Milestone 3 – Artificial Intelligence

This is an *INDIVIDUAL* assignment.

Due
March 3rd (Friday) at 11:55 PM

Late Policy
See milestone one for the late policy (2^(n+1) late policy).

Description
The goal for this assignment is to adapt your character controller into an AI-controlled non-player character (NPC). This will be used to create a simple game experience where NPCs compete in the dodge ball variant of prison ball.

Details
You will create an NPC class that can play the dodge ball variant prison ball. Prison ball is a team sport and involves a split playing field with two prisons. The game is won by sending all of the opponent team members to prison. Opponents are sent to prison with successful hits with a dodge ball. Players can be rescued from prison with a ball thrown from their own team and caught by the prisoner. Players must remain on their side during play unless they are sent to their opponent's prison.

Note: We are open to prison ball rules variations if you like (e.g. neutral zone in the middle, attack from prisoners, etc.). We might not necessarily allow them depending on the effects on learning objectives of the milestone. If interested, just ask on Piazza for consideration of a game rules modification.

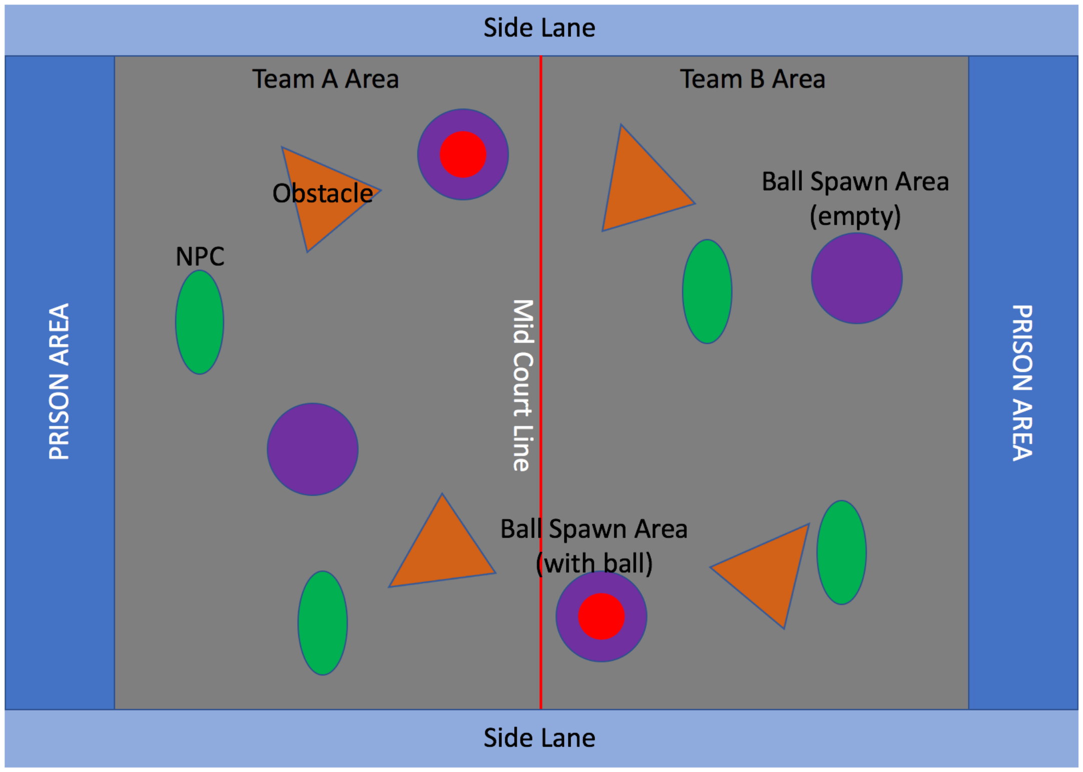

Figure: Graphic Depicting Prison Ball Playing Field

NPC Behavior Requirements

The NPCs will remain on their side of the court unless they have been hit by a dodge ball from the opponent. If hit by an opponent, they will first navigate to an out of bounds side lane then follow the lane to a prison area on the opponent's side. (If hit NPCs happened to be holding a ball of their own, they immediately lose it.) If in prison, NPCs will wait there and try to catch any balls thrown from their team to the prison area. If prisoners catch (collide with) a ball thrown by a teammate, they return to their side via the side lane. If all of one team has been sent to prison, the game ends and then resets for another play. NPCs not in prison will attempt some combination of: collecting available balls, throwing balls at opponents, throwing balls to teammates in prison, taking cover, making evasive moves from incoming balls, etc., as deemed appropriate.

Instead of regular dodge balls, the balls for this game will self-delete from the scene if they hit anything after being thrown (no bouncing around). This means that prisoners don't actually catch balls from teammates to escape. Instead, they just get hit and this counts as a catch. This behavior is meant to simplify the assignment. New balls spawn in randomly at various spawn locations. The dodge balls can have gravity disabled such that they follow a perfectly straight line when thrown. Trajectories are a bit easier to calculate this way. (Extra credit opportunity for gravity-influenced lobbed balls.)

You will implement one NPC type that can be replicated (via prefab) to create two teams with *at least* two team members each. To view the action, place a static camera above the playing field rather than chase individual players.

Implementation Requirements

You will build a custom state-based AI system with real-time steering and path planning capabilities. You will clone your existing player-controlled character (from M1/M2) and replace the controller script that interfaces with user inputs and instead send mecanim input commands dictated by your AI state machine. Alternatively, you can use Rival Theory's RAIN AI Unity plug-in (https://www.assetstore.unity3d.com/en/#!/content/23569). It's quite powerful, but does have a bit of a learning curve. The available documentation is also somewhat limited. (While CS4455/CS6457 students often have success with RAIN AI, we've also had many that initially start with it but then ultimately abandon it due some of the difficulties working within its framework.)

Additionally, you must tweak the various parameters of your AI and level such that the grader can observe all outcomes. That means that the grader should be able to observe NPCs collect balls from spawn points and throw the balls, both hitting and missing opponents. Additionally, NPCs should be observed going to prison and occasionally being rescued. Finally, team wins should eventually occur.

You NPC AI should have various context-specific strategies of play. Important strategic elements that you *need* to track and react to in your AI logic include:

- NPC is free/in-prison?
- NPC has/doesn't have ball?
- Free ball available in a spawn area?
- Opponent has ball ready to throw?
- NPC is out in the open/in a covered position?
- Strategic spots occupied by teammate or free?
- Opponent's current position and velocity (heading+speed)? (Used for predictive throw)

Other context you *might* consider:

- Opponent aiming in NPC's general direction?
- Opponent close/far away?
- While in prison: NPC in best spot to receive prison-escape throw?
- Live ball heading to NPC's side?
- Opponent doesn't have ball: Closest ball available to them?
- Teammate headed to strategic spot, but not there yet?

Visual and Audio Requirements

Your environment should not just be a large open field. It needs to provide a variety of low-profile and skinny obstacles such that the players only ever have partial cover from dodge ball attacks. The navigable surfaces can all be at the same elevation (to simplify projectile calculations). There are no specific graphic quality requirements for the scene other than it should be easy to distinguish where the various areas of the playing field begin and end. Feel free to add pendulums, moving barricades, etc., in front of the prison to make escape throws more challenging. (This may be the easiest way to address game balance issues of prison escape potential.)

You do not need to implement ball handling animations for the character (unless implementing extra credit). However, a player possessing a ball must have a clear visual indication that they have a ball. The easiest thing to do is to place the ball directly above the player's head. It's not necessary that all the players look unique, have team colors, etc. However, you may want to add some simple team color visualization such as a colored halo.

Rich audio feedback of game events should play accordingly. Also, a text description of the NPCs current AI state should be shown on the screen somewhere if implementing with navmesh+custom state machine. This can be displayed as 3D text attached to the NPC or on the HUD. (RAIN AI visualization is better supported with the built-in behavior tree editor, but you still need the audio feedback of interactions.)

Itemized Requirements:

1) Your NPC must be controlled with a state machine with at least three (3) states and must be predominantly controlled by root motion of mecanim.

   For instance, navMeshAgent.updatePosition and navMeshAgent.updateRotation should be set to false. Slight position corrections are allowed in OnAnimatorMove(), but should be subtle enough not to notice any sliding.

   If using RAIN AI, you must control via a behavior tree and mecanim motor. The tree must have branches representing at least three (3) specific strategies of play.

   Also, your NPC should be replicated to create the two (2) teams of at least two (2) team members each on a playing field of your design with fixed, overhead camera.

   (15 points)

2) When without a ball, your NPC will try to safely get a new ball when one is available. The NPCs will pick up collectable balls from spawn locations only on their side. The acquired ball will appear somewhere on/near the NPCs (such as floating above their head). Note

that the character does not need animations for picking up a ball, unless you are doing the extra credit.

(15 points)

3) Your NPC has a dodge ball throw/launch ability that involves position prediction based on the position and velocity of the opponent. The projectile itself must be animated with a velocity and not be an instantaneous hit (like a laser gun). Effects of gravity are not required, unless you are implementing the extra credit. Earn 10 points for launching at the current position of the opponent. Earn additional 10 points for launching at the predicted position (linear extrapolation using velocity). Note that the character does not need animations for throwing the ball, unless you are doing the extra credit.

   (20 points)

4) When in danger of attack from an opponent with the ball, the NPC will take cover behind obstacles and make evasive maneuvers.

   (15 points)

5) Your NPC uses Unity's navmesh and built-in A* support to navigate to its desired destinations. (If using RAIN AI, use equivalent capabilities.)

   (10 points)

6) Your NPC can get hit with a ball. Once hit, the NPC uses the side lane to go to prison. Furthermore, an NPC in prison that catches a throw from a teammate returns to their side via the side lane. Note that the character does not need animations for getting hit by the ball, unless you are doing the extra credit.

   (10 points)

7) Informative sound effects are utilized to denote throws, getting hit, picking up a new ball, catching a prison-escape ball, game ends with win, etc.

   If **not** using RAIN AI, 3D follow text or HUD clearly displays the current AI state of each NPC.

   (15 points)

Extra Credit Opportunities

Do one of the options below for **UP TO** 5 points. Do two of the options below for **UP TO** 10 points. Be sure your readme clearly documents that you have completed extra credit and want the grader to assess. Actual credit awarded is determined by the grader, specifically determining if the spirit of the extra credit task is met and other assignment requirements are met.

Human Player joins a Team with Three Players (or more) per Team

Add support for human player (with chase cam instead of overhead arena cam). The human player should be able to collect balls, aim, throw, escape prison, etc., just like the NPCs. Make sure than the grader can easily achieve all possible outcomes.

Advanced Projectile

Your AI's projectile is modified to be under the influence of gravity, but it is still launched so as to intercept the opponent's predicted positions. The projectile is thrown with a parabolic arc. The AI may adjust a variable (but bounded) throwing force and a variable launch angle. You may solve directly, or use an iterative approach. In either case, make sure that there is not a noticeable hit on frame rate during the calculation. *Your AI's throwing force should be bounded such that a lob angle of around 45% is necessary to reach the prison with a throw from near mid-court.*

New Animations for AI

Your character has an animation for picking up, holding, getting hit by, and throwing the dodge balls. You will likely need to take advantage of Mecanim's animation layers and avatar masks.

Tips

General Projectile Tips:

You should be aware of the benefits of the atan**2**() function for calculating headings. https://en.wikipedia.org/wiki/Atan2

A Normal distribution can be used to add some realistic aiming error to your AI, should you need it for game tuning.

For tips on predictive aim, check out "Strategy #3 - Assuming Zero Acceleration" under http://www.gamasutra.com/blogs/KainShin/20090515/83954/Predictive_Aim_Mathematics_for_AI_Targeting.php

Other sections of the article give some tips on lobbed projectiles.

General Strategy Tips

You may find it simplifies your logic to have designated spots to throw from, spots with good cover, waypoints, etc. This is easier than analyzing 2D/3D regions of the map.

If using RAIN AI:

You will definitely want to first become familiar with general *behavior tree* principles: http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

If using navmesh + custom state machine:

Don't forget to mark objects as "static" if you want the navmesh baking process to pick up on them.

Coupling Animation and Unity navmesh:

Specifically refer to “Animation Driven Character using Navigation”
https://docs.unity3d.com/Manual/nav-CouplingAnimationAndNavigation.html

Also, a NavMeshAgent+Mecanim demo is here (scene CS4455_AI_Demo):
https://github.gatech.edu/IMTC/CS4455_MecanimTute

Don’t forget to set navMeshAgent.updatePosition and navMeshAgent.updateRotation to false since we want mecanim to control movement. Also, go ahead and add OnAnimatorMove() with full Mecanim control of transform:

```
this.transform.position = anim.rootPosition; //npc only changes pos if mecanim says so
this.transform.rotation = anim.rootRotation; //npc only changes rotation if mecanim says so
```

Additionally add NavMeshAgent position resets to OnAnimatorMove():

```
agent.nextPosition = this.transform.position; //pull agent back if she went too far
```

Try changing the NavMeshController’s default vehicle properties to match your root motion performance envelope as best you can. Also, configure the NavMeshController to match your capsule dimensions and other locomotion abilities. You can look at your individual animations in the Inspector for a summary of your average velocity and angular velocity. Use these values from your fastest running/turning animations for configuring your NavmeshController. You now need to analyze the NavMeshAgent’s planned movements in (Fixed)Update() and map them to Mecanim inputs (hint: normalize velocity and angular velocity relative to the performance envelope to get in right form for mecanim inputs). Then start tweaking/filtering things slowly to reduce jittery movement. For instance, a scaling factor on turn angles is useful to deal with turning that is too aggressive. Finally, add filtering on your mecanim inputs using Lerp() to smooth out the last bit of jitter (but only do so once you have exhausted all other tweaks). Some filtering strategies are demonstrated in the github project above.

State Machine Implementation:
Test and develop one AI feature at a time, perhaps hard-coding your AI to stay in one state as you work on it.

If you need to slow down your AI for gameplay debugging, just put a cap on the maximum mecanim speed input passed from your steering calculations. For instance, if 1.0 is full speed then only allow a value of 0.8 for 80% of full speed. You can also adjust your NavMeshController top speed.

State machines can be implemented in a very simple manner. Consider an object-oriented approach: https://unity3d.com/learn/tutorials/topics/scripting/using-interfaces-make-state-machine-ai

Alternatively, you might consider a procedural approach similar to below:

```
public enum AIState
{
  Patrol,
  GoToAmmoDepot,
  AttackPlayerWithProjectile,
  InterceptPlayer,
  AttackPlayerWithMelee,
  ChasePlayer
  //TODO more? states…
};

public AIState aiState;

// Use this for initialization
void Start ()
{
  aiState = AIState.Patrol;
}

void Update ()
{

  //state transitions that can happen from any state might happen here
  //such as:
  //if(inView(enemy) && (ammoCount == 0) &&
  //  closeEnoughForMeleeAttack(enemy))
  //  aiState = AIState.AttackPlayerWithMelee;


  //Assess the current state, possibly deciding to change to a different state
  switch (aiState) {

    case AIState.Patrol:

    //if(ammoCount == 0)
    //    aiState = AIState.GoToAmmoDepot;
    //else
    //  SteerTo(nextWaypoint);

    break;

    case AIState.GoToAmmoDepot:

    //SteerToClosestAmmoDepot()

    break;

    //... TODO handle other states

    default:

    break;

  }

}
```

Submission:

You should submit a 7ZIP/ZIP file of your Unity project directory via t-square. **Please clean the project directory to remove unused assets, intermediate build files, etc., to minimize**

the file size and make it easier for the TA to understand.

The submissions should follow these guidelines:

a) Your name should appear on the HUD of your game when it is running.
b) ZIP file name: <lastName_firstInitial>_mX.zip (X is milestone #)
c) A /build/ directory should contain a build of your game. Please make sure you preserve the data directory that accompanies the EXE (if submitting a Windows build)
d) Readme file should be in the top level directory: < lastName_firstInitial >_mX_readme.txt and should contain the following
    i. Full name, email, and TSquare account name
    ii. Detail which platform your executable build targets (Windows, OSX, GlaDOS, etc.) Also let us know if you implemented game controller support, and which one you used.
    iii. Specify which requirements you have completed, which are incomplete, and which are buggy (be specific)
    iv. Detail any and all resources that were acquired outside of class and what they are being used for (e.g. "Asset Bundles downloaded from the Asset Store for double sided cutout shaders," or "this file was found on the internet at link http://example.com/test and does the orbit camera tracking"). This also includes other students that helped you or that you helped.
    v. Detail any special install instructions the grader will need to be aware of for building and running your code, including specifying whether your developed and tested on Windows or OSX
    vi. Detail exact steps grader should take to demonstrate that your game meets assignment requirements.
    vii. Which scene file is the main file that should be opened first in Unity
e) Complete Unity project (any file you acquired externally should be attributed with the appropriate source information)

Submission total: (**up to 20 points deducted** by grader if submission doesn't meet submission format requirements)

Be sure to save a copy of the Unity project in the state that you submitted, in case we have any problems with grading (such as forgetting to submit a file we need).