In [42]:

```python
import random
import matplotlib.pyplot as plt

# Define the Floorplanning constraints
room_data = {
    "Living": {"length": (8, 20), "width": (8, 20), "area": (120, 300), "proportion": 1.
    "Kitchen": {"length": (6, 18), "width": (6, 18), "area": (50, 120), "proportion": 1e
    "Bath": {"length": (5.5, 5.5), "width": (8.5, 8.5), "area": (None, None), "proportio
    "Hall": {"length": (5.5, 5.5), "width": (3.5, 6), "area": (19, 72), "proportion": 1e
    "Bed1": {"length": (10, 17), "width": (10, 17), "area": (100, 180), "proportion": 1.
    "Bed2": {"length": (9, 20), "width": (9, 20), "area": (100, 180), "proportion": 1.5}
    "Bed3": {"length": (8, 18), "width": (8, 18), "area": (100, 180), "proportion": 1.5}
}

# Additional constraints
doorway_space = 3.0  # units

# Length and width will be represented by 6 bits each

chromosome_length = 6

def encode_chromosome():
    chromosome = {}
    for room, constraints in room_data.items():
        min_length, max_length = constraints["length"]
        min_width, max_width = constraints["width"]

        length = random.uniform(min_length, max_length)
        width = random.uniform(min_width, max_width)

        while True:
            length_bits = format(int(length), f'0{chromosome_length}b')
            width_bits = format(int(width), f'0{chromosome_length}b')

            if length_bits != '0' * chromosome_length and width_bits != '0' * chromosome
                break
            else:
                length = random.uniform(min_length, max_length)
                width = random.uniform(min_width, max_width)

        chromosome[room] = {"length": length_bits, "width": width_bits}
    return chromosome


def decode_chromosome(chromosome):
    decoded_chromosome = {}
    for room, dimensions in chromosome.items():
        decoded_chromosome[room] = {
            "length": int(dimensions["length"], 2),
            "width": int(dimensions["width"], 2)
        }
    return decoded_chromosome

def fitness_function(chromosome):
    cost = 1
    for room, dimensions in chromosome.items():
        length = int(dimensions["length"], 2)
        width = int(dimensions["width"], 2)
        area = length * width

        min_len, max_len, min_width, max_width, min_area, max_area, prop =  \
        room_data[room]["length"][0], room_data[room]["length"][1], \
        room_data[room]["width"][0], room_data[room]["width"][1], \
        room_data[room]["area"][0], room_data[room]["area"][1], \
```

```
        room_data[room]["proportion"]

        # Include doorway space constraint
        if room == "Living" or room == "Bed1" or room == "Bed2" or room == "Bed3":
            if (length - doorway_space) < min_len or (length + doorway_space) > max_len
                    or (width - doorway_space) < min_width or (width + doorway_space) >
                    or min_area and area < min_area or max_area and area > max_area \
                    or prop and length / width != prop:
                area =  1e6
            cost += 2 * area if room in ('Kitchen', 'Bath') else area
        else:
            if room == "Hall":
                # Check if the space for doorway is available between Bed2, Bed3, and Ha
                bed2_length = int(chromosome["Bed2"]["length"], 2)
                bed2_width = int(chromosome["Bed2"]["width"], 2)
                bed3_length = int(chromosome["Bed3"]["length"], 2)
                bed3_width = int(chromosome["Bed3"]["width"], 2)

                if bed2_length + bed3_length >= length + doorway_space and \
                   bed2_width >= width and bed3_width >= width:
                    pass  # Doorway space is available
                else:
                    area =  1e6  # Adjust area if doorway space is not available
            else:
                if min_len and length < min_len or max_len and length > max_len \
                        or min_width and width < min_width or max_width and width > max_
                        or min_area and area < min_area or max_area and area > max_area
                        or prop and length / width != prop:
                    area =  1e6
                cost += 2 * area if room in ('Kitchen', 'Bath') else area

    return cost

def crossover(parent1, parent2, crossover_probability):
    child1 = {}
    child2 = {}

    if random.random() < crossover_probability:
        crossover_point = random.choice(list(parent1.keys()))
        for room in room_data.keys():
            if room < crossover_point:
                child1[room] = parent1[room]
                child2[room] = parent2[room]
            else:
                child1[room] = parent2[room]
                child2[room] = parent1[room]
    else:
        child1 = parent1.copy()
        child2 = parent2.copy()

    return child1, child2


def mutate(chromosome, mutation_probability):
    mutated_chromosome = chromosome.copy()
    for room, constraints in room_data.items():
        if random.random() < mutation_probability:
            length_bits = ''.join(random.choice('01') for _ in range(chromosome_length))
            width_bits = ''.join(random.choice('01') for _ in range(chromosome_length))
            mutated_chromosome[room]["length"] = length_bits
            mutated_chromosome[room]["width"] = width_bits
    return mutated_chromosome


def genetic_algorithm(population_size, num_generations, crossover_probability, mutation_
```

```
    # Initialize population
    population = [encode_chromosome() for _ in range(population_size)]

    best_fitness_values = []

    # Main loop
    for generation in range(num_generations):
        # Evaluate fitness
        fitness_scores = [fitness_function(chromosome) for chromosome in population]
        best_fitness_values.append(min(fitness_scores))

        # Select parents for crossover
        parents = random.choices(population, weights=fitness_scores, k=population_size)

        # Perform crossover
        new_population = []
        for i in range(0, population_size, 2):
            child1, child2 = crossover(parents[i], parents[i+1], crossover_probability)
            new_population.extend([mutate(child1, mutation_probability), mutate(child2,

        # Replace old population with new population
        population = new_population

    # Get the best solution
    best_chromosome = min(population, key=fitness_function)
    best_dimensions = decode_chromosome(best_chromosome)
    return best_chromosome, best_dimensions, best_fitness_values


# Setting GA parameters

parameter_combinations = [
    (50, 100, 0.7, 0.001),
    (100, 200, 0.8, 0.002),
    (30, 150, 0.6, 0.001),
    (200, 300, 0.8, 0.002),
    (50, 75, 0.95, 0.05),
    (100, 75, 0.5, 0.01)
]

for params in parameter_combinations:
    population_size, num_generations, crossover_probability, mutation_probability = para

    #print(f"Running with parameters: {params}")
    print(f"\033[1;31mRunning with parameters: {params}\033[0m")
    print("\n")
    best_chromosome, best_floorplan, best_fitness_values = genetic_algorithm(population_

    print(f"\033[1;32mBest Chromosome: {params}\033[0m")
    for room, dimensions in  best_chromosome.items():
        print(f"Room: {room}, Length: {dimensions['length']}, Width: {dimensions['width'

    print(f"\033[1;32mBest Floorplan\033[0m")
    for room, dimensions in  best_floorplan.items():
        length = dimensions['length']
        width = dimensions['width']
        area = length * width
        print(f"Room: {room}, Length: {length}, Width: {width}, Area: {area}")

        #print(f"Room: {room}, Length: {dimensions['length']}, Width: {dimensions['width
    print("\n")
```

Running with parameters: (50, 100, 0.7, 0.001)

Best Chromosome: (50, 100, 0.7, 0.001)

Room: Living, Length: 110111, Width: 110011
Room: Kitchen, Length: 010100, Width: 111101
Room: Bath, Length: 001100, Width: 111100
Room: Hall, Length: 100100, Width: 010111
Room: Bed1, Length: 101100, Width: 110100
Room: Bed2, Length: 100110, Width: 100100
Room: Bed3, Length: 100111, Width: 011011
Best Floorplan
Room: Living, Length: 55, Width: 51, Area: 2805
Room: Kitchen, Length: 20, Width: 61, Area: 1220
Room: Bath, Length: 12, Width: 60, Area: 720
Room: Hall, Length: 36, Width: 23, Area: 828
Room: Bed1, Length: 44, Width: 52, Area: 2288
Room: Bed2, Length: 38, Width: 36, Area: 1368
Room: Bed3, Length: 39, Width: 27, Area: 1053

Running with parameters: (100, 200, 0.8, 0.002)

Best Chromosome: (100, 200, 0.8, 0.002)
Room: Living, Length: 001000, Width: 001001
Room: Kitchen, Length: 110101, Width: 001001
Room: Bath, Length: 001000, Width: 010010
Room: Hall, Length: 011110, Width: 011110
Room: Bed1, Length: 011111, Width: 000110
Room: Bed2, Length: 000111, Width: 000000
Room: Bed3, Length: 010111, Width: 110000
Best Floorplan
Room: Living, Length: 8, Width: 9, Area: 72
Room: Kitchen, Length: 53, Width: 9, Area: 477
Room: Bath, Length: 8, Width: 18, Area: 144
Room: Hall, Length: 30, Width: 30, Area: 900
Room: Bed1, Length: 31, Width: 6, Area: 186
Room: Bed2, Length: 7, Width: 0, Area: 0
Room: Bed3, Length: 23, Width: 48, Area: 1104

Running with parameters: (30, 150, 0.6, 0.001)

Best Chromosome: (30, 150, 0.6, 0.001)
Room: Living, Length: 110110, Width: 000001
Room: Kitchen, Length: 010101, Width: 101010
Room: Bath, Length: 010100, Width: 010100
Room: Hall, Length: 010110, Width: 100010
Room: Bed1, Length: 110011, Width: 101000
Room: Bed2, Length: 010000, Width: 000110
Room: Bed3, Length: 010000, Width: 111101
Best Floorplan
Room: Living, Length: 54, Width: 1, Area: 54
Room: Kitchen, Length: 21, Width: 42, Area: 882
Room: Bath, Length: 20, Width: 20, Area: 400
Room: Hall, Length: 22, Width: 34, Area: 748
Room: Bed1, Length: 51, Width: 40, Area: 2040
Room: Bed2, Length: 16, Width: 6, Area: 96
Room: Bed3, Length: 16, Width: 61, Area: 976

Running with parameters: (200, 300, 0.8, 0.002)

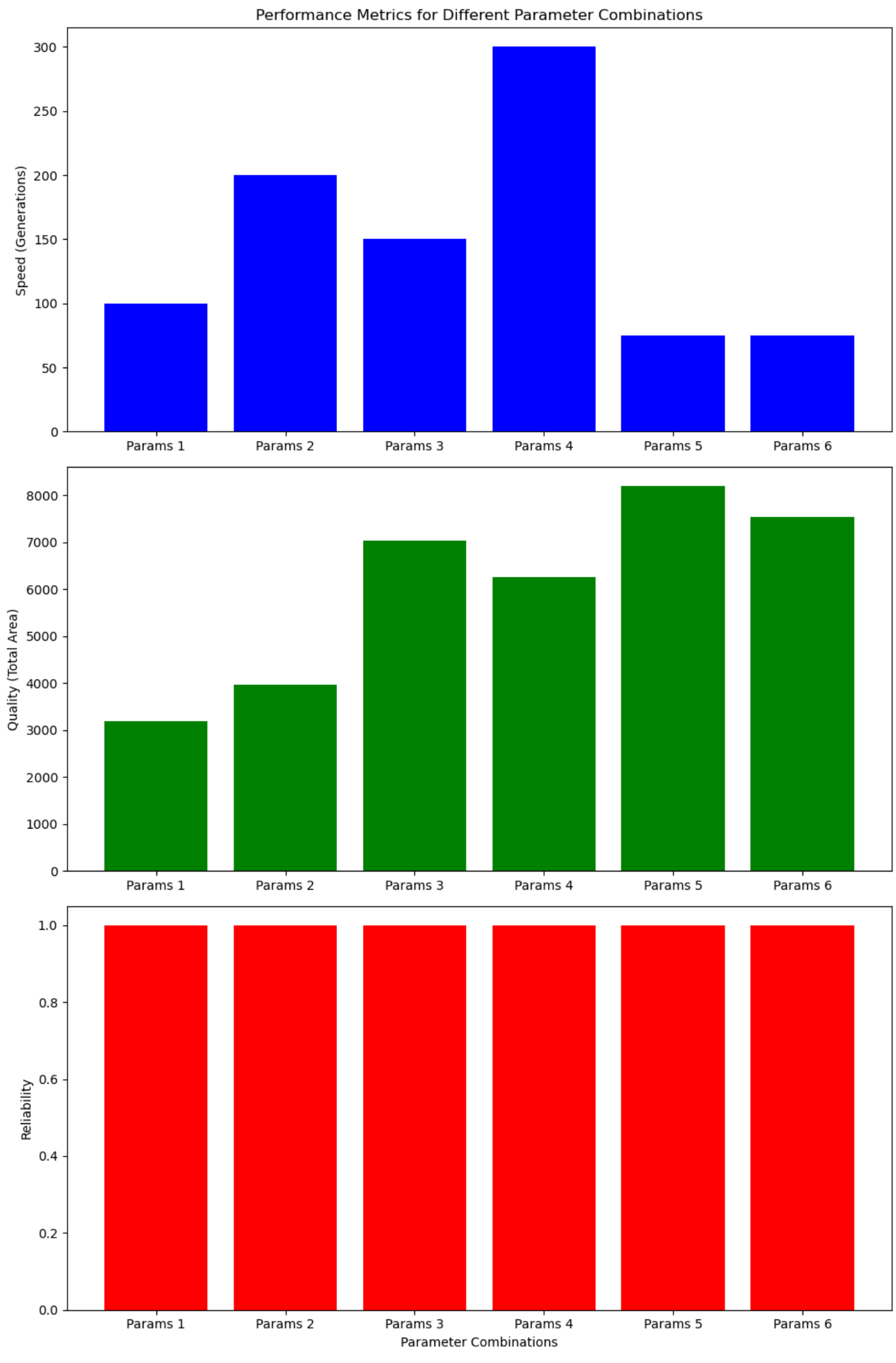
Best Chromosome: (200, 300, 0.8, 0.002)
Room: Living, Length: 000010, Width: 010011
Room: Kitchen, Length: 110101, Width: 010111
Room: Bath, Length: 110100, Width: 100110

```
Room: Hall, Length: 110110, Width: 001111
Room: Bed1, Length: 100000, Width: 110101
Room: Bed2, Length: 001101, Width: 011001
Room: Bed3, Length: 000011, Width: 000101
Best Floorplan
Room: Living, Length: 2, Width: 19, Area: 38
Room: Kitchen, Length: 53, Width: 23, Area: 1219
Room: Bath, Length: 52, Width: 38, Area: 1976
Room: Hall, Length: 54, Width: 15, Area: 810
Room: Bed1, Length: 32, Width: 53, Area: 1696
Room: Bed2, Length: 13, Width: 25, Area: 325
Room: Bed3, Length: 3, Width: 5, Area: 15


Running with parameters: (50, 75, 0.95, 0.05)


Best Chromosome: (50, 75, 0.95, 0.05)
Room: Living, Length: 000010, Width: 001001
Room: Kitchen, Length: 110010, Width: 101110
Room: Bath, Length: 100001, Width: 001011
Room: Hall, Length: 100011, Width: 101111
Room: Bed1, Length: 110101, Width: 100000
Room: Bed2, Length: 000101, Width: 010111
Room: Bed3, Length: 111001, Width: 100010
Best Floorplan
Room: Living, Length: 2, Width: 9, Area: 18
Room: Kitchen, Length: 50, Width: 46, Area: 2300
Room: Bath, Length: 33, Width: 11, Area: 363
Room: Hall, Length: 35, Width: 47, Area: 1645
Room: Bed1, Length: 53, Width: 32, Area: 1696
Room: Bed2, Length: 5, Width: 23, Area: 115
Room: Bed3, Length: 57, Width: 34, Area: 1938


Running with parameters: (100, 75, 0.5, 0.01)


Best Chromosome: (100, 75, 0.5, 0.01)
Room: Living, Length: 010000, Width: 000111
Room: Kitchen, Length: 101100, Width: 010101
Room: Bath, Length: 110011, Width: 101101
Room: Hall, Length: 100111, Width: 011001
Room: Bed1, Length: 000001, Width: 110100
Room: Bed2, Length: 011001, Width: 110001
Room: Bed3, Length: 011110, Width: 100000
Best Floorplan
Room: Living, Length: 16, Width: 7, Area: 112
Room: Kitchen, Length: 44, Width: 21, Area: 924
Room: Bath, Length: 51, Width: 45, Area: 2295
Room: Hall, Length: 39, Width: 25, Area: 975
Room: Bed1, Length: 1, Width: 52, Area: 52
Room: Bed2, Length: 25, Width: 49, Area: 1225
Room: Bed3, Length: 30, Width: 32, Area: 960
```

In [50]:

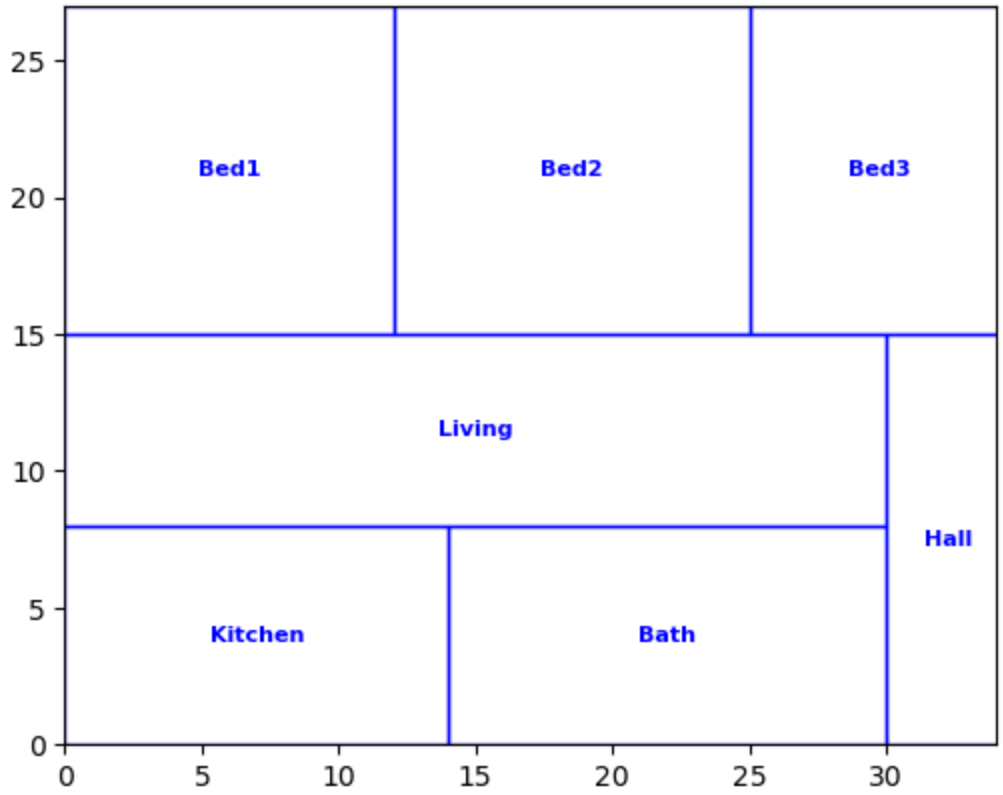
```python
import matplotlib.pyplot as plt


# Create a list of labels for the x-axis (parameter combinations)
labels = [f'Params {i+1}' for i in range(len(parameter_combinations))]

# Create a figure with subplots
fig, axs = plt.subplots(3, 1, figsize=(10, 15))
```

```
# Plot Speed
axs[0].bar(labels, speed_values, color='blue')
axs[0].set_ylabel('Speed (Generations)')
axs[0].set_title('Performance Metrics for Different Parameter Combinations')

# Plot Quality
axs[1].bar(labels, quality_values, color='green')
axs[1].set_ylabel('Quality (Total Area)')

# Plot Reliability
axs[2].bar(labels, reliability_values, color='red')
axs[2].set_ylabel('Reliability')
axs[2].set_xlabel('Parameter Combinations')

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```


Performance Metrics for Different Parameter Combinations
Speed (Generations)
0
50
100
150
200
250
300
Params 1
Params 2
Params 3
Params 4
Params 5
Params 6
Quality (Total Area)
0
1000
2000
3000
4000
5000
6000
7000
8000
Params 1
Params 2
Params 3
Params 4
Params 5
Params 6
Reliability
0.0
0.2
0.4
0.6
0.8
1.0
Params 1
Params 2
Params 3
Params 4
Params 5
Params 6
Parameter Combinations

In [18]:

```
import matplotlib.pyplot as plt

# Define the dimensions of the rooms
room_dimensions = {
    "Living": (30, 7),
    "Kitchen": (14, 8),
    "Bath": (16, 8),
    "Hall": (4.5, 15),
    "Bed1": (12, 12),
    "Bed2": (13, 12),
    "Bed3": (9.5, 12)
}

# Define the positions of the rooms
room_positions = {
    "Living": (0, 8),
    "Kitchen": (0, 0),
    "Bath": (14, 0),
    "Hall": (30, 0),
    "Bed1": (0, 15),
    "Bed2": (12, 15),
    "Bed3": (25, 15)
}


fig, ax = plt.subplots()

# Draw rooms as rectangles
for room, dimensions in room_dimensions.items():
    width, height = dimensions
    x, y = room_positions[room]
    ax.add_patch(plt.Rectangle((x, y), width, height, fill=None, edgecolor='b'))

    # Annotate room names
    ax.annotate(room, (x + width/2, y + height/2), color='b', weight='bold', fontsize=8,

    # Annotate dimensions
    #ax.text(x + width/2, y - 1, f'Length: {width}', color='r', weight='bold', fontsize=
    #ax.text(x - 1, y + height/2, f'Width: {height}', color='r', weight='bold', fontsize

# Draw corridors (example positions, you'll need to define them)
#corridor_positions = [((5, 5), (5, 15)), ((15, 5), (15, 15)), ((0, 5), (30, 5)), ((0, 1
#for (x1, y1), (x2, y2) in corridor_positions:
    #ax.plot([x1, x2], [y1, y2], color='w')

# Set axis limits
ax.set_xlim(0, 34)
ax.set_ylim(0, 27)


# Set aspect of the plot to be equal, so squares look like squares
ax.set_aspect('equal', 'box')

# Show the plot
plt.show()
```



In [ ]: