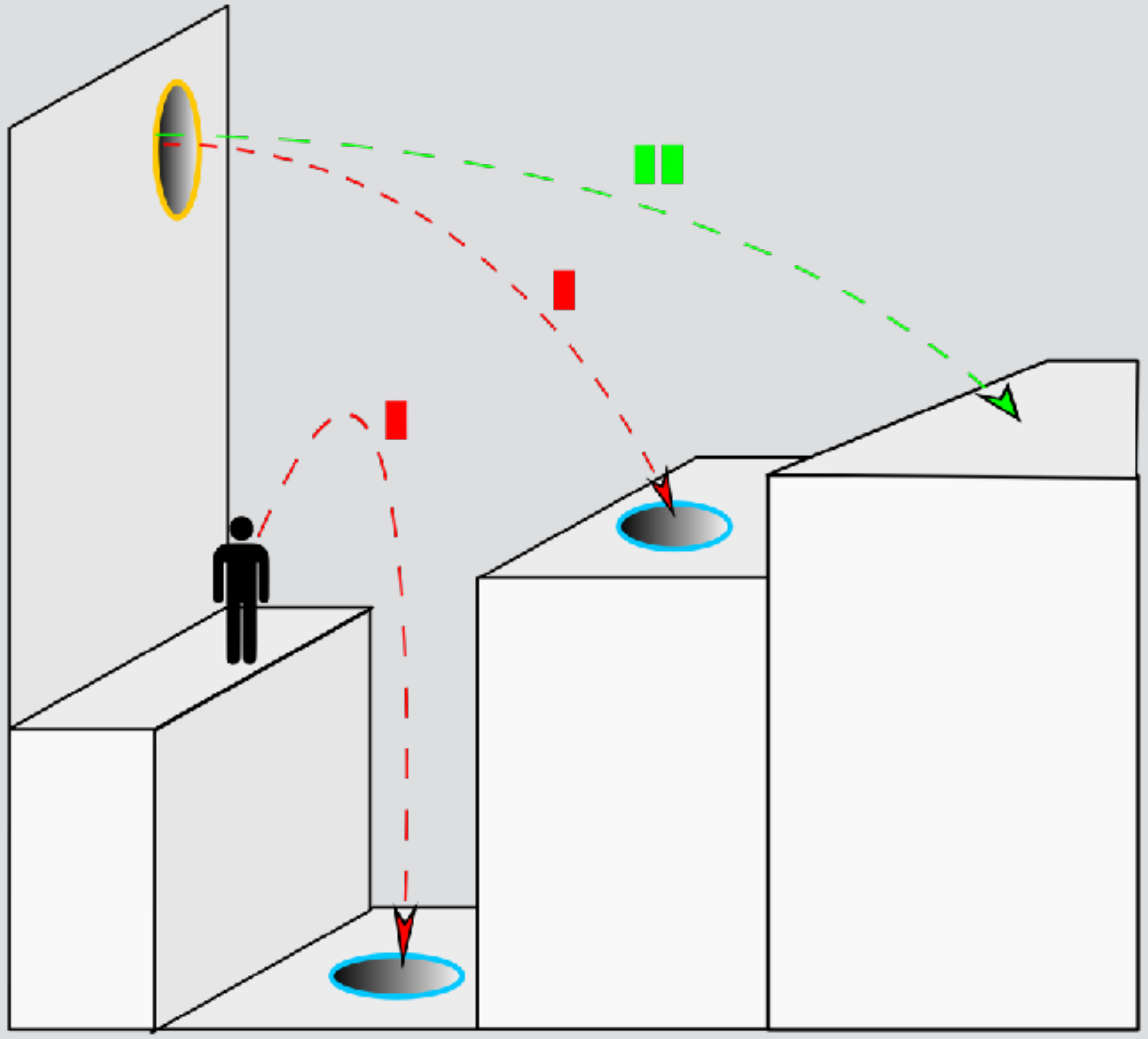
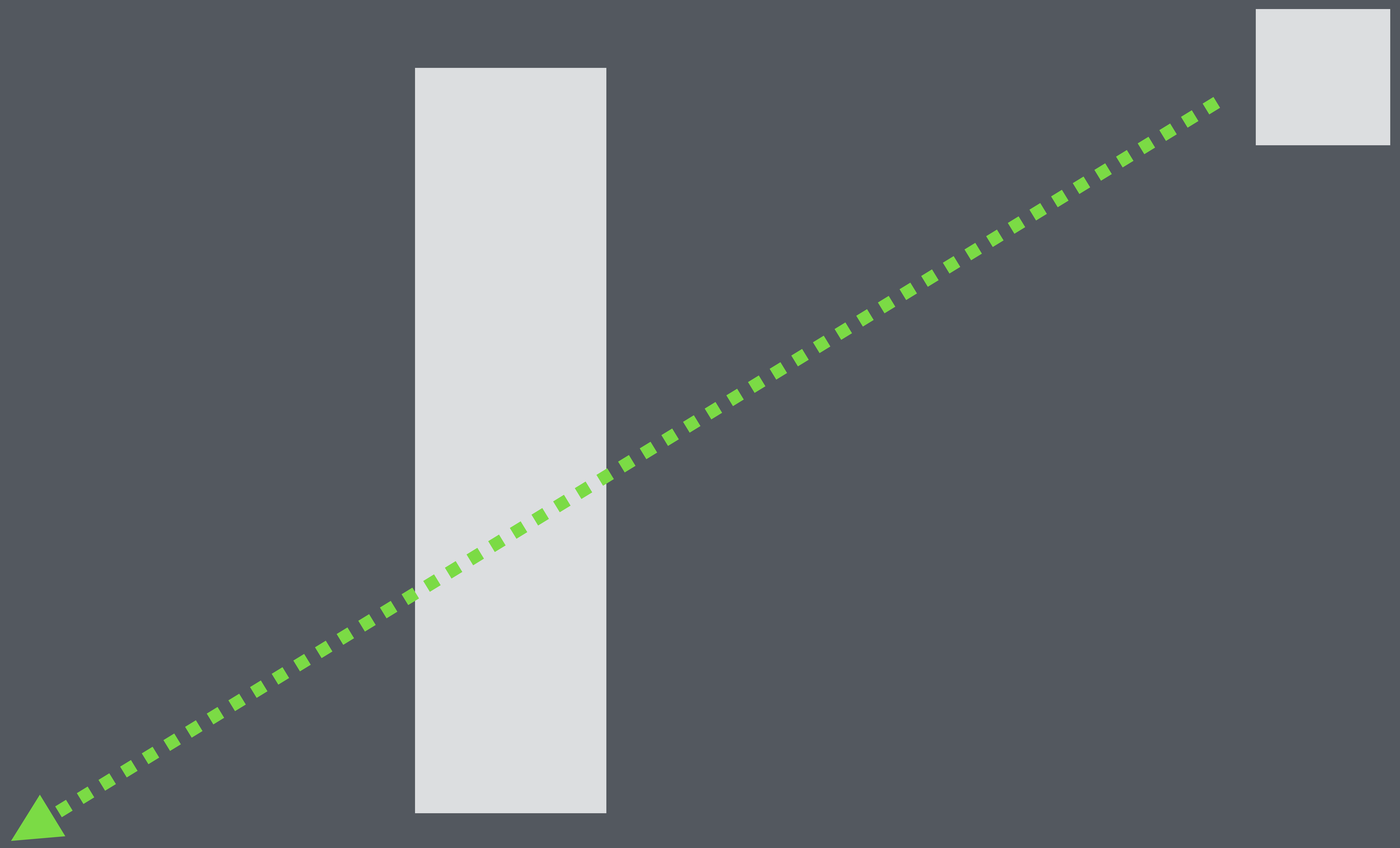
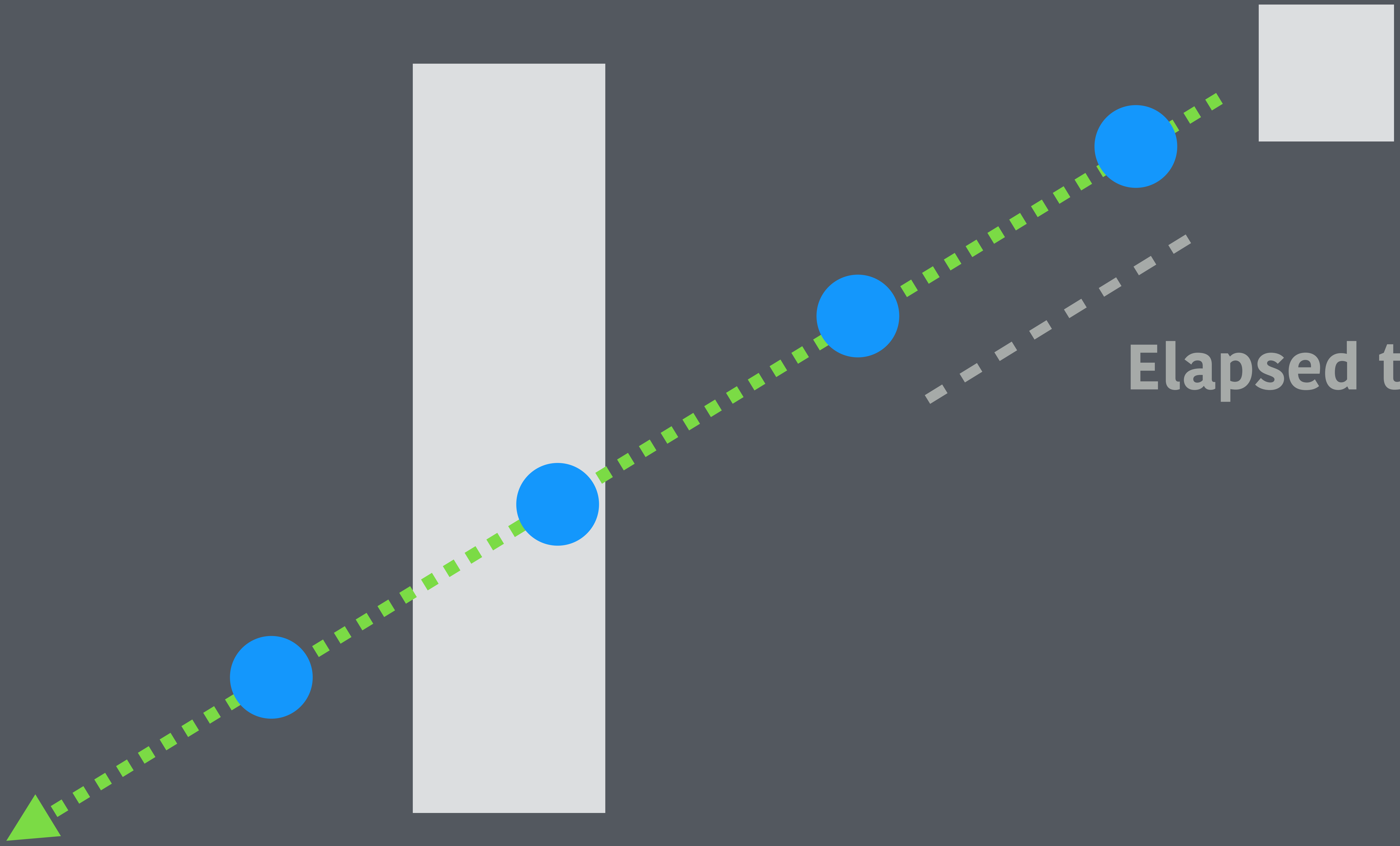
Basic physics and collision response.



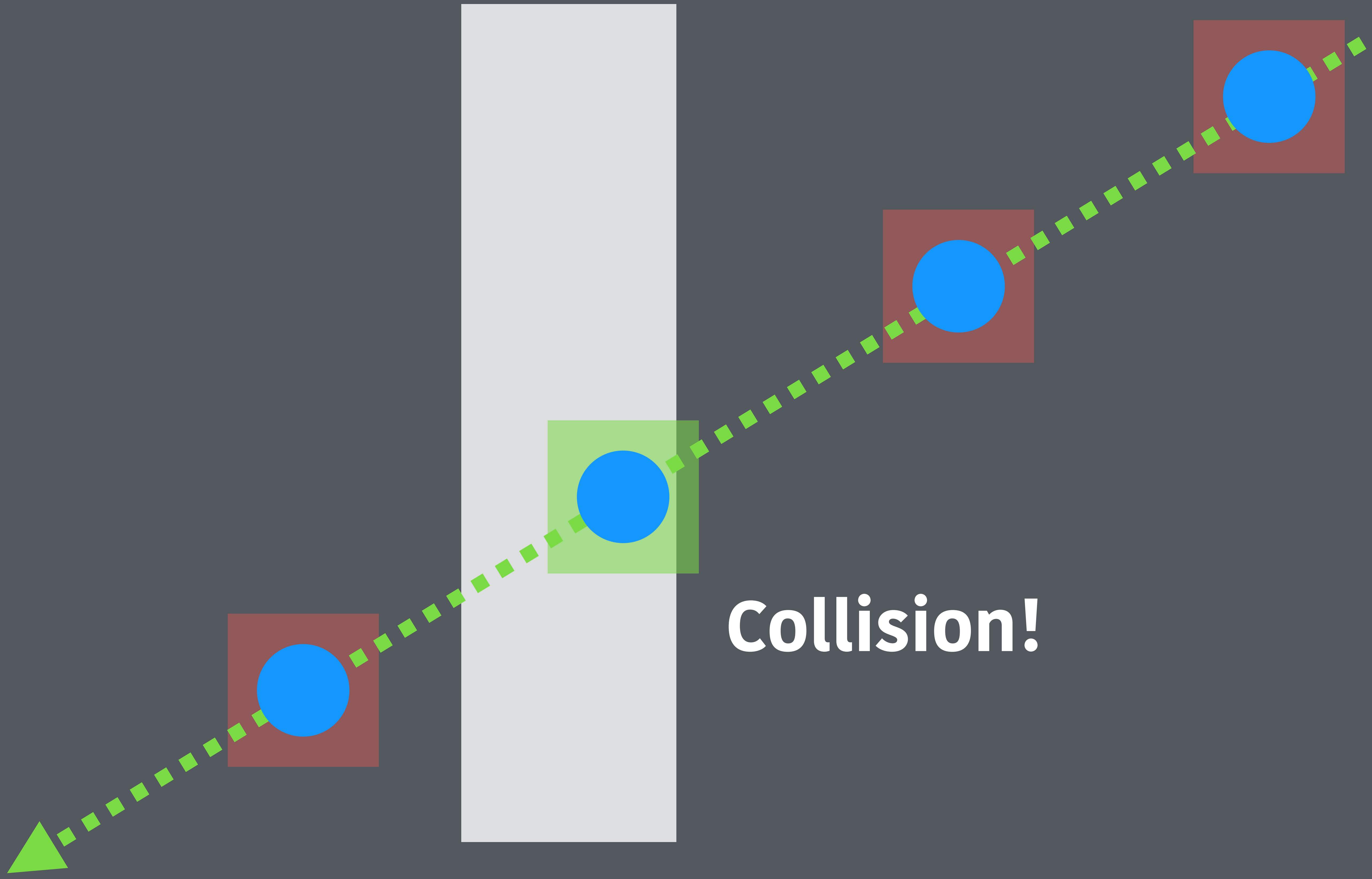
Fixing the timestep.

Problems with variable timestep.

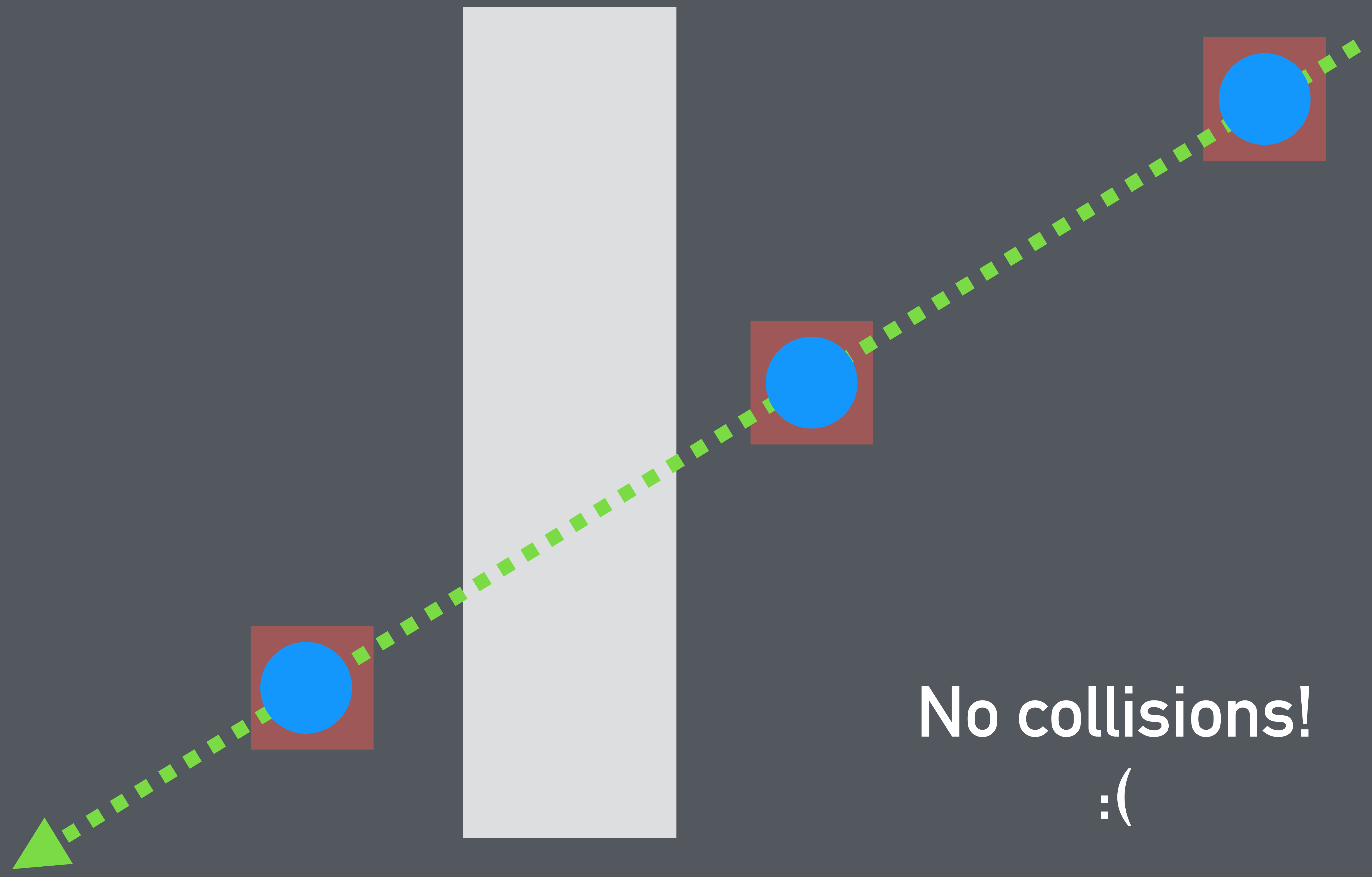




Elapsed time since last frame

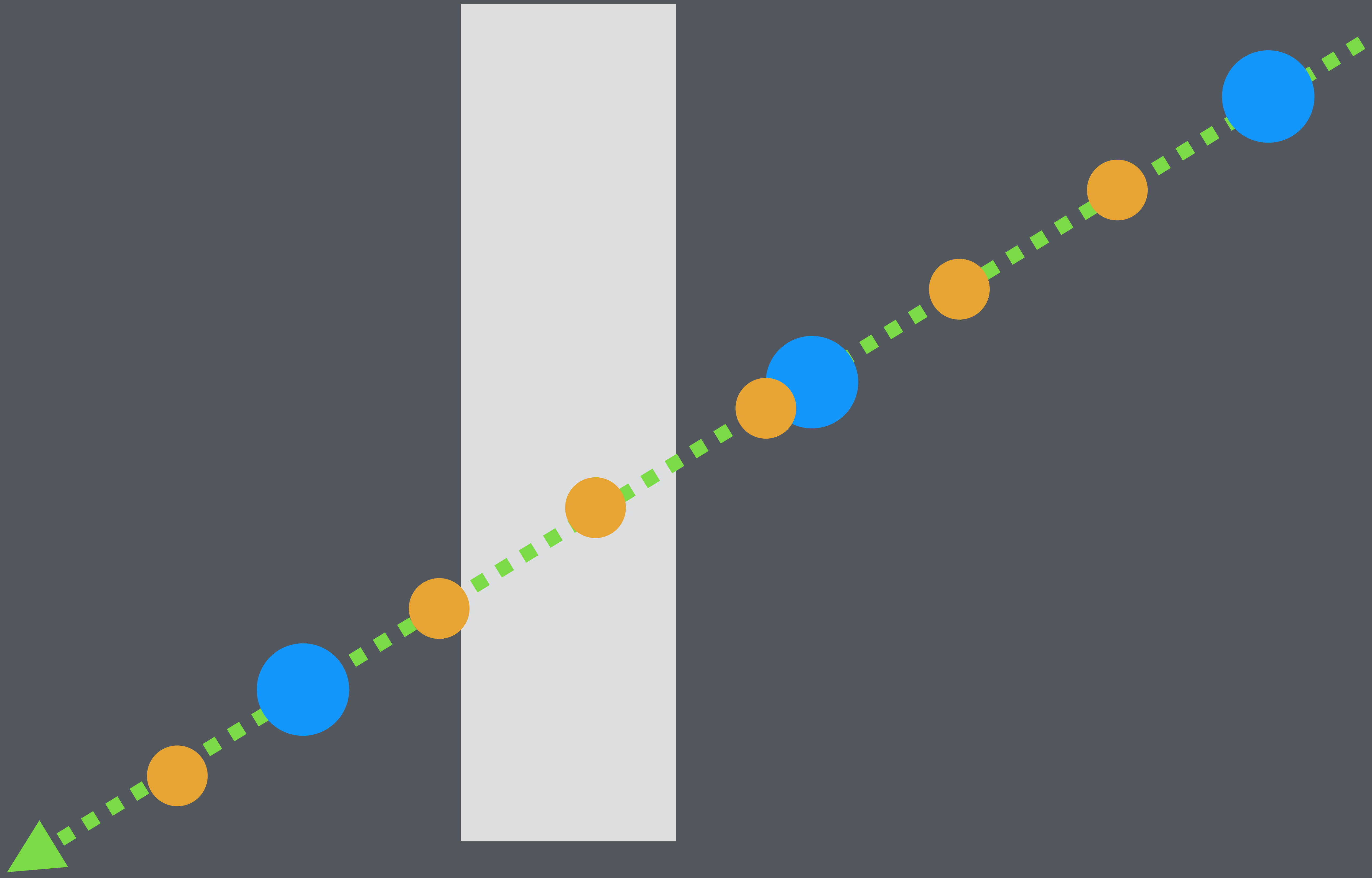


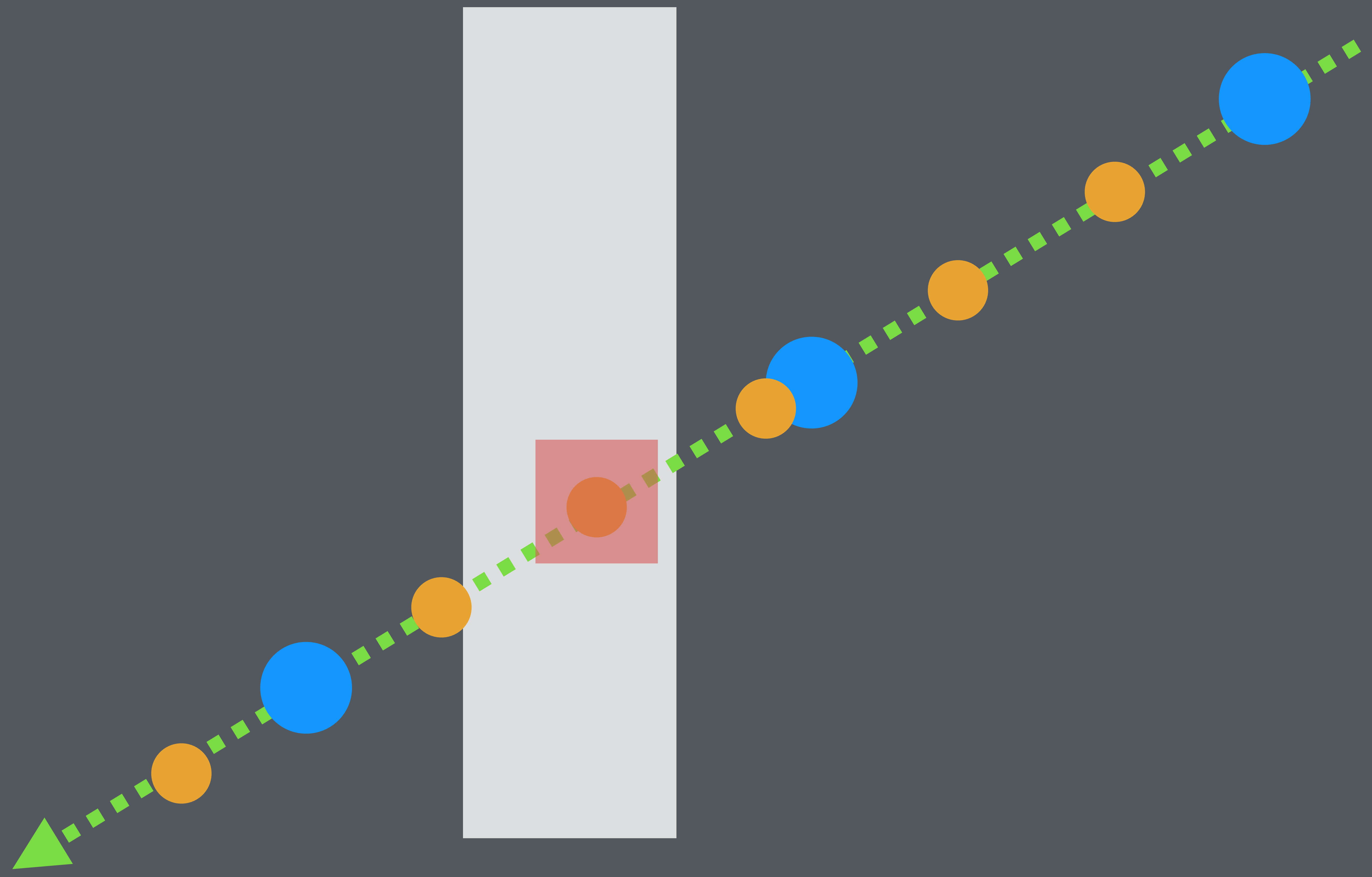
Collision!



No collisions!
:(

Fixed timestep.





```c
// 60 FPS (1.0f/60.0f)
#define FIXED_TIMESTEP 0.0166666f
#define MAX_TIMESTEPS 6




float fixedElapsed = elapsed;
if(fixedElapsed > FIXED_TIMESTEP * MAX_TIMESTEPS) {
    fixedElapsed = FIXED_TIMESTEP * MAX_TIMESTEPS;
}
while (fixedElapsed >= FIXED_TIMESTEP ) {
    fixedElapsed -= FIXED_TIMESTEP;
    Update(FIXED_TIMESTEP);
}
Update(fixedElapsed);
```

Basic game physics.

Velocity and acceleration.

Velocity.

The rate of change of the position of an object.

(speed * direction)

```
position_x += velocity_x * elapsed;
position_y += velocity_y * elapsed;
```

Acceleration.

The rate of change of velocity.

```
velocity_x += acceleration_x * elapsed;
velocity_y += acceleration_y * elapsed;
```

Friction.

Friction.

The rate of decrease of velocity.

```
velocity_x = lerp(velocity_x, 0.0f, elapsed * friction_x);
velocity_y = lerp(velocity_y, 0.0f, elapsed * friction_y);
```

Lerp?

LERP

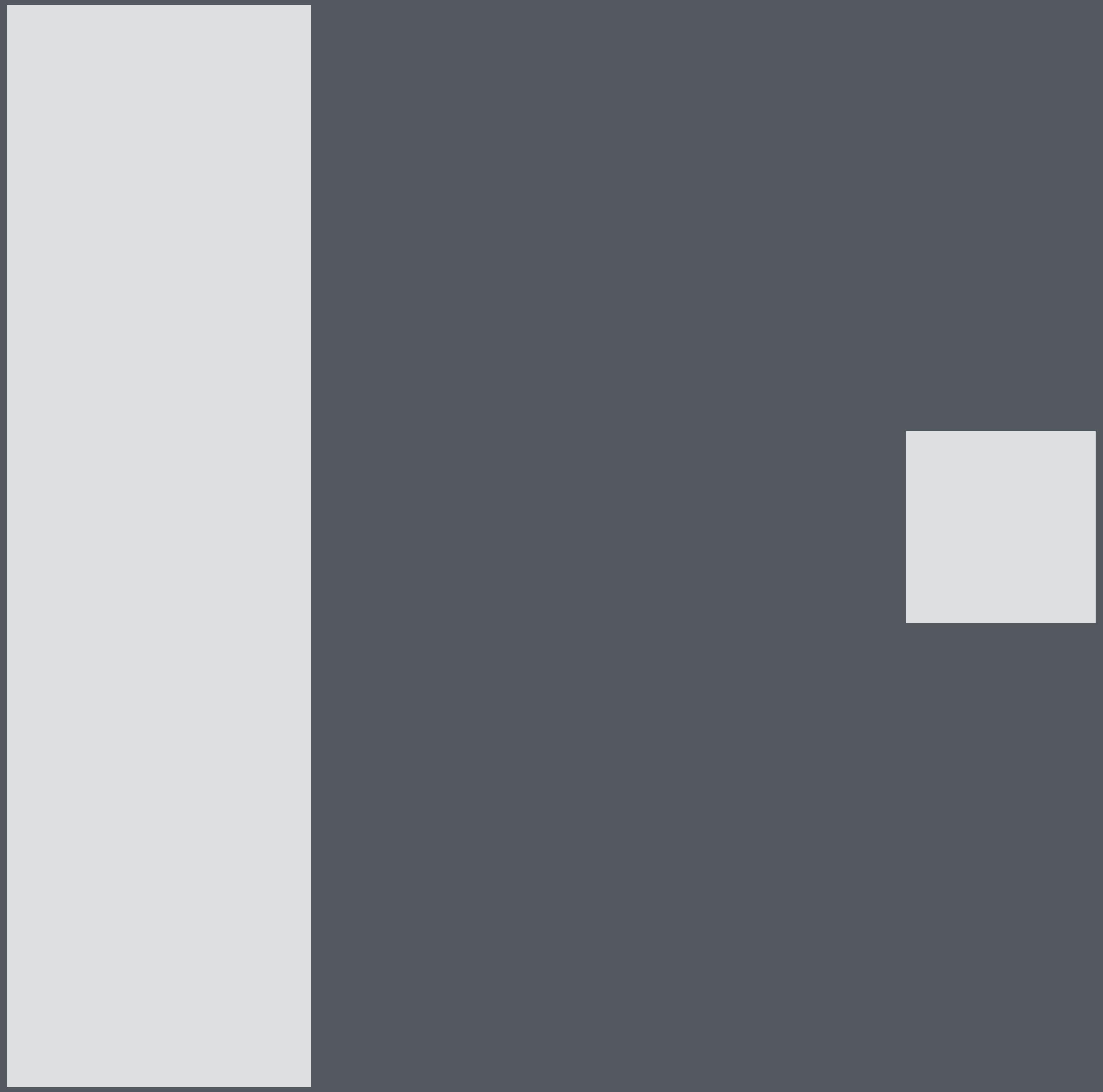
LinEar InteRPolation

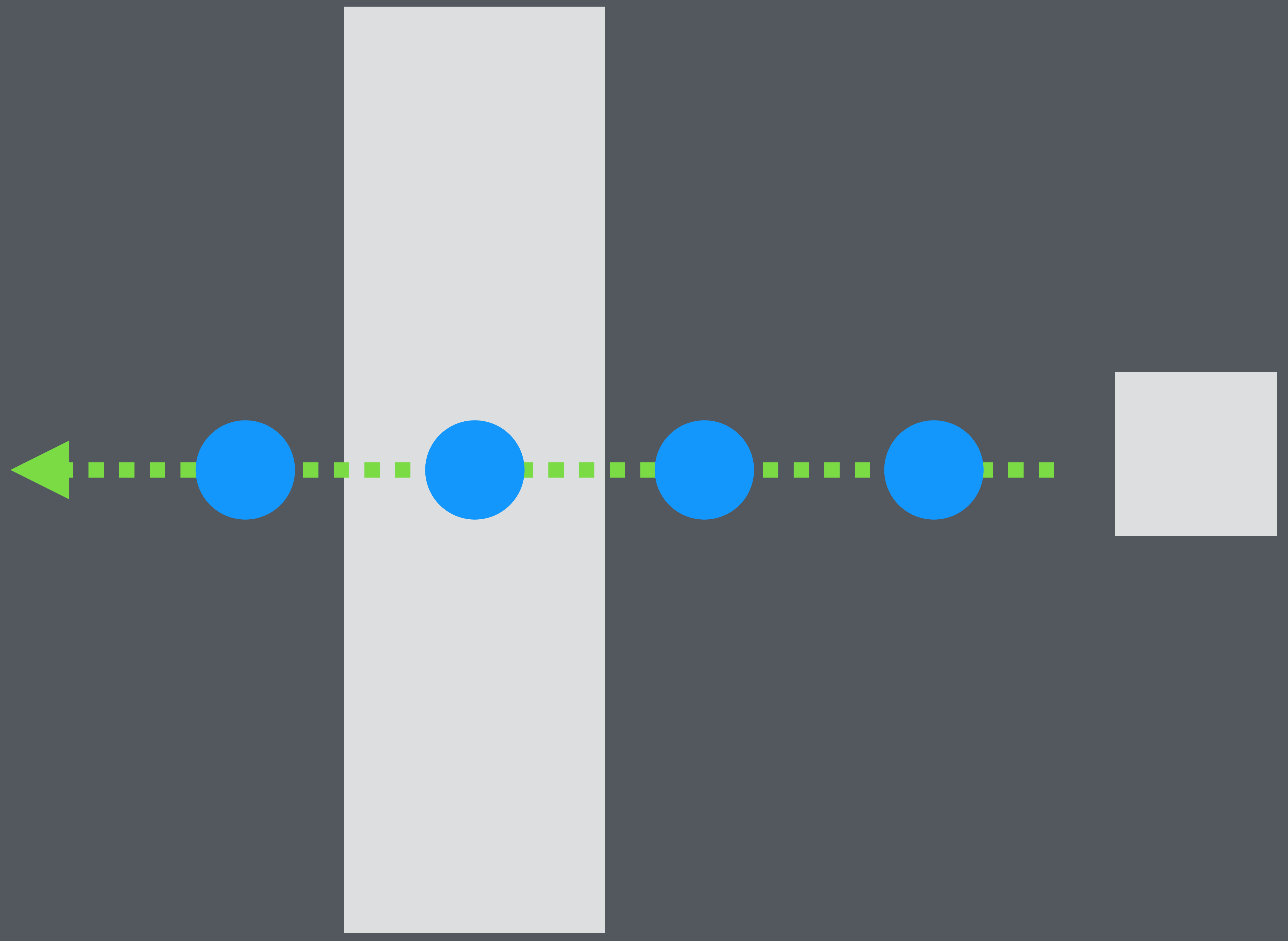
```
float lerp(float v0, float v1, float t) {
    return (1.0-t)*v0 + t*v1;
}
```

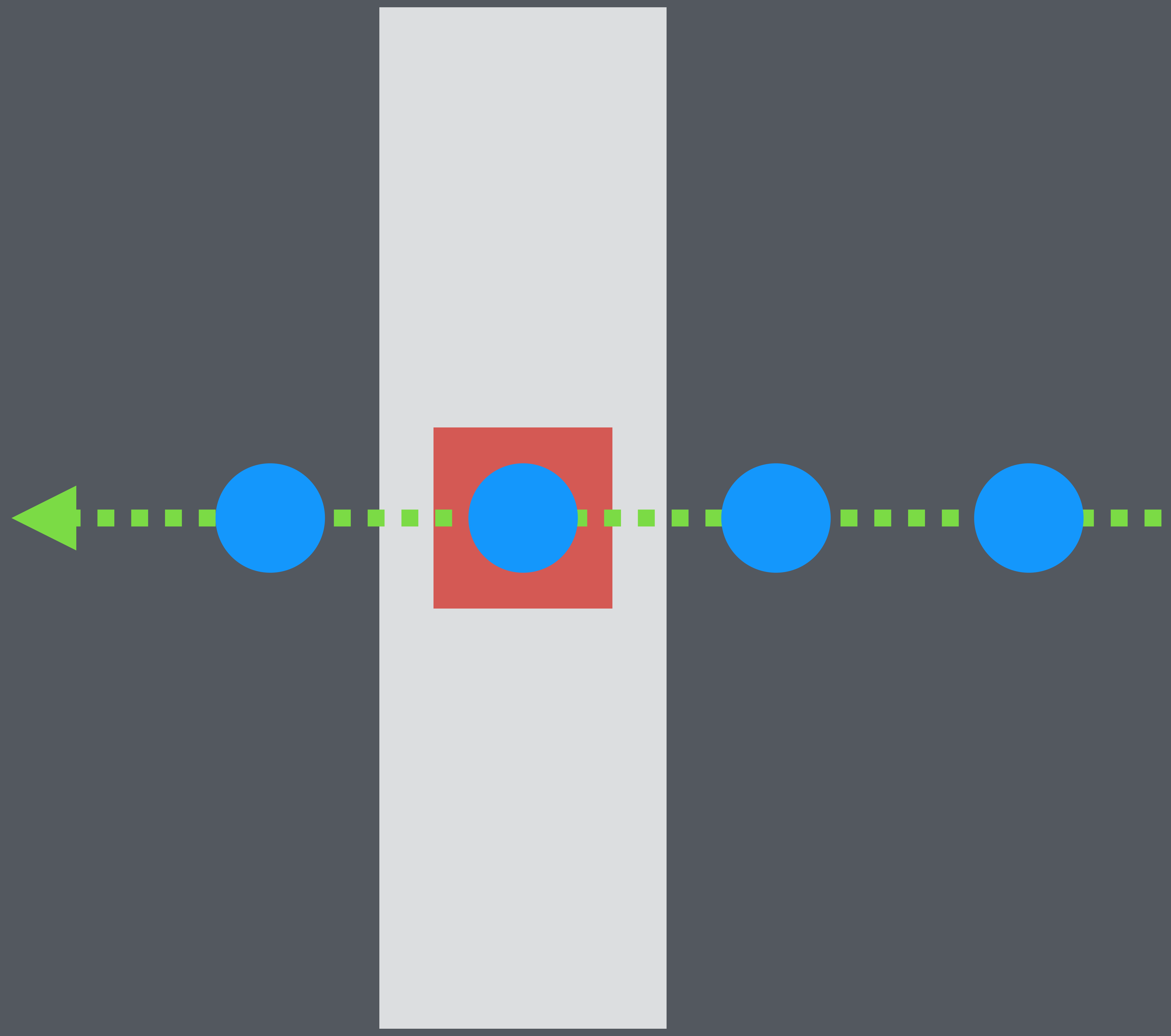
Combined movement.

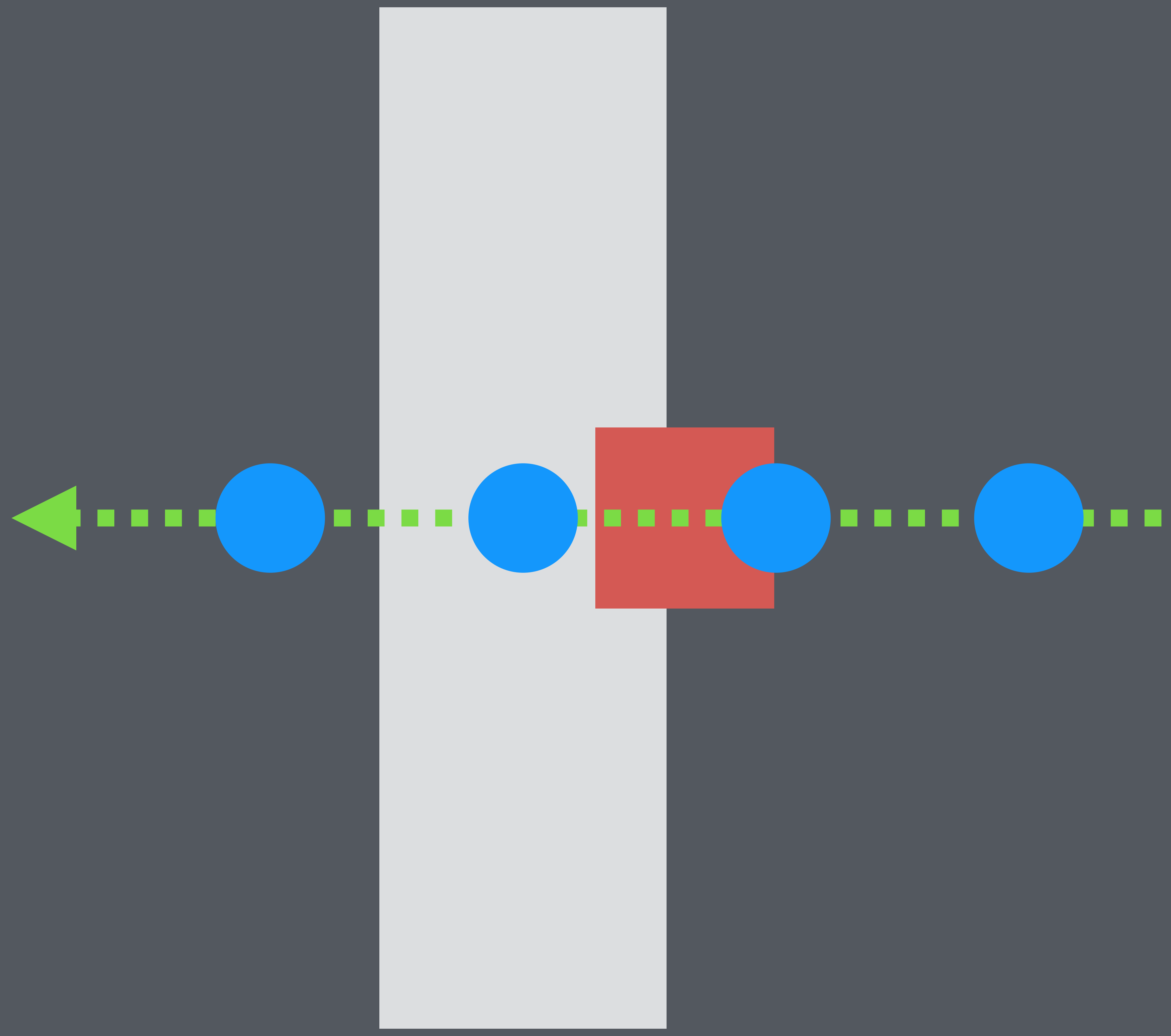
```
velocity_x = lerp(velocity_x, 0.0f, elapsed * friction_x);
velocity_y = lerp(velocity_y, 0.0f, elapsed * friction_y);

velocity_x += acceleration_x * elapsed;
velocity_y += acceleration_y * elapsed;

x += velocity_x * elapsed;
y += velocity_y * elapsed;
```

Collision response.



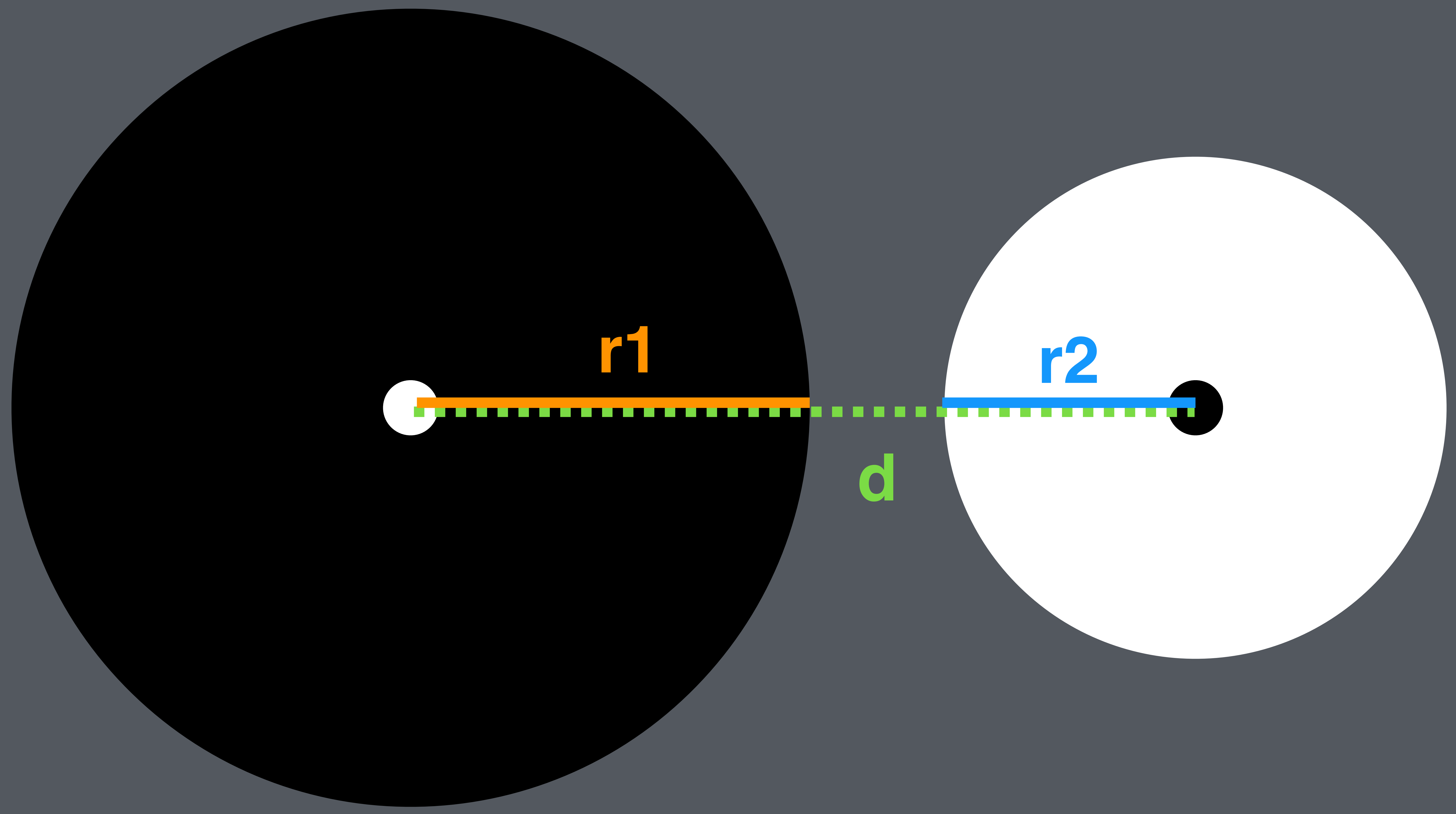




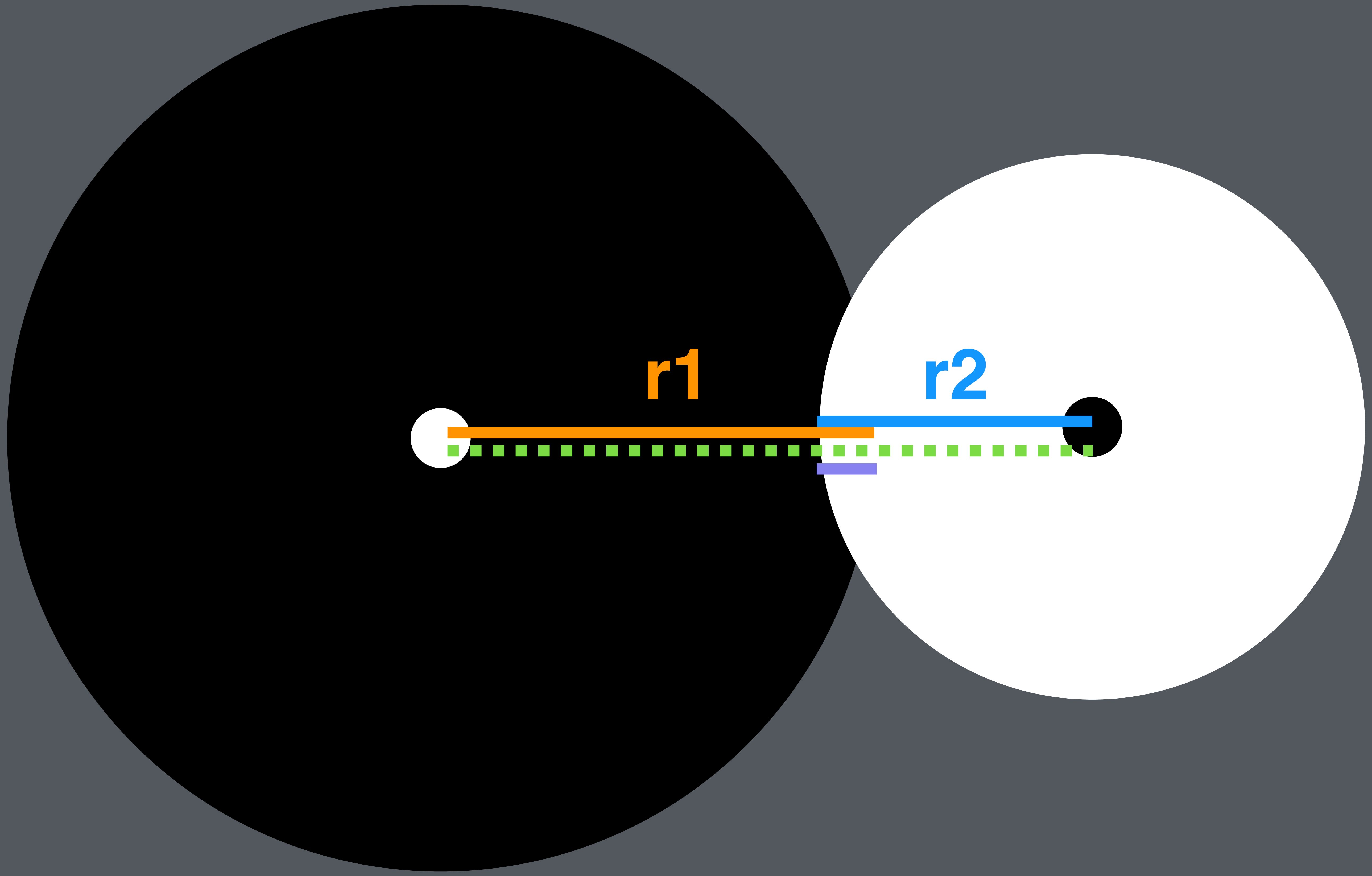


Calculating collision penetration.

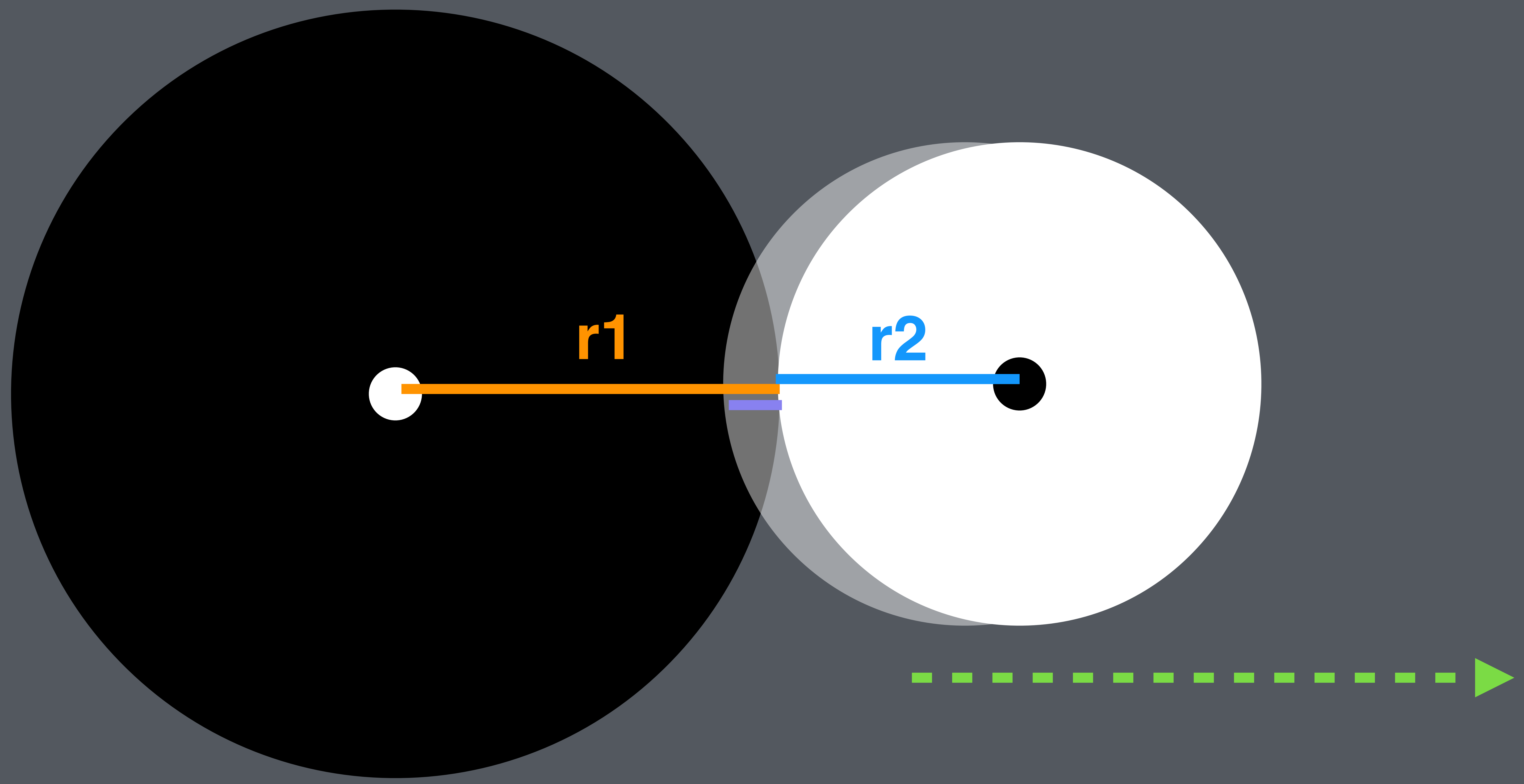
Circle - circle collision penetration.



If the **distance between two circles** is **less than or equal to the sum of their radii**, the circles are **colliding**!



```
penetration = fabs(distance - radius1 - radius2)
```



```
adjust = penetration * direction_vector
```

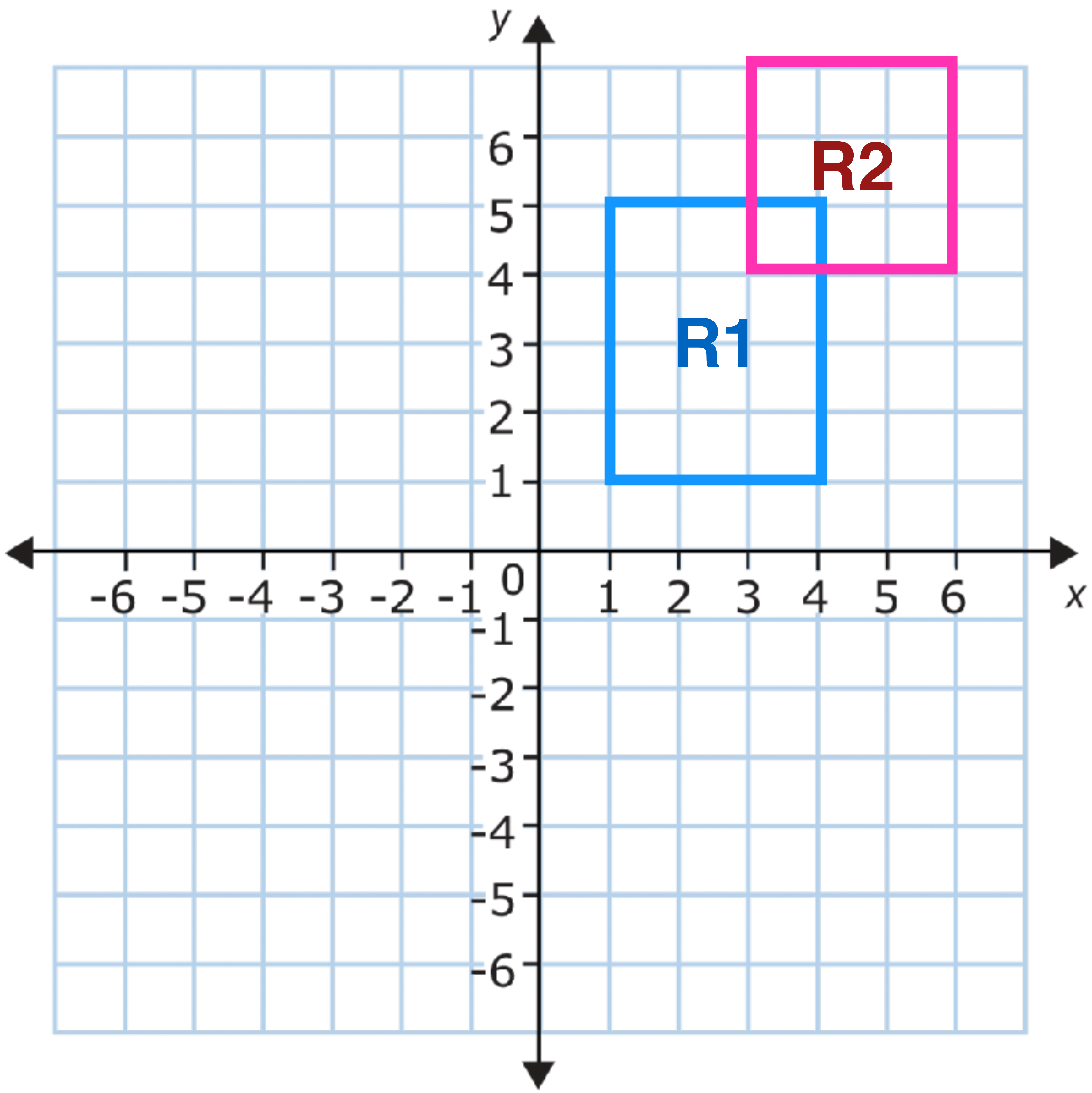
Box-box collision detection.

a) is R1's bottom higher than R2's top?
b) is R1's top lower than R2's bottom?
c) is R1's left larger than R2's right?
d) is R1's right smaller than R2's left

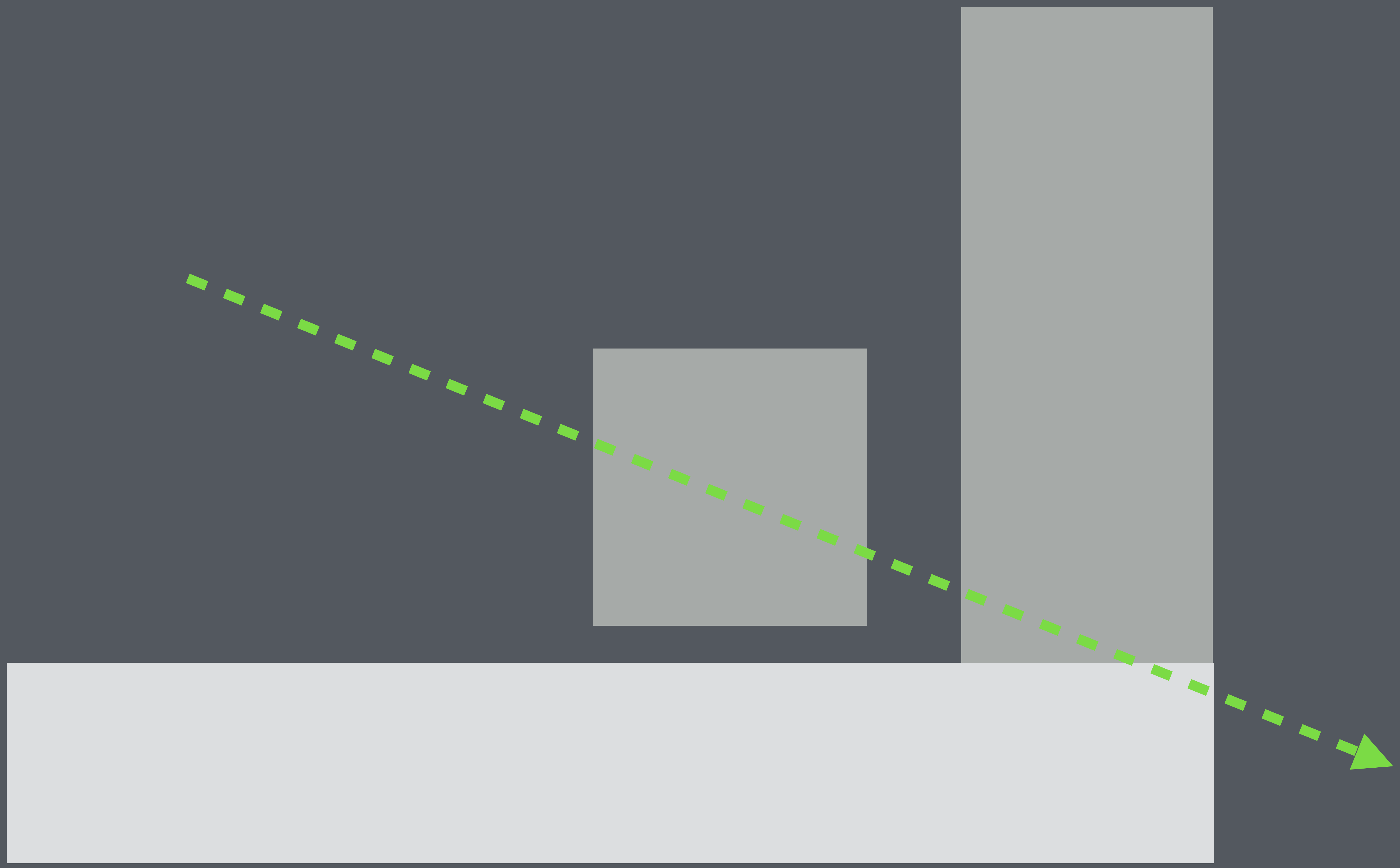
If ANY of the above are true, then the two rectangles are NOT intersecting!

OR

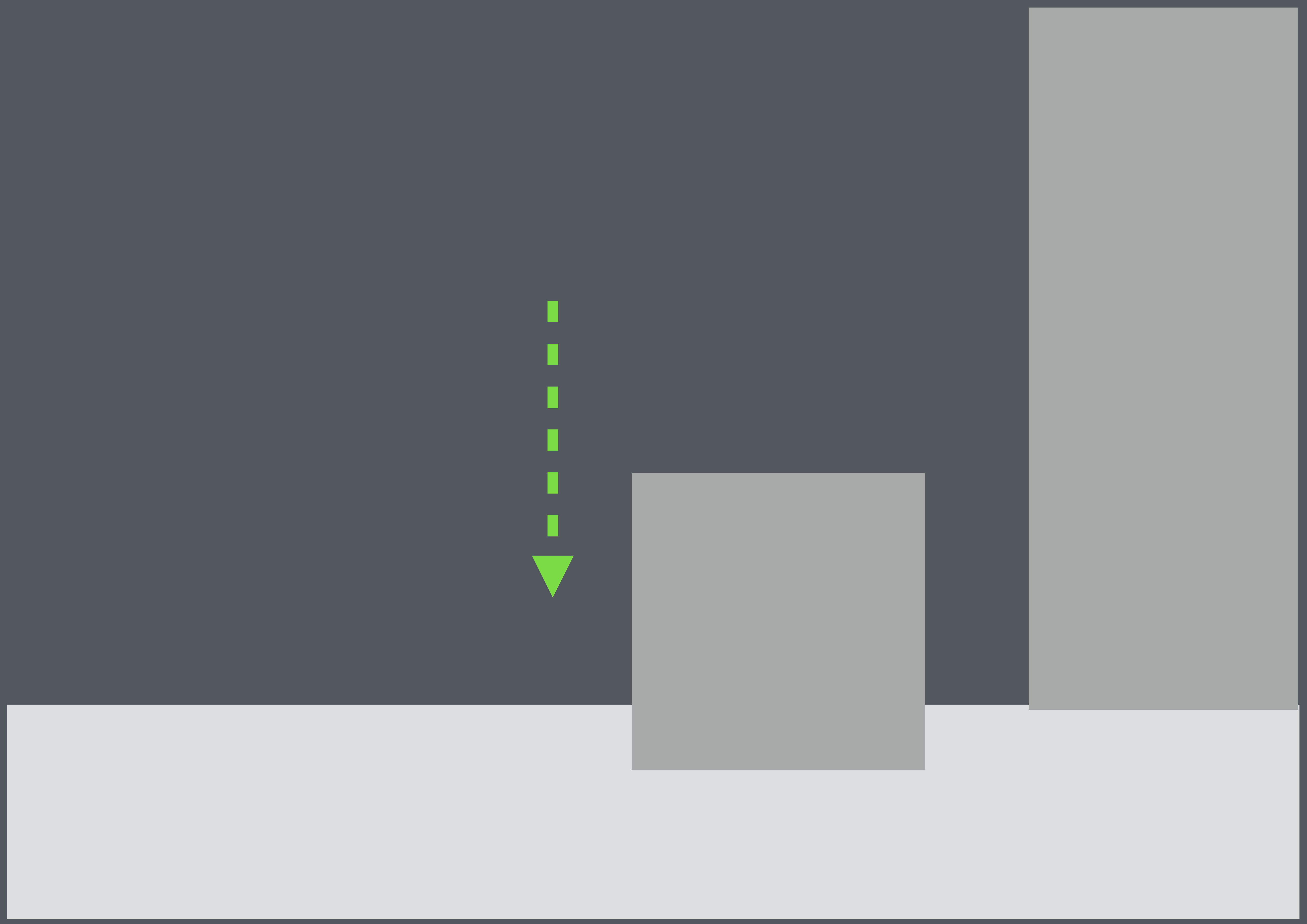
The rectangles are intersecting if NONE of the above are true.



Separate movement and collision on each axis!

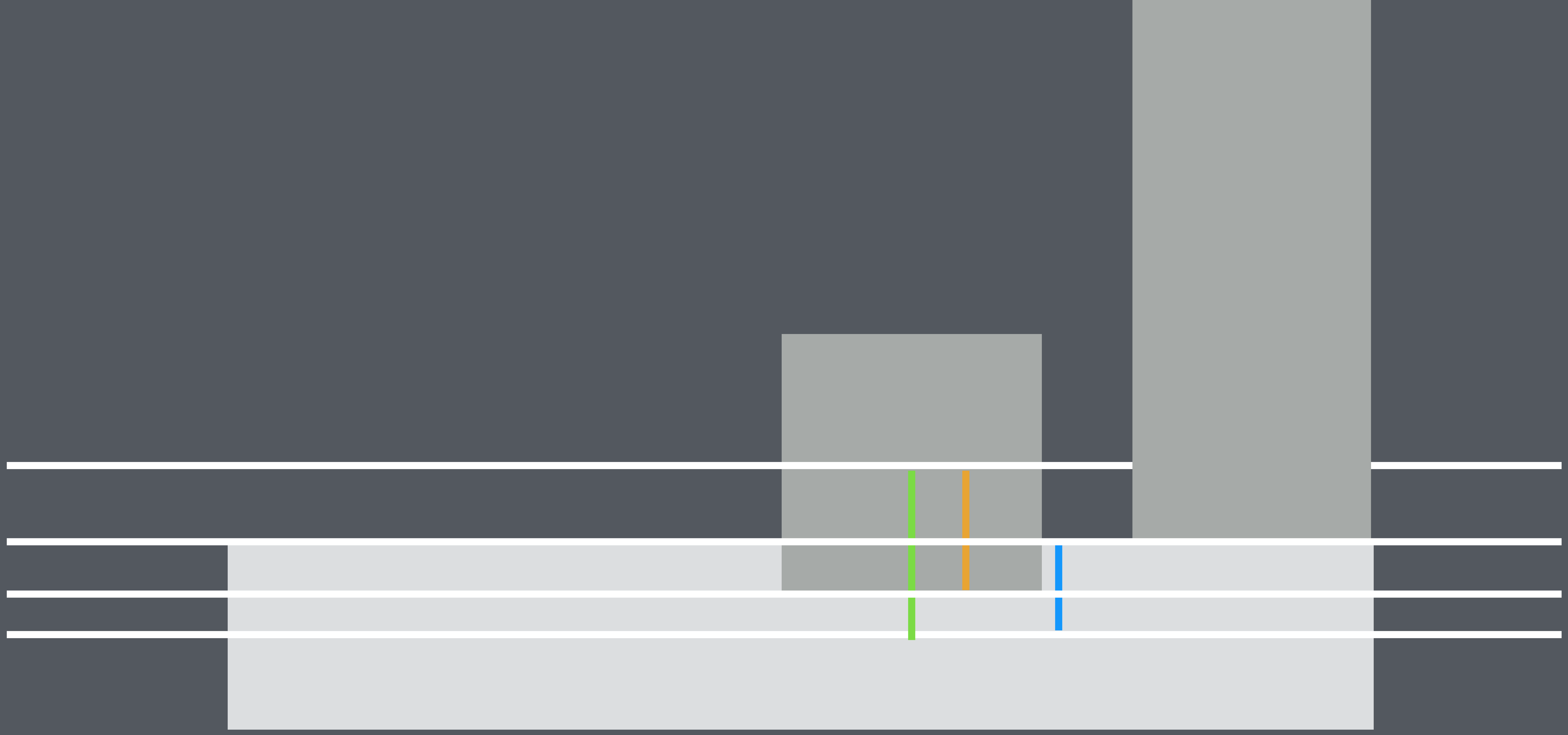


First only apply velocity to position on Y-axis!



Check full box/box collision against all entities.

If collided check Y-penetration.

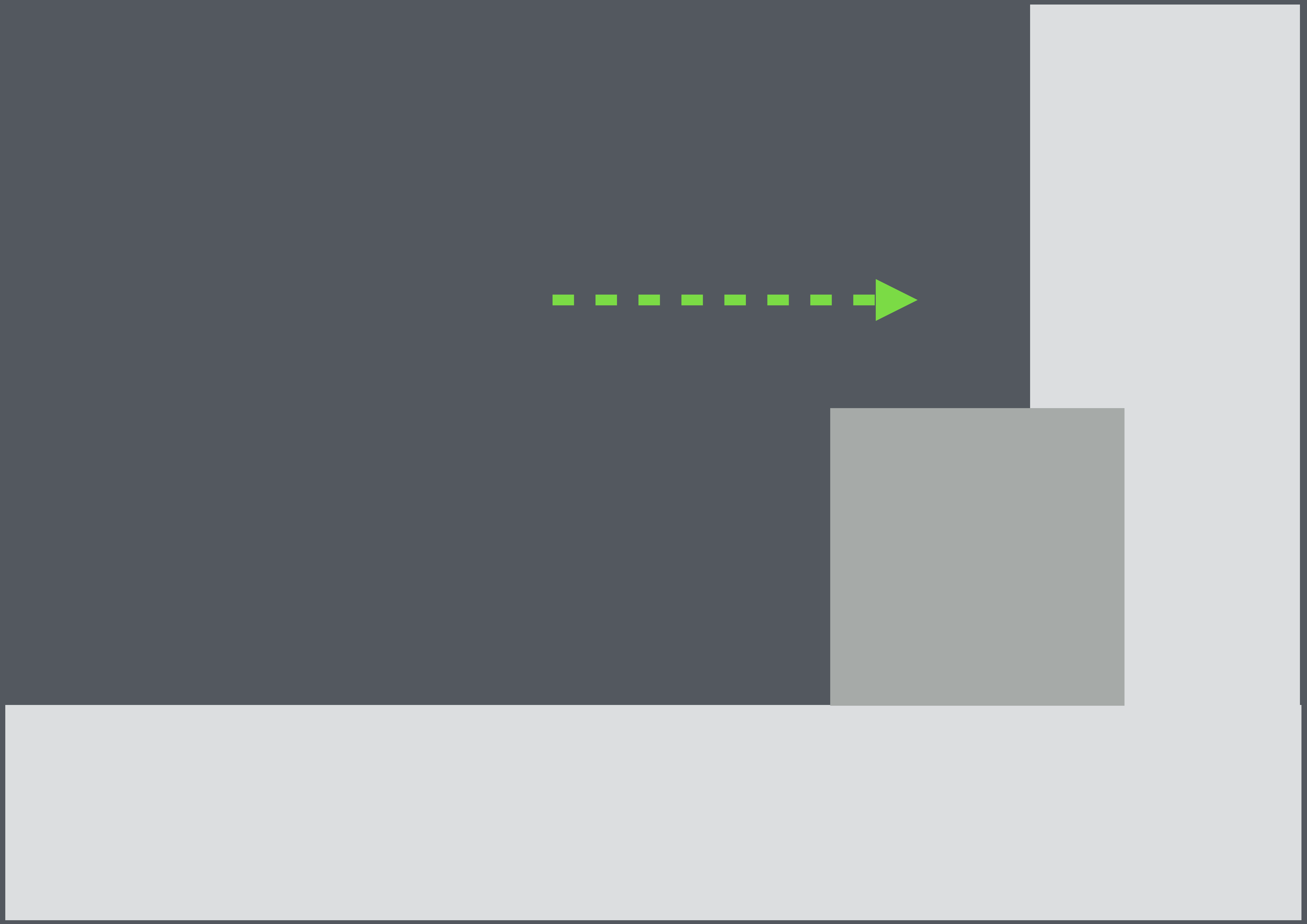


```
penetration = fabs(y_distance - height1/2 -
height2/2)
```

Move on Y-axis by the amount of penetration + tiny amount.

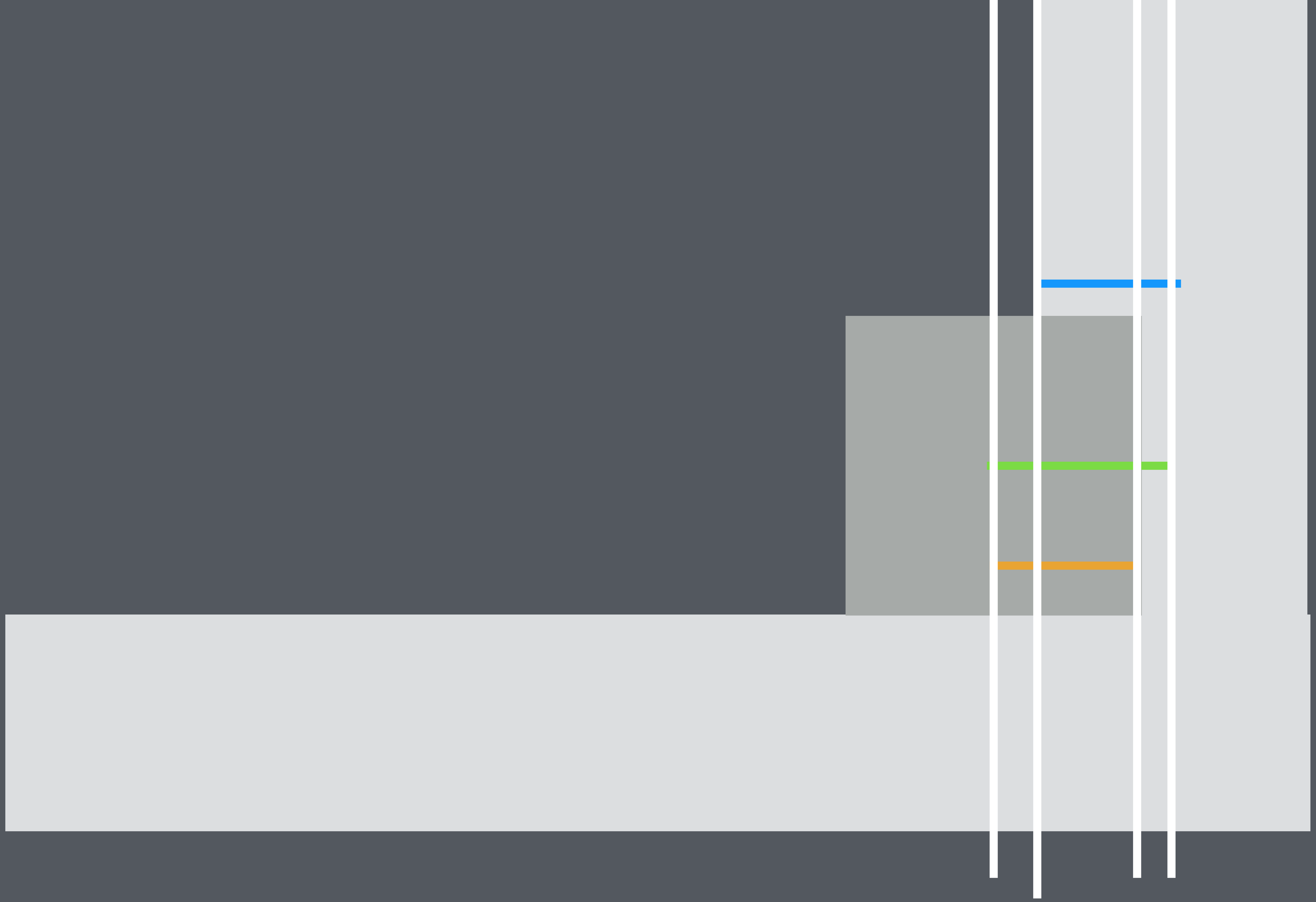
(Move up if above the other entity, otherwise move down!)

Now only apply velocity to position on X-axis!



Check full box/box collision against all entities.

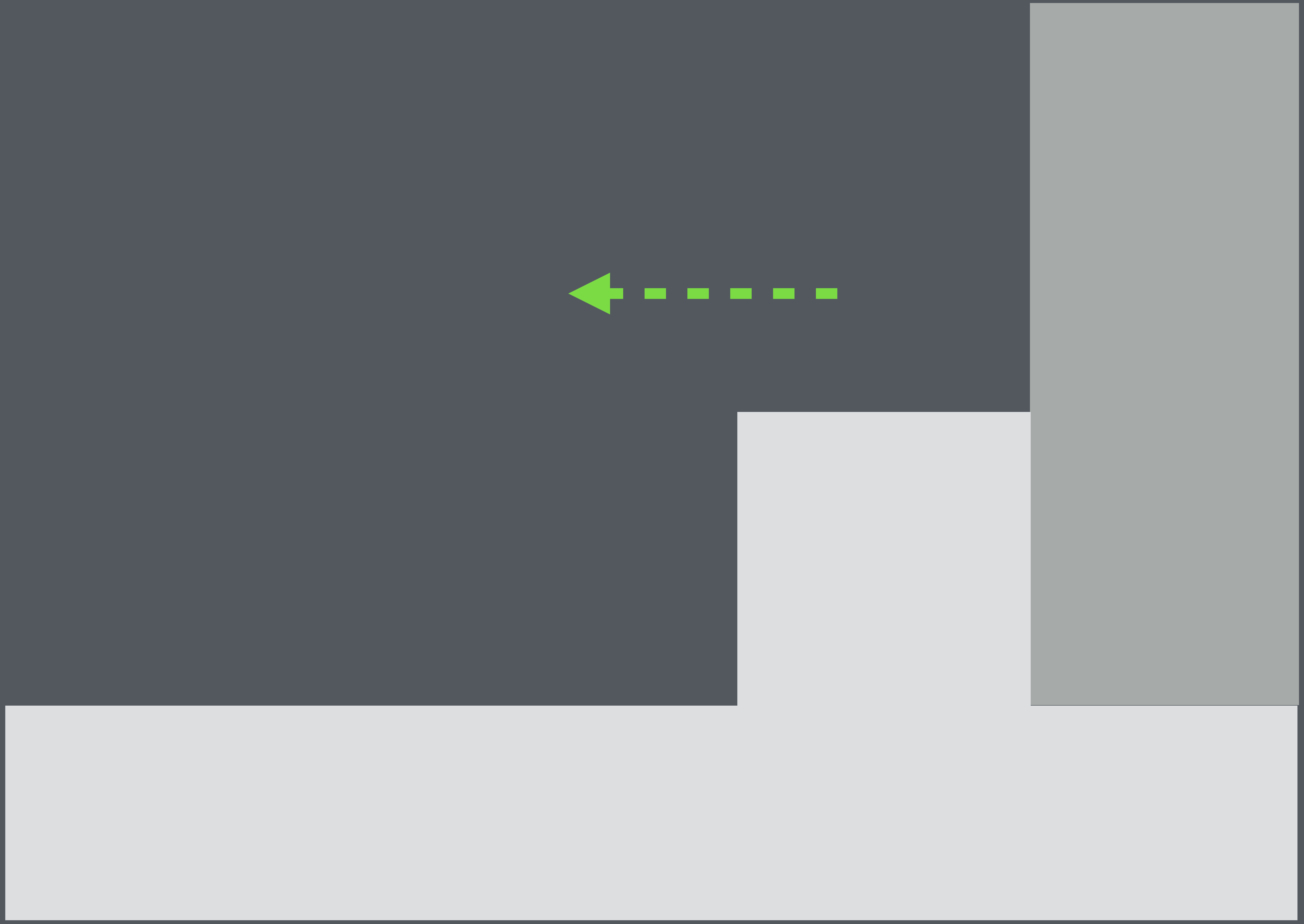
If collided check X-penetration.



```
penetration = fabs(x_distance - width1/2 -
width2/2)
```

Move on X-axis by the amount of penetration + tiny amount.

(Move left if to the left of the other entity, otherwise move right!)



```
velocity_x = lerp(velocity_x, 0.0f, elapsed * friction_x);
velocity_y = lerp(velocity_y, 0.0f, elapsed * friction_y);

velocity_x += acceleration_x * elapsed;
velocity_y += acceleration_y * elapsed;

y += velocity_y * elapsed;
collisionY();

x += velocity_x * elapsed;
collisionX();
```

Gravity.

Gravity.

A constant acceleration.

```
velocity_x += gravity_x * elapsed;
velocity_y += gravity_y * elapsed;
```