GAME AI

with Jeff Wilson, PhD

Ballistic Projectiles

Introduction

Ballistic Projectiles

- Sometimes agents need to fire projectiles at targets

Ballistic Projectiles

- It's easy to simulate Newtonian Physics with particle and rigid body dynamics

Ballistic Projectiles

- It's generally easy to simulate Newtonian Physics with particle dynamics and rigid body dynamics
- But solving for intercepts is not always easy…

Intercept

- Recall Steering Behavior Pursue
- An approximate intercept was calculated by estimating how much time it would take for the agent to get in the neighborhood of the target
- Using that time estimate, the target's movement could then be extrapolated assuming a constant velocity

Intercept

- Pursue's intercept time is only estimated because it is going to be updated again the next simulation frame and doesn't need to be precise
- However, with a ballistic projectile, there is only one chance to get the trajectory correct
- Therefore, more computational effort is needed for an accurate trajectory

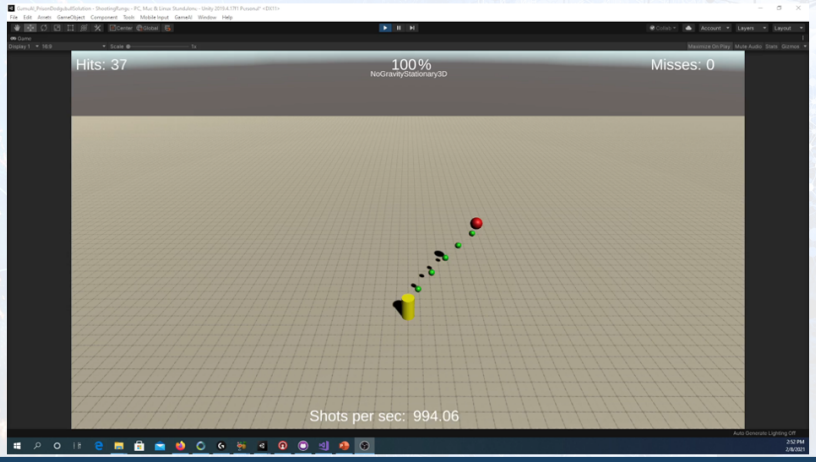
Intercept

- Projectiles like grenades or footballs aren't the only game objects that need a precise velocity solved
- Consider a game character like Rhino in Spider-Man
- Rhino might charge Spider-Man but not be able to change direction while charging
- If Spider-Man is running, Rhino's AI needs to calculate a precise intercept

Intercept

- Sometimes it's straightforward to solve an intercept
- For instance, if the target does not move at all and the projectile is constant velocity (not affected by gravity or drag) then only a relative position vector is needed to get launch direction (scale it by desired speed for full velocity)

Simple Intercept Trajectory



Ballistic Projectiles

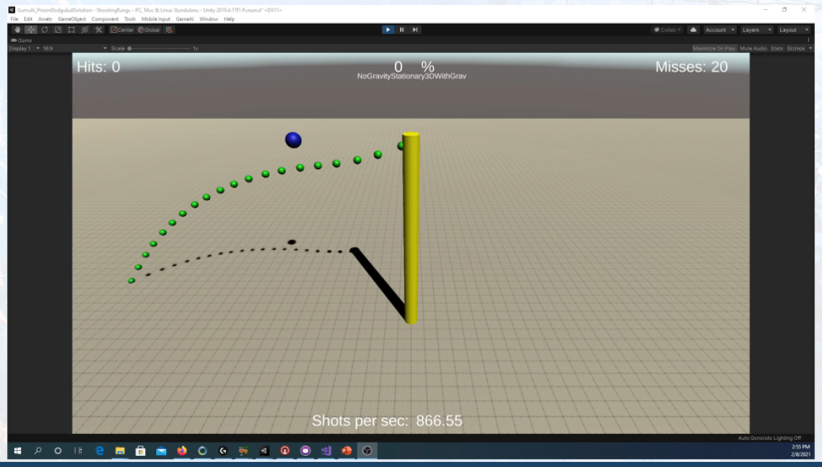
Georgia Tech

No Gravity

Intercept

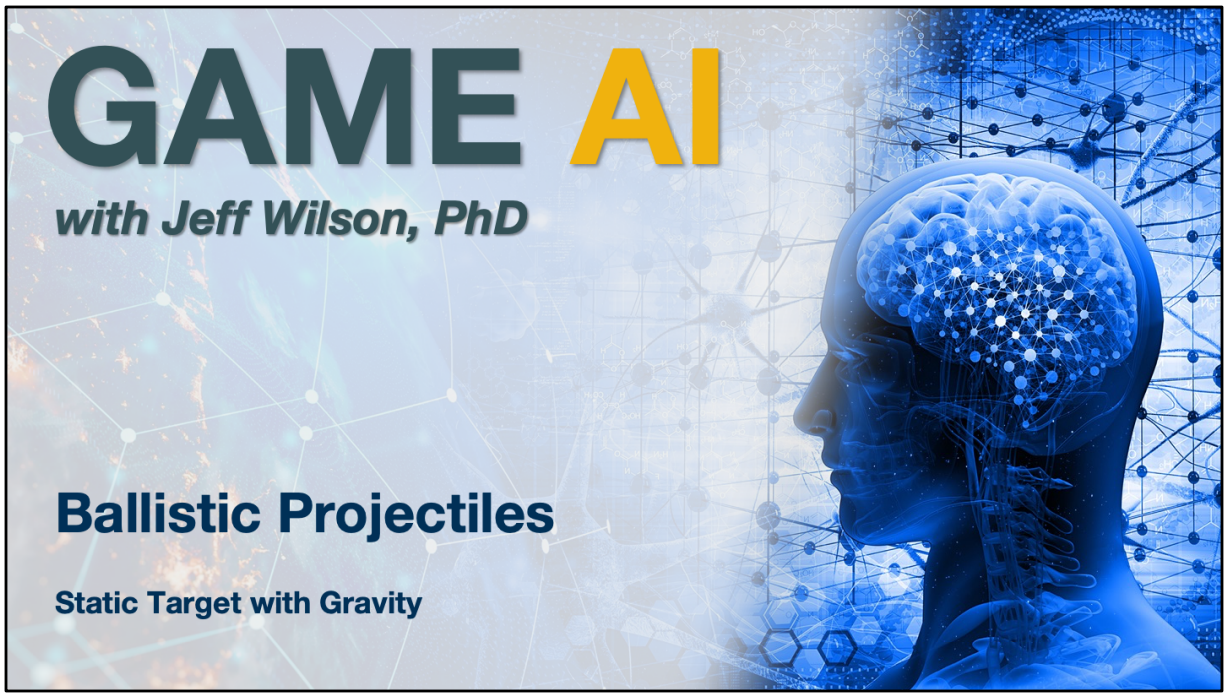
- But what if gravity affects the projectile? Or the target is moving?
- The more degrees of freedom allowed, the more complicated the system of equations needed solving gets
- Eventually, the equations can get so complicated that they cannot be directly solved, or become overly complicated or error prone

Simple Intercept Trajectory Fail



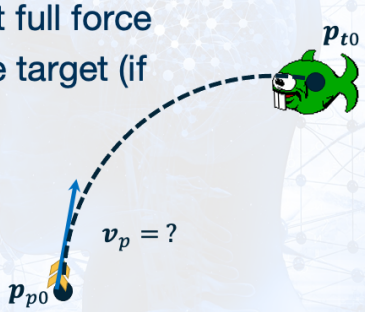
Methods

- There are two general methods for solving ballistic projectile launch velocities:
- Directly Solve – Formulate a system of equations that represents the problem and directly solve for the unknown variables
- Iteratively Solve – Estimate a reasonable solution, then refine parameters until the solution is close enough to the desired outcome.
- To iteratively solve, the movements of projectile and target must be able to be modeled according to the defined parameters


GAME AI
with Jeff Wilson, PhD
Ballistic Projectiles
Static Target with Gravity

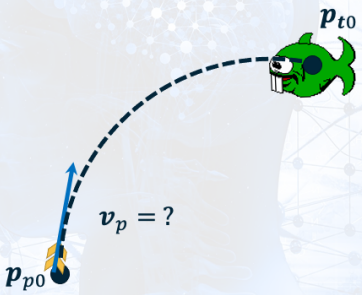
Ballistic Trajectory for Static Target

- Assume 3D positions of projectile and target
- Assume projectile is affected by gravity (but no drag)
- Assume projectile will always be launched at full force
- What is the direction of the velocity to hit the target (if possible)?



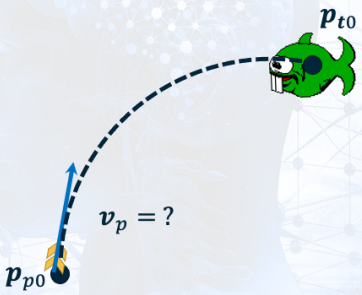
Ballistic Trajectory for Static Target

- $\boldsymbol{p}_t - \textit{Stationary Target Position}$
- $\boldsymbol{p}_{p0} - \textit{Initial Projectile Position}$
- $\boldsymbol{u}_{p0} - \textit{Initial (Launch) Projectile Direction}$
- $s_p - \textit{Initial (Launch)Projectile Speed}$
- $\boldsymbol{v}_p - \textit{Initial (Launch) Projectile Velocity}$
- $\boldsymbol{g}_p - \textit{Gravity Affecting Projectile}$
- $t_c - \textit{Time of collision}$
- Solving for $\boldsymbol{u}_{p0}$ because a full speed solution is desired. We only need the direction.



Ballistic Trajectory for Static Target

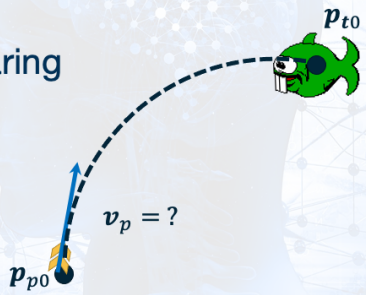
- From Particle Dynamics:
- $\boldsymbol{p}_t = \boldsymbol{p}_{p0} + \boldsymbol{u}_{p0} s_p t_c + \frac{1}{2}\boldsymbol{g}_p {t_c}^2$
- This represents three equations (3D):
- $p_{tx} = p_{p0x} + u_{p0x} s_p t_c + \frac{1}{2} g_{px} {t_c}^2$
- $p_{ty} = p_{p0y} + u_{p0y} s_p t_c + \frac{1}{2} g_{py} {t_c}^2$
- $p_{tz} = p_{p0z} + u_{p0z} s_p t_c + \frac{1}{2} g_{pz} {t_c}^2$



Unknowns are u_{p0x}, u_{p0y}, u_{p0z}, and t_c
So where is the fourth equation?

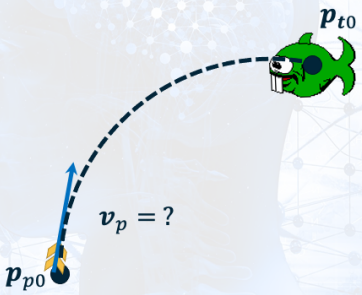
Ballistic Trajectory for Static Target

- A fourth equation can be added:
- $\|\boldsymbol{u}_{p0}\| = 1$
- $1 = {u_{p0x}}^2 + {u_{p0y}}^2 + {u_{p0z}}^2$
- Note that we got rid of Square Root by squaring both sides



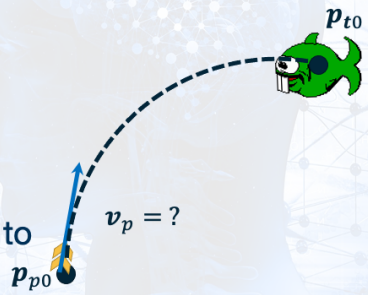
Ballistic Trajectory for Static Target

- With:
- $\boldsymbol{p}_t = \boldsymbol{p}_{p0} + \boldsymbol{u}_{p0} s_p t_c + \frac{1}{2}\boldsymbol{g}_p {t_c}^2$
- $\boldsymbol{u}_{p0}$ can be isolated
- $\boldsymbol{u}_{p0} = \frac{\boldsymbol{p}_t - \boldsymbol{p}_{p0} - \frac{1}{2}\boldsymbol{g}_p {t_c}^2}{s_p t_c}$
- Let $\Delta = \boldsymbol{p}_t - \boldsymbol{p}_{p0}$ so:
- $\boldsymbol{u}_{p0} = \frac{\Delta - \frac{1}{2}\boldsymbol{g}_p {t_c}^2}{s_p t_c}$



Ballistic Trajectory for Static Target

- Plug each dimension of $\boldsymbol{u}_{p0} = \frac{\Delta - \frac{1}{2}\boldsymbol{g}_p t_c^2}{s_p t_c}$ into:
- $1 = \|\boldsymbol{u}_{p0}\| = u_{p0x}^2 + u_{p0y}^2 + u_{p0z}^2$ giving terms:
- $u_{p0x}^2 = \left[\frac{\Delta_x - \frac{1}{2}g_{px}t_c^2}{s_p t_c}\right]^2$
- $u_{p0x}^2 = \frac{\Delta_x^2 - g_{px}\Delta_x t_c^2 + \frac{1}{4}g_{px}^2 t_c^4}{s_p^2 t_c^2}$
- Similarly solve for u_{p0y}^2 and u_{p0z}^2
- Note that vectors are losing dimensionality due to the magnitude operation



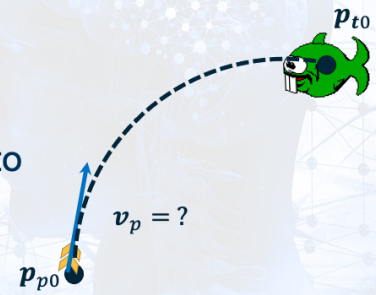
Ballistic Projectiles

Georgia Tech

Only showing the x term

Ballistic Trajectory for Static Target

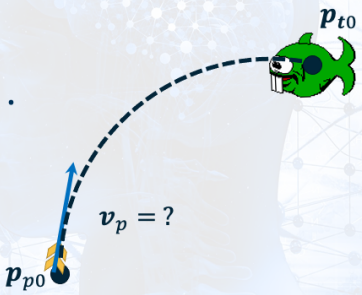
- Now:
- $1 = \frac{{\Delta_x}^2 - g_{px}\Delta_x {t_c}^2 + \frac{1}{4}{g_{px}}^2 {t_c}^4}{{s_p}^2 {t_c}^2} + \cdots$
- Move the denominator out:
- ${s_p}^2 {t_c}^2 = {\Delta_x}^2 - g_{px}{t_c}^2 + \frac{1}{4}{g_{px}}^2 {t_c}^4 + \cdots$
- We could stay in vector form, but we need to understand some vector operations…



Ballistic Trajectory for Static Target

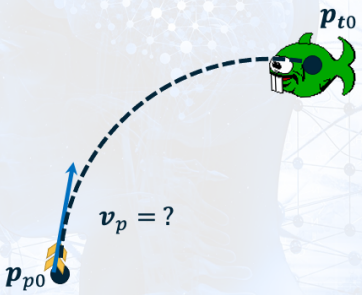
- We can see that multiplying two different Euclidean vectors (standard basis) together can only have non-zero terms along the Inner Product (similar to Dot Product):
- $\boldsymbol{r} \cdot \boldsymbol{s} = (r_0\hat{\boldsymbol{\imath}} + r_1\hat{\boldsymbol{\jmath}} + r_2\hat{\boldsymbol{k}})(s_0\hat{\boldsymbol{\imath}} + s_1\hat{\boldsymbol{\jmath}} + s_2\hat{\boldsymbol{k}})$
- $= (r_0\hat{\boldsymbol{\imath}} \cdot s_0\hat{\boldsymbol{\imath}}) + (r_1\hat{\boldsymbol{\jmath}} \cdot s_1\hat{\boldsymbol{\jmath}}) + (r_2\hat{\boldsymbol{k}} \cdot s_2\hat{\boldsymbol{k}}) + (r_0\hat{\boldsymbol{\imath}} \cdot s_2\hat{\boldsymbol{k}}) + (s_0\hat{\boldsymbol{\imath}} \cdot r_1\hat{\boldsymbol{k}}) + (r_1\hat{\boldsymbol{\jmath}} \cdot s_2\hat{\boldsymbol{k}}) + (s_1\hat{\boldsymbol{\jmath}} \cdot r_2\hat{\boldsymbol{k}}) + (r_0\hat{\boldsymbol{\imath}} \cdot s_1\hat{\boldsymbol{\jmath}}) + (s_0\hat{\boldsymbol{\imath}} \cdot r_1\hat{\boldsymbol{\jmath}})$
- $= r_0s_0\hat{\boldsymbol{\imath}} \cdot \hat{\boldsymbol{\imath}} + r_1s_1\hat{\boldsymbol{\jmath}} \cdot \hat{\boldsymbol{\jmath}} + r_2s_2\hat{\boldsymbol{k}} \cdot \hat{\boldsymbol{k}} = r_0s_0\hat{\boldsymbol{\imath}} + r_1s_1\hat{\boldsymbol{\jmath}} + r_2s_2\,\hat{\boldsymbol{k}}$
- $\boldsymbol{r} \cdot \boldsymbol{s} = r_0s_0 + r_1s_1 + r_2s_2$ (Dot Product)

The dot product of a vector with itself is the square of the vector's magnitude. Applied to unit vectors, the dot product is always 1.



Ballistic Trajectory for Static Target

- For Euclidean Vectors: $\boldsymbol{r} - is\ a\ 3D\ Vector$
- $\boldsymbol{r} \cdot \boldsymbol{r} = (r_0\hat{\boldsymbol{\imath}} + r_1\hat{\boldsymbol{\jmath}} + r_2\hat{\boldsymbol{k}})(r_0\hat{\boldsymbol{\imath}} + r_1\hat{\boldsymbol{\jmath}} + r_2\hat{\boldsymbol{k}})$
- $= (r_0\hat{\boldsymbol{\imath}} \cdot r_0\hat{\boldsymbol{\imath}}) + (r_1\hat{\boldsymbol{\jmath}} \cdot r_1\hat{\boldsymbol{\jmath}}) + (r_2\hat{\boldsymbol{k}} \cdot r_2\hat{\boldsymbol{k}}) + 2(r_0\hat{\boldsymbol{\imath}} \cdot r_2\hat{\boldsymbol{k}}) + 2(r_1\hat{\boldsymbol{\jmath}} \cdot r_2\hat{\boldsymbol{k}}) + 2(r_0\hat{\boldsymbol{\imath}} \cdot r_1\hat{\boldsymbol{\jmath}})$
- $= (r_0\hat{\boldsymbol{\imath}} \cdot r_0\hat{\boldsymbol{\imath}}) + (r_1\hat{\boldsymbol{\jmath}} \cdot r_1\hat{\boldsymbol{\jmath}}) + (r_2\hat{\boldsymbol{k}} \cdot r_2\hat{\boldsymbol{k}})$

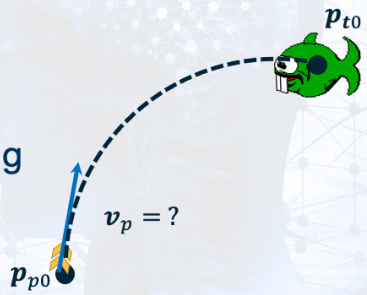


Georgia Tech

Basis vectors are orthonormal unit vectors

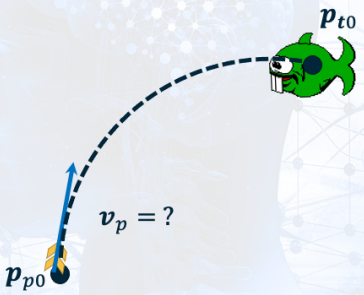
Ballistic Trajectory for Static Target

- So, $\boldsymbol{r} \cdot \boldsymbol{r} = {r_0}^2 \hat{\boldsymbol{\imath}} \cdot \hat{\boldsymbol{\imath}} + {r_1}^2 \hat{\boldsymbol{\jmath}} \cdot \hat{\boldsymbol{\jmath}} + {r_2}^2 \hat{\boldsymbol{k}} \cdot \hat{\boldsymbol{k}} = {r_0}^2 \hat{\boldsymbol{\imath}} + {r_1}^2 \hat{\boldsymbol{\jmath}} + {r_2}^2 \hat{\boldsymbol{k}} = {r_0}^2 \hat{\boldsymbol{\imath}} + {r_1}^2 \hat{\boldsymbol{\jmath}} + {r_2}^2 \hat{\boldsymbol{k}} = {r_0}^2 + {r_1}^2 + {r_2}^2$
- But we know the magnitude of $\boldsymbol{r}$ is:
- $\|\boldsymbol{r}\| = \sqrt{{r_0}^2 + {r_1}^2 + {r_2}^2}$
- So, $\boldsymbol{r} \cdot \boldsymbol{r} = \|\boldsymbol{r}\|^2 = {r_0}^2 + {r_1}^2 + {r_2}^2$
- Now we can handle multiplying and squaring vectors in our system of equations



Ballistic Trajectory for Static Target

- Now we can go back and simplify:
- $s_p{}^2 t_c{}^2 = \Delta_x{}^2 - g_{px}\Delta_x t_c{}^2 + \frac{1}{4} g_{px}{}^2 t_c{}^4 + \cdots$
- To:
- $s_p{}^2 t_c{}^2 = \|\Delta\|^2 - \boldsymbol{g}_p \cdot \Delta t_c{}^2 + \frac{1}{4}\|\boldsymbol{g}_p\|^2 t_c{}^4$
- And solve for 0:
- $0 = \|\Delta\|^2 - \boldsymbol{g}_p \cdot \Delta t_c{}^2 - s_p{}^2 t_c{}^2 + \frac{1}{4}\|\boldsymbol{g}_p\|^2 t_c{}^4$

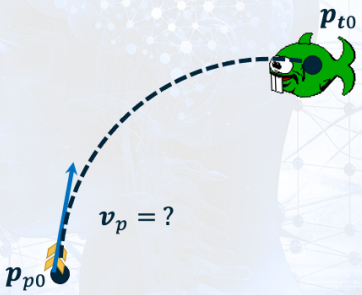


Ballistic Trajectory for Static Target

- Simplify:
- $0 = \frac{1}{4}\|\boldsymbol{g}_p\|^2 {t_c}^4 - (\boldsymbol{g}_p \cdot \Delta + {s_p}^2){t_c}^2 + \|\Delta\|^2$
- Quadratic formula is now in play solving for ${t_c}^2$
- ${t_c}^2 = \frac{\boldsymbol{g}_p \cdot \Delta + {s_p}^2 \pm \sqrt{(\boldsymbol{g}_p \cdot \Delta + {s_p}^2)^2 - \|\boldsymbol{g}_p\|^2 \|\Delta\|^2}}{\frac{1}{2}\|\boldsymbol{g}_p\|^2}$
- Then:
- $t_c = \sqrt{\frac{\boldsymbol{g}_p \cdot \Delta + {s_p}^2 \pm \sqrt{(\boldsymbol{g}_p \cdot \Delta + {s_p}^2)^2 - \|\boldsymbol{g}_p\|^2 \|\Delta\|^2}}{\frac{1}{2}\|\boldsymbol{g}_p\|^2}}$

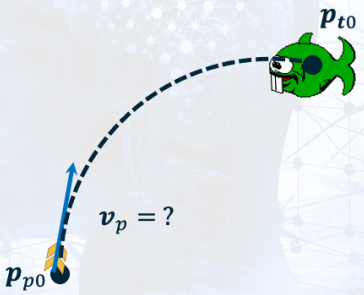
Quadratic Formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
$$when\ ax^2 + bx + c = 0$$



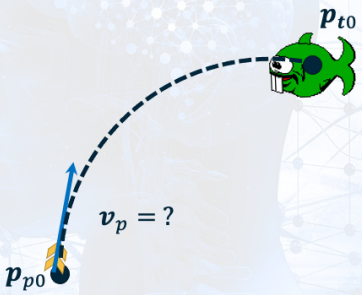
Ballistic Trajectory for Static Target

- If inner radicand is less than zero, there is no solution
- If either candidate t_c is negative, discard it.
- If no positive values are left, there is no solution.
- If both are positive, take the smaller one.
- Otherwise, return the one valid solution.
- In code implementation, do not repeat the same computation more than once. Cache common expressions in variables to reuse.

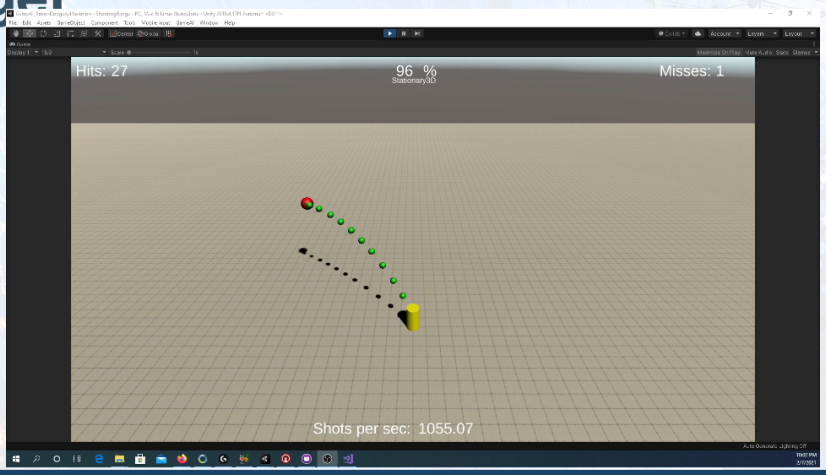


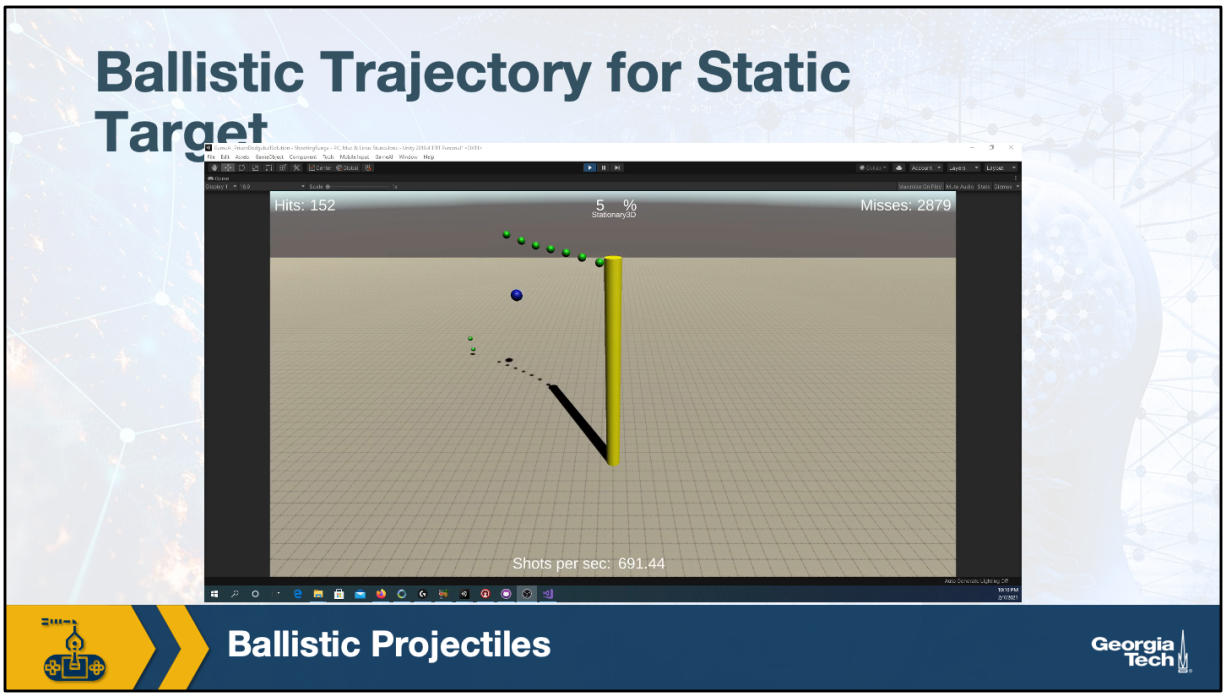
Ballistic Trajectory for Static Target

- Using this previously defined projectile motion equation, solve for $\boldsymbol{u}_{p0}$ with the found t_c and return the result:
- $\boldsymbol{p}_t = \boldsymbol{p}_{p0} + \boldsymbol{u}_{p0} s_p t_c + \frac{1}{2} \boldsymbol{g}_p {t_c}^2$
- $\boldsymbol{u}_{p0} = \frac{\boldsymbol{p}_t - \boldsymbol{p}_{p0} - \frac{1}{2} \boldsymbol{g}_p {t_c}^2}{s_p t_c}$
- $\boldsymbol{u}_{p0} = \frac{2\Delta - \boldsymbol{g}_p {t_c}^2}{2 s_p t_c}$



Ballistic Trajectory for Static Target

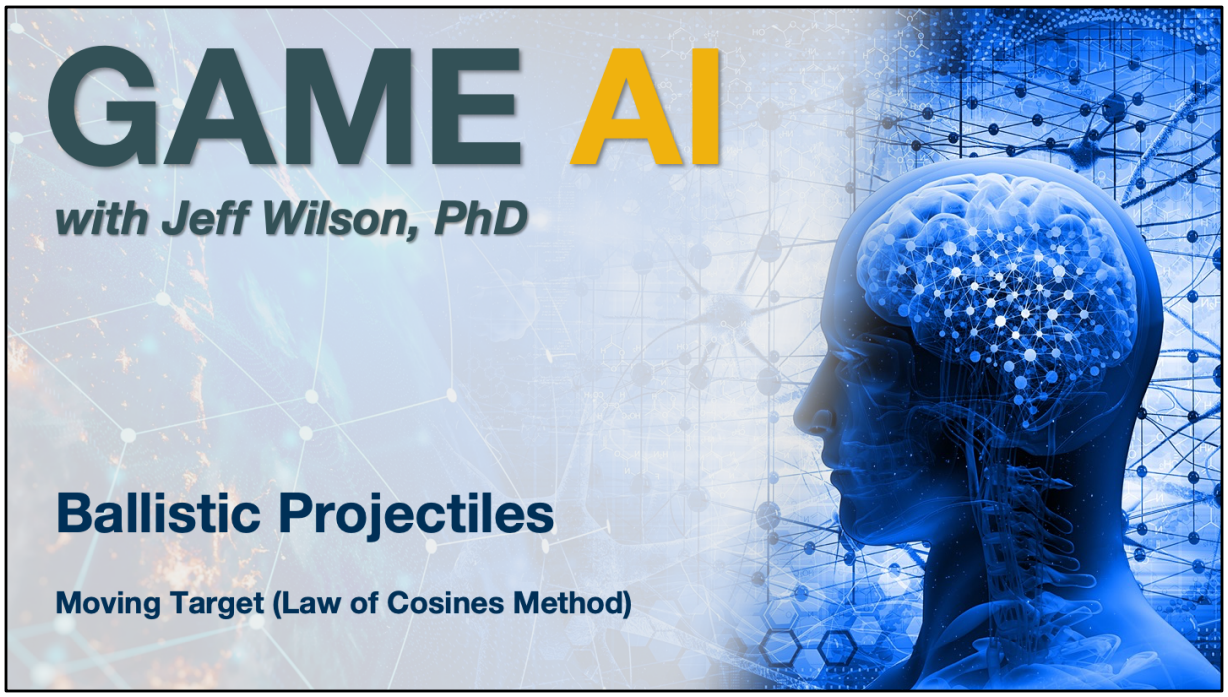




Can’t hit a fast moving target though

Ballistic Trajectory for Static Target

- Static Target Approach
- Pros:
  - Solves directly for a 3D target position with gravity affecting projectile
- Cons:
  - Does not handle moving targets!

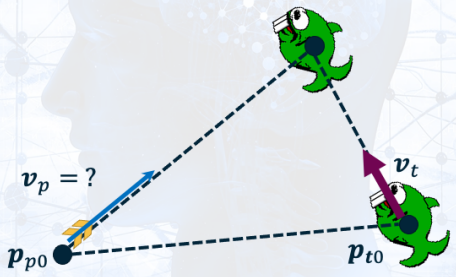

GAME AI
with Jeff Wilson, PhD
Ballistic Projectiles
Moving Target (Law of Cosines Method)

Moving Target

- The static target solution works well for that purpose, but what about moving targets?
- We might consider an iterative solution that leverages the static target method
- But first, let's consider solving for a moving target

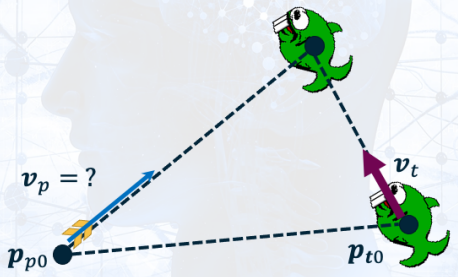
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- Assume top-down (XZ plane) with 2D vectors
  - This is necessary to reduce complexity so that the equations are still solvable
- Target and Projectile move with constant velocity
- But for projectile, only the speed is known
- We will also see how to adapt the 2D solution to 3D, possibly with gravity



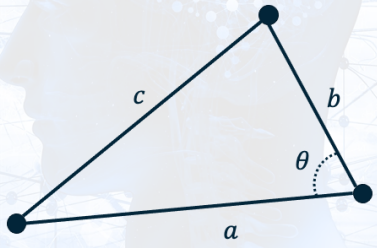
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- $\boldsymbol{p}_{t0} -$ *Initial Target Position*
- $\boldsymbol{p}_{p0} -$ *Initial Projectile Position*
- $\boldsymbol{v}_t -$ *Target Velocity*
- $s_t -$ *Target Speed*
- $s_p -$ *Projectile Speed*
- $\boldsymbol{v}_p -$ *Projectile Velocity*
- $t_c -$ *Time of collision*



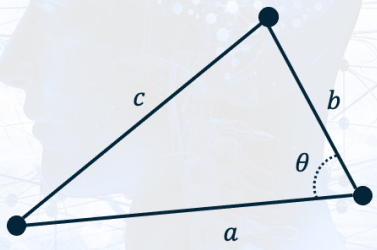
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- **Law of Cosines:**
- $a^2 + b^2 - 2ab\cos\theta = c^2$



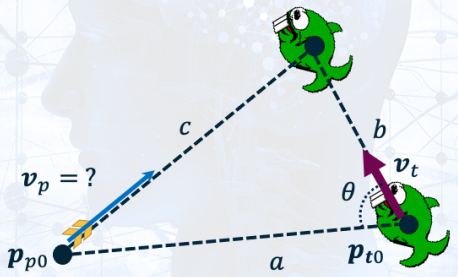
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- **Law of Cosines:**
- $a^2 + b^2 - 2ab\cos\theta = c^2$
- **Dot Product:**
- $\boldsymbol{a} \cdot \boldsymbol{b} = \|\boldsymbol{a}\|\|\boldsymbol{b}\|\cos\theta$
- $\hat{\boldsymbol{a}} \cdot \hat{\boldsymbol{b}} = \cos\theta$



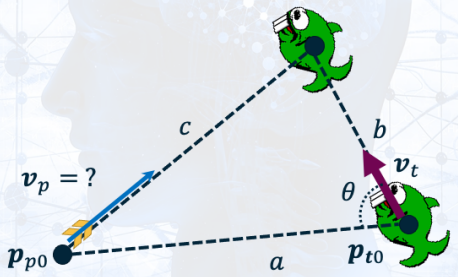
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- $\boldsymbol{a} = \boldsymbol{p}_{p0} - \boldsymbol{p}_{t0}$
- $a = \|\boldsymbol{a}\|$
- $s_t = \|\boldsymbol{v}_t\|$
- $\cos\theta = \hat{\boldsymbol{a}} \cdot \hat{\boldsymbol{v}}_t$



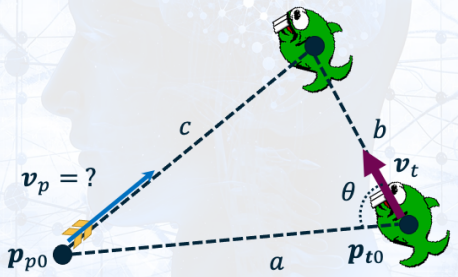
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- **Displacement by Velocity:**
- $\boldsymbol{d} = \boldsymbol{v}t$
- $\|\boldsymbol{d}\| = \|\boldsymbol{v}t\|$
- $d = \|\boldsymbol{v}t\|$
- $d = \|\boldsymbol{v}\|t$
- $b = s_t t_c$
- $c = s_p t_c$



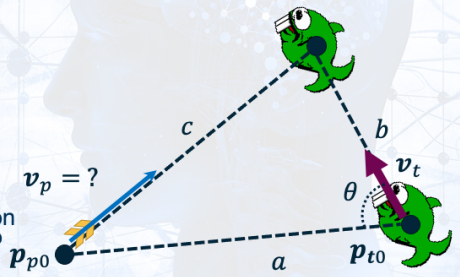
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- Plug $a, b, \& c$ into Law of Cosines
- $a^2 + (s_t t)^2 - 2as_t t\hat{\boldsymbol{a}} \cdot \hat{\boldsymbol{b}} = (s_p t)^2$
- Simplify
- ${s_p}^2 t^2 - {s_t}^2 t^2 + 2as_t t\hat{\boldsymbol{a}} \cdot \hat{\boldsymbol{b}} - a^2 = 0$
- $({s_p}^2 - {s_t}^2)t^2 + (2as_t\hat{\boldsymbol{a}} \cdot \hat{\boldsymbol{b}})t - a^2 = 0$



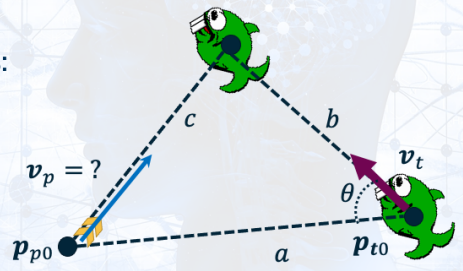
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- $(s_p{}^2 - s_t{}^2)t_c{}^2 + (2as_t\hat{\boldsymbol{a}} \cdot \hat{\boldsymbol{b}})t_c - a^2 = 0$
- Apply Quadratic Formula
- $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ *when* $ax^2 + bx + c = 0$
- $t_c = \frac{-(2as_t\hat{\boldsymbol{a}} \cdot \hat{\boldsymbol{b}}) \pm \sqrt{(2as_t\hat{\boldsymbol{a}} \cdot \hat{\boldsymbol{b}})^2 + 4(s_p{}^2 - s_t{}^2)a^2}}{2(s_p{}^2 - s_t{}^2)}$
- Test radicand for negative, and denominator for zero. Only use a positive t. User smaller of two valid t's.
- In code implementation, do not repeat the same computation more than once. Cache common expressions in variables to reuse.



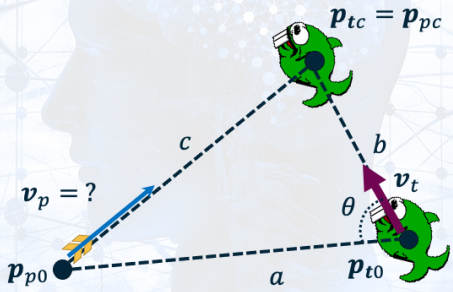
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- If denominator is zero, a solution may still be possible
- The zero occurs if s_p and s_t are the same
- If that is the case, we know that c and b of Law of Cosines equation are the same, so:
- $a^2 + b^2 - 2ab\cos\theta = b^2$ and solve for b:
- $b = \frac{a}{2\cos\theta}$ and then plug in known trajectory terms:
- $s_t t_c = \frac{\|\boldsymbol{a}\|}{2\hat{\boldsymbol{a}}\cdot\hat{\boldsymbol{v}}_t}$, where $\mathbf{a} = \boldsymbol{p}_{p0} - \boldsymbol{p}_{t0}$ and solve for t_c:
- $t_c = \frac{\|\boldsymbol{a}\|}{2s_t\hat{\boldsymbol{a}}\cdot\hat{\boldsymbol{v}}_t}$, and only valid if $\hat{\boldsymbol{a}}\cdot\hat{\boldsymbol{v}}_t > 0$



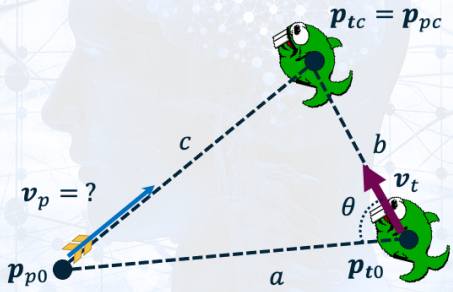
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- Now have time(s) of collision, still need to solve $\boldsymbol{v_p}$
- $\boldsymbol{p_{tc}}$ $-$ *Target position at collision*
- $\boldsymbol{p_{pc}}$ $-$ *Projectile position at collision*
- $\boldsymbol{p_{tc}} = \boldsymbol{p_{pc}}$
- $\boldsymbol{p_{tc}} = \boldsymbol{p_{t0}} + \boldsymbol{v_t} * t_c$ (particle dynamics)
- $\boldsymbol{v_p} = \frac{(\boldsymbol{p_{pc}} - \boldsymbol{p_{p0}})}{t_c} = \frac{\left((\boldsymbol{p_{t0}} + \boldsymbol{v_t} * t_c) - \boldsymbol{p_{p0}}\right)}{t_c}$
- **Finally:** $\boldsymbol{v_p} = \frac{(\boldsymbol{p_{t0}} - \boldsymbol{p_{p0}})}{t_c} + \boldsymbol{v_t}$



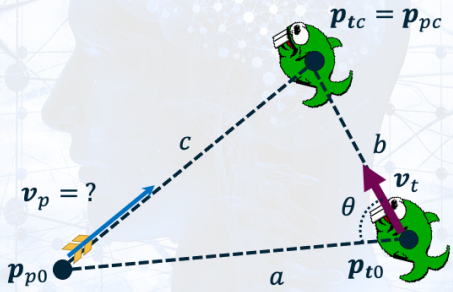
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- If the game is actually 3D, then $\boldsymbol{v}_p$ can be solved with 3D vectors and still get a collision
- However, $\|\boldsymbol{v}_p\| \geq S_p$ (y component adds speed)
- This may or may not be acceptable for gameplay
- To be addressed later…



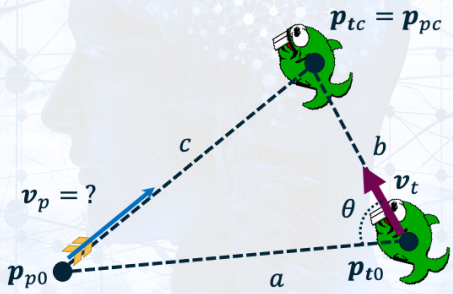
Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- What if the projectile is lobbed in 3D with the effects of gravity?
- Particle Dynamics: $\boldsymbol{p} = \boldsymbol{p}_0 + \boldsymbol{v}_0 t + \frac{1}{2}\boldsymbol{a}t^2$
- $\boldsymbol{a}_p$ - Gravity vector (with sign)
- $\boldsymbol{p}_{pc} = \boldsymbol{p}_{p0} + \boldsymbol{v}_p t_c + \frac{1}{2}\boldsymbol{a}_p {t_c}^2$
- $\boldsymbol{p}_{t0} + \boldsymbol{v}_t * t_c = \boldsymbol{p}_{p0} + \boldsymbol{v}_p t_c + \frac{1}{2}\boldsymbol{a}_p {t_c}^2$
- $\boldsymbol{v}_p = \frac{\boldsymbol{p}_{t0} - \boldsymbol{p}_{p0}}{t_c} + \boldsymbol{v}_t - \frac{1}{2}\boldsymbol{a}_p t_c$

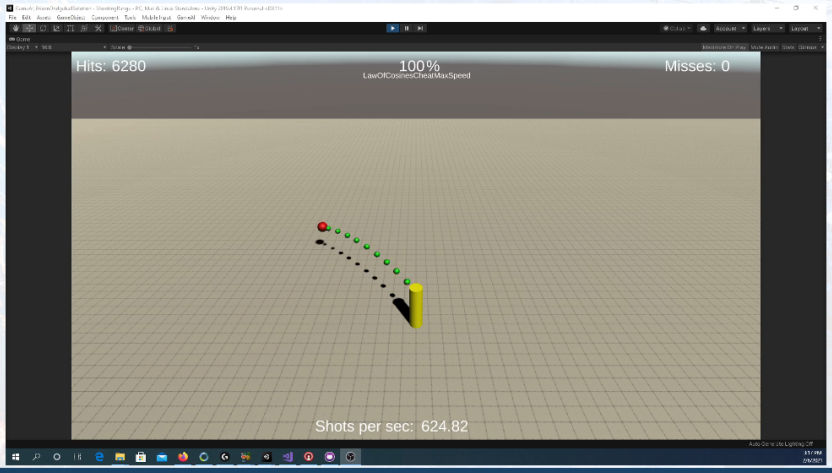


Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

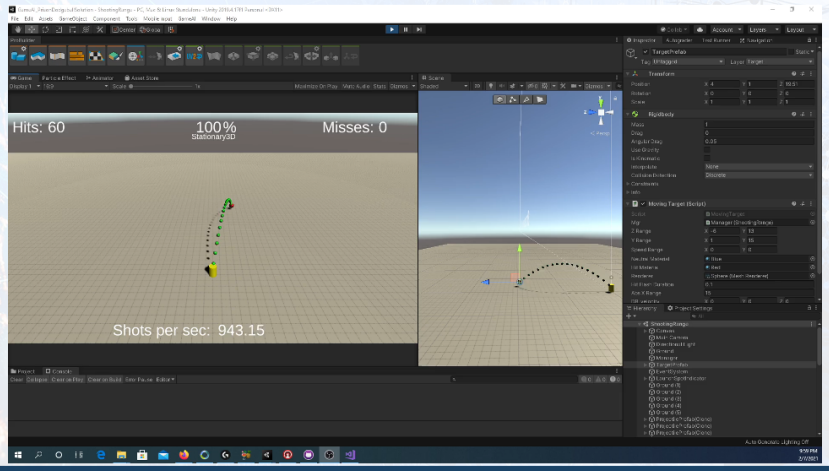
- Note that the lobbed projectile has the same problem with $\|\boldsymbol{v}_p\| \geq S_p$ (y component adds speed)
- Again, this may or may not be acceptable for gameplay
- Could separately enforce max intercept time or distance to intercept



Law of Cosines Approach

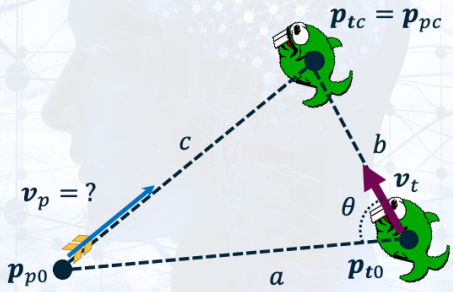


Law of Cosines Cheat Problem



Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- Law of Cosines Approach
- Pros:
  - Solves directly for a moving target in 2D
  - Can be made to work in 3D with limitations
- Cons:
  - Does not guarantee a max speed in 3D
  - Not ideal for extreme height differences or target moving significantly in the *y* direction
  - Not ideal for flight sims or space combat

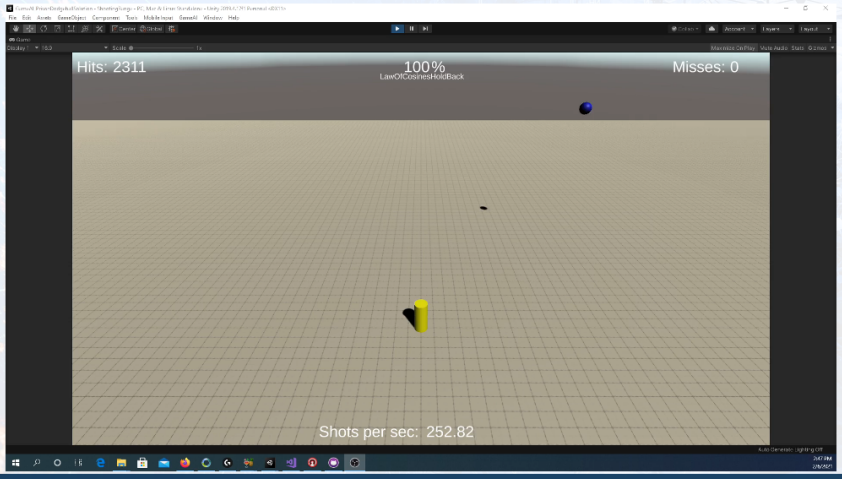


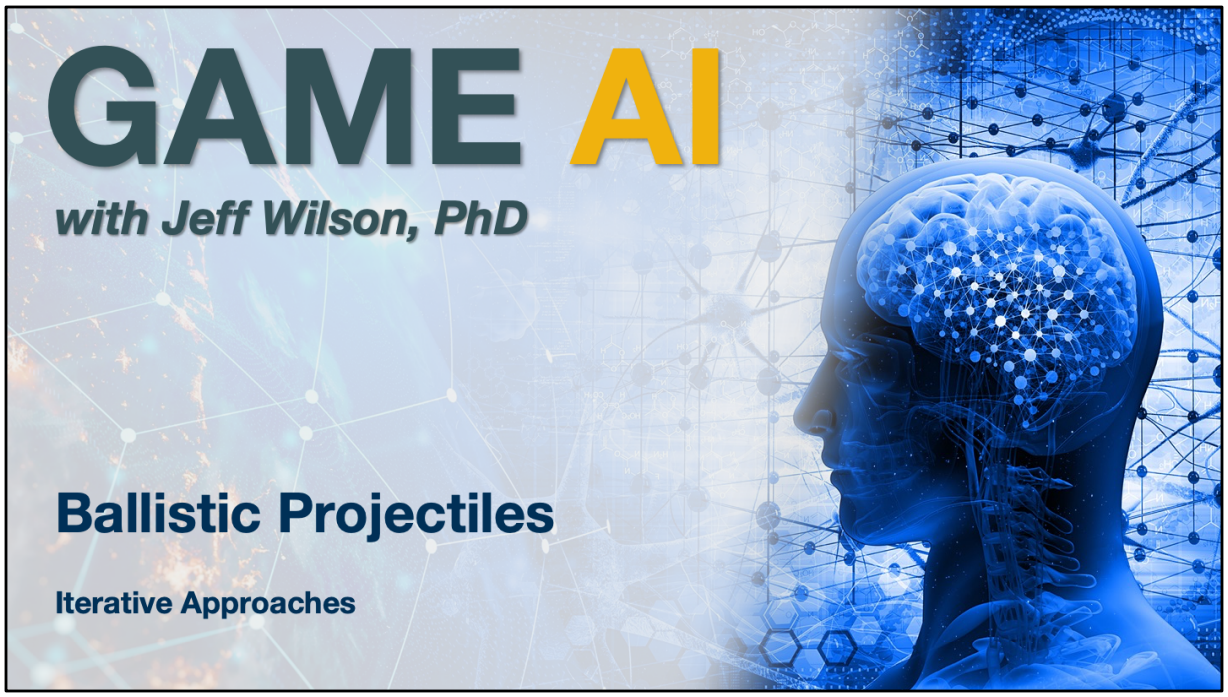
Could probably still be used in a flight/space sim if the algorithm selected the best axis aligned plane to use in any give circumstance

Ballistic Trajectory Intercept for Constant Velocity Projectile and Target

- If Law of Cosines is used but the projectile speed limit must be imposed on a 3D or 3D Lobbed launch, then one approach is to replace S_p with $S_{holdback}$, which is reduced by a Holdback percentage of S_p (like 90%)
- This leaves a fraction of total speed that can be applied to the y direction
- If solved successfully, check to see if $\|\boldsymbol{v}_p\| > S_p$
- If it is, fail to launch (or launch with capped speed knowing the projectile will not hit)
- Pros:
  - Often works when initial projectile position and target stay at close to the same elevation
  - Tendency to screen low probability launches (e.g., close to 45° launch angle)
- Cons:
  - Can fail to find a solution when one is possible
  - Can result in a projectile traveling at less than full speed

Law of Cosines Holdback




GAME AI
with Jeff Wilson, PhD
Ballistic Projectiles
Iterative Approaches

Ballistic Projectile: What about accelerating targets, or other advanced scenarios?

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

Ballistic Trajectory: Simple Iterative Approach for Static Target Solver

- The method for solving a ballistic trajectory for a static target can be used to aim at moving targets
- An iterative method can leverage the static target solver

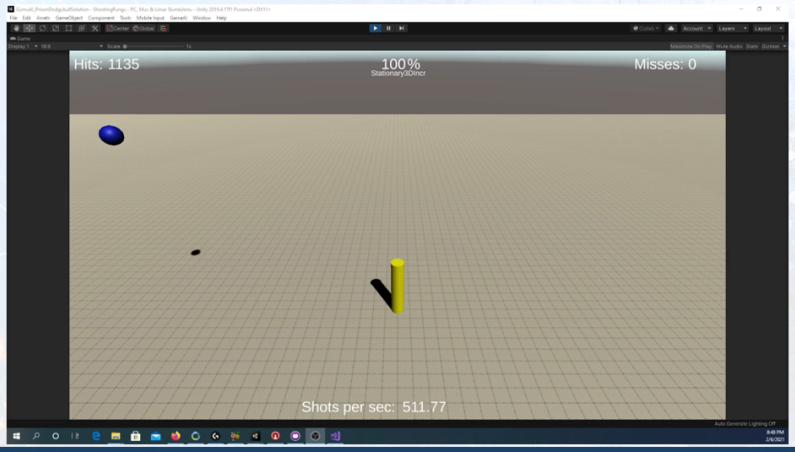
Ballistic Trajectory: Simple Iterative Approach for Static Target Solver

1. Assume target is not moving
2. Solve for t_n projectile intercept of target in step 1.
   - Perhaps use static 3D target method
3. Plug in t_0 for target's kinematic eqn (must be solvable):
   - $\boldsymbol{p}_{tn} = \boldsymbol{p}_{t0} + \boldsymbol{v}_{t0} t_n + \frac{1}{2} \boldsymbol{a}_t {t_n}^2$ (possible equation)
4. Now go to 2. but replace target position with $\boldsymbol{p}_{tn}$
5. Repeat until a terminating condition (like depth count or minimal change in positions, etc.)

Ballistic Trajectory: Simple Iterative Approach for Static Target Solver

- Cap the number of iterations (around 3 to 6)
- Preserve previous valid to return if future iterations fail to refine
- Fail immediately if first attempt fails
- Can perform a sanity check on final intercept time by plugging into projectile and target kinematic equations to determine if the iterative estimate is within some minimum distance
- This has added bonus of allowing for shots that are likely to get close enough to a target game object but not necessary hit the center

Static 3D Target with Iterative Refinement



Ballistic Projectiles

Georgia Tech

Ballistic Trajectory: Simple Iterative Approach for Static Target Solver

- **Pros**
  - Easy to implement
- **Cons**
  - May not be able to refine to a stable solution, especially around the point of closest approach of the target, or with higher derivatives of motion
  - Tends to favor shooting behind target. Not good for dramatic affect (e.g., FPS player won't see enemy shots missing)
  - Target that is initially unreachable, but will be reachable in the future, creates implementation difficulties
  - Doesn't work if kinematic movement of projectile and/or target cannot be directly evaluated (drag and other scenarios)

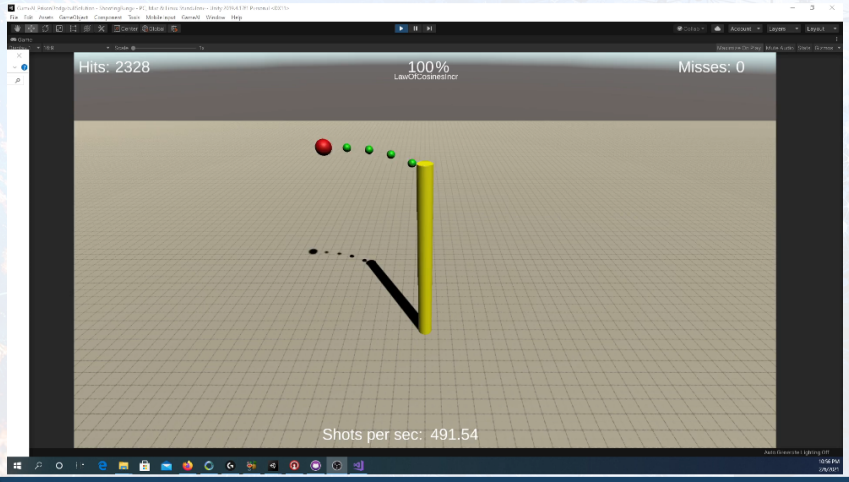
2D Law of Cosines Iterative Algorithm

- An iterative algorithm is also possible with Law of Cosines to enforce a max speed
- Start with $S_{holdback} = S_p$ This sets the target speed in the *XZ* plane
- If a solution is found, test for $\|\boldsymbol{v}_p\| > S_p$ (overall speed too fast?)
- If too fast, try again with an $S_{holdback}$ adjusted to be halfway closer to length of optimal launch (45 degrees)
- Otherwise, check if $\|\boldsymbol{v}_p\| \geq (1-\varepsilon)S_p$ AND $S_{holdback} \geq S_{XZAtOptimalAngle}$
- If it is, it's good enough. Immediately exit with result
- If too slow, try again with: $S_{holdback} = S_{prevHB} + \frac{1}{2}(S_p - S_{prevHB})$
- If the angle is high, try again with an $S_{holdback}$ adjusted to be halfway closer to length of optimal launch (45 degrees)

2D Law of Cosines Iterative Algorithm

- Cap the number of iterations (around 3 to 6)
- Preserve previous best valid result (fastest in *XZ* plane and within overall speed limit) to return if future iterations fail to refine
- Fail immediately if first attempt fails
- *XZ* vector length for optimal (45 deg.) launch can be solved with Pythagorean Theorem for 45-45-90 triangle

Law of Cosines Iterative



Ballistic Projectiles

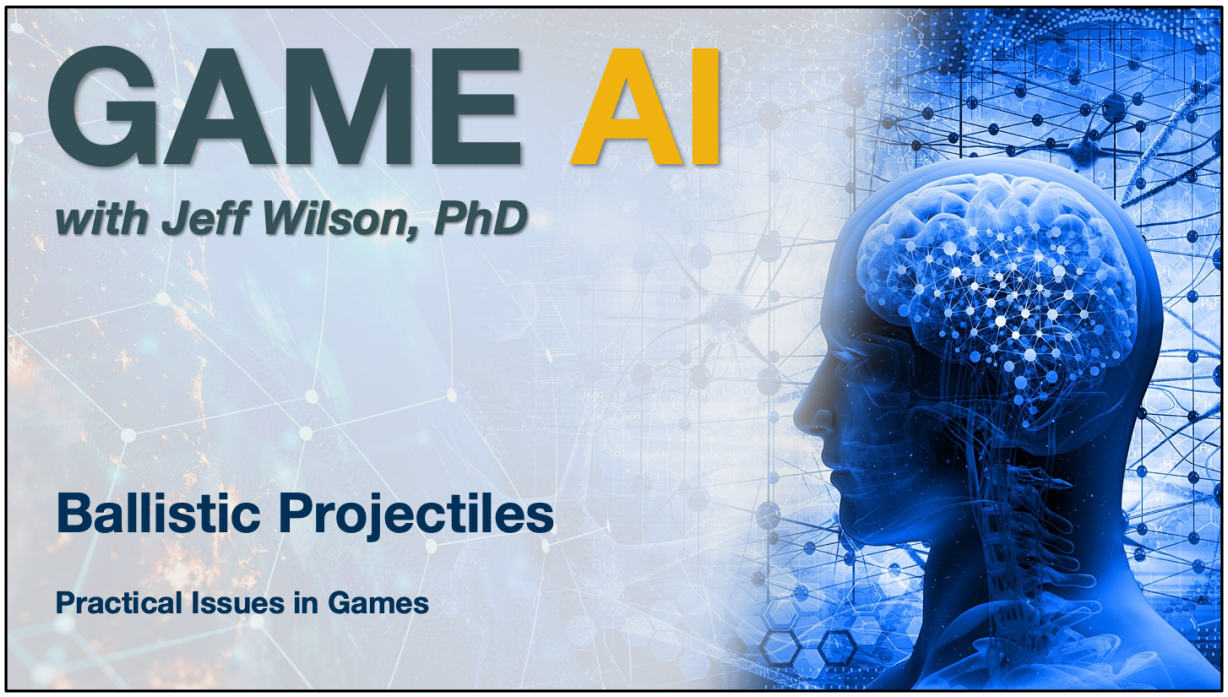
Georgia Tech

2D Law of Cosines Iterative Algorithm

- **Pros:**
  - Finds exact solutions for moving targets
- **Cons:**
  - May miss solutions near the limits
  - May not optimize shot for maximum speed and lowest angle (soonest intercept)
  - May not be able to refine to a stable solution, especially around the point of closest approach of the target, or with higher derivatives of motion
  - Target that is initially unreachable, but will be reachable in the future, creates implementation difficulties
  - Doesn't work if kinematic movement of projectile and/or target cannot be directly evaluated (drag and other scenarios)

Other Iterative Approaches

- Sometimes projectile and/or target movement can only be modeled incrementally (position/state at current timestep dependent on position/state at previous timestep)
- In this case, an incremental algorithm will need to increment through a simplified simulation of projectiles with candidate trajectories and determine the point of closest approach to the target
- Millington, *Artificial Intelligence for Games* discusses further


GAME AI
with Jeff Wilson, PhD
Ballistic Projectiles
Practical Issues in Games

Ballistic trajectory: with animated characters

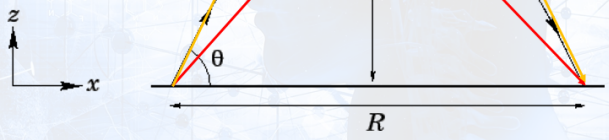
- Animation delay from projectile launch (is especially pronounced with throwing motion)
- Must make prediction based on when throw will actually happen.
- Predict throw from predicted future character position and relative launch position (e.g., where is the hand at throw release point?)

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…
  - Do you know target's goal? Anticipate arrival at that location

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?



Georgia Tech

Example

- Check out the Minion Targeting Video posted separately