GAME AI

with Jeff Wilson, PhD

Steering Behaviors

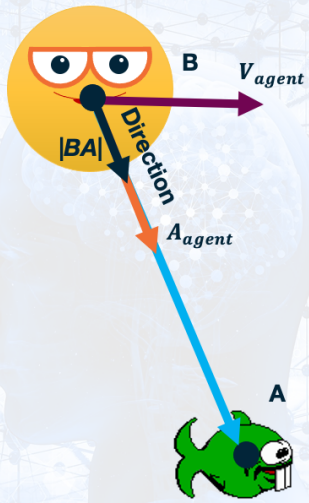
Introduction and Basics

Improve on Kinematic Movement

- With kinematic movement, velocity can change instantly
- Agent movement can look more natural with acceleration

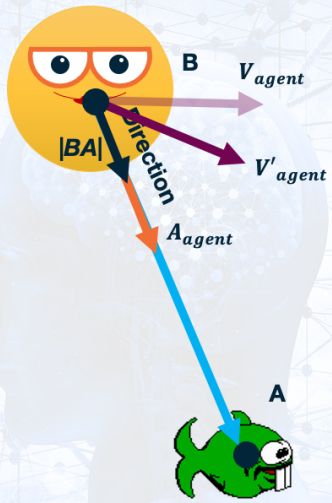
Acceleration

- With kinematic movement, velocity can change instantly
- Agent movement can look more natural with acceleration
- This can be accomplished by generating a steering acceleration/force according to the desired direction
- Velocity is maintained as kinematic state (along with statics) and is updated according to the steering force calculated each frame



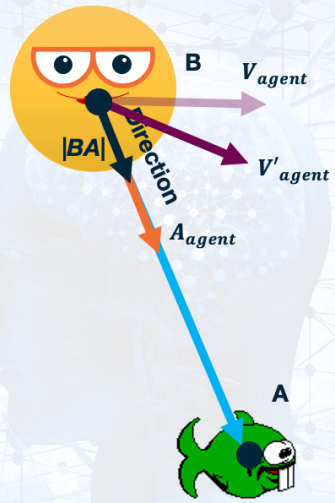
Acceleration

- For any calculated acceleration, the current agent velocity is updated
- This update likely won't orient the agent with the desired direction in a single frame; it will take multiple frames



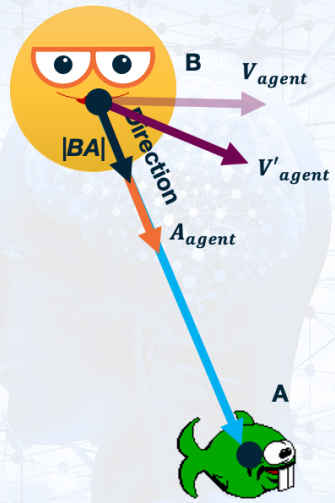
Acceleration

- Unbounded acceleration can lead to extreme velocities
- Therefore, when kinematics are updated a speed limit is usually imposed



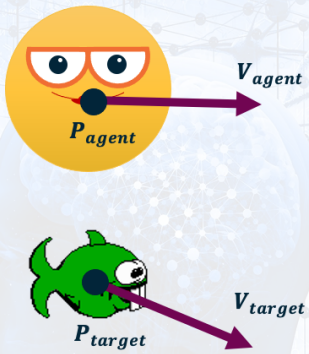
Acceleration

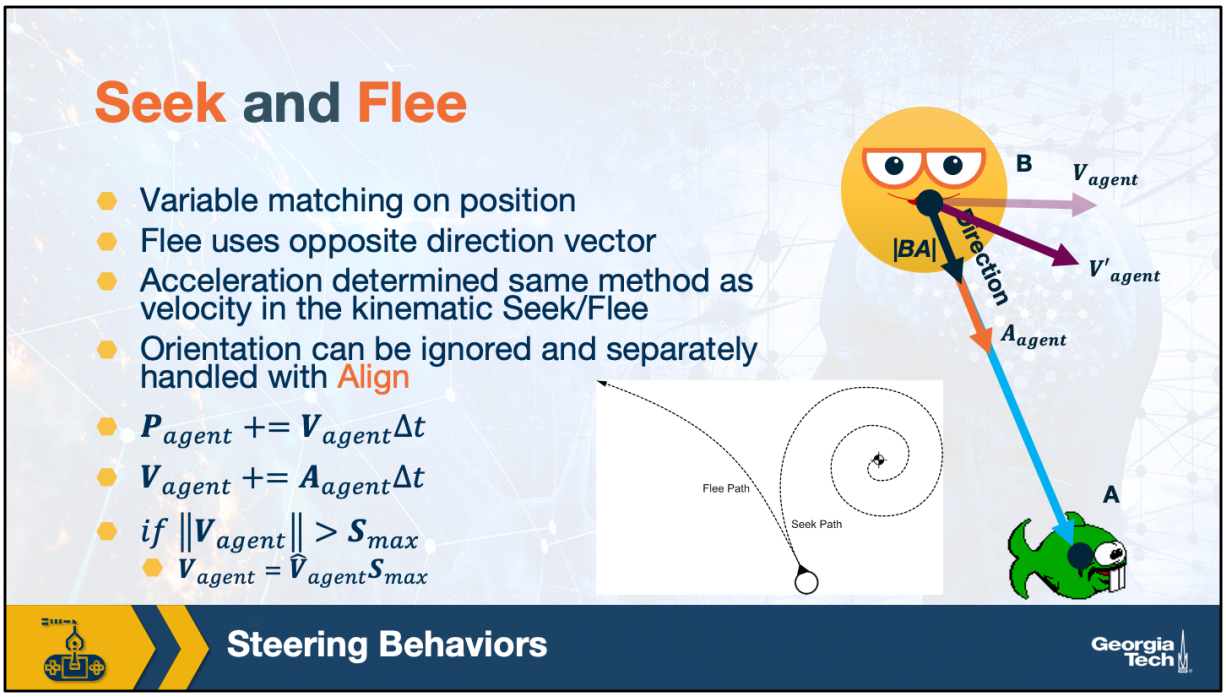
- Instead of a speed limit, drag can be used
- This is common with agents simulated by physics engines



Variable Matching

- With the notion of accelerating, one can consider objectives other than directing the agent to a target
- For instance, the agent could accelerate towards matching the target's velocity
- Possibilities for matching: position, velocity, orientation, angular velocity
- Also, the opposite of matching can be considered (maximize the difference)





Millington: Figure 3.8: Seek and flee

Seek Orbit Problem

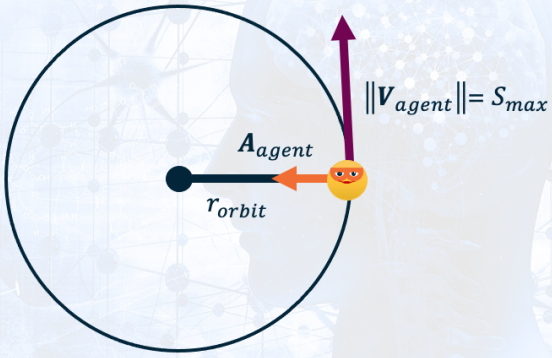
- Static target positions can result in orbit

Georgia Tech

Red Dead Redemption 2 – bear agent orbiting an injured NPC
https://streamable.com/5yudw

Seek Orbit Problem

- For steering behavior with acceleration
- Centripetal Acceleration Formula
  - $a = \frac{v^2}{r}$
- Agent's orbit radius:
  - $r_{orbit} = \frac{{S_{max}}^2}{\|A_{agent}\|}$
  - r_{orbit} is the minimum turning radius for the velocity and acceleration



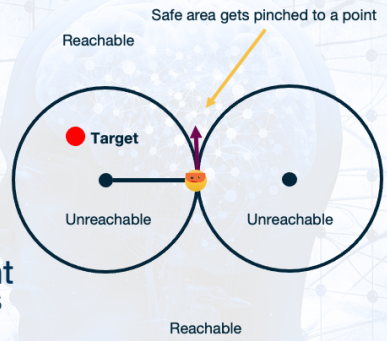
Steering Behaviors

Georgia Tech

Acceleration towards target is probably the maximum allowed for the agent

Seek Orbit Problem

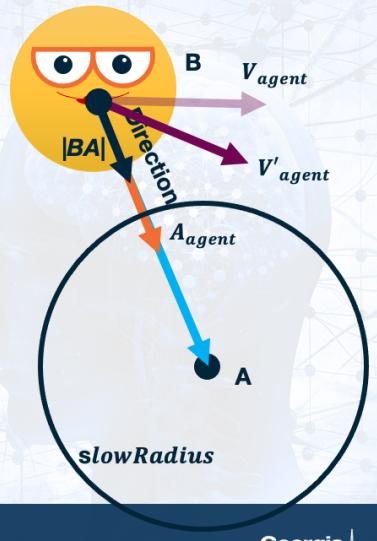
- The center of minimum turning radius is perpendicular to agent's velocity (left or right)
- A target within minimum turning radius cannot be reached at maximum velocity and maximum acceleration *towards* the target. Agent will orbit (possibly precessing) around target.
- Either slow down or move away and come back
- Demonstrates the instability of reaching a point target as reachable area outside two radii gets ever smaller just in front of agent
  - Why a capture radius is needed

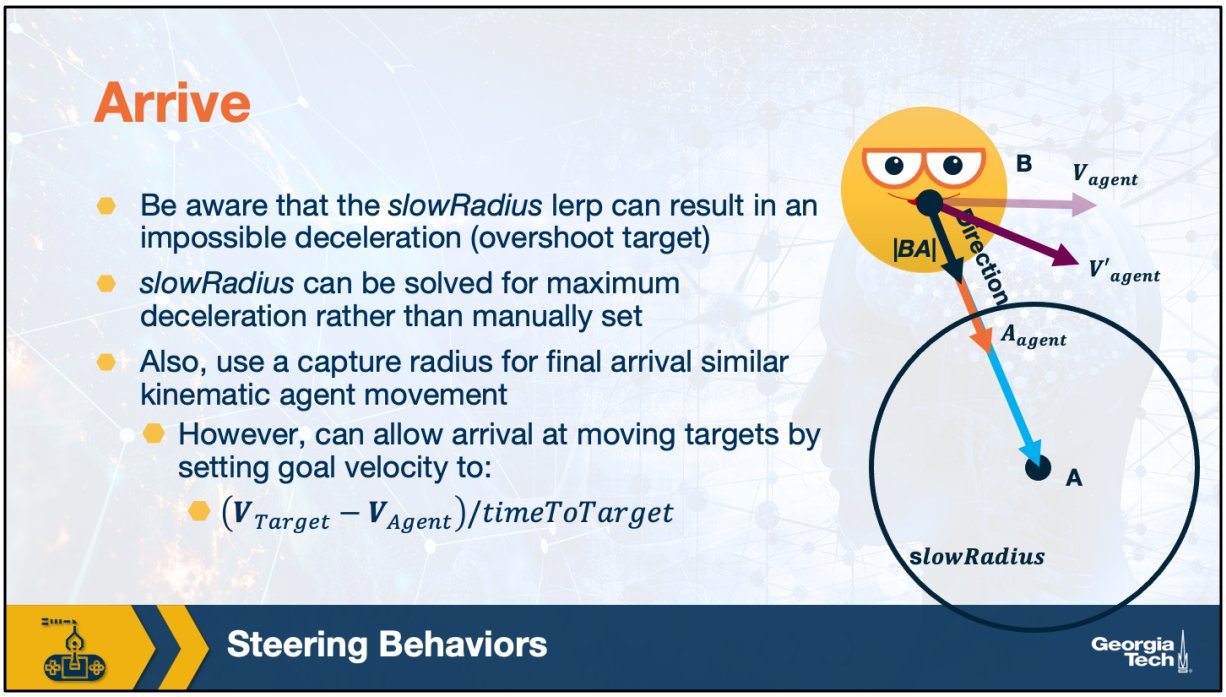


Acceleration towards target is probably the maximum allowed for the agent

Arrive

- Variable matching on position
- But stopping at the position (unlike seek)
- Agent must decelerate at the right time, otherwise will overshoot target
- The target speed is often implemented as a lerp of *maxSpeed* down to zero according to distance to target relative to *slowRadius*



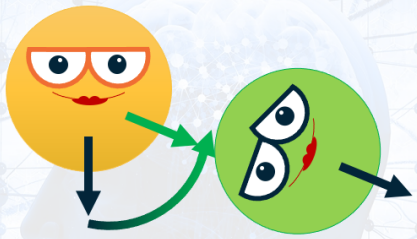


The capture radius is much smaller than the slowRadius

timeToTarget can be smaller than kinematic arrive since already decelerating with the slowRadius

Align

- Match target's orientation
- Same basic algorithm as Arrive but working with orientation and angular velocities and acceleration
- Use a slow angle and a capture angle
- Angle are unique and must be mapped to a $[-\pi, \pi]$ range relative to current facing angle
  - Turns in the shorter direction



Slow angle is like slow radius from Arrive (begin decelerating). Capture angle is like capture radius (match orientation or angular vel of target)

Velocity Matching

- Agent matches the velocity of the target
- Not especially useful on its own, but can be combined with other behaviors (e.g., Boids multi-agent flocking)
- Implementation is basically the capture radius from Arrive:
  - $(\boldsymbol{V}_{Target} - \boldsymbol{V}_{Agent})/timeToTarget$
  - Acceleration is clipped to A_{max}

Basic Steering Behaviors

- Seek/Flee
- Arrive/Flee
- Align
- Velocity Match
- From these, all other steering behaviors can be created

Alternate Steering Behavior Design

- Orientation and the direction of velocity can be locked together
- One approach is for the agent's angular velocity to dictate direction of translation
- The main advantage is that the agent won't ever appear to slide

Steering Behavior Sliding

- Example of sliding with steering behaviors
- Seek with independent Align to own velocity
- New target immediately behind agent requires heading back

Steering Behaviors

Georgia Tech

Minion is going to random waypoints. This example the minion has to double back

Turn-First Steering Behavior

- Agent first attempts to align with relative vector to target (angular acceleration)
- Next, agent moves towards target
- Simplest case: agent always goes at full velocity (unless arriving or matching speeds)
  - Problem: Wide turns
- More advanced:
  - Implement turn-induced deceleration
  - Implement variable forward acceleration

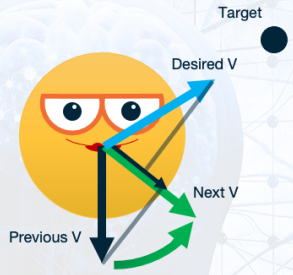
Steering Behaviors

Georgia Tech

Slow angle is like slow radius from Arrive (begin decelerating). Capture angle is like capture radius (match orientation or angular vel of target)

Advanced Turn-First Steering Behavior

- Project previous frame's velocity vector onto new forward (orientation) vector
- This deceleration is immediately applied as a velocity change (due to turning)
- But the agent *also* accelerates
- Can always try to accelerate to max speed (unless arriving, etc.) or…
- Project the relative target unit vector scaled by max speed onto current velocity
  - This is the target velocity to accelerator towards
  - If negative direction, agent can even back up while turning!



Steering Behaviors

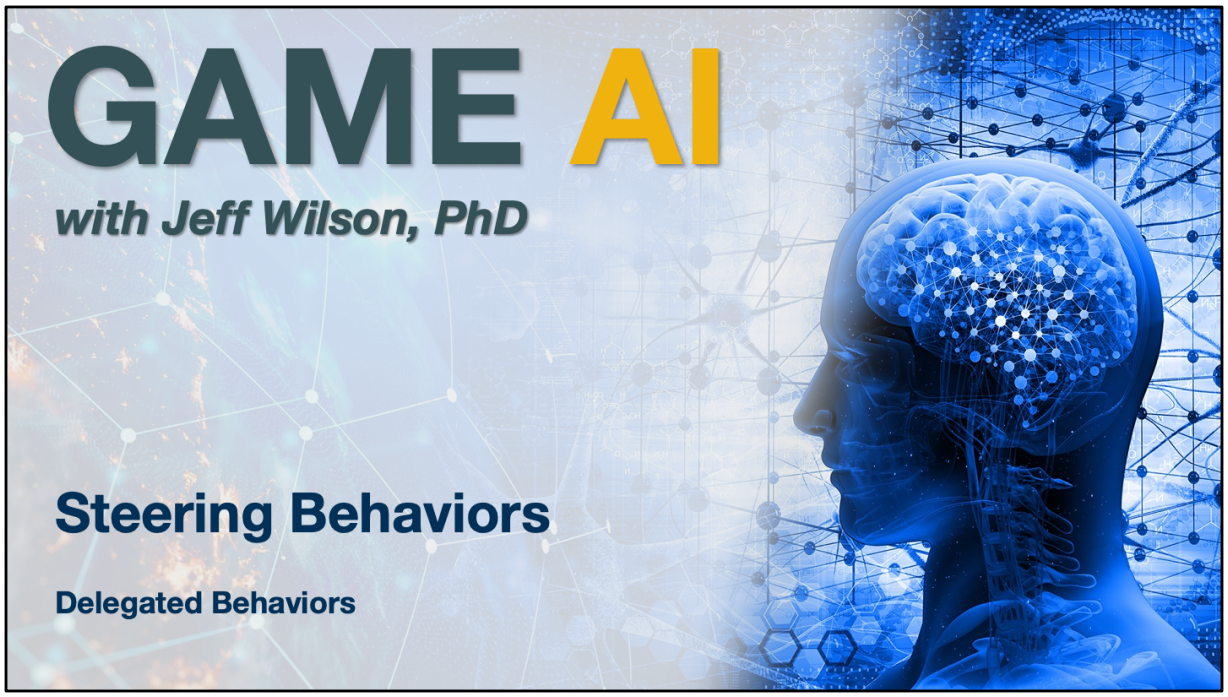
Georgia Tech

If previousV and nextV are close, then the projection will be about the same length (little or no decel)
Similarly, if DesiredV and NextV are close the projection will be about the same length (accel instead of decel)

Turn-First Steering Behavior

- Orbits change to: $r_{orbit} = \frac{S_{max}}{\|V_{angularMax}\|}$
  - Doesn't apply to fully advanced version of turn-first
- Cannot mix some steering behaviors together the same way
  - E.g., cannot align orientation of agent with one target while also seeking a different target

AngularV must be in radians

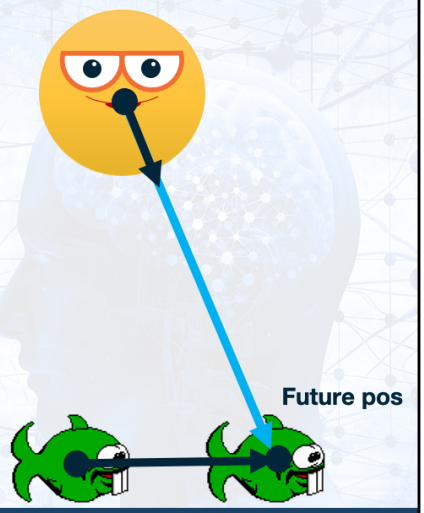

GAME AI
with Jeff Wilson, PhD
Steering Behaviors
Delegated Behaviors

Delegated Behaviors

- Seek, Align, and Velocity matching are the only fundamental steering behaviors
- We can seek/align to different calculated positions (such as offsets from obstacles)
- This can create new delegated steering behaviors

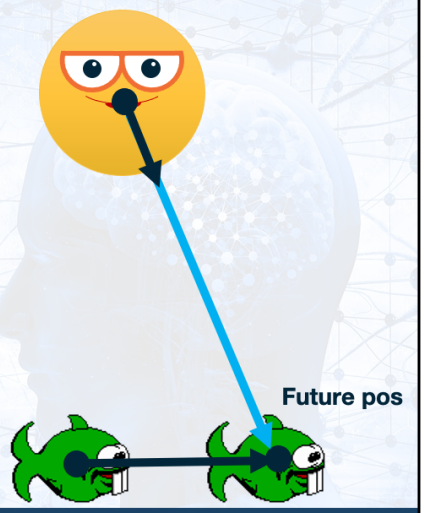
Pursue

- Seek with prediction
- No need to calculate precise intercept
- The calculation gets made every frame, so we want as efficient as possible!
- We can use a heuristic



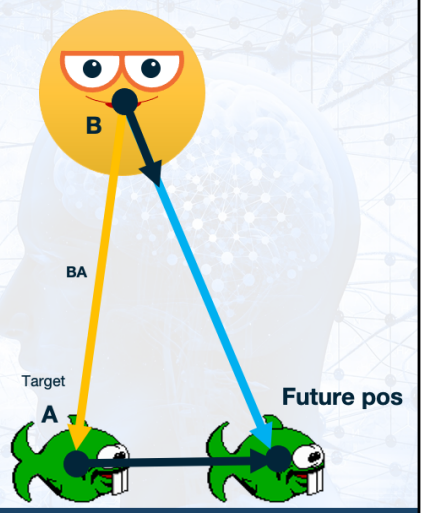
Pursue

- Seek with prediction
- No need to calculate precise intercept
- Linear extrapolation assumption (zero acceleration/constant velocity)
- We *could* calculate a precise intercept but…
- The calculation gets made every frame, so we want as efficient as possible!
- We can use a heuristic

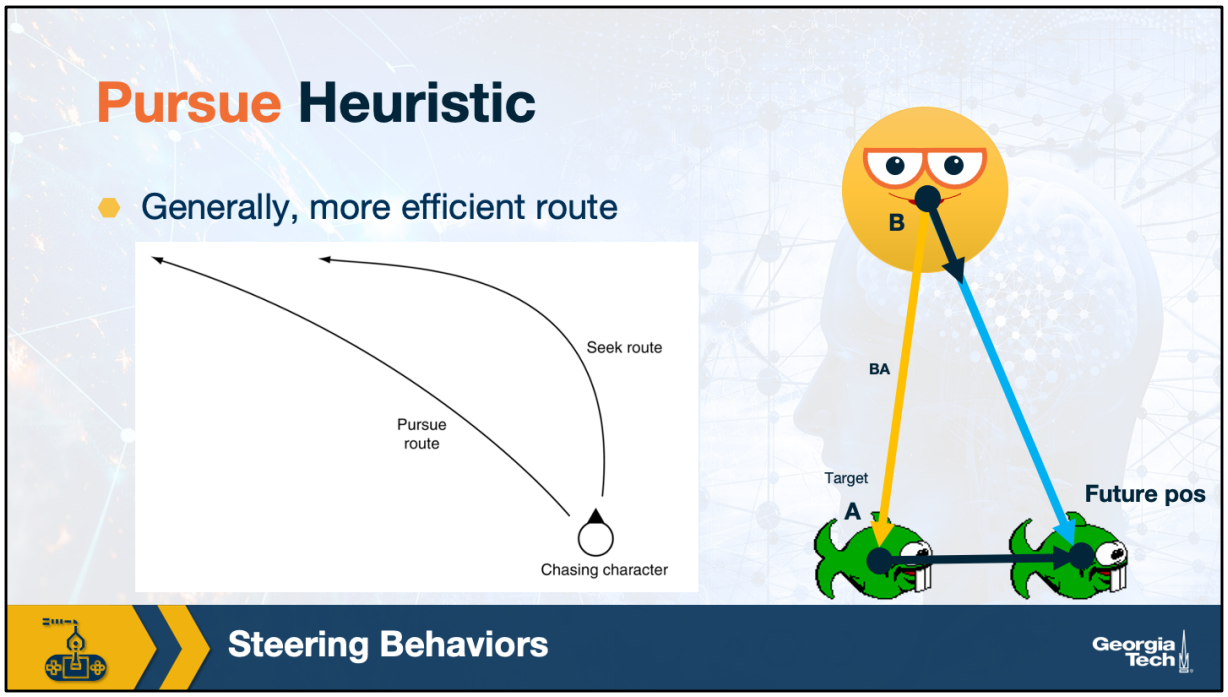


Pursue Heuristic

- Determine time (t) to lookahead for estimated intercept
- $t = \frac{\|\overrightarrow{BA}\|}{S_{AgentMax}}$
- $\boldsymbol{P}_{futureTarget} = \boldsymbol{P}_{Target} + t * \boldsymbol{V}_{target}$
- Calculate $\boldsymbol{P}_{futureTarget}$ every frame
- Seek $\boldsymbol{P}_{futureTarget}$
- Can also Evade (Flee from $\boldsymbol{P}_{futureTarget}$)



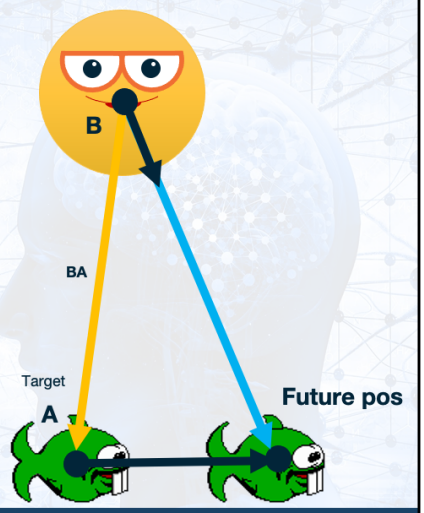
A is the $\boldsymbol{P}_{Target}$



Millington: Figure 3.12: Seek and pursue

Prediction Gotchas

- Watch out for extreme predictions (very large lookahead *t* values)
- Could be sending your agent off the map or result in odd behavior
- Consider clamping max time prediction (and even minimum)
- Consider clipping extrapolated future positions to fit on navmesh or map, etc.



Face

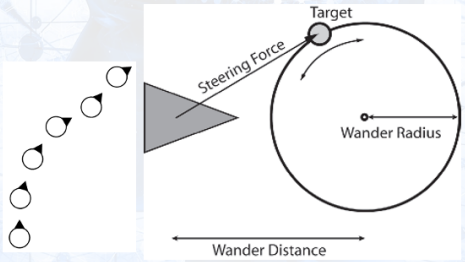
- Align towards a position
- Determine a relative position vector to target for a direction
- Use $Atan2()$ to calculate goal orientation angle from x, y components of direction
- Apply Align

Look Where Going

- Align towards current velocity
- Apply in conjunction with other behaviors affecting velocity
- Same as Face but use the agent's linear velocity to determine orientation angle

Wander

- With acceleration, can do better than kinematic wander
- A circle is placed in front of agent by some offset
- A target point rotates around the circle according to random function (see kinematic wandering)
- Face the target
- Max acceleration toward orientation

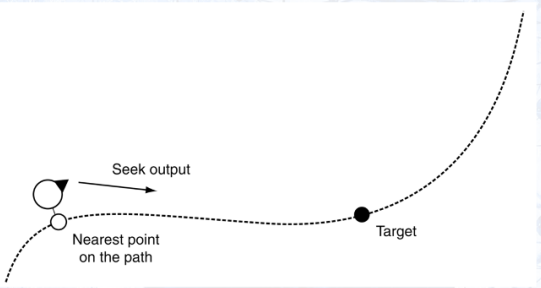


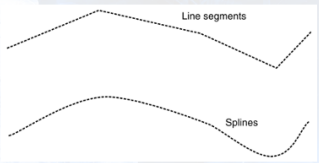
Georgia Tech

From Buckland Fig 3.4 and Millington Fig 3.7

Path Following

- First, *find-nearest-point-on-path*
- Second, pick a target some point further down the path
- Seek the target
- *find-nearest-point-on-path* may be challenging
  - Line segments
  - Bézier Splines (non-linear)
  - May need Space Partitioning to quickly identify candidate segments

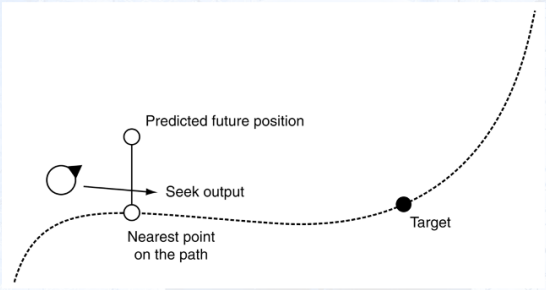




Millington Figure 3.15: Path following behavior

Predictive Path Following

- First, predict a future position
- Second, *find-nearest-point-on-path* using future position
- Third, pick a target some point further down the path
- Seek the target
- Can be smoother with the prediction



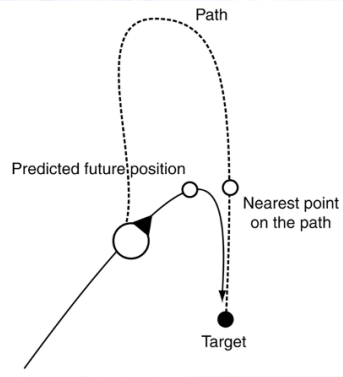
Steering Behaviors

Georgia Tech

Millington Figure 3.16: Predictive path following behavior

Predictive Path Following

- However, may cut corners
- This could be undesirable for guard patrols, races, etc.



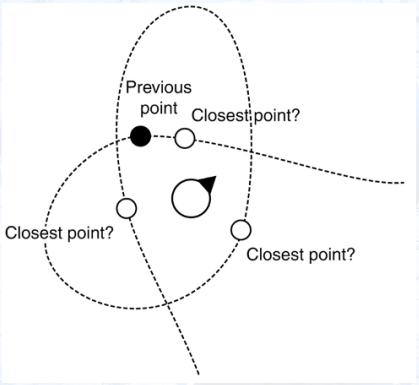
Steering Behaviors

Georgia Tech

Millington Figure 3.17: Vanilla and predictive path following

Path Following

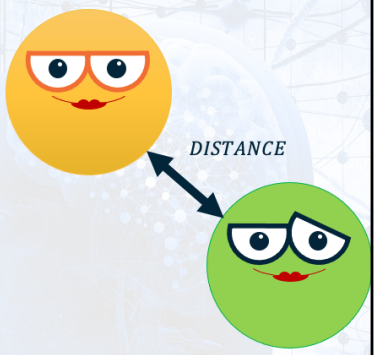
- Very useful to store last known point on the path to limit search for closest path segment **and** disambiguate
- Overlapping path might cause ambiguity
- Example of coherence



Millington Figure 3.19: Coherence problems with path following

Separation

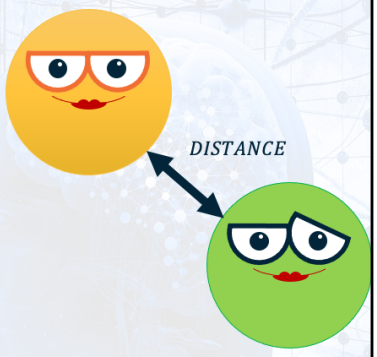
- Separate agents without fleeing forever
- Good for crowd simulation (heading in same direction)
- Only separate within some distance threshold
- $STRENGTH_{Linear} = \frac{A_{max}(THRESHOLD - DISTANCE)}{THRESHOLD}$
- $STRENGTH_{InvSqr} = Min\left(\frac{k}{DISTANCE^2}, A_{max}\right)$



A_{max} is acceleration
Inverse Square Law

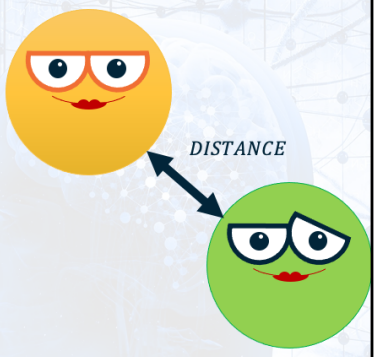
Separation

- In a crowd: can separate from only the closest neighbor or sum of forces from nearby neighbors (clipped to A_{max})
- Need Space Partitioning and Coherence to quickly identify neighbors



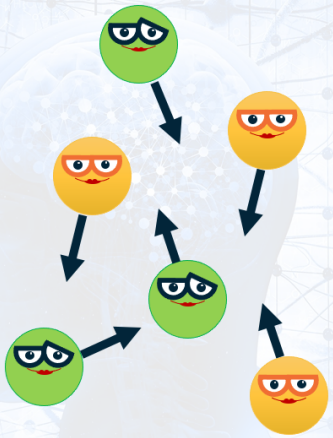
Attraction

- Opposite of Separation
- Not often useful. Mainly for group behaviors such as flocking
- Typical to use $STRENGTH_{InvSqr}$ with a negative k (constant of decay)



Collision Avoidance

- Avoid other agents
- Especially large numbers in the same space

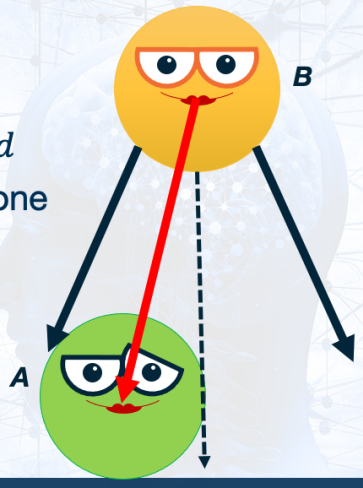


Steering Behaviors

Georgia Tech

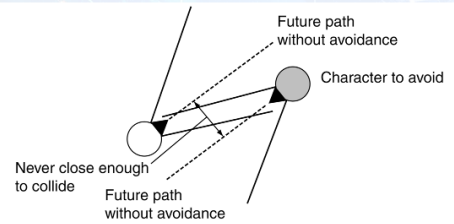
Collision Avoidance Cone

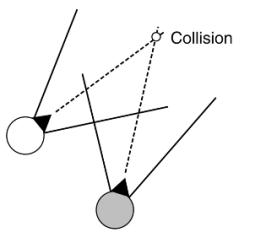
- Implement cone test with dot product
- $isCollision = \hat{V}_{agent} \cdot \widehat{\overrightarrow{BA}} > ConeThreshold$
- $ConeThreshold$ is cosine of half angle of cone
- If $isCollision$, avoid similarly to separation
- Doesn't work well with a lot of agents
- Agent tends to overreact



Collision Avoidance Cone

- More problems:
  - Avoidance when no collision imminent
  - No avoidance when collision imminent





Steering Behaviors

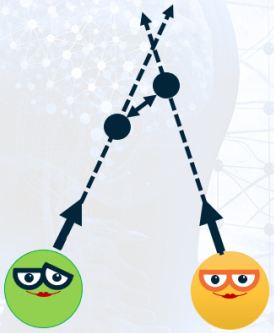
Georgia Tech

Millington Figure 3.21: Two in-cone characters who will not collide

Figure 3.22: Two out-of-cone characters who will collide

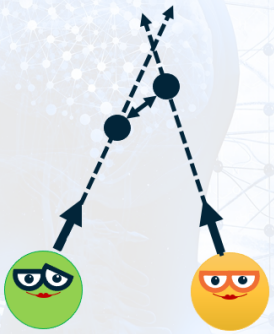
Collision Avoidance with Constant Velocity Assumption

- Points of Closest Approach
- Will be different than intersection of trajectories
- Due to differing speeds
- Find time (*t*) of closest approach



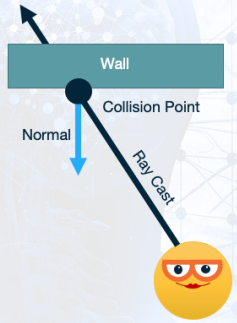
Collision Avoidance with Constant Velocity Assumption

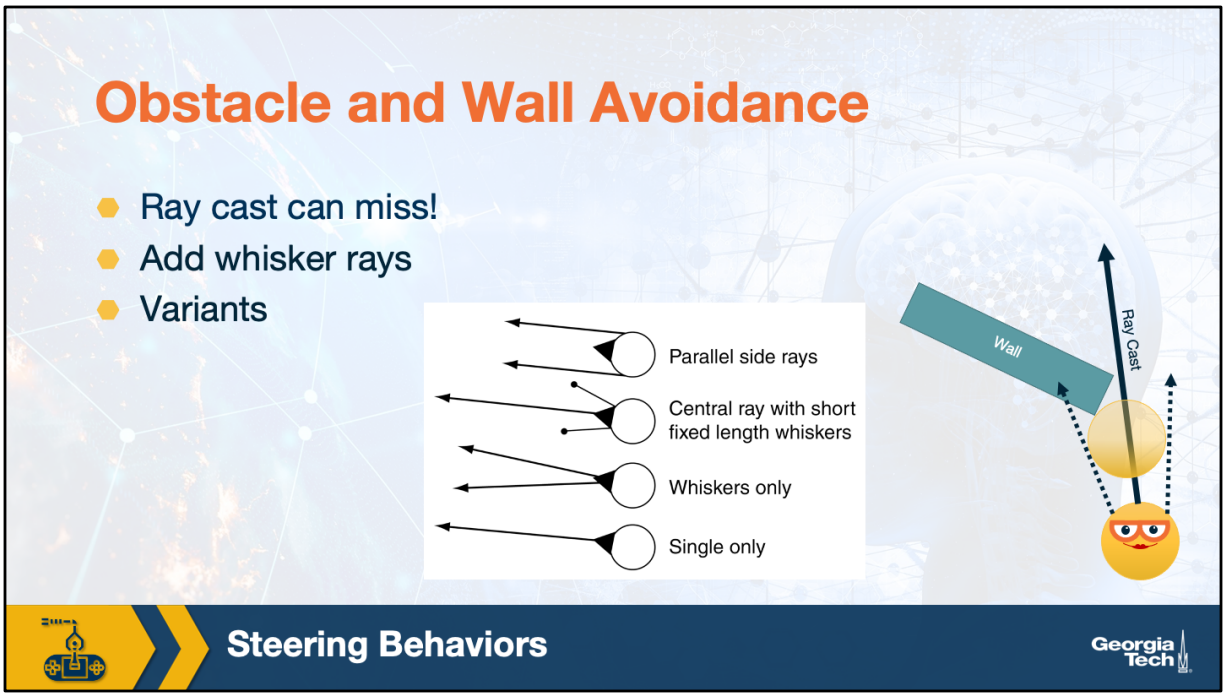
- $t_{closest} = \frac{d_p \cdot d_v}{\|d_v\|^2}$ where $d_p = p_{target} - p_{agent}$ and $d_v = v_{target} - v_{agent}$
- If $t_{closest}$ is negative, entities are moving away from each other, otherwise…
- $p'_c = p_c + v_c t_{closest}$ and $p'_t = p_t + v_t t_{closest}$
- These positions used for avoidance
- React to only the closest (doesn't work well with multiple agents)
- Also need special case for already colliding



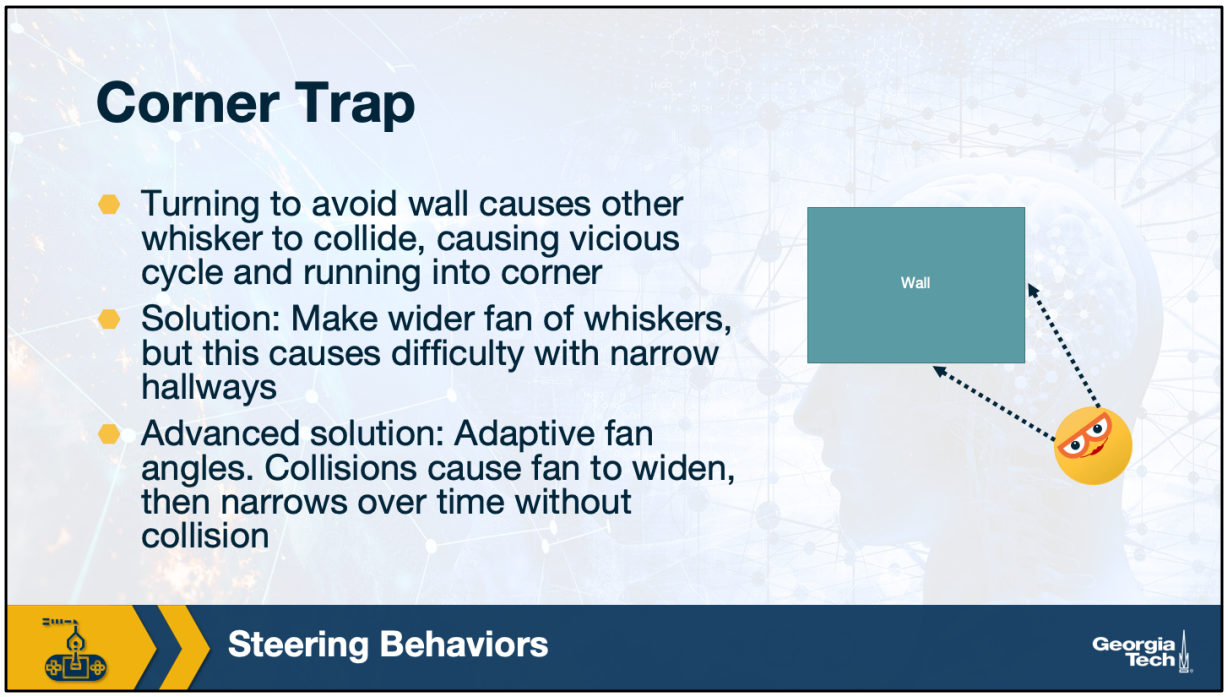
Obstacle and Wall Avoidance

- Cast a ray against nearby obstacles in direction of current velocity multiplied by some lookahead *t*
- If there is a hit, find the normal of the obstacle at collision
- This normal is multiplied by an avoidance distance
- The avoidance distance can be a function of ray penetration
- Determine offset from collision and Seek





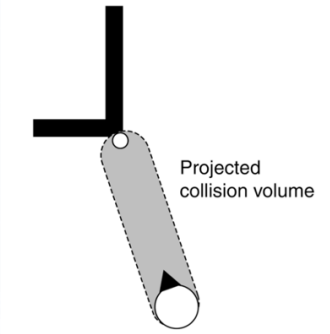
Millington Figure 3.26: Ray configurations for obstacle avoidance



Millington Figure 3.26: Ray configurations for obstacle avoidance

Obstacle and Wall Avoidance

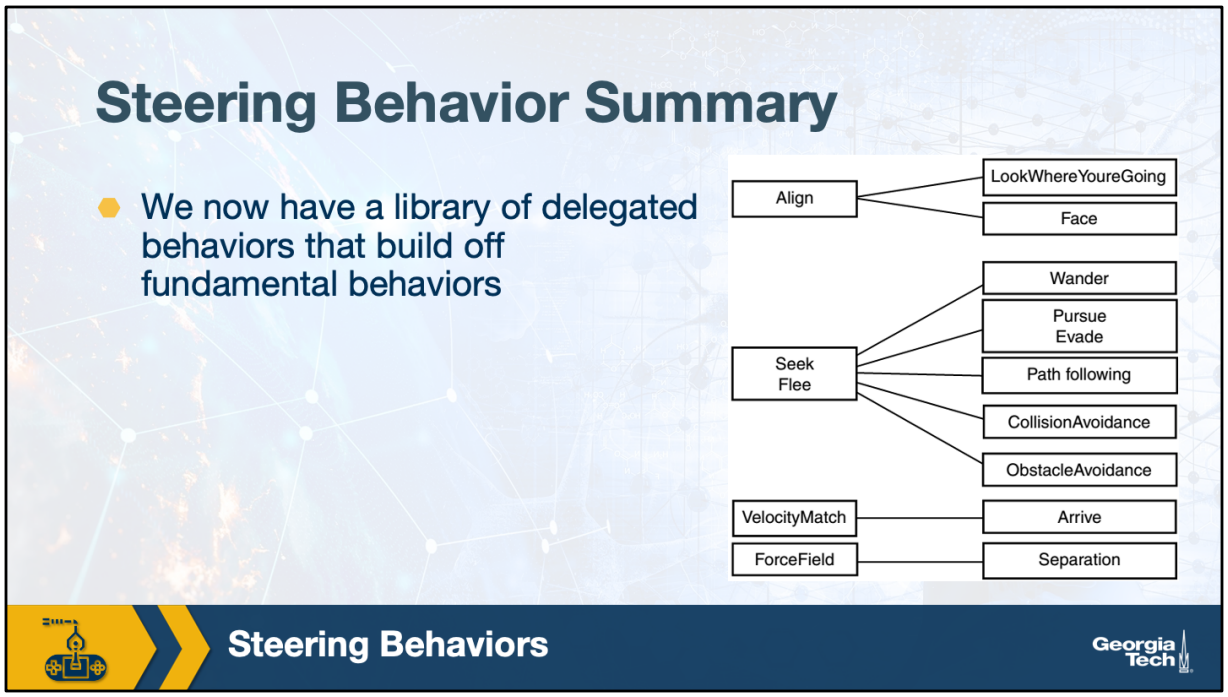
- Collision detection with physics engine support
- But computationally expensive and difficult to interpret collisions and appropriate avoidance behavior



Steering Behaviors

Georgia Tech

Millington Figure 3.28: Collision detection with projected volumes

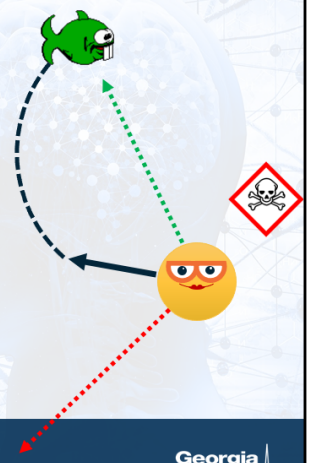


Millington Figure 3.29: Steering family tree


GAME AI
with Jeff Wilson, PhD
Steering Behaviors
Blended Behaviors

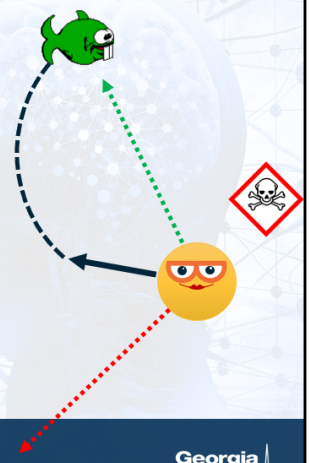
Multiple Steering Goals? Combining Steering Behaviors

- What if agent needs to steer towards a goal while also avoiding a danger?
- Solution: Blend steering forces additively or arbitrate



Blend Steering Behaviors

- Sum of Forces
  - Need to enforce max acceleration
- Weighted Sum
  - Each force has a weight that is multiplied by the generated force
  - Weights don't need to sum to 1.0
  - Weights can be based on some metric like health, to favor one strategy over the other given situation

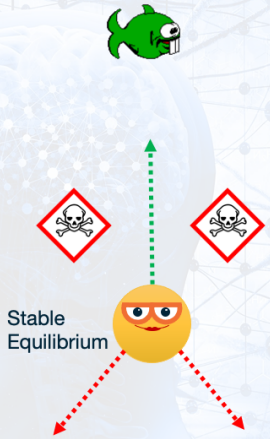


Problems with Blending

- Balanced forces can cancel out
- Can be an unstable equilibrium
  - Forced cancel out, but numerical instability will eventually lead to a shift to left or right
  - Still anomalous behavior observed
- Or there can be a stable equilibrium…

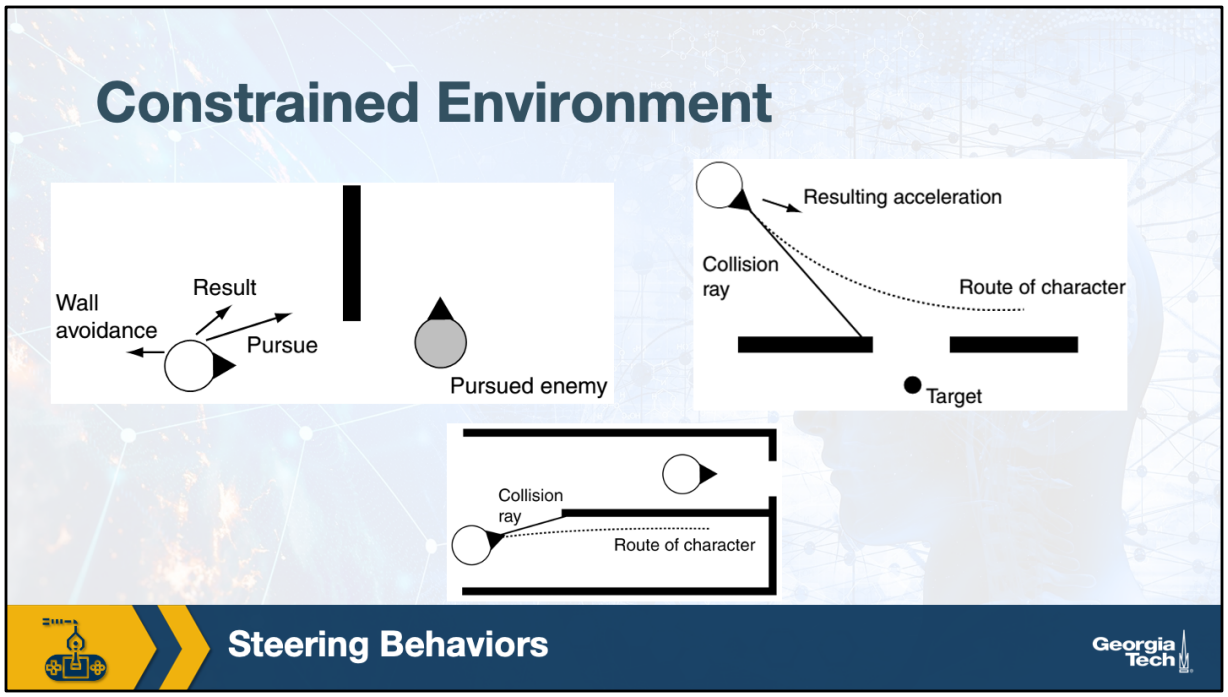
Problems with Blending

- Stable equilibrium has a basin of attraction
- Agent gets stuck at the equilibrium point (forces balance to zero)



Constrained Environment

- Indoor environments, canyons, lots of agents, can create constrained environments
- Constraints that lead to lots of steering forces can create obvious failures detrimental to the gameplay experience
- Steering behavior agents act most strongly to local influences. This creates nearsightedness that leads to failures



Millington: Figure 3.35: Can't avoid an obstacle and chase , Figure 3.36: Missing a narrow doorway Figure 3.37: Long distance failure in a steering behavior

Solution to Blending Problems

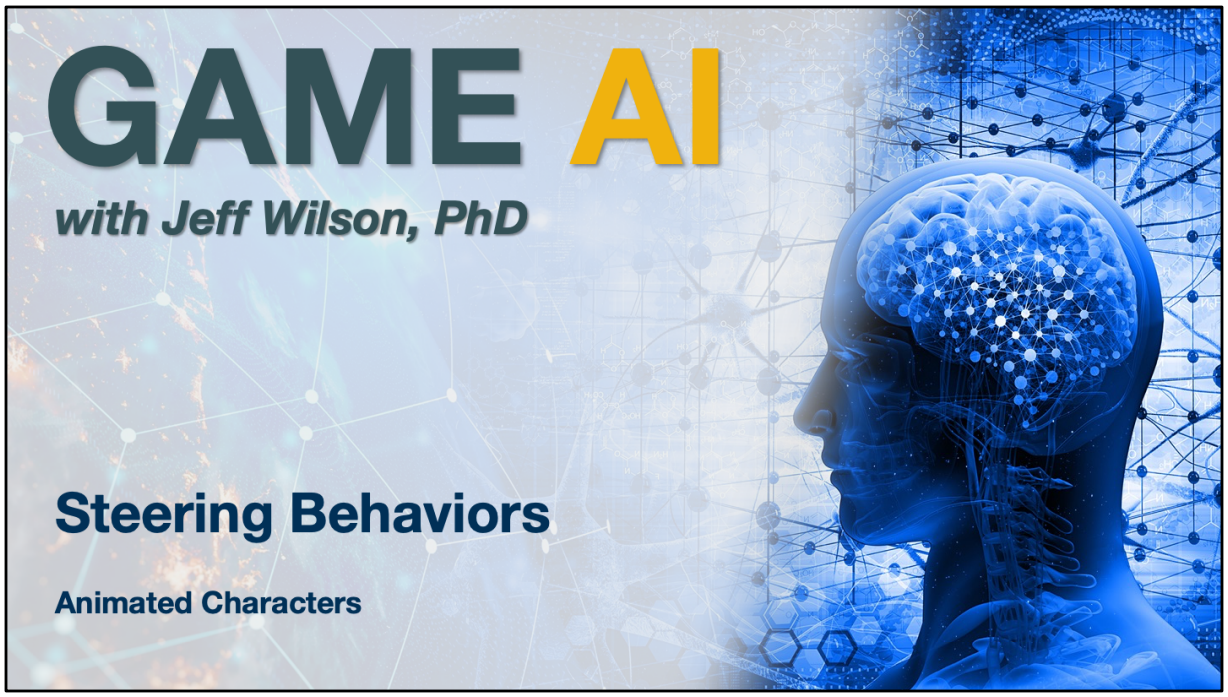
- Higher level planning
- Path Planning
- Decision Making
  - Often Reactive Decision Making

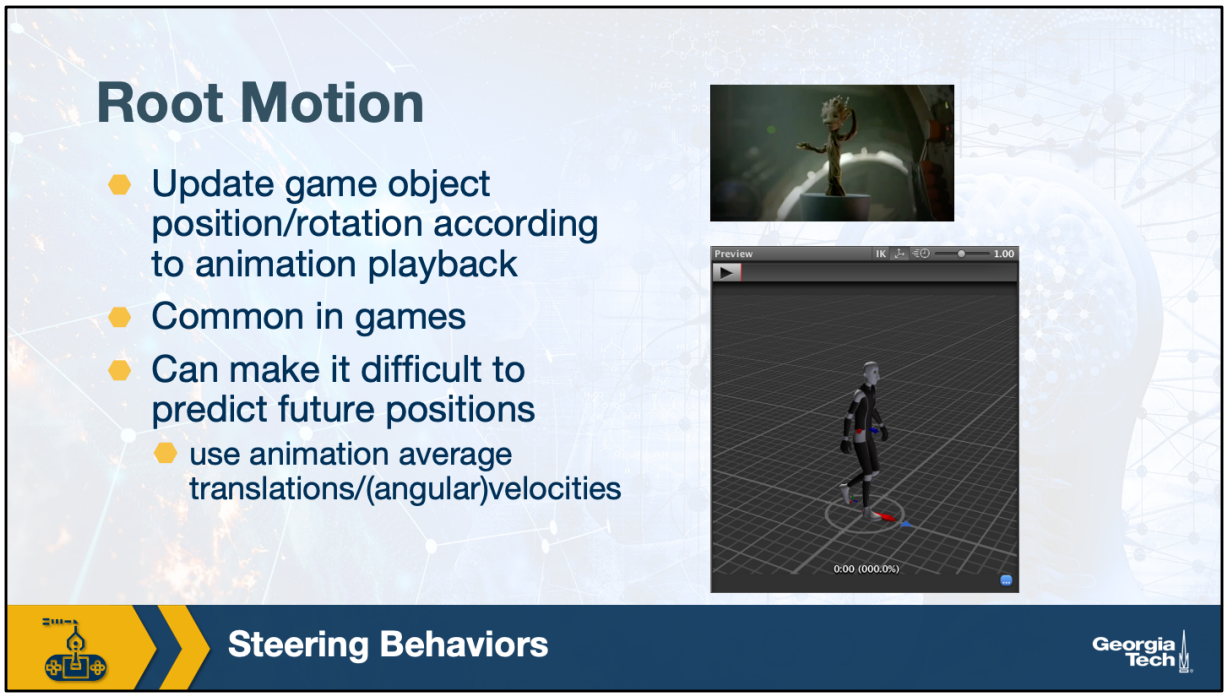
Priority Blending

- Beginning to blur the line between simple steering behaviors and integration of reactive decision making
- Observation: Many of the steering behaviors we have seen only return a steering force in the presence of a stimulus (e.g., obstacle avoidance)
- These forces are generally important (responding to imminent threats)
- These forces can become diluted by other forces

Priority Blending

- Can prioritize steering behaviors that activate above a certain force threshold
- Can be applied to a single steering behavior or a set (combination)
- Can immediately avoid unstable equilibria and stable equilibria with small enough basin of attraction
- Constrained environments and large basins of attraction are still an issue


GAME AI
with Jeff Wilson, PhD
Steering Behaviors
Animated Characters



https://docs.unity3d.com/Manual/RootMotion.html

The Body Transform is the mass center of the character. It is used in Mecanim's retargeting engine and provides the most stable displacement model. The Body Orientation is an average of the lower and upper body orientation relative to the Avatar T-Pose.

Generic Root Motion and Loop Pose

This works in essentially the same as Humanoid Root Motion, but instead of using the Body Transform to compute/project a Root Transform, the transform set in **Root Node** (e.g. the hips) is used. The Pose (all the bones which transform below the Root Motion bone) is made relative to the Root Transform

Animation Blending

- Achieve variations between different extremes of similar animations
- Basic case, usually must be ”similar” animations
- Skeletal animation highly beneficial
- **Root motion can be blended too!**

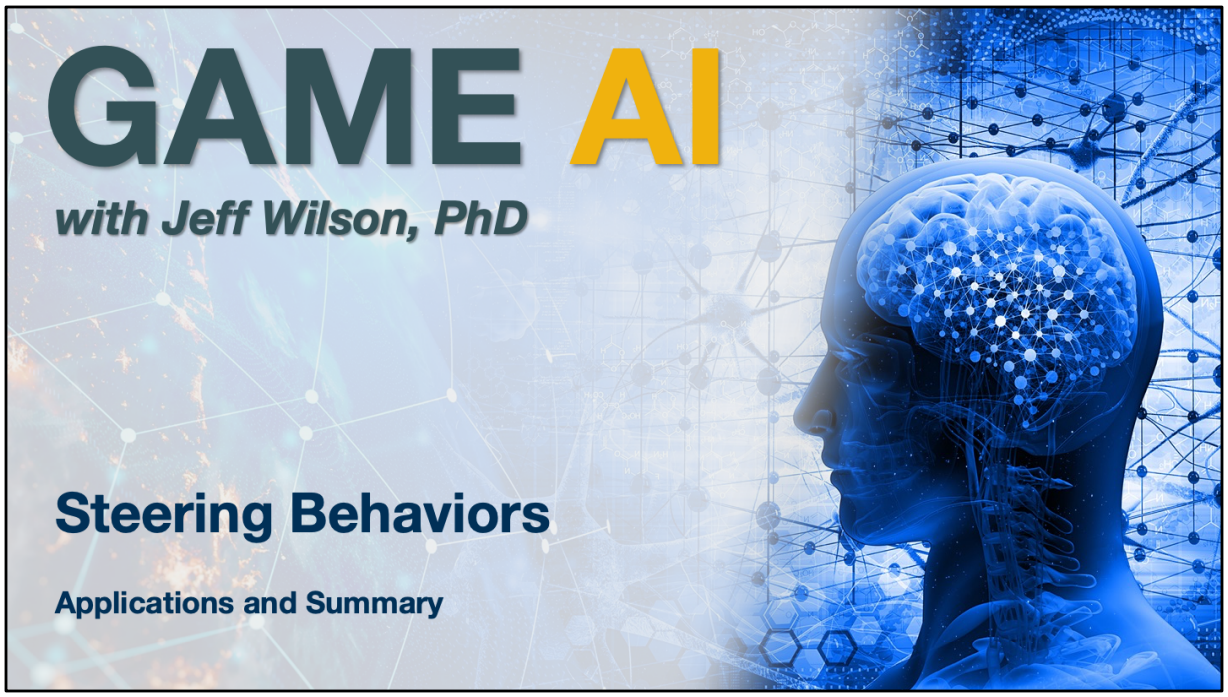
https://markobl.com/2014/12/03/animation-blending-in-monogame-xna/

Motion Table for Blended Root Motion

- Problem: How can agent select animation blend parameters that best direct its movement towards steering target?
- Could fit a function, but can become difficult for higher order functions. Or…
- **Create a lookup table**
- Table dims: Anim_param_x, Anim_param_y: [-1.0,1.0] (left/right, forward/bkwd)
- Choose some table resolution (e.g. epsilon=0.01)
- Store expected root motion (x_vel, y_vel, angVel, etc.)
- Use table to find anim params with closest root motion to steering target
- Note that animation params are often filtered (acts like an acceleration towards a velocity)

Motion Table for Blended Root Motion

- Improvement: Add linear interpolation to table selection
- For any parameters looked up, find the closest cells
- Then perform a weighted average according to distance to each cell


GAME AI
with Jeff Wilson, PhD
Steering Behaviors
Applications and Summary

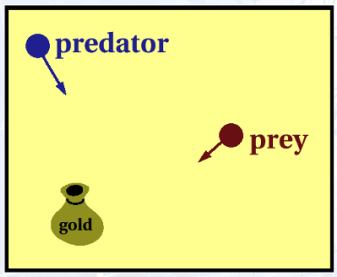
Creating Unique Agents

Unique agents can be authored by:

- Blending steering behaviors
  - E.g., add small perturbations with wandering along with seeking
  - Use different parameters for prediction in pursue, obstacle avoidance, etc.
- Defining a performance envelope
  - Set unique max (angular) velocity, max (angular) acceleration
  - Possibly driven by animation system (e.g., root motion)

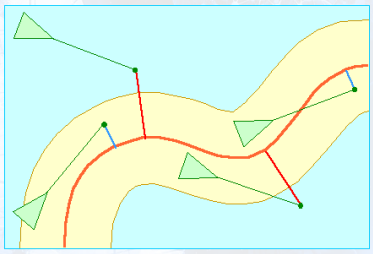
Creating Unique Agents

Steering behaviors designed around rules of the game



OpenSteer

- OpenSteer
  - Craig Reynolds
  - http://www.red3d.com/cwr/steer/
  - GDC'99 Paper: http://www.red3d.com/cwr/steer/gdc99/
- Pursue
- Evade
- Wander
- Obstacle Avoidance
- Wall/Path following
- Queuing
- Combine behaviors with weights



Summary

- Simple but powerful technique to implement real-time agent movement
- Good for handling local movement
- Susceptible to getting stuck in equilibrium or not performing behavior with desired outcome
- More advanced agents that rely on steering behaviors will also need support of a higher-level planning system such as path planning and/or reactive decision making like a Finite State Machine