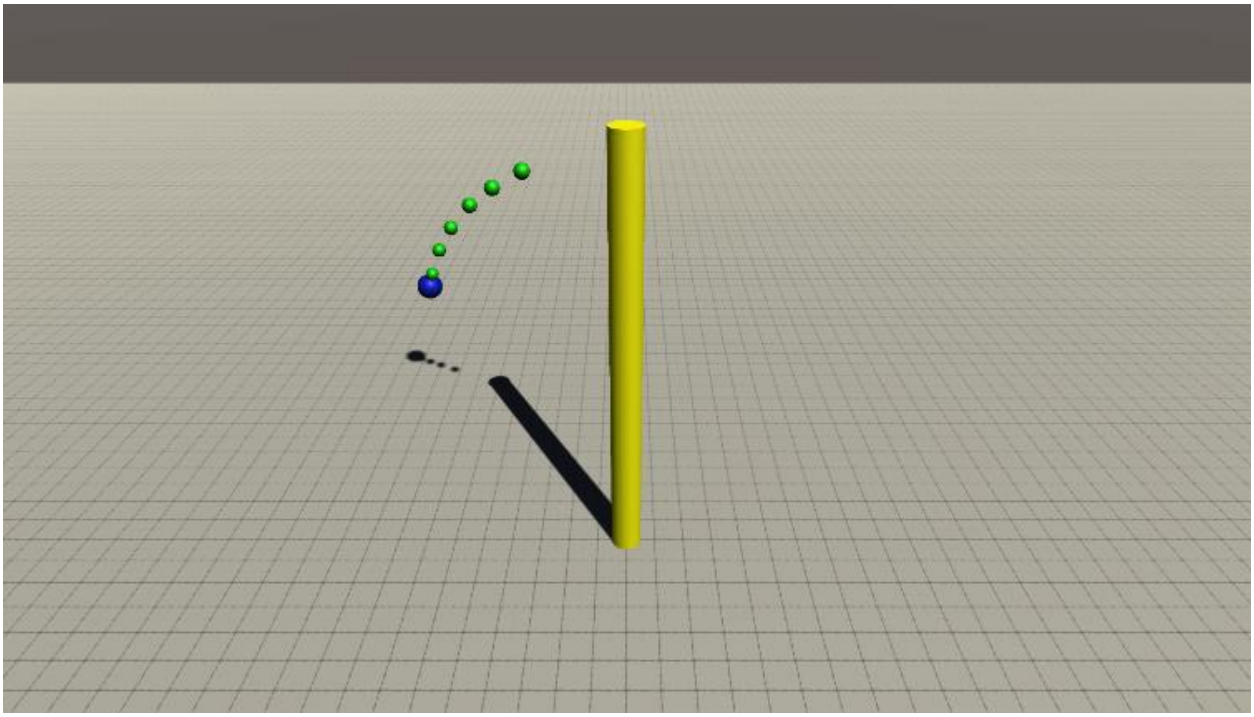
Homework 5 – Ballistic Projectiles for Prison Dodgeball



In this assignment you will be implementing prediction code for calculating the trajectory of a projectile to intercept a moving target.

Assessment

Grading will be based on the following aspects of your code:

1.) Implement ballistic trajectory prediction for projectiles
2.) Implement shot selection logic (decide when it is appropriate throw a dodgeball)

Your ballistic trajectory implementation is important for good results in Prison Dodgeball gameplay. The results of this homework can aid you in your homework 6.

Ballistic Trajectory Prediction

You can use any method for prediction including Millington's static target method coupled with iterative refinement, Law of Cosines (LoC) with 10% Holdback, LoC with iterative refinement, or directly solve with a more advanced method (you will likely need to research quartic solving methods and implement).

In terms of difficulty of implementation, Millington's static targeting and LoC are about equal in difficulty to implement. Adding iterative refinement to static targeting (to adjust for target movement) is only slightly more effort to code than LoC with 10% holdback and it gives better

range. LoC iterative is arguably difficult to implement. So, if you are looking for a recommendation you might start with Millington plus iterative refinement.

Regarding assessment of your ballistic trajectory implementation, your code will be tested with isolated calls to your ThrowMethods.PredictThrow() method, within the Shooting Range scene checking accuracy and shots per minute, and within the AdvancedMinionTestThrowScenario scene (this also tests your Shot Selection logic).

Isolated trajectory test scenarios will include both targets that are static and undergoing constant velocity. You will not be expected to accurately predict intercepts for targets that are accelerating, initially unreachable but become reachable after some time, are moving at extreme speeds, or are at extreme elevation differences. But positions, directions, speeds, gravity (always down in -Y direction, or zero), etc., can all vary. Your implementation is expected to reliably determine whether a target is reachable or not (correct Boolean return value).

Shot selection will be tested within AdvancedMinionTestThrowScenario. The Minion AI logic found in MinionThrowTester.cs will call your ShotSelection.SelectThrow() implementation. You will select shots based on the context of the scenario. Is the opponent clearly breaking the constant velocity assumption (or will imminently)? If so, defer throwing. Is the opponent's intercept position going to be occluded by an obstacle? If so, defer throwing.

See the existing source code for parameters and scaffolding (along with hints).

Homework Resources

You will build upon the Unity project provided to you via git.

This project contains scenes relevant to this assignment: ShootingRange, MinionTestThrowScenario (probably more useful for next assignment), and AdvancedMinionTestThrowScenario. There is also Prisonball used in the next assignment.

The files you will be working with are within Assets/Scripts/GameAIStudentWork/

These are:

- ThrowMethods.cs
- ShotSelection.cs

Here are useful features of code you will need to access:

Minion

You won't need much information from the MinionScript for HW5, but you will need to know about a few things. See **bolded** text for most relevant details.

Tip: When referring to minion information, all the minions have the same abilities and dimensions. So, a minion can just reference itself for things like ThrowSpeed and Radius/Height. However, transform.position, Velocity, etc., are unique to specific minions. You can't directly access opponent minions and must use the OpponentInfo available to you.

transform.position/rotation – minion pose (centered on capsule collider centroid)
Radius – radius of minion (capsule collider)
Height – height of minion (capsule collider)
MaxPathSpeed – how fast minion can run in a straight line
ThrowSpeed – speed in m/s of fastest throw minion can make
Velocity – current velocity on the navmesh
SpawnIndex – what order of spawning into simulation? Good for an arbitrary id relative to team
MaxAllowedOffAngleThrow – how far from directly facing target is allowed in ABS degrees?
TurnToFaceSpeedDegPerSec – how fast does FaceTowards() turn?
EvadeCoolDownTimeSec - how long (in sec) till another evasive action can be taken
HasBall – Does the minion have a ball?
DodgeballIndex – If a ball is held, the integer index into the DodgeballInfo of the PrisonDodgeballManager (see GetDodgeballInfo()). The value is -1 if a ball is not held
HeldBallPosition – Where is the ball for throwing start position?
IsPrisoner – Has minion been tagged and marked a prisoner?
TouchingPrison – is touching the prison area?
IsFreedPrisoner – Is minion freed but still walking back to playing area?
CanCollectBall – Can the minion pick up a ball?
CanBeRescued – Can the minion be rescued by throwing a ball to him?

DisplayText(string s) – Write text above minion's head. Helpful for debugging.
FaceTowards(Vector3 target) – Turn to face target while standing still
FaceTowardsForThrow(Vector3 target) – Turn so that throwing hand is facing towards target. You will generally want to use this in conjunction with ThrowBall().
SignedAngleWith(Vector3 target) – Angle minion forward vec makes with vector to target
AbsAngleWith(Vector3 target) – ABS angle minion forward vec makes with vector to target
GoTo(Vector3 target) – Navigate to target
ReachedTarget() – Did agent reach target from GoTo() call?
Stop() – Stop following navmesh path
Evade(EvasionDirection ed, float strength) – take an evasive action with varying strengths (normalized scale)
ThrowBall(Vector3 unitVDir, float normSpeed) – Throw ball in direction with percentage of ThrowSpeed

NOTE: You may **not** use any methods marked INTERNAL_ (even if public) from any objects in the simulation. If there is something that is public but not mentioned above, just ask about it on newsgroup.

OpponentInfo

You only have access to limited info about opponents (especially for the Prison Dodgeball game).

float Index – persistent spawn order index (can be used to identify same minion frame to frame)
Vector3 Pos – World position
Vector3 Vel – Velocity of minion
Vector3 Forward – Forward vector of minion
Vector3 PrevPos – Position of minion last fame
Vector3 PrevVel – Velocity of minion last frame
Vector3 PrevForward – Forward vector of minion last frame
float Radius – opponent radius
float Height – opponent height
bool HasBall – Is holding a ball
bool IsPrisoner – Is a prisoner
bool IsFreedPrisoner – Is a prisoner returning to field of play

PrisonDodgeBallManager

You shouldn't need to access this for HW5, but you may see it referenced for creating bitmasks for raycasts.

MinionThrowTester

You shouldn't need to modify this file. It is a limited implementation of AI logic to support testing of your ShotSelection and ThrowMethods. Just leave it as is and it will call the necessary methods. An original version of this file will be present in the autograder. You cannot override it.

It is possible that you can access much more information (e.g., cheat), but please limit yourself to the public properties/methods not labeled INTERNAL_. We will screen for disallowed use of INTERNAL_ methods and also GetComponent<T>(), low level runtime access such as Reflection, and some Unity game state managers. **If you want to use a particular data source but are unsure if allowed, just ask on Ed.**

Throwing

You must implement *PredictThrow()* in your ThrowMethods without modifying the method signature or changing its namespace. Refer to the lectures on Ballistic Projectile Trajectory solving to determine the method you prefer.

ThrowMethods.PredictThrow() cannot call "live" game state. This includes Physics and Navmesh namespace. Analysis of live game state should only be done in ShotSelection.cs. If your PredictThrow() passes EditorMode Test Runner tests then you can be assured that your code isn't accessing live game state.

Tip: You can implement an incremental solver that itself calls a static position solver with the same method signature as below. This can be useful for testing because you can test that your static solver works with a static or very slowly moving target before adding the iterative refinement for a moving target (see Testing section regarding the shooting range).

// Returns TRUE if the throw is possible, FALSE otherwise **<- Make sure to implement this logic and DO NOT just always return true!**

```
public static bool PredictThrow(
    // The initial launch position of the projectile
    Vector3 projectilePos,
    // The maximum ballistic speed of the projectile
    float maxProjectileSpeed,
    // The gravity vector affecting the projectile (likely passed from Physics.gravity)
    Vector3 projectileGravity,
    // The initial position of the target
    Vector3 targetInitPos,
    // The constant velocity of the target (zero acceleration assumed)
    Vector3 targetConstVel,
    // The forward facing direction of the target. Possibly of use if the target
    // velocity is zero
    Vector3 targetForwardDir,
    // For algorithms that approximate the solution, this sets a limit for how far
    // the target and projectile can be from each other at the interceptT time
    // and still count as a successful prediction
    float maxAllowedErrorDist,
    // Output param: The solved projectileDir for ballistic trajectory that intercepts target
    out Vector3 projectileDir,
    // Output param: The speed the projectile is launched at in projectileDir such that
    // there is a collision with target. projectileSpeed must be <= maxProjectileSpeed
    out float projectileSpeed,
```

```
    // Output param: The time at which the projectile and target collide
    out float interceptT,
    // Output param: An alternate time at which the projectile and target collide
    // Note that this is optional to use and does NOT coincide with the solved projectileDir
    // and projectileSpeed. It is possibly useful to pass on to an incremental solver.
    // It only exists to simplify compatibility with the ShootingRange
    out float altT)
```

You must also implement ShotSelection.SelectThrow(). This method can call "live" game state such as Physics.Raycast() and Navmesh.Raycast(). It can only be tested in PlayMode.

```
public static SelectThrowReturn SelectThrow(
        // the minion doing the throwing, can also be used to query generic params true of all
            minions
        MinionScript thisMinion,
        // info about the target
        PrisonDodgeballManager.OpponentInfo opponent,
        // What is the navmask that defines where on the navmesh the opponent can traverse
        int opponentNavmask,
        // typically this is a value a tiny bit smaller than the radius of minion added with radius
            of the dodgeball
        float maxAllowedThrowErrDist,
        // Time since last frame
        float deltaT,
        // Output param: The solved projectileDir for ballistic trajectory that intercepts target
        out Vector3 projectileDir,
        // Output param: The speed the projectile is launched at in projectileDir such that
        // there is a collision with target. projectileSpeed must be <= maxProjectileSpeed
        out float projectileSpeed,
        // Output param: The time at which the projectile and target collide
        out float interceptT,
        // Output param: where the shot is expected to hit
        out Vector3 interceptPos
    )
```

Useful Unity Features

For this assignment, you will probably want to use many of these. Check the Unity documentation for details.

Mathf – general math methods, constants, etc.

Vector3/Vector2 – represent vectors, and access common methods to manipulate

Navmesh.Raycast() – Useful to check extrapolated path of opponent. (Only in SelectThrow())

Physics.Raycast() – Useful to check for projectile collisions before target reached (Only in SelectThrow())

Debug.DrawLine() – Useful to see visualization of your raycasts to make sure they are correct (Only in SelectThrow())

Testing

There are a few resources to aid with testing.

Shooting Range Scene

Open Assets/Scenes/ShootingRange.scene
You can test out your ThrowBall() implementation in a controlled environment.
The Assets/Scripts/ShootingRange/ShootingRange.cs code can be modified if you like (but you won't be submitting it). Just be careful, as changes could lead to misleading results as compared to the auto-grader. An unmodified ShootingRange.cs file is present in the autograder and cannot be overridden.

The Shooting Range already has some keyboard presets for different types of motion, which you can extend/modify. Mode "4" is most similar to the Prison Dodgeball scenario. The Spacebar can be used to toggle between multiple shooting algorithms. **Make sure you are aware of whether your code is being tested. Check the name that appears.** More aim methods can be added in Awake() provided that each method has the method signature defined above for PredictThrow(). You can reset stats with "r".

For more extensive customization, refer to the files in Assets/Scripts/ShootingRange/ as well as the game objects in the scene.

Unit/Integration Testing

You can find unit/integration tests under Assets/Scripts/GameAIStudentWork/EditorModeTests and Assets/Scripts/GameAIStudentWork/PlayModeTests. For editor mode, there is an example of calling PredictThrow(). For play mode, there is a test that allows you to determine how well your code scores in the Shooting Range (ballistic trajectory solver performance) test and the AdvancedMinionTestThrowScenario (shot selection + ballistic trajectory solver).

Note that you must select either the PlayMode or the EditorMode tab in the Unity Test Runner to access each test group.

Rubric

- Isolated tests of PredictThrow() (see EditorMode test for a start) – 20%
- Shooting Range performance (see PlayMode test) – 40%
- Shot Selection performance (see PlayMode test) – 40%

For the Shooting Range and Shot Selection performance tests, you are given the tests with the same scoring algorithm as the autograder (before weighting as described above).

Submission

Submit files: *ThrowMethods.cs* and *ShotSelection.cs*

- Be sure to change the student name strings
- **Remove ALL debug print statements for more efficient code**
    - **TEST** your code after you remove print statements!
- Only submit the required source files