Custom Project:

Collision Checking Tutorial in Ruby

Student name: Khang Vo
Student ID: 104220556

1. Problem

- In game development, there are numerous situations where we have to check for collision between two characters. For example: Checking collision between player and bullet, between enemy and bullet, or between player and enemy, etc.

- In some cases, where the character has the same width for all dimensions, we can measure the distance between two character to check for collision. However, in other cases, that method would be very inaccurate.

- We will solve this problem by using Separating Axis Theorem (SAT) collision checking method.

2. Definition

- The SAT states that if there exists at least one axis onto which there is a gap between the projections of two convex objects, the two objects must not be collided.

- For example, the following image shows two objects and a separating axis. There is a gap between the projections of two objects on that axis, so the two objects must be separated.

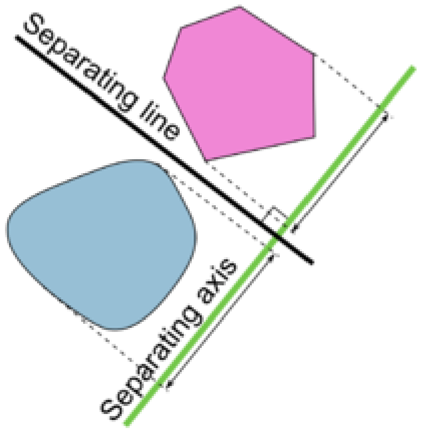


Figure 1: Illustration of Separating Axis Theorem (source: Wikipedia)

- However, this theorem only works for convex objects.

- In this tutorial, it will be simplified to only check collisions between rectangular hitboxes whose sides are in parallel with horizontal axis or vertical axis.

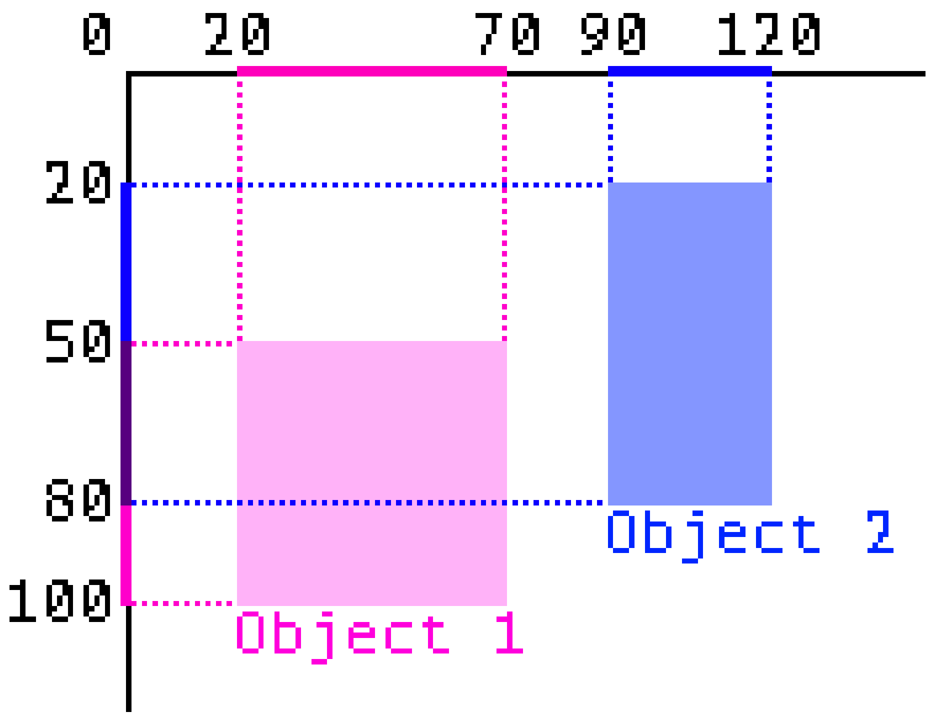
3. Tutorial

- Step 1: create a hitbox with left, right, top, bottom variables to indicates four sides of the hitbox, then assign a hitbox for each object.

```ruby
class Hitbox
  attr_accessor :top, :bottom, :left, :right
  def initialize(top, bottom, left, right)
    @top = top
    @bottom = bottom
    @left = left
    @right = right
  end
end
```

```ruby
class Object1
  attr_accessor :hitbox
  def initialize
    @hitbox = Hitbox.new(50, 100,
20, 70)
  end
end
```

```ruby
class Object2
  attr_accessor :hitbox
  def initialize
    @hitbox = Hitbox.new(20, 80,
90, 120)
  end
end
```



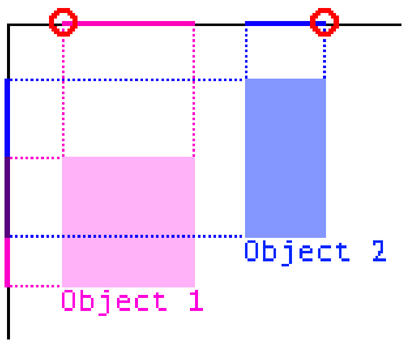
- Step 2: define a collision checking function that receive two hitboxes.

```ruby
def collision_checking(hitbox1, hitbox2)
  ...
end
```

- Step 3: begin with the horizontal axis, find the left most and the right most point of BOTH projections of the two hitboxes. This can be achieved by finding the minimum and maximum values among left and right values of each hitbox.

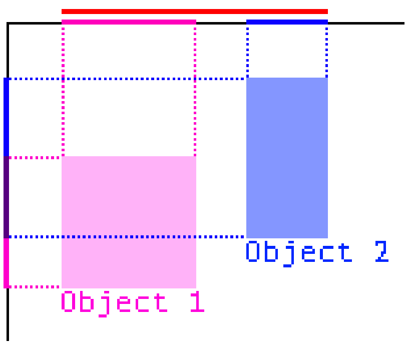
The reason why we have to check both left and right sides of each hitboxes is that we do not know which side of Object 2 that the Object 1 is on.

```
  left_most = [hitbox1.left, hitbox1.right, hitbox2.left,
hitbox2.right].min
  right_most = [hitbox1.left, hitbox1.right, hitbox2.left,
hitbox2.right].max
```



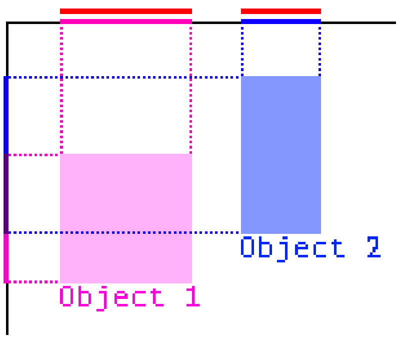
- Step 4: calculate the length from the left most point to the right most point that we just found above.

```
  length = right_most - left_most
```



- Step 5: calculate the sum of the width of two hitboxes.

```
  sum = (hitbox1.right - hitbox1.left) + (hitbox2.right -
hitbox2.left)
```

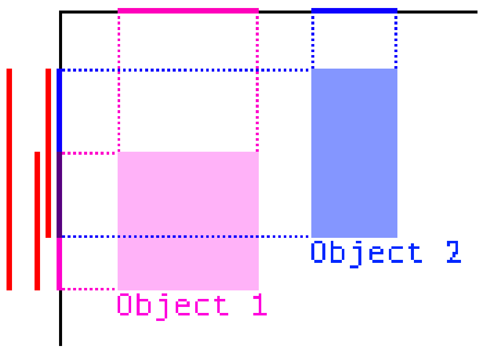


- Step 6: compare the calculated length and sum. If length > sum then there is a gap on this axis. Return false for no collision and exit the function.

```
  if length > sum
    return false
  end
```

- Step 7: if there is no gap, repeat steps 3-6 on the vertical axis.

```
  top_most = [hitbox1.top, hitbox1.bottom, hitbox2.top,
hitbox2.bottom].min
  bottom_most = [hitbox1.top, hitbox1.bottom, hitbox2.top,
hitbox2.bottom].max
  length = bottom_most - top_most
  sum = (hitbox1.bottom - hitbox1.top) + (hitbox2.bottom -
hitbox2.top)
  if length > sum
    return false
  end
```



- Step 8: if there is no gap on both axes, then the two hitboxes are collided. Return true for collision happened and exit the function.

```
  return true
```

4. Conclusion

- In conclusion, this SAT algorithm can be a useful and easy way to check for collisions between two convex bodies, especially for rectangular hitboxes.

- However, this method becomes more complicated and expensive to use when it is going to complex polygons, because we will have to check all the normals of each polygon’s edge.

- Also, if this algorithm detects any axis with a gap between projections, it can exit without checking other axes. In this case, the worst scenario is when collision happens, as it has to check all of the available axes to go to that conclusion.

- Nonetheless, we are working with well-oriented rectangular hitboxes, which has only two axes, so the difference is barely noticeable.

- Code:

```
def collision_checking(hitbox1, hitbox2)
  horizontal_projection = [hitbox1.left, hitbox1.right,
hitbox2.left, hitbox2.right]
  vertical_projection = [hitbox1.top, hitbox1.bottom, hitbox2.top,
hitbox2.bottom]

  # Check on horizontal axis
  length = horizontal_projection.max - horizontal_projection.min
  sum = (hitbox1.right - hitbox1.left) + (hitbox2.right -
hitbox2.left)
  return false if length > sum

  # Check on vertical axis
  length = vertical_projection.max - vertical_projection.min
  sum = (hitbox1.bottom - hitbox1.top) + (hitbox2.bottom -
hitbox2.top)
  return false if length > sum

  # Collided if no gap found
  return true
end
```

References

Chong, K. S. (2012, August 6). *Collision Detection Using the Separating Axis Theorem*. Retrieved December 2, 2022, from Envatotuts+: https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169

Wikipedia. (n.d.). *Hyperplane separation theorem*. Retrieved December 02, 2022, from Wikipedia: https://en.wikipedia.org/wiki/Hyperplane_separation_theorem