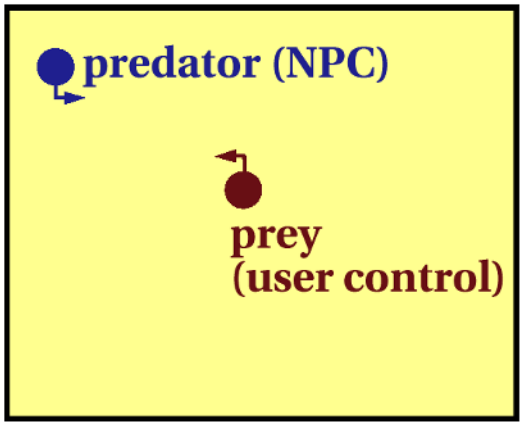
Movement and Steering Behaviors

Chase/Evade

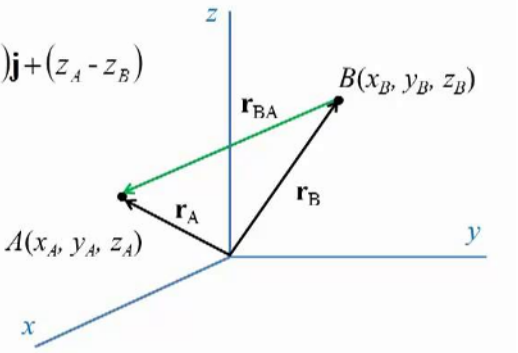
- Algorithm for the predator?



Identify target

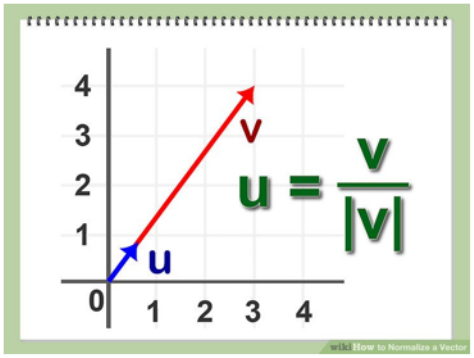
- Calculate relative position vector

$$\begin{aligned}\mathbf{r}_{BA} &= \mathbf{r}_A - \mathbf{r}_B \\ &= (x_A - x_B)\mathbf{i} + (y_A - y_B)\mathbf{j} + (z_A - z_B) \\ &= -\mathbf{r}_{AB}\end{aligned}$$



Direction and Distance

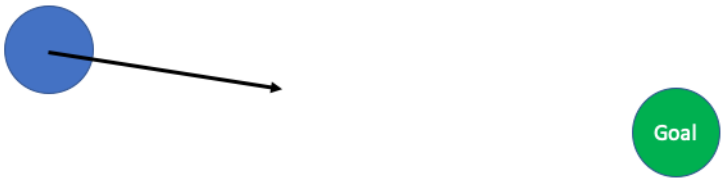
- Normalize for vector
- Magnitude for distance



$$\text{2D: } |\mathbf{v}| = \sqrt{x^2 + y^2}$$
$$\text{3D: } |\mathbf{v}| = \sqrt{x^2 + y^2 + z^2}$$

Simple Movement

- Orient agent velocity in target direction
- Very aggressive and doesn't look very natural (can change dir instantaneously)
- Fine for discrete movement (on a grid)



Steering Movement

- Turn towards target vector
- Adjust speed, perhaps slowing forward velocity if target angle is large and accelerating to max velocity as target angle becomes small

Fleeing

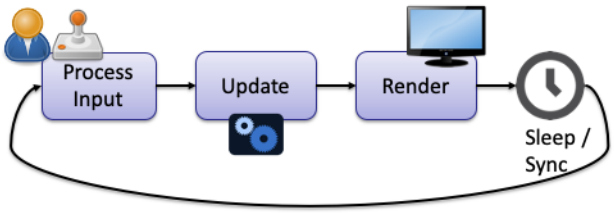
- Simply calculate opposite ordered position vector

Steering Behavior: Performance Envelope

- Typically: constant acceleration and enforced max velocity for translations and rotations (speed and turn rates)
- Can get progressively more advanced, introducing acceleration curves, separate deceleration rates, speed dependent turn rates, etc.
- Can be based on forces/torques, and agent mass

Continuous Simulation

- Frame-based
- Closed-loop where player is part of the overall system



We make assumption that player is only acting/considering info within the system and isn't influenced by outside information

Time Dependency

- Unity Demonstration
- Frame rates are not constant. Repercussions?
- Impact on steering behaviors?
- newPos = oldPos + ConstantVelOffset
- Vs
- newPos = oldPos + vel * deltaT ←**BETTER!**

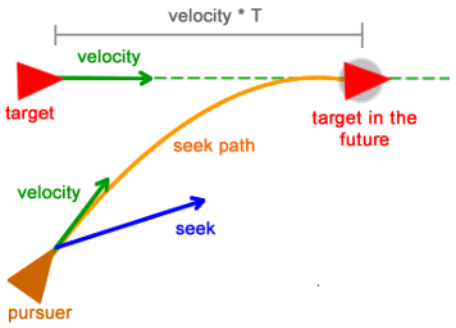
Once upon a time, frame rate didn't matter so much. MSDOS was effectively a real-time system (no multi-tasking). Also, games were often sprite based, with a fixed amount of graphical objects onscreen (so processing load was effectively constant). Games could simplify calculations for animations by just counting frames. Provided that everyone had the same clock speeds on their CPU, this approach worked fine for 2D sprite-based games, which have pretty constant processing demands.

A side effect of this, is that newer computers ran the games too fast. Utilities such as "Mo'Slo" allowed a partial fix

Chrono Trigger

Prediction

- No need to calculate precise intercept
- Recalculating every frame anyways
- Dist = (target.pos – agent.pos).Length()
- lookAheadT = Dist/agent.maxSpeed
- futureTarget = target.pos + lookAheadT * target.velocity
- //Steer towards futureTarget



Prediction Gotchas

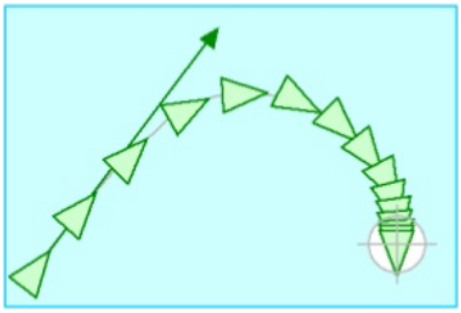
- Watch out for extreme predictions (very large lookahead ***t*** values)
- Could be sending your agent off the map or result in odd behavior
- Consider clamping max time prediction (and even minimum)
- Consider clipping extrapolated future positions to fit on navmesh or map, etc.

Zero/Small Vectors

- If relative position vector AB is zero or very small, you can get undesired behavior! (Consider vector normalization operation)
- Potentially dividing by a very small value (possibly zero)
- Best to check all vectors
- Possibly a condition to decide upon alternative steering strategy (e.g. a state machine state transition)
- Or just hold current heading and speed (perhaps a buffered sliding avg of the last several measures)

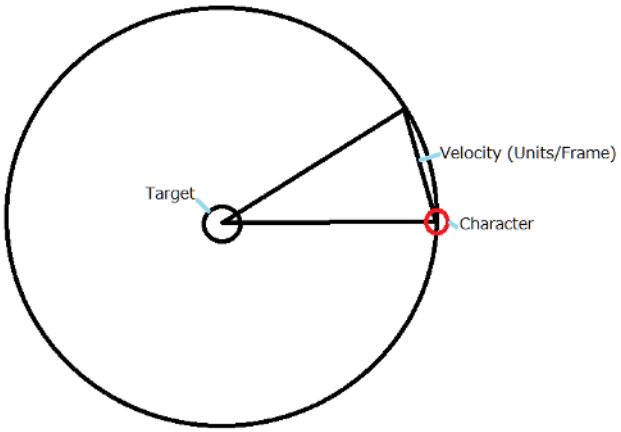
Arriving at static targets

- Full speed ahead when outside of ***stopping radius (sr)***
- sr = distance to go from max v down to 0, abiding by max deceleration possible (or some desired deceleration that agent is capable of)
- Linear scaling down of desired speed inside the stopping radius as target position is approached
- Issue: Missing target, huge steering angle correction vicious cycle
- Fixes: approximate test for reached goal and/or slide the agent over without requiring steering



Visiting Static Targets: Orbit Problem

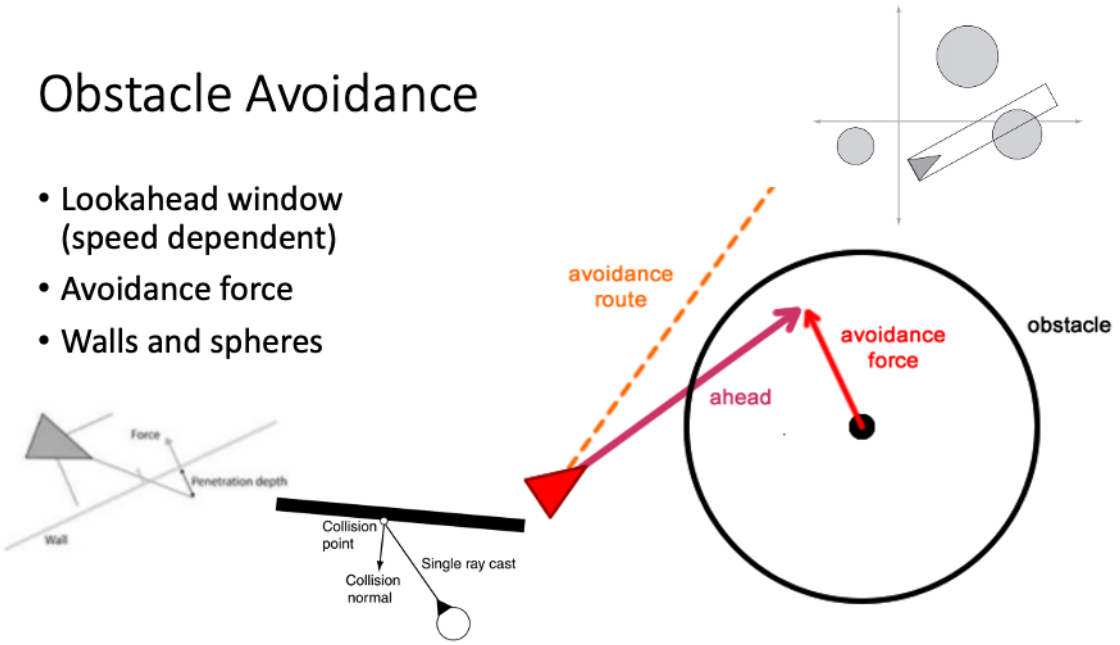
- Not planning on stopping at goal via stopping radius? (e.g. waypoint, collectable pick up, running attack, etc.)
- Fix: slow down agent velocity if target within orbit radius of agent (function of agent's velocity)
- Radius = tangential_vel / angular_vel_in_radians
- (tangential vel is agent's linear vel)



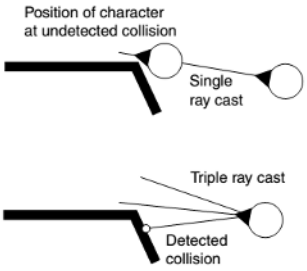
Red Dead Redemption 2 - bear
https://streamable.com/5yudw

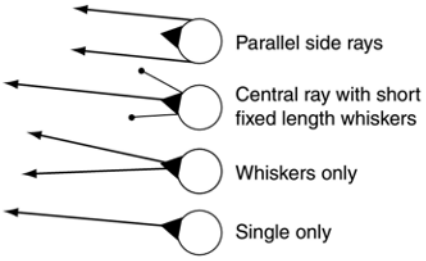
Obstacle Avoidance

- Lookahead window (speed dependent)
- Avoidance force
- Walls and spheres



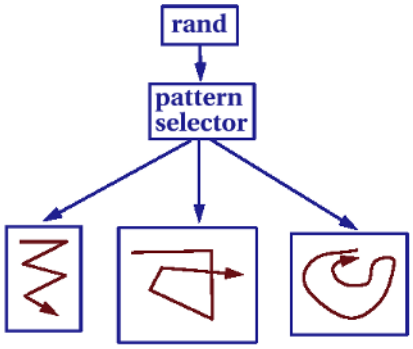
Obstacle Test – Likely need multiple tests





Enhancements to Chase

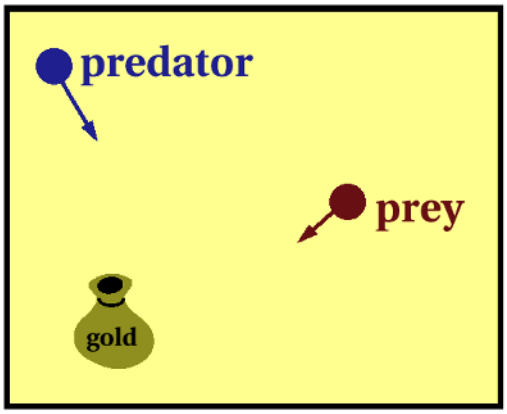
- Speed Control
  - Velocity, Acceleration max/min
  - Limited turning Radius
- Randomness
  - Moves
  - Patterns
- Prediction

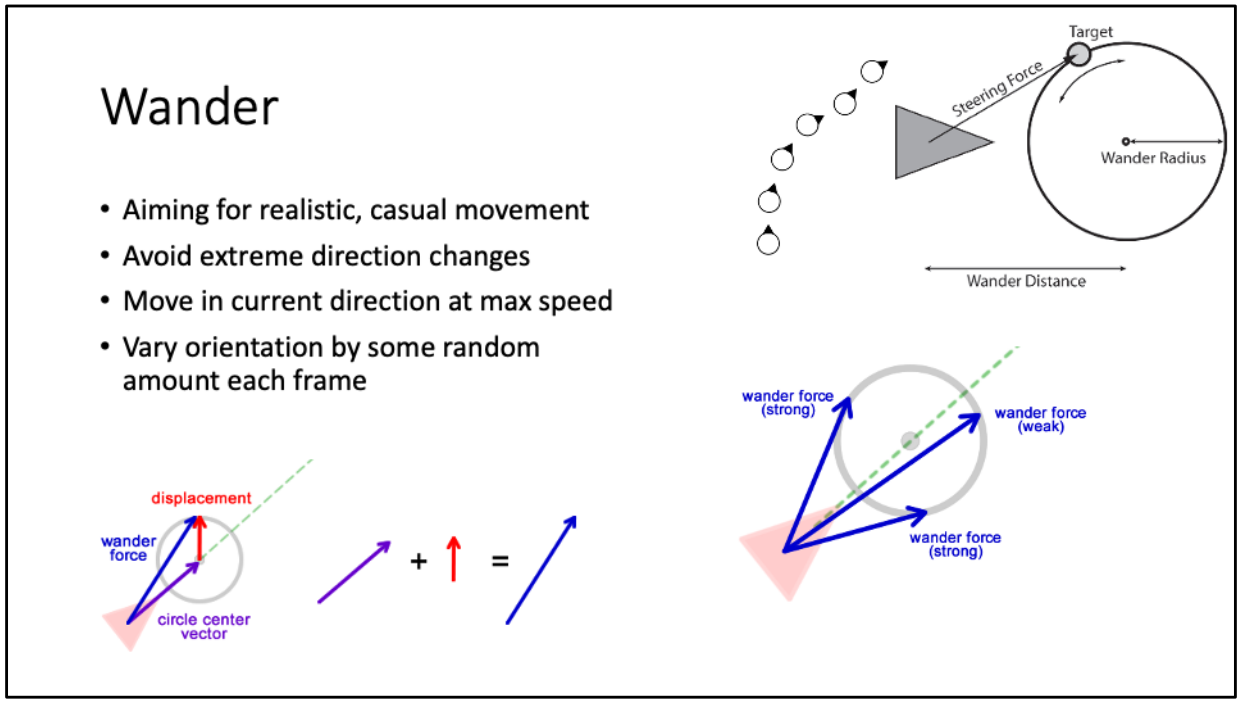


Enhancements to Chase

Anticipation

Build a model of user behavior



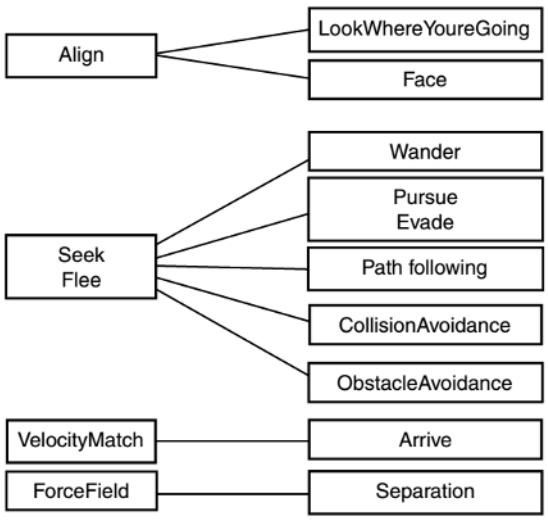


From Millington Fig 3.7 and Buckland Fig 3.4

And

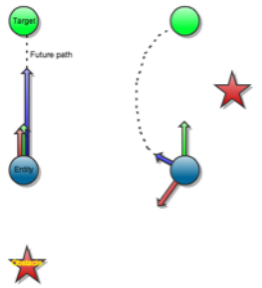
https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-wander--gamedev-1624

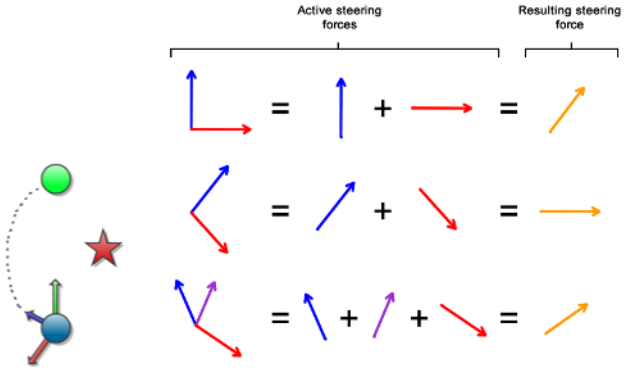
Steering Behavior Summary



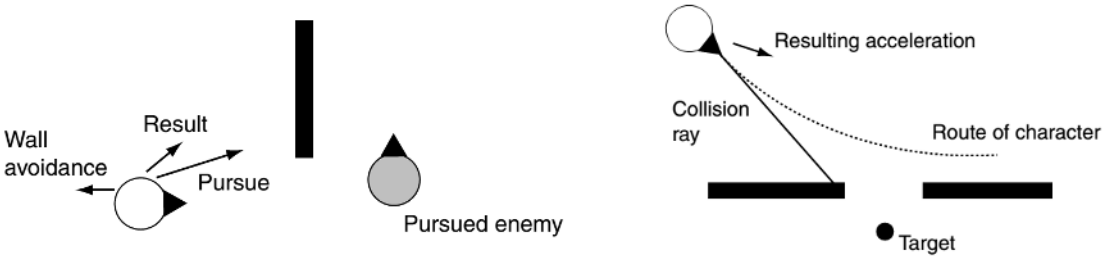
Multiple Steering Goals? Combining steering behaviors

- Sum
  - Need to enforce max speed
- Weighted sum
  - Weights can be based on some metric like distances, to favor one strategy over the other given situation



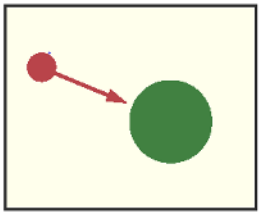


Combine Steering Behaviors Problem

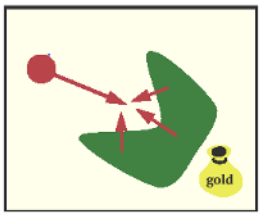


Combine Steering Behaviors: Problem

- What if steering behaviors are opposed?
- Zero vector!
- Or back and forth forever!
- Or orbiting!
- Need higher level control logic!



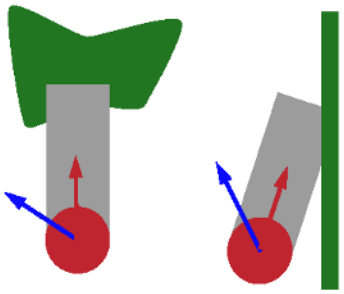
Exactly aligned



Forces balance out in dead end

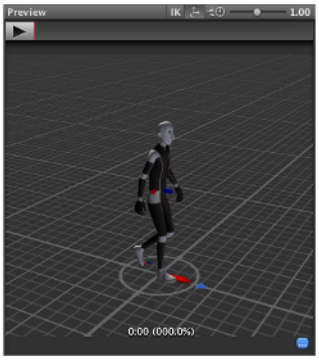
Steering Behaviors

- OpenSteer
- Pursue
- Evade
- Wander
- Obstacle Avoidance
- Wall/Path following
- Queuing
- http://www.red3d.com/cwr/steer/
- Combine behaviors with weights
- What could go wrong?



Root Motion

- Update game object position/rotation according to animation playback
- Common in games
- Can make it difficult to predict future positions (use animation averages)



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

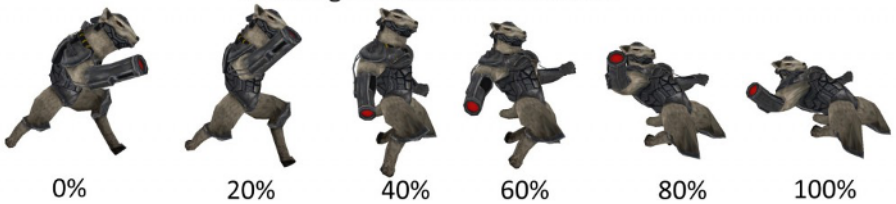
The Body Transform is the mass center of the character. It is used in Mecanim's retargeting engine and provides the most stable displacement model. The Body Orientation is an average of the lower and upper body orientation relative to the Avatar T-Pose.

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!**

blending run and slide animation



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

Motion Table for Blended Root Motion

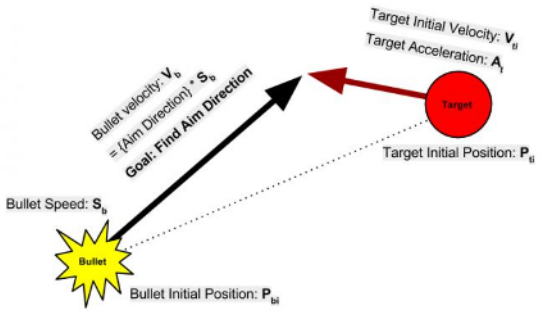
- Problem: How can agent select animation blend parameters that best direct its movement towards steering target?
- Could fit a function, but can become difficult for higher order functions. Or…
- Create a lookup table
- Table dims: Anim_param_x, Anim_param_y: [-1.0,1.0] (left/right, forward/bkwd)
- Choose some table resolution (e.g. epsilon=0.01)
- Store expected root motion (x_pos, y_pos, y_rot, etc.) diff from prev frame at avg framerate
- Use table to find anim params with closest root motion to steering target

Motion Table for Blended Root Motion

- Improvement: Add interpolation to table selection
- Could add more dimensions to motion table such as normalized time position within locomotion loop [0.0, 1.0] (beginning of loop to end). This is because many root motions vary over time

Ballistic Projectile Aiming

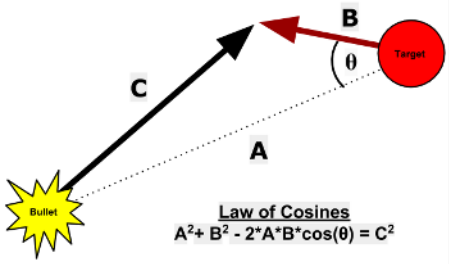
- Non instantaneous projectiles benefit from prediction
- Fast moving agents with slow turn rate may also benefit
- Directly solve (when possible... e.g. no accelerations)
- Iterative Techniques (e.g. like Newton's Method)



https://www.gamasutra.com/blogs/KainShin/20090515/83954/Predictive_Aim_Mathematics_for_AI_Targeting.php

Ballistic Projectile Aiming – Assuming zero acceleration

- Zero accel (constant V) simplifies problem
- **cos(theta) = DotProduct(Normalize(P_{bi} - P_{ti}), Normalize(V_t))**
- **D = Length(P_{ti} - P_{bi}) //represents initial distance from gun to target**
- **S_t = Length(V_t) //represents Target Speed**
- **t = [-2*D*S_t*cos(theta) ± Sqrt[(2*D*S_t*cos(theta))^2 + 4*(S_b^2 - S_t^2)*D^2]] / (2*(S_b^2 - S_t^2)) //application of quadratic formula**
- **Throw away negative or imaginary t, use smaller of the valid t's**
- Finally: **V_b = V_t + [(P_{ti} - P_{bi}) / t] //use good t**



Bullet Initial Position (i.e. muzzle position): P_{bi}
Bullet Speed (scalar constant, finite): S_b
Target Initial Position: P_{ti}
Target Velocity: V_t
Target Speed: S_t
Bullet Velocity Vector (Solving for): V_b

Gravity/Lobbed? Replace last equation with: **V_b = V_t - 0.5*A_b*t + [(P_{ti} - P_{bi}) / t]**
A_b is bullet accel (in this case just gravity)

Note that gravity/lobbed adjustment isn't completely ideal sometimes because it doesn't simultaneously adjust firing speed and angle. A bullet type projectile is reasonable, but a baseball player can adjust both speed and angle

Consider that the projectile is not affected by agent's speed for simplicity. Maybe fine for missiles, but perhaps not good for thrown projectiles

Ballistic Projectile: What about solving with target's acceleration, or other advanced models?

- Can sometimes solve directly but often challenging high order polynomials.
- Usually best to use an iterative approach

Ballistic Trajectory: Simple Iterative Approach

1. Assume target is not moving
2. Solve for ***t*** bullet intercept of 1.) [$t_0 = \text{Length}(P_{ti} - P_{bi}) / S_b$]
3. Plug in ***t*** for target's kinetic eqn: [$P_{t1} = P_{ti} + V_{ti} * t_0 + 0.5 * A_t * t_0^2$]
4. Now go to 2.) but replace target position with updated pos
5. Repeat until a terminating condition (like depth count)

- Downside: tends to favor shooting behind target. Not good for dramatic affect (e.g. player won't see shots missing in front of them)

Ballistic trajectory: with animated characters

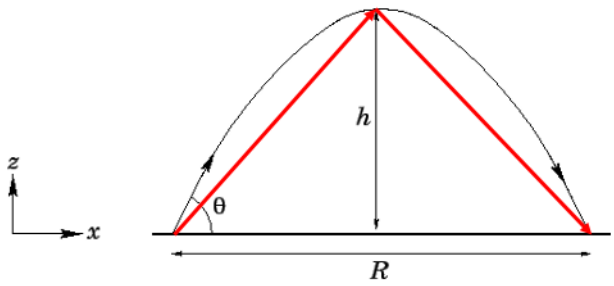
- Animation delay from projectile launch (especially pronounced with throwing motion)
- Make prediction based on animation duration till throw actually happens.
- Predict throw from predicted future character position and relative launch position (e.g. where is the hand at throw release point?)

Ballistic Trajectory: Resource Management

- Agent maybe should not waste valuable ammo. Or commit to long animation and be unable to throw at a better time
- Implement heuristic for throw decision. Don't throw if conditions are bad
- Consider:
  - Is target at top speed? No? Probably accelerating. Just wait a bit…
  - Target turning? Yes? Wait till going straight…
  - Is target predicted to hit edge of navmesh? Will likely turn. Lay off throwing…
  - Is projectile predicted to hit something sooner than target intercept time? Don't throw…

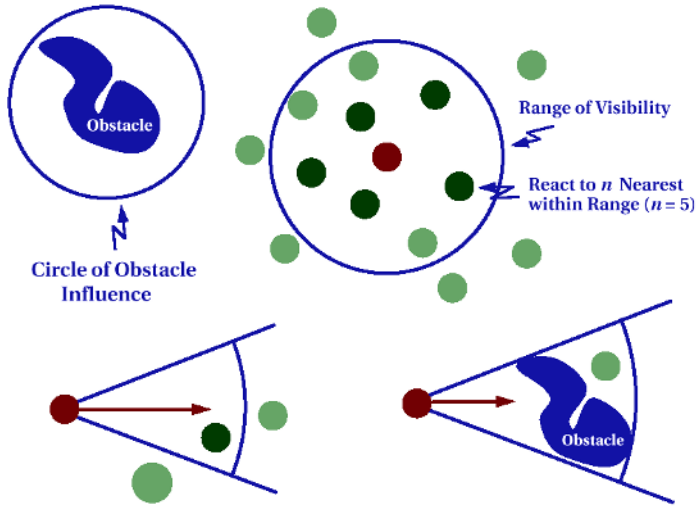
Ballistic Trajectory: Predicting lobbed projectile collisions

- Break parabola into line segments
- Raycast for each segment
- Can projectile bounce?
  - Maybe adjust aim lower and skip it, but consider friction slowing projectile down
- What if agent can control projectile speed and angle?



"Perceptual" Models

- Fairness?



38

Projectiles and Fairness

- Gaussian Distribution
- Variance
- Reaction time
- Fitt's Law Modeling