Steering Behaviours – Part 2

Pursue, Evade, Arrive and Avoid

Steering Behaviours Recap

- Steering Behaviour are a way to add locomotion of autonomous agents
  - They calculate a force to apply to an agent's velocity to steer them in a certain direction

- Previously we've looked at Seek, Flee and Wander behaviours
  - We will now look at Pursue, Evade, Arrive, Avoidance and ways to combine behaviours

Pursue Behaviour

- Instinctively, when pursuing a moving target, animals and humans tend to not head directly at the location of the target
  - They take in to account the target's movements and head to where they predict the target will be by the time they arrive at the target
  - This is Pursuit behaviour

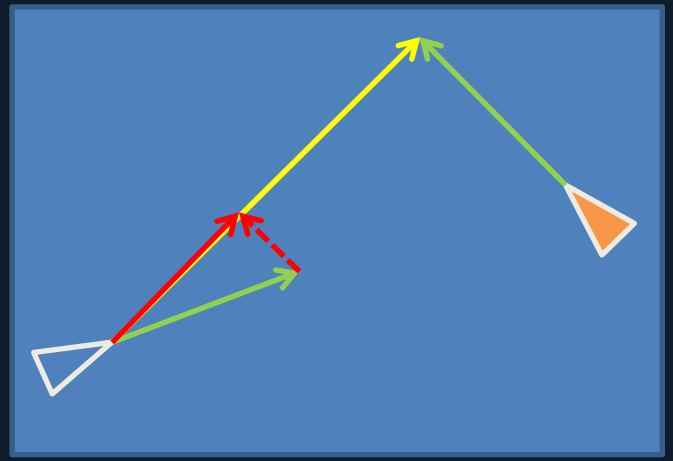
- Pursuit is near identical to Seek
  - We simply include the target's velocity with its position to calculate a Seek target

Pursue Behaviour

- For example:

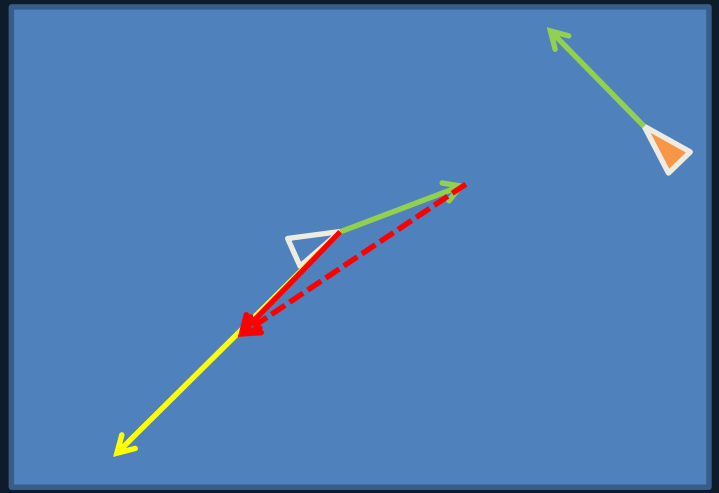
  V = Target's Position + Target's Velocity – Agent's Position

  Force = normalise(V) x maxVelocity – Agent's Velocity



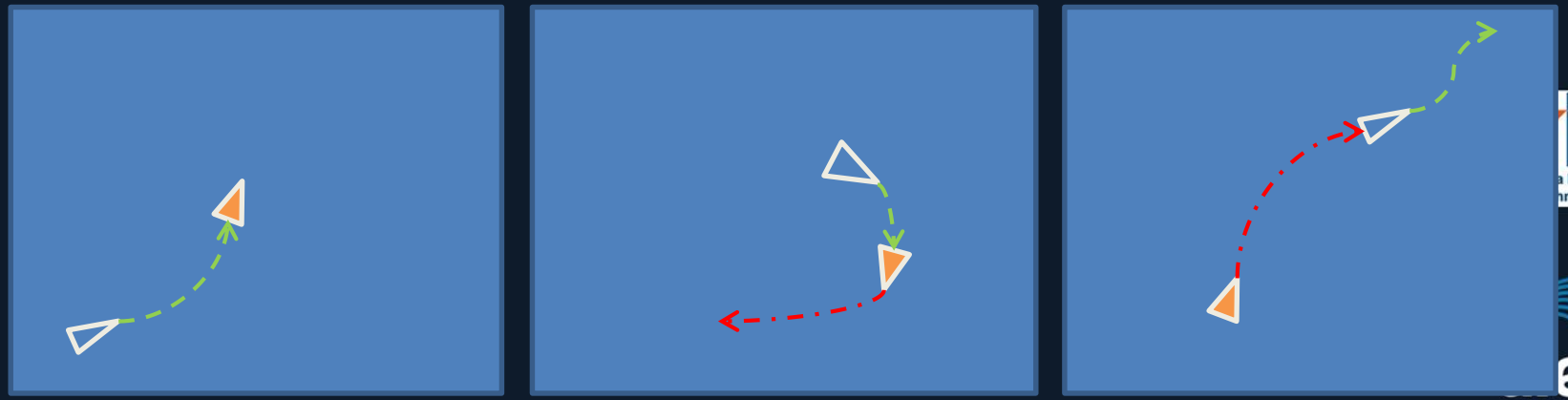
Evade Behaviour

- Much like Flee is the opposite of Seek, Evade is the opposite of Pursue
  - We Flee from the location that the target is heading towards, rather than Seek to it as we would with Pursue



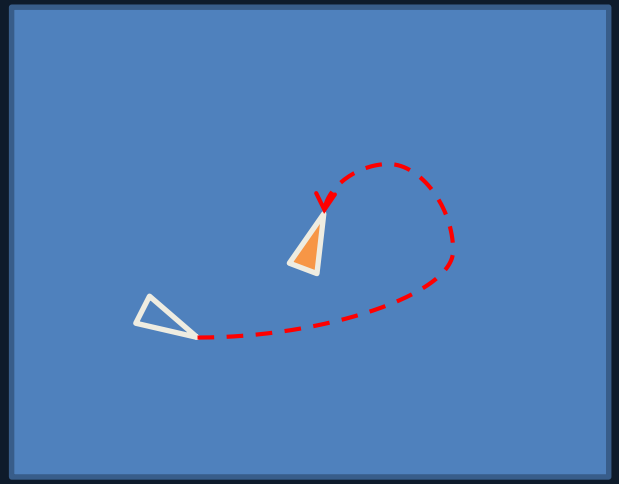
Evade Behaviour

- Evasion and Pursuit can be used on two different agents to create a simple game of tag
  - The pursuer could have a faster speed than the evader
  - When the pursuer catches the evader they switch behaviours, with the pursuer waiting a couple seconds before pursuing



Arrival Behaviour

- There is an issue with the current behaviours
  - Traveling at maximum velocity means the agent may overshoot its target

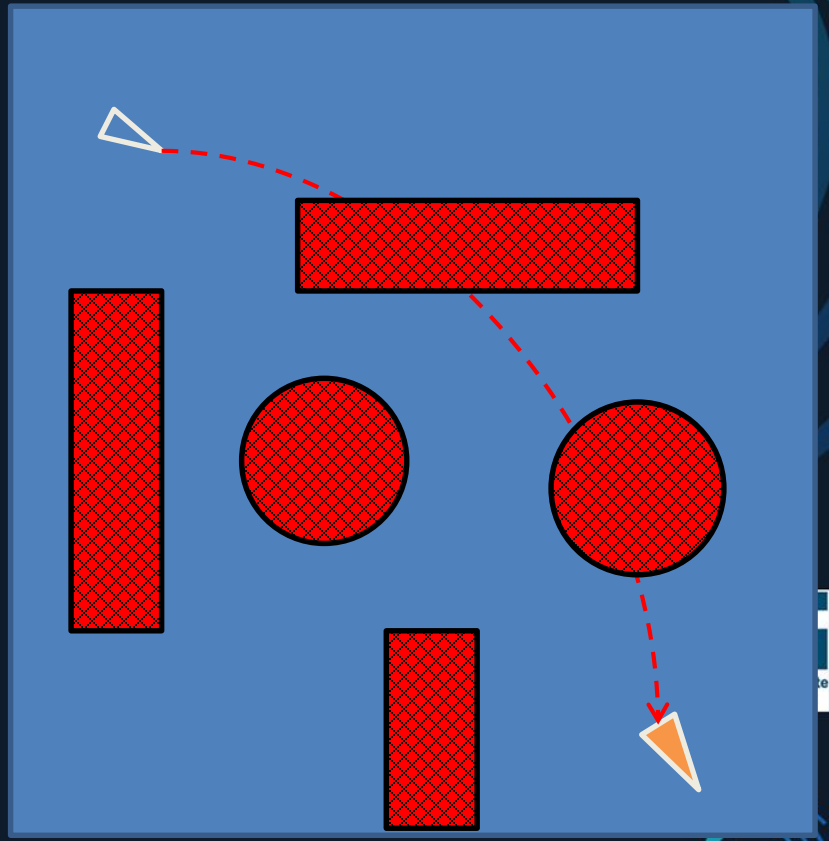


Arrival Behaviour

- A way to fix this is by making a slight change to the Seek behaviour
  - This new behaviour is called Arrival
- As the agent approaches its target its velocity is scaled down
  - This allows the agent to turn sharper and not overshoot
  - A radius around the target is used to scale down the agent's velocity
    - ***velocity*** **= min(** ***distanceToTarget*** **/** ***radius*****,** ***maxVelocity*** **)**

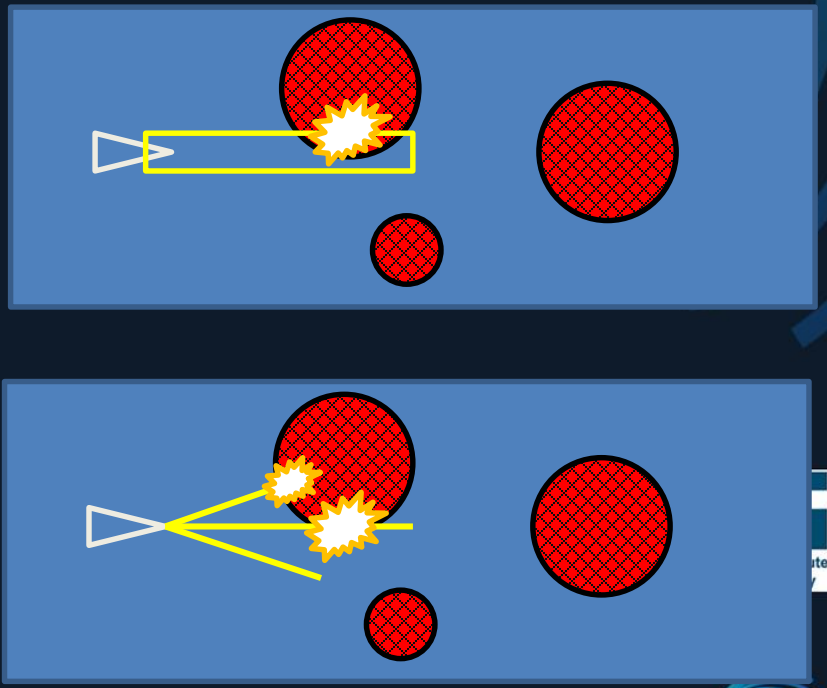
Avoiding Obstacles

- So far all of our behaviours have dealt with single targets
  - Seek towards a target
  - Flee from a target
- None of these behaviours take the surrounding environment in to account
  - The seeking agent may walk straight in to a wall or off a cliff



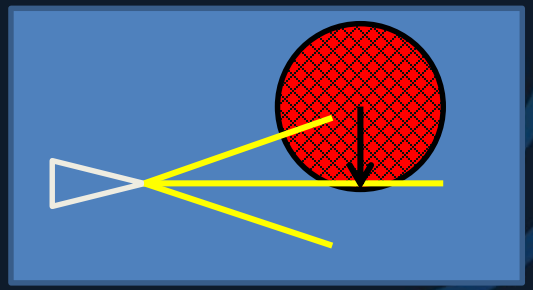
Obstacle Avoidance Behaviour

- Obstacle Avoidance is a behaviour that detects oncoming collisions and applies a steering force away from the object
  - Detecting the collisions usually consists of a box-vs-shape test, or multiple ray tests against surrounding obstacles
  - This behaviour is typically combined with other behaviours, which we will talk about shortly

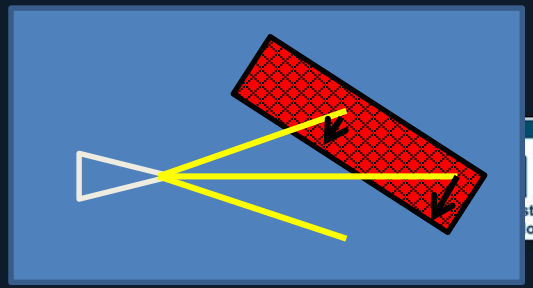


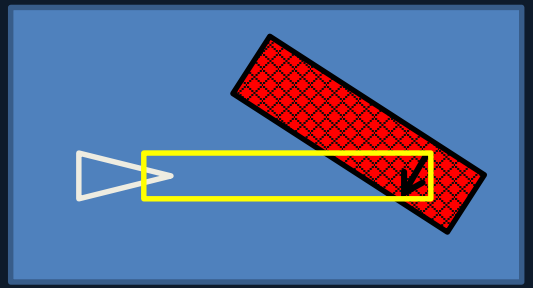
Obstacle Avoidance Behaviour

- In the case of the Ray version:
  - Multiple rays are cast in front of the agent at various angles
  - If a ray intersects with an object then a force away from the object is calculated and applied to the agent



- For the Box version:
  - Much like the ray version, if an intersection occurs then a force away from the object is calculated





Combining Behaviours

- So far we have only discussed 1 steering behaviour being active at a time for an agent
  - Flee or Seek, Wander or Pursue, etc

- Behaviours can be combined together by combining the force that they each calculate
  - Combining the forces can be tricky

Combining Behaviours

- The reason combining forces can be tricky is:
  - If we simply sum the forces together then it may exceed the agent's maximum velocity

- A common technique for combatting this is called **Weighted Truncated Running Sum with Priority**

Weighted Truncated Running Sum with Priority

- This technique has three aspects to it
  - Each steering behaviour calculates its force and multiplies it by a weighted value specific to that behaviour
  - Each force is then added together, in a set priority
    - Flee before Seek, then Avoid before Pursue, etc
  - After each force is added we check if the velocity has reached the maximum or passed it
    - If true then we truncate / clamp the velocity to the maximum and don't add any more steering forces

```
Force = 0
Behaviours : Prioritised

for each Behaviour B in Behaviours
    Force += B.Calculate() * B.Weight

    if Force >= MaxVelocity
        Force = MaxVelocity
        break
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Summary

- There are many different steering behaviours
  - We have not covered all of them, just the common behaviours
- Combining behaviours can be difficult
  - Care must be taken to tweak weights and select an appropriate priority