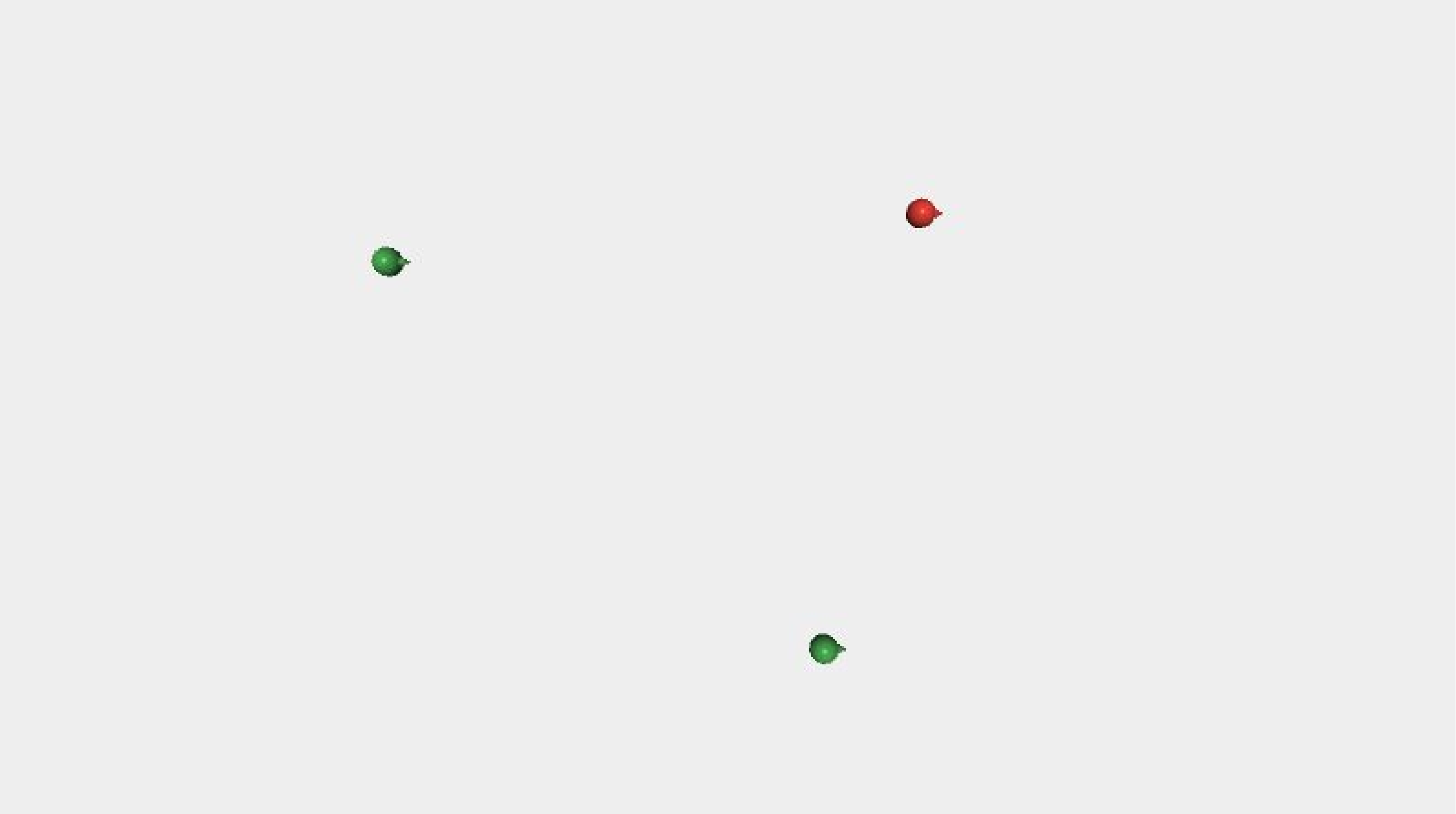
AI: Steering Behaviors

Ricard Pillosu - UPC

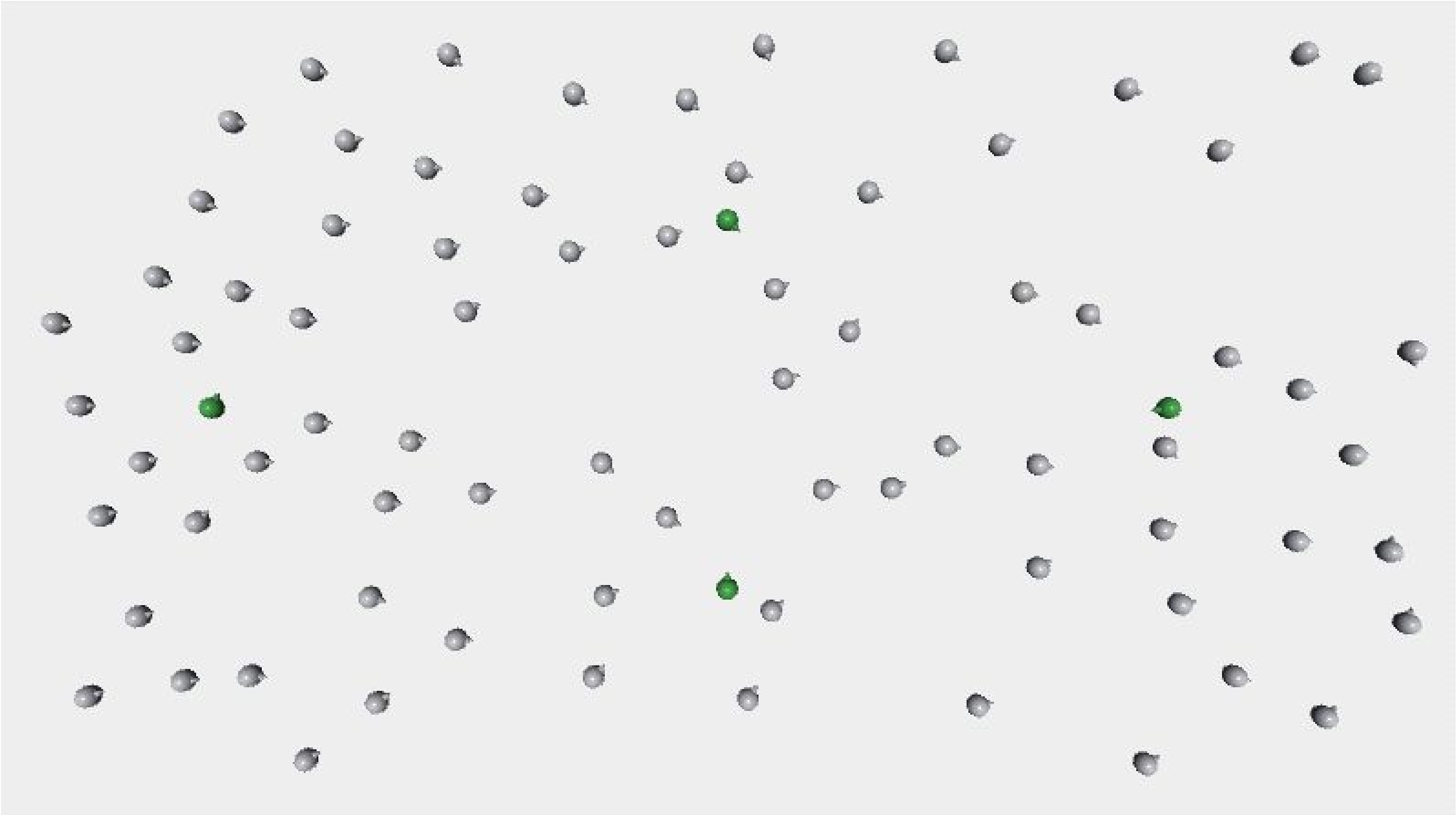
Steering Behaviors

- Introduced by Craig Reynold in 1999
- Main website with many links to resources
- The idea is to create simple atomic behaviours that could be combined
- It is used in several areas, but shines in crowd simulation and vehicles
- Used in many crowd simulation software (movies, crisis prevention, etc.)



Hide

Avoidance



Flocking

Kinematic vs. Steering

- “Flocking” or “Boids” is the most well known result of this family of algorithms
- Kinematic: simple behaviours that output the final velocity vector.
    - You only do position += velocity
    - Not very realistic
- (Dynamic) Steering: behaviours that output a desired acceleration.
    - Harder since we push things around, so we need to -accelerate to stop
- We always have a end point were we cap velocity to a maximum
- We actually have to cap *movement* and *rotation* velocity!
- Remember to always use **dt**

Some simple math

- We will use a lot of 2D logical representation (called 2 ½ D)
- 3D is still valuable to move things like the camera
- **Unity** have Y up, look at Z and have X to their right (left-hand)
- Angles can be scalar or unit vector:
  - orientation.x = -sin(orientation_scalar)
  - orientation.y = cos(orientation_scalar)
  - orientation_scalar = atan2(orientation.x, orientation.y)
- Scalar angles have a range of [-PI,+PI]

Some simple math

- We will use a lot of 2D logical representation (called 2 ½ D)
- 3D is still valuable to move things like the camera
- **Unity** have Y up, look at Z and have X to their right (left-hand)
- Angles can be scalar or unit vector:
  - orientation.x = -sin(orientation_scalar)
  - orientation.y = cos(orientation_scalar)
  - orientation_scalar = atan2(orientation.x, orientation.y)
- Scalar angles have a range of [-PI,+PI]

TODO 1

“Make sure mov_velocity is never bigger that max_mov_velocity”

- Just cap the vector so the magnitude never exceeds max_mov_velocity

TODO 2

“Rotate the arrow to point to mov_velocity direction. First find out the angle then create a Quaternion with that expressed that rotation and apply it to aim.transform”

- Check how to create a Quaternion to rotate around a vector

TODO 3

“Stretch it the arrow (arrow.Slider) to show how fast the tank is getting push in that direction. Adjust with some factor so the arrow is visible.”

- Arrow should be longer based on the final velocity magnitude
- Adjust by some factor for easy visibility (a factor of 4.0f looks good)

TODO 4

“Update tank position based on final mov_velocity and deltatime”

- The core of everything, where we actually change the position of the agent
- Check Time class from Unity

TODO 5

“Set movement velocity to max speed in the direction of the target”

- Seek should never reach its target
- It is the simplest of behaviors
- Remember you can read all public info from “move” property

TODO 6

“To create flee, just switch the direction to go”

- Mostly a copy&paste from seek

TODO 6

“To create flee, just switch the direction to go”

- Mostly a copy&paste from seek

TODO 7

“Rotate the whole tank to look in the movement direction extremely similar to second TODO”

- We are rotating the tank instantly so it always faces movement direction
- Not very smooth but works good enough

TODO 8

"Calculate the distance. If we are in min_distance radius, we stop moving. Otherwise divide the result by time_to_target (0.25 feels good) Then call move.SetMovementVelocity()"

- We are rotating the tank instantly so it always faces movement direction
- Not very smooth but works good enough
- Try changing the time_to_target value in real time to understand better the output

TODO 9

“Generate a velocity vector in a random rotation (use RandomBinominal) and some attenuation factor.”

- Try changing the factor some number during execution