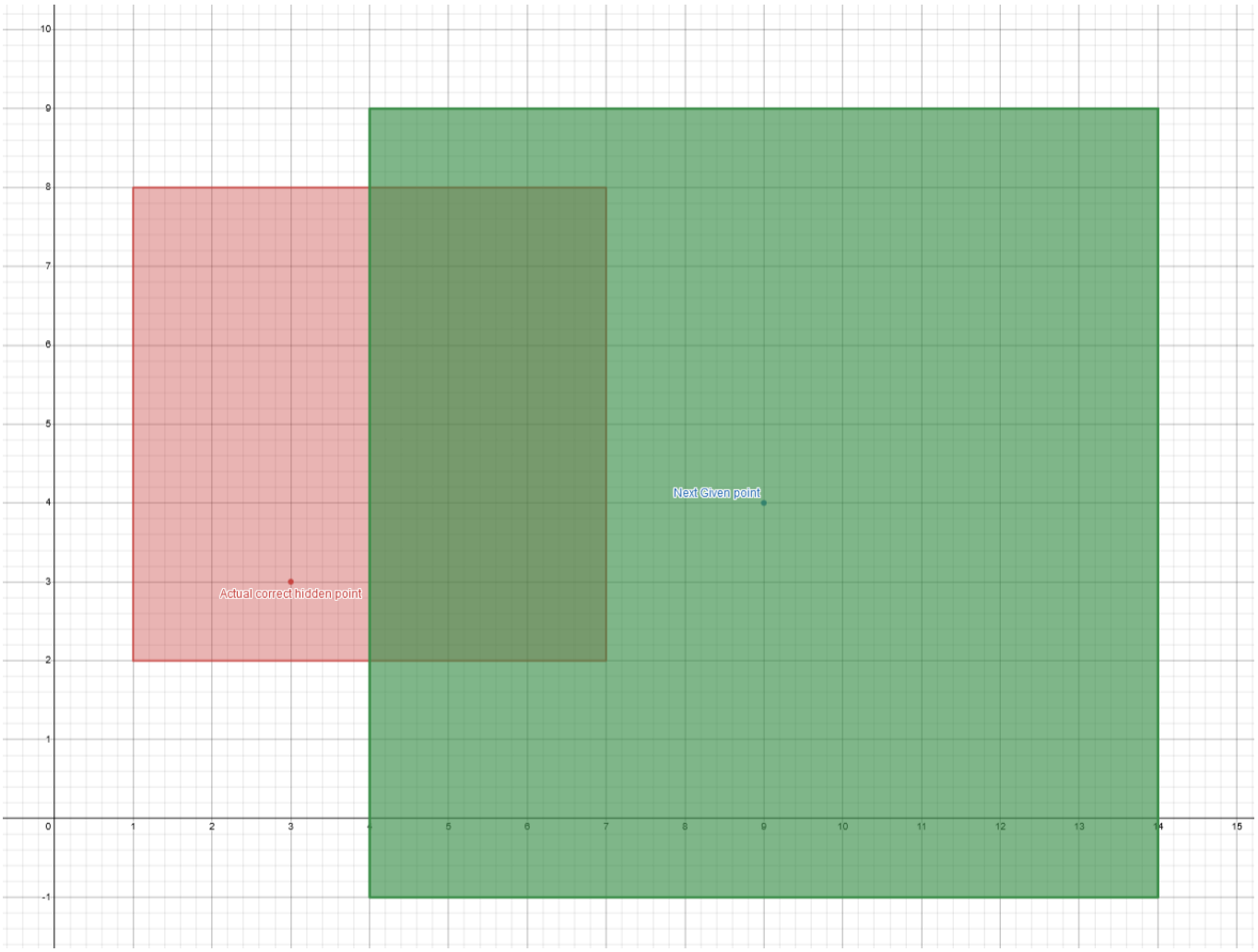
By adhering to the '3/4 rule' when choosing a radius, we can ensure that a miss leaves us with a rectangular bounding box.



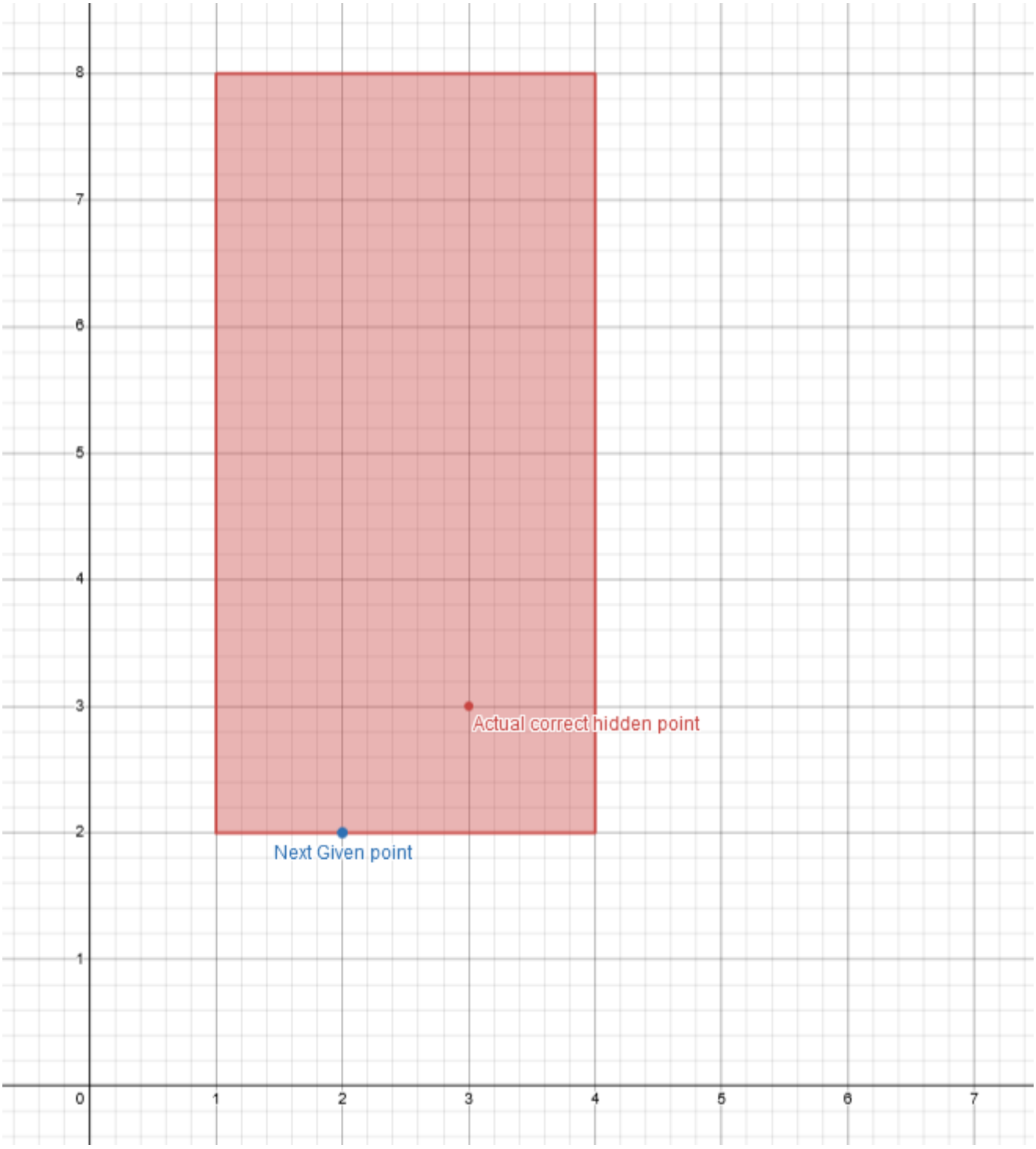
Here choosing a radius that satisfies the 3/4 rule ensures we are left with a rectangle if we miss.



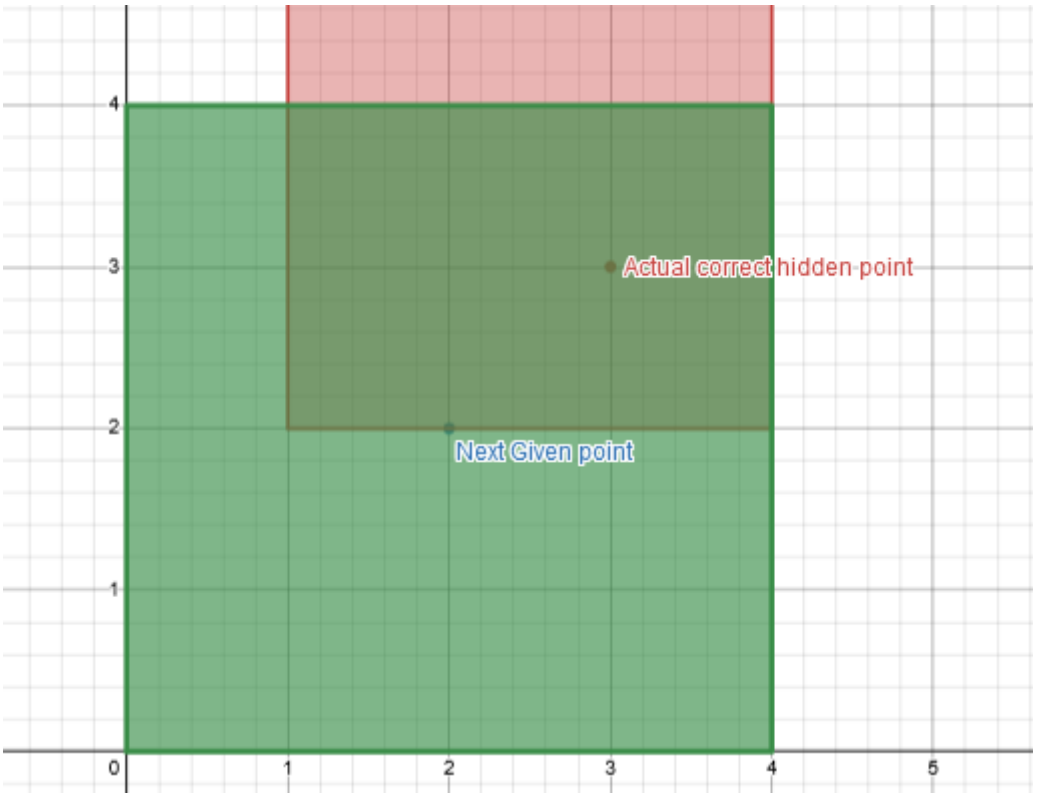
To demonstrate a bounding box update again on a miss


Next Given point
Actual correct hidden point

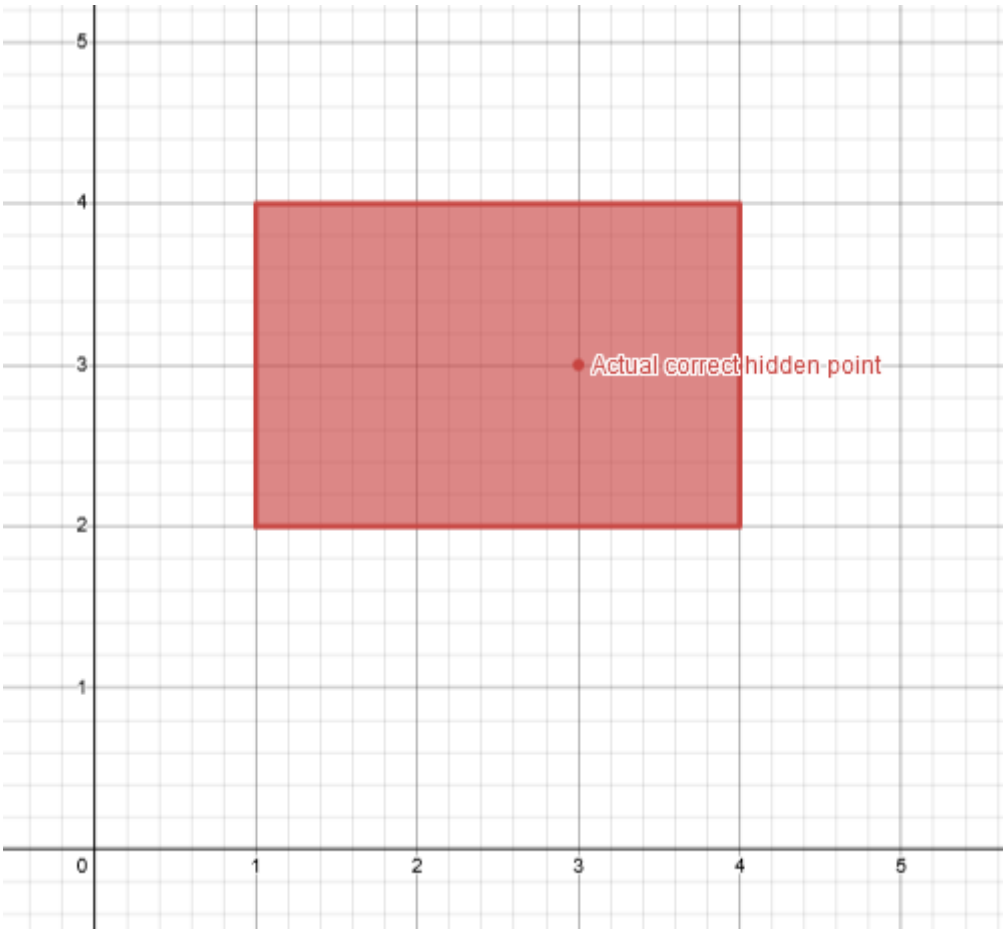
Below shows the updating bounding box after the miss.



Picking a new point showing and example of a hit the bounding box will be updated by the overlap of the new guess square and the bounding box. Below a radius of 2 was selected.



Updated bounding box



The goal is to continue this process of guessing and updating until the bounding box narrows down to a single point.

Keep in mind the location of the hidden point was only shown here for demonstration purposes and the player is not privy to this knowledge.

Example of game + code in action

Connecting to game

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python3 square_solve_live.py
[+] Opening connection to 34.139.98.117 on port 6668: Done
Welcome to the Point Locator Challenge!

The object of this game is to find the coordinates of a hidden point on the grid.
Each round you will be given a new random point on the grid and you will give us a radius of a square.
We will return if the hidden point was in the area of your square.
After 100 rounds you will be asked to submit your guess of the coordinates of the hidden point x,y.
To win you will have to win 10 games in a row.

The size of your grid is 1000000, 1000000


You have an error of 10000.0


Number of games you must win: 10
Number of rounds per game: 100
```

Main Logic

```python
# Main function
def main():
    # Read the initial messages from the server
    initial_data = conn.recvuntil(b"The size of your grid is 1000000, 1000000")
    print_data(initial_data)
    c = 1
    bounding_box = False
    try:
        while True:
            data = conn.recvline(timeout=3).decode().strip()
            print(data)  # Print every line received from the server
            response = ''
            if 'Point is' in data:
                point = parse_point(data)
                if bounding_box == False:
                    radius = 370000
                else:
                    radius = calculate_radius(point, bounding_box)
                print(f"Radius: {radius}")  # Print the radius we're sending
                conn.sendline(str(radius))  # Send the radius
                c += 1
            elif 'idden point is' in data:
                bounding_box = handle_response(data, point, radius, bounding_box)
                guess = your_guessing_logic_here(bounding_box)
                print(f"Current Guess: {guess}")  # Print the guess
            elif c >= 100 and 'Guess' in data:
                print(data)
                # If the server is prompting for a guess, handle it here
                guess = your_guessing_logic_here(bounding_box)
                print(f"Guess: {guess}")  # Print the guess
                conn.sendline(f'{guess[0]},{guess[1]}')
                bounding_box = False
                c = 1
```

If a bounding box has not been established, guess 370000 as radius until a hit is recorded. Use that as a bounding box.

If a bound box has been selected, select a radius that would form a square that would split the bounding box in half.

If hit, update bounding box with overlap, else, bounding box = guess square - bounding box.

Winning final game

```
***Game: 10***

Round 1: Point is (278615, 857877)
Radius: 370000
Enter Radius Length>
The hidden point is inside your square!
Current Guess: (278615, 857877)


Round 2: Point is (548049, 237764)
Radius: 814773
Enter Radius Length>
The hidden point is inside your square!
Current Bounding Box {'min_x': -91385, 'max_x': 648615, 'min_y': 487877, 'max_y': 1052537}
Current Guess: (278615, 770207)


Round 3: Point is (959549, 293485)
Radius: 904993
Enter Radius Length>
The hidden point is inside your square!
Current Bounding Box {'min_x': 54556, 'max_x': 648615, 'min_y': 487877, 'max_y': 1052537}
Current Guess: (351585, 770207)
```

```
Round 99: Point is (985798, 221502)
Radius: 554766
Enter Radius Length>
The hidden point is inside your square!
Current Bounding Box {'min_x': 623036, 'max_x': 623041, 'min_y': 776263, 'max_y': 776268}
Current Guess: (623038, 776265)


Round 100: Point is (756735, 593642)
Radius: 182626
Enter Radius Length>
The hidden point is inside your square!
Current Bounding Box {'min_x': 623036, 'max_x': 623041, 'min_y': 776263, 'max_y': 776268}
Current Guess: (623038, 776265)


Guess the hidden point (format 'x,y'):
Guess the hidden point (format 'x,y'):
Guess: (623038, 776265)
Correct! You've found the hidden point.


Congratulations, you've done it. Here is your flag: L3AK{5qu4r35_574y_5h4rp}

Connection closed by the server.
[*] Closed connection to 34.139.98.117 port 6668
```

We can see by round 15 the algo had already achieves a guess that would be within the 1% error range.

```
Round 14: Point is (593670, 605070)
Radius: 171738
Enter Radius Length>
The hidden point is inside your square!
Current Bounding Box {'min_x': 616803, 'max_x': 627407, 'min_y': 775865, 'max_y': 776808}
Current Guess: (622105, 776336)


Round 15: Point is (883399, 633302)
Radius: 261294
Enter Radius Length>
The hidden point is inside your square!
Current Bounding Box {'min_x': 622105, 'max_x': 627407, 'min_y': 775865, 'max_y': 776808}
Current Guess: (624756, 776336)
```

By round 60 the algorithm typically finds the exact coordinate.

Full python code @ square_solve_live.py