

Game AI.

Part 1



<https://www.youtube.com/watch?v=GLKSuuOvnTw> 1

What is game AI?

Basic AI components.

- Sensing
- Pathfinding
- States

Sensing

Sensing the player.



Sensing the world.



Pathfinding

Navigating the level.



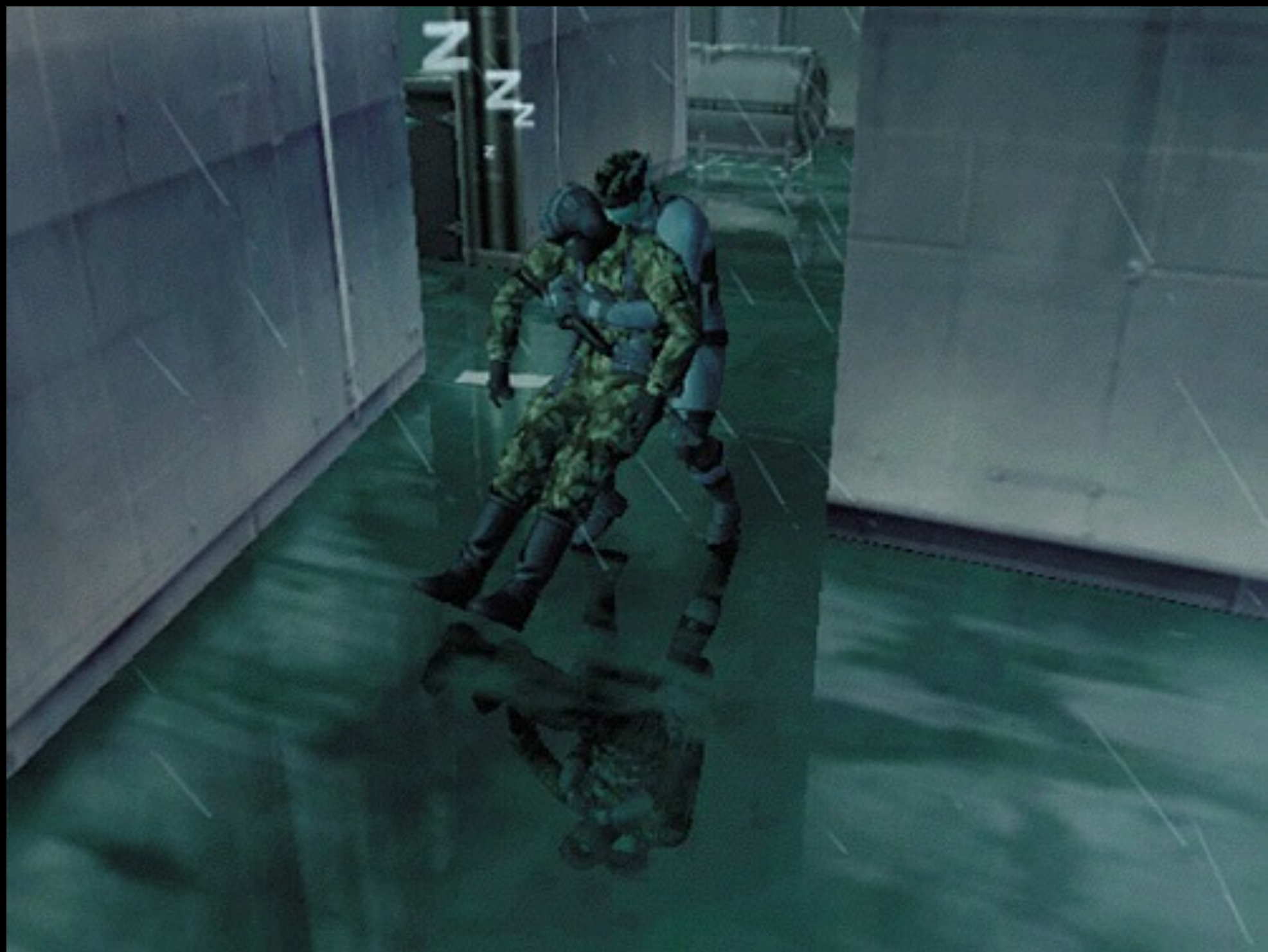
States



Normal



Angry

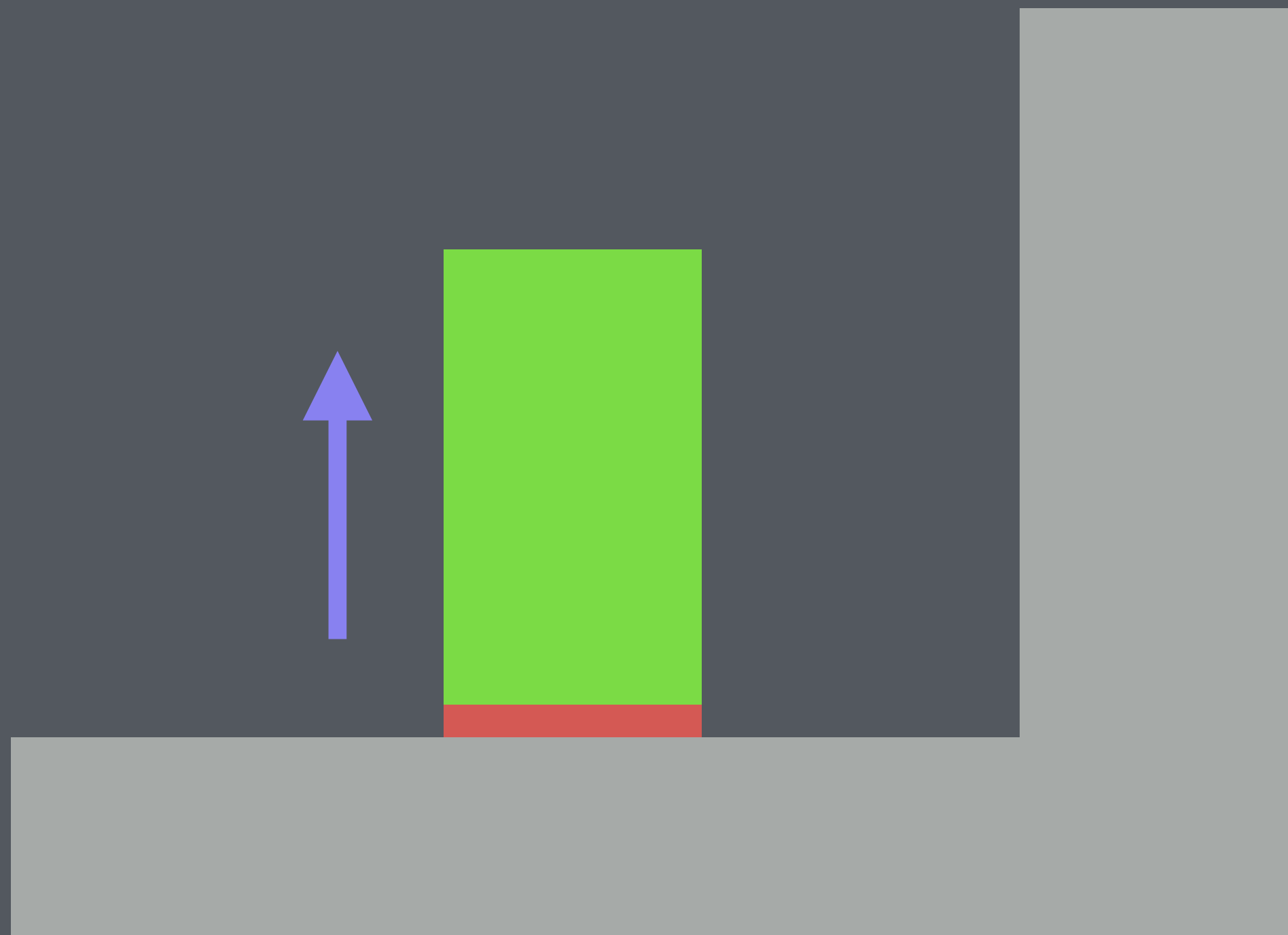


Sensing basics.

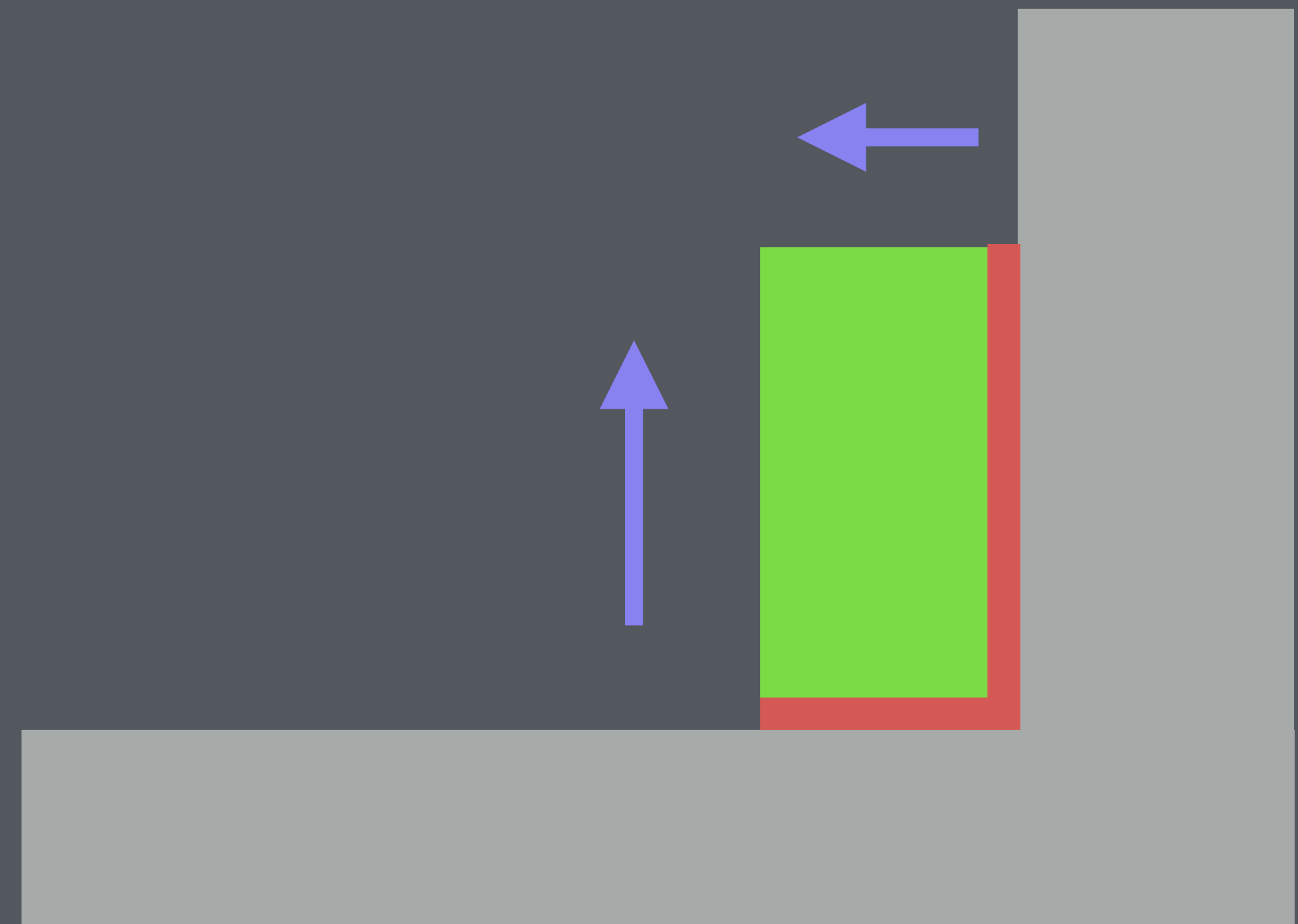
Collision sensing.

Collision flags

When we do our adjust, set collision flag for that direction to true.



collidingBottom = true



collidingBottom = true

collidingRight = true

Collision flag sensing example:
turning around at a wall.

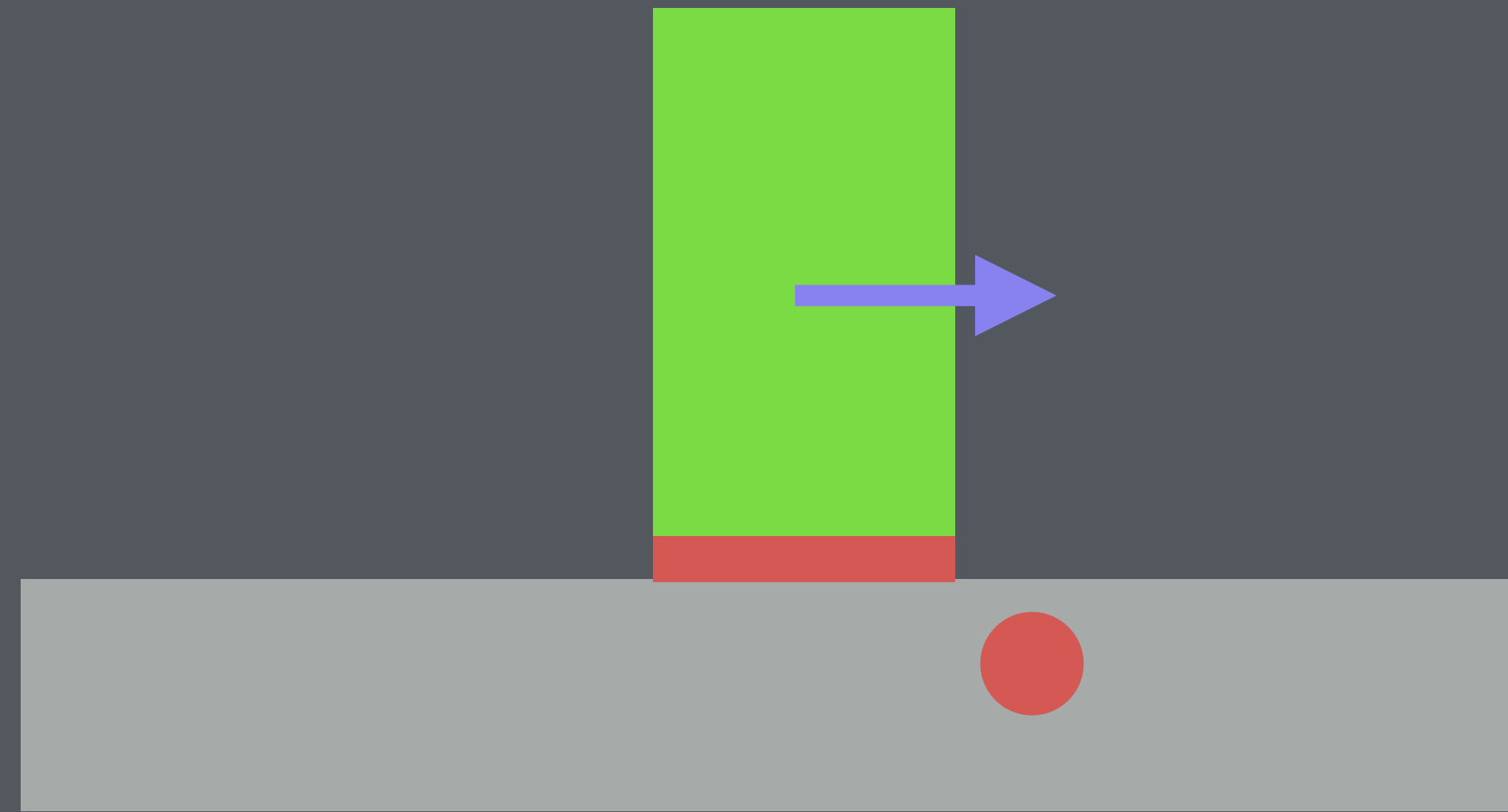


When an X-axis collision flag is set, inverse the X
acceleration.



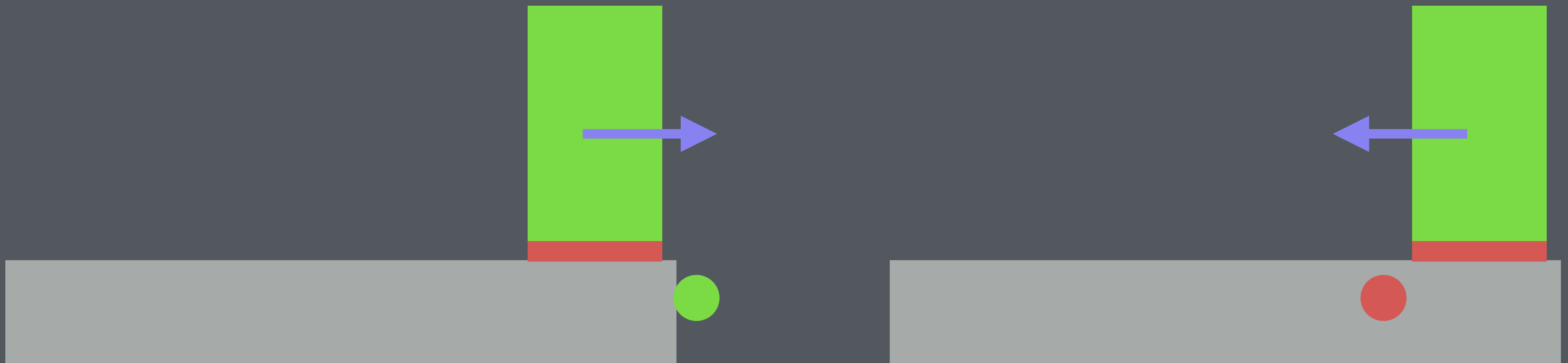
Problem: turning around at an **edge**.

Collision sensors.

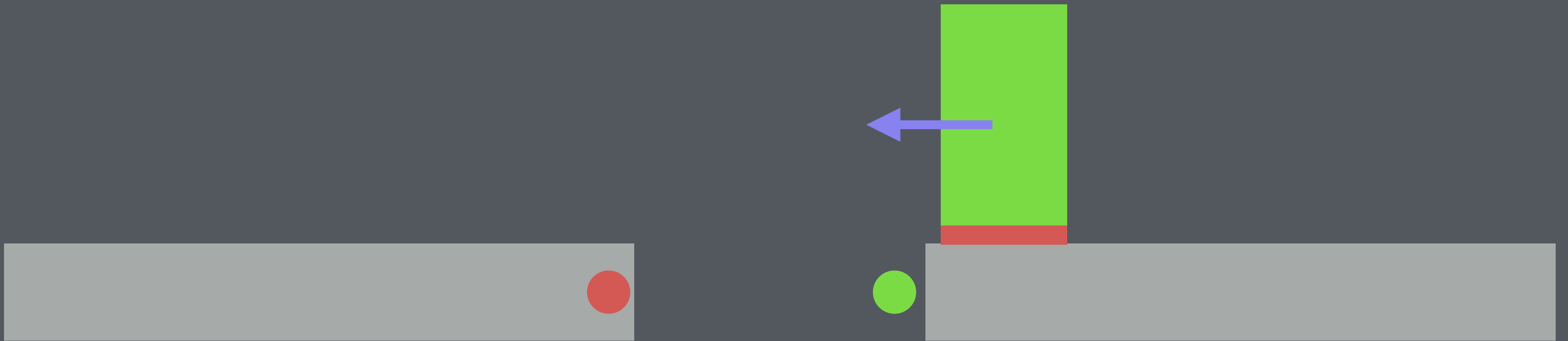


Check for collision using a **detector point**.

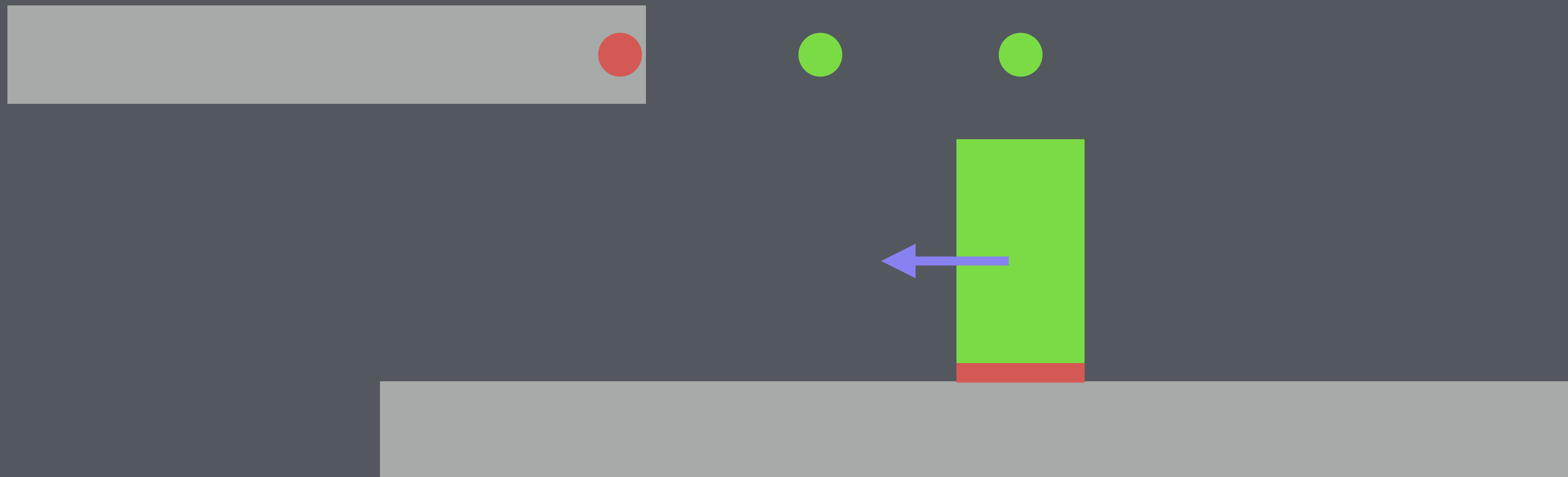
(use **worldToTileCoordinates** if using a tilemap)



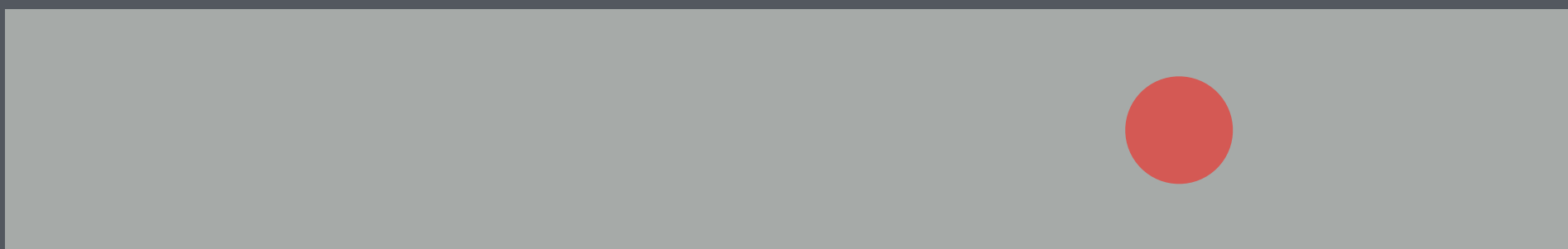
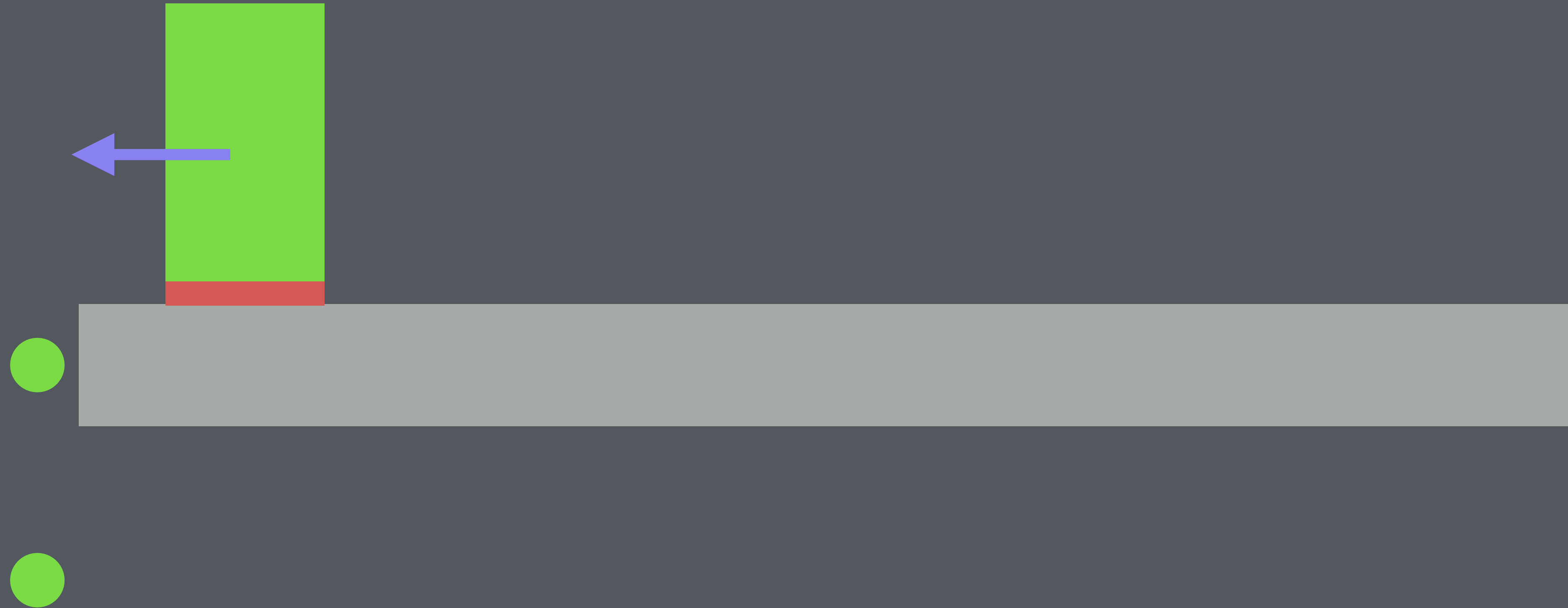
If **detector point** not colliding and our bottom collision flag is set (don't want to set it off when jumping), we've reached an edge!



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

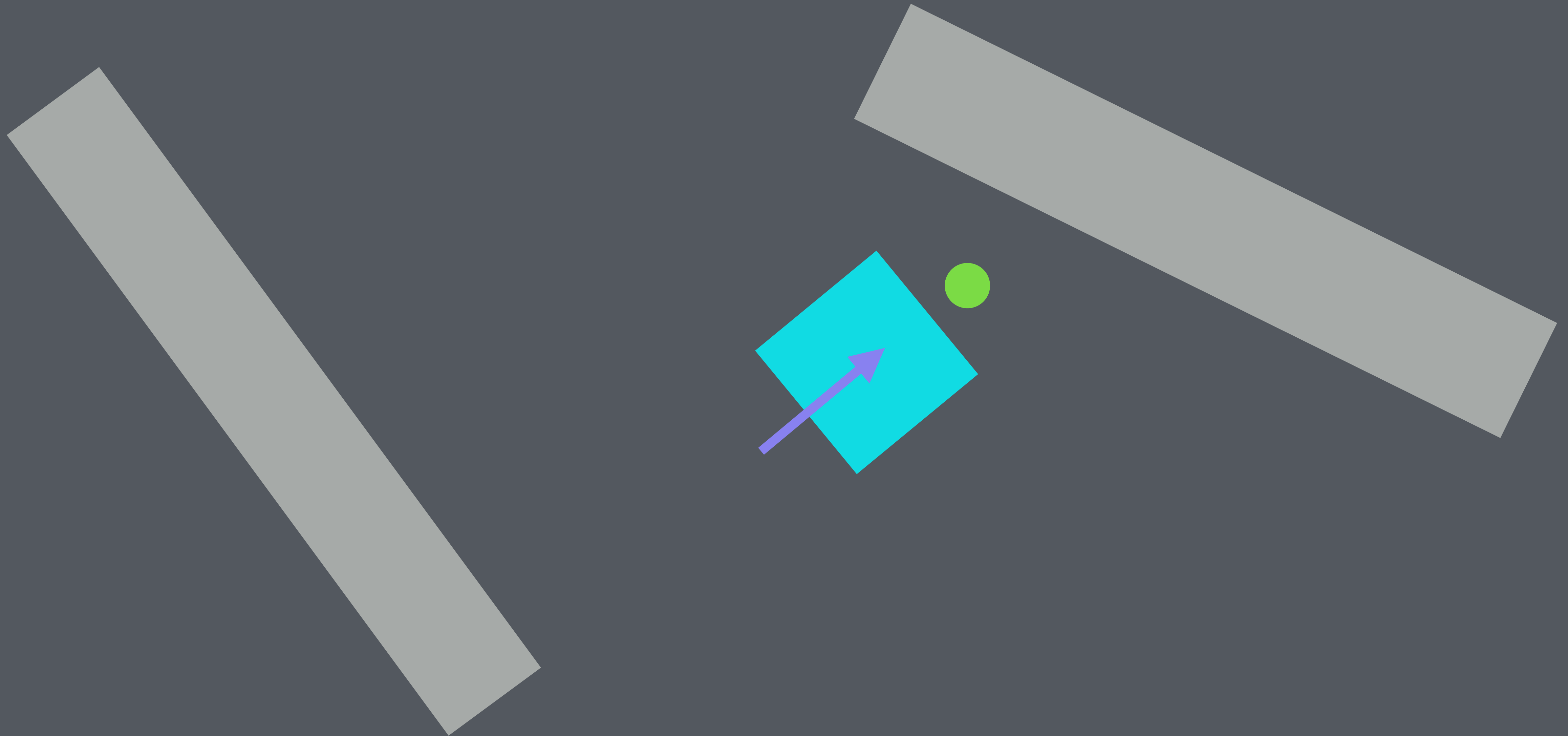


Can use 2 or 3 sensor points to see if possible
to jump up to a platform.

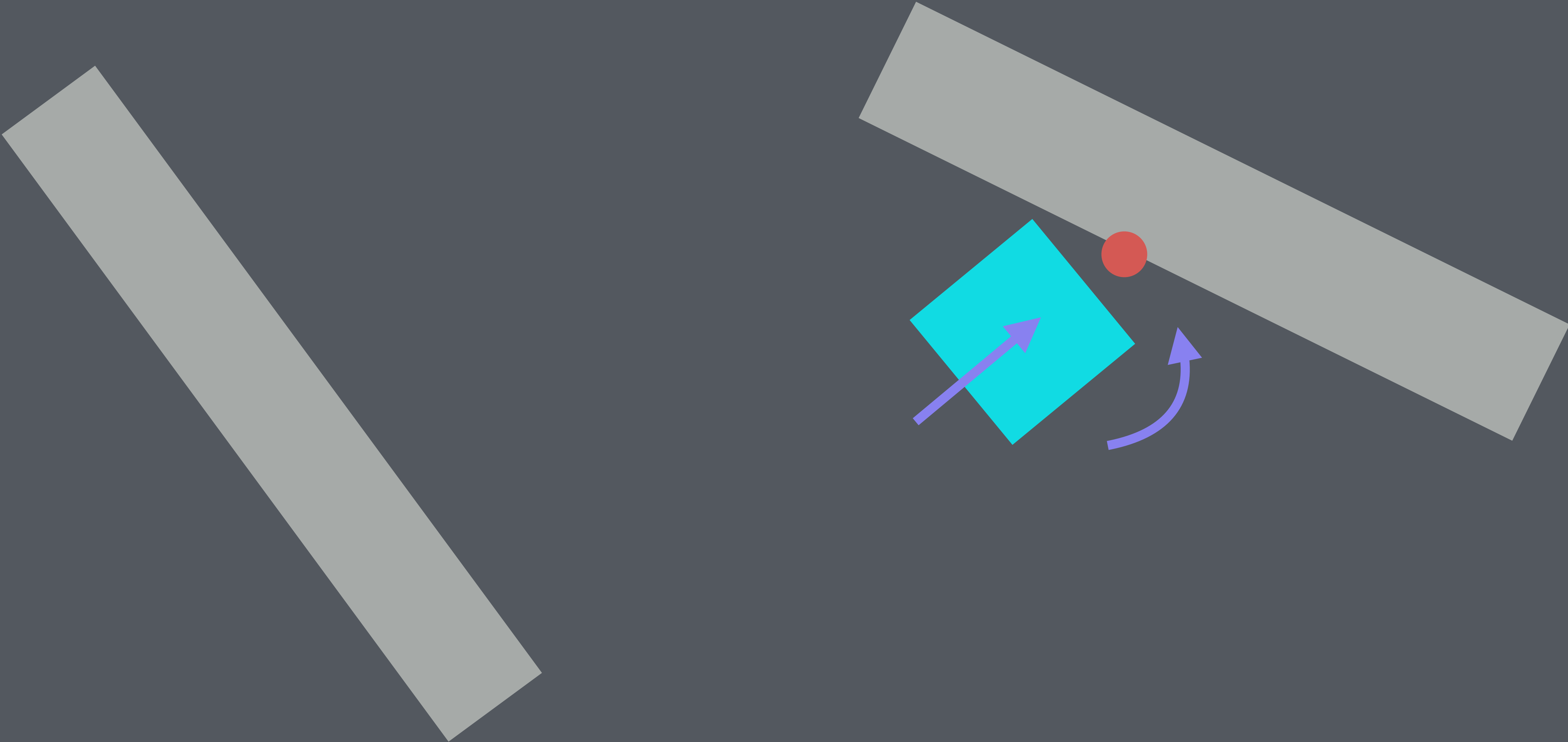


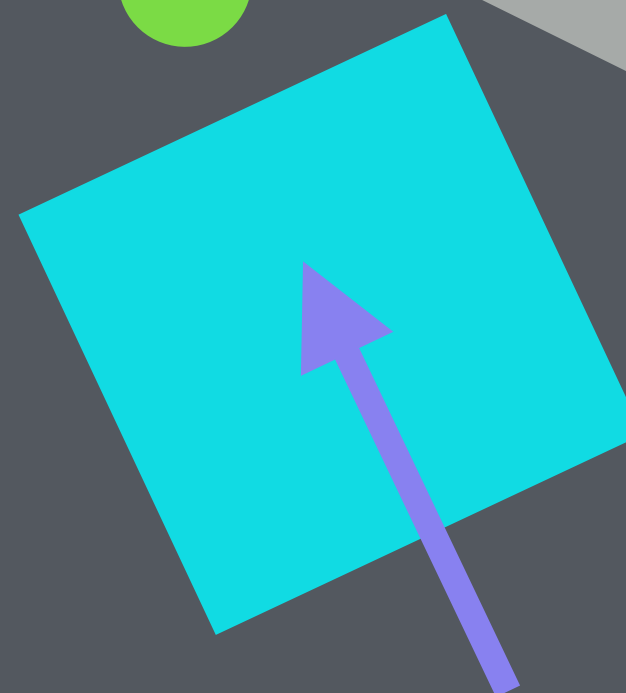
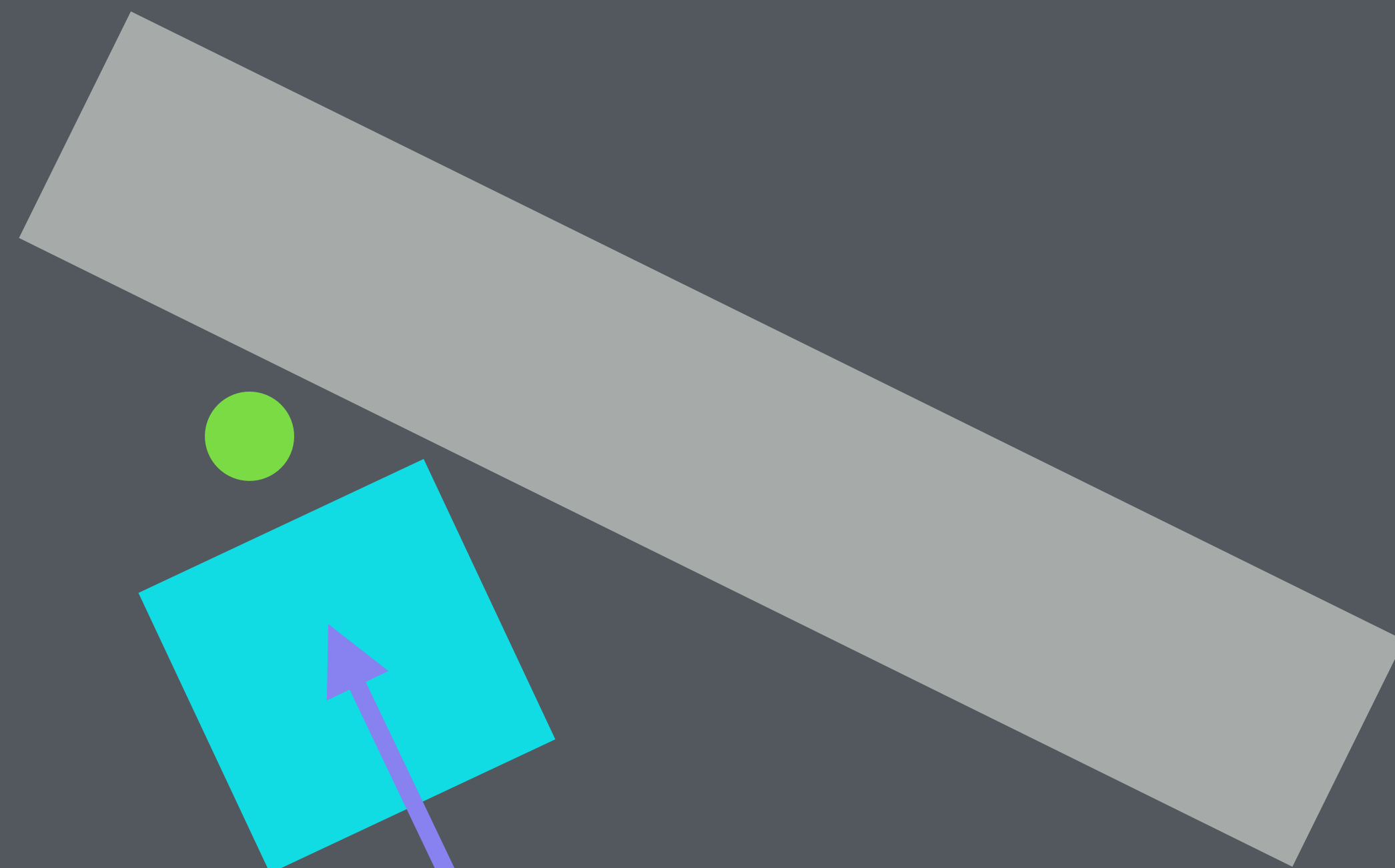
Or if it's safe to jump down...

Collision sensors in non-axis aligned collisions.

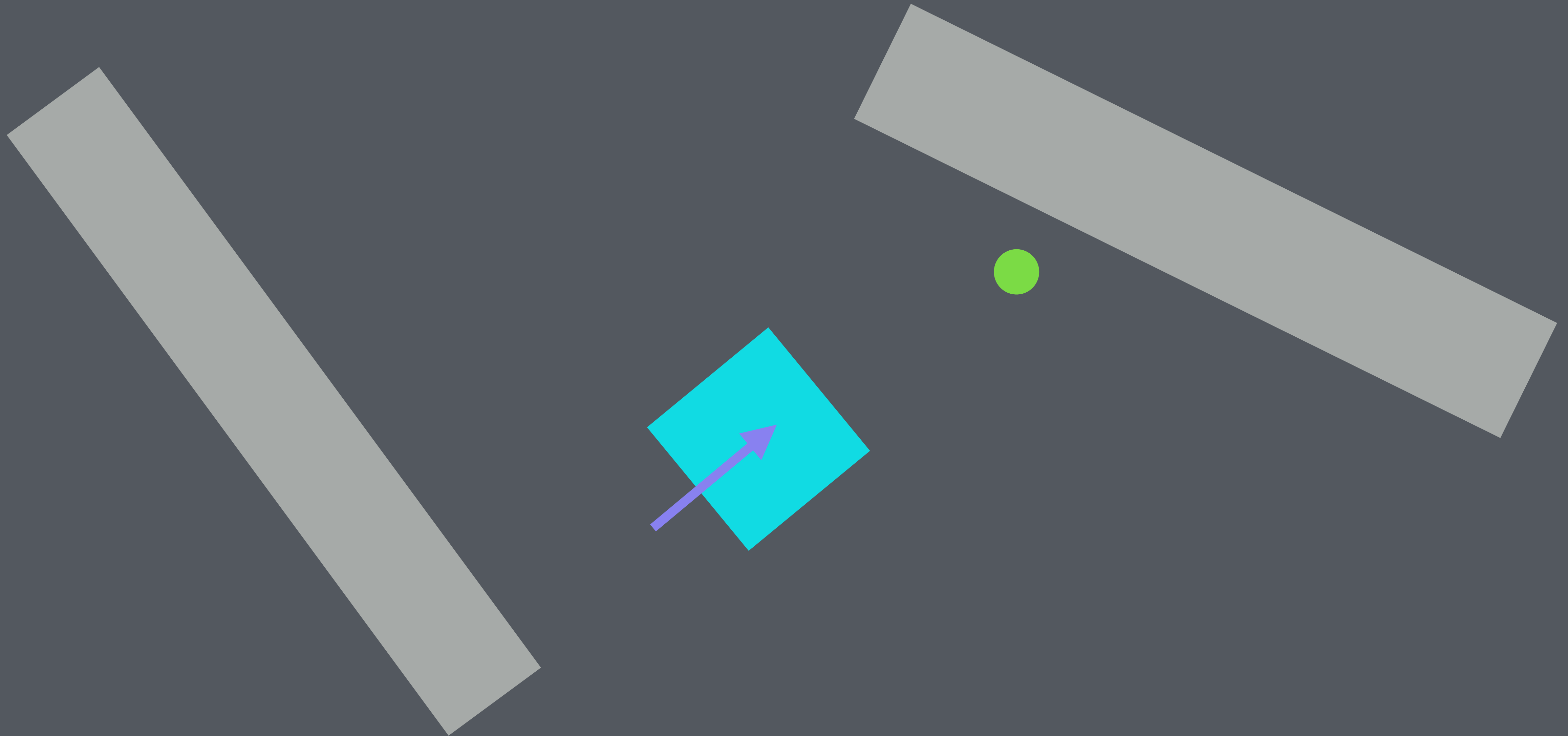


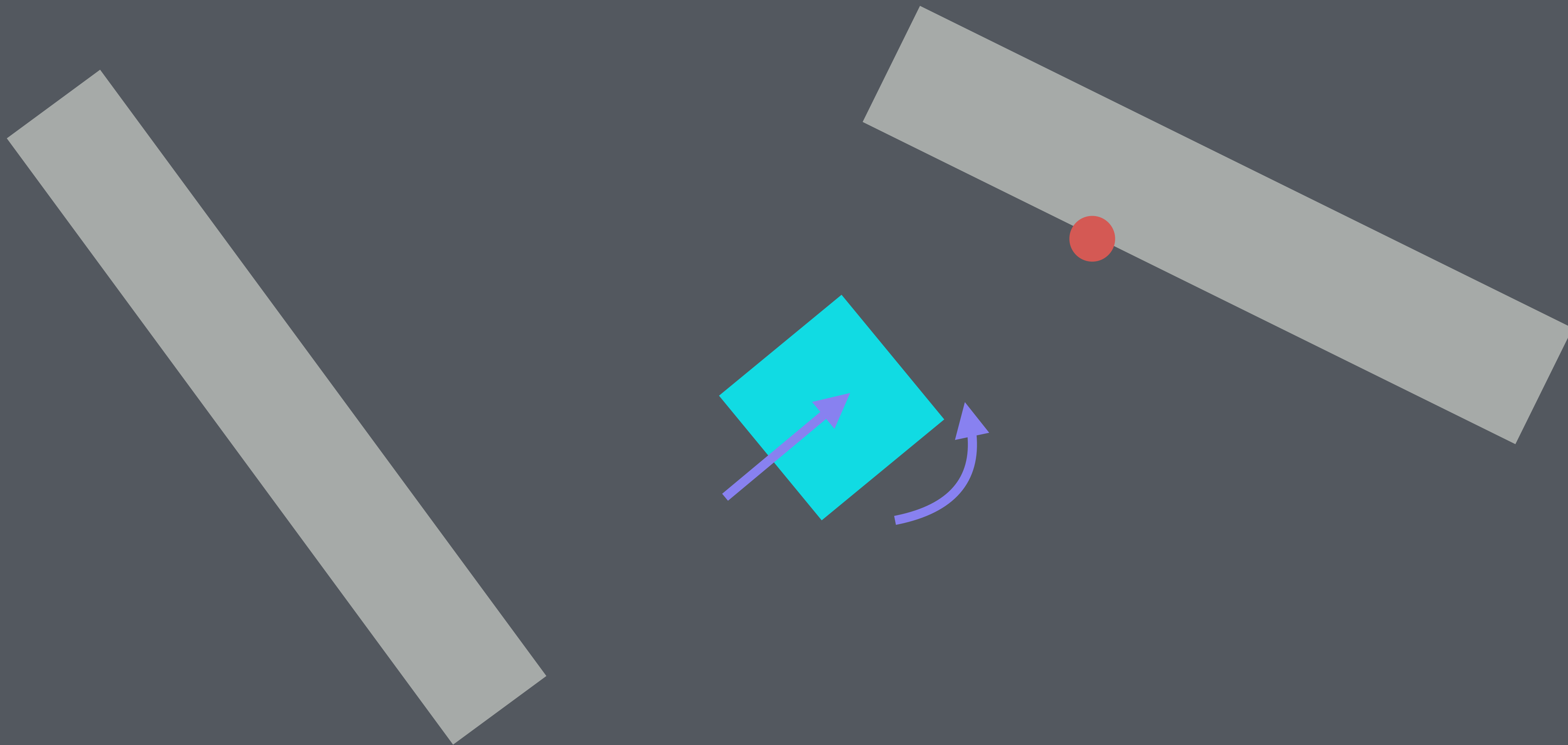
Point in rotated rectangle collision detection.

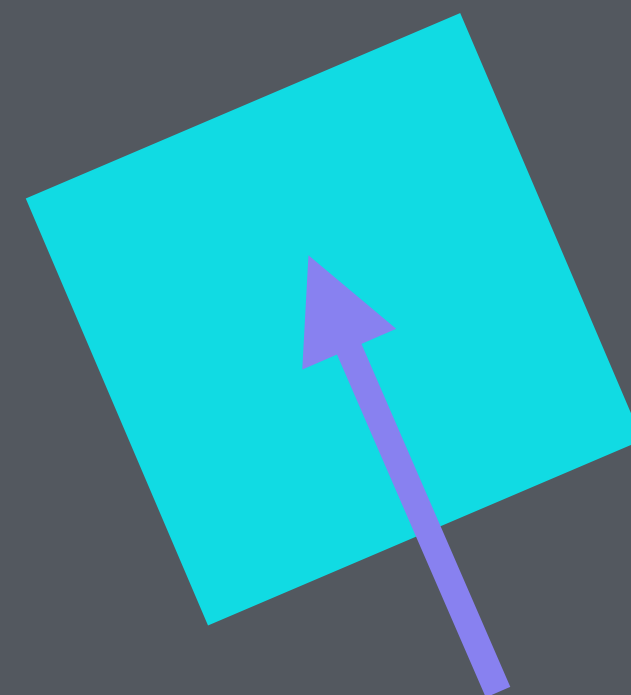
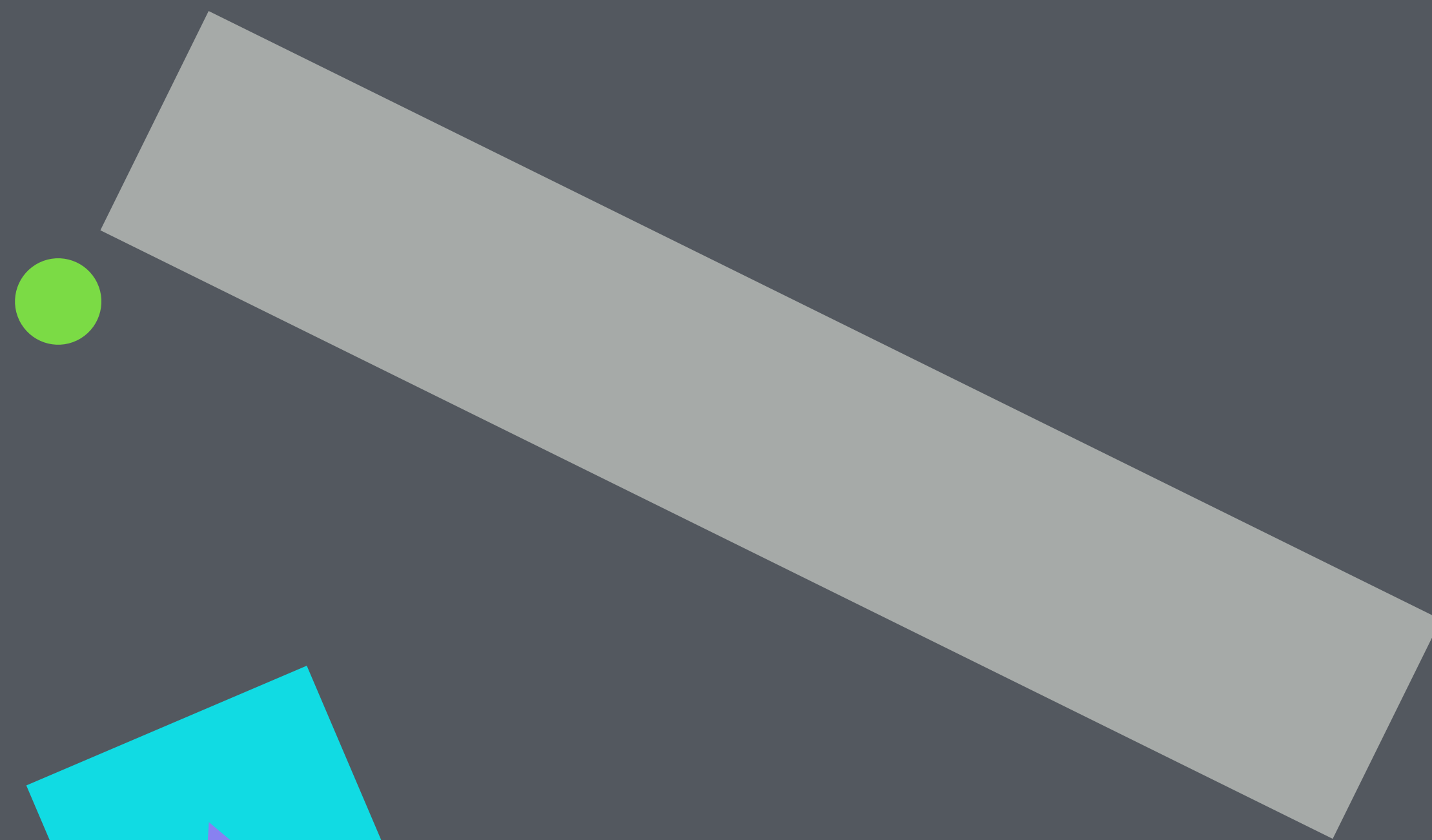




Changing sensor distance.

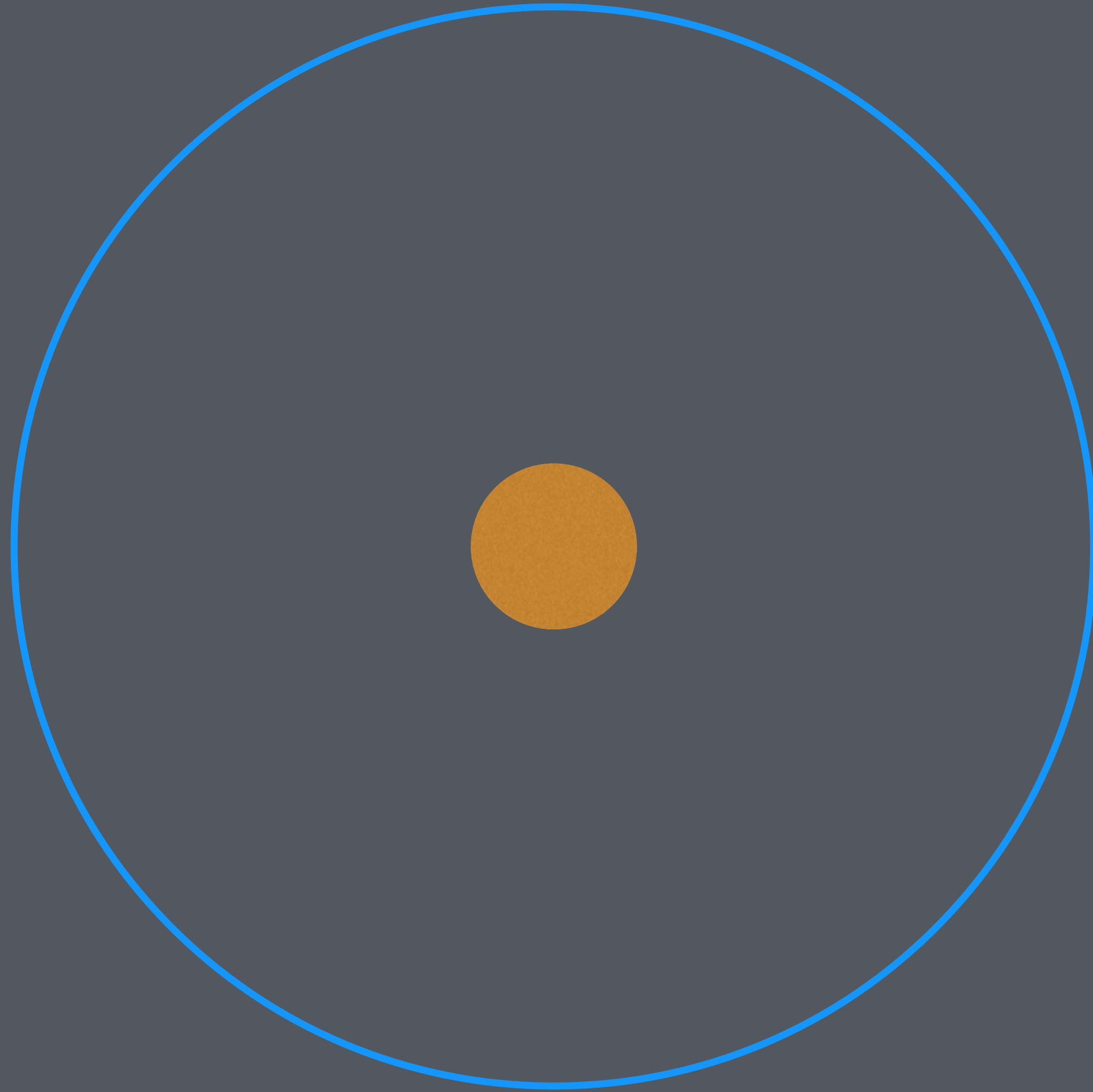


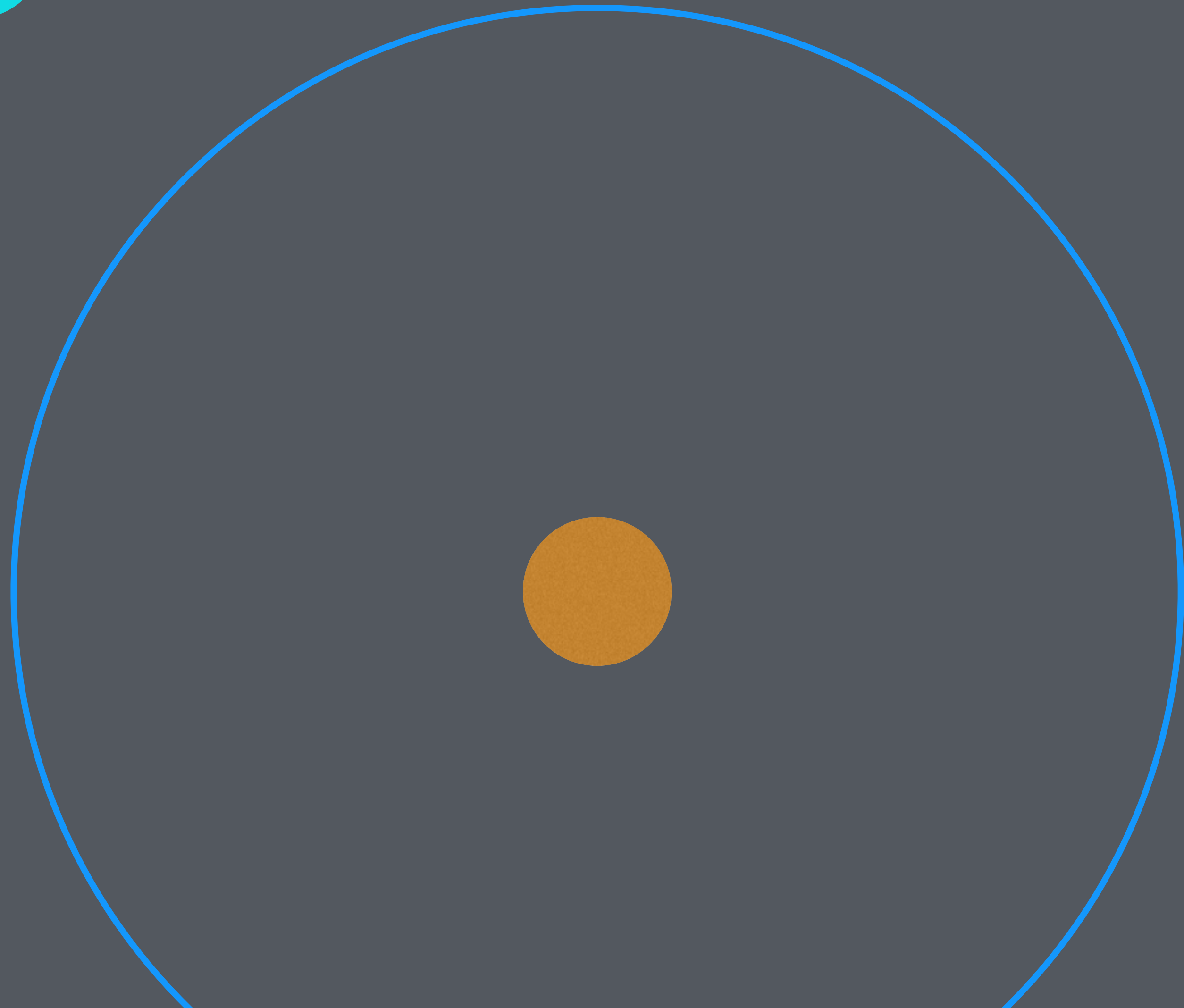




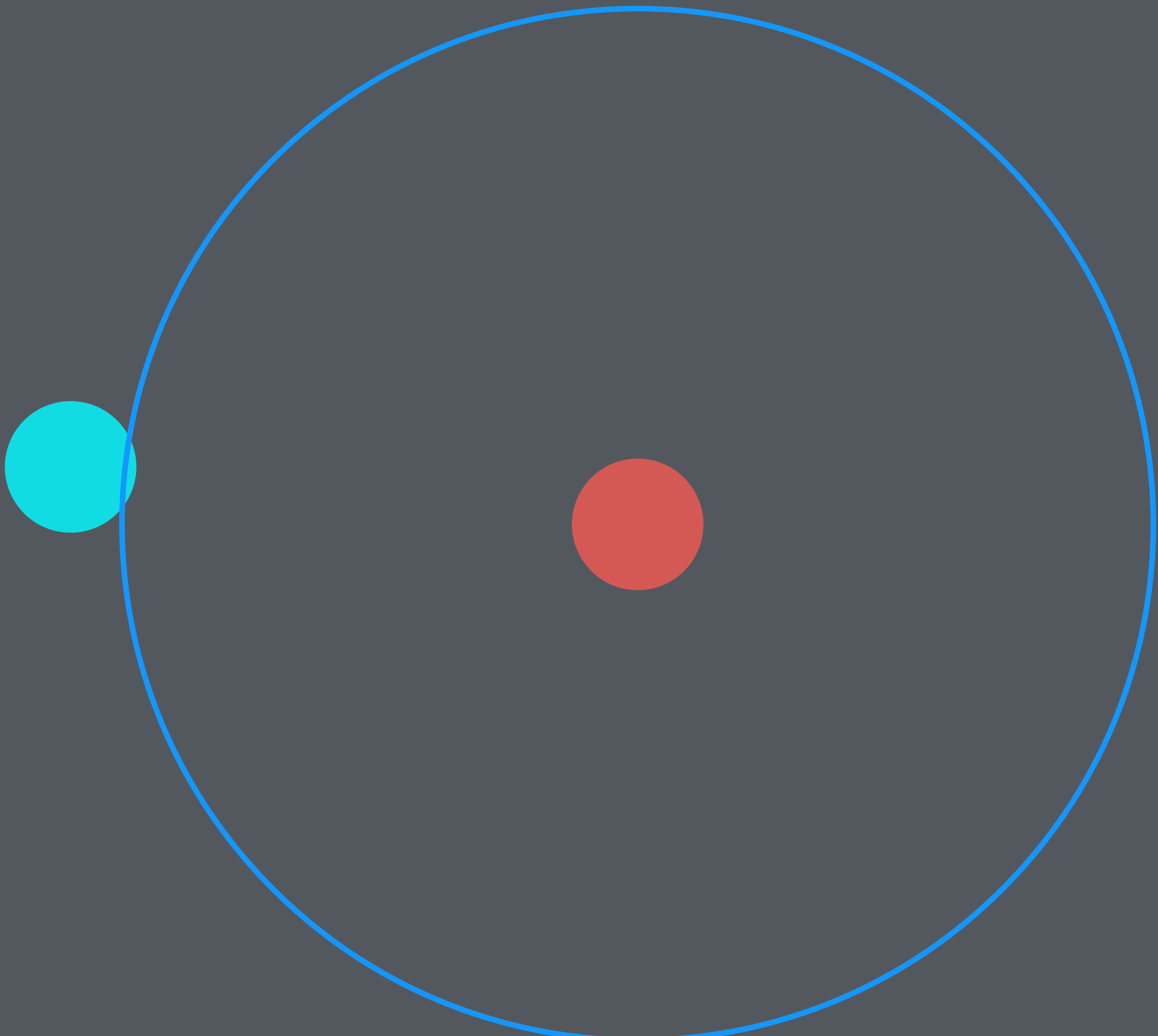
Simulating senses.

Sensor range.

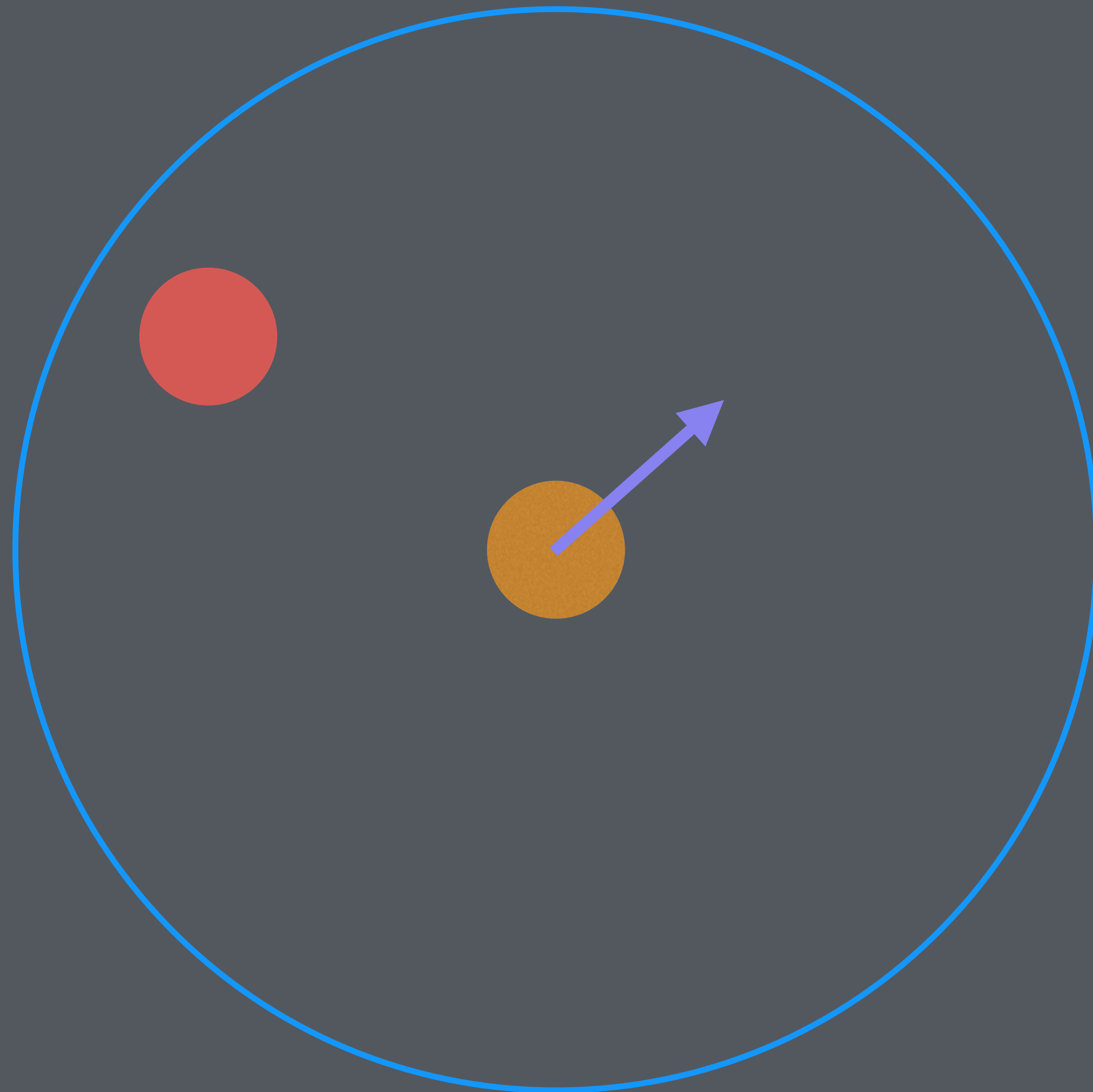


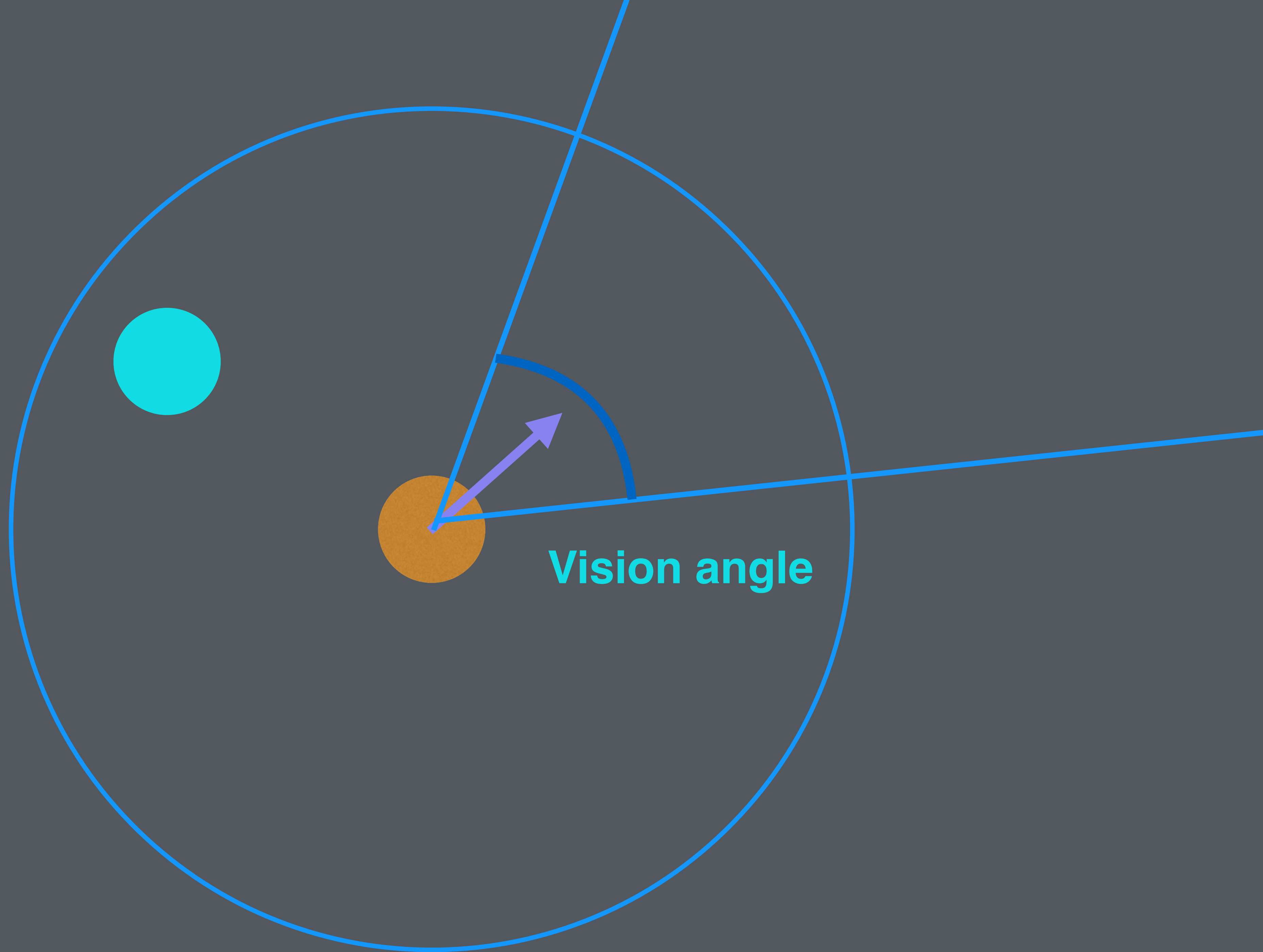






Vision cone.

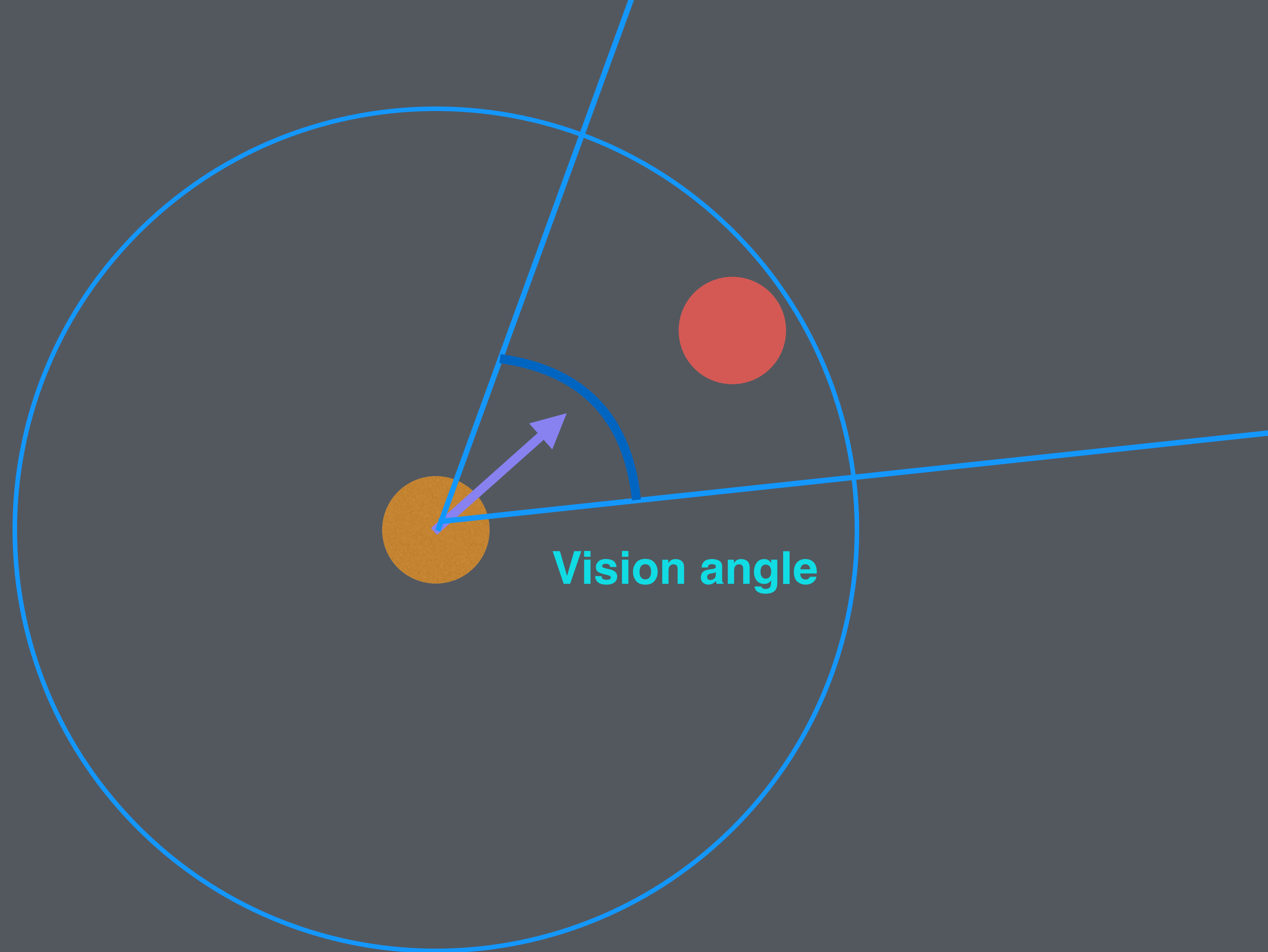


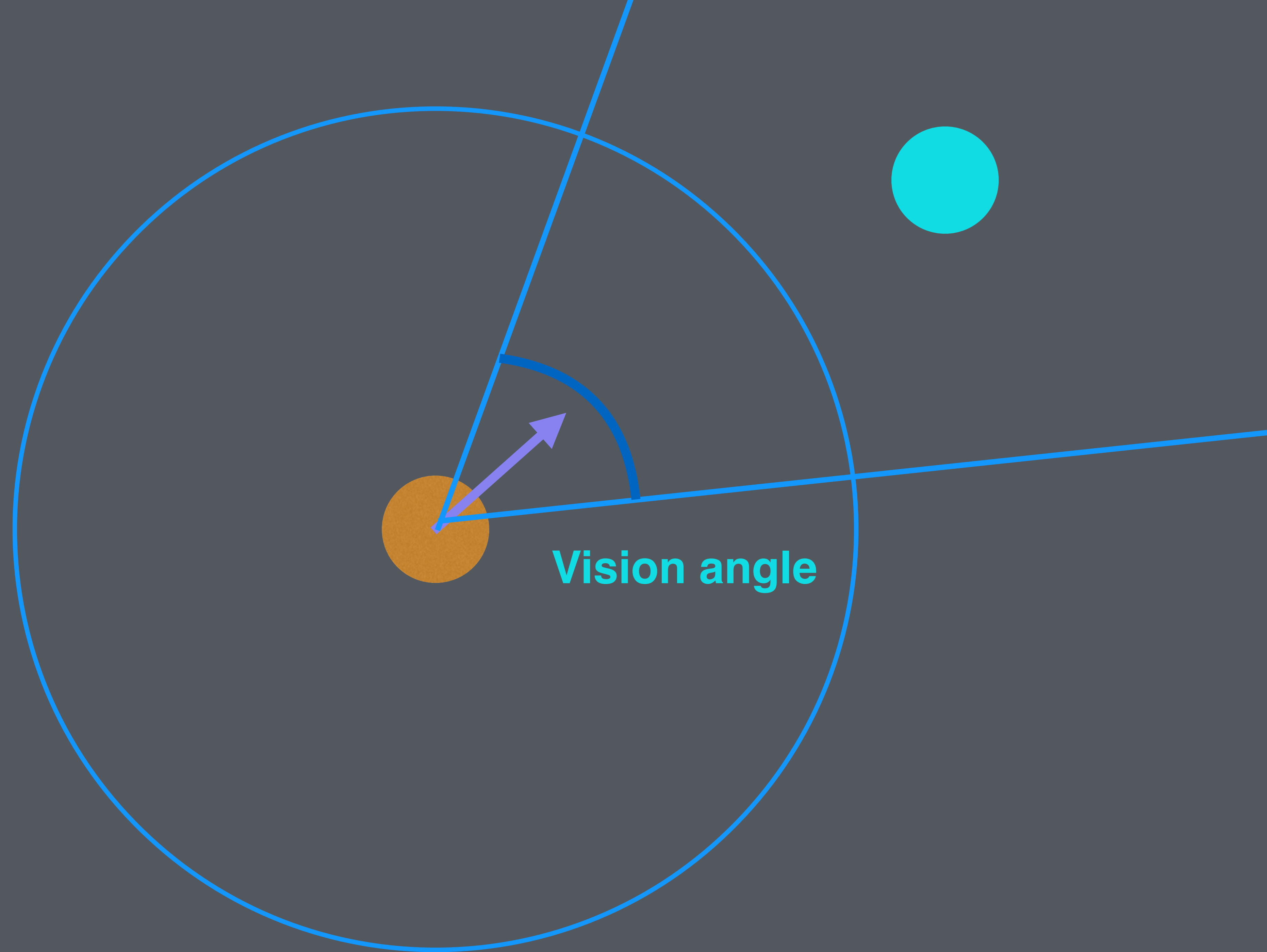




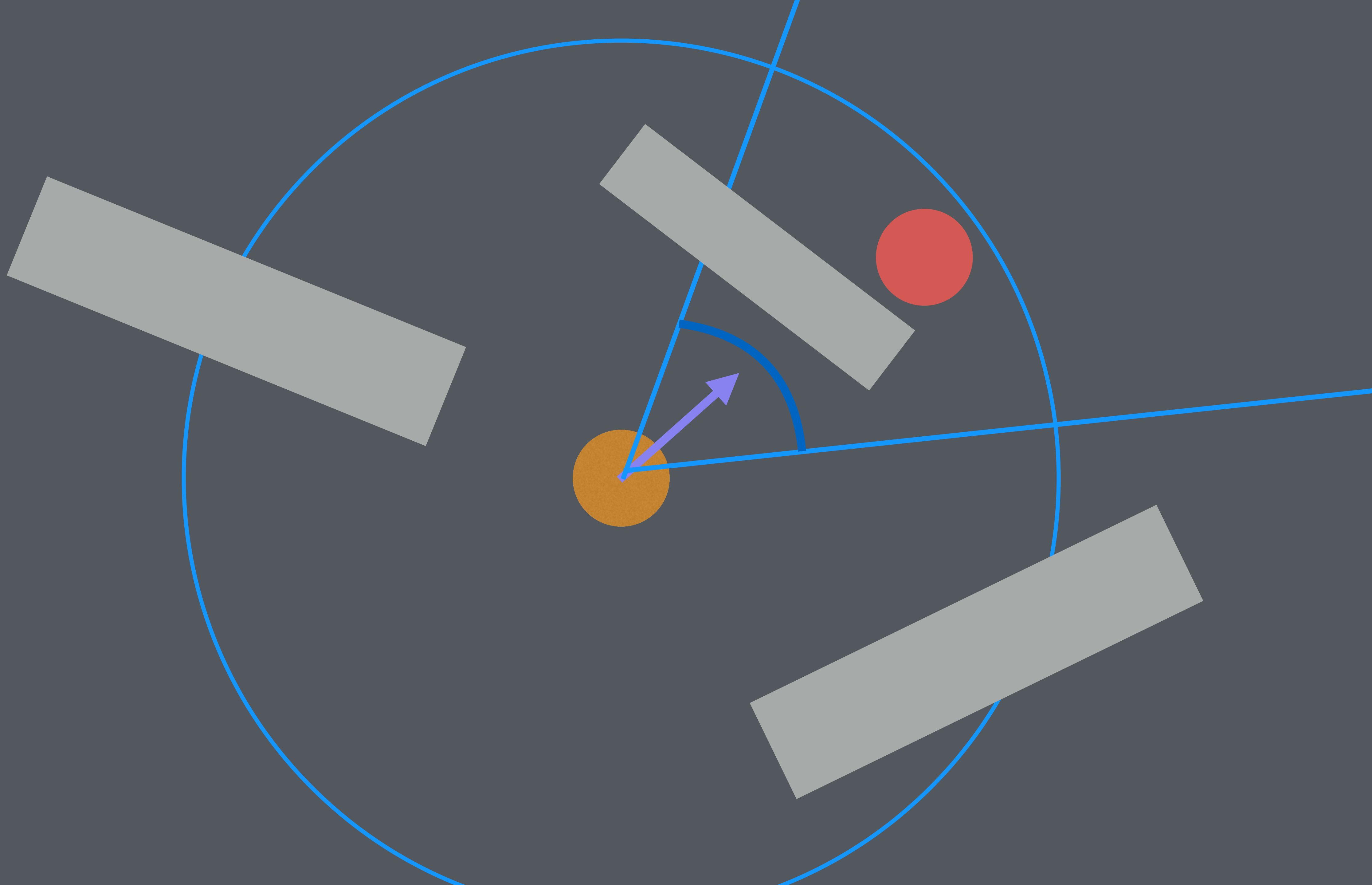
Calculating vision angle:

1. Calculate angle towards object. **$\text{atan2}(\text{normalizedDirectionToObject})$**
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



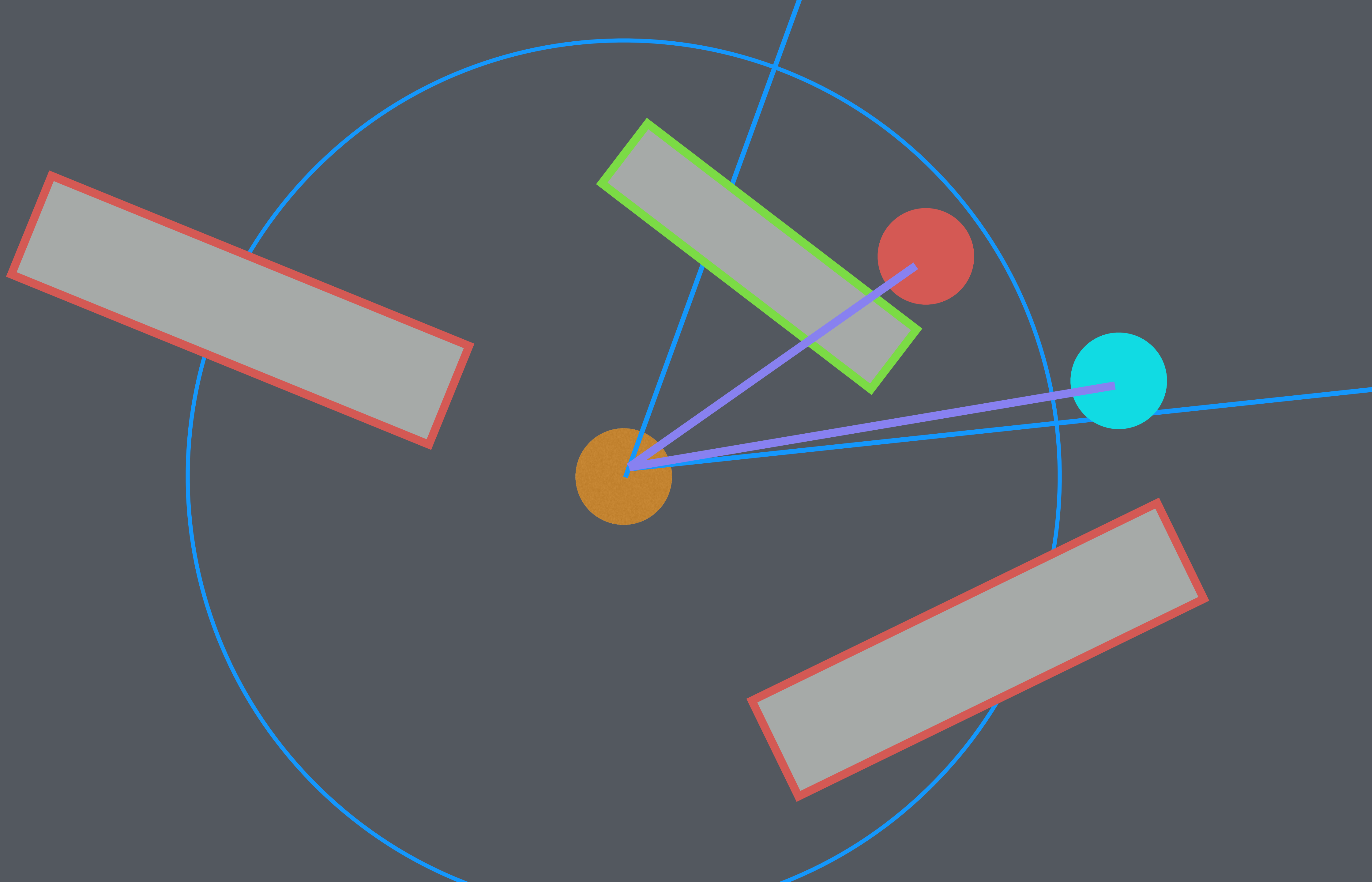


Vision obstacles.

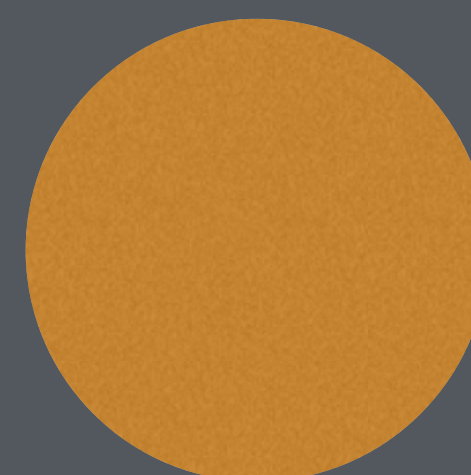
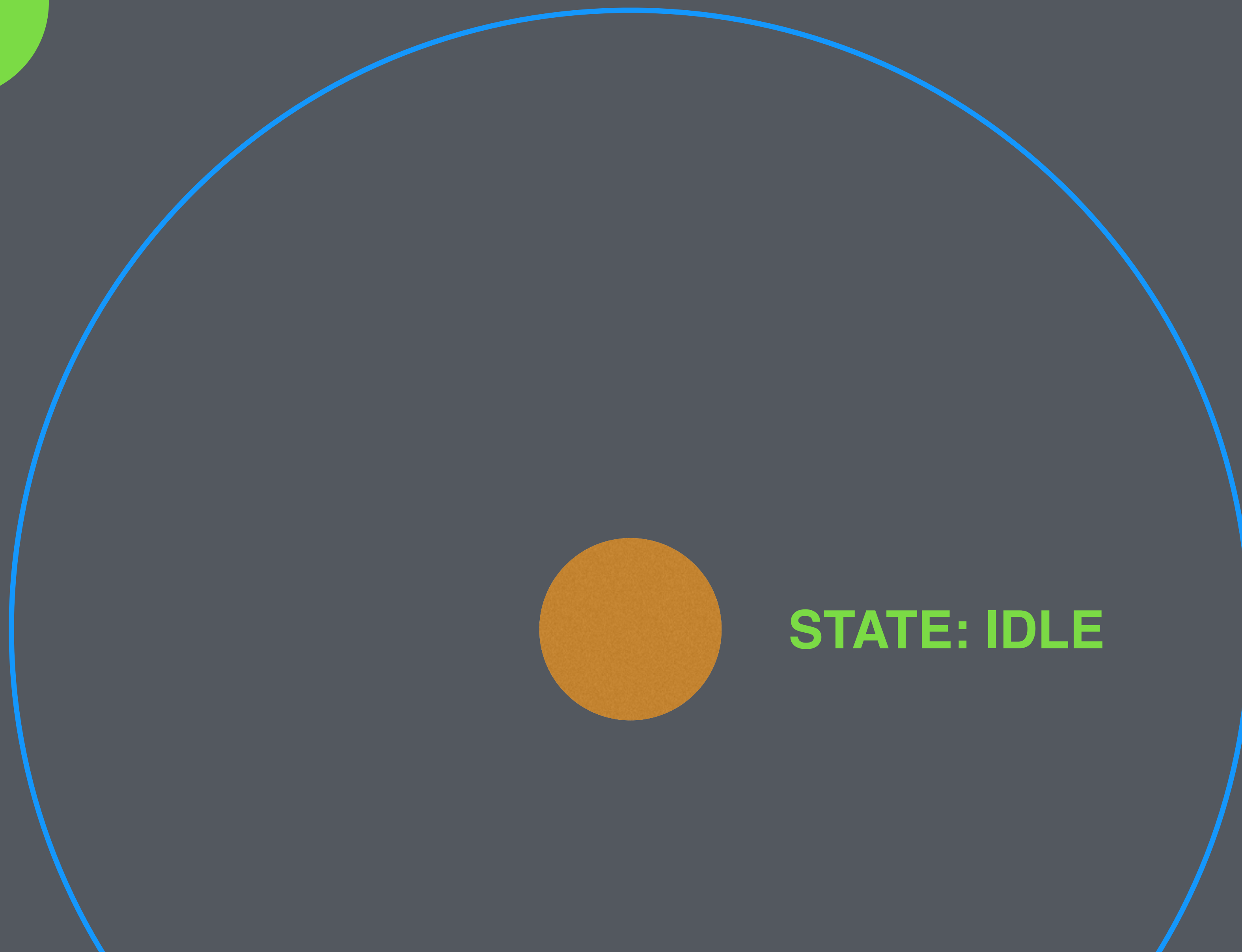


When seeing something, do a **ray test** from current position towards that object's position against all obstacles in the level.

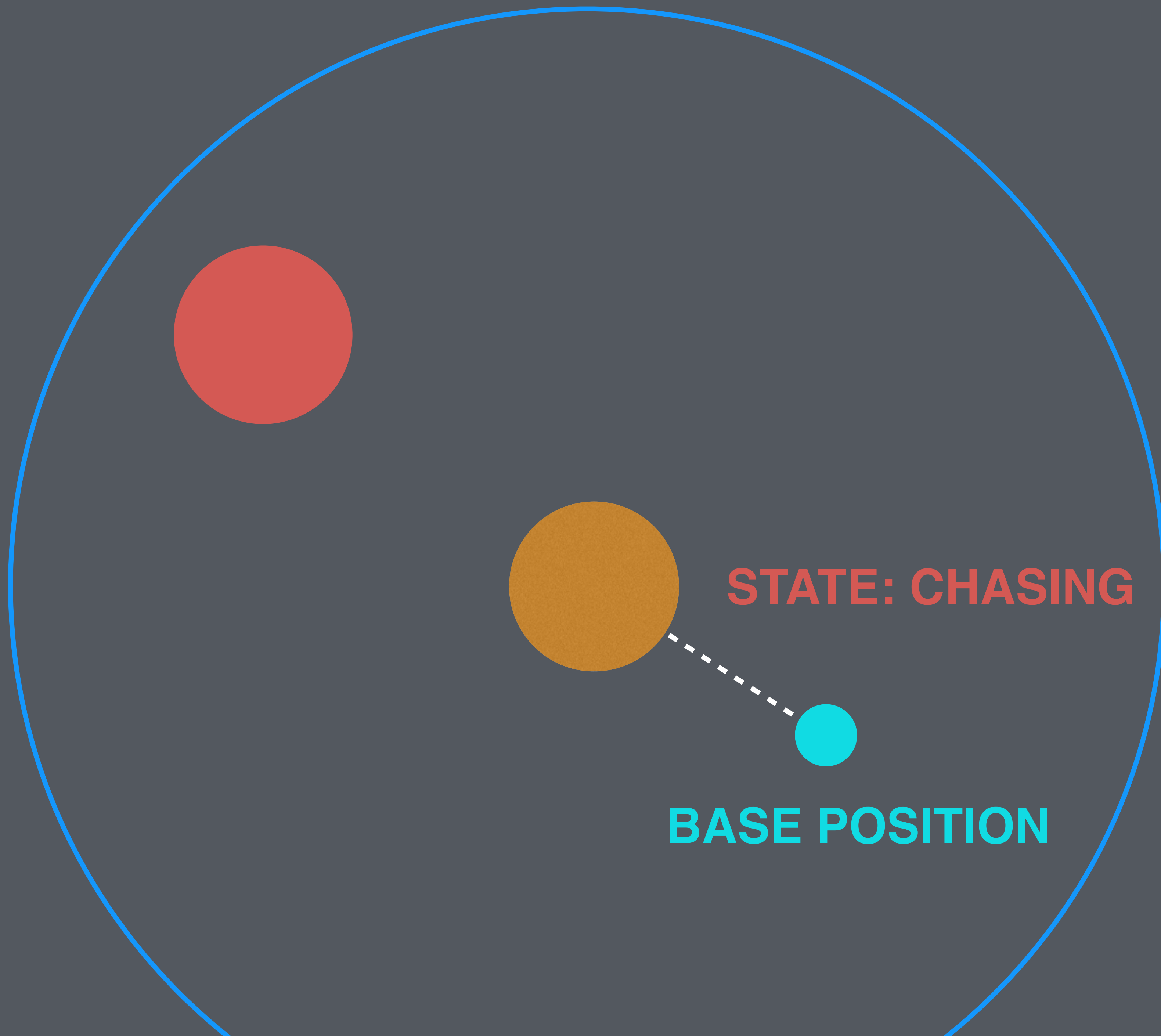
If it intersects an entity and the distance is closer than the object, then we can't see it.

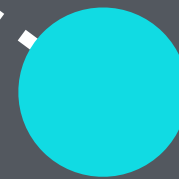
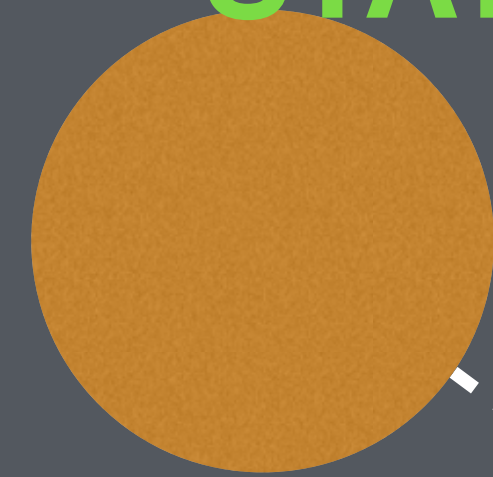
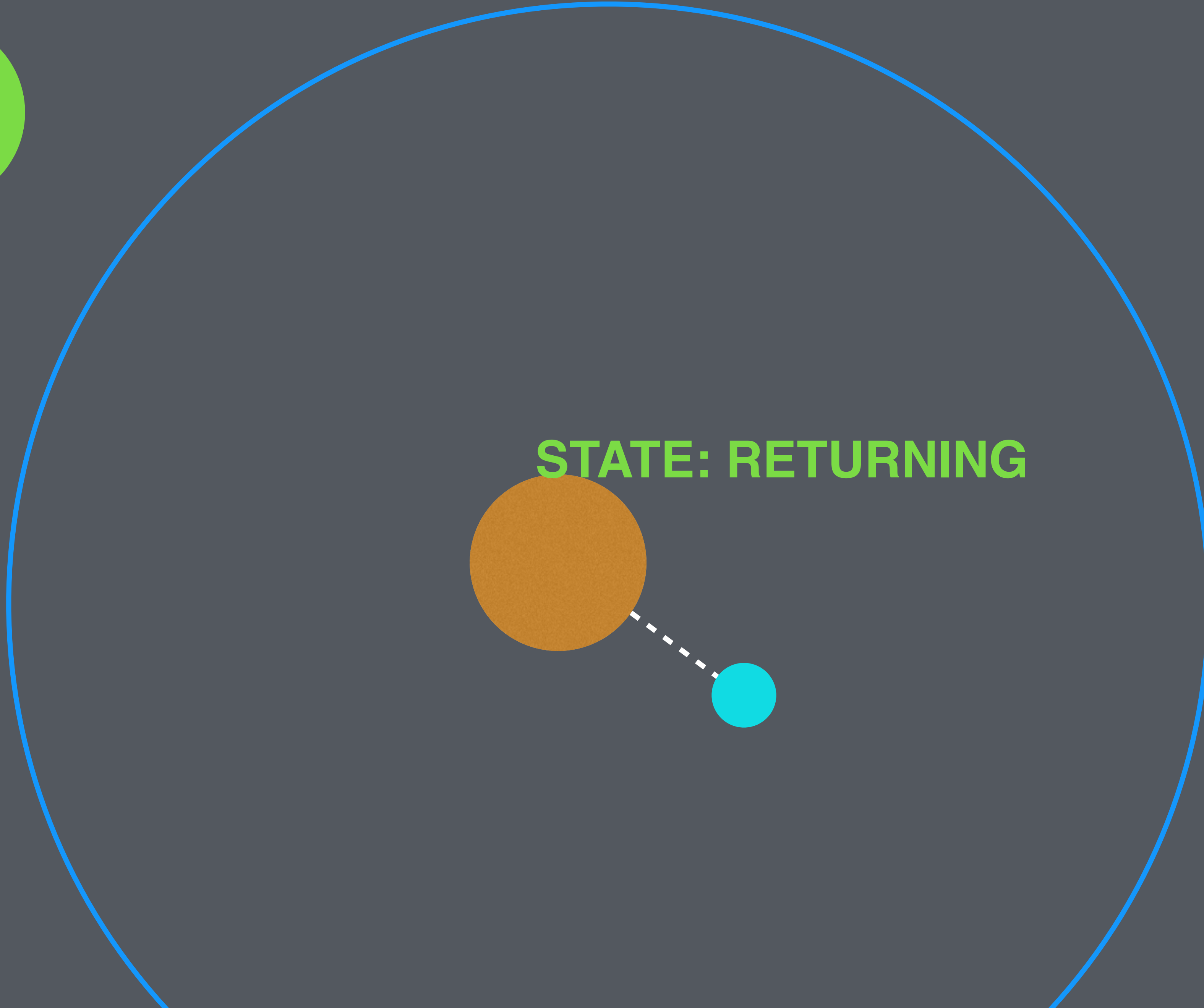


States



STATE: IDLE





STATE: RETURNING

Finite state machines.

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

IDLE

**CURRENT
STATE**

CHASING

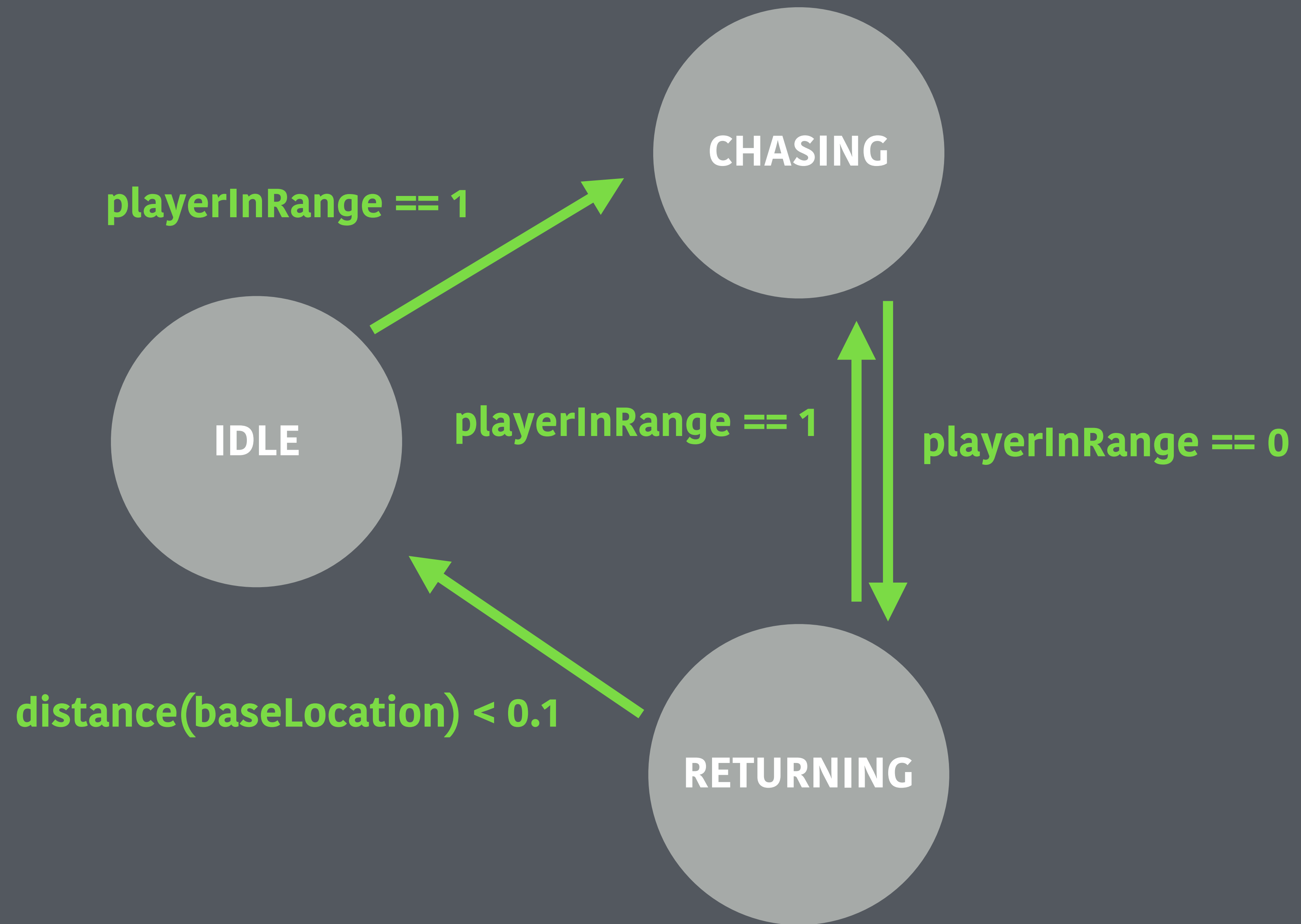
IDLE

**RETURNIN
G**

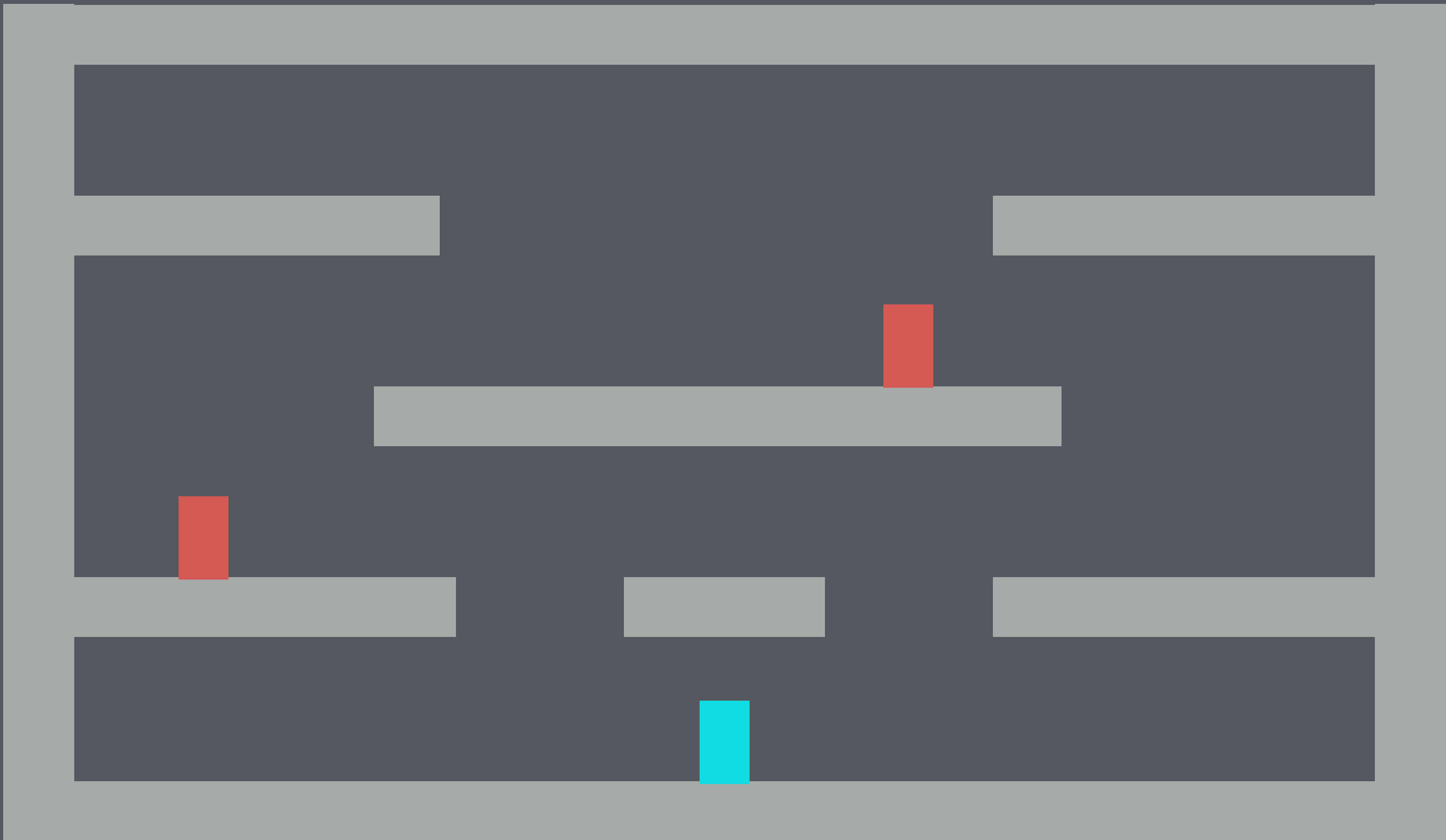
Update every frame based on
the **current state**.

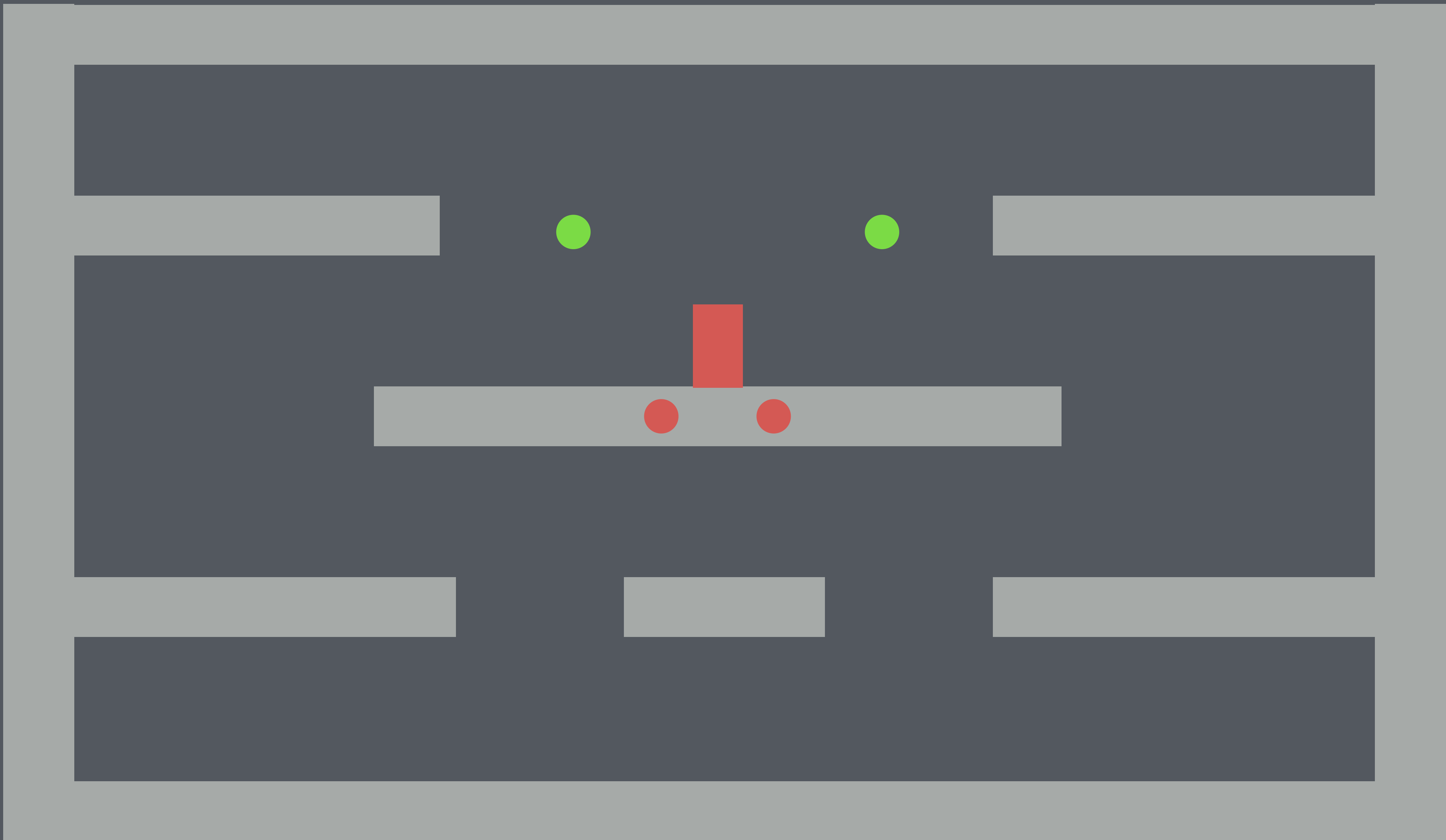
React to **sensing data** based on
the **current state**.

	PLAYER IN RANGE	PLAYER NOT IN RANGE	AT BASE POSITIO N	NOT AT BASE POSITIO N
IDLE	CHASING			
CHASING		RETURNING		
RETURNING	CHASING		IDLE	



Example:
Versus platformer.



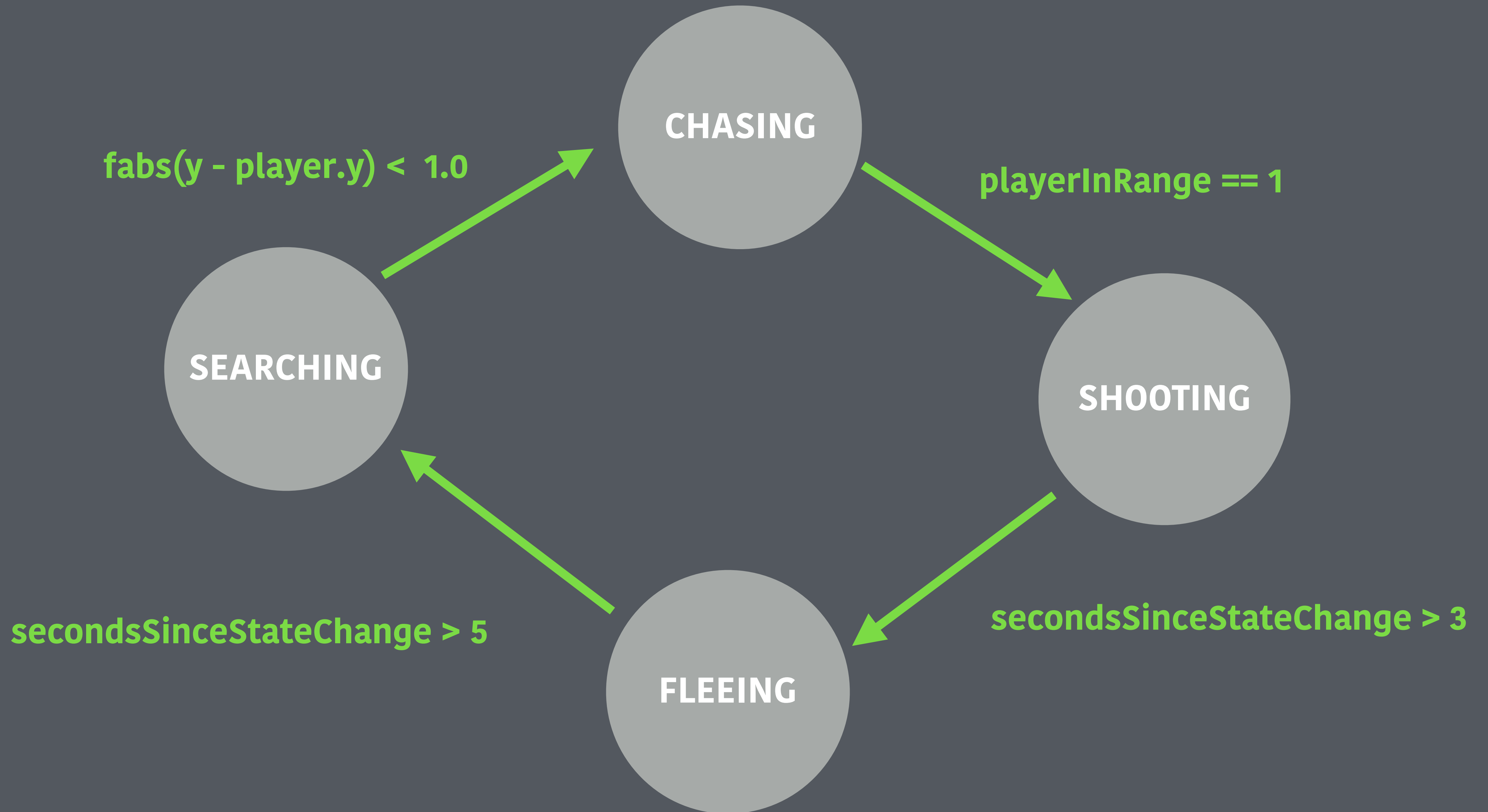


States

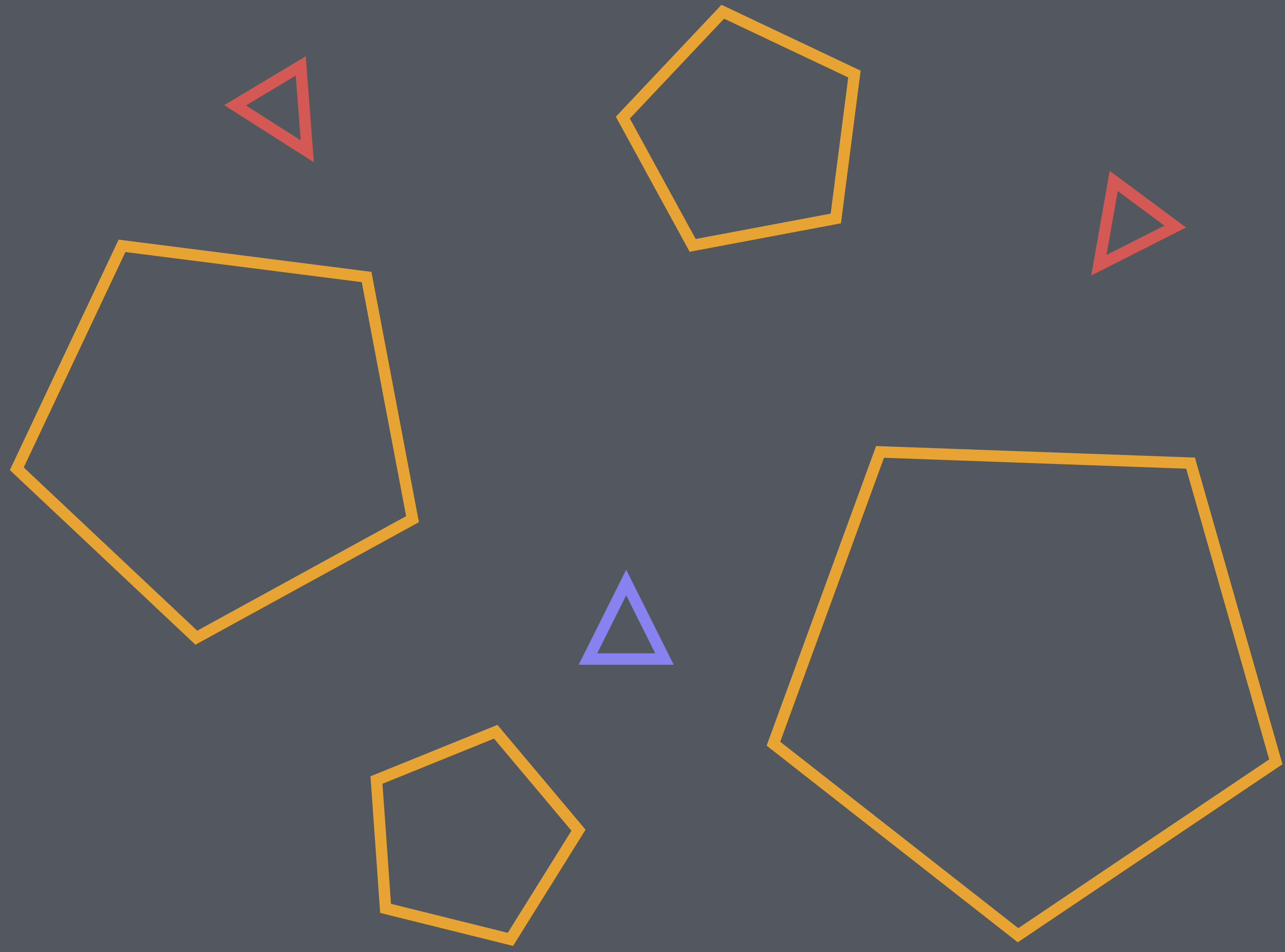
SEARCHING - Move forward, if edge sensor isn't colliding, turn around or drop down $X\%$ of the time. If jump sensor is colliding, jump $X\%$ of the time.

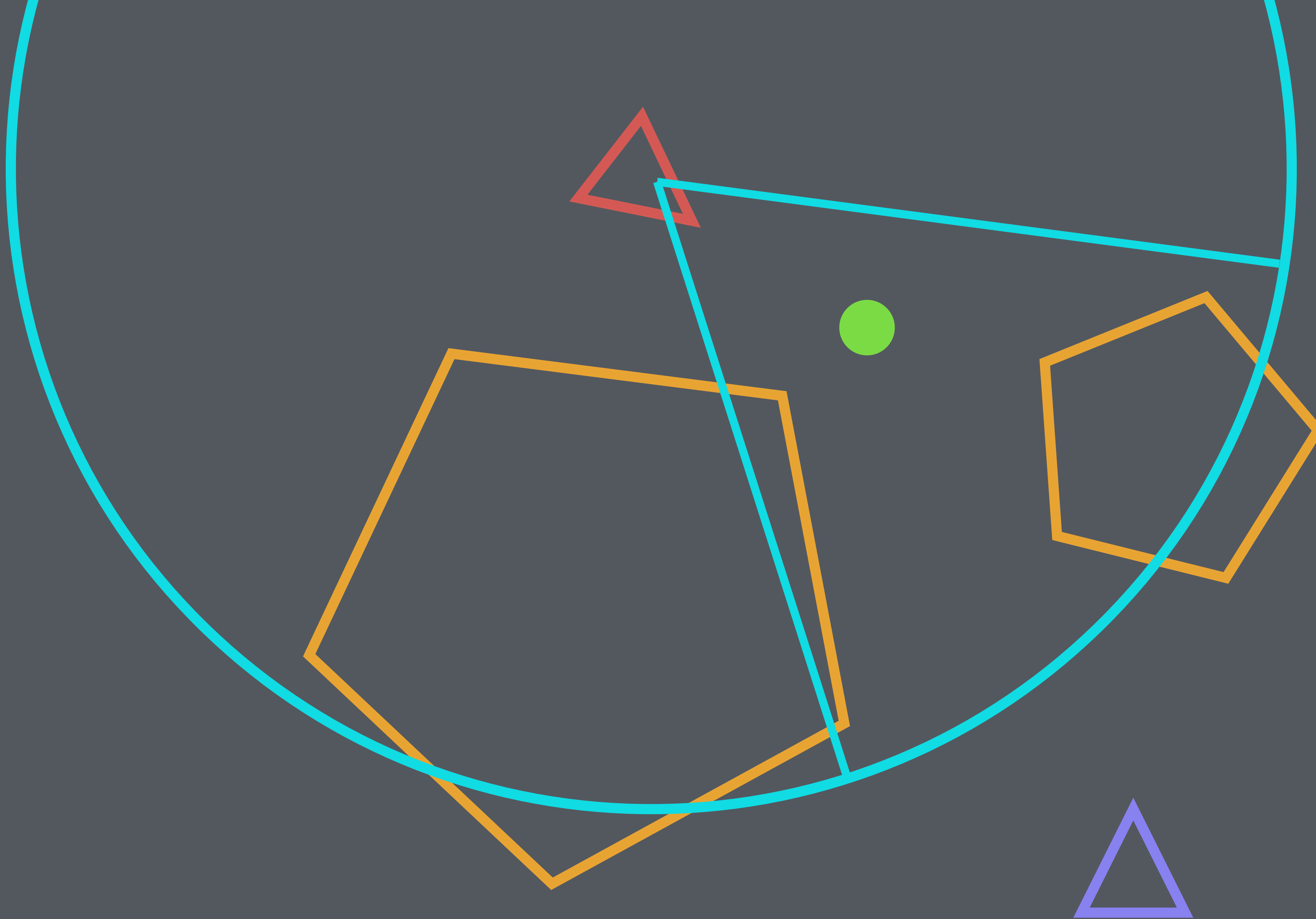
CHASING - Move towards player, if edge sensor isn't colliding, jump.

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.



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.

