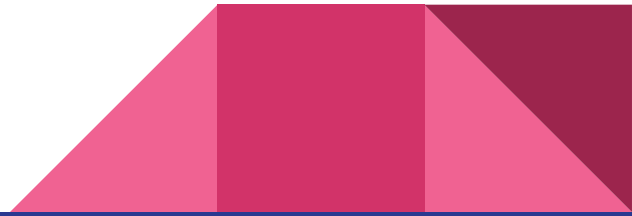
AI: Steering Behaviors 2

Ricard Pillosu - UPC

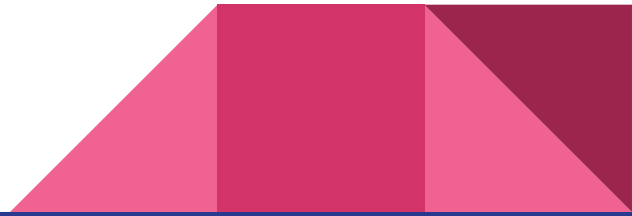
Variable matching

- Steering behaviours are basically about matching:
  - Rotation
  - Position
  - Velocity
- We will create small behaviours and later use techniques for mixing them
- Most behaviours have their opposite (seek vs. flee)
- Then we move to delegated behaviours like:
  - Pursue / Evade
  - Path Following, Obstacle Avoidance



New code

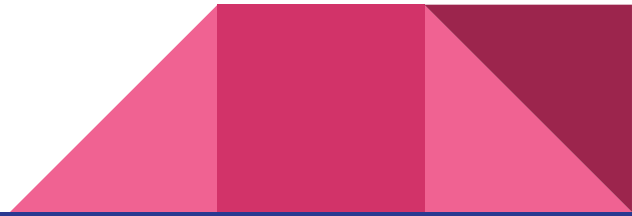
- **Move.cs** now supports:
  - Adding acceleration
  - Adding angular velocity
  - Adding angular acceleration
  - Clamp angular velocity
  - Rotation



TODO 1

SteeringSeek.cs: *“Accelerate towards our target at max_acceleration”*

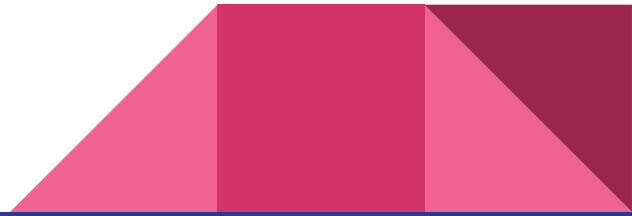
- Very similar to kinematic seek but with acceleration



TODO 2

SteeringFlee.cs: *“Same as Steering seek but opposite direction”*

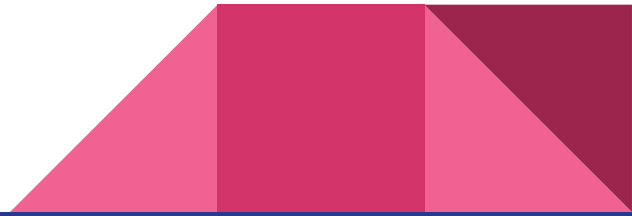
- No brainer, just copy&paste



TODO 3

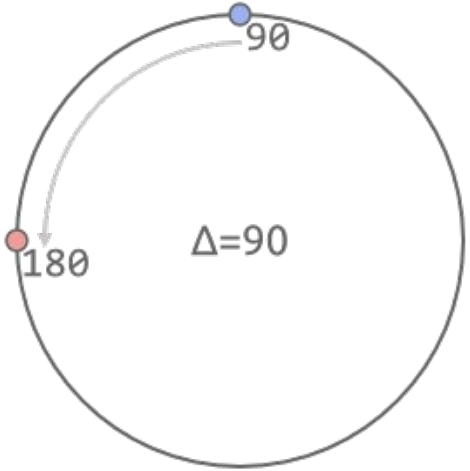
SteeringArrive.cs: *“Create a vector to calculate our ideal velocity then calculate the acceleration needed to match that velocity before sending it to move.AccelerateMovement() clamp it to move.max_mov_acceleration”*

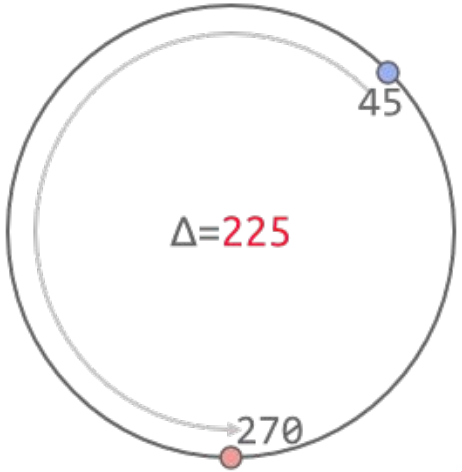
- Note new variable: slow_distance
- If inside that distance adjust your ideal velocity linearly
- Check solution.exe for expected result



Some more math

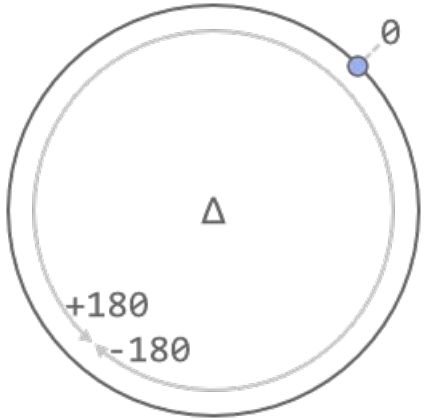
- For angular rotation we need to calculate difference between angles
- Left it works without problem, on the right not so much ...





Angular Rotation (more info here)

- We can only allow solutions between 180 to -180
- Any other values need to be wrapped around PI
- Unity handles this in some methods from Mathf:
  - Mathf.DeltaAngle()
  - Mathf.LerpAngle()
  - Mathf.MoveTowardsAngle()
- My take in C++:




```
// wrap diff around [-pi, pi]
diff += PI;
diff -= floorf( diff * INV_TWO_PI ) * TWO_PI;
diff -= PI;
```

TODO 4

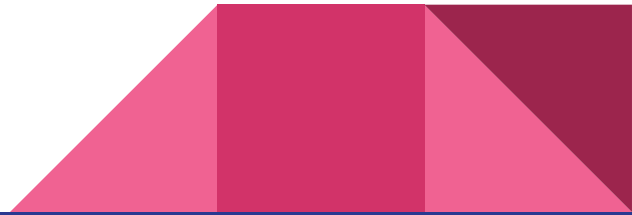
SteeringAlign.cs: *“As with arrive, we first construct our ideal rotation then accelerate to it. Use Mathf.DeltaAngle() to wrap around PI. Is the same as arrive but with angular velocities”*

- Focus on your ideal angular rotation
- Once you have it, accelerate to it
- In the end remember to Mathf.Clamp() it to move.max_rot_acceleration

TODO 5

SteeringVelocityMatching.cs: *“ First come up with your ideal velocity then accelerate to it”*

- Our target must be another tank
- We access directly it’s movement script to read their movement velocity
- Now just change your velocity magnitude to match target’s
- You only move toward +Z
- Only useful in combination with others



TODO 6

SteeringPursue.cs: *“Create a fake position to represent enemies predicted movement. Then call Steer() on our Steering Arrive”*

- We are actually executing arrive.Steer() but with a new position
- We should have a guard of a max_prediction time
- Then calculate a *naive* prediction for better Pursue

Homework

- Create Evade: opposite of pursue (still predicting enemy position)
- Create a Steering Wander: “Seek” to a random point in a circle in front of the agent: