Game AI.
Part 1

https://www.youtube.com/watch?v=mgWhF1WkBJk

What is game AI?

Basic AI components.

- Sensing
- Pathfinding
- States

Sensing

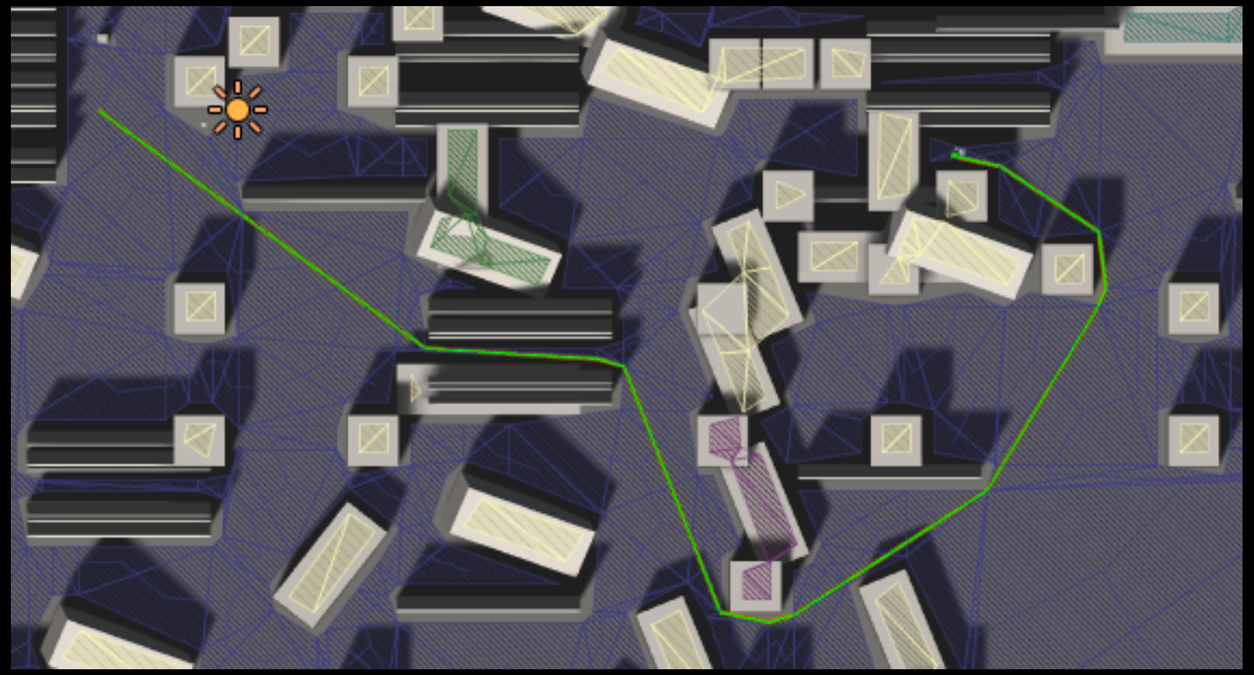
Sensing the player.

Sensing the world.

Pathfinding

Navigating the level.

Navigating the level.



States

Normal

Angry


IGN.COM


104/106

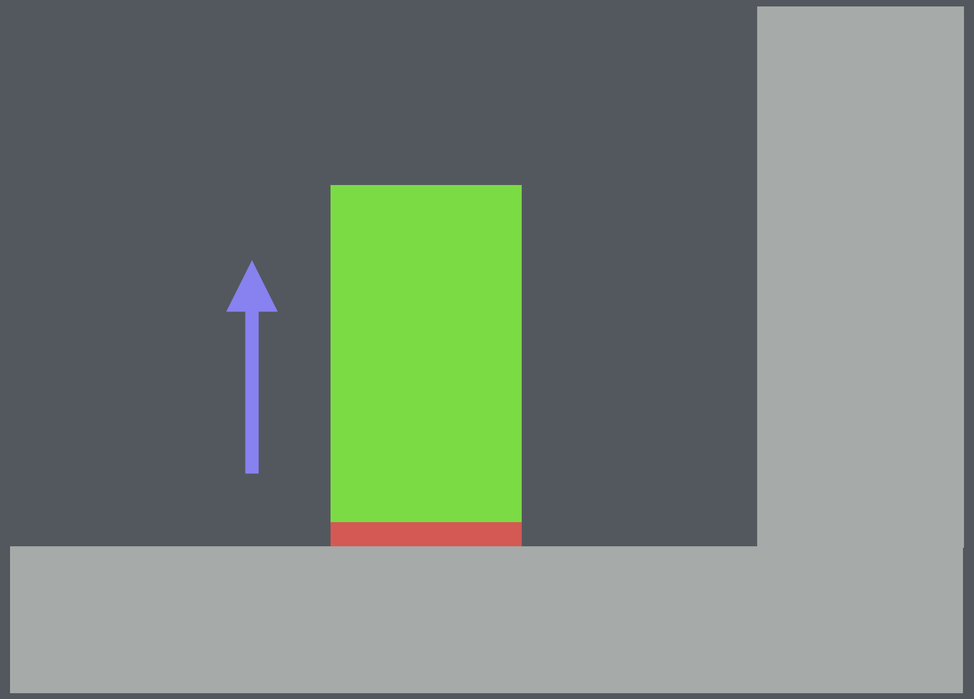

Z
Z

Sensing basics.

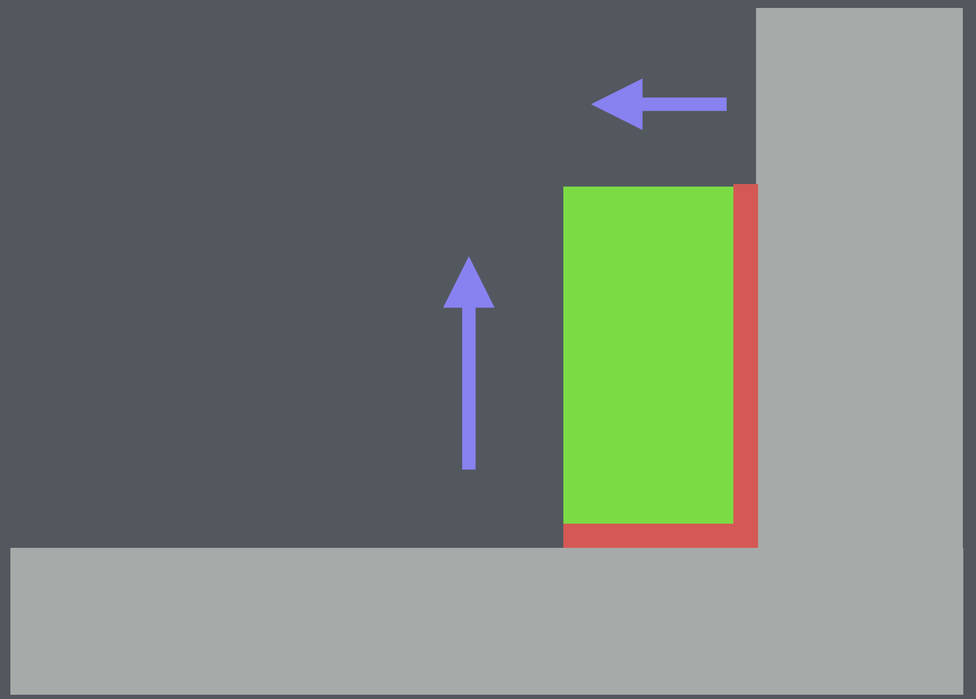
Collision sensing.

Collision flags

When we adjust out of collision, we the set collision flag for that direction to true.



`collidingBottom = true`



`collidingBottom = true`

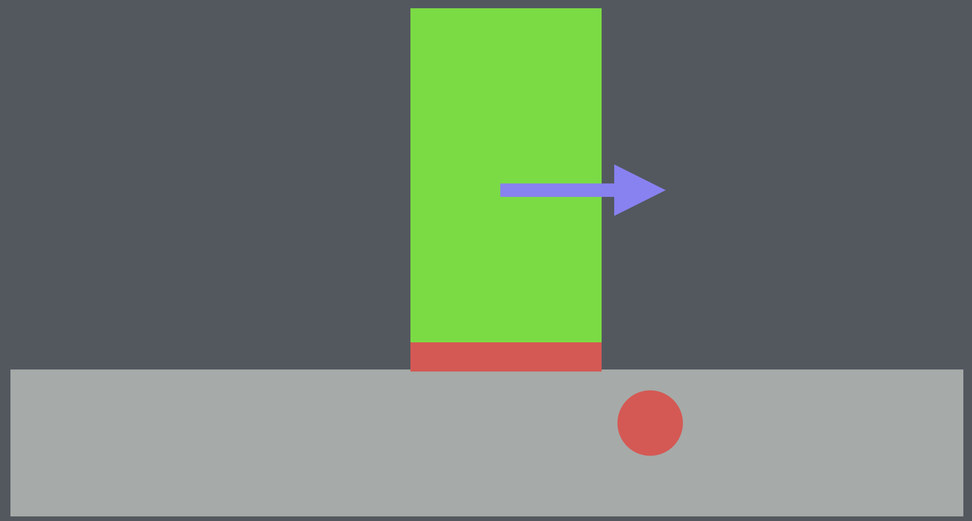
`collidingRight = true`

Collision flag sensing example: turning around at a wall.

When an X-axis collision flag is set, inverse the X acceleration.

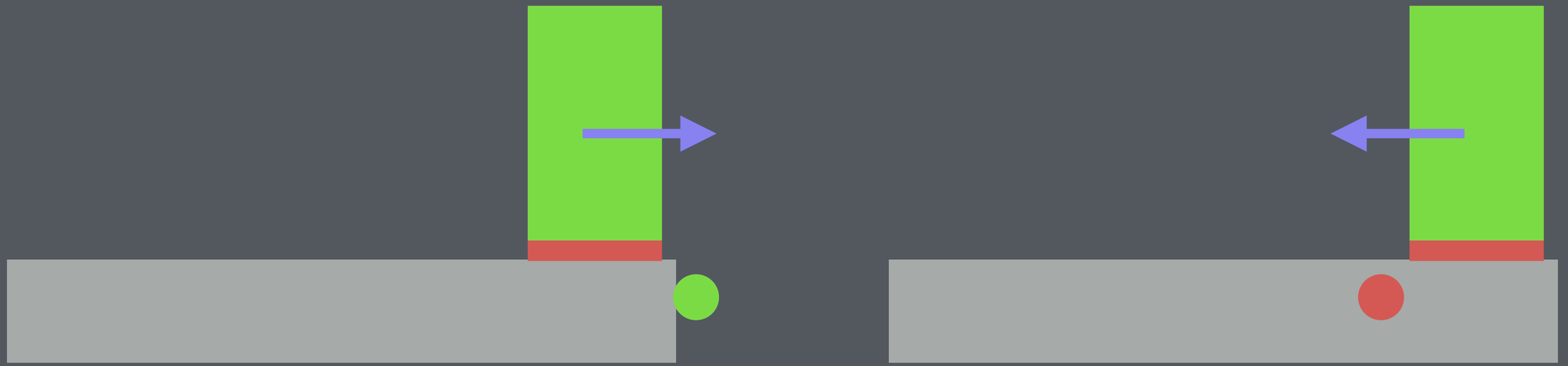
Problem: turning around at an edge.

Collision sensors.

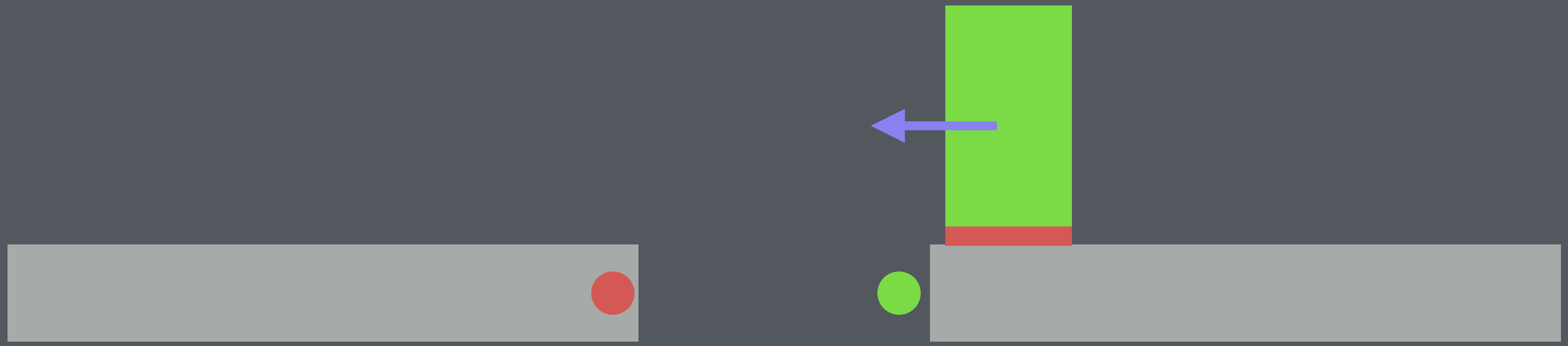


Check for collision using a detector point.

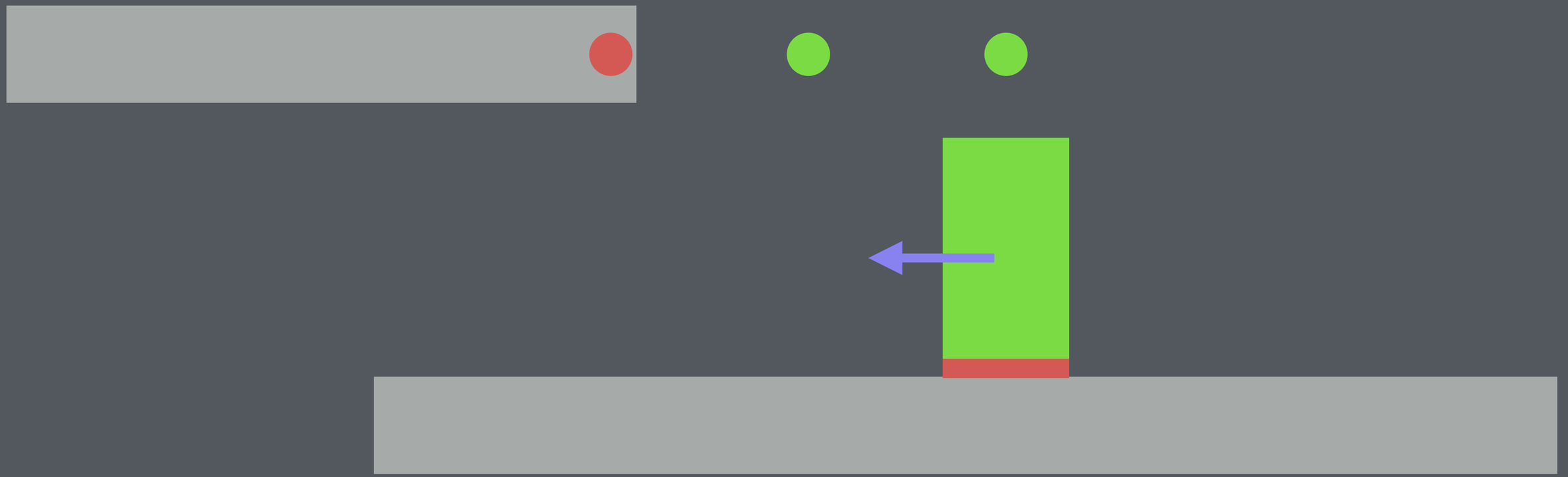
(use **worldToTileCoordinates** if using a tilemap)



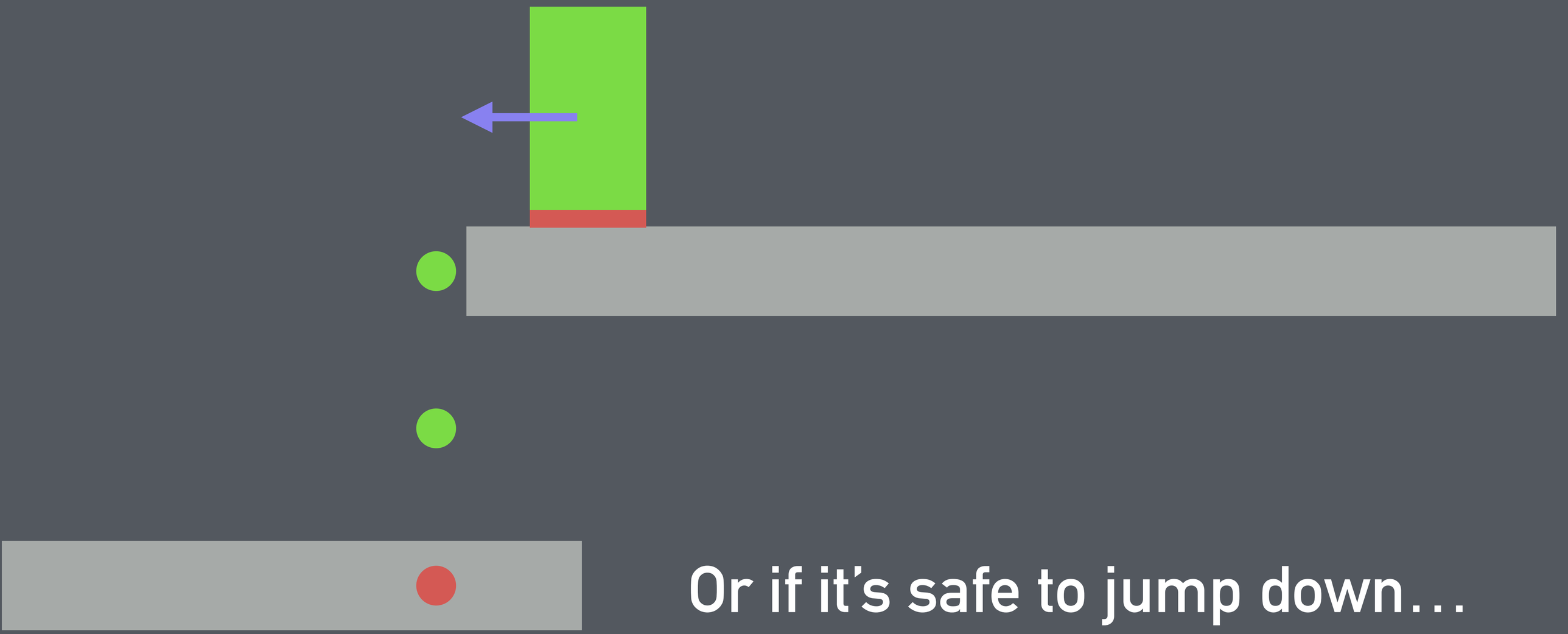
If detector point not colliding and our bottom collision flag is set (don’t want to set it off when jumping), we’ve reached an edge!



Can use another sensor point to see if we can jump across a gap.

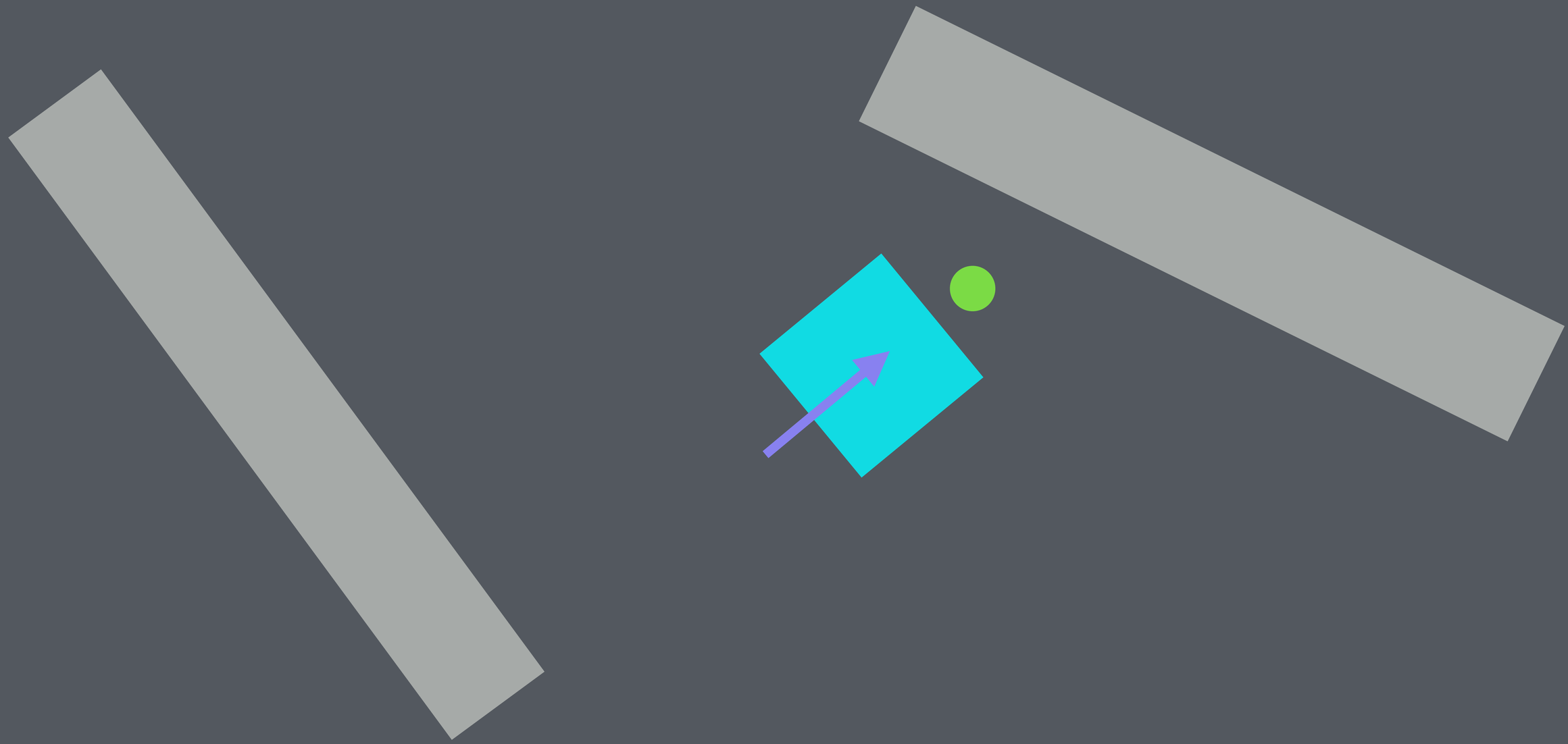


Can use 2 or 3 sensor points to see if possible to jump up to a platform.

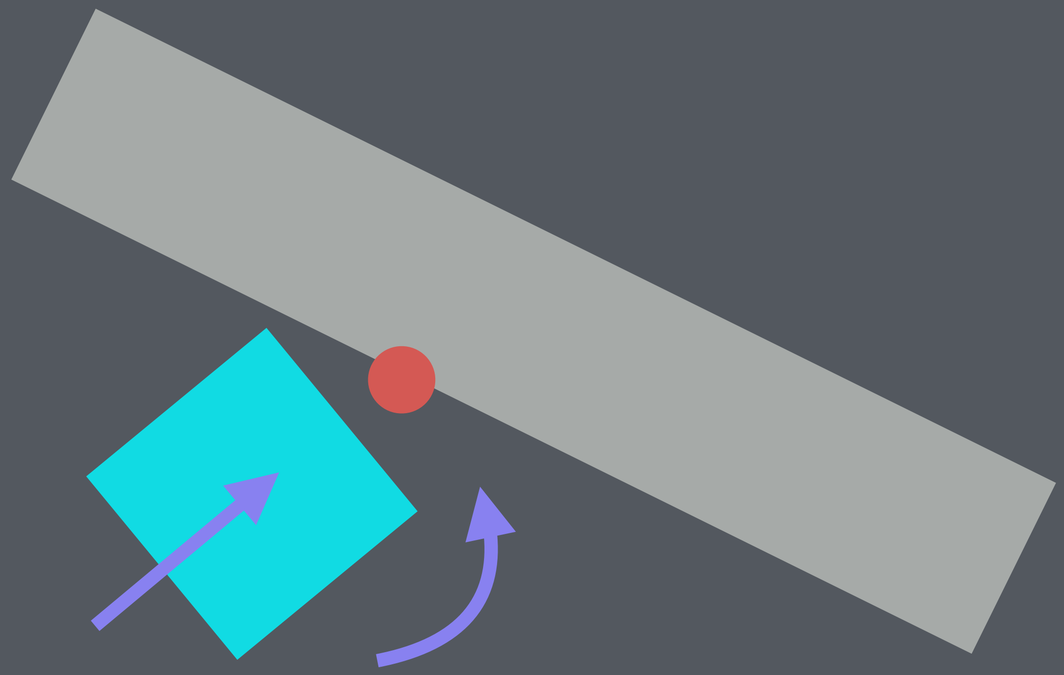


Or if it’s safe to jump down…

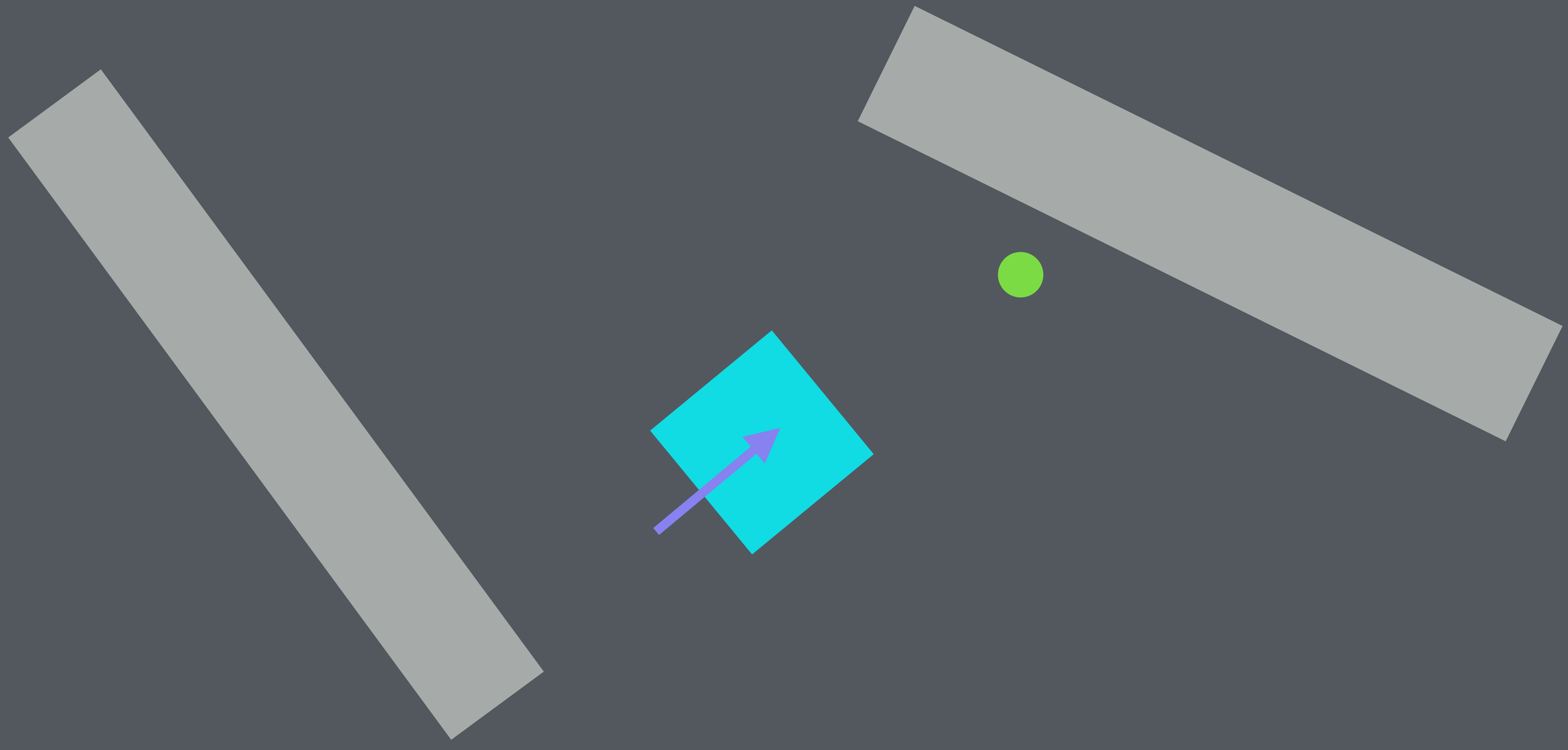
Collision sensors in non-axis aligned collisions.

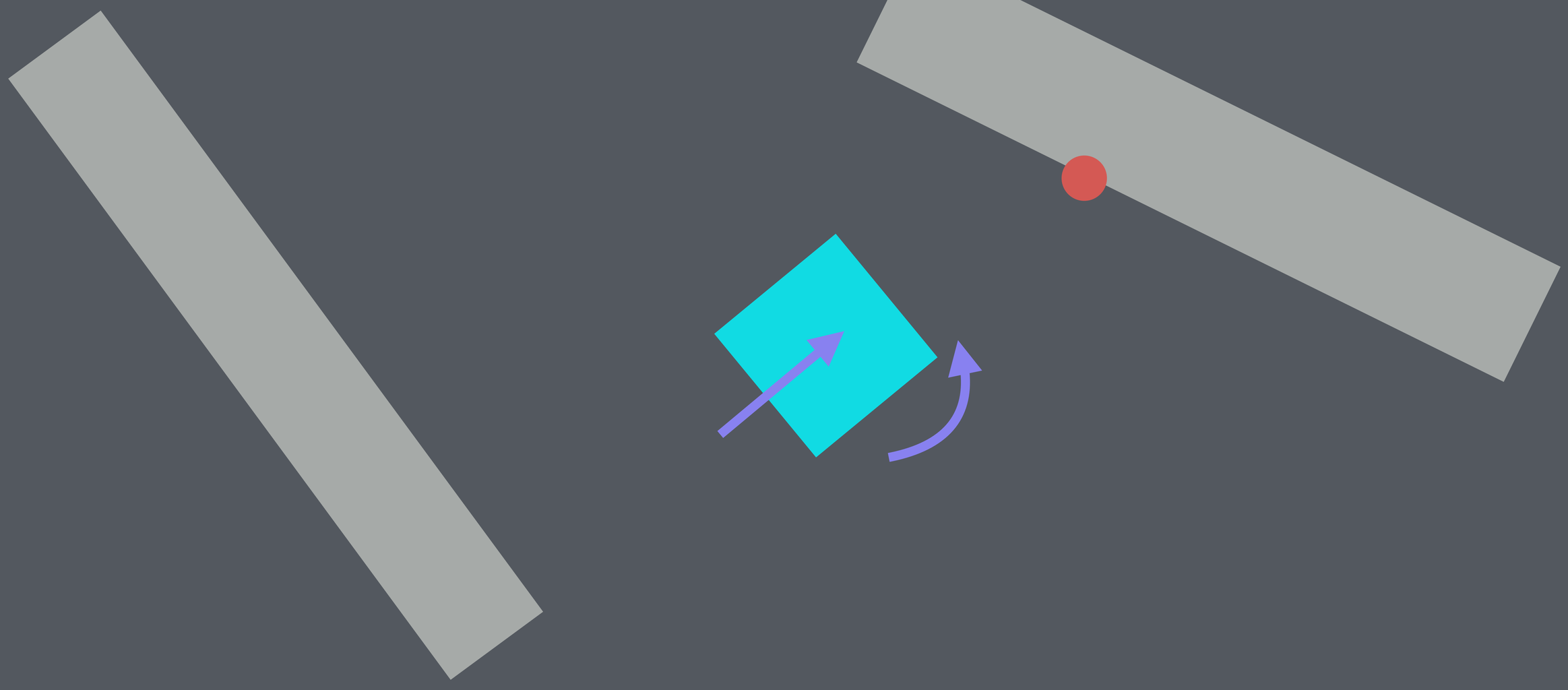


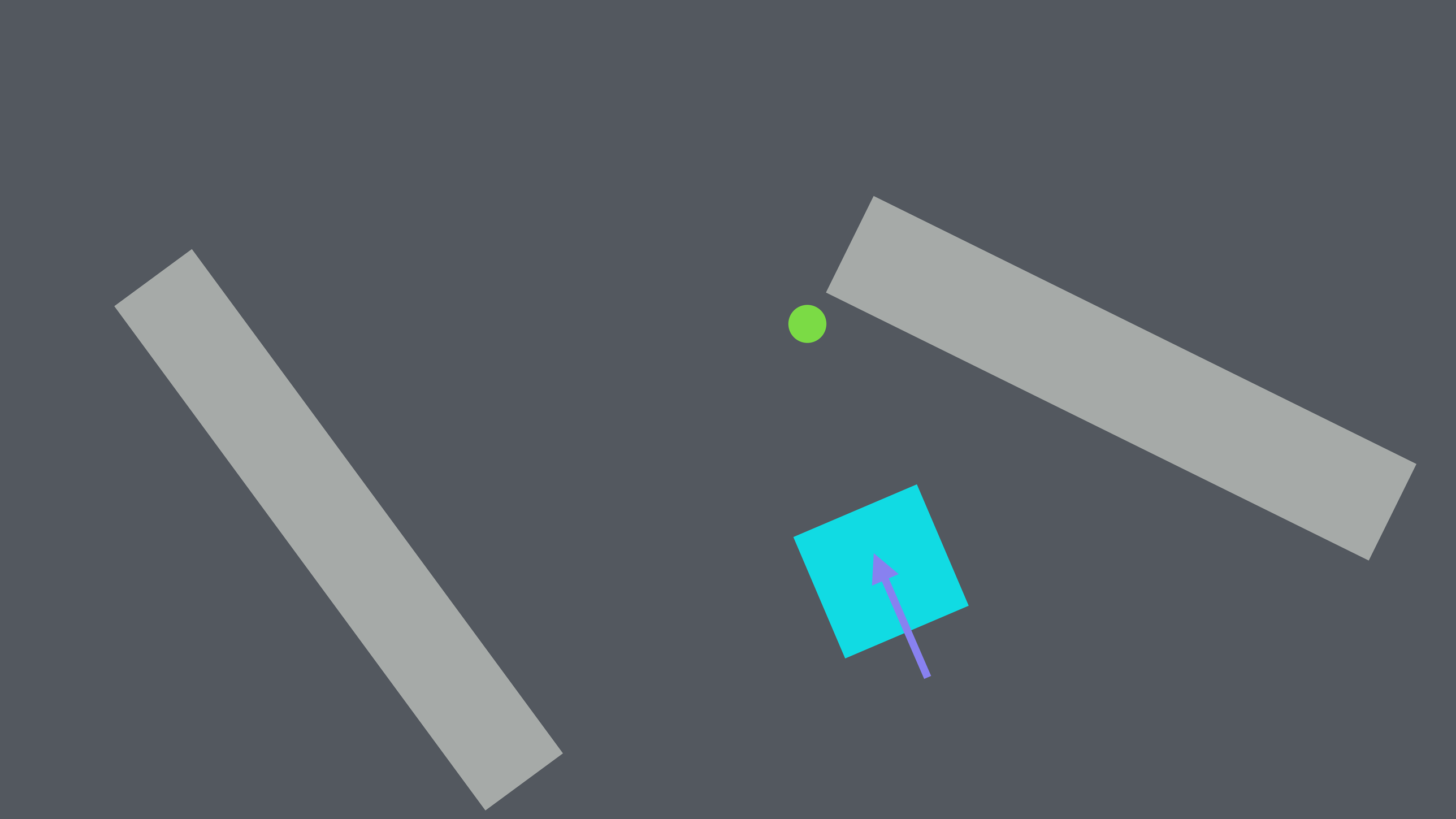
Point in rotated rectangle collision detection. Rotate if point is inside any entity.



Changing sensor distance.

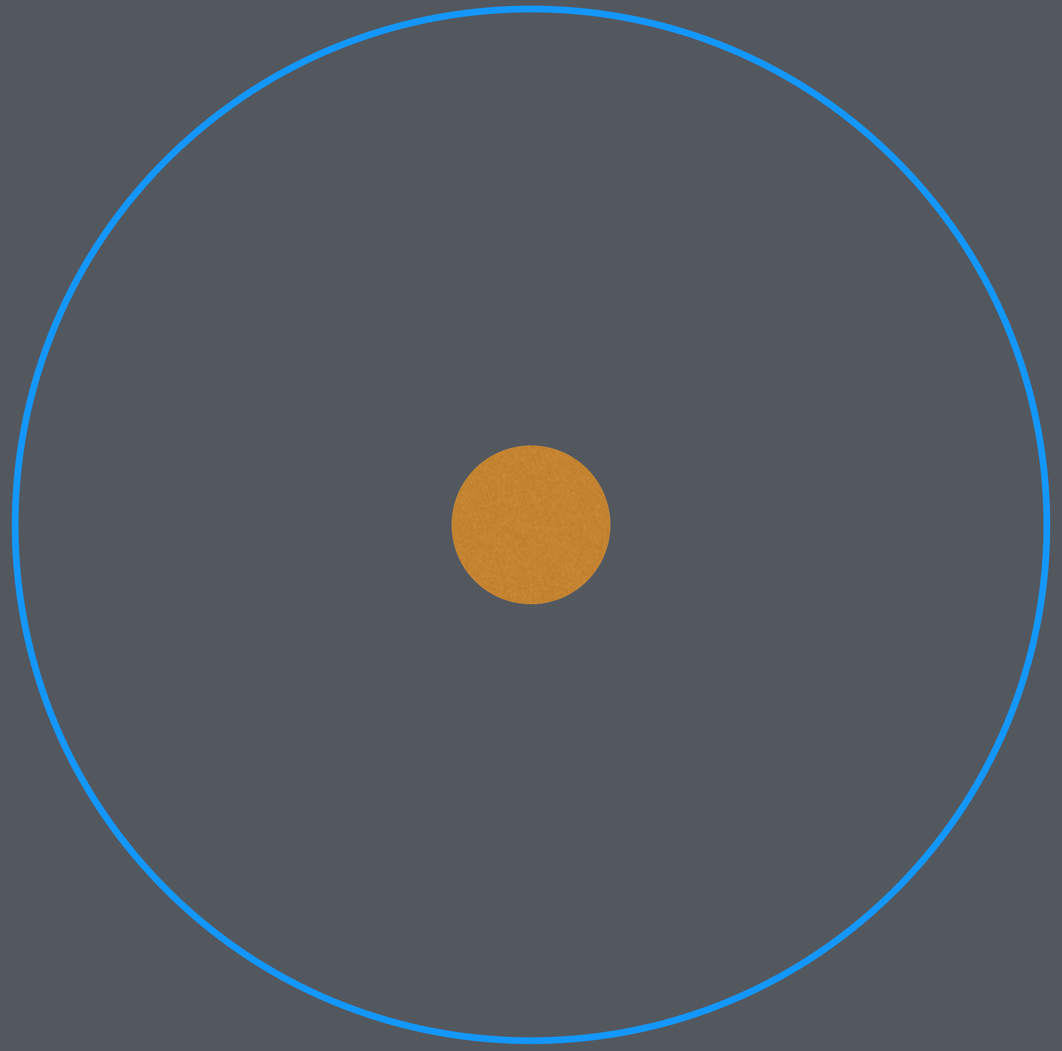


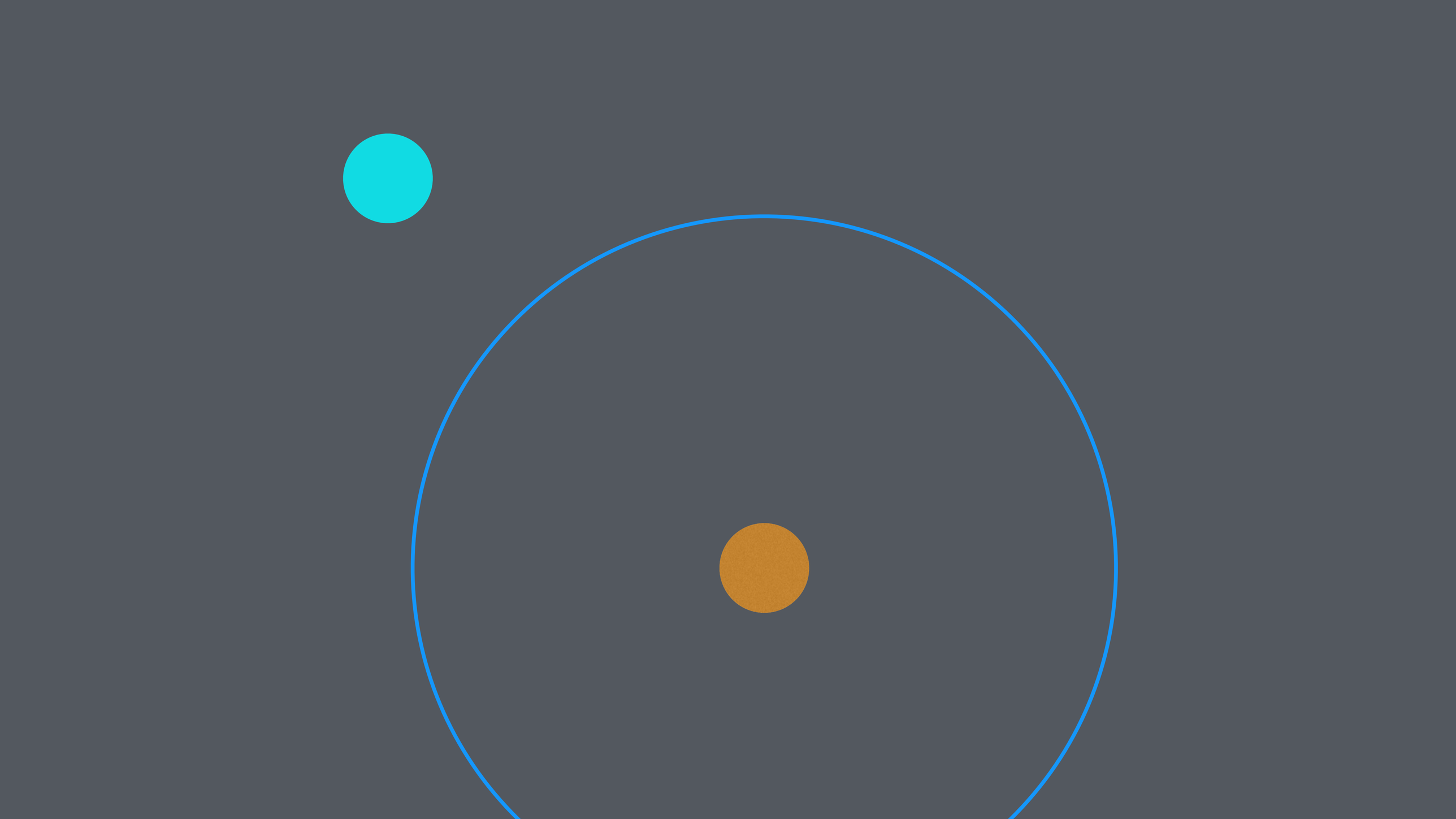


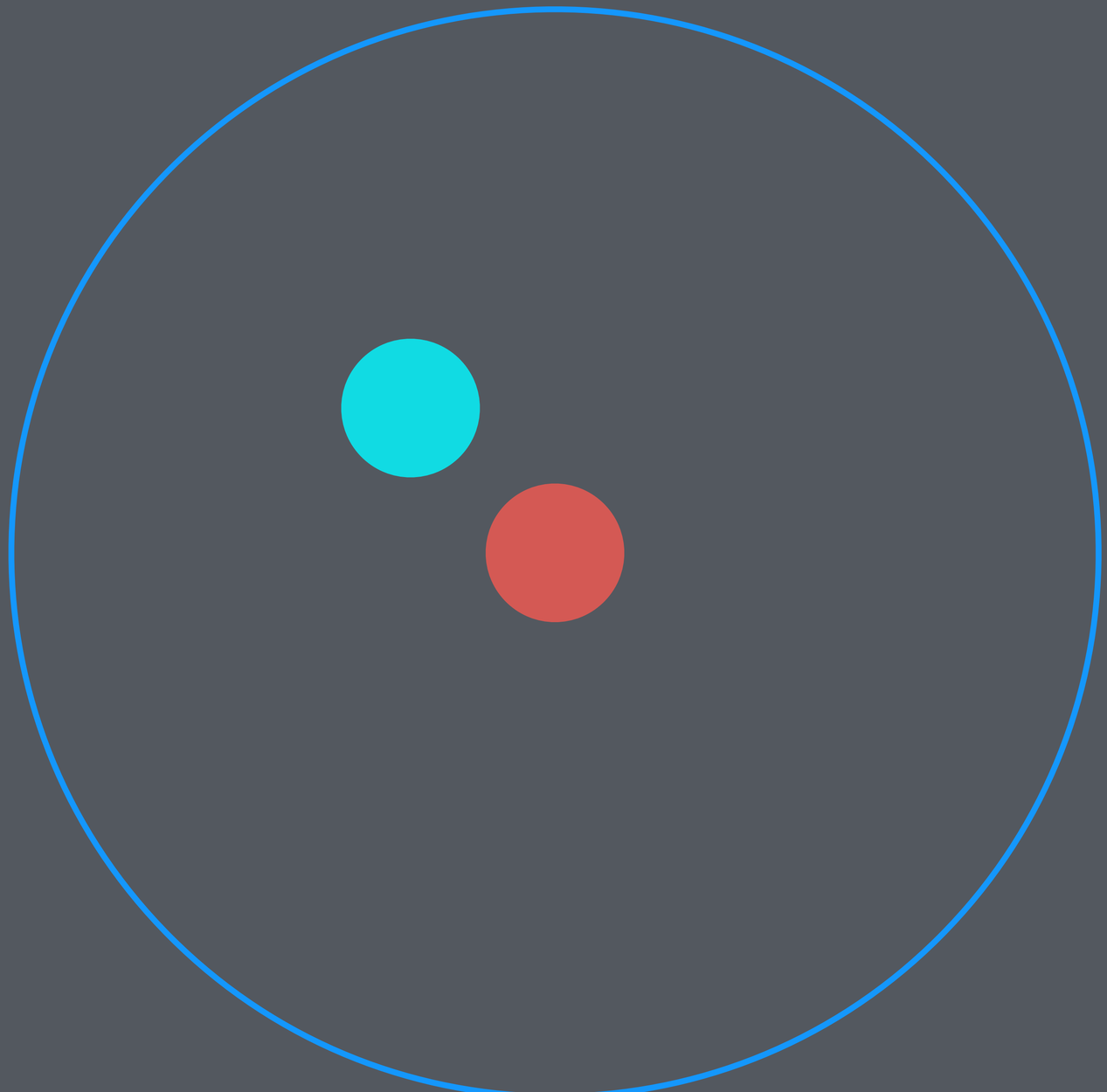


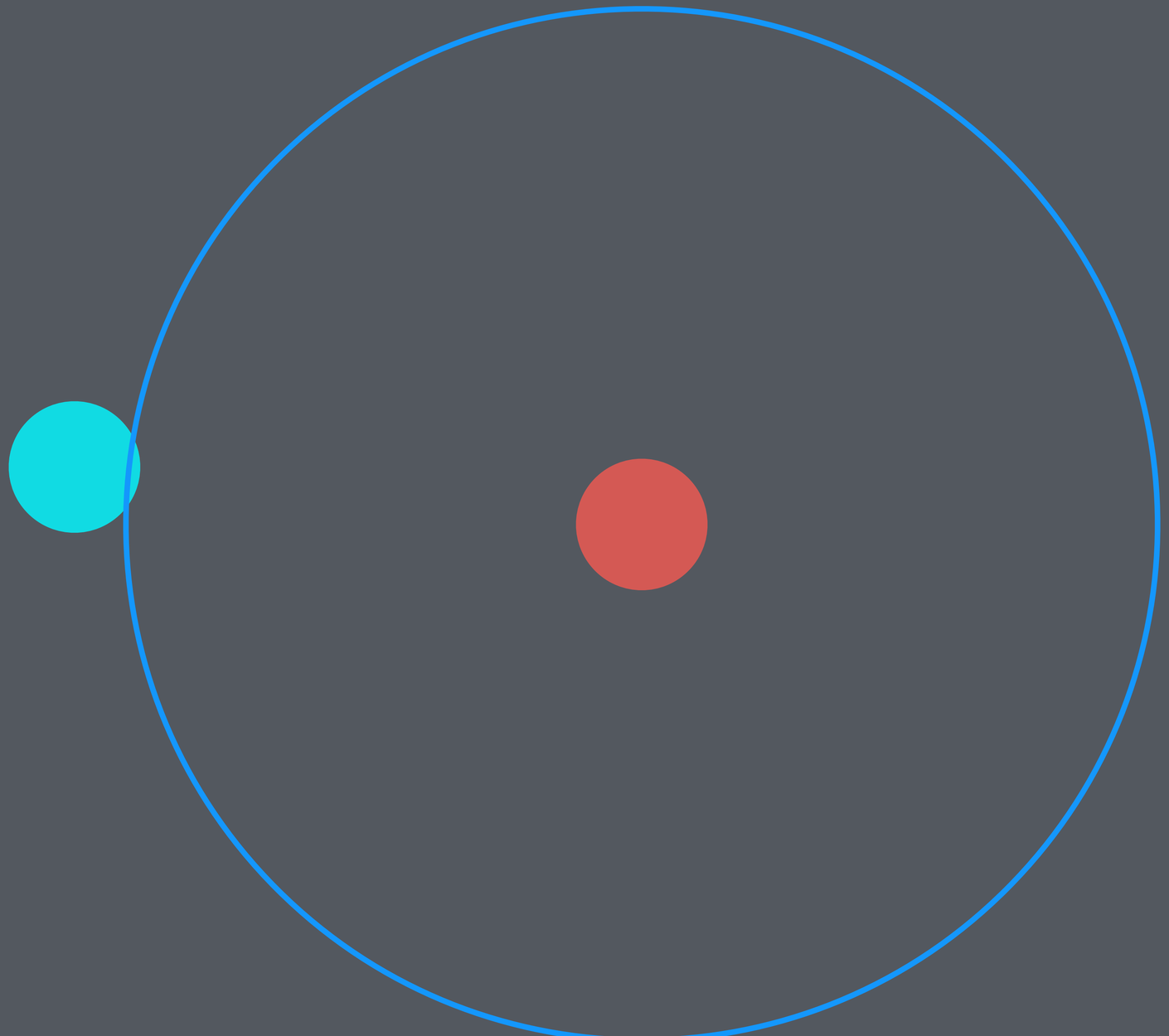
Simulating senses.

Sensor range.

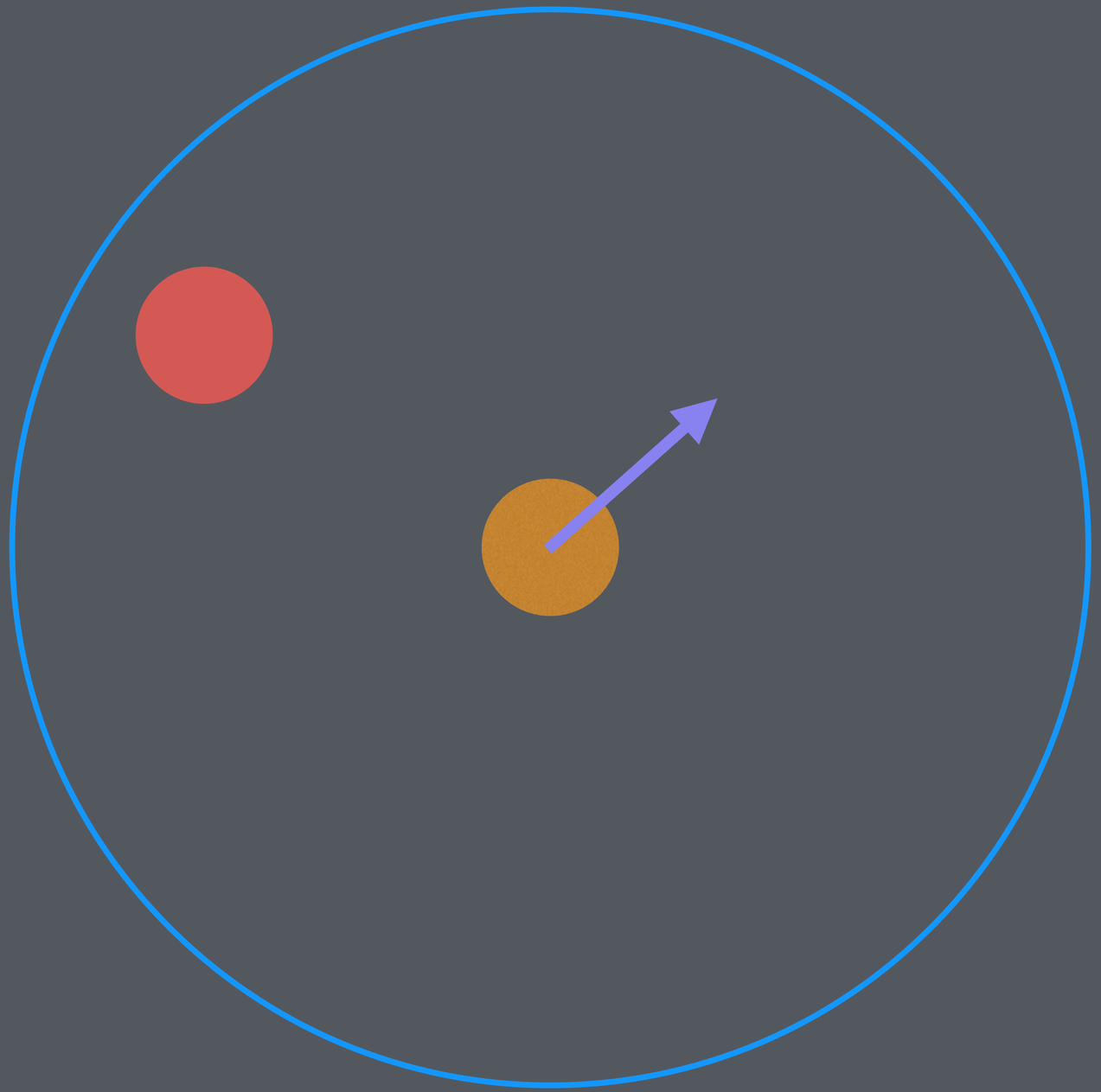


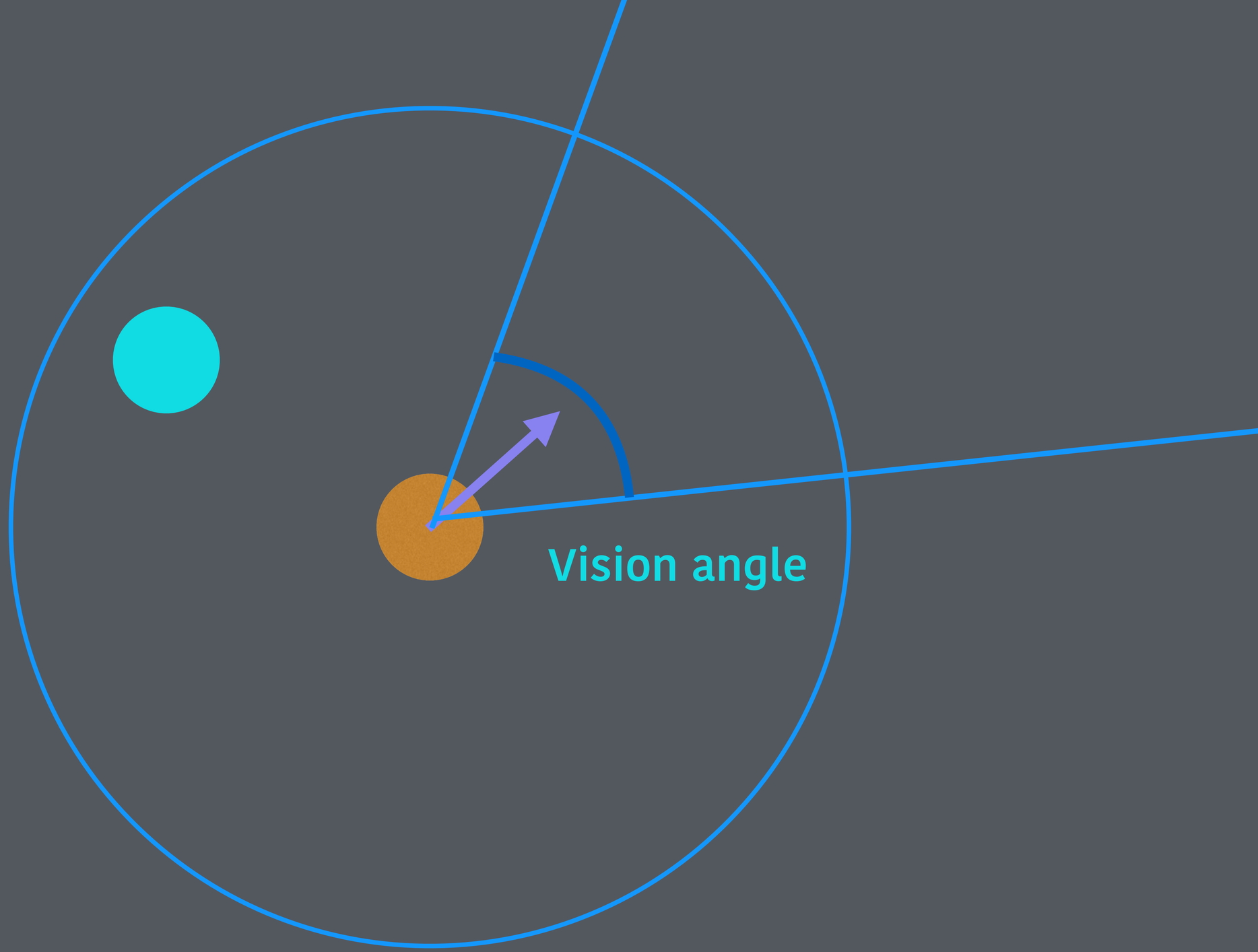


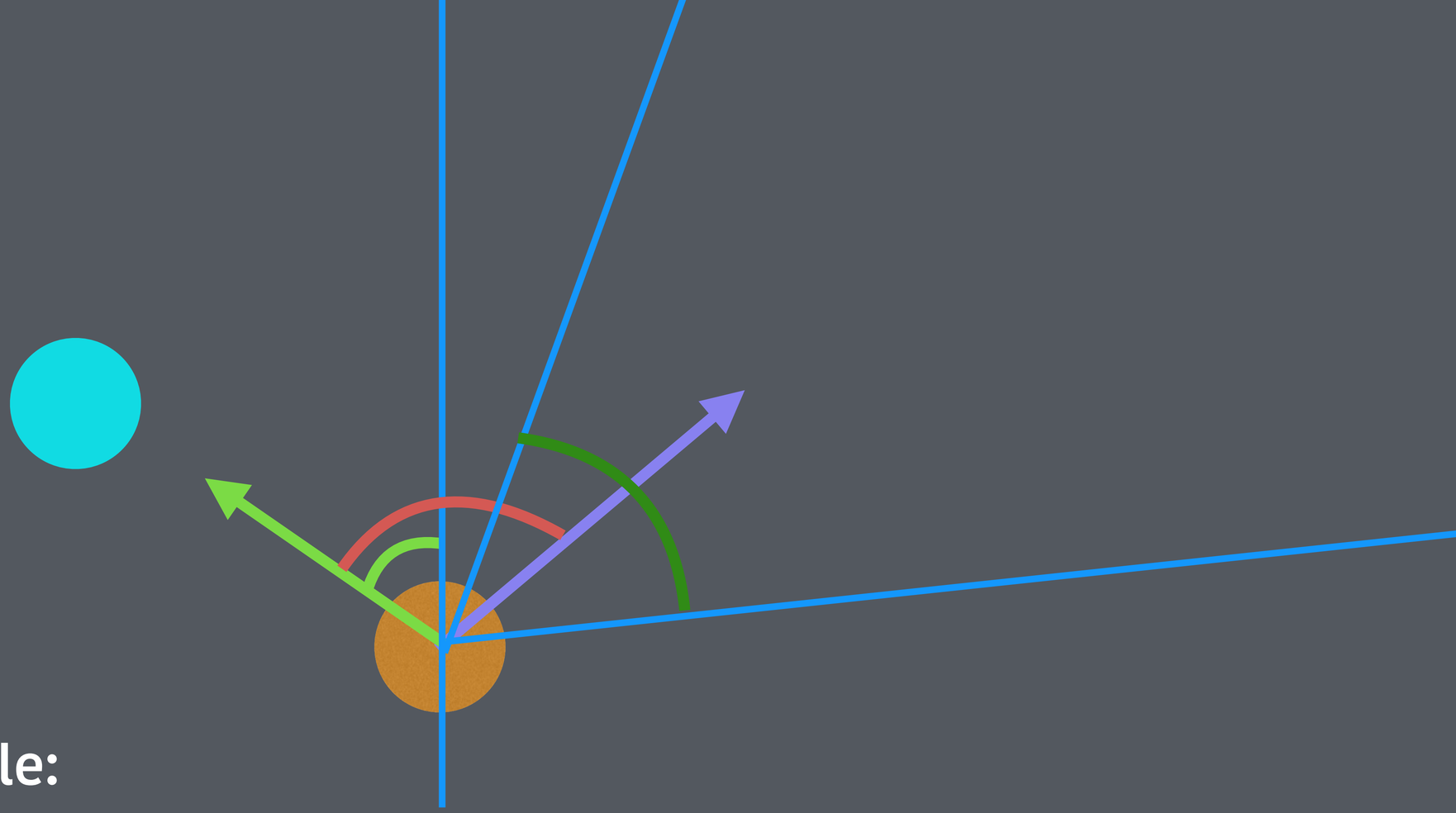




Vision cone.

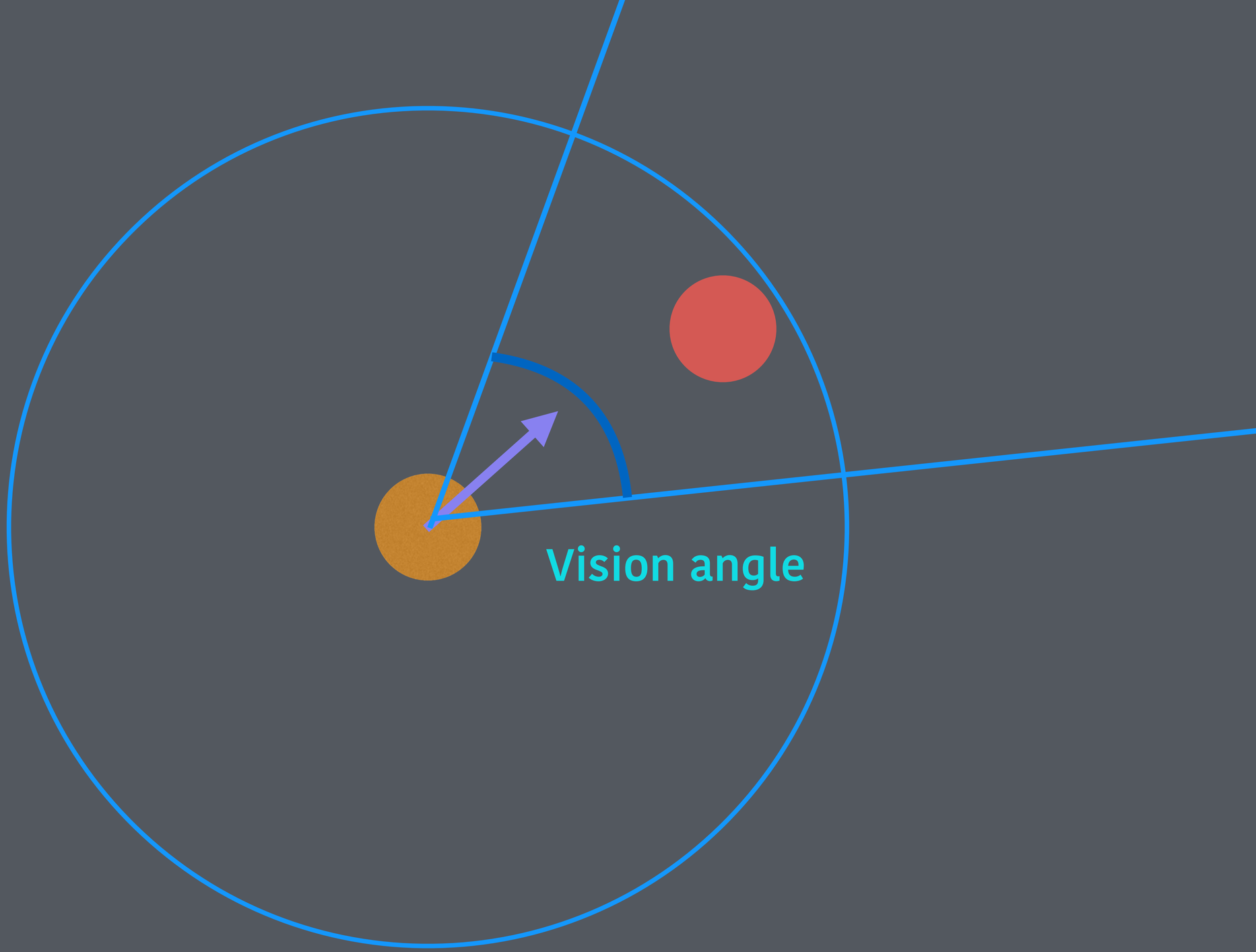


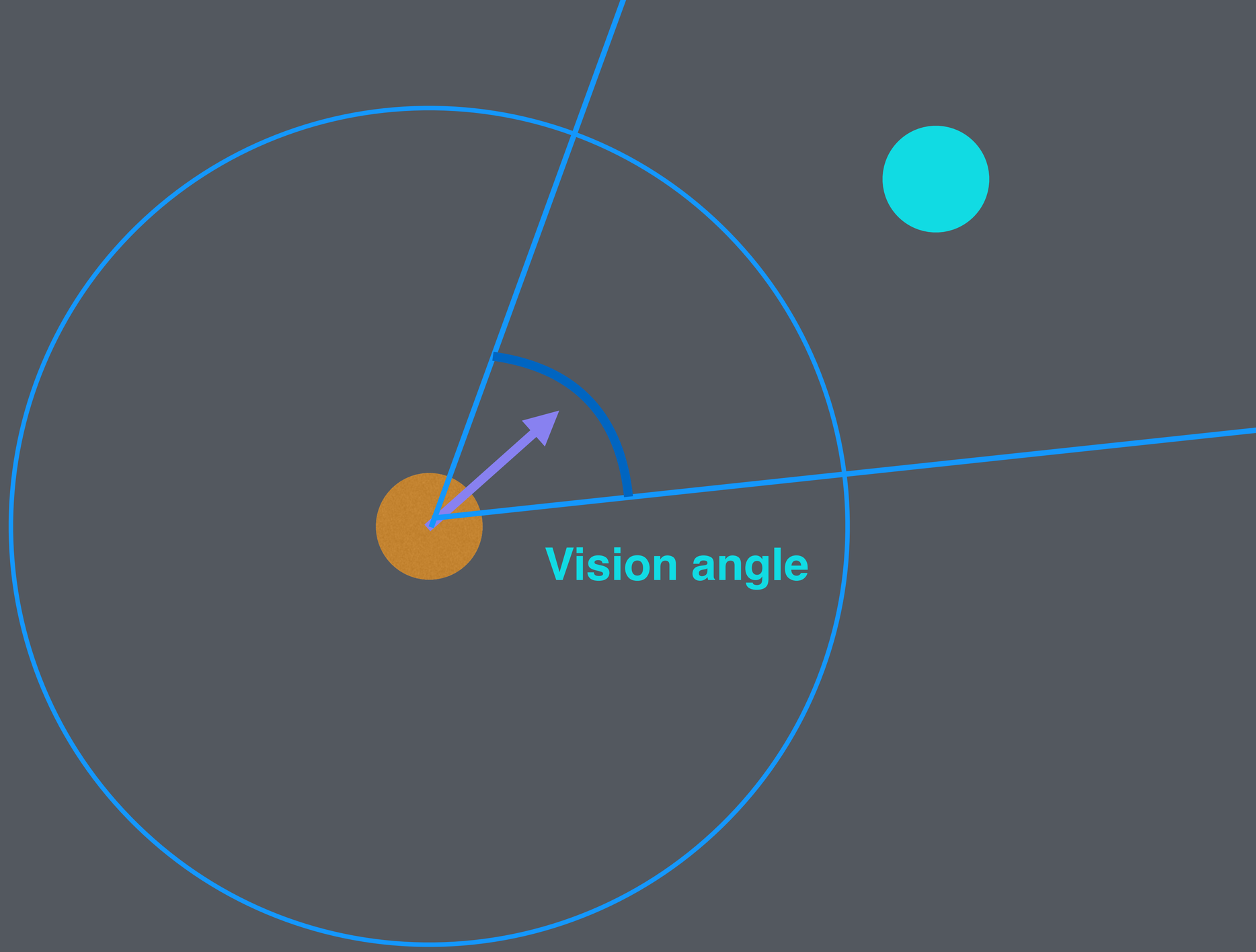

Vision angle



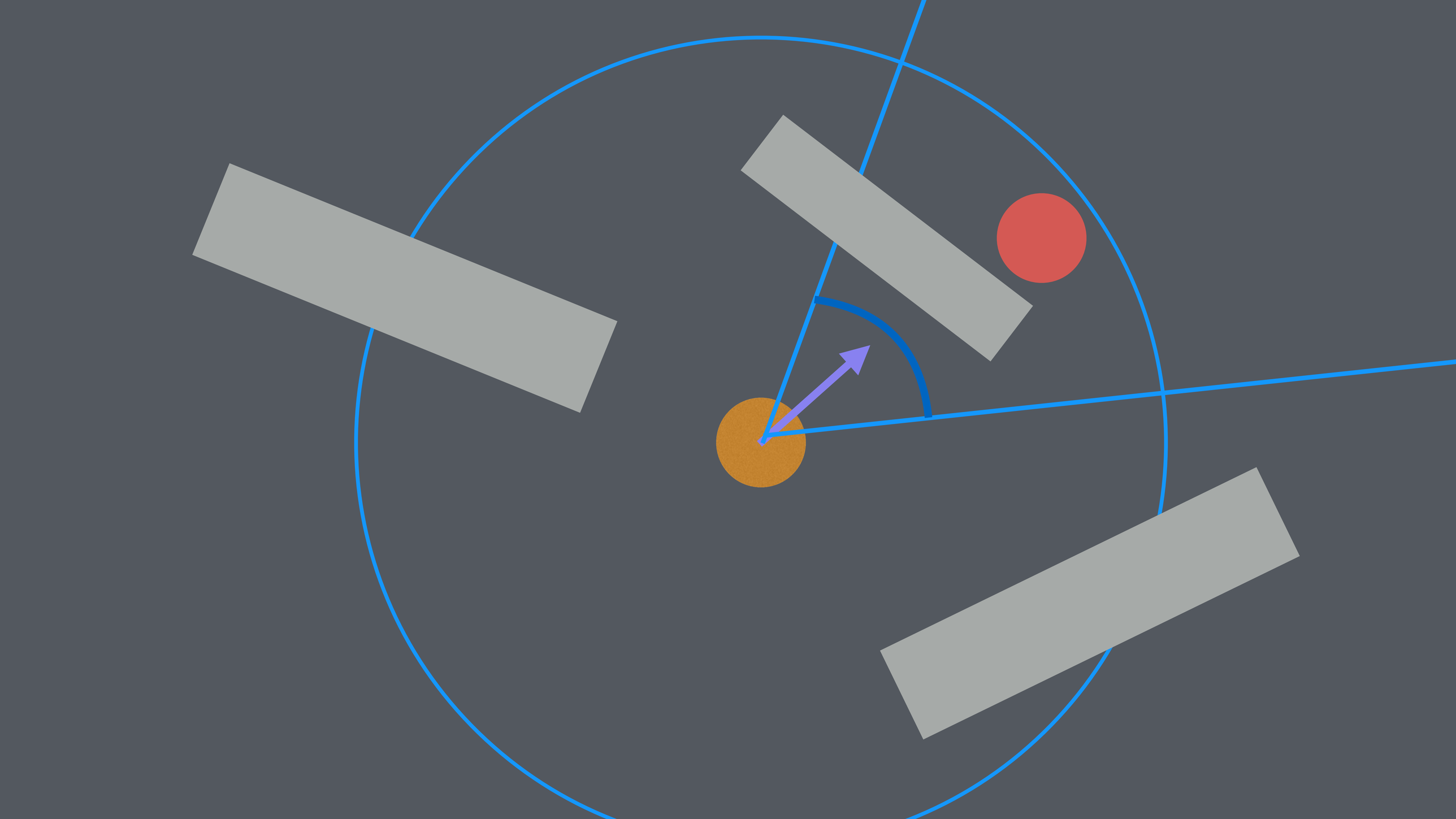
Calculating vision angle:

1. Calculate angle towards object. **atan2(normalizedDirectionToObject)**
2. Get difference between the rotation angle and angle towards object.
3. If absolute value of the difference is larger than half of our vision angle, we can’t see the object


Vision angle

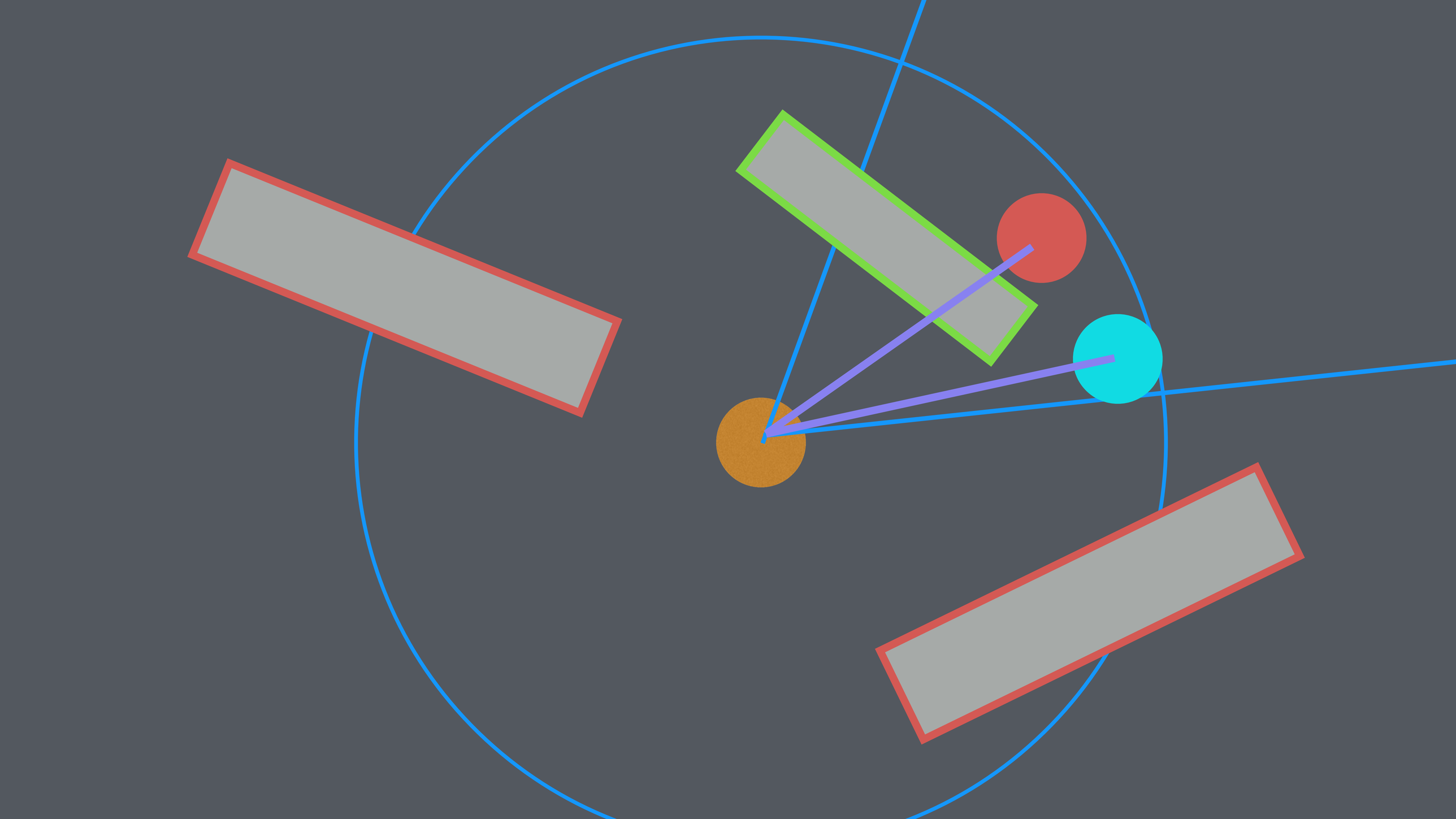

Vision angle

Vision obstacles.

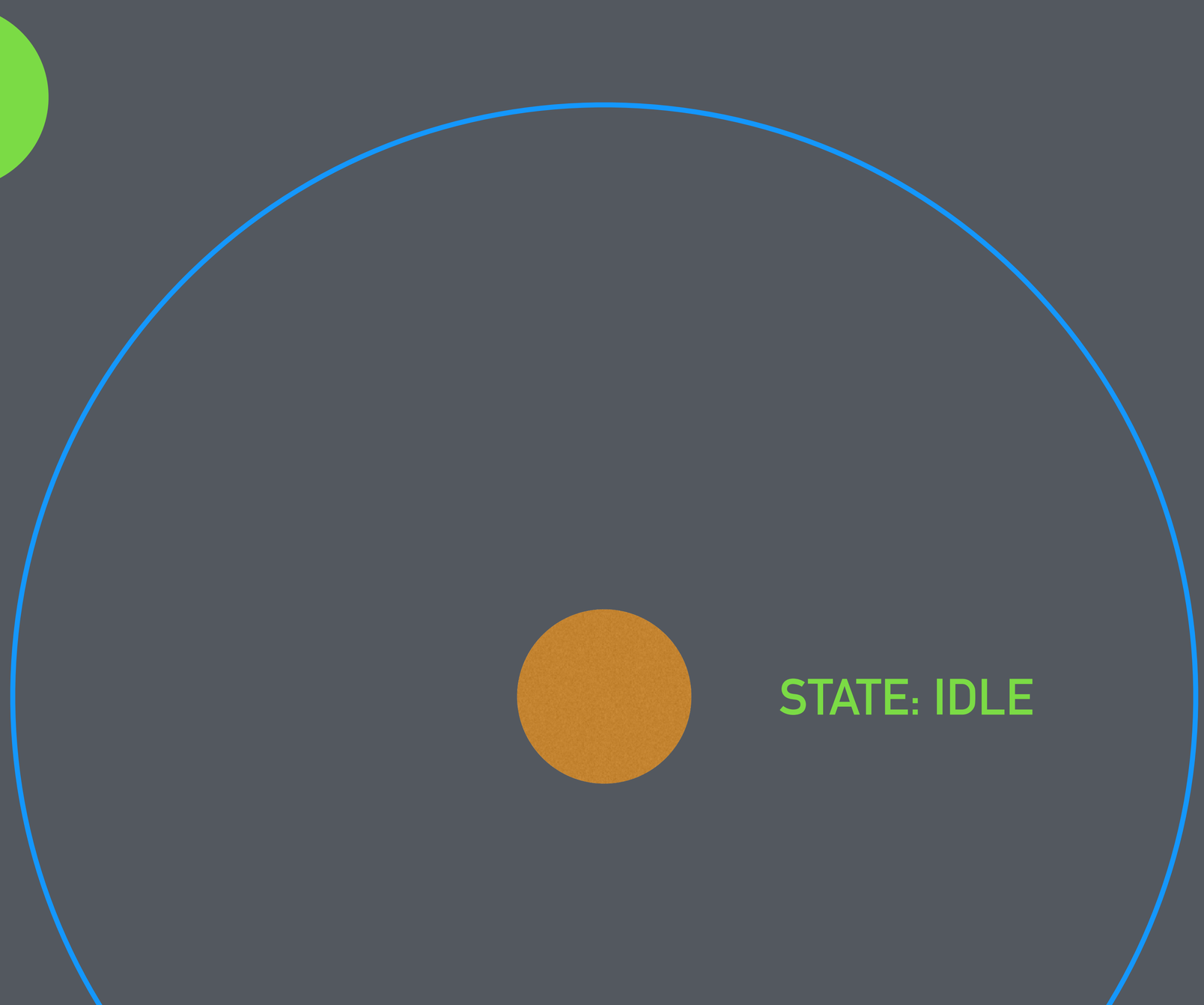


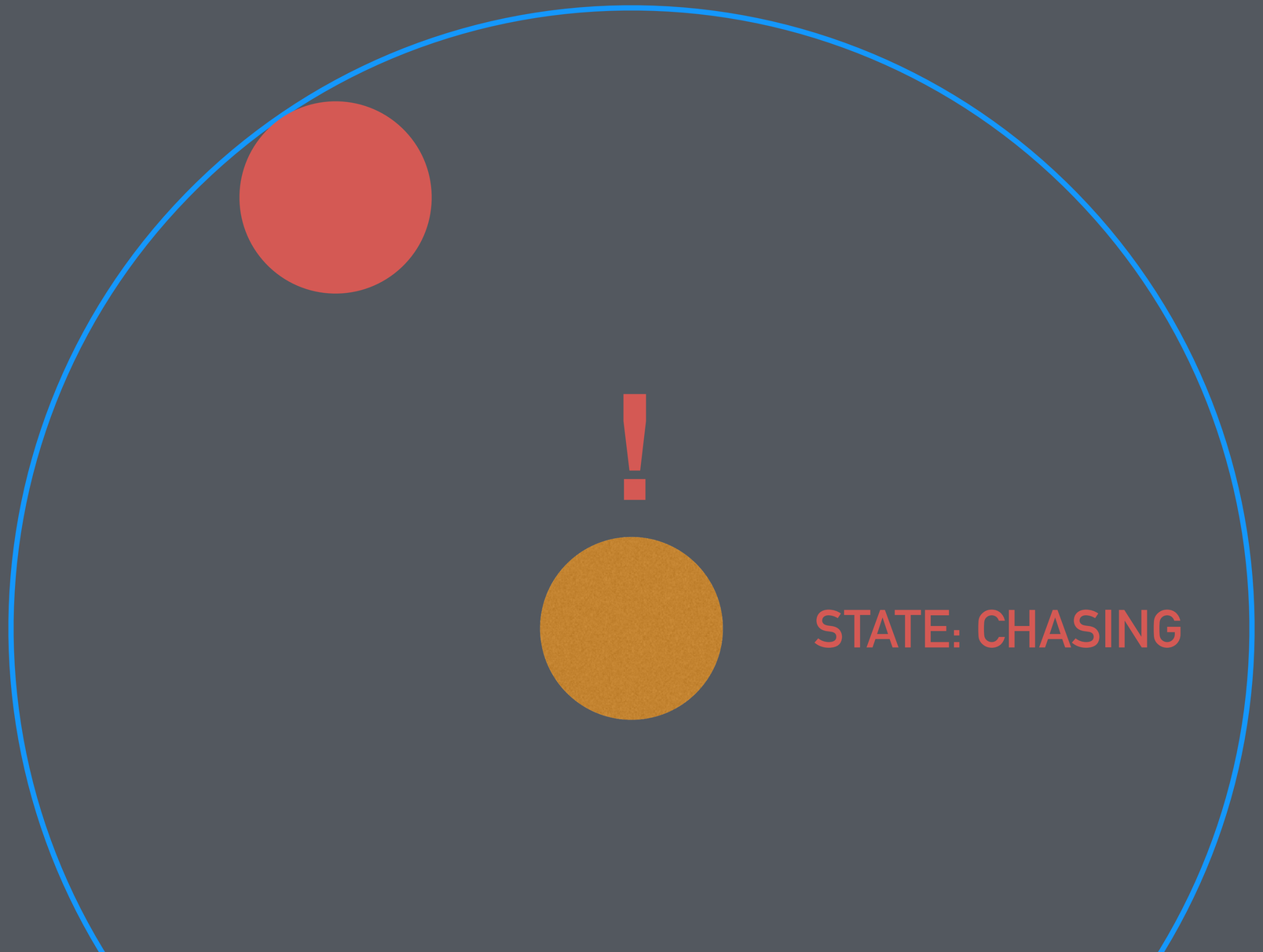
When seeing something, do a ray test from current position towards that object's position against all obstacles in the level.

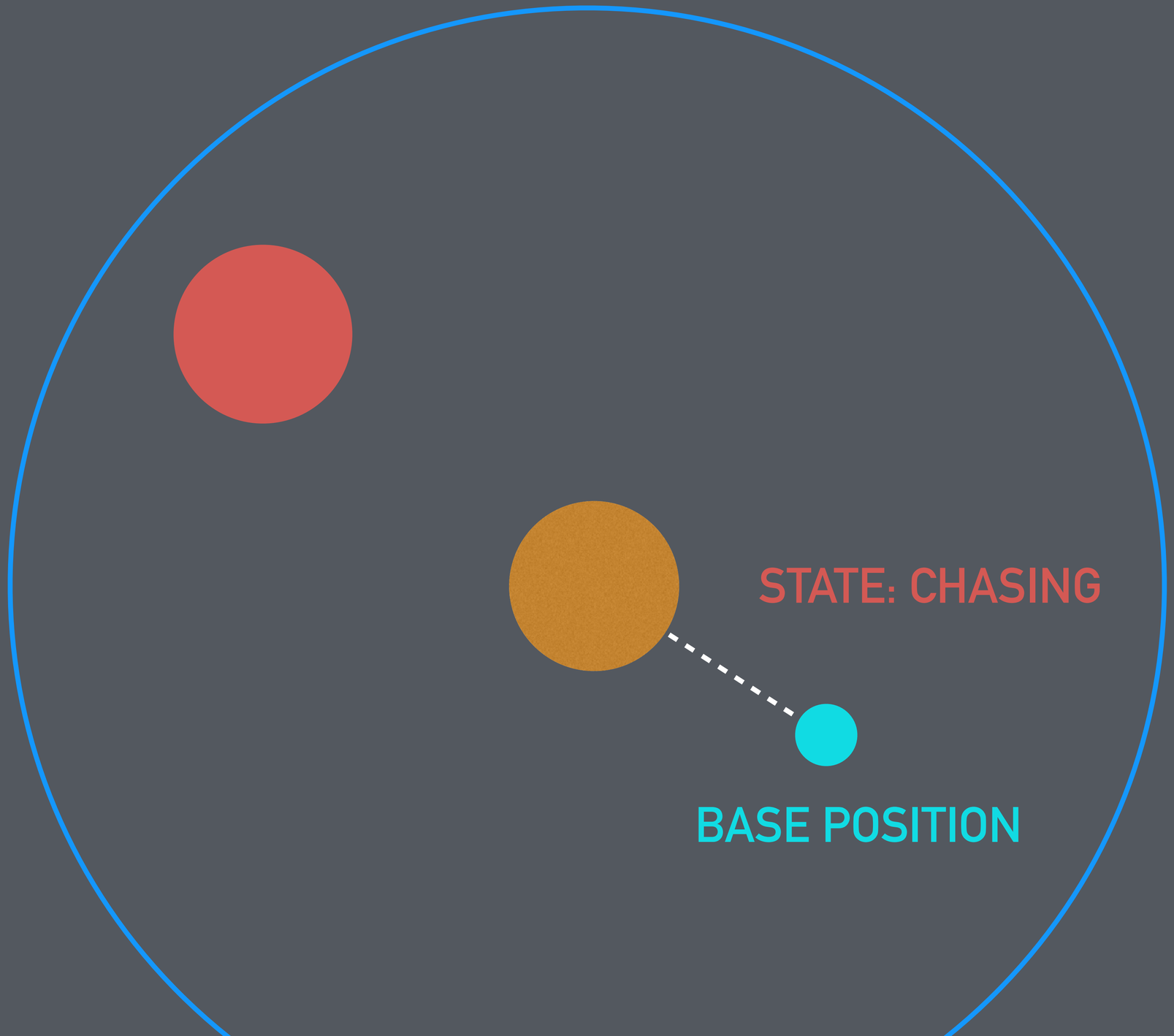
If it intersects an entity and the distance is closer than the object, then we can't see it.

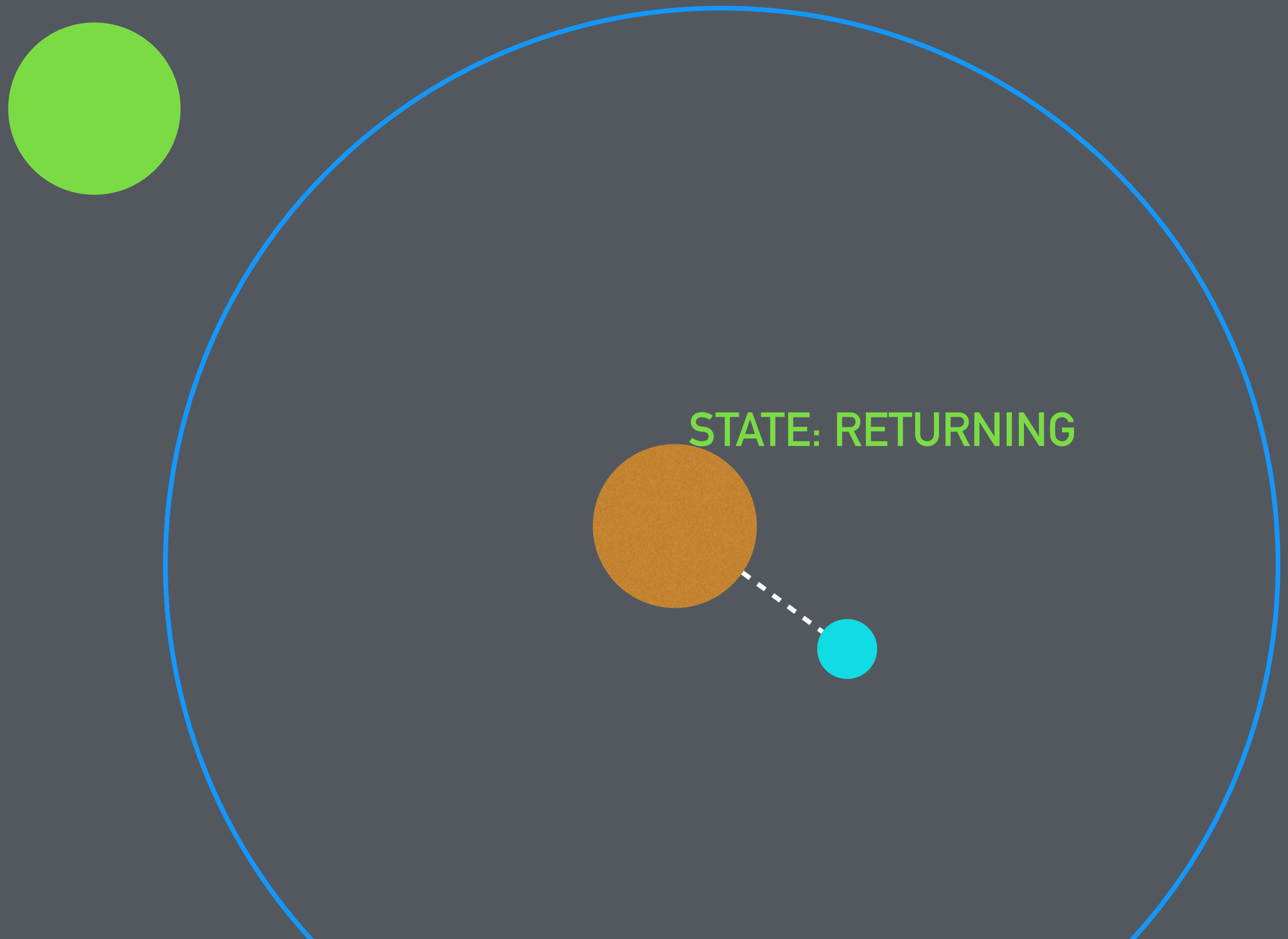


States


STATE: IDLE


!
STATE: CHASING


STATE: CHASING
BASE POSITION


STATE: RETURNING

Finite state machines.

Can only be in one of some predetermined states at a time.

IDLE

CHASING

RETURNING

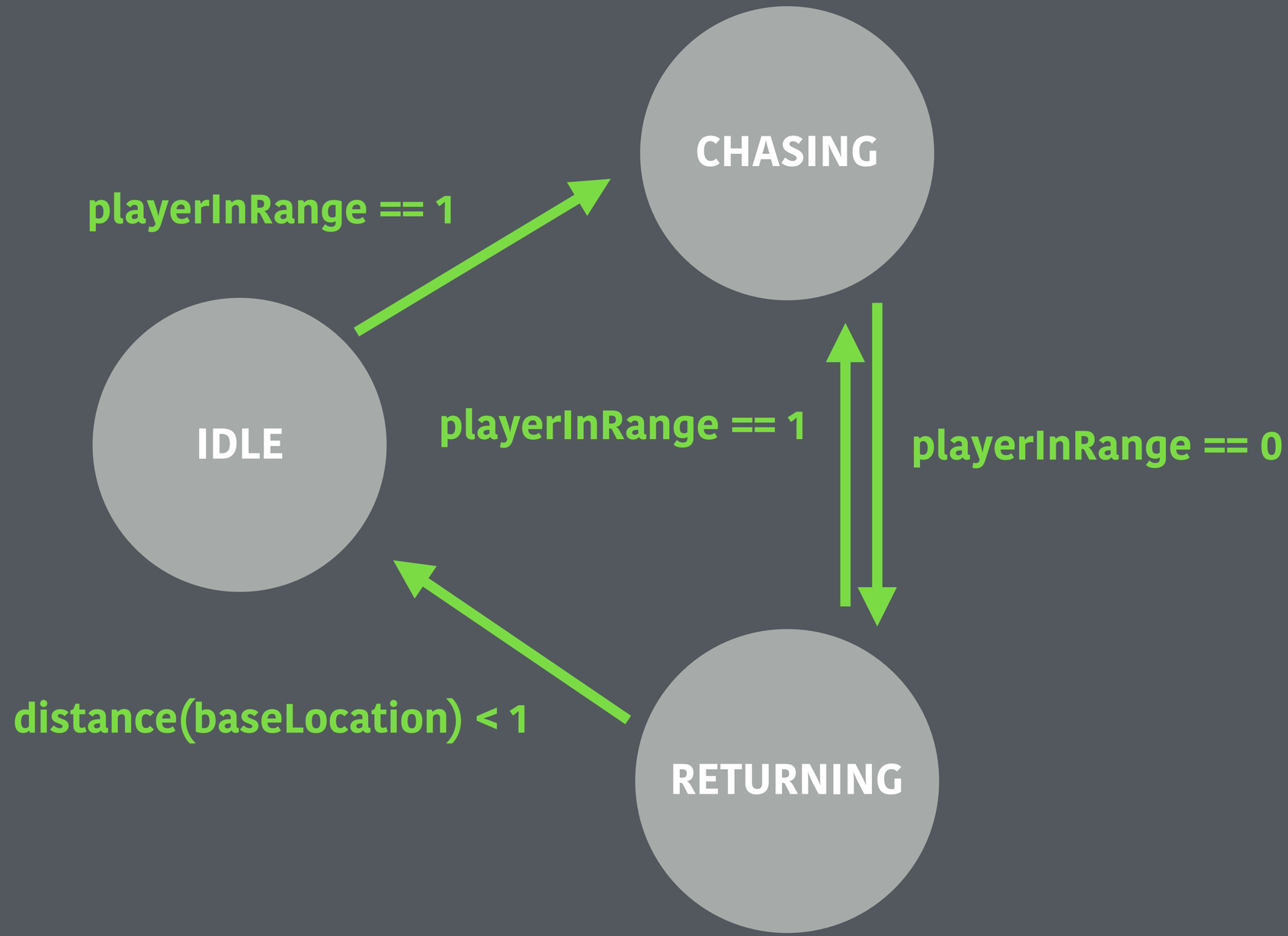
CURRENT STATE

IDLE

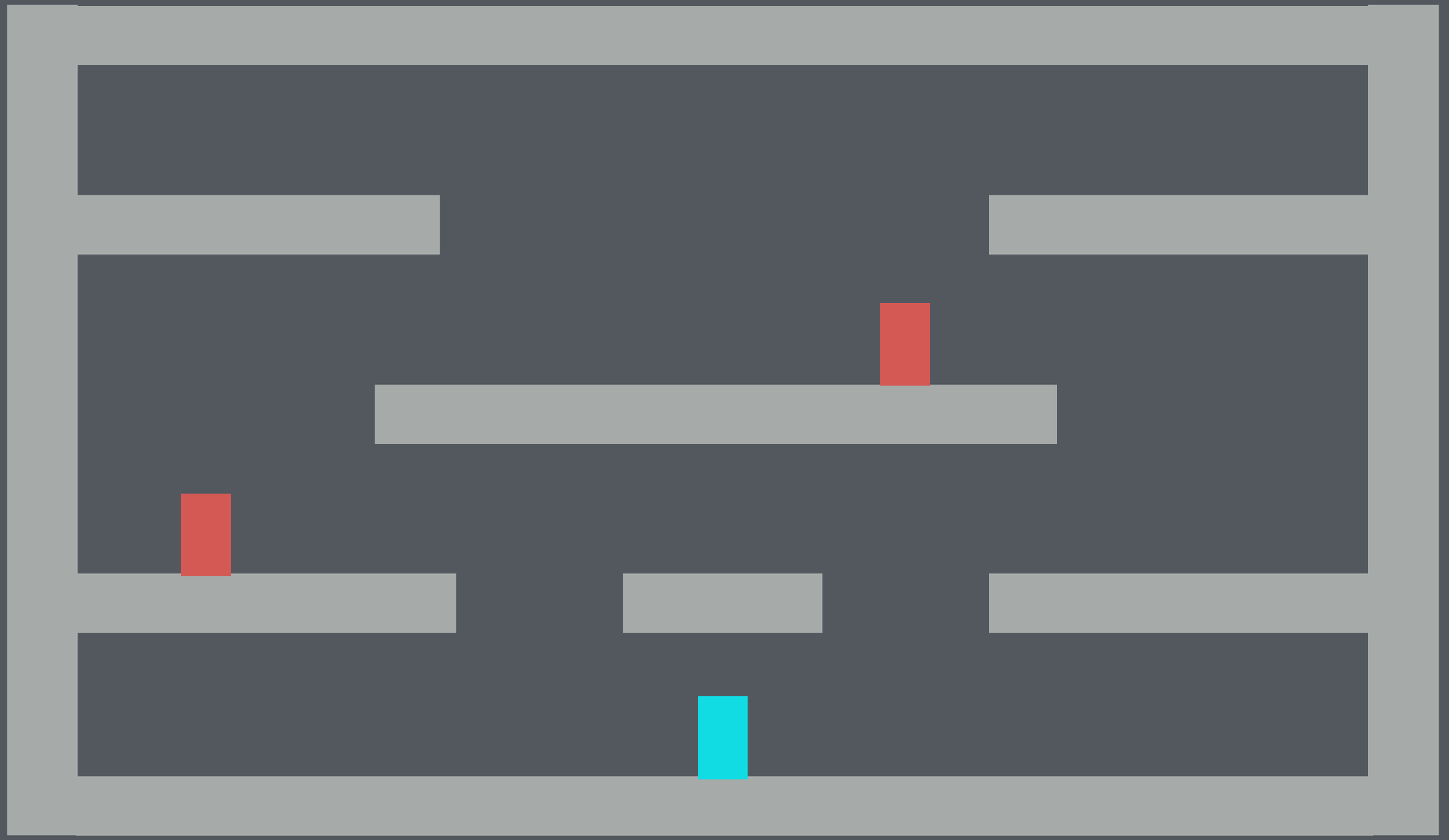
Update every frame based on
the current state.

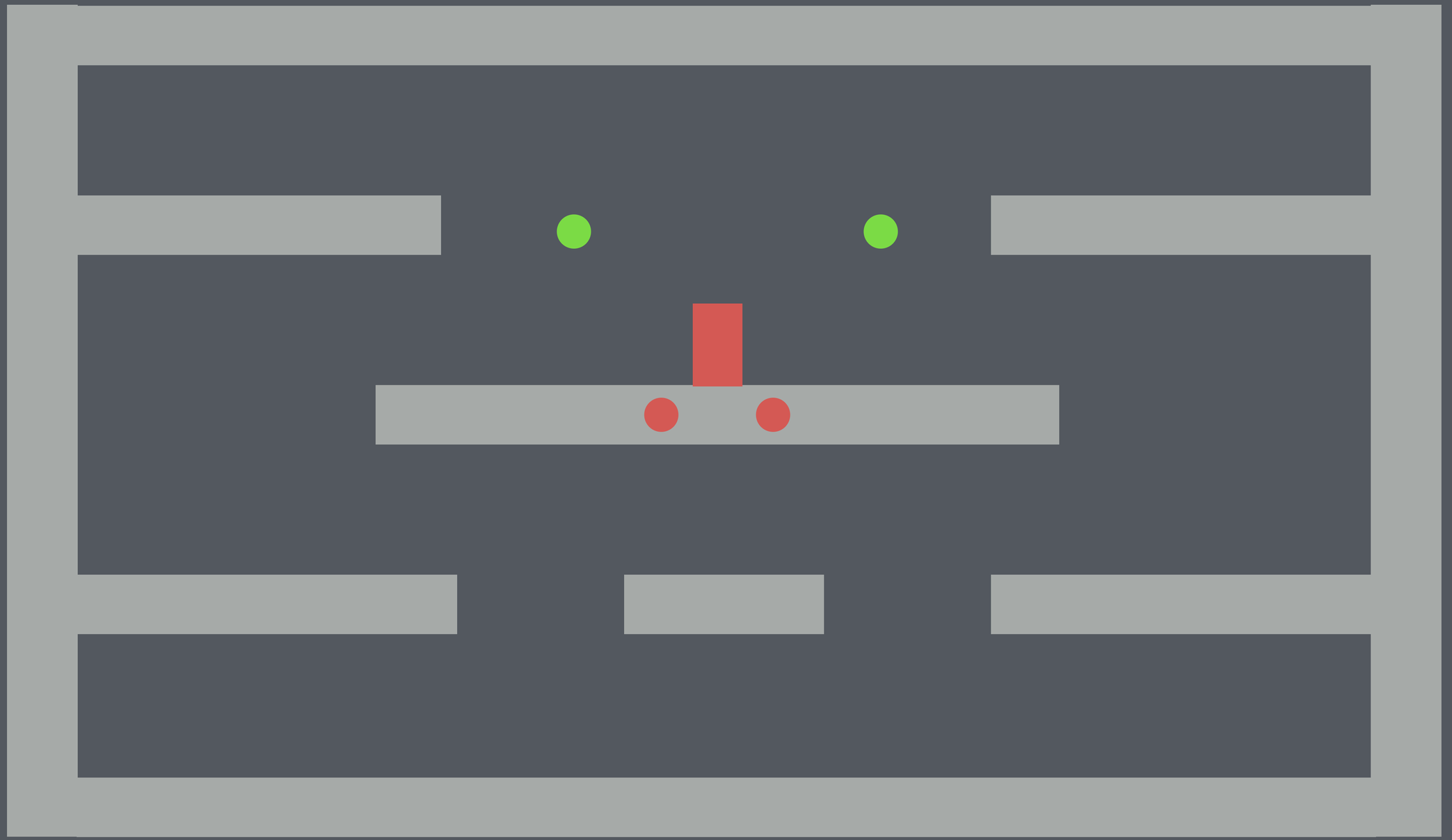
React to sensing data based on
the current state.

	PLAYER IN RANGE	PLAYER NOT IN RANGE	AT BASE POSITION	NOT AT BASE POSITION
IDLE	CHASING			
CHASING		RETURNING		
RETURNING	CHASING		IDLE	


CHASING
playerInRange == 1
IDLE
playerInRange == 1
playerInRange == 0
distance(baseLocation) < 1
RETURNING

Example:
Versus platformer.





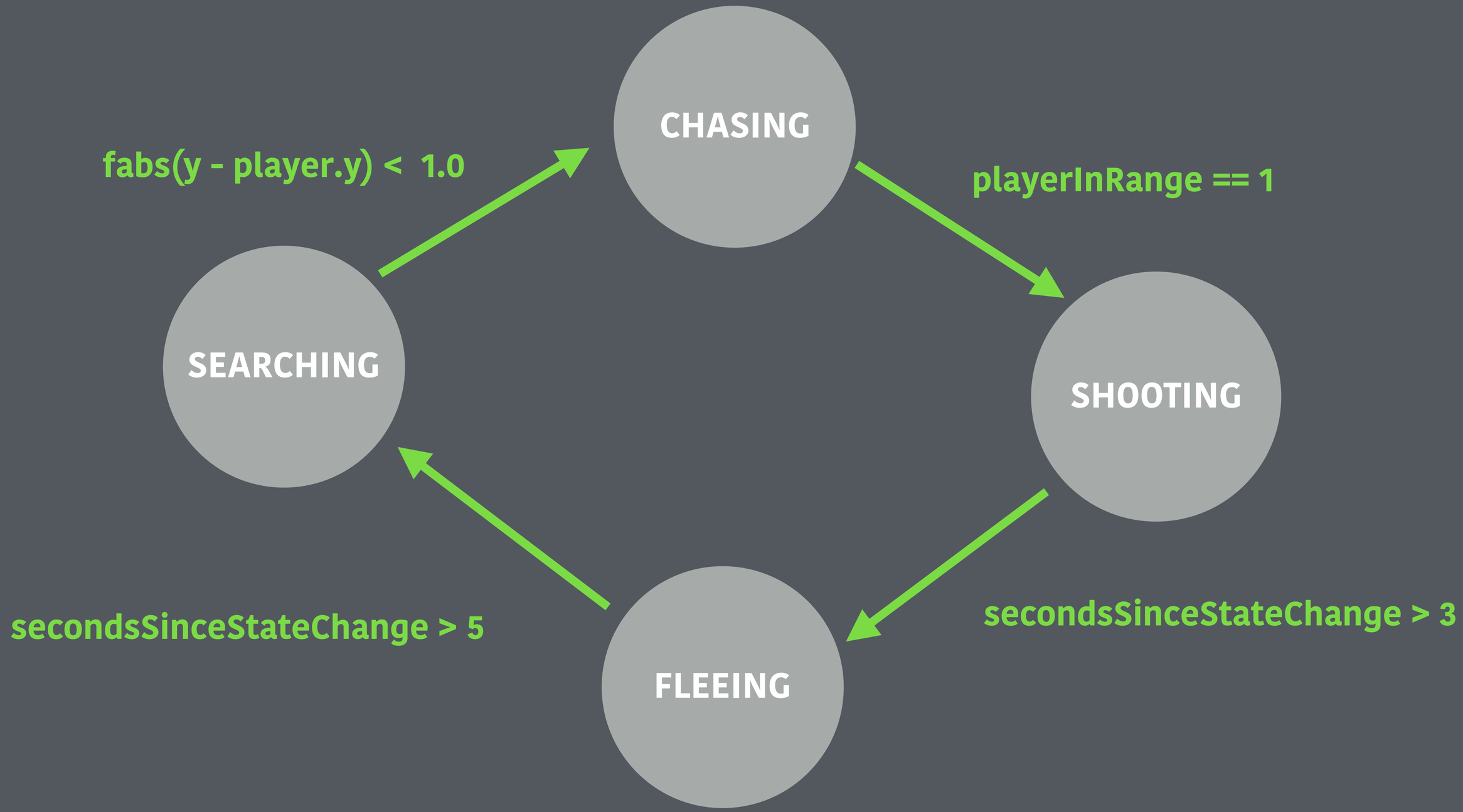
States

SEARCHING - Move forward, if edge sensor isn't colliding, turn around or drop down X% of the time. If jump sensor is colliding, jump N% of the time.

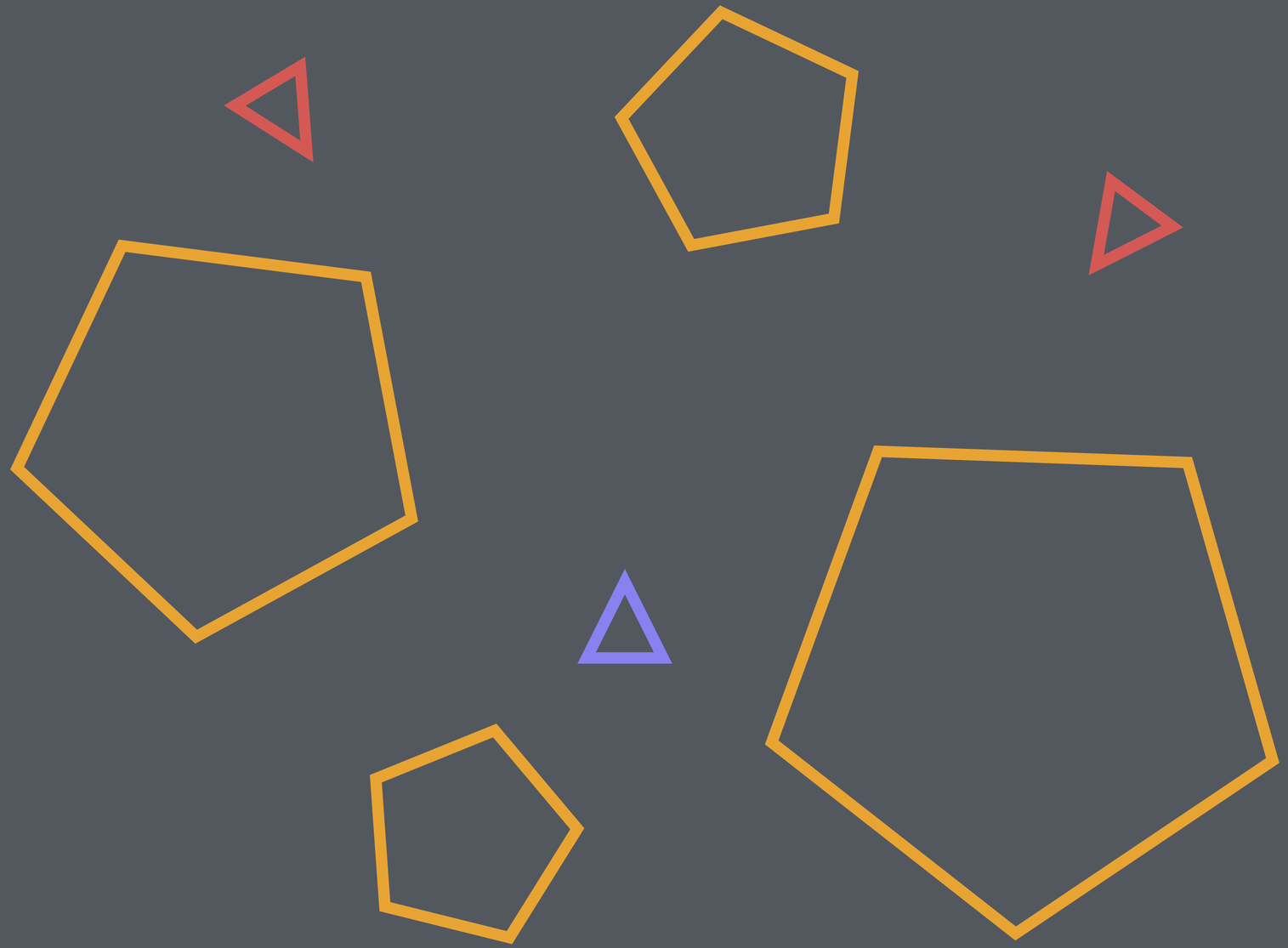
CHASING - Move towards player, if edge sensor isn't colliding, jump.

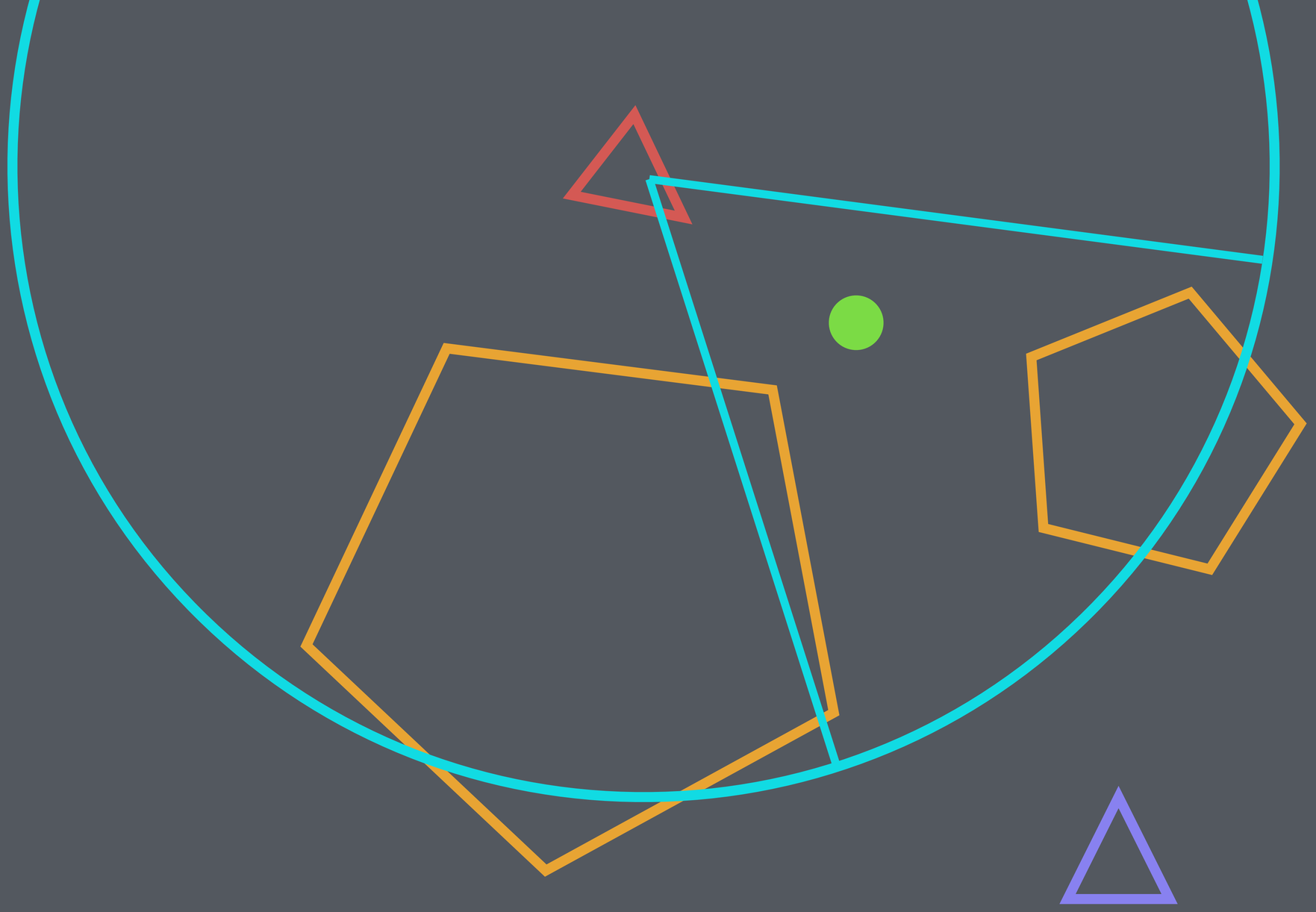
SHOOTING - Shoot gun.

FLEEING - Move away from player, if edge sensor isn't colliding, do nothing. If jump sensor is colliding, jump 100% of the time.


CHASING
fabs(y - player.y) < 1.0
playerInRange == 1
SEARCHING
SHOOTING
secondsSinceStateChange > 5
secondsSinceStateChange > 3
FLEEING

Example:
Versus Asteroids.





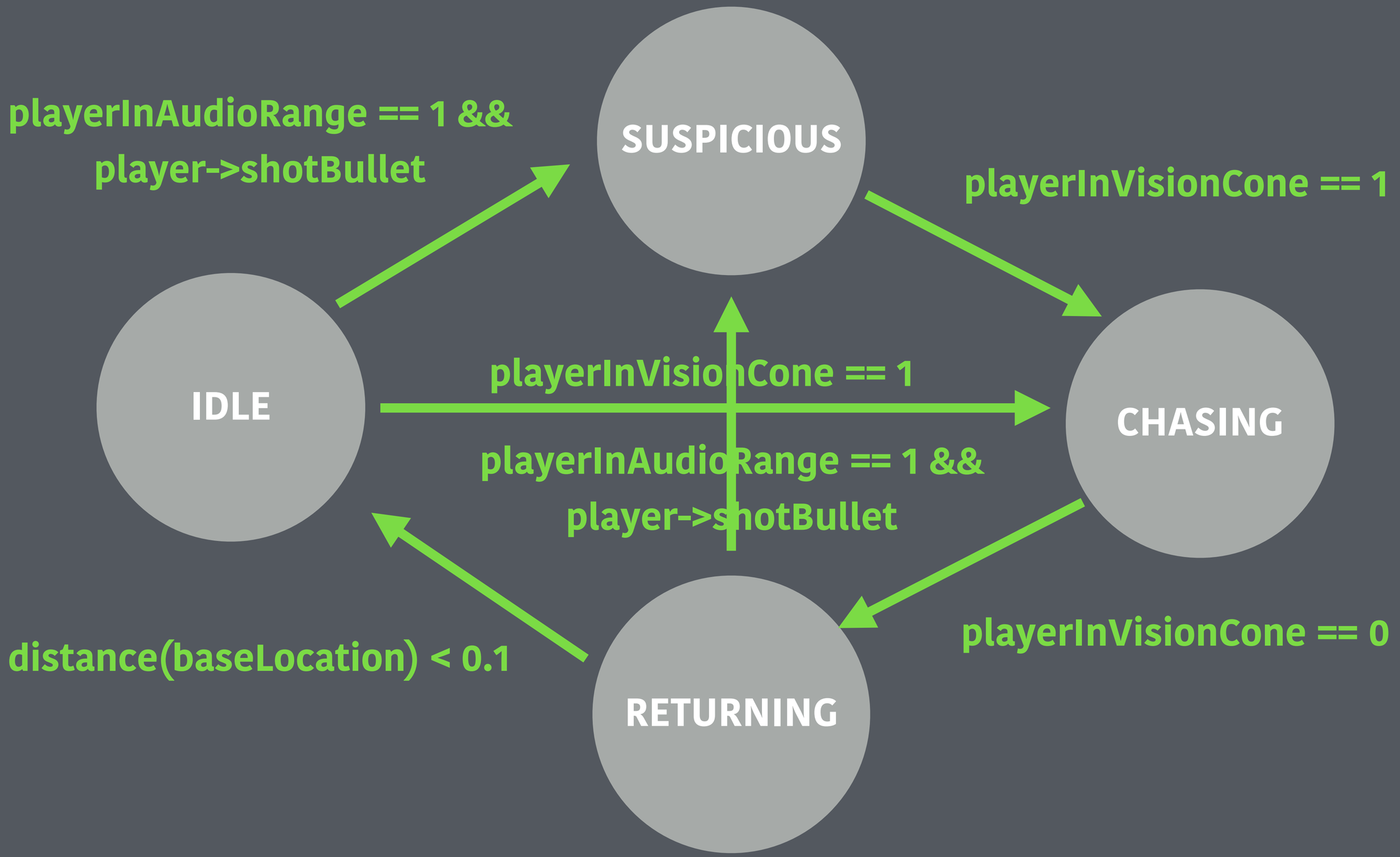
States

IDLE - Stay in place, maybe slowly rotate.

SUSPICIOUS - Rotate and move towards suspicious position.

CHASING - Rotate and move towards player while shooting.

RETURNING - Rotate and move towards base point.


playerInAudioRange == 1 &&
player->shotBullet
SUSPICIOUS
playerInVisionCone == 1
IDLE
playerInVisionCone == 1
playerInAudioRange == 1 &&
player->shotBullet
CHASING
distance(baseLocation) < 0.1
RETURNING
playerInVisionCone == 0