Planning in Continuous Spaces: Motion Planning

Tom Silver
Machine Learning for Robot Planning
Princeton University
Fall 2025

Recap and Preview

Earlier:

- Planning in finite "tabular" state and action spaces
- Careful treatment of uncertainty in transitions and observations
- Offline planning and online planning

Last Time:

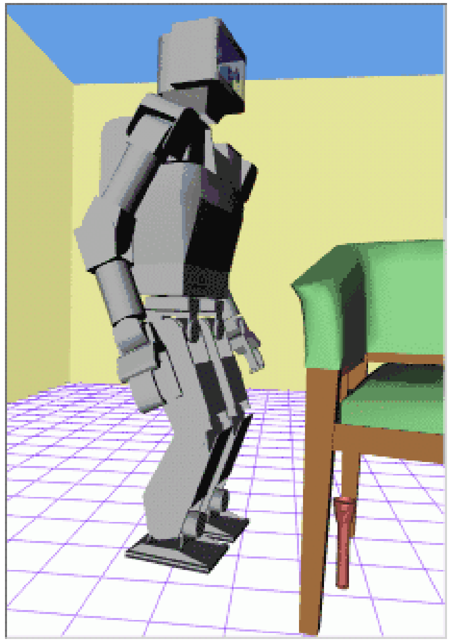
- Planning in finite "factored" state and action spaces
- No more uncertainty
- Online planning only

This Time:

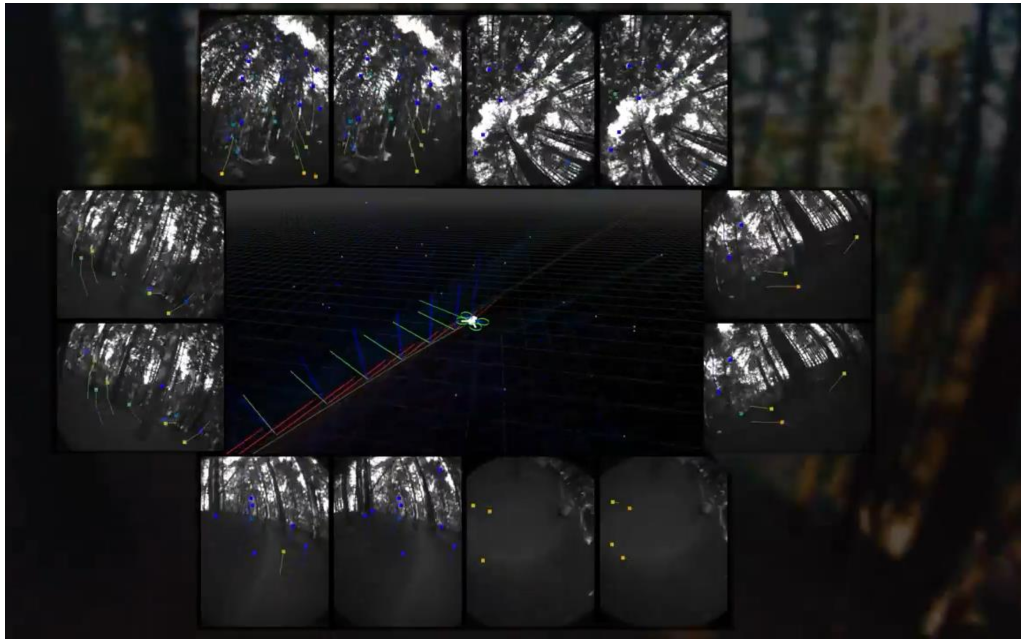
- Planning in continuous spaces

But with a very specific structure

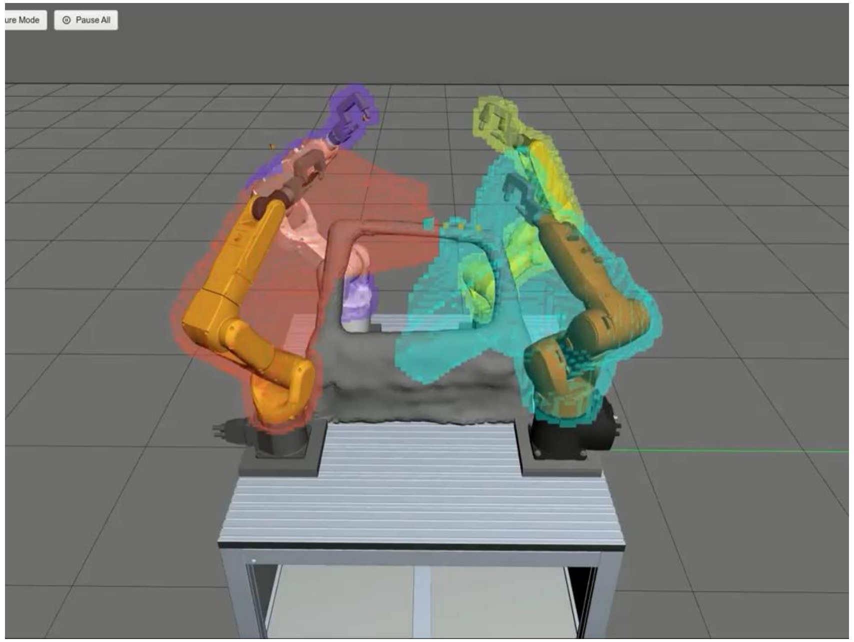
Motion Planning



Kuffner (2002)



Skydio (2019)



Realtime Robotics (2023)

Motion Planning Problem Setting

A motion planning problem includes:

- A *configuration space* $\mathcal{X}$

Must be bounded & have distance metric & have other nice properties…

- An initial configuration $x_0 \in \mathcal{X}$
- A goal configuration $x_g \in \mathcal{X}$

Alternative definition: *set* of configurations

- A *feasibility check* $f: \mathcal{X} \to \{\mathrm{T}, \mathrm{F}\}$

Usually: feasible (T) if robot not in collision

Motion Planning Solution

A *solution* to a motion planning problem is a trajectory

$$\alpha : [0, H] \rightarrow \mathcal{X}$$

where

1. $\alpha(0) = x_0$
2. $\alpha(H) = x_g$
3. The robot can follow the trajectory

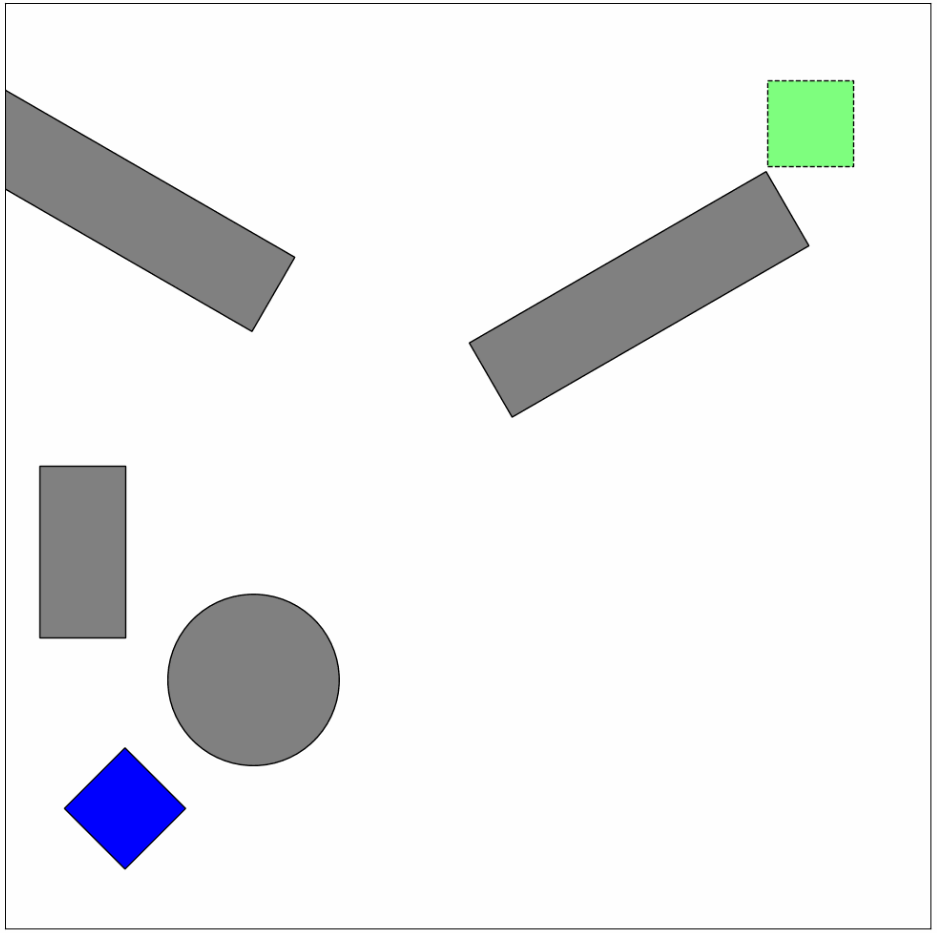
Continuous states *and* continuous time!

Many different ways to specify this

Actions are now implicit

Example: Motion Planning with 2D Shapes

- Configuration space:
  - (x position, y position, rotation)
  - x and y position are bounded
  - Subset of SE(2)
- Initial configuration: see image
- Goal configuration: see image
- Feasibility check:
  - Configuration is feasible if robot is not in collision with obstacles



A Stupidest Possible Algorithm

1. Discretize the configuration space

2. Run path planning (e.g., A* with distance-to-goal heuristic)

When will this go badly?

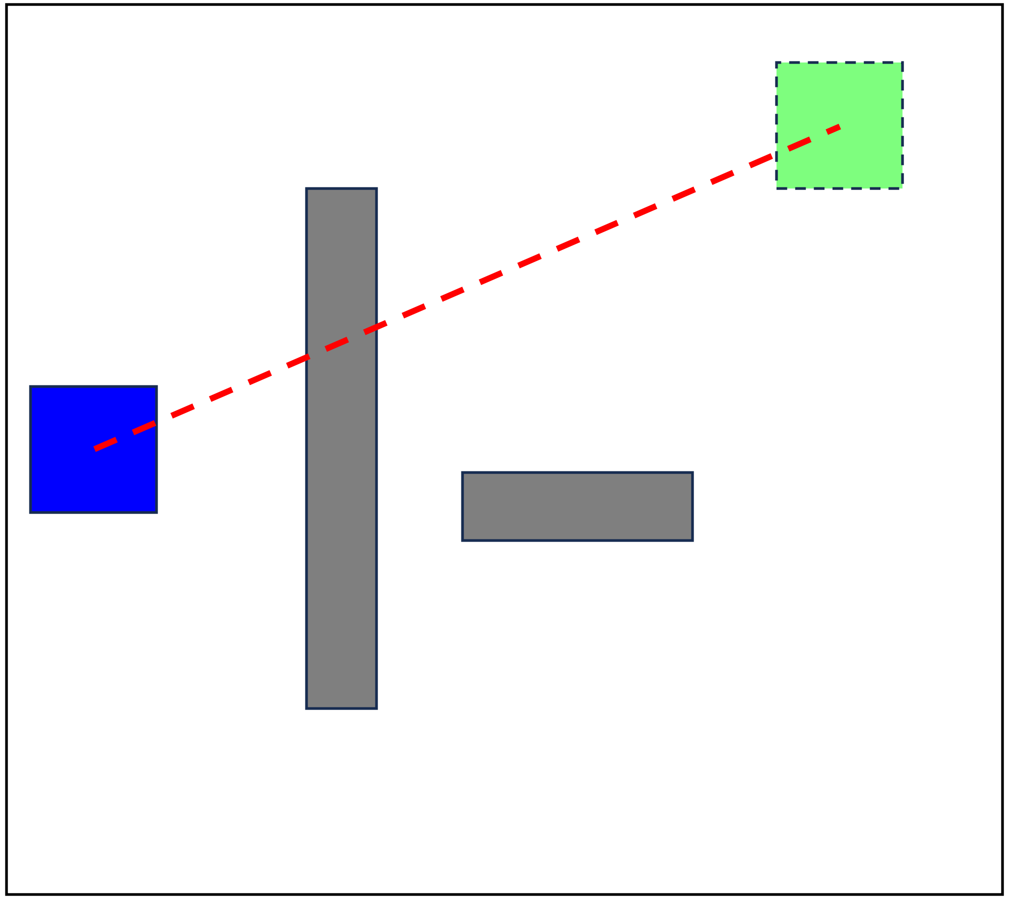
Is this sound, complete, optimal?

A "Bug Algorithm"

At each time step:

1. If can move directly towards goal, do so

2. Otherwise, move clockwise around obstacle

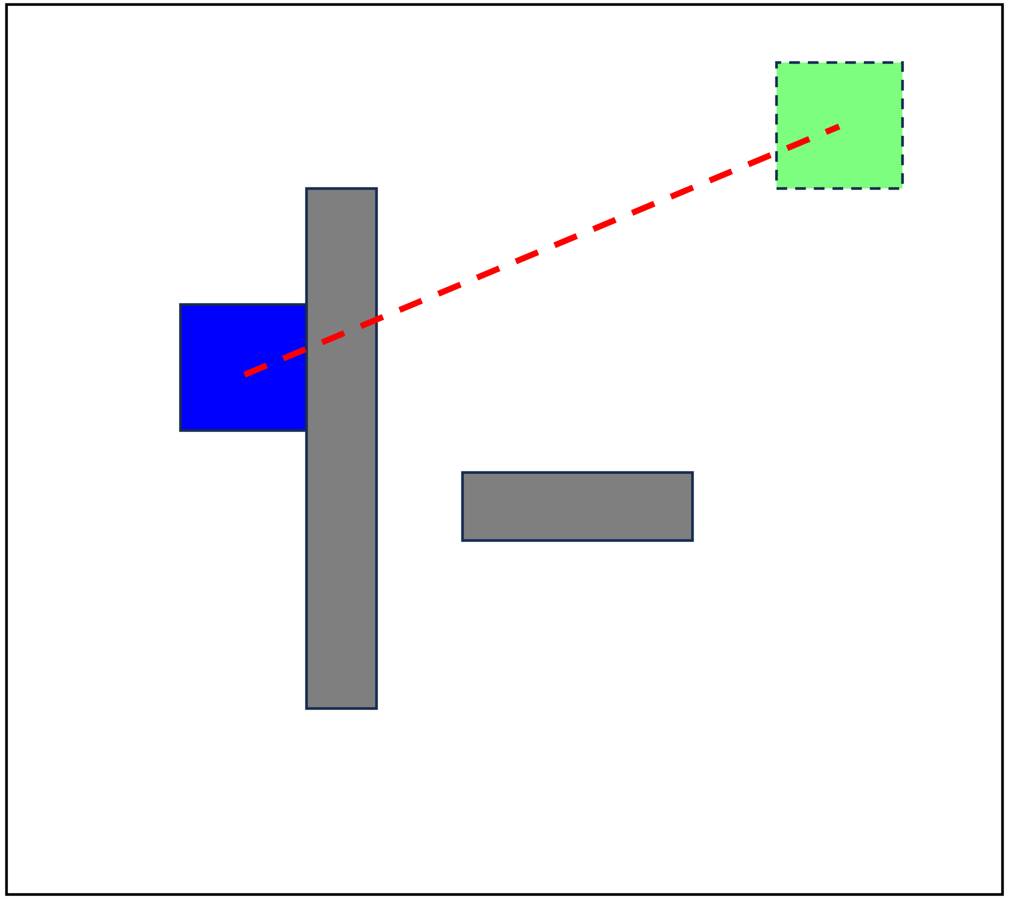


A “Bug Algorithm”

At each time step:

1. **If can move directly towards goal, do so**

2. Otherwise, move clockwise around obstacle

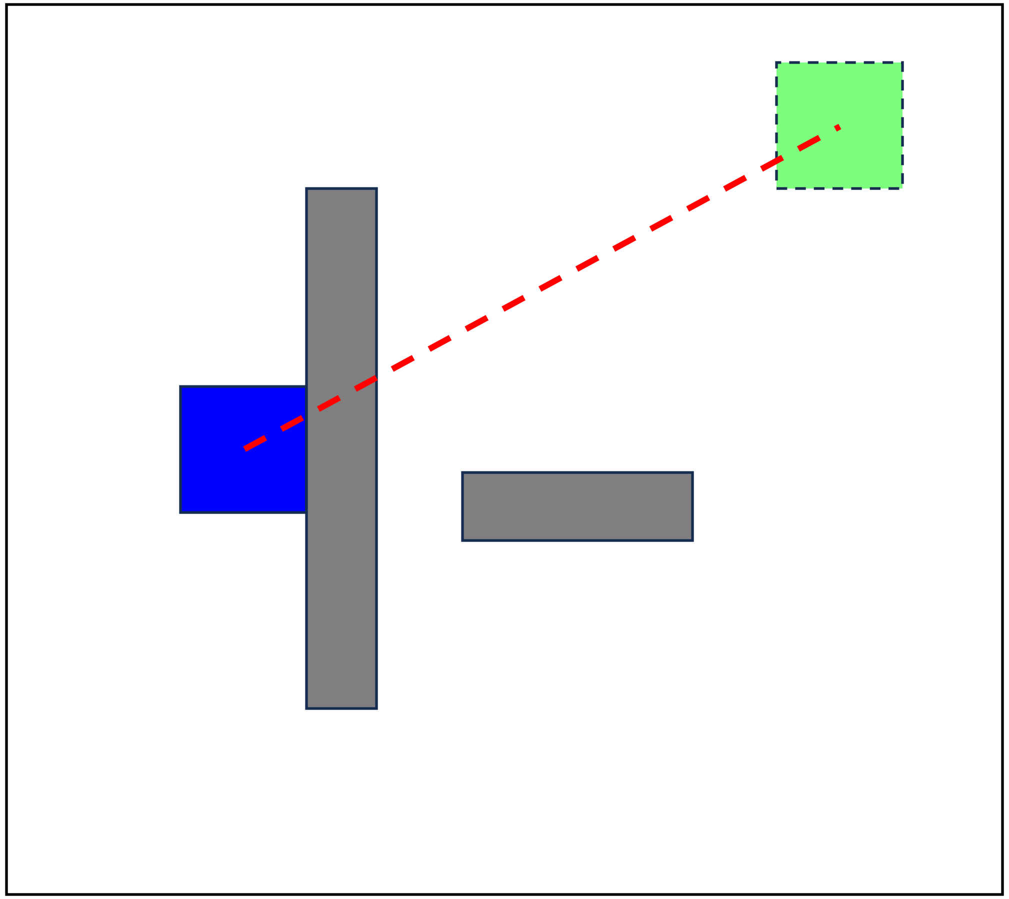


A "Bug Algorithm"

At each time step:

1. If can move directly towards goal, do so

2. **Otherwise, move clockwise around obstacle**

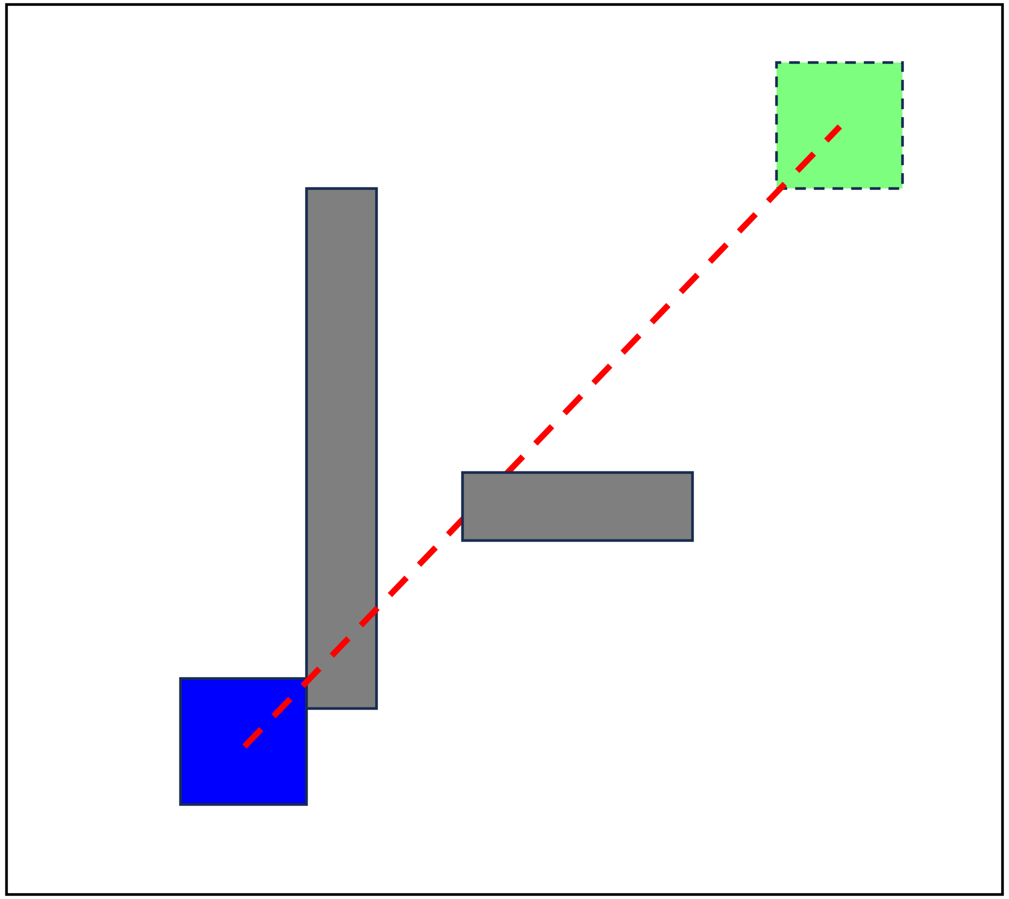


A "Bug Algorithm"

At each time step:

1. If can move directly towards goal, do so

2. **Otherwise, move clockwise around obstacle**

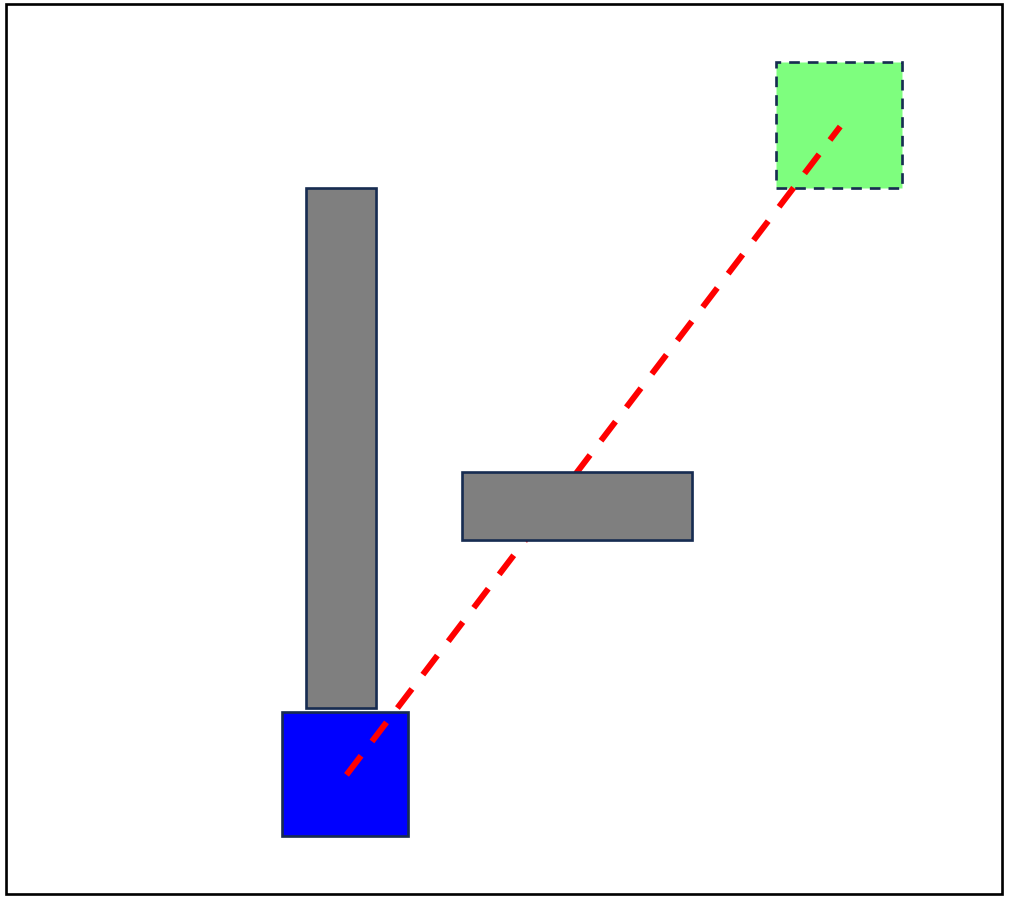


A "Bug Algorithm"

At each time step:

1. If can move directly towards goal, do so

2. **Otherwise, move clockwise around obstacle**

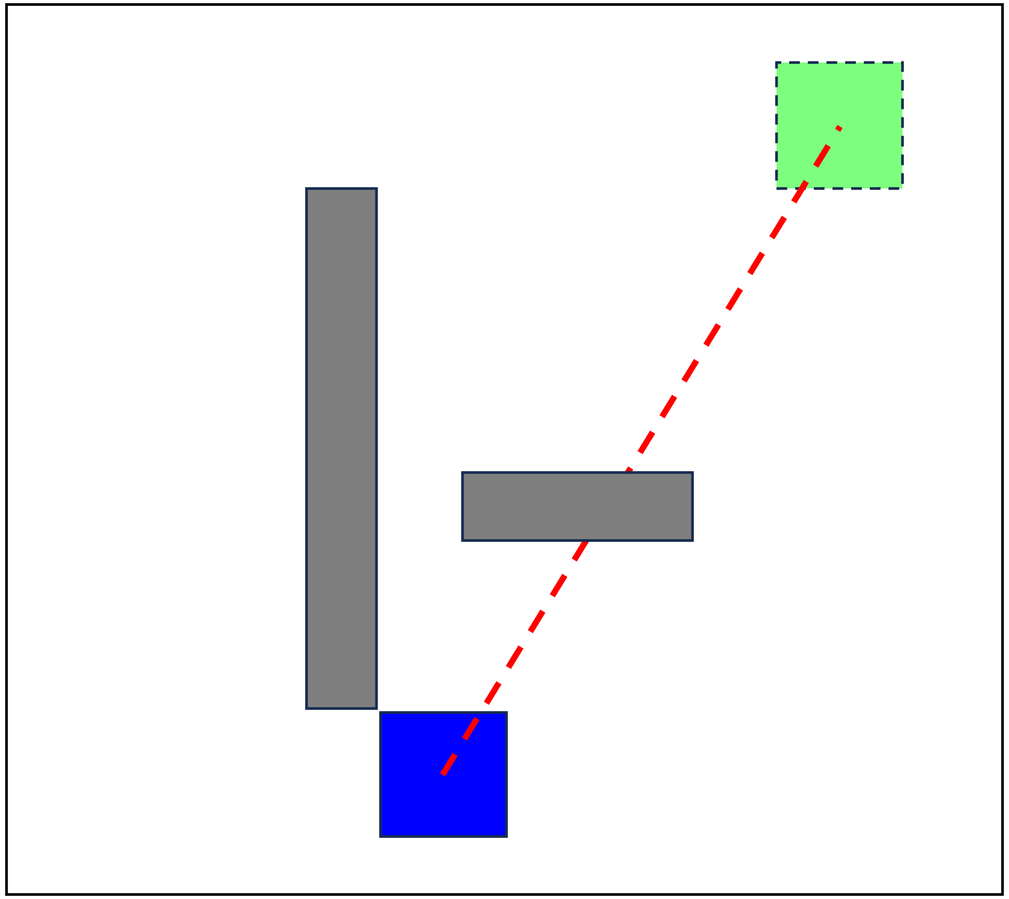


A "Bug Algorithm"

At each time step:

1. If can move directly towards goal, do so

2. **Otherwise, move clockwise around obstacle**

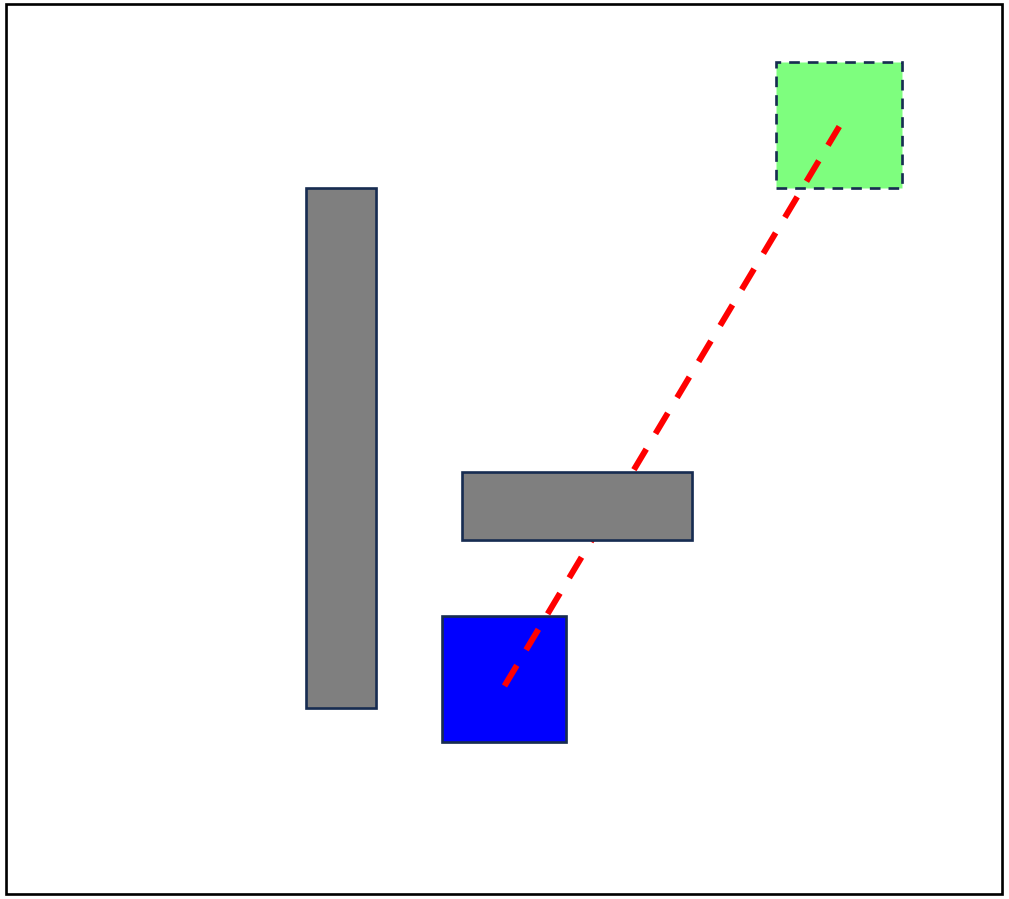


A "Bug Algorithm"

At each time step:

1. **If can move directly towards goal, do so**

2. Otherwise, move clockwise around obstacle

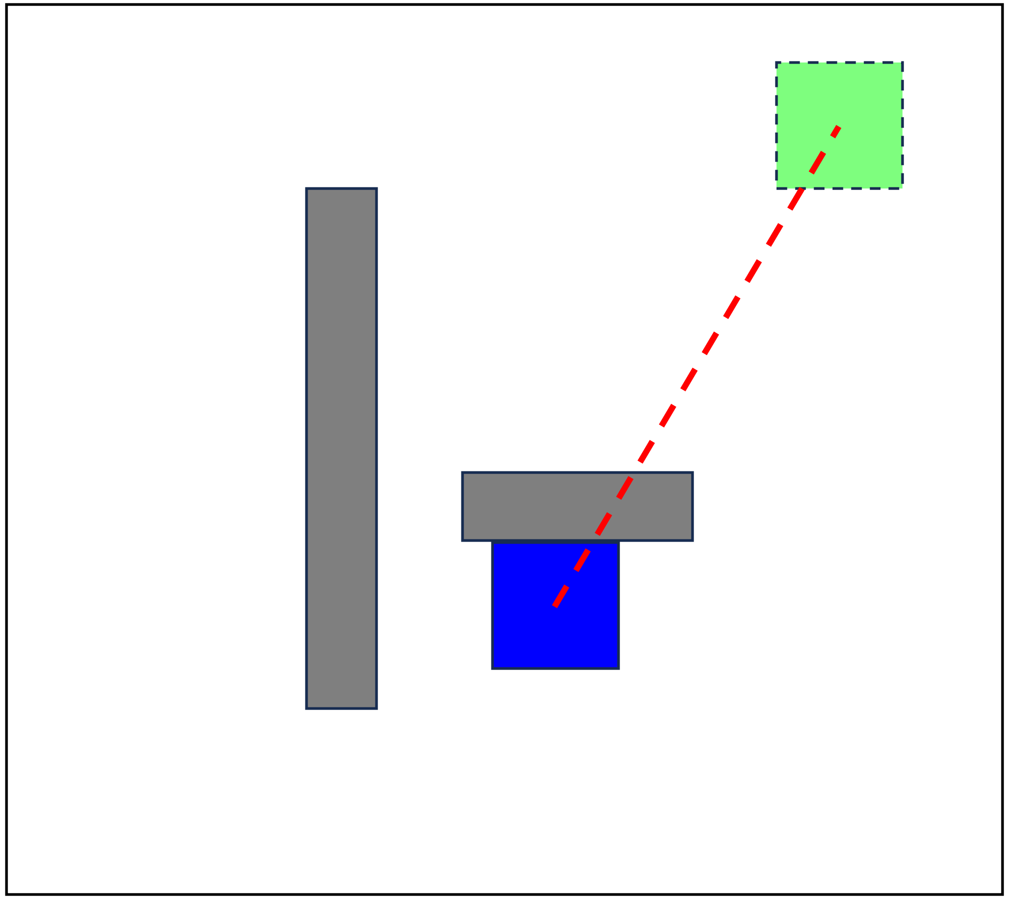


A “Bug Algorithm”

At each time step:

1. **If can move directly towards goal, do so**

2. Otherwise, move clockwise around obstacle

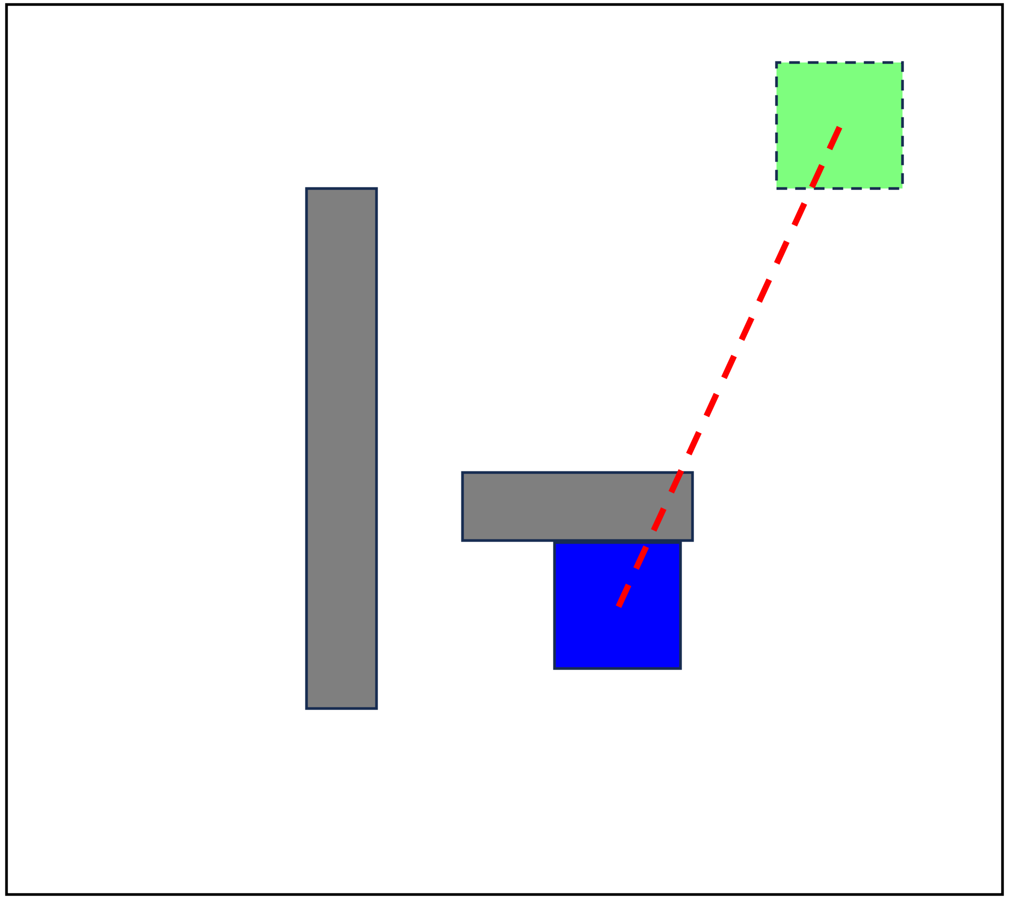


A “Bug Algorithm”

At each time step:

1. If can move directly towards goal, do so

2. **Otherwise, move clockwise around obstacle**

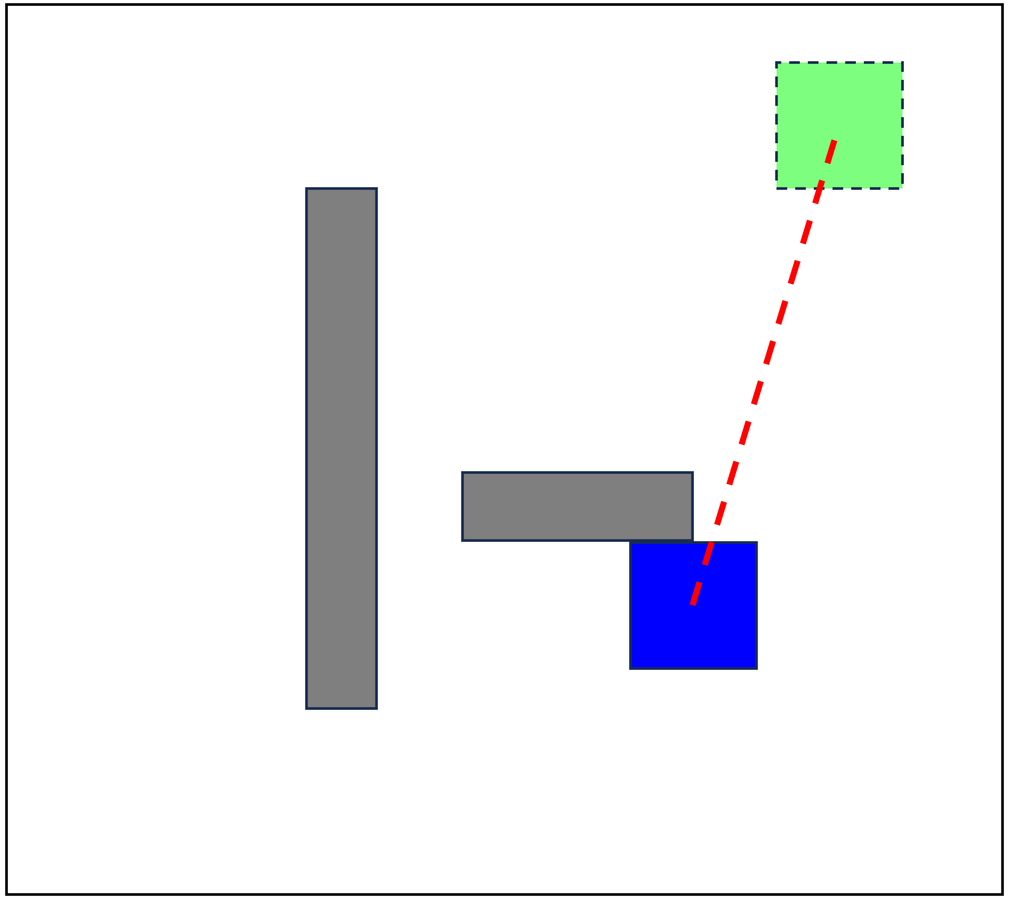


A "Bug Algorithm"

At each time step:

1. If can move directly towards goal, do so

2. **Otherwise, move clockwise around obstacle**

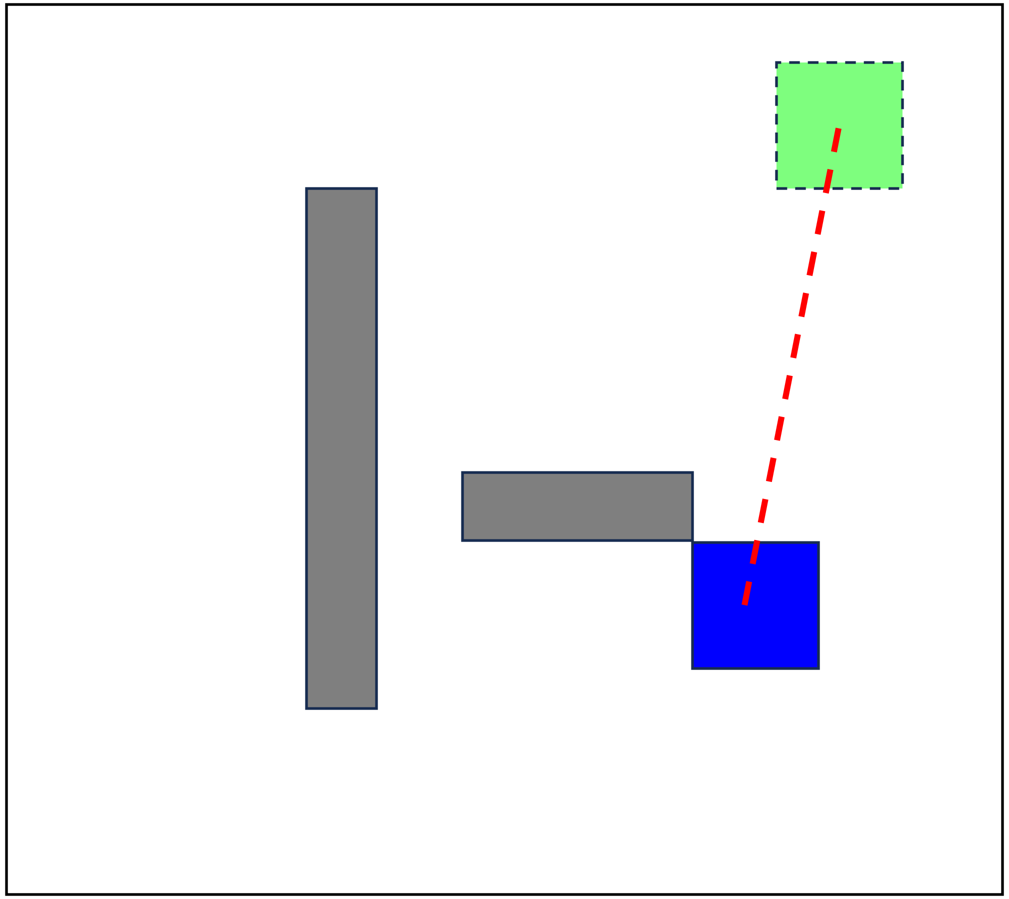


A "Bug Algorithm"

At each time step:

1. If can move directly towards goal, do so

2. **Otherwise, move clockwise around obstacle**

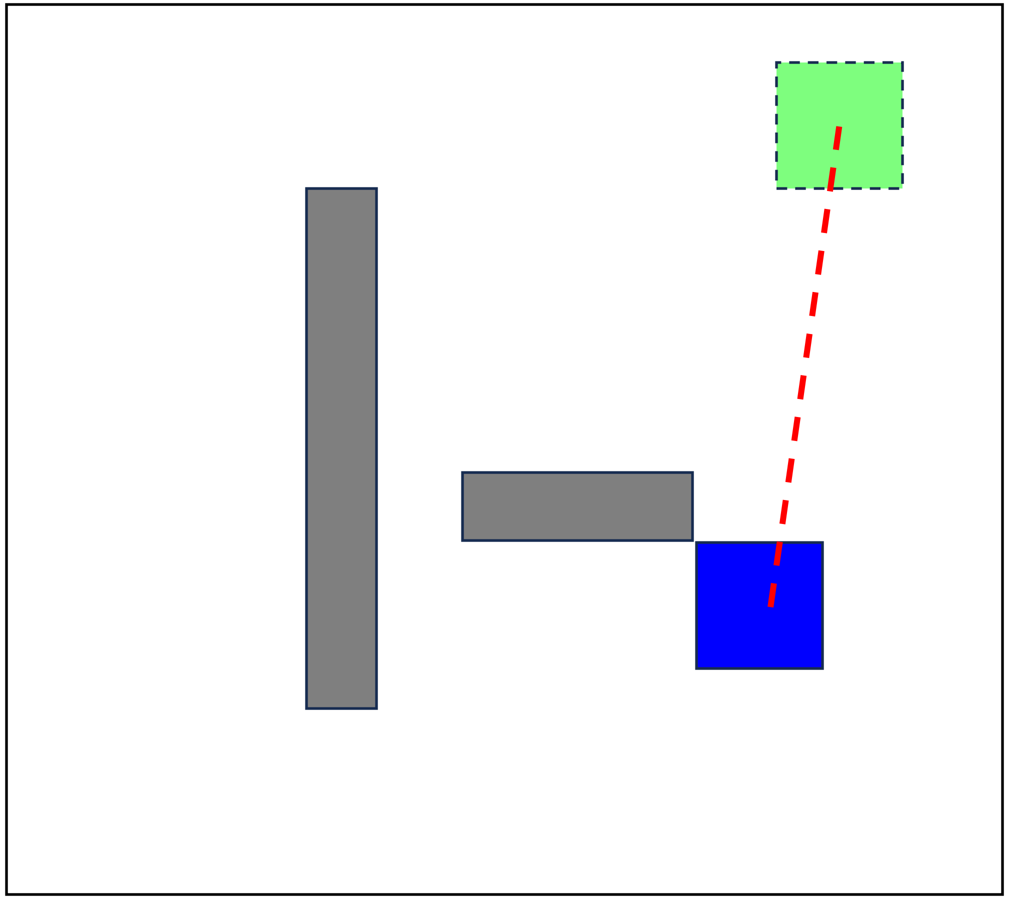


A "Bug Algorithm"

At each time step:

1. **If can move directly towards goal, do so**

2. Otherwise, move clockwise around obstacle

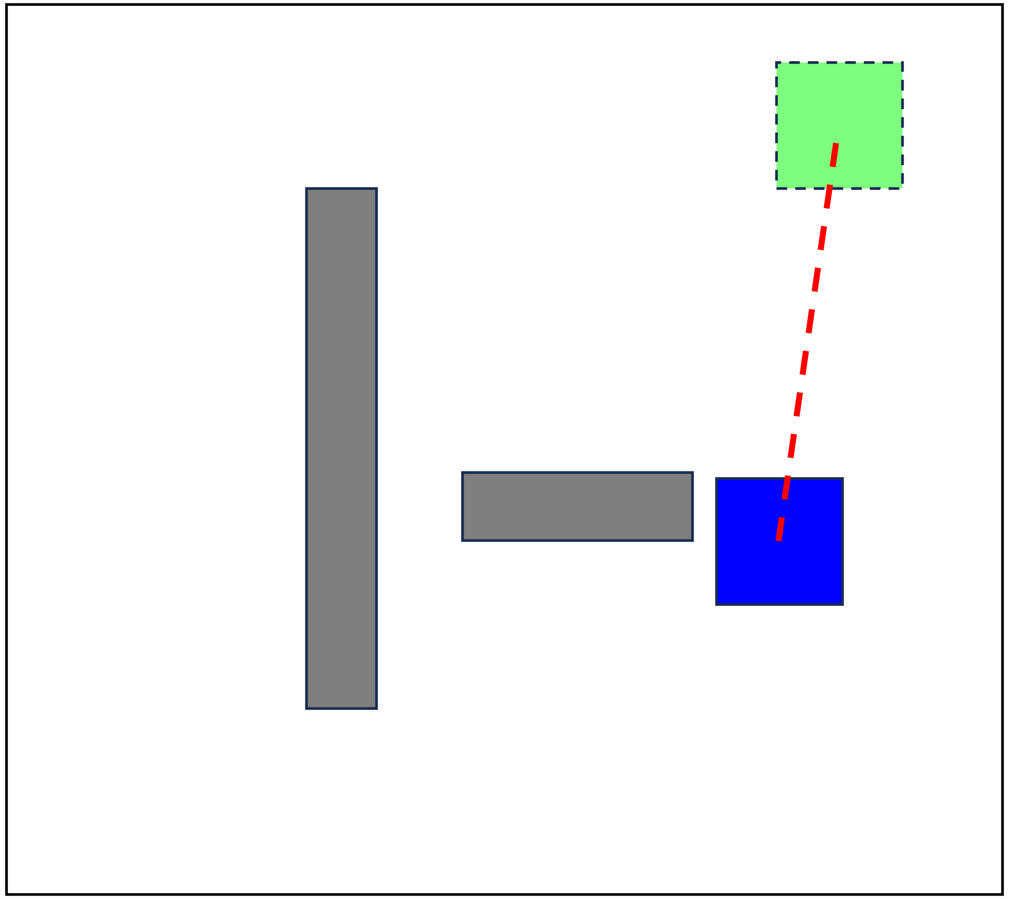


A "Bug Algorithm"

At each time step:

1. **If can move directly towards goal, do so**

2. Otherwise, move clockwise around obstacle

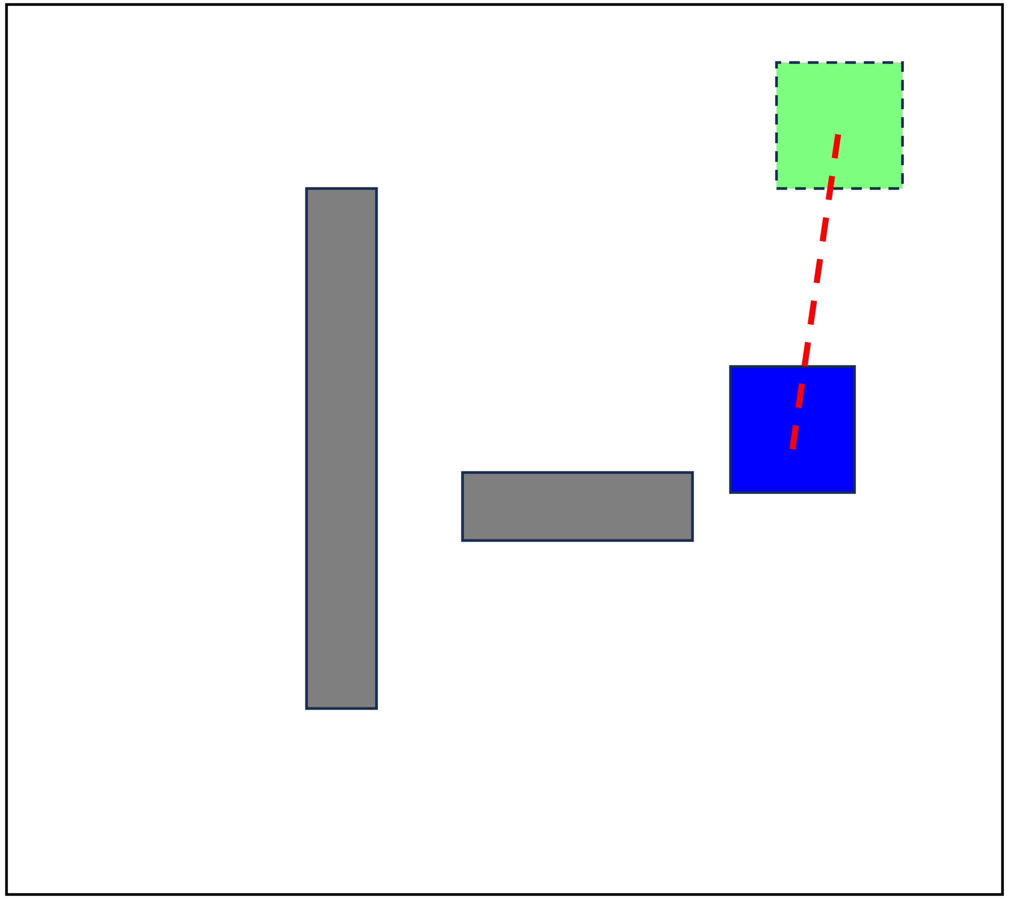


A "Bug Algorithm"

At each time step:

1. **If can move directly towards goal, do so**

2. Otherwise, move clockwise around obstacle

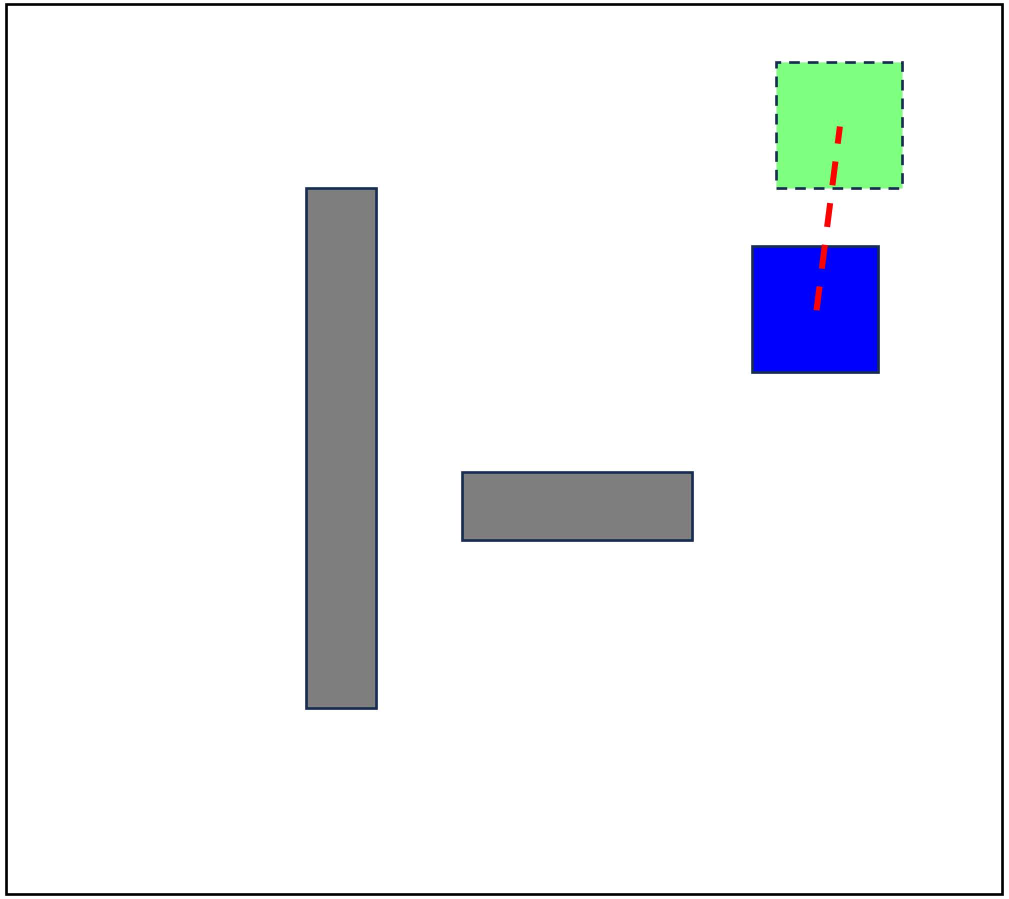


A “Bug Algorithm”

At each time step:

1. **If can move directly towards goal, do so**

2. Otherwise, move clockwise around obstacle

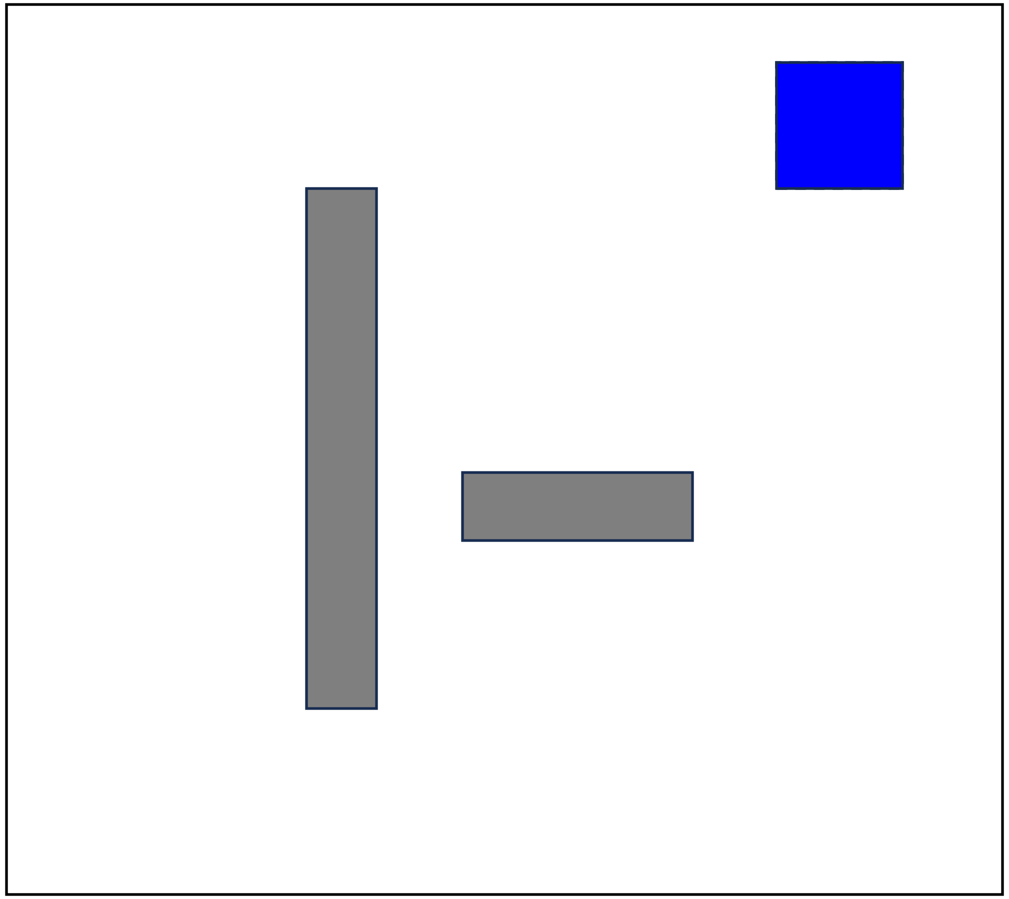


A "Bug Algorithm"

At each time step:

1. **If can move directly towards goal, do so**

2. Otherwise, move clockwise around obstacle



A "Bug Algorithm"

At each time step:

1. If can move directly towards goal, do so

2. Otherwise, move clockwise around obstacle

Show that this algorithm is not complete

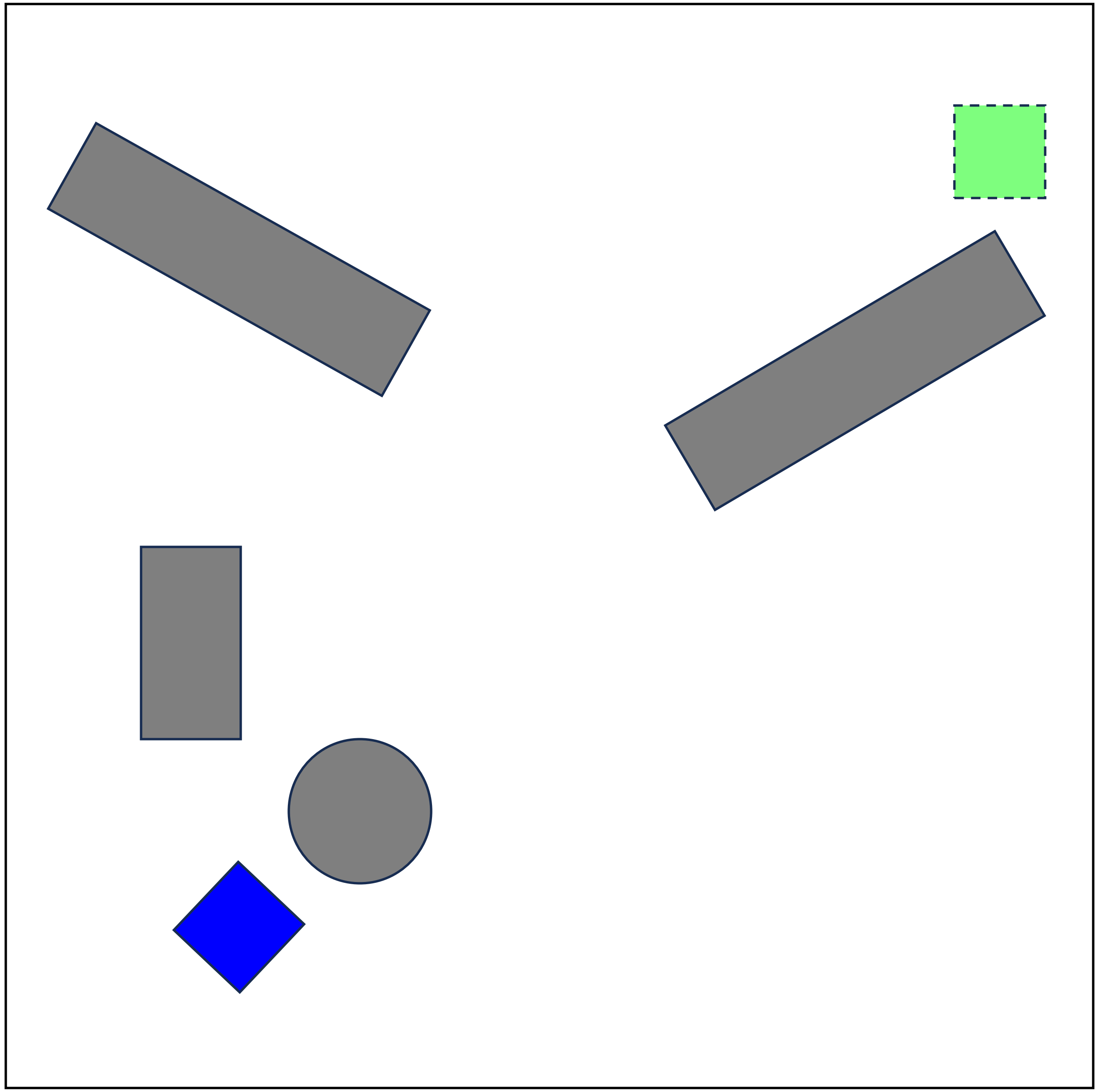
There are complete bug algorithms

Sampling-Based Motion Planning

Let’s assume we can sample from the configuration space

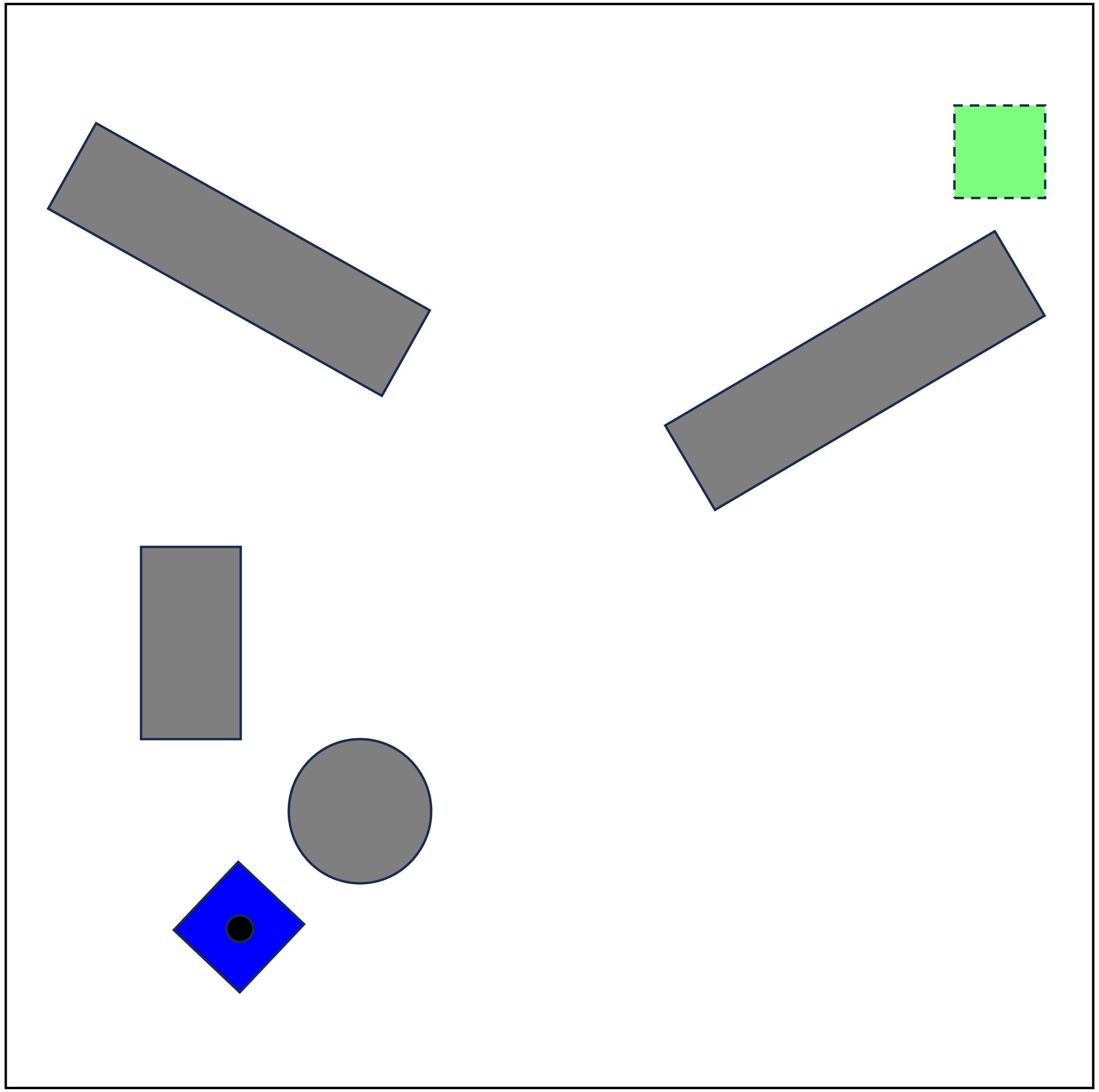
Rapidly Exploring Random Trees (RRT)

```
RRT(x_0, x_g, 𝒳, f)
// Initialize tree at x_0
nodes = [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        x_target = x_g
    else
        // Sample a target configuration
        x_target = sample(𝒳)
    // Extend the tree towards the target
    node = getClosest(nodes, x_target)
    x_node = node.conf
    for x in extend(x_node, x_target)
        if f(x):
            nodes.add(Node(x))
            if x = x_g:
                return finish(nodes)
        else
            break
```



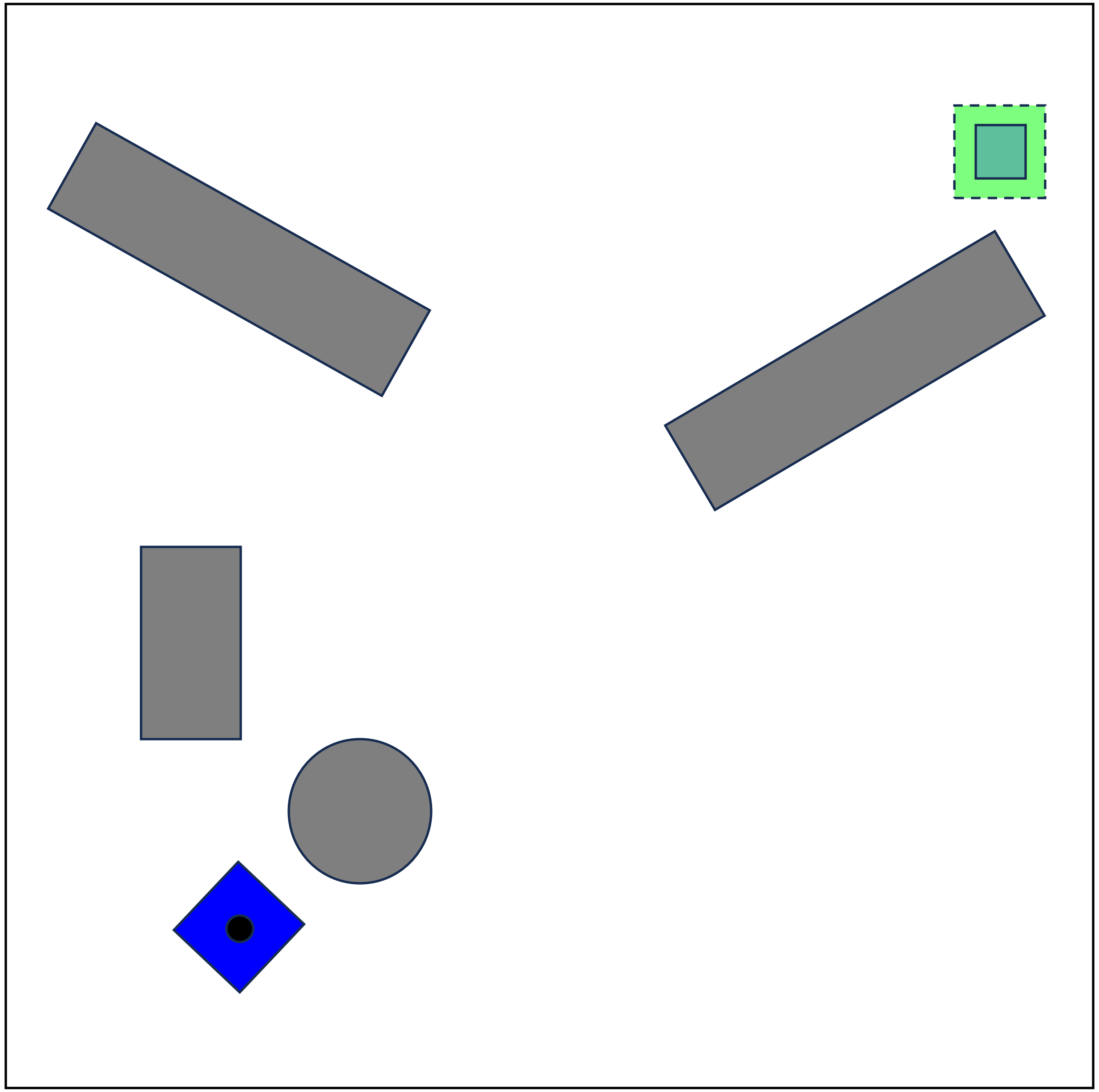
RRT(x_0, x_g, $\mathcal{X}$, f)

// Initialize tree at x_0
`nodes` $=$ [`Node`(x_0)]
repeat:
    if uniform() < `goalSampleProb`
        // Try to go directly to the goal
        $x_{\text{target}} = x_g$
    else
        // Sample a target configuration
        $x_{\text{target}} =$ `sample`($\mathcal{X}$)
    // Extend the tree towards the target
    `node` $=$ `getClosest`(`nodes`, x_{target})
    $x_{\text{node}} =$ `node.conf`
    for x **in** `extend`(x_{node}, x_{target})
        if $f(x)$:
            `nodes.add`(`Node`(x))
            if $x = x_g$:
                return `finish(nodes)`
        else
            break

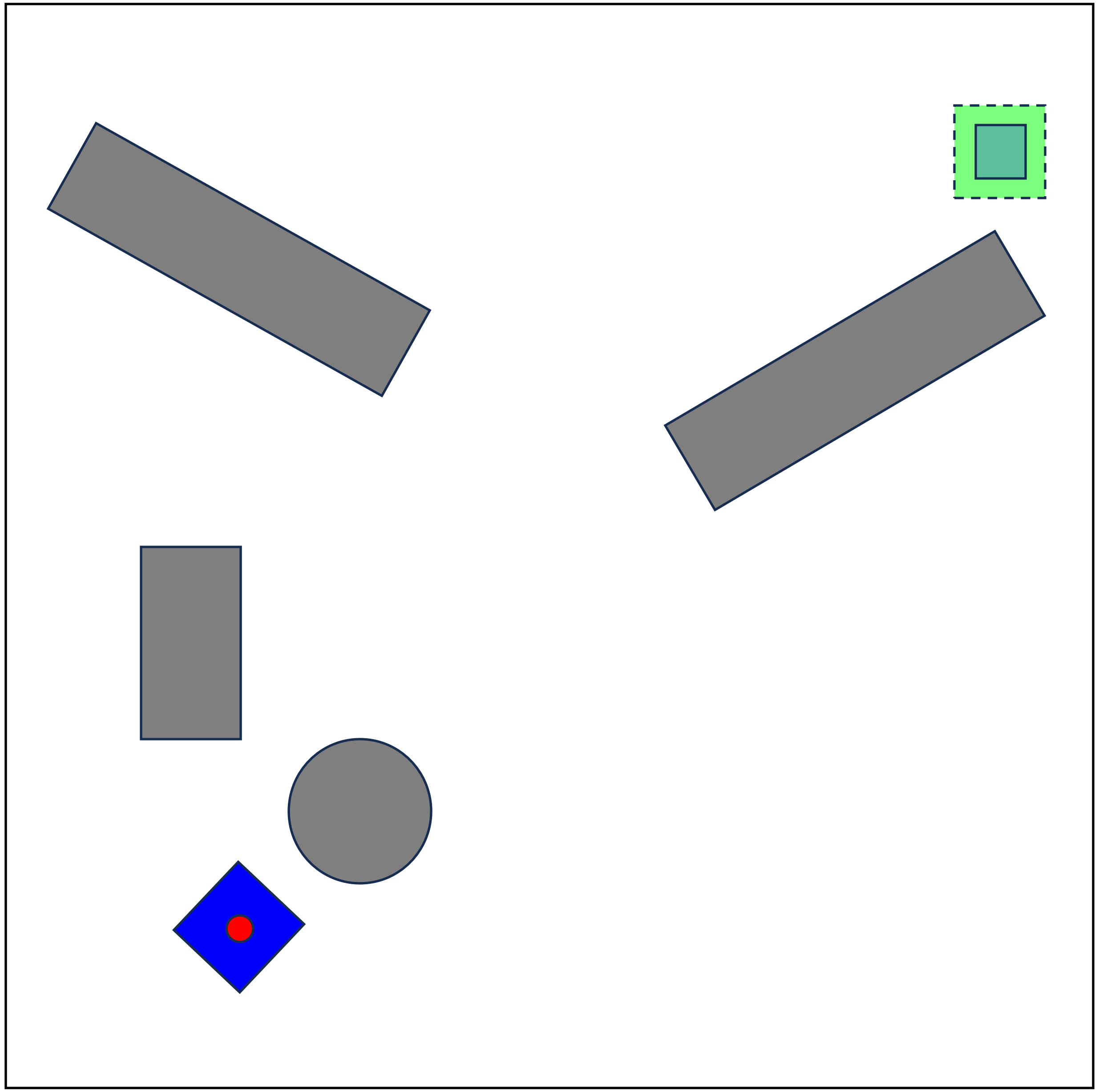


RRT$(x_0, x_g, \mathcal{X}, f)$

// Initialize tree at x_0
`nodes` $=$ [`Node`(x_0)]
repeat:
    if uniform() < `goalSampleProb`
        // Try to go directly to the goal
        $x_{\text{target}} = x_g$
    else
        // Sample a target configuration
        $x_{\text{target}} =$ `sample`$(\mathcal{X})$
    // Extend the tree towards the target
    `node` $=$ `getClosest`(`nodes`, x_{target})
    $x_{\text{node}} =$ `node.conf`
    for x **in** `extend`$(x_{\text{node}}, x_{\text{target}})$
        if $f(x)$:
            `nodes.add`(`Node`(x))
            if $x = x_g$:
                return `finish(nodes)`
        else
            break

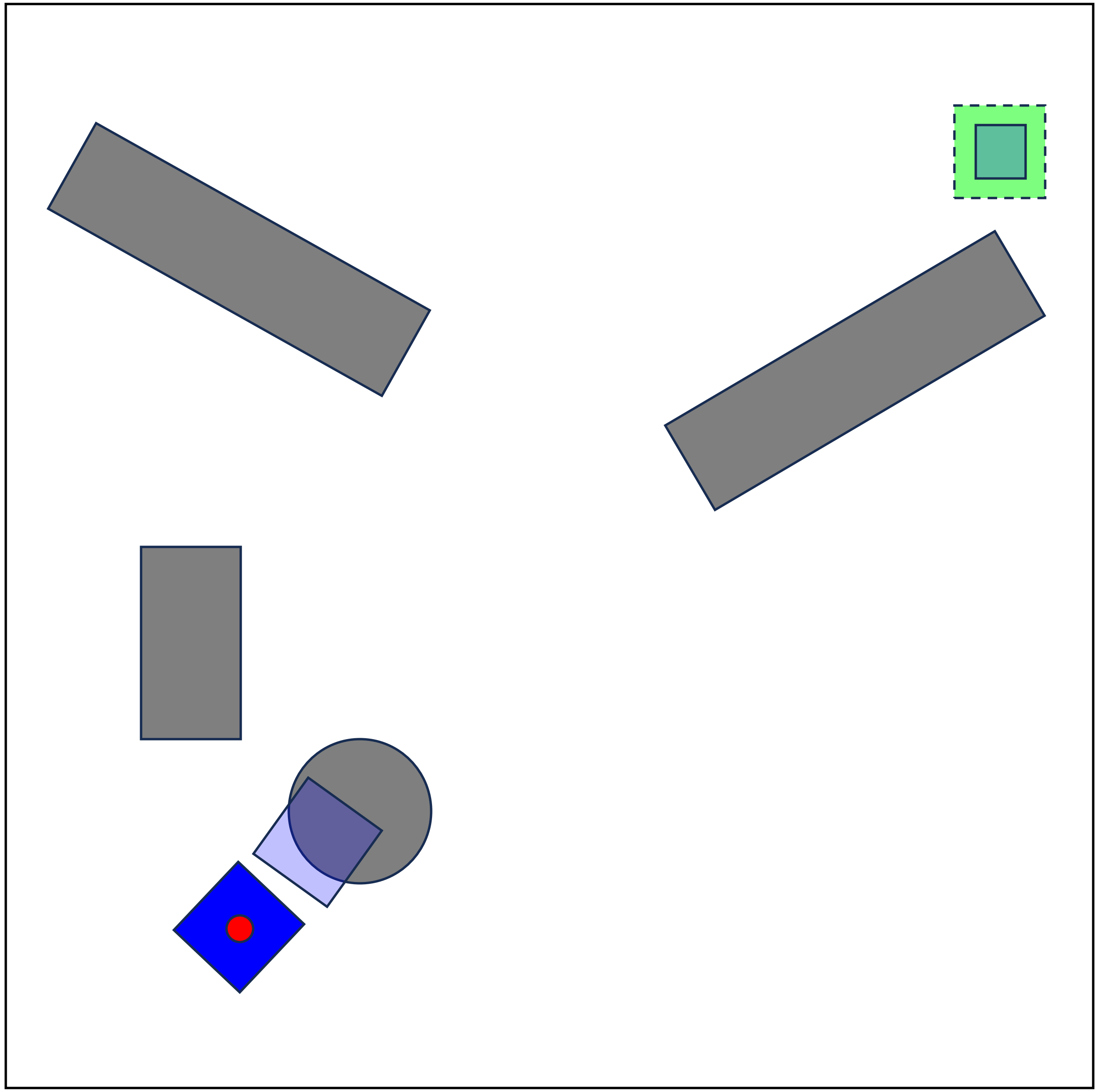


```
RRT(x_0, x_g, 𝒳, f)
    // Initialize tree at x_0
    nodes = [Node(x_0)]
    repeat:
        if uniform() < goalSampleProb
            // Try to go directly to the goal
            x_target = x_g
        else
            // Sample a target configuration
            x_target = sample(𝒳)
        // Extend the tree towards the target
        node = getClosest(nodes, x_target)
        x_node = node.conf
        for x in extend(x_node, x_target)
            if f(x):
                nodes.add(Node(x))
                if x = x_g:
                    return finish(nodes)
            else
                break
```

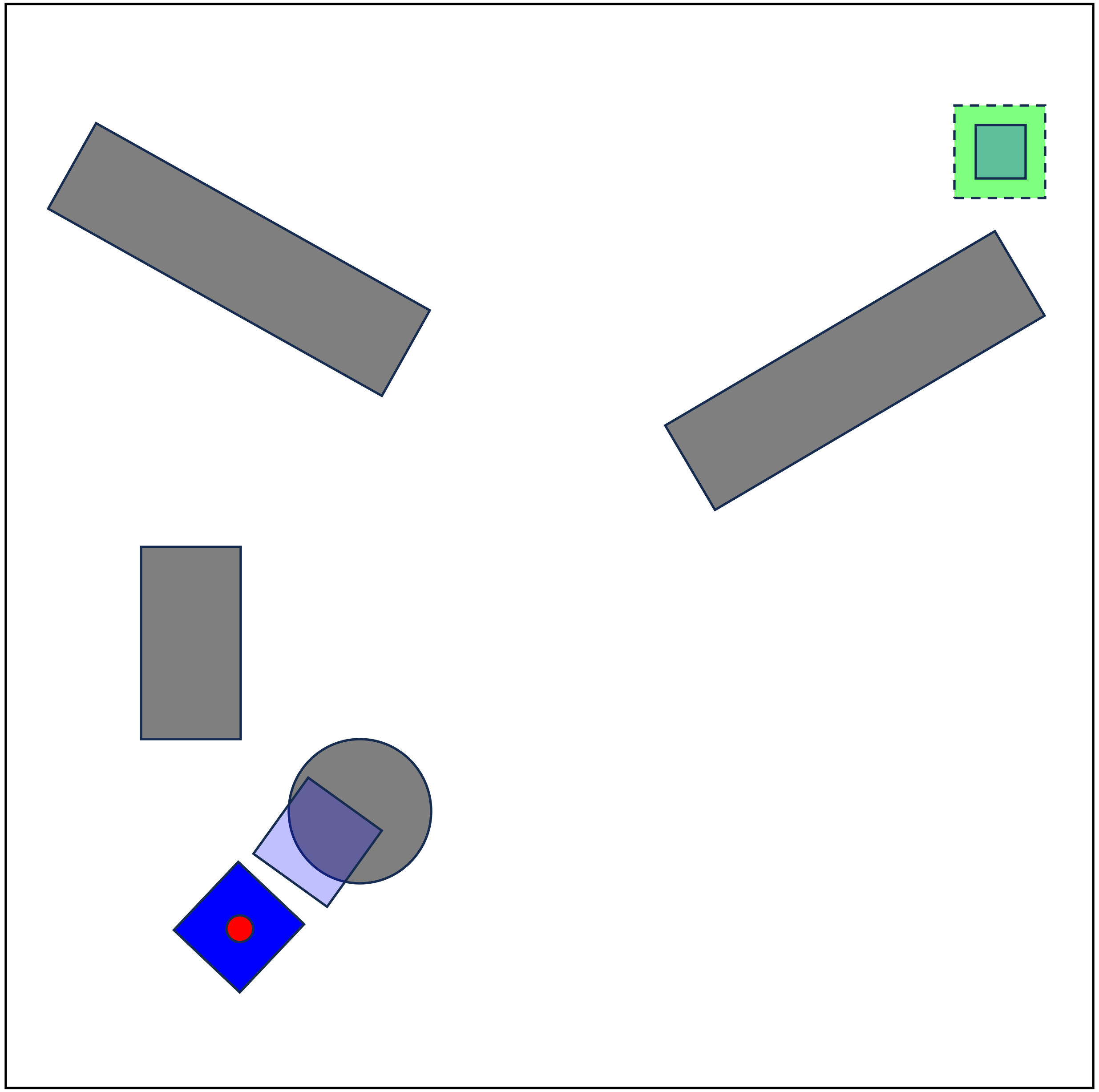


RRT$(x_0, x_g, \mathcal{X}, f)$

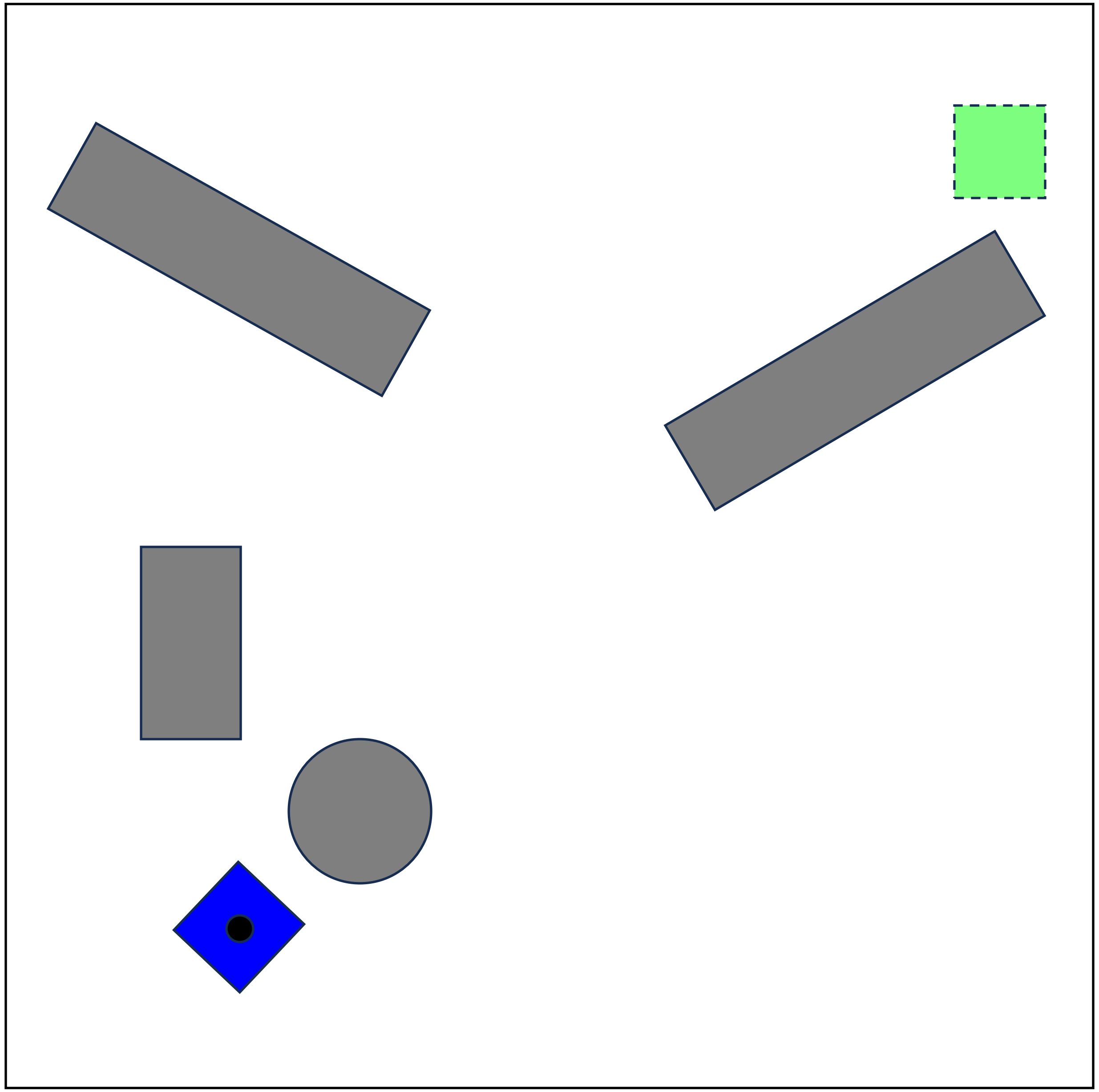
// Initialize tree at x_0
nodes $=$ [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        $x_{\text{target}} = x_g$
    else
        // Sample a target configuration
        $x_{\text{target}} =$ sample$(\mathcal{X})$
    // Extend the tree towards the target
    node $=$ getClosest(nodes, x_{target})
    $x_{\text{node}} =$ node.conf
    for x **in** extend$(x_{\text{node}}, x_{\text{target}})$
        if $f(x)$:
            nodes.add(Node(x))
            if $x = x_g$:
                return finish(nodes)
        else
            break



```
RRT(x_0, x_g, 𝒳, f)
// Initialize tree at x_0
nodes = [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        x_target = x_g
    else
        // Sample a target configuration
        x_target = sample(𝒳)
    // Extend the tree towards the target
    node = getClosest(nodes, x_target)
    x_node = node.conf
    for x in extend(x_node, x_target)
        if f(x):
            nodes.add(Node(x))
            if x = x_g:
                return finish(nodes)
        else
            break
```

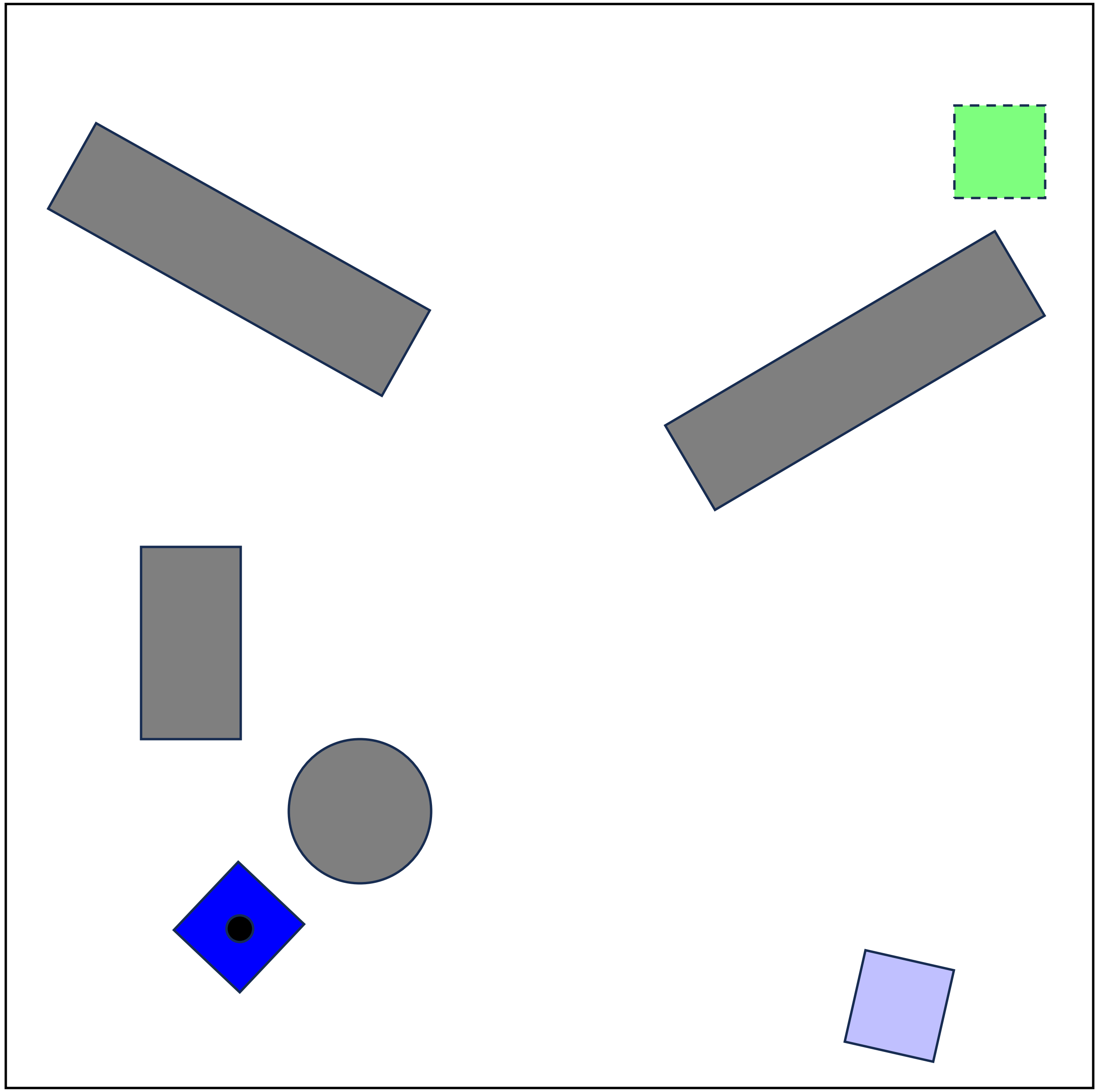


```
RRT(x_0, x_g, 𝒳, f)
// Initialize tree at x_0
nodes = [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        x_target = x_g
    else
        // Sample a target configuration
        x_target = sample(𝒳)
    // Extend the tree towards the target
    node = getClosest(nodes, x_target)
    x_node = node.conf
    for x in extend(x_node, x_target)
        if f(x):
            nodes.add(Node(x))
            if x = x_g:
                return finish(nodes)
        else
            break
```



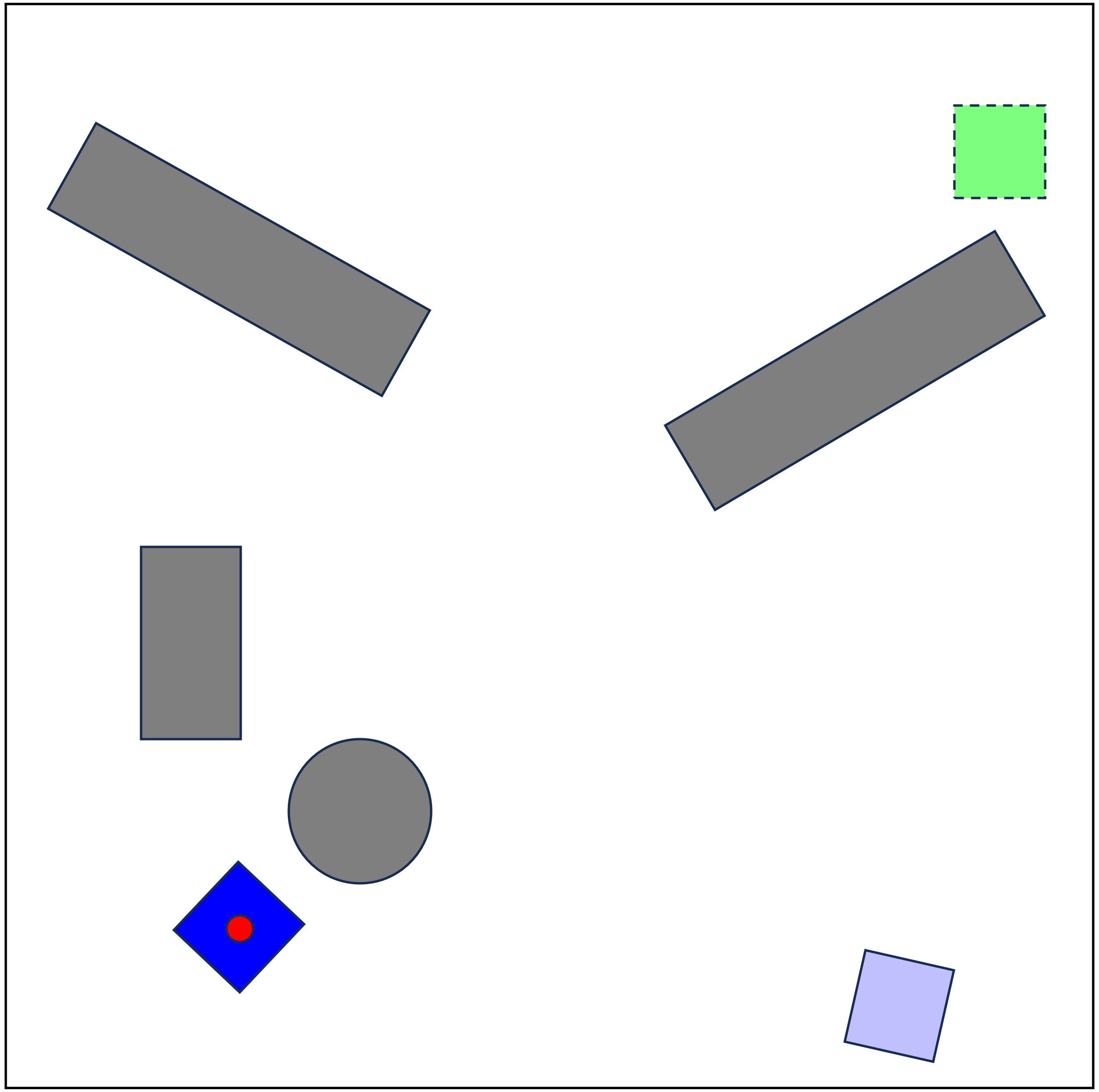
RRT(x_0, x_g, $\mathcal{X}$, f)

// Initialize tree at x_0
nodes $=$ [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        $x_{\text{target}} = x_g$
    else
        // Sample a target configuration
        $x_{\text{target}} =$ sample($\mathcal{X}$)
    // Extend the tree towards the target
    node $=$ getClosest(nodes, x_{target})
    $x_{\text{node}} =$ node.conf
    for x **in** extend(x_{node}, x_{target})
        if $f(x)$:
            nodes.add(Node(x))
            if $x = x_g$:
                return finish(nodes)
        else
            break



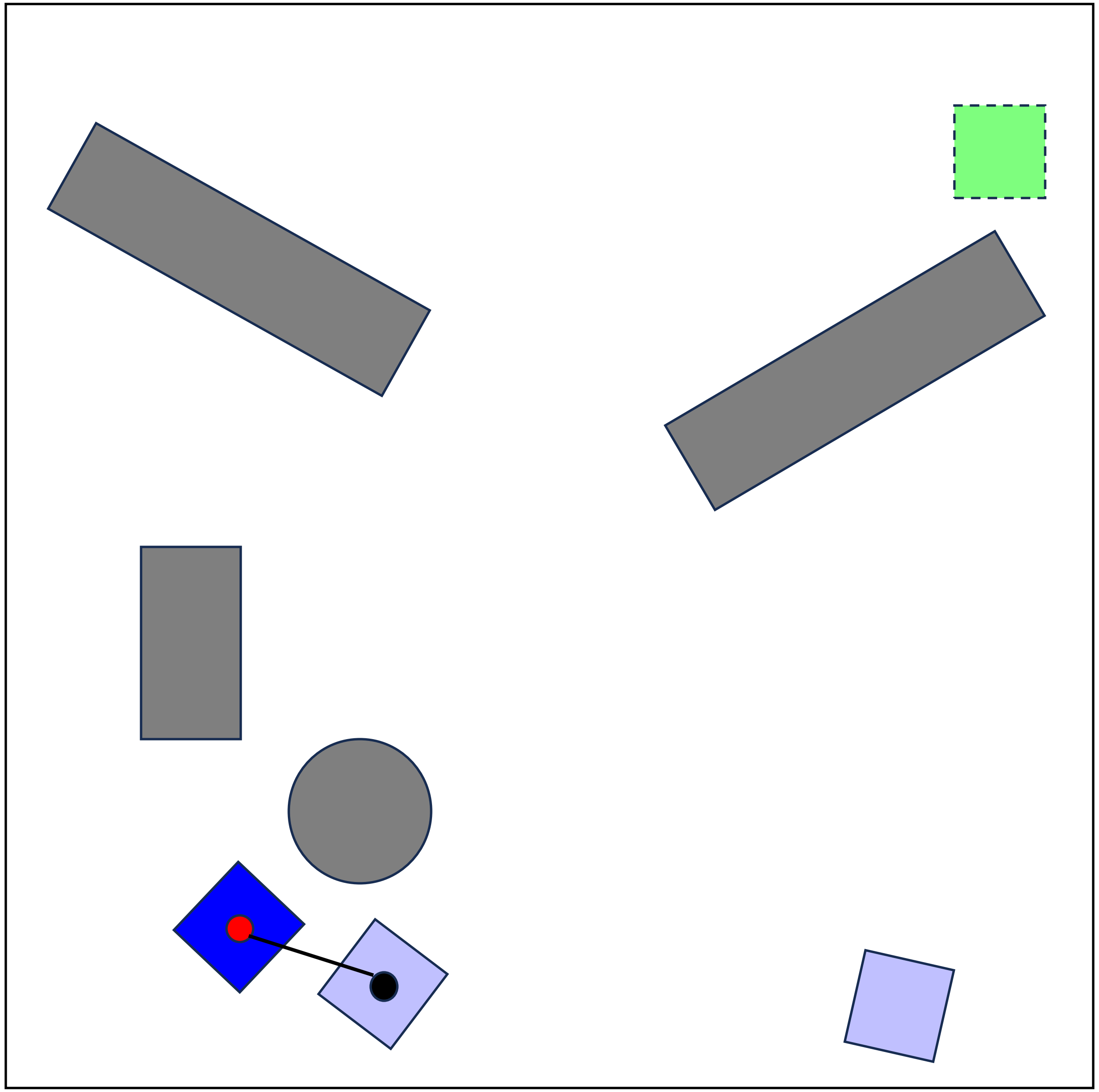
RRT(x_0, x_g, $\mathcal{X}$, f)

// Initialize tree at x_0
nodes $=$ [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        $x_{\text{target}} = x_g$
    else
        // Sample a target configuration
        $x_{\text{target}} =$ sample($\mathcal{X}$)
    // Extend the tree towards the target
    node $=$ getClosest(nodes, x_{target})
    $x_{\text{node}} =$ node.conf
    for x **in** extend(x_{node}, x_{target})
        if $f(x)$:
            nodes.add(Node(x))
            if $x = x_g$:
                return finish(nodes)
        else
            break

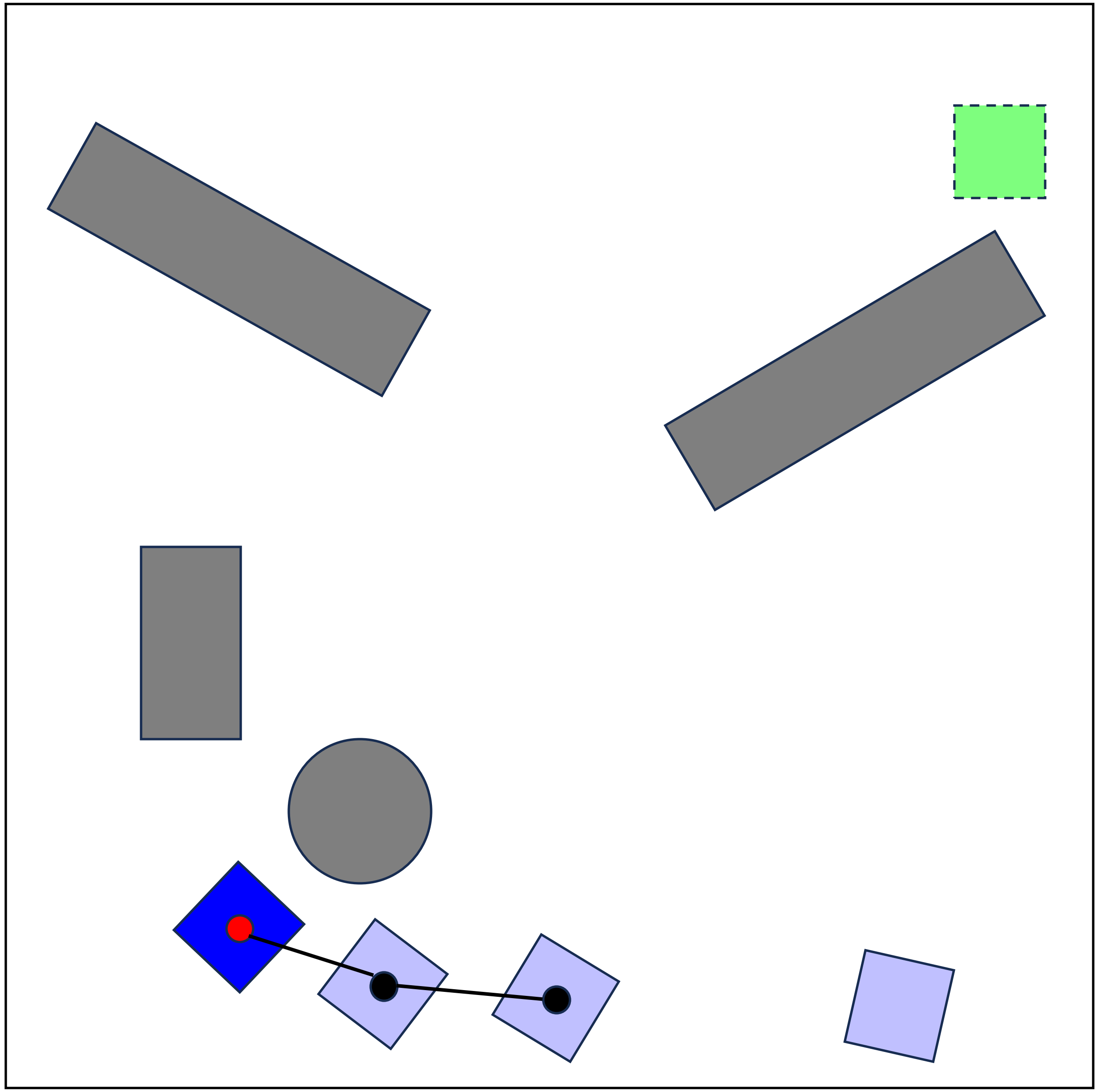


RRT(x_0, x_g, $\mathcal{X}$, f)

```
// Initialize tree at x_0
nodes = [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        x_target = x_g
    else
        // Sample a target configuration
        x_target = sample(𝒳)
    // Extend the tree towards the target
    node = getClosest(nodes, x_target)
    x_node = node.conf
    for x in extend(x_node, x_target)
        if f(x):
            nodes.add(Node(x))
            if x = x_g:
                return finish(nodes)
        else
            break
```

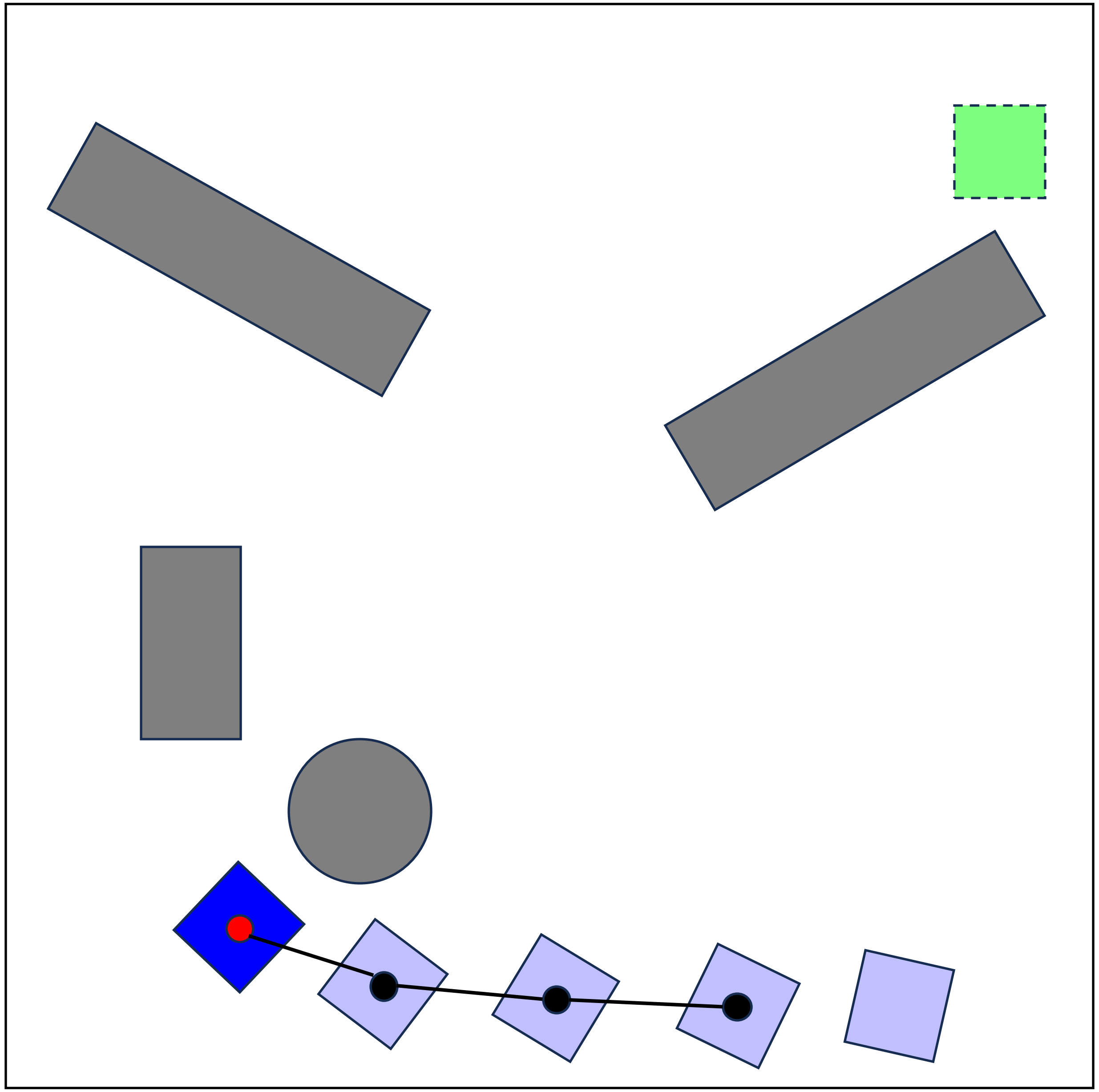


```
RRT(x_0, x_g, 𝒳, f)
// Initialize tree at x_0
nodes = [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        x_target = x_g
    else
        // Sample a target configuration
        x_target = sample(𝒳)
    // Extend the tree towards the target
    node = getClosest(nodes, x_target)
    x_node = node.conf
    for x in extend(x_node, x_target)
        if f(x):
            nodes.add(Node(x))
            if x = x_g:
                return finish(nodes)
        else
            break
```



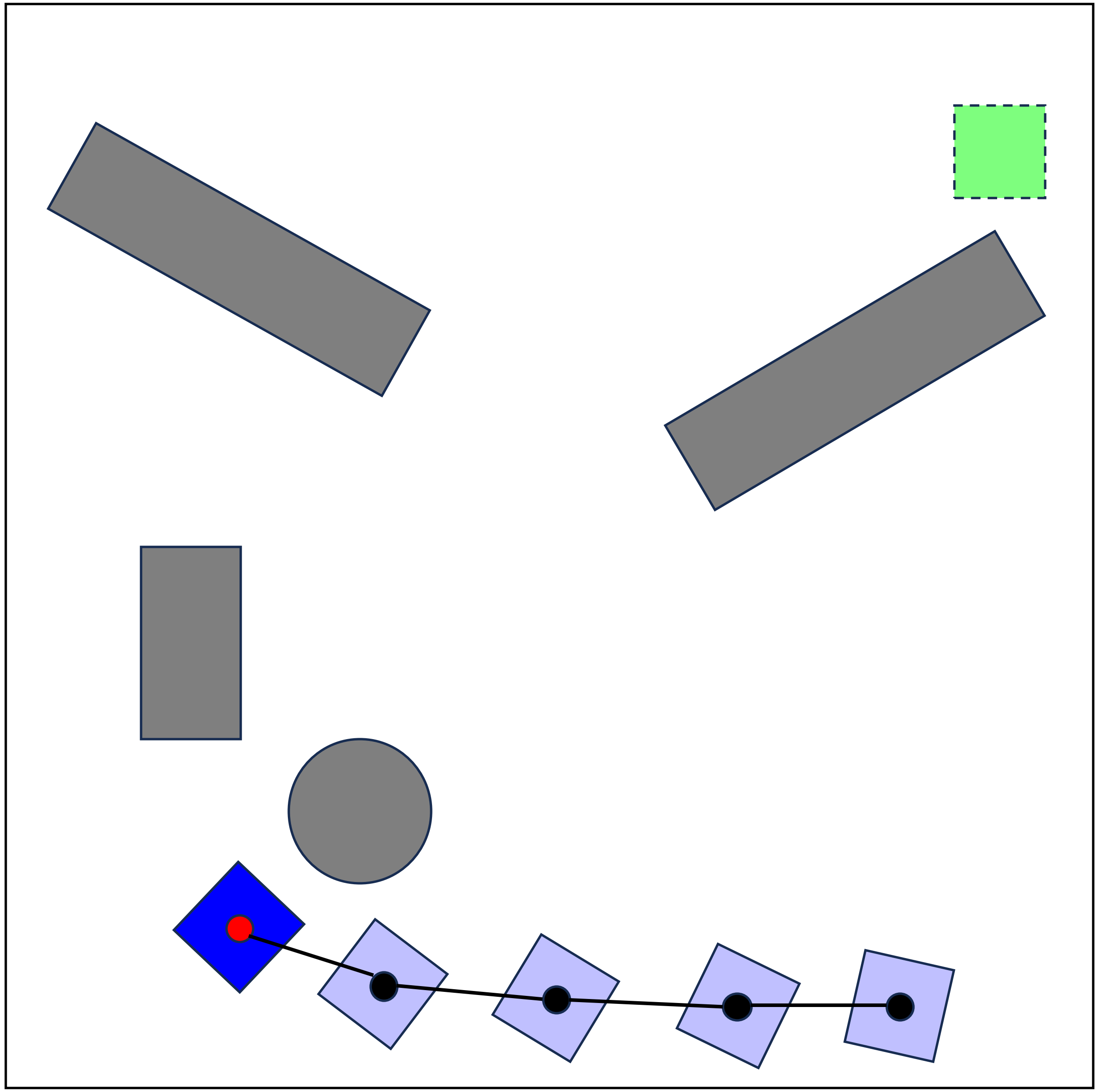
RRT$(x_0, x_g, \mathcal{X}, f)$

// Initialize tree at x_0
`nodes` $=$ [`Node`(x_0)]
repeat:
  if uniform() < `goalSampleProb`
    // Try to go directly to the goal
    $x_{\text{target}} = x_g$
  else
    // Sample a target configuration
    $x_{\text{target}} =$ `sample`$(\mathcal{X})$
  // Extend the tree towards the target
  `node` $=$ `getClosest`(`nodes`, x_{target})
  $x_{\text{node}} =$ `node.conf`
  for x **in** `extend`$(x_{\text{node}}, x_{\text{target}})$
    if $f(x)$:
      `nodes.add`(`Node`(x))
      if $x = x_g$:
        return `finish(nodes)`
    else
      break

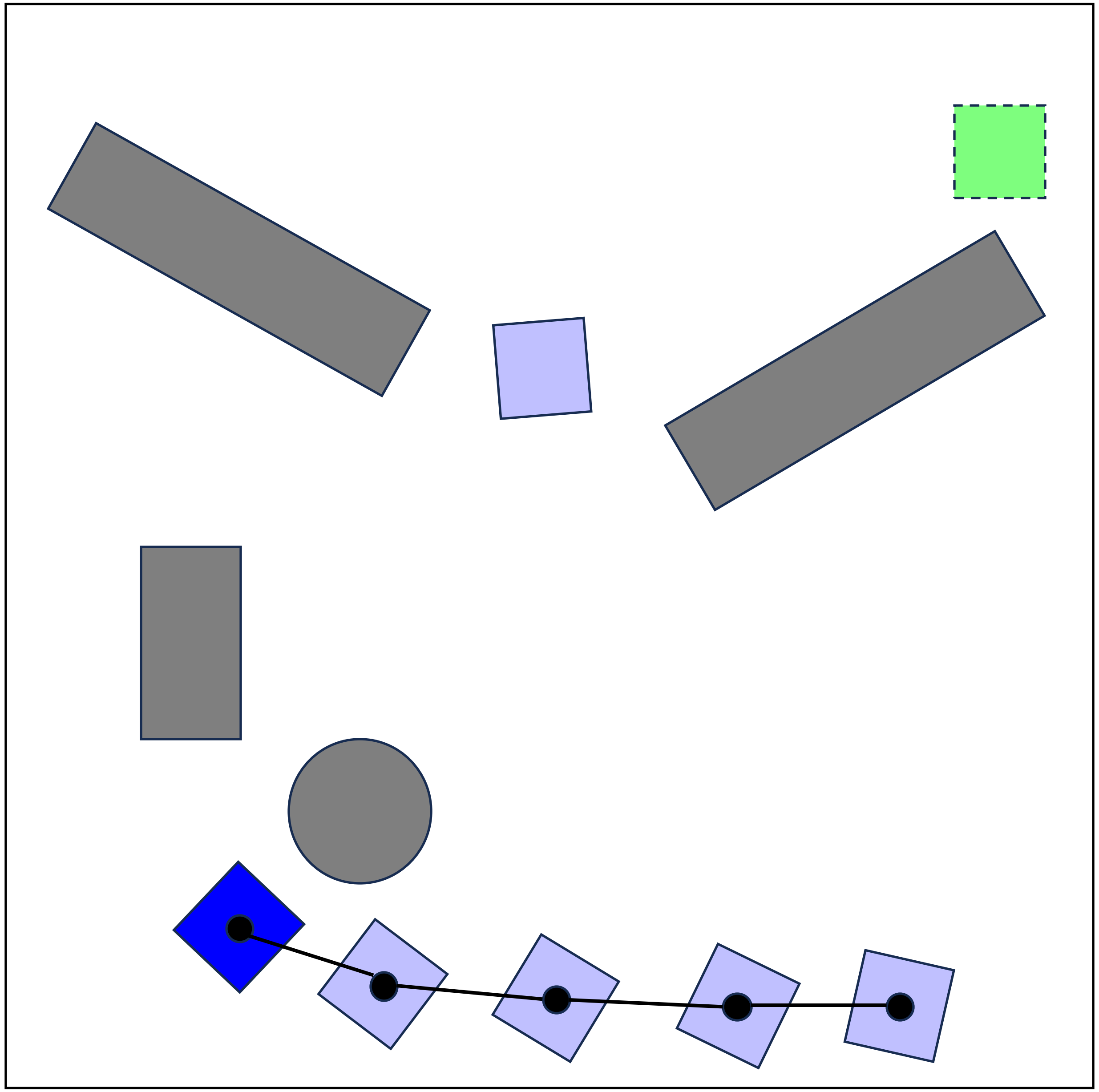


RRT$(x_0, x_g, \mathcal{X}, f)$

// Initialize tree at x_0
`nodes` $=$ [`Node`(x_0)]
repeat:
    if uniform() < `goalSampleProb`
        // Try to go directly to the goal
        $x_{\text{target}} = x_g$
    else
        // Sample a target configuration
        $x_{\text{target}} =$ `sample`$(\mathcal{X})$
    // Extend the tree towards the target
    `node` $=$ `getClosest`(`nodes`, x_{target})
    $x_{\text{node}} =$ `node.conf`
    for x **in** `extend`$(x_{\text{node}}, x_{\text{target}})$
        if $f(x)$:
            `nodes.add`(`Node`(x))
            if $x = x_g$:
                return `finish(nodes)`
        else
            break

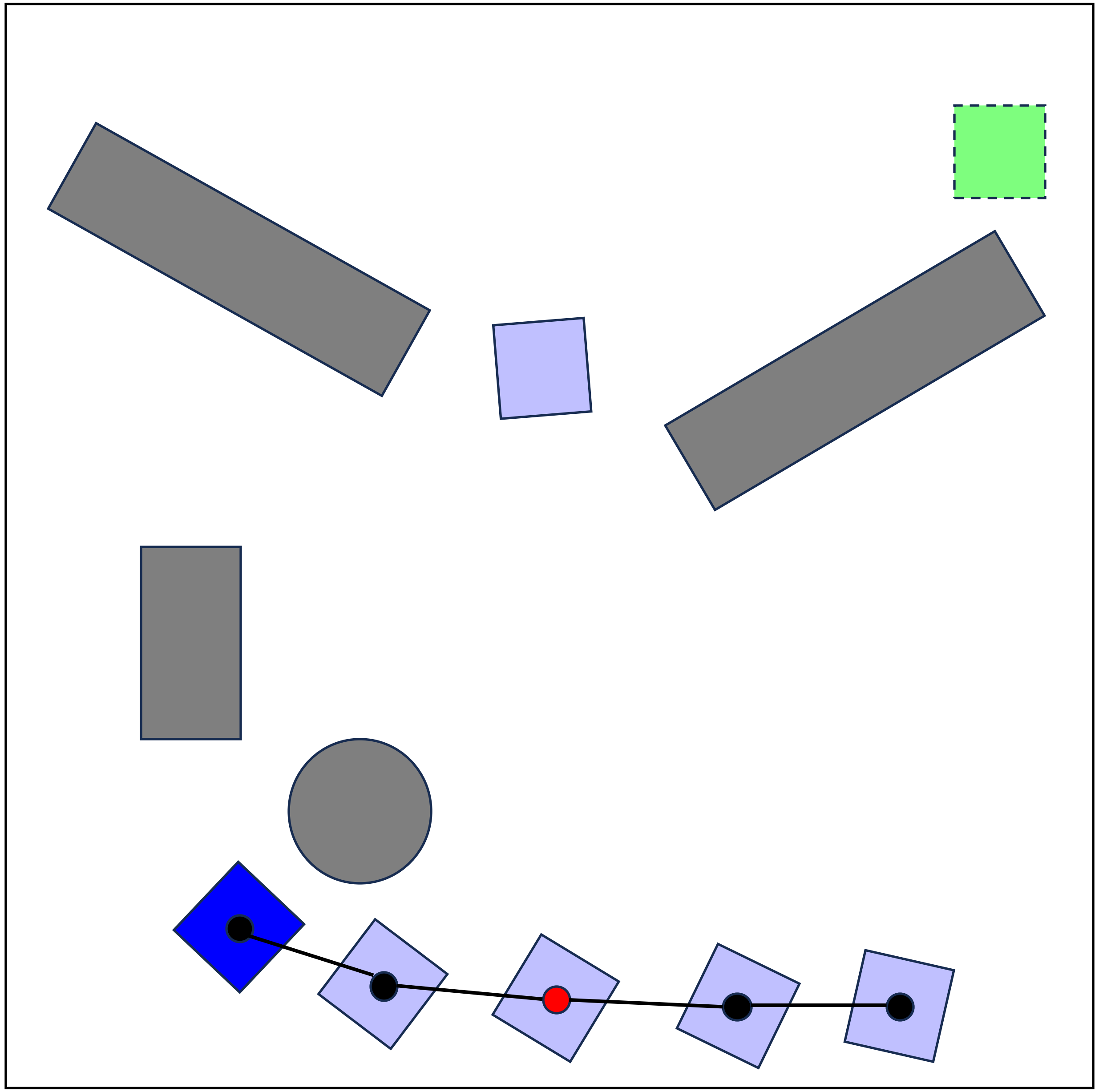


```
RRT(x_0, x_g, 𝒳, f)
// Initialize tree at x_0
nodes = [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        x_target = x_g
    else
        // Sample a target configuration
        x_target = sample(𝒳)
    // Extend the tree towards the target
    node = getClosest(nodes, x_target)
    x_node = node.conf
    for x in extend(x_node, x_target)
        if f(x):
            nodes.add(Node(x))
            if x = x_g:
                return finish(nodes)
        else
            break
```



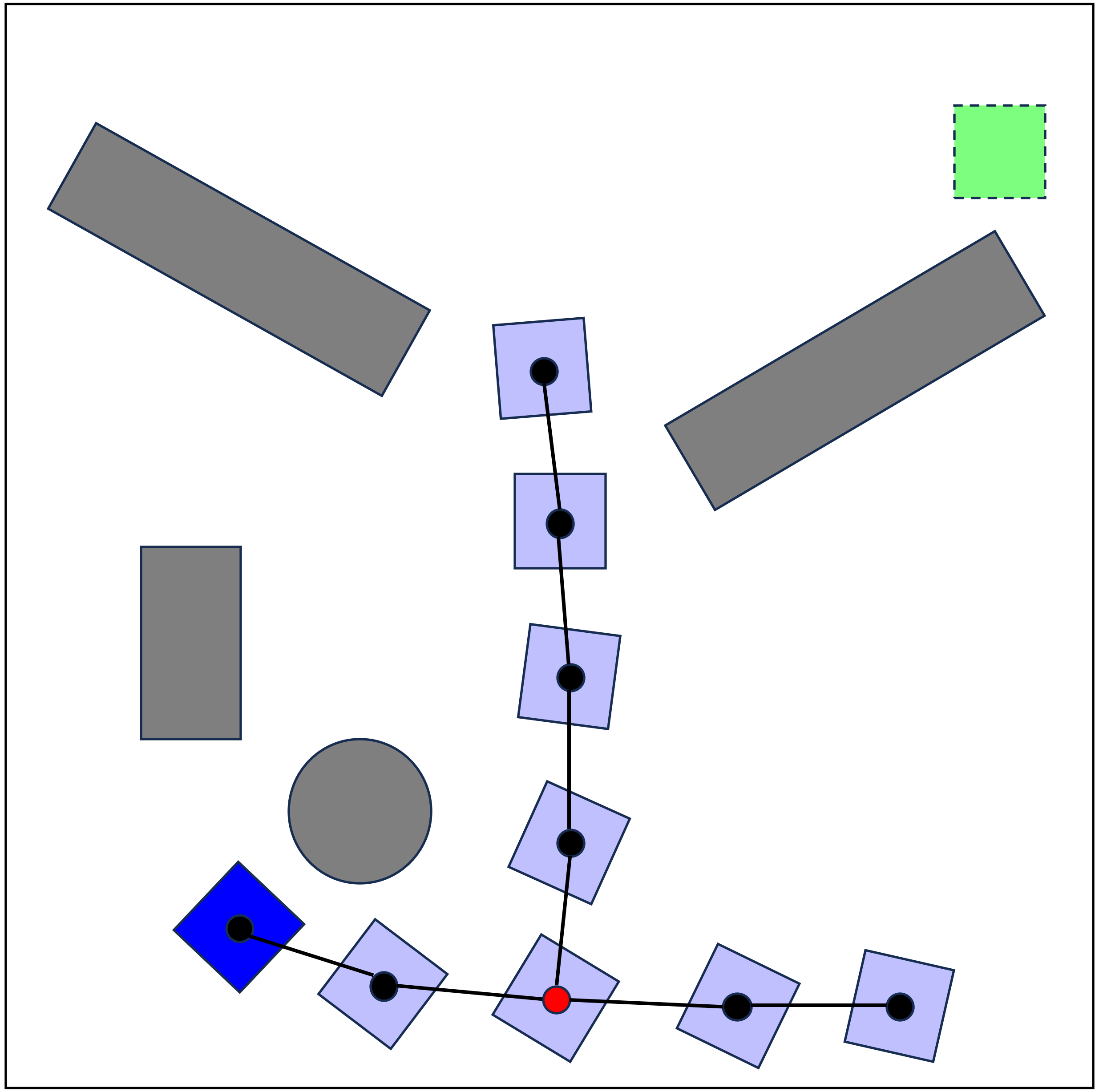
RRT$(x_0, x_g, \mathcal{X}, f)$

// Initialize tree at x_0
`nodes` $=$ [`Node`(x_0)]
repeat:
    if uniform() < `goalSampleProb`
        // Try to go directly to the goal
        $x_{\text{target}} = x_g$
    else
        // Sample a target configuration
        $x_{\text{target}} =$ `sample`$(\mathcal{X})$
    // Extend the tree towards the target
    `node` $=$ `getClosest`(`nodes`, x_{target})
    $x_{\text{node}} =$ `node.conf`
    for x **in** `extend`$(x_{\text{node}}, x_{\text{target}})$
        if $f(x)$:
            `nodes.add`(`Node`(x))
            if $x = x_g$:
                return `finish(nodes)`
        else
            break

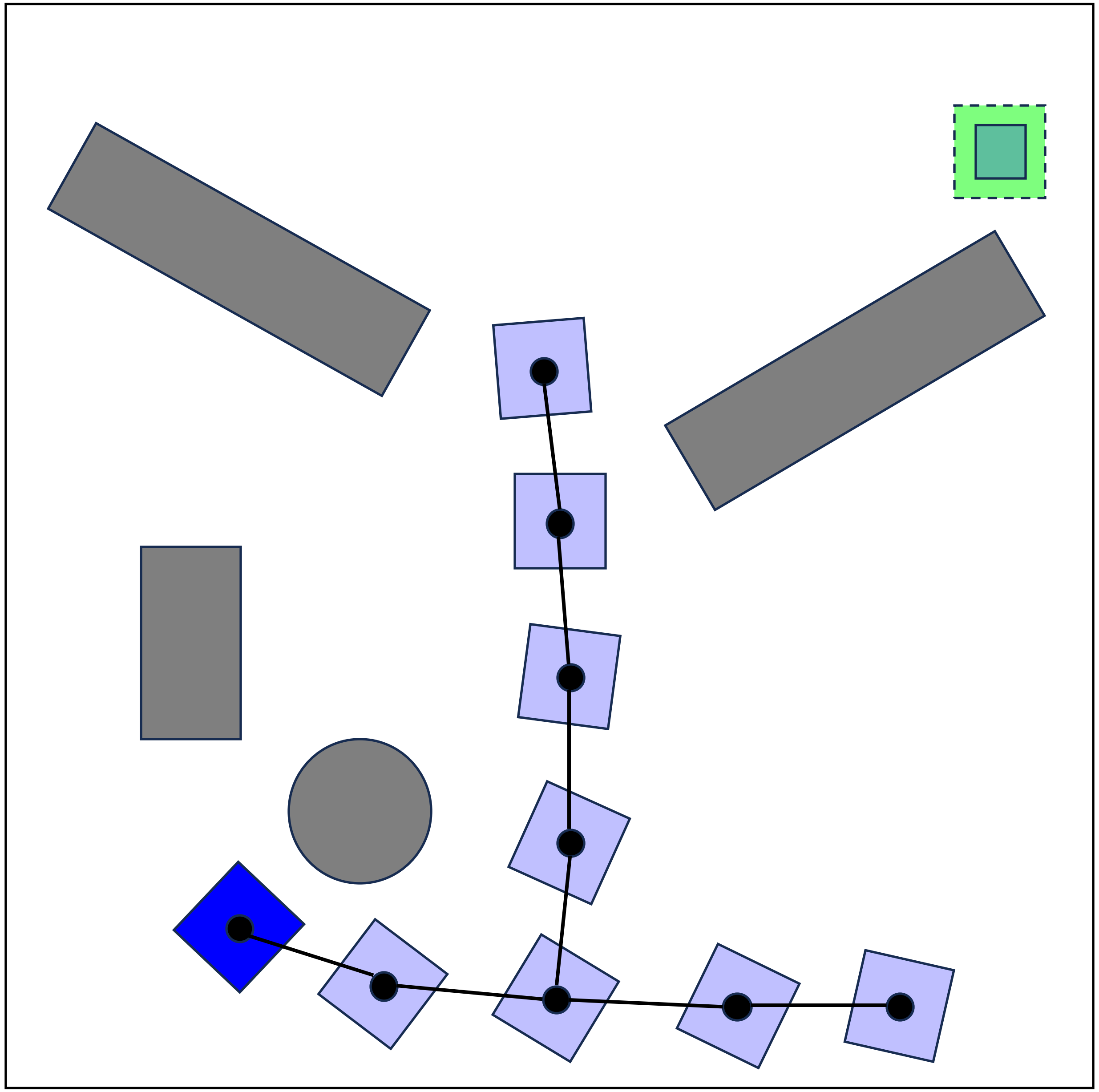


RRT(x_0, x_g, $\mathcal{X}$, f)

// Initialize tree at x_0
`nodes` $=$ [`Node`(x_0)]
repeat:
    if uniform() < `goalSampleProb`
        // Try to go directly to the goal
        $x_{\text{target}} = x_g$
    else
        // Sample a target configuration
        $x_{\text{target}} =$ `sample`($\mathcal{X}$)
    // Extend the tree towards the target
    $\boxed{\texttt{node} = \texttt{getClosest}(\texttt{nodes}, x_{\text{target}})}$
    $\boxed{x_{\text{node}} = \texttt{node.conf}}$
    for x **in** `extend`(x_{node}, x_{target})
        if $f(x)$:
            `nodes.add`(`Node`(x))
            if $x = x_g$:
                return `finish(nodes)`
        else
            break

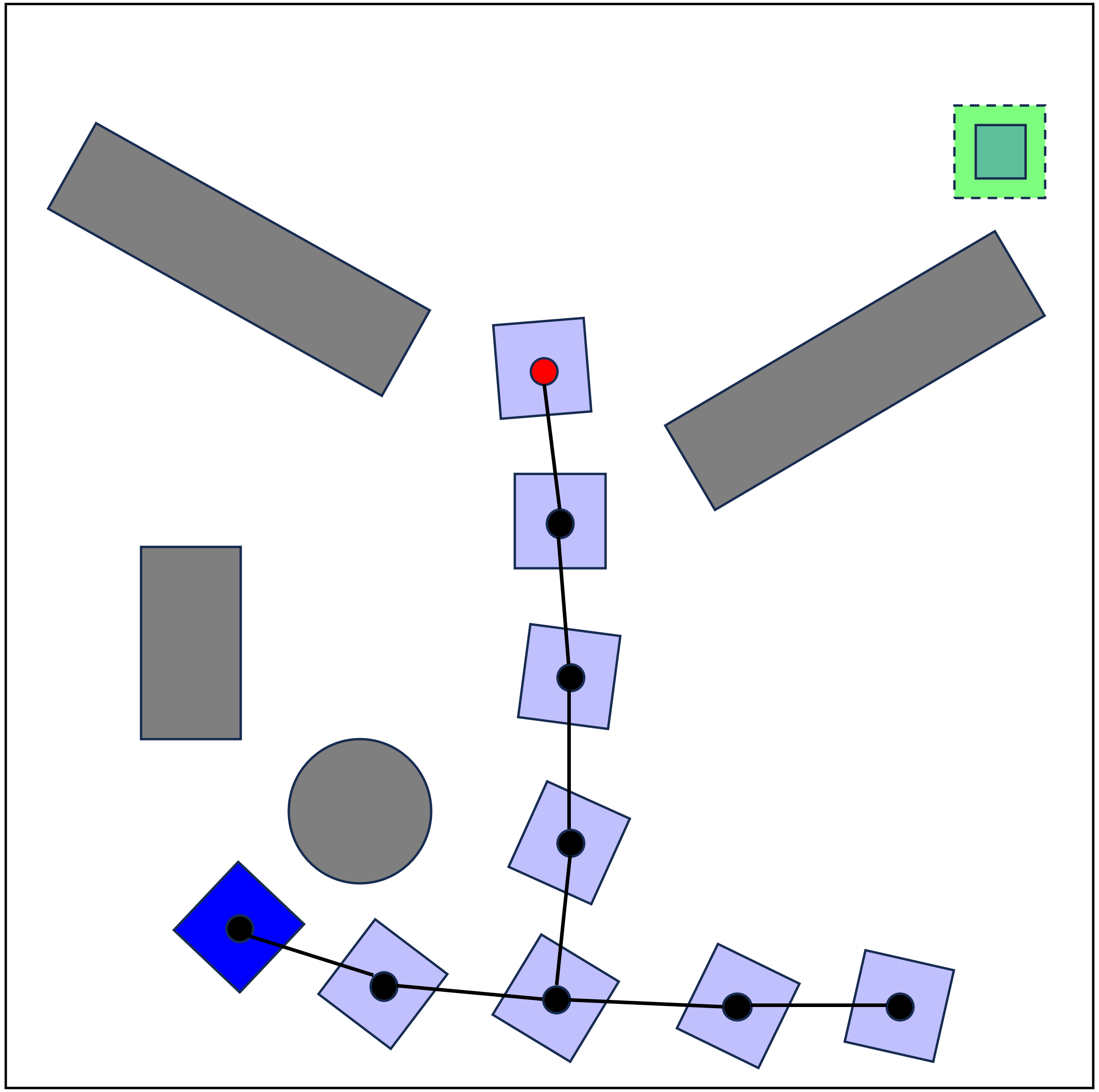


```
RRT(x_0, x_g, 𝒳, f)
    // Initialize tree at x_0
    nodes = [Node(x_0)]
    repeat:
        if uniform() < goalSampleProb
            // Try to go directly to the goal
            x_target = x_g
        else
            // Sample a target configuration
            x_target = sample(𝒳)
        // Extend the tree towards the target
        node = getClosest(nodes, x_target)
        x_node = node.conf
        for x in extend(x_node, x_target)
            if f(x):
                nodes.add(Node(x))
                if x = x_g:
                    return finish(nodes)
            else
                break
```



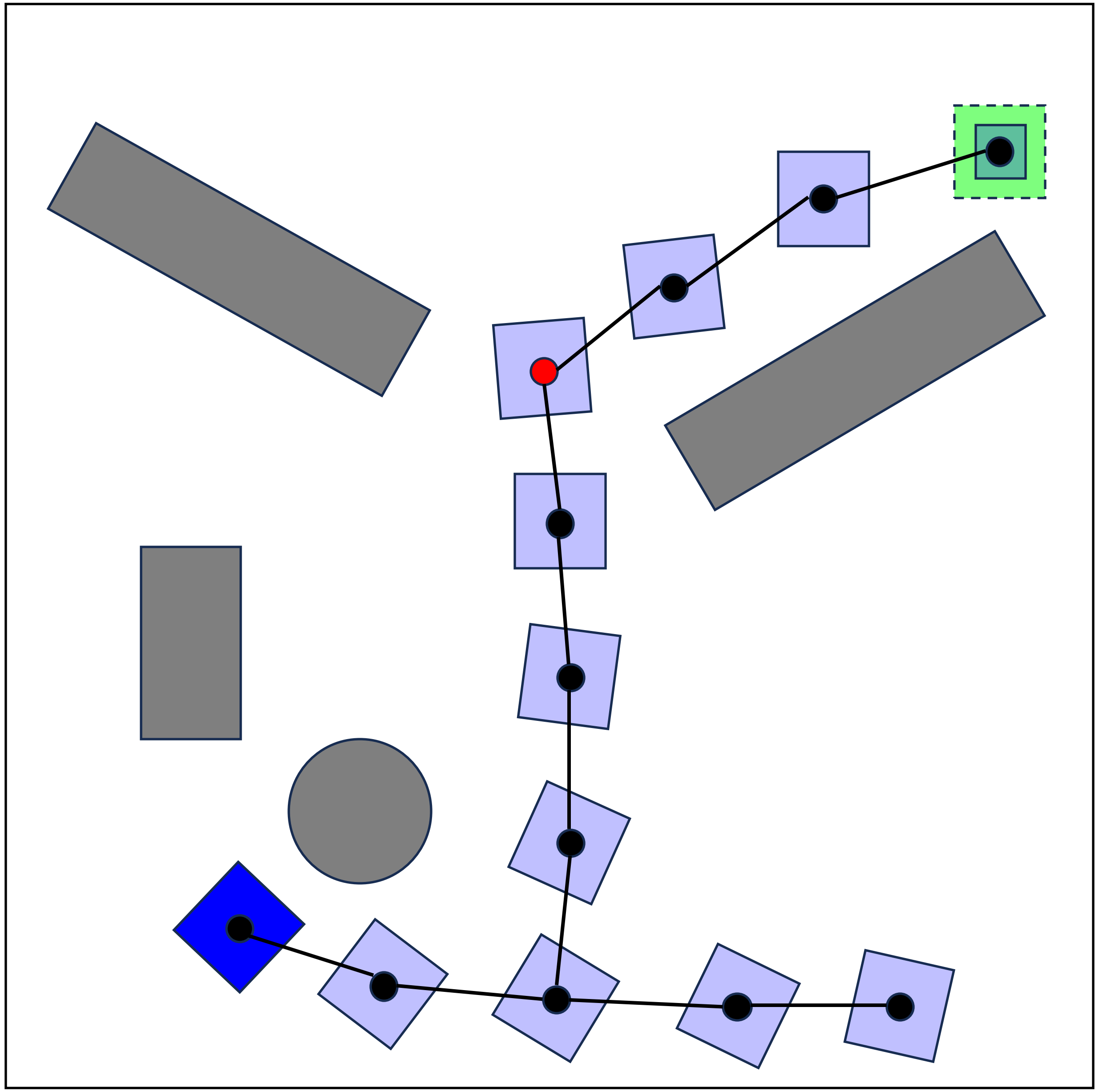
RRT$(x_0, x_g, \mathcal{X}, f)$

// Initialize tree at x_0
nodes $=$ [Node(x_0)]
repeat:
  if uniform() < goalSampleProb
    // Try to go directly to the goal
    $x_{\text{target}} = x_g$
  else
    // Sample a target configuration
    $x_{\text{target}} =$ sample$(\mathcal{X})$
  // Extend the tree towards the target
  node $=$ getClosest(nodes, x_{target})
  $x_{\text{node}} =$ node.conf
  for x **in** extend$(x_{\text{node}}, x_{\text{target}})$
    if $f(x)$:
      nodes.add(Node(x))
      if $x = x_g$:
        return finish(nodes)
    else
      break



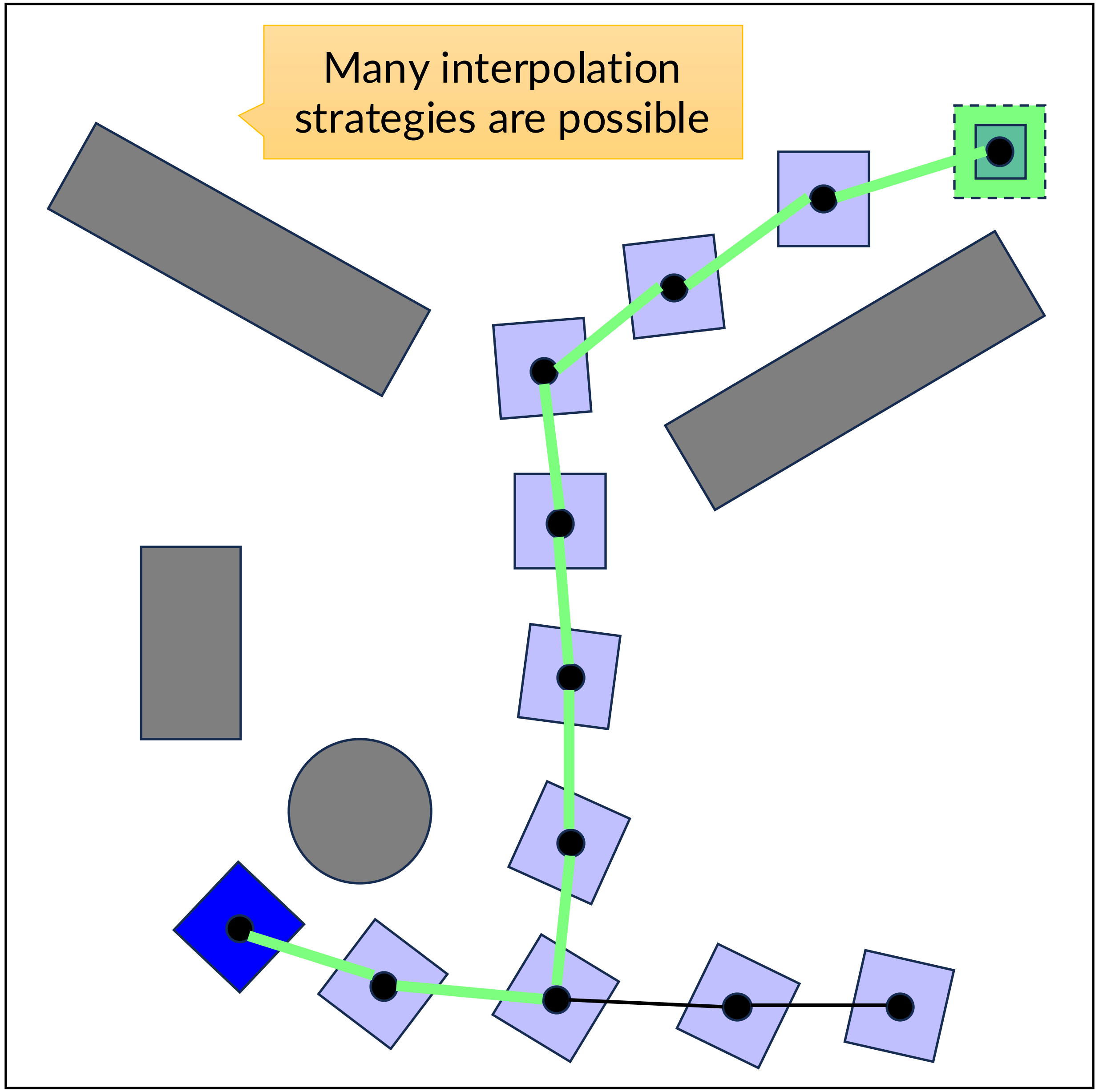
RRT(x_0, x_g, $\mathcal{X}$, f)

```
// Initialize tree at x_0
nodes = [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        x_target = x_g
    else
        // Sample a target configuration
        x_target = sample(𝒳)
    // Extend the tree towards the target
    node = getClosest(nodes, x_target)
    x_node = node.conf
    for x in extend(x_node, x_target)
        if f(x):
            nodes.add(Node(x))
            if x = x_g:
                return finish(nodes)
        else
            break
```



RRT(x_0, x_g, $\mathcal{X}$, f)

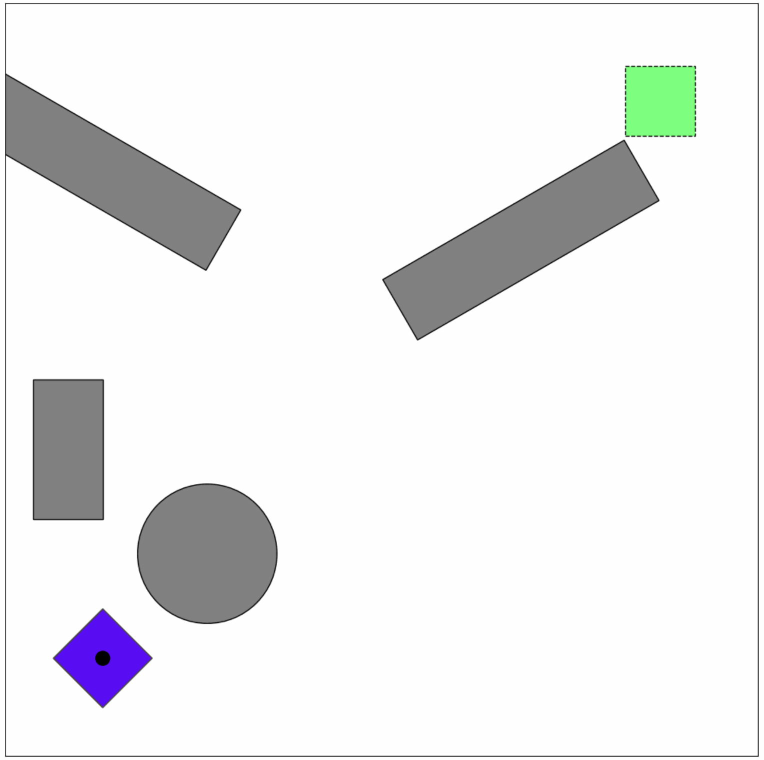
```
// Initialize tree at x_0
nodes = [Node(x_0)]
repeat:
    if uniform() < goalSampleProb
        // Try to go directly to the goal
        x_target = x_g
    else
        // Sample a target configuration
        x_target = sample(𝒳)
    // Extend the tree towards the target
    node = getClosest(nodes, x_target)
    x_node = node.conf
    for x in extend(x_node, x_target)
        if f(x):
            nodes.add(Node(x))
            if x = x_g:
                return finish(nodes)
        else
            break
```



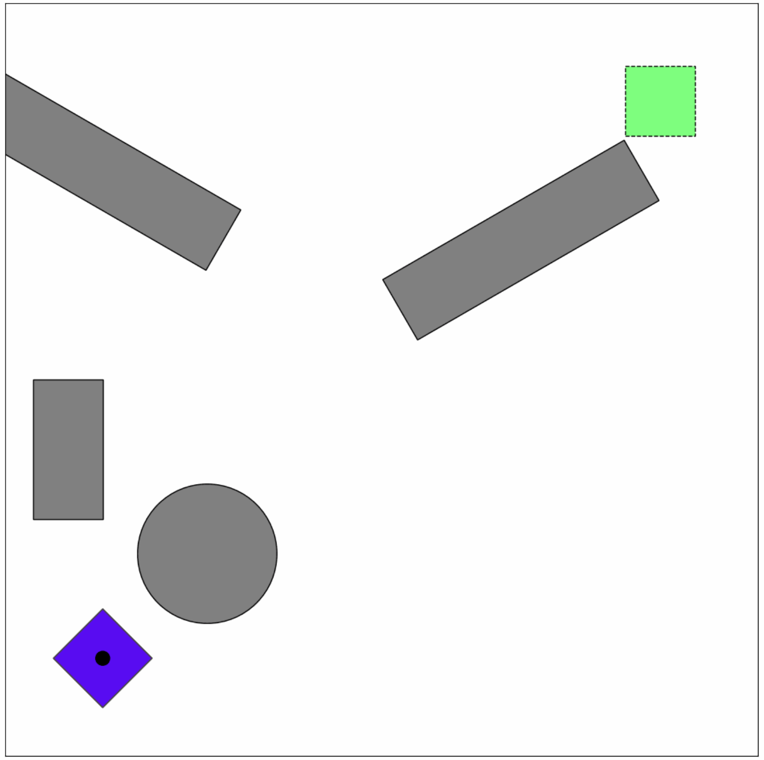

```
RRT(x_0, x_g, 𝒳, f)
  // Initialize tree at x_0
  nodes = [Node(x_0)]
  repeat:
      if uniform() < goalSampleProb
          // Try to go directly to the goal
          x_target = x_g
      else
          // Sample a target configuration
          x_target = sample(𝒳)
      // Extend the tree towards the target
      node = getClosest(nodes, x_target)
      x_node = node.conf
      for x in extend(x_node, x_target)
          if f(x):
              nodes.add(Node(x))
              if x = x_g:
                  return finish(nodes)
          else
              break
```

Important Hyperparameter: Feasibility Check Distance

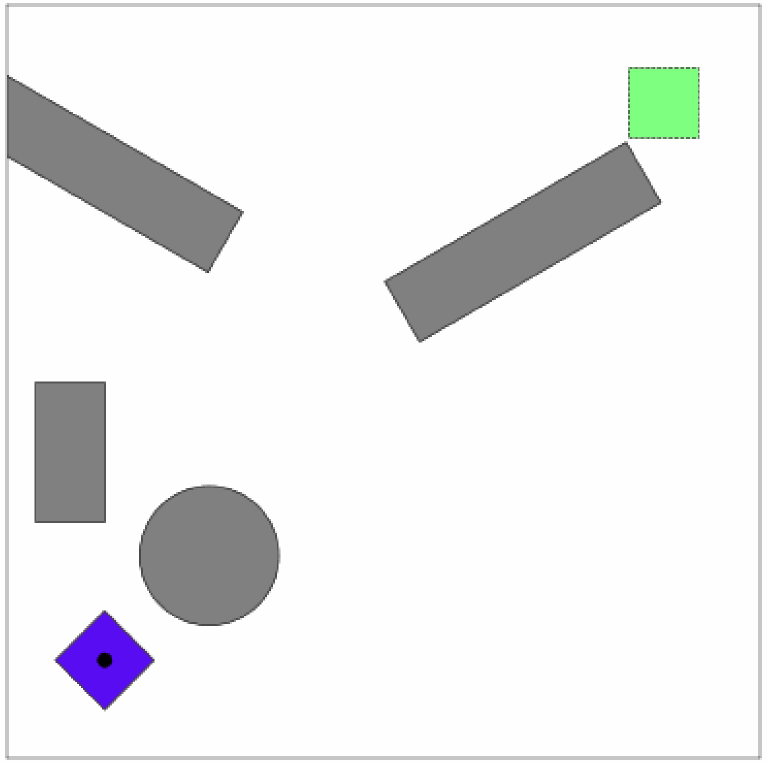
Check Distance = 5.0



Check Distance = 1.0



Check Distance = 0.2

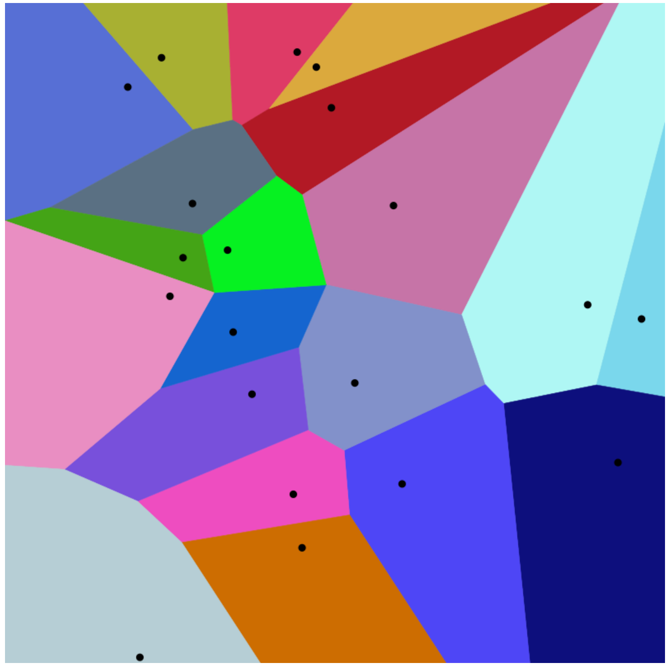


Properties of RRT

- Probabilistically complete
- *Not optimal*
- *Single query*
- Works with underactuated systems
- Works for kinodynamic problems

The probability of expanding a tree node is proportional to the size of its Voronoi region

- Has *Voronoi bias*

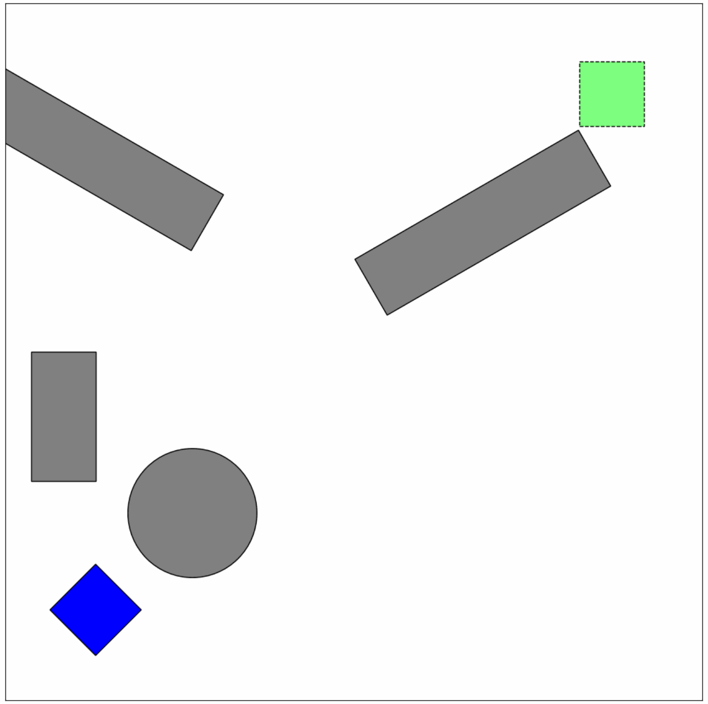


https://en.wikipedia.org/wiki/Voronoi_diagram

Post-Processing with Shortcuts

Repeatedly sample two points on the trajectory and check if a direct line between them is feasible (rewire if so)

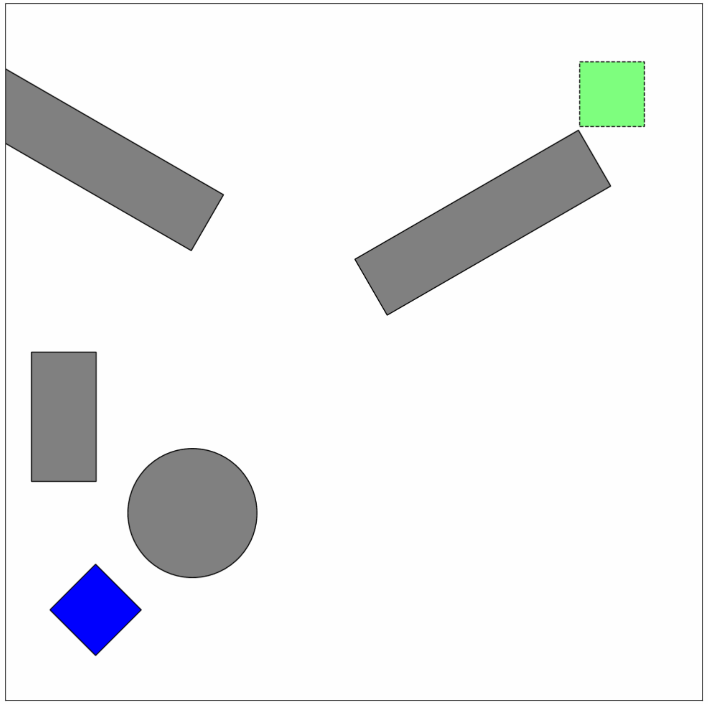
Attempts = 0



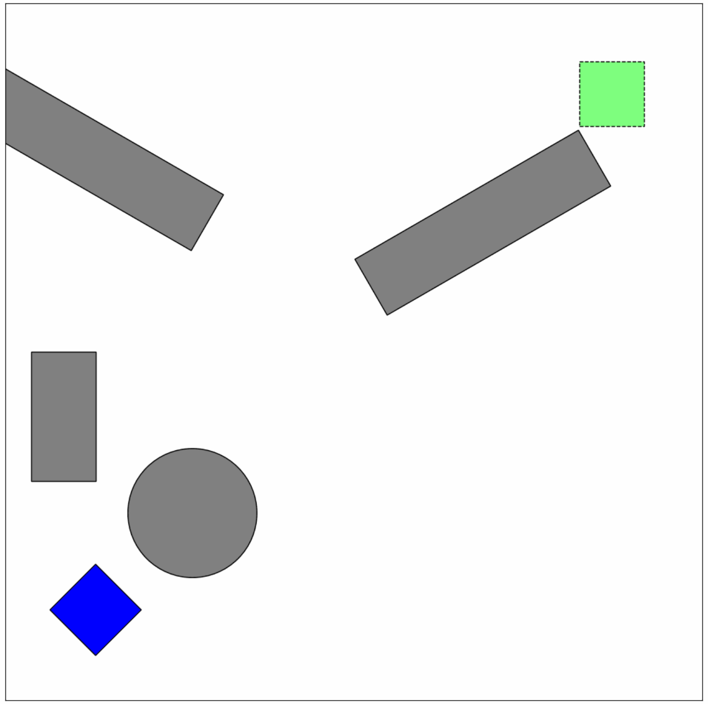
Post-Processing with Shortcuts

Repeatedly sample two points on the trajectory and check if a direct line between them is feasible (rewire if so)

Attempts = 0

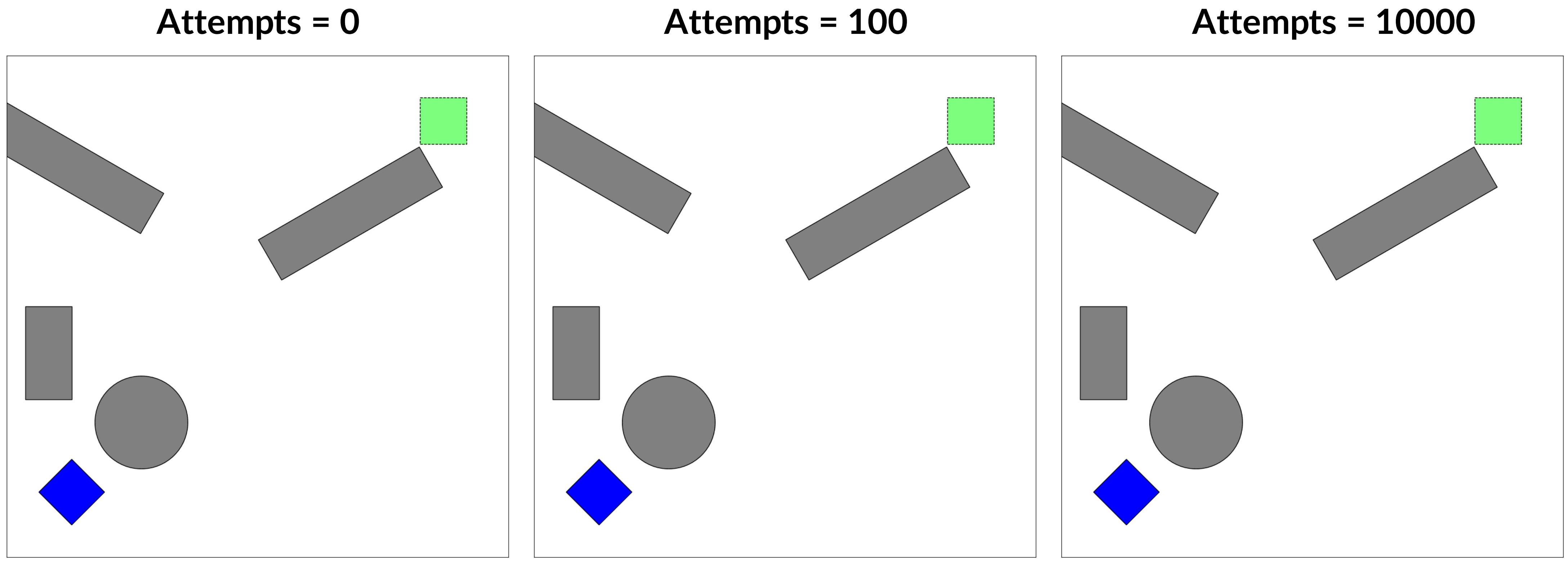


Attempts = 100



Post-Processing with Shortcuts

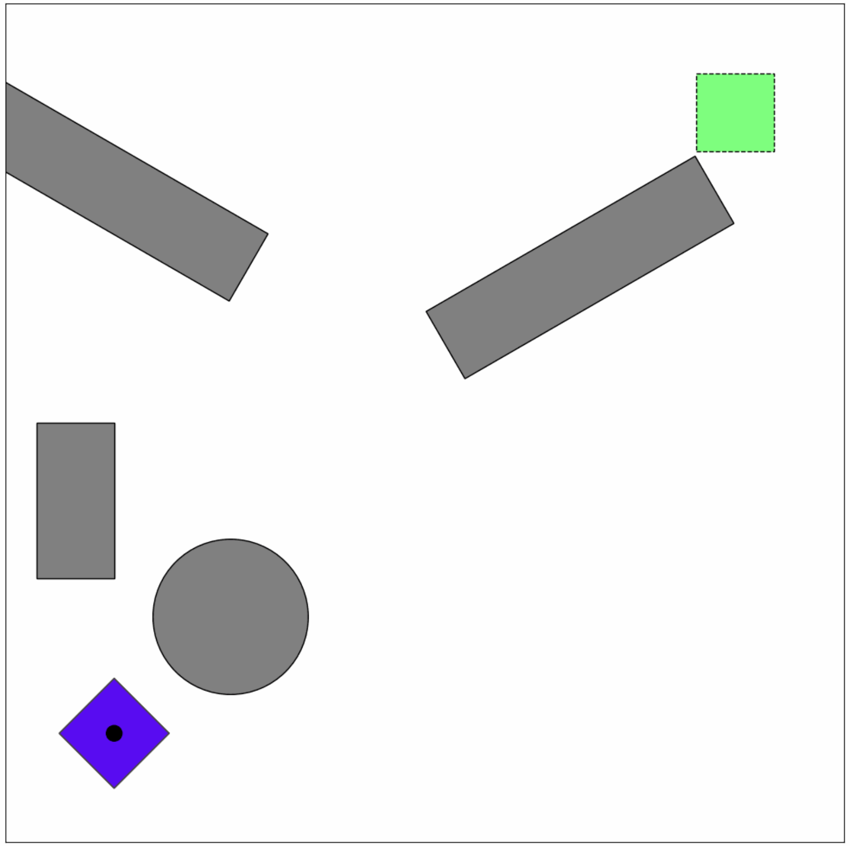
Repeatedly sample two points on the trajectory and check if a direct line between them is feasible (rewire if so)

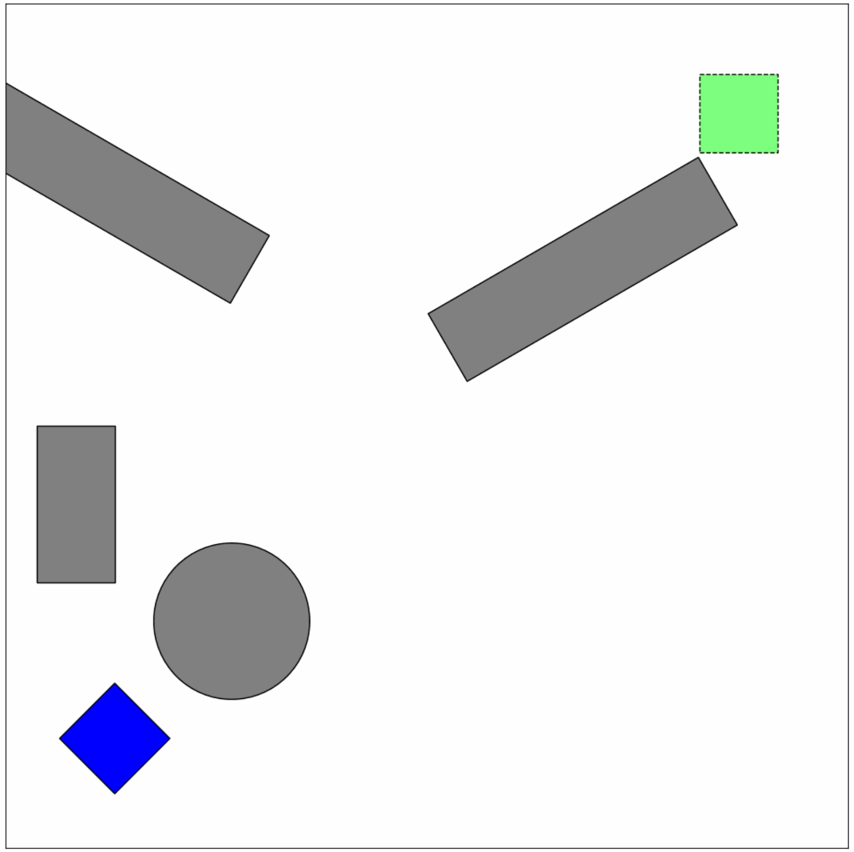


Bidirectional RRT

Does not work for underactuated systems

Grow two trees: one from start, the other from goal



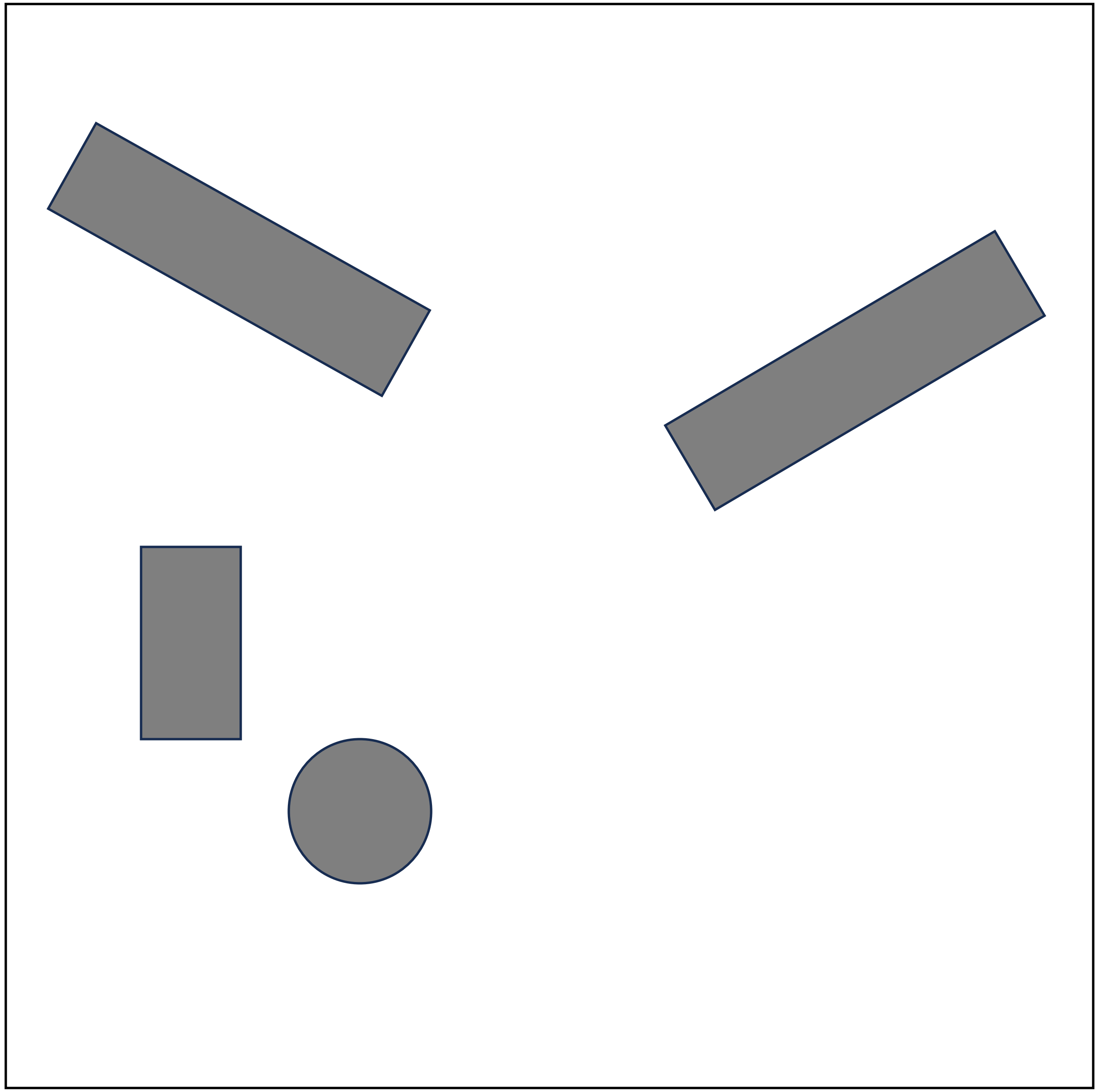


Multi-Query Motion Planning

- RRT grows tree once, from initial to goal
- After the robot moves, need to start from scratch
- Can we build a representation once, up front, instead?
- Need a *graph* instead of a *tree*

Probabilistic Roadmaps (PRMs)

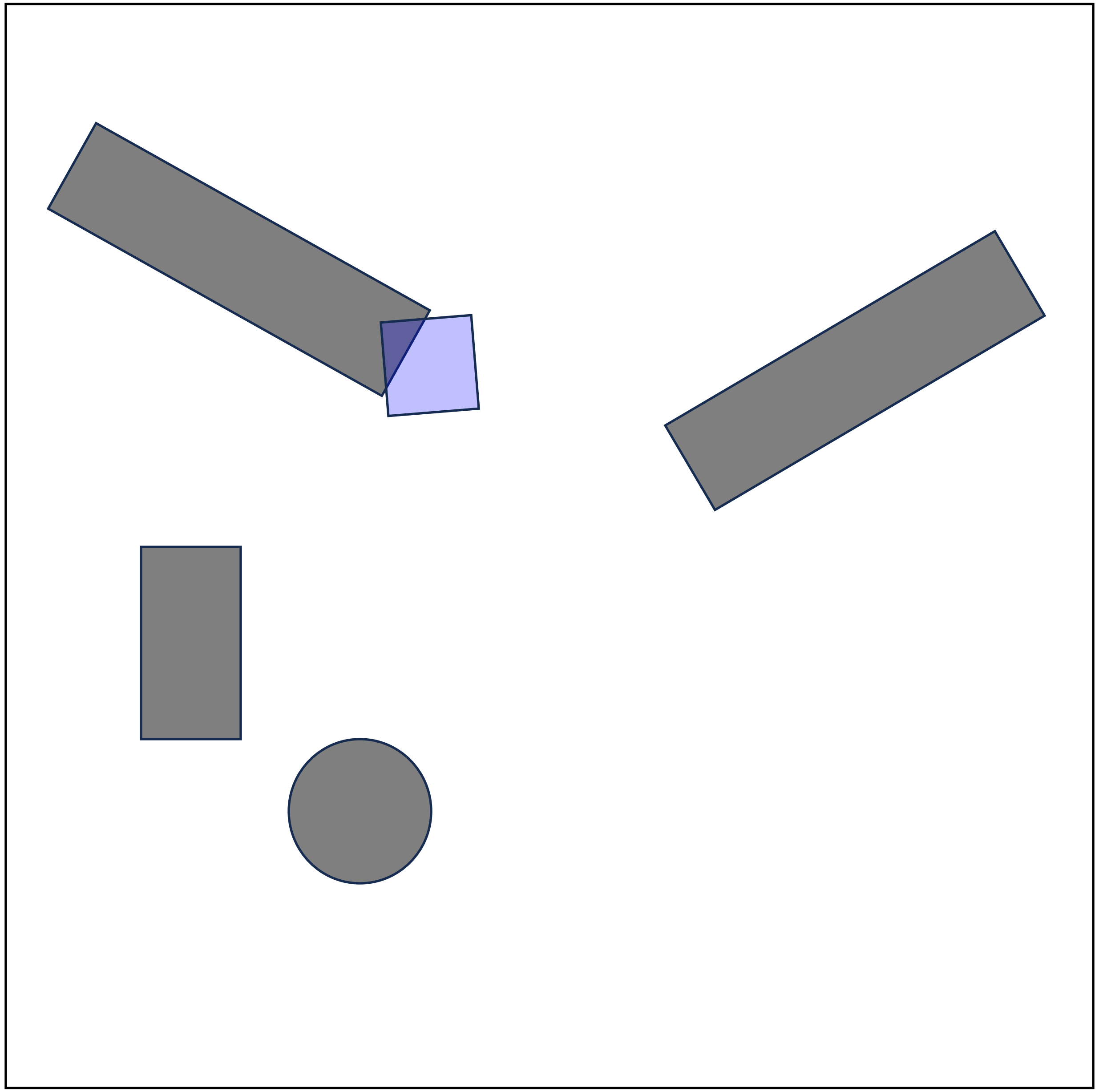
```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph

UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode

QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



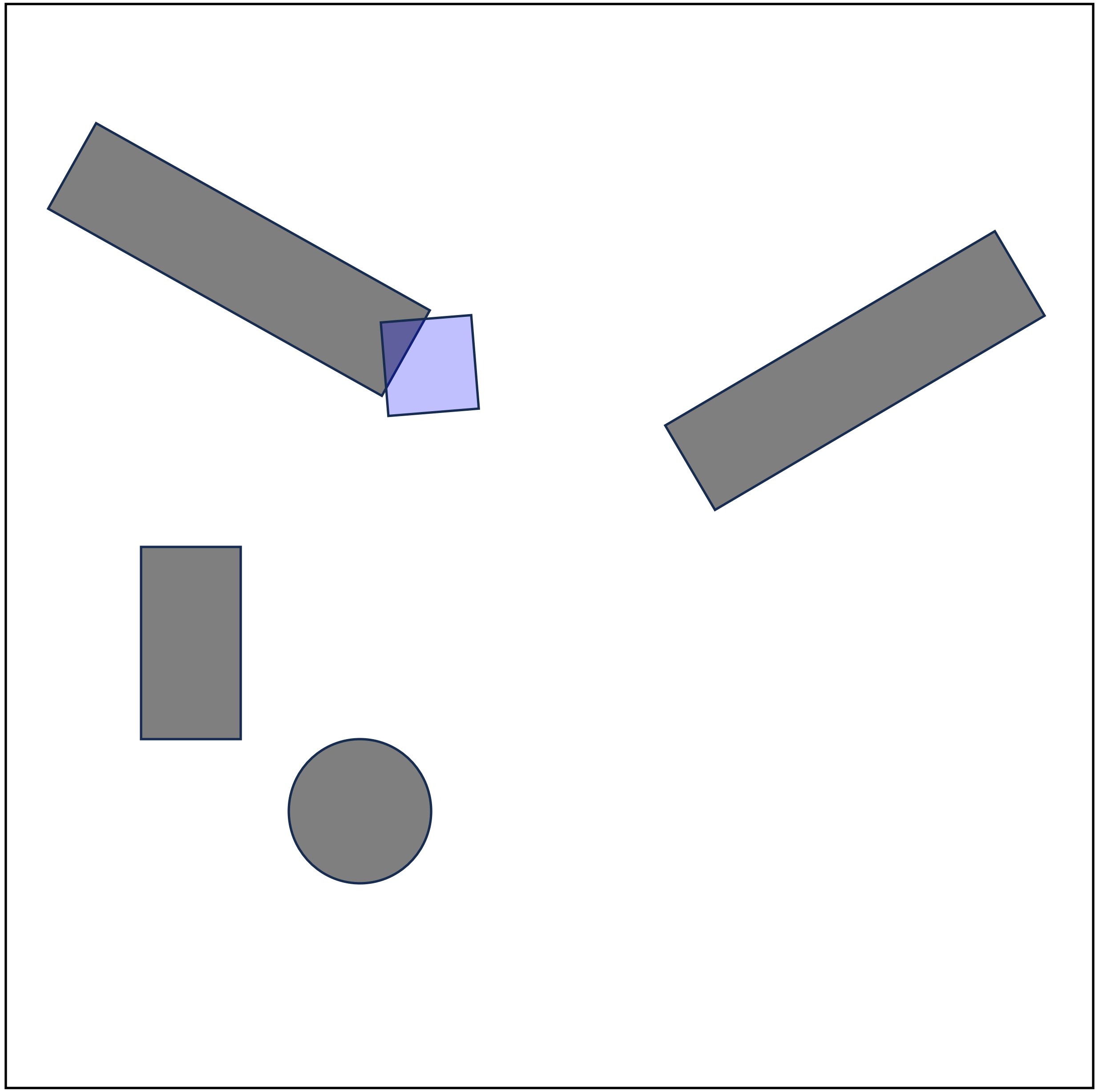
```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph

UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode

QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

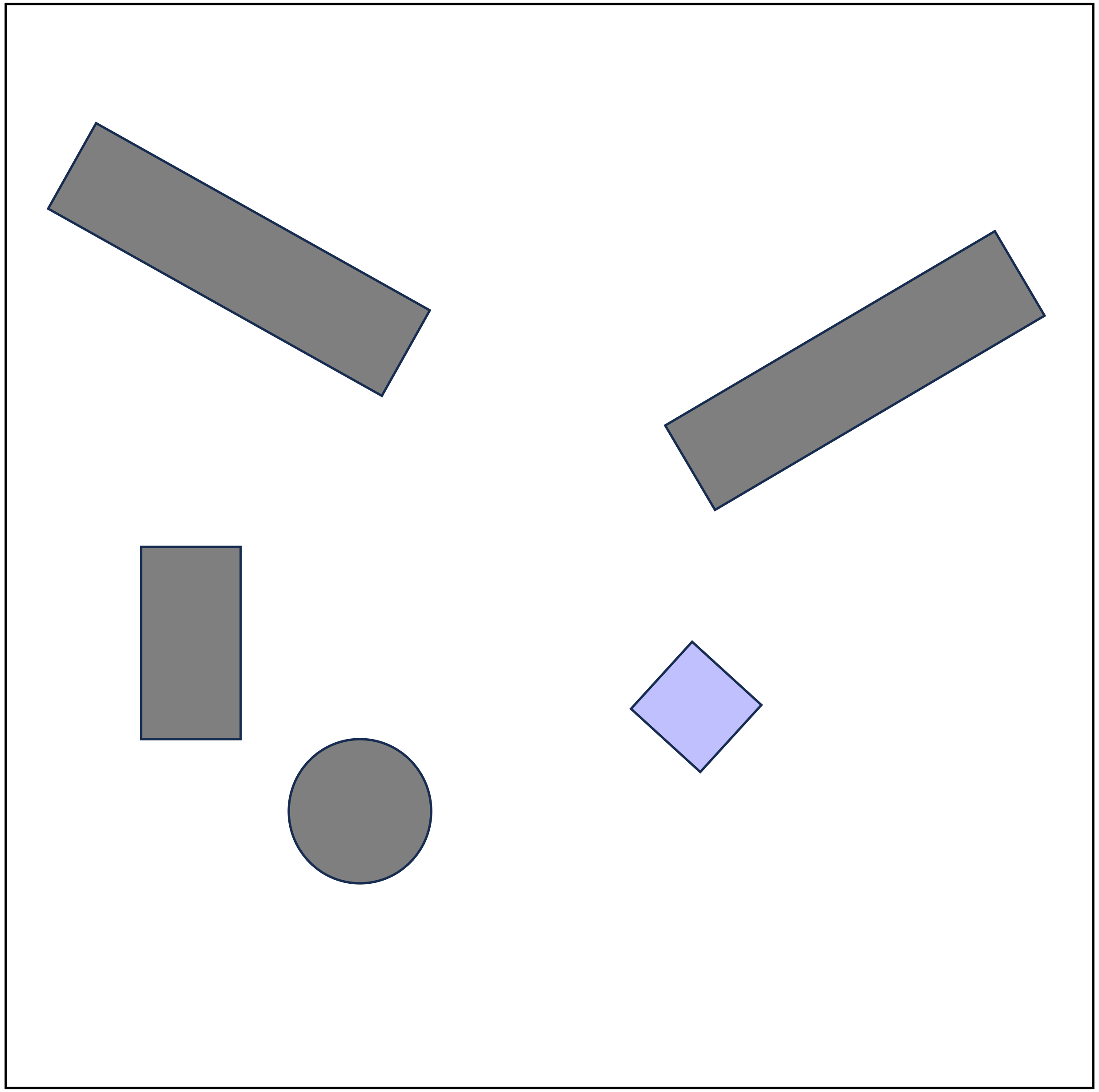
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

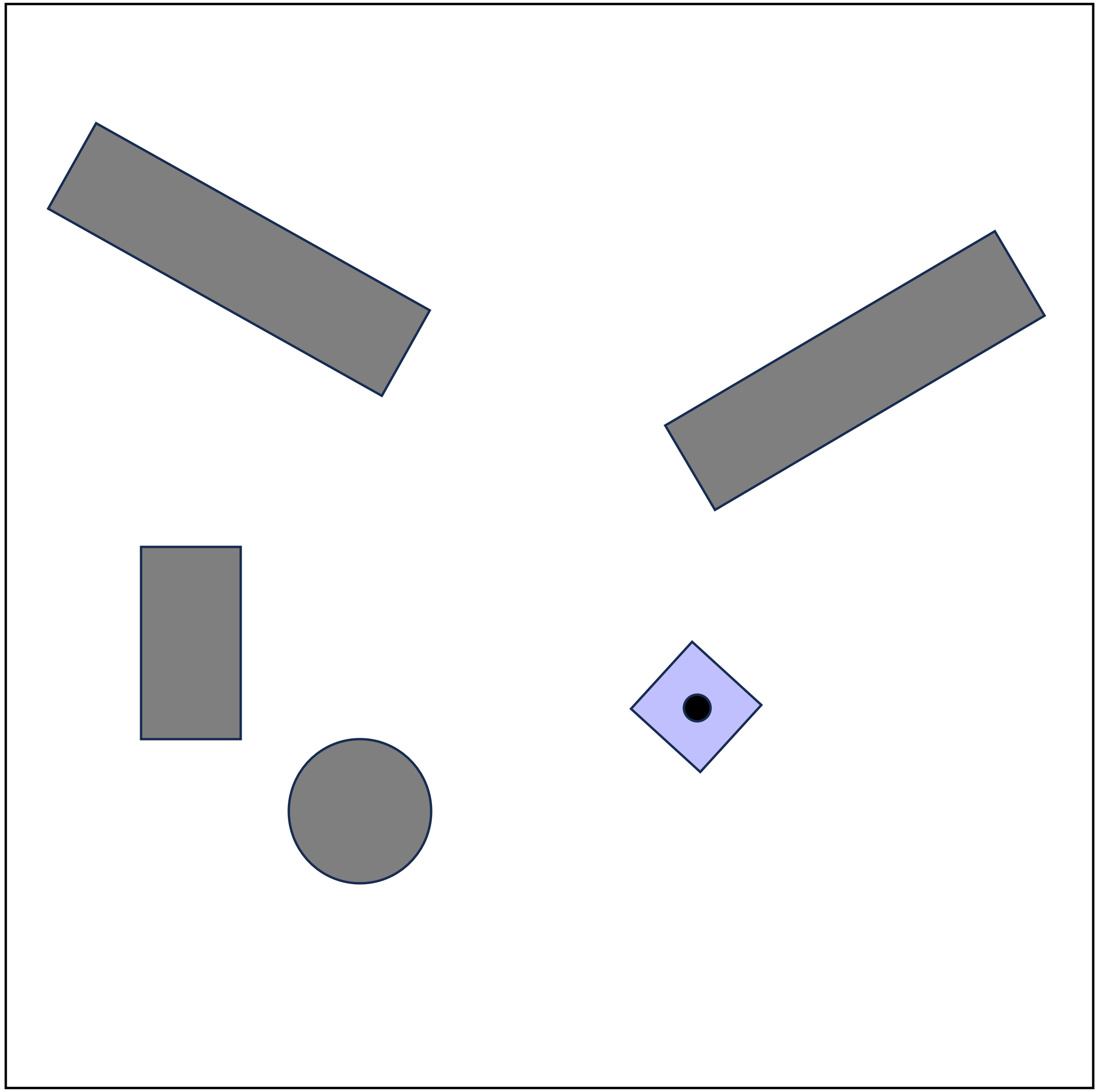
```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



```
BuildPRM(\mathcal{X}, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(\mathcal{X})
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

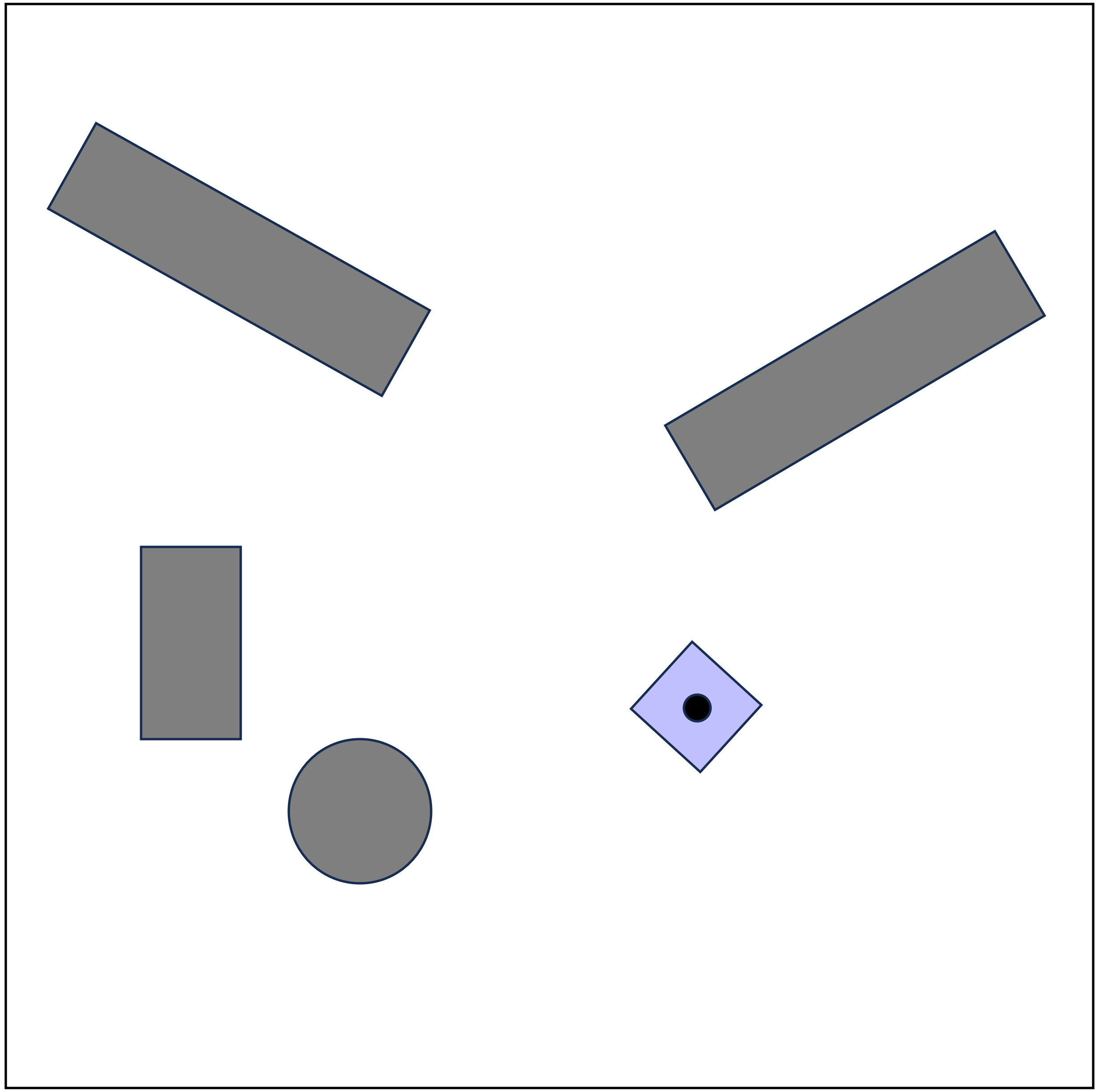
```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



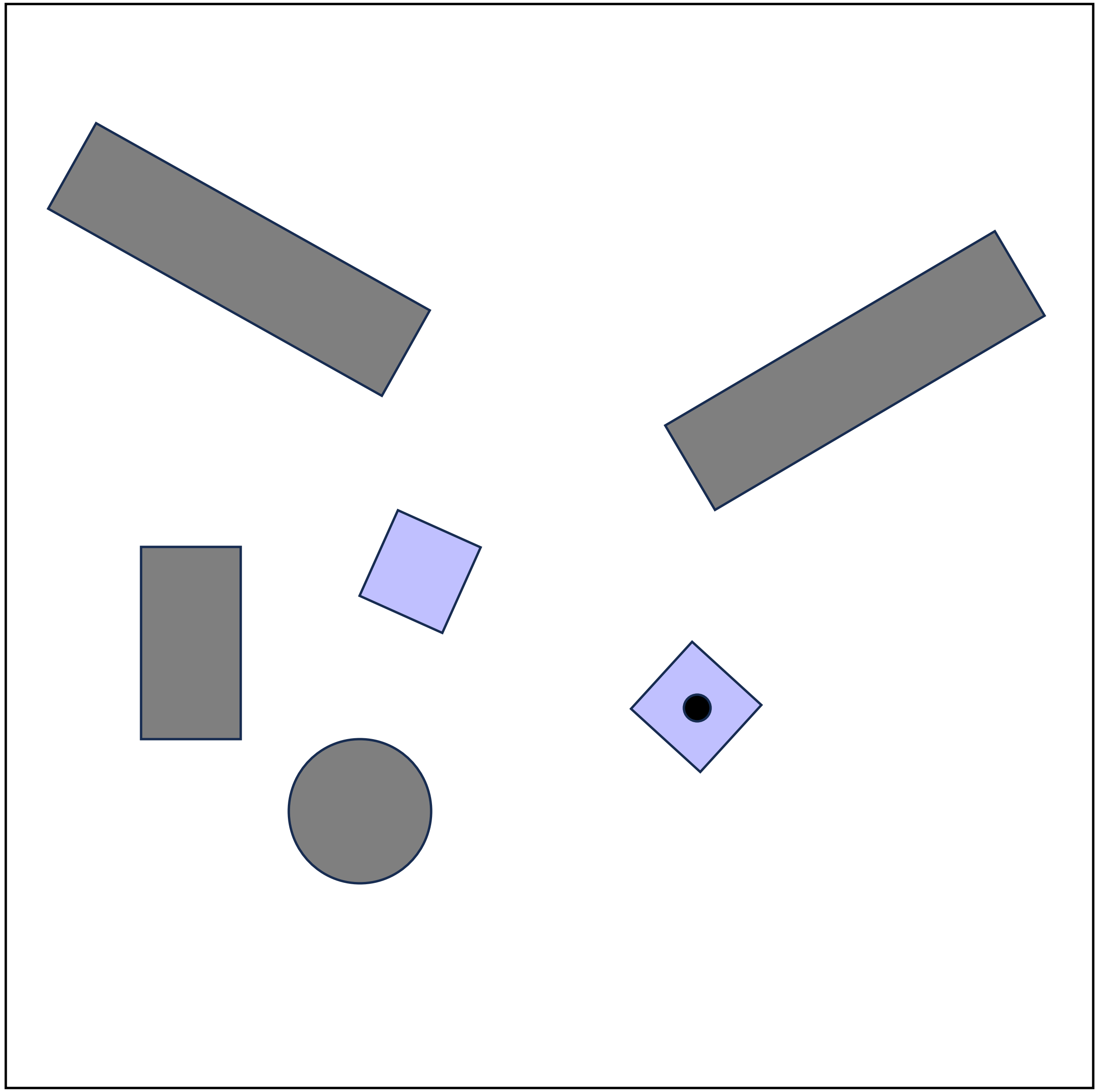
```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

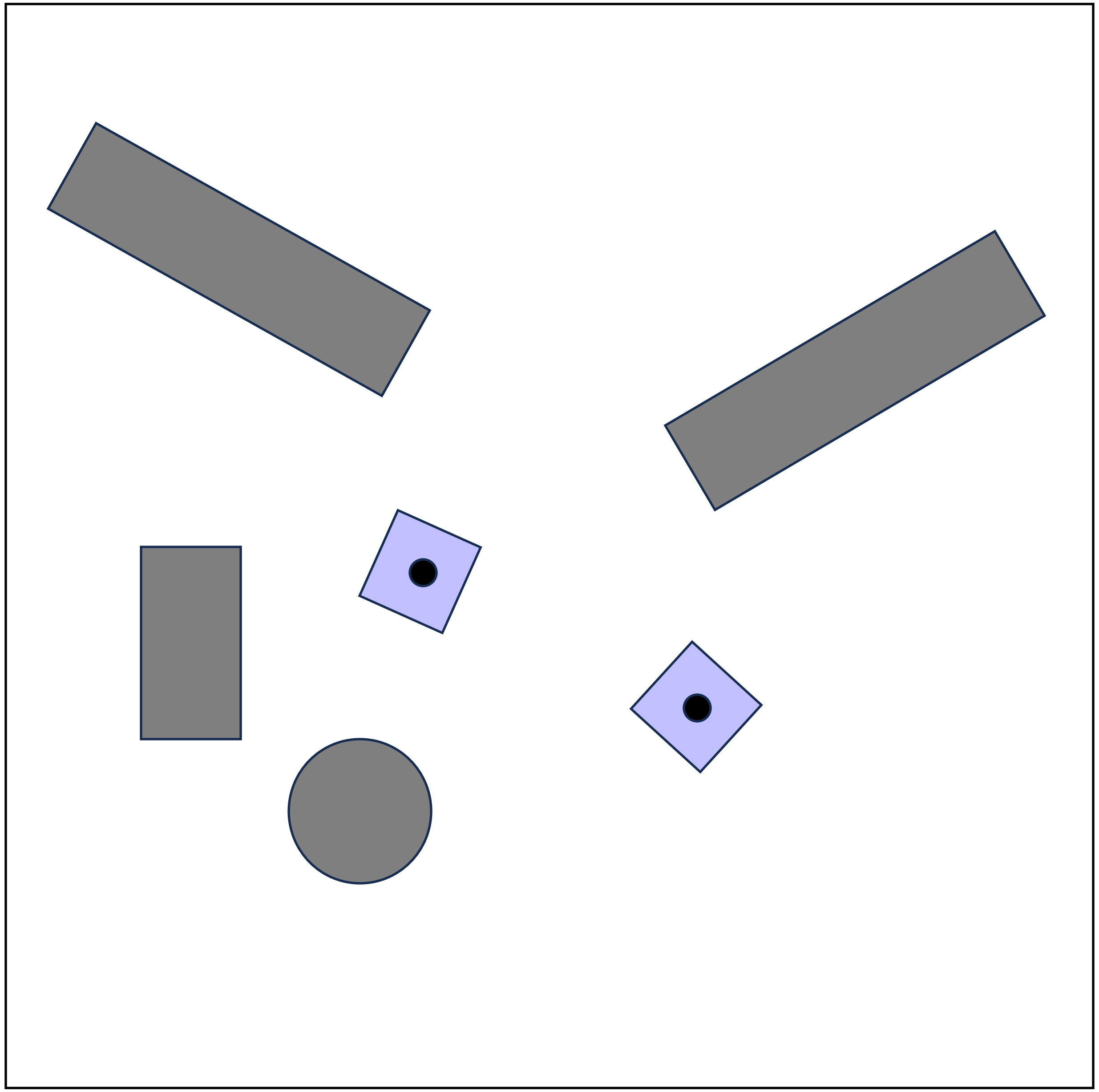
```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph

UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode

QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph

UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode

QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

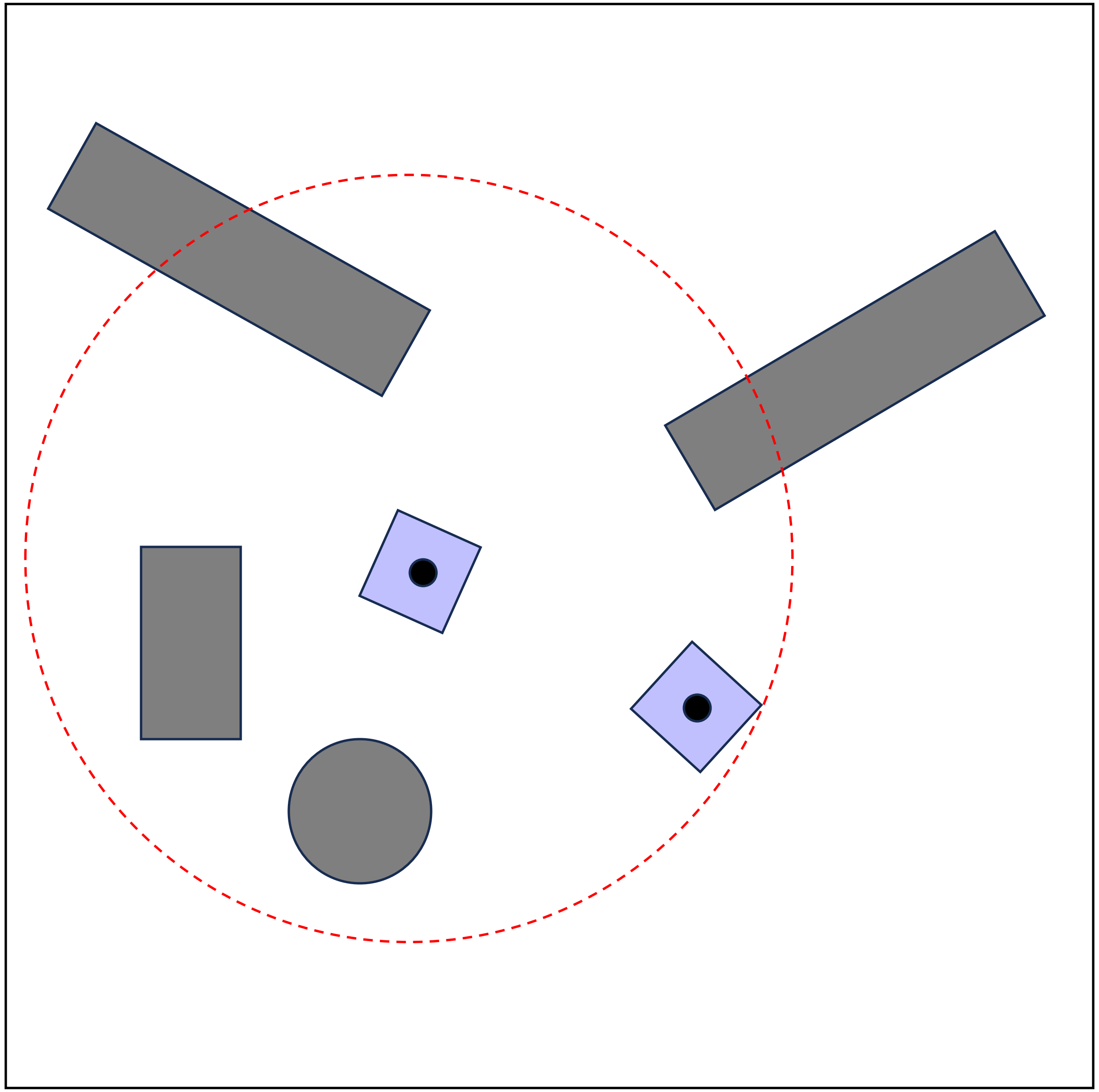
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample($\mathcal{X}$)
    // Skip if not feasible
    if not $f(x)$: **continue**
    // Update the graph
    UpdatePRM(x, graph, f)
return graph

UpdatePRM(x, graph, f)

newNode = addNode(graph, x)
for node $\in$ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode

QueryPRM(x_0, x_g, graph, f)

initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)



BuildPRM($\mathcal{X}$, f)

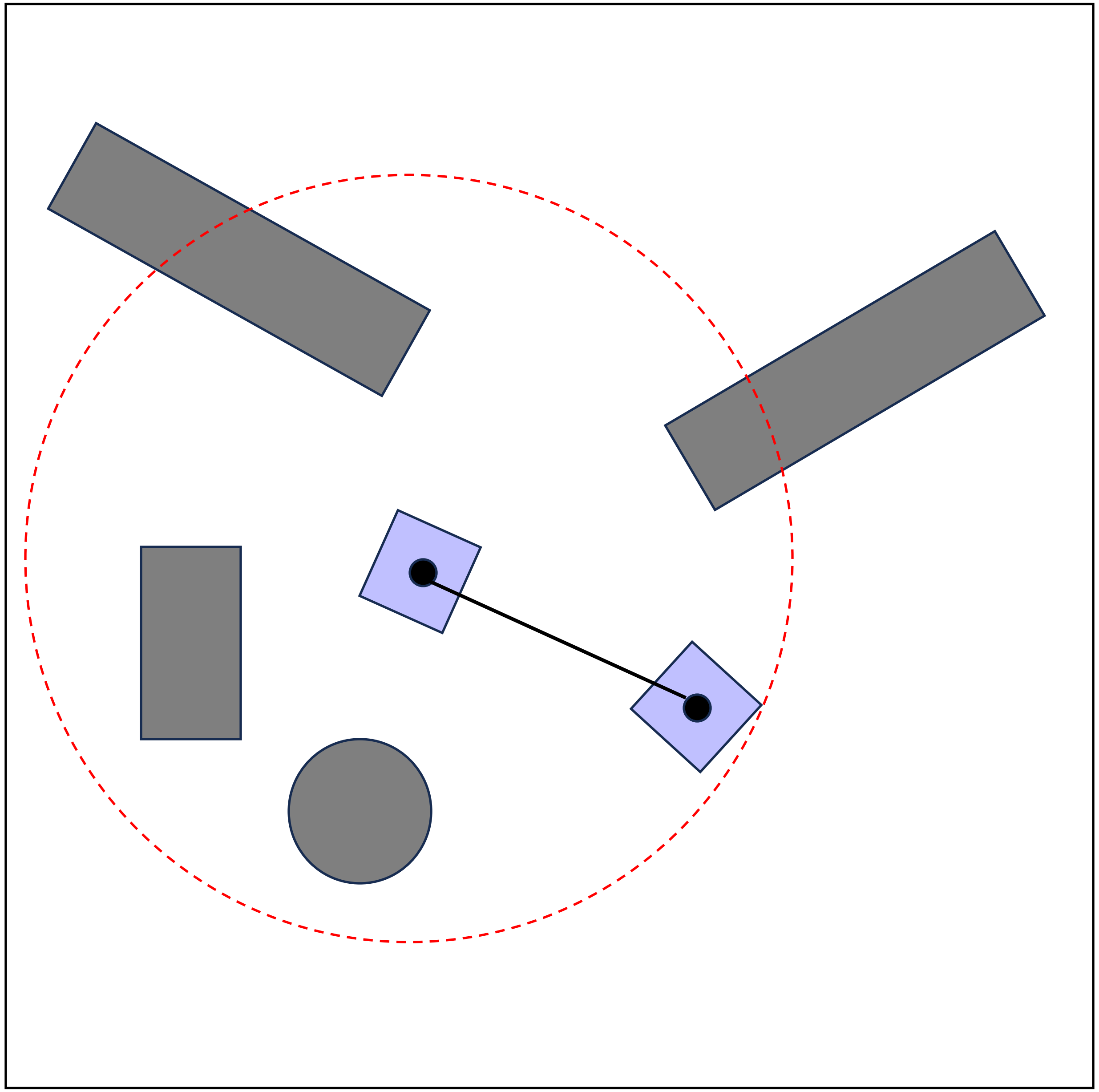
graph $=$ UndirectedGraph()
repeat:
  // Sample from configuration space
  x $=$ sample($\mathcal{X}$)
  // Skip if not feasible
  if not $f(x)$: **continue**
  // Update the graph
  UpdatePRM(x, graph, f)
return graph

UpdatePRM(x, graph, f)

newNode $=$ addNode(graph, x)
for node $\in$ getNeighbors(x)
  if pathFeasible(node.conf, x, f):
    addEdge(graph, node, newNode)
return newNode

QueryPRM(x_0, x_g, graph, f)

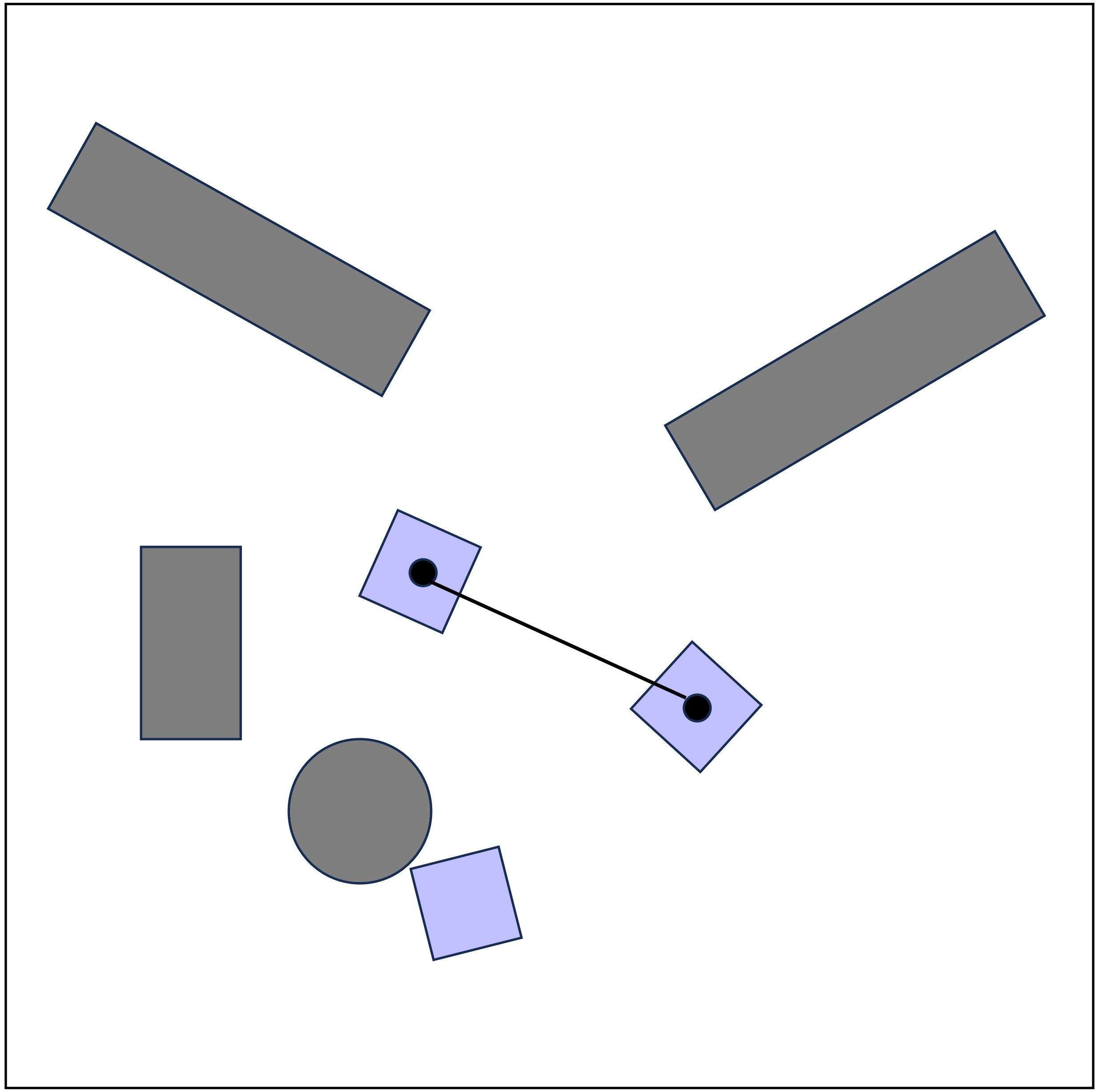
initNode $=$ UpdatePRM(x_0, graph, f)
goalNode $=$ UpdatePRM(x_g, graph, f)
nodePath $=$ graphSearch(initNode, goalNode)
return finish(nodePath)



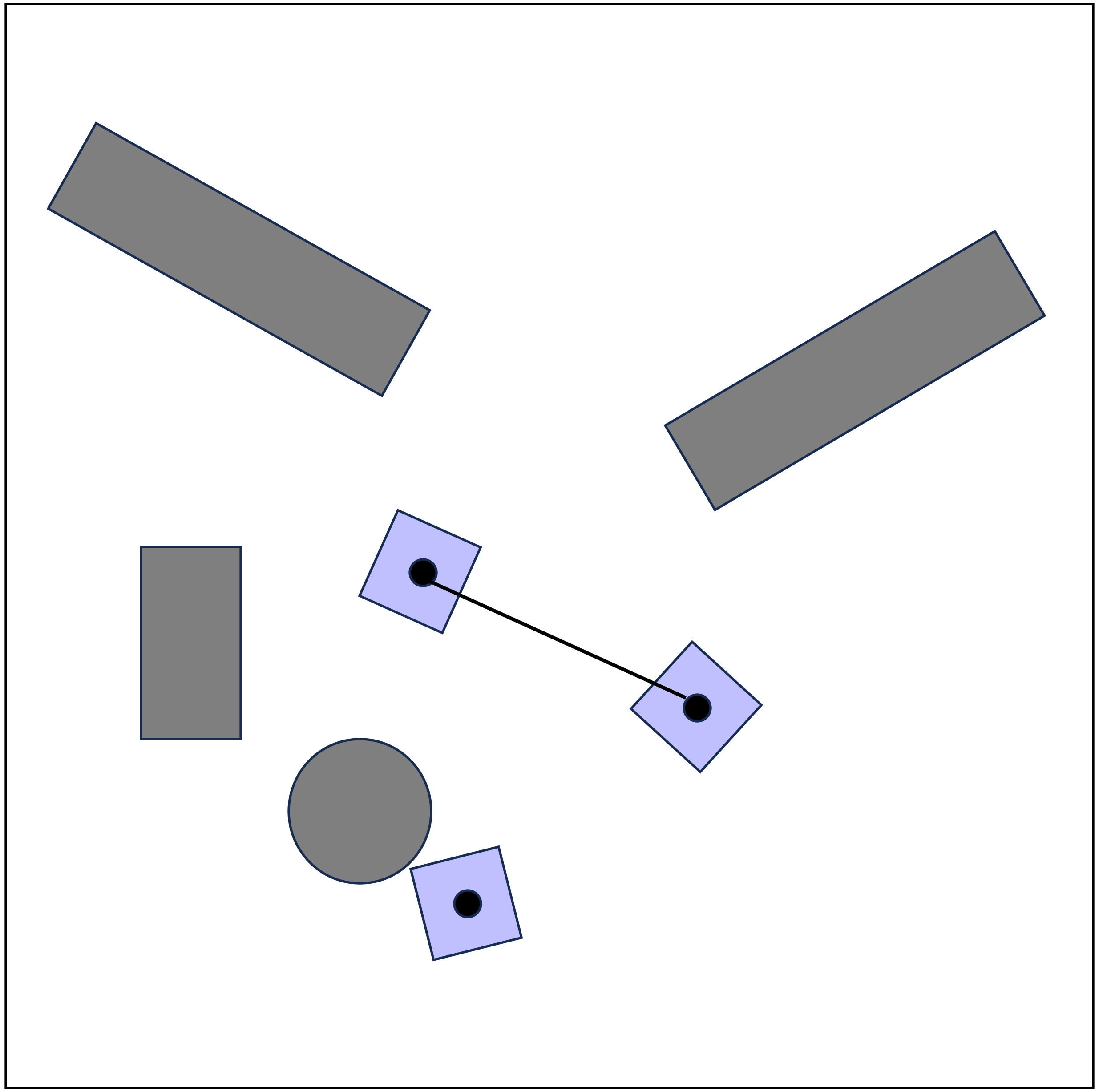
```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



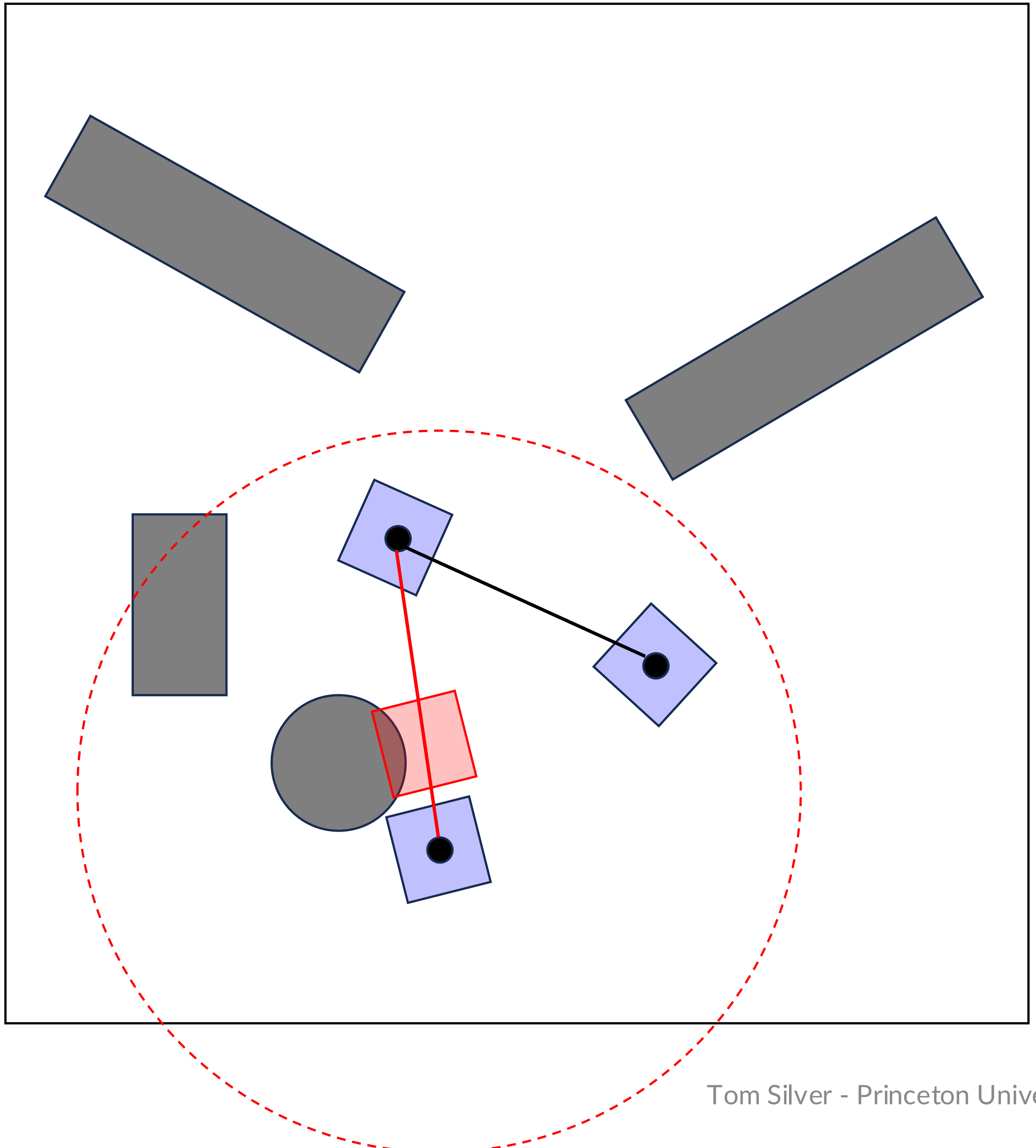
```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph

UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode

QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x):
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

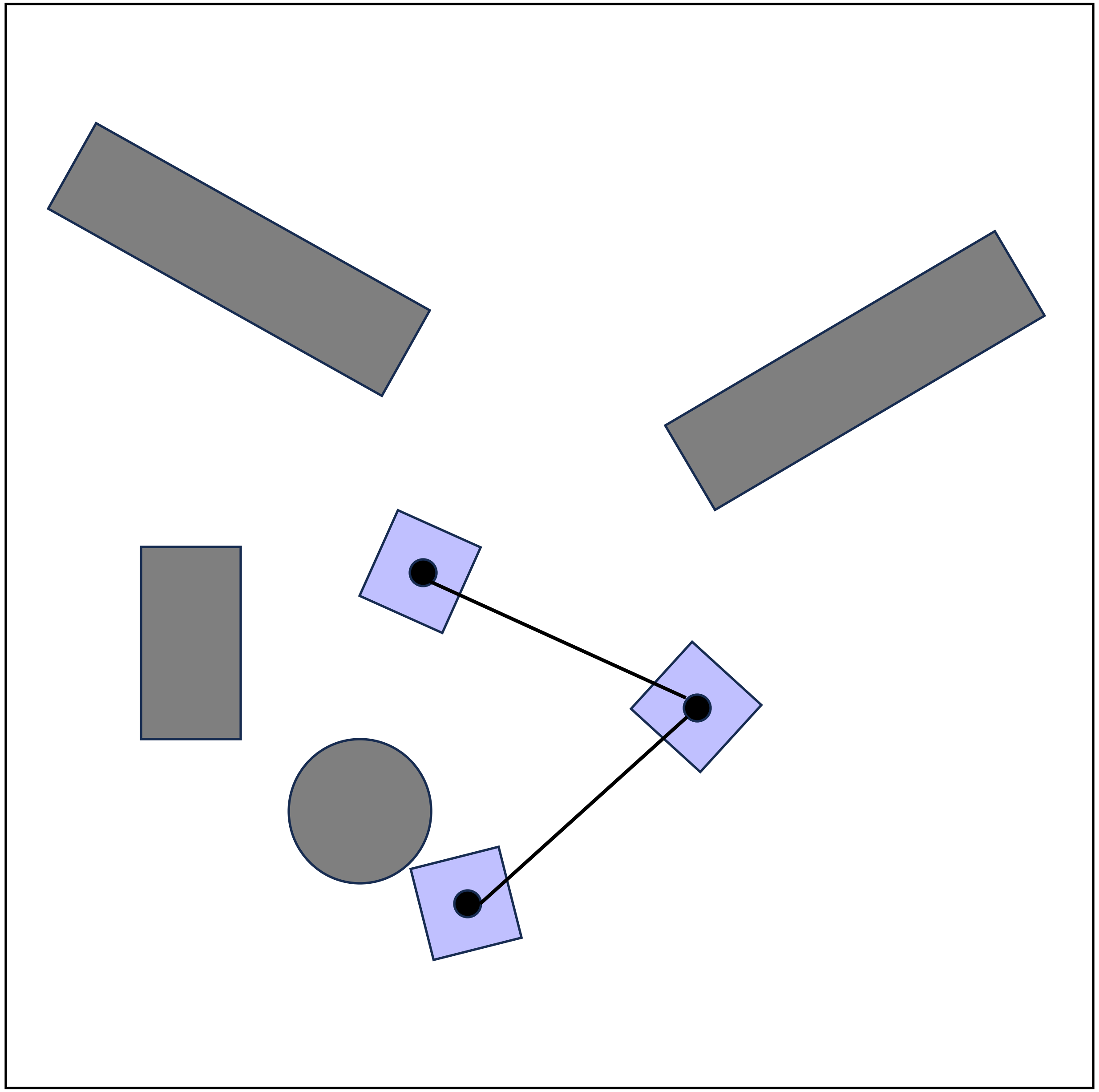
graph $=$ `UndirectedGraph()`
repeat:
    // Sample from configuration space
    x $=$ `sample`($\mathcal{X}$)
    // Skip if not feasible
    if not $f(x)$: **continue**
    // Update the graph
    `UpdatePRM`(x, `graph`, f)
return `graph`

UpdatePRM(x, `graph`, f)

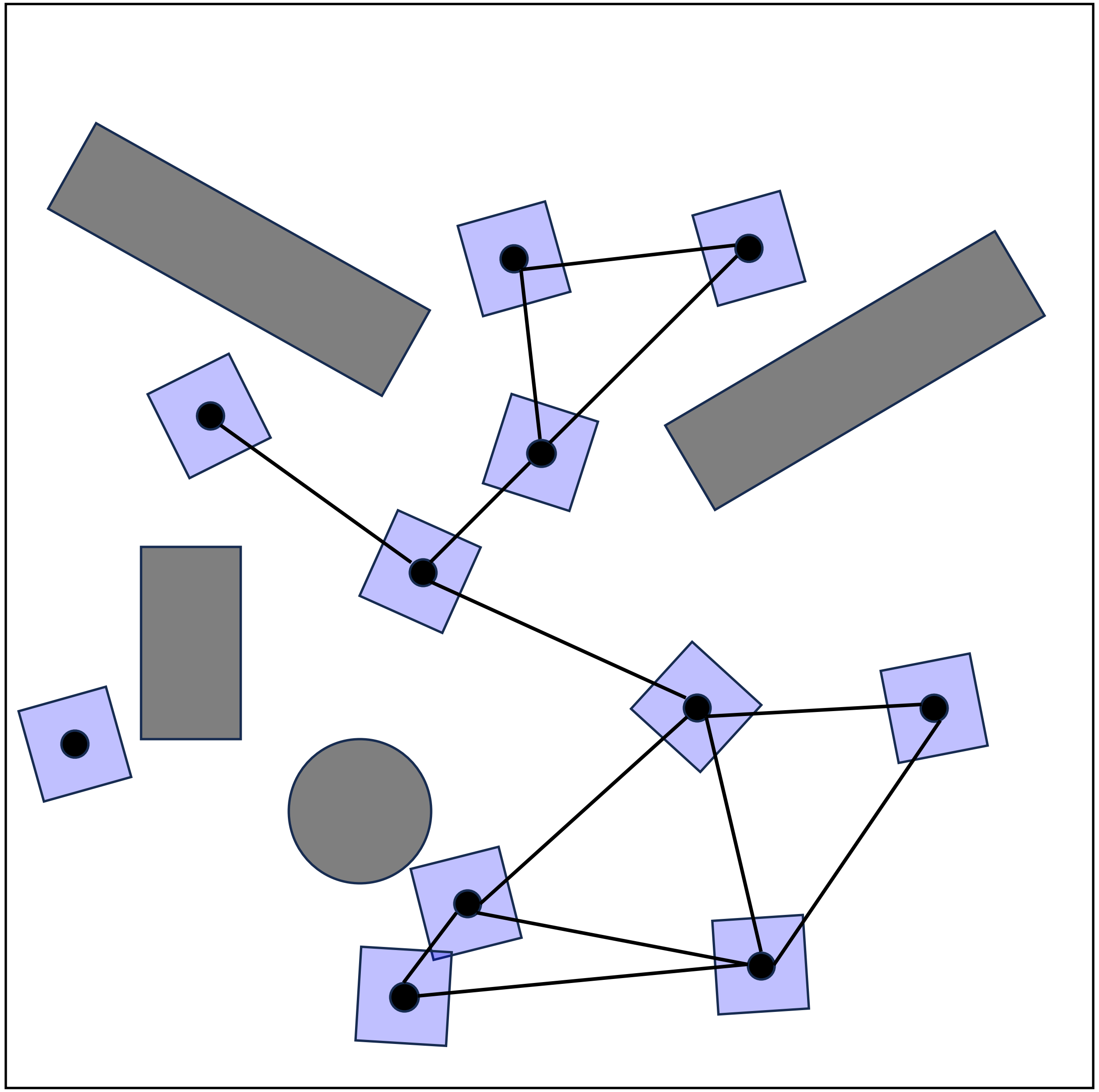
`newNode` $=$ `addNode`(`graph`, x)
for `node` $\in$ `getNeighbors`(x)
    if `pathFeasible`(`node.conf`, x, f):
        `addEdge`(`graph`, `node`, `newNode`)
return `newNode`

QueryPRM(x_0, x_g, `graph`, f)

`initNode` $=$ `UpdatePRM`(x_0, `graph`, f)
`goalNode` $=$ `UpdatePRM`(x_g, `graph`, f)
`nodePath` $=$ `graphSearch`(`initNode`, `goalNode`)
return `finish`(`nodePath`)



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph

UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode

QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

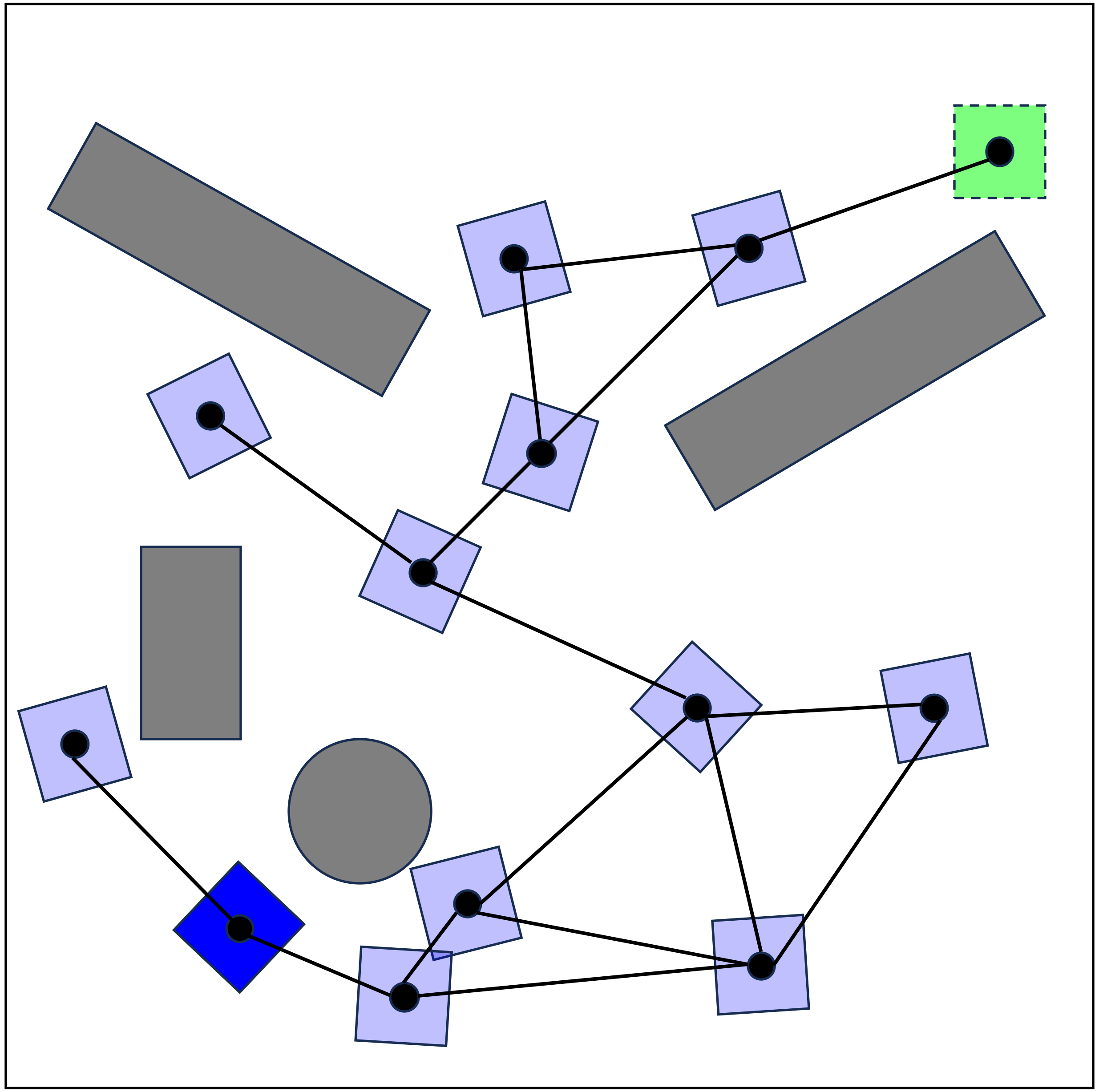
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

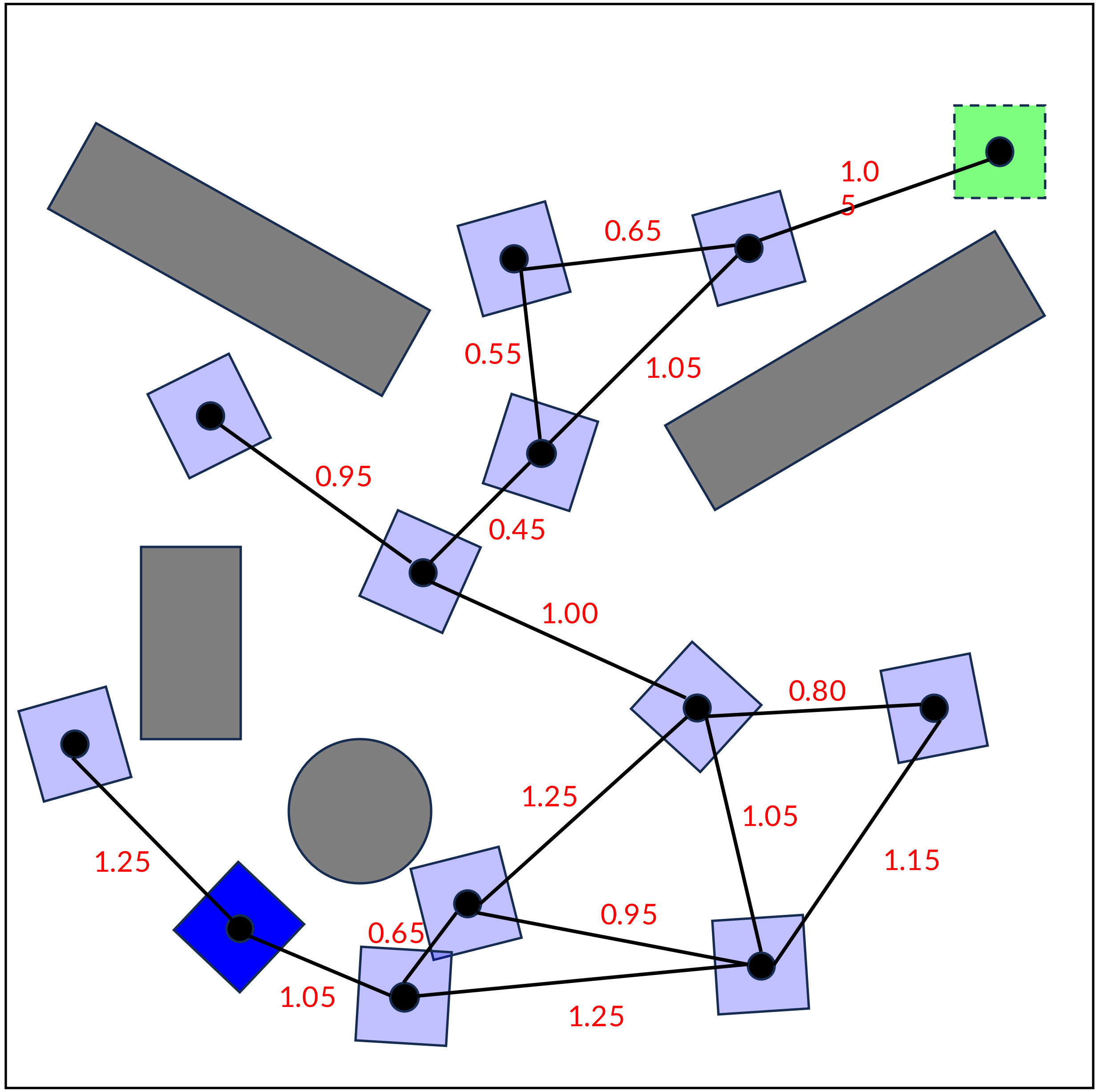
```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph

UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode

QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

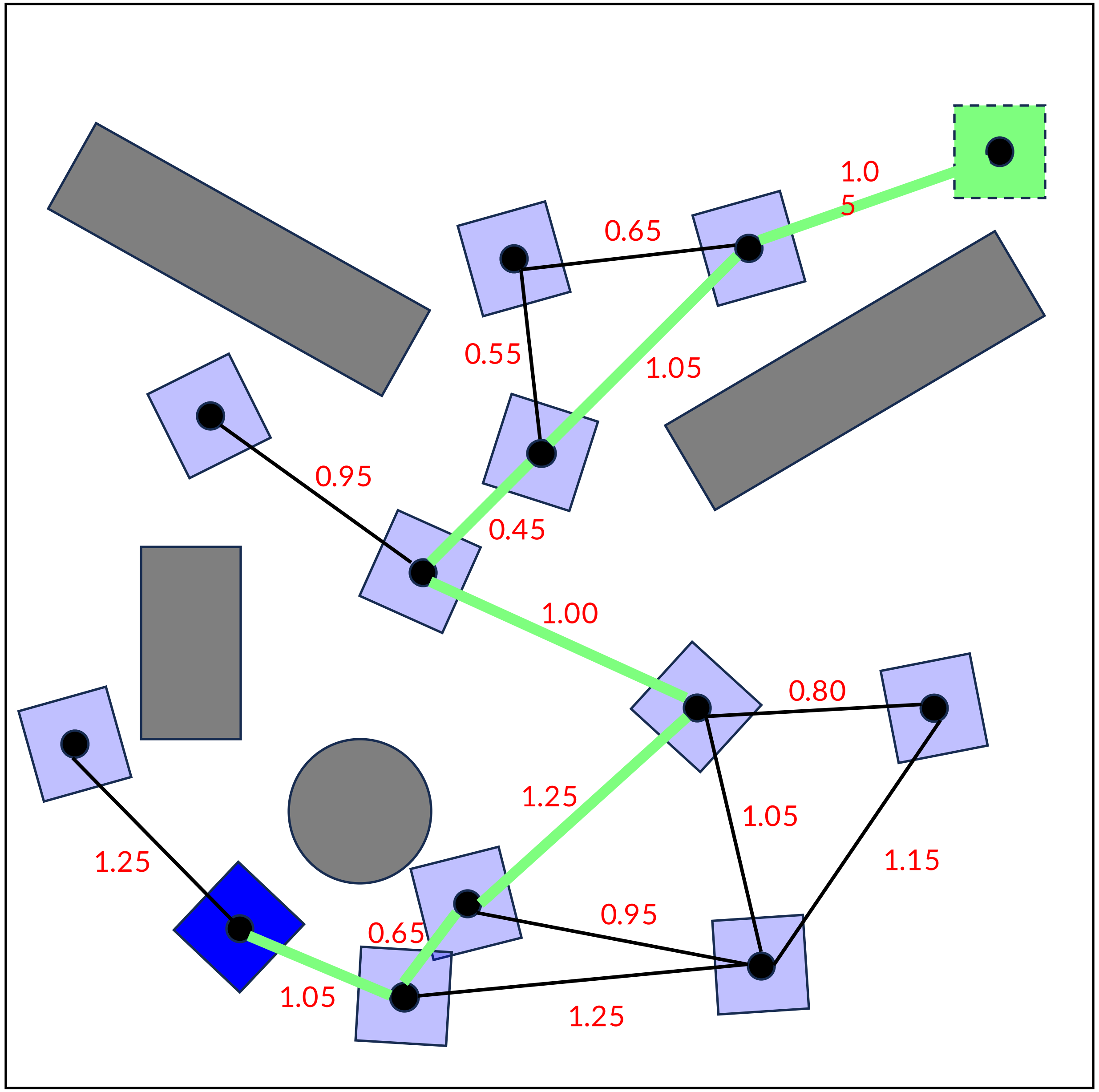
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BUILDPRM($\mathcal{X}$, f)

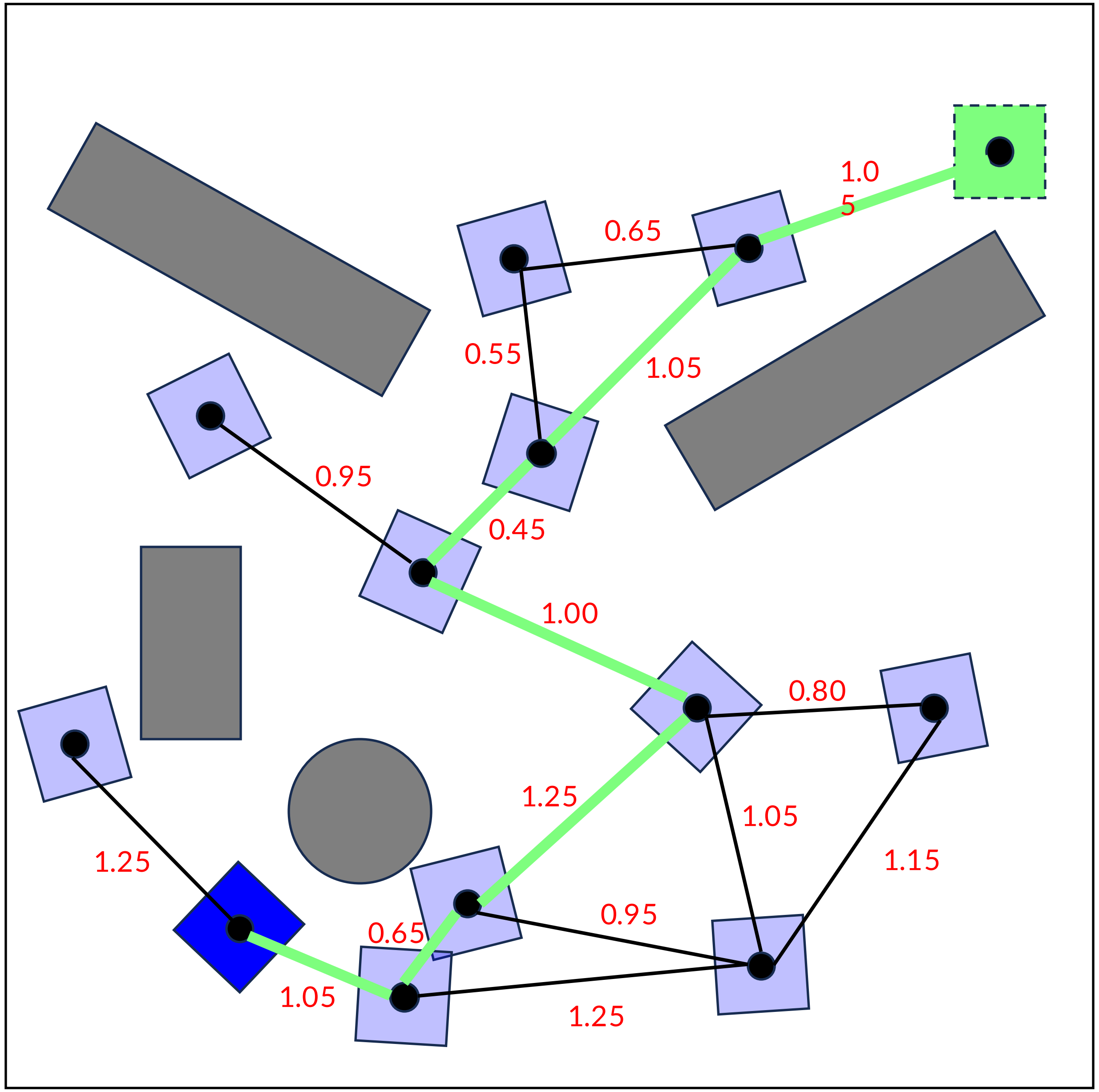
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UPDATEPRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QUERYPRM(x_0, x_g, graph, f)

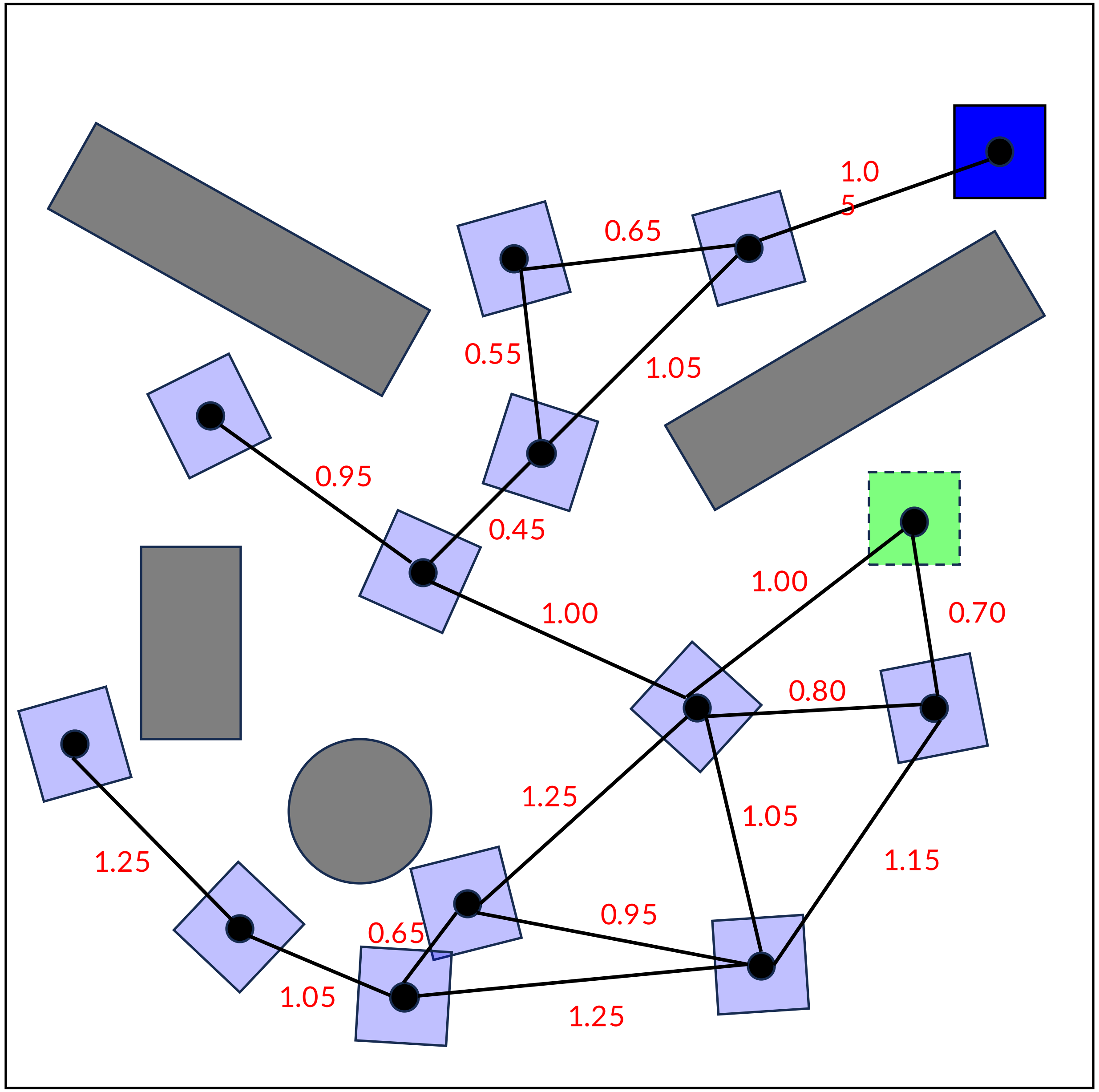
```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```




```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

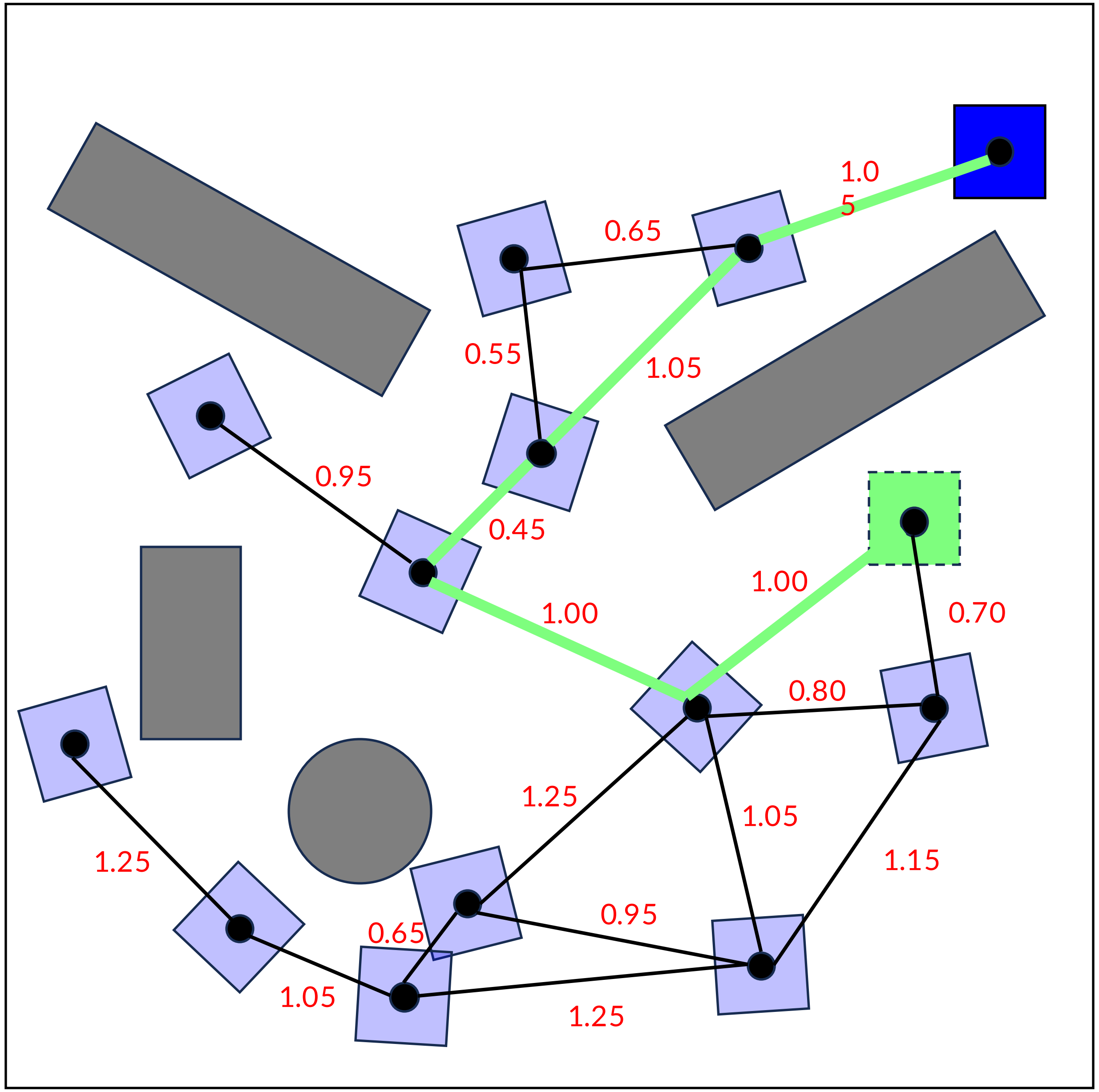
```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```




```
BUILDPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph

UPDATEPRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, nc
return newNode

QUERYPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```

Reuse graph for new query



BuildPRM$(\mathcal{X}, f)$

```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, no
return newNode
```

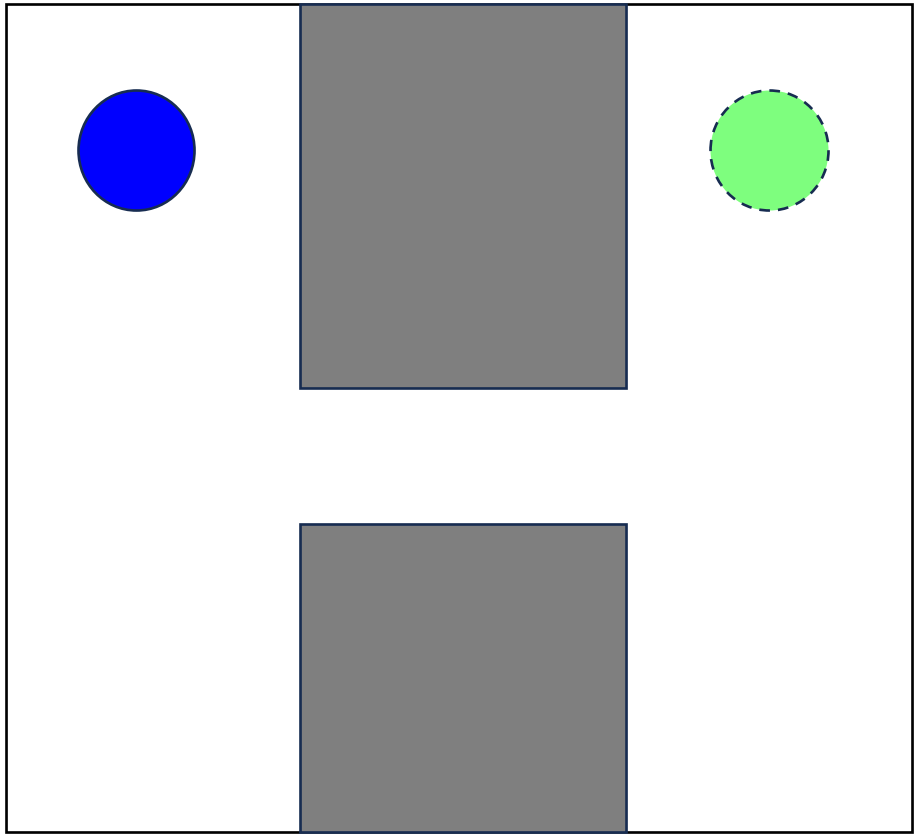
Reuse graph for new query

QueryPRM$(x_0, x_g, \text{graph}, f)$

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```

When and Why Does Sampling-Based Motion Planning Work?

- Works well when every configuration “sees” a significant fraction of feasible space
- Prototypical bad situation: narrow passages
- If interested: there are formal guarantees in terms of ϵ-goodness and β-lookout



Implementation Considerations

1. Sampling

Need to be careful depending on configuration space

Example: Sampling uniformly on a sphere requires some thought

Introduction.

This post gives a comprehensive list of the twenty most frequent and useful methods to uniformly sample from a the surface of a d-sphere, and the interior of the d-ball.

https://extremelearning.com.au/how-to-generate-uniformly-random-points-on-n-spheres-and-n-balls/

Implementation Considerations

1. Sampling

2. Distance metrics

Need to be careful when angles are involved

Example: Weighted distance function in SE(2)

$$d(q_1, q_2) = \sqrt{(q_1.x - q_2.x)^2 + (q_1.y - q_2.y)^2 + w(q_1.\theta - q_2.\theta)^2}$$

Implementation Considerations

1. Sampling
2. Distance metrics
3. Nearest neighbors

Can often do better than naïve quadratic algorithm

Example: KD trees (for Euclidean configuration spaces)

Implementation Considerations

1. Sampling
2. Distance metrics
3. Nearest neighbors
4. Collision checking

Usually the speed bottleneck. Use existing optimized libraries!

Implementation Considerations

1. Sampling
2. Distance metrics
3. Nearest neighbors
4. Collision checking
5. Interpolation

Linear, polynomial, trapezoidal…