Assignment Instructions

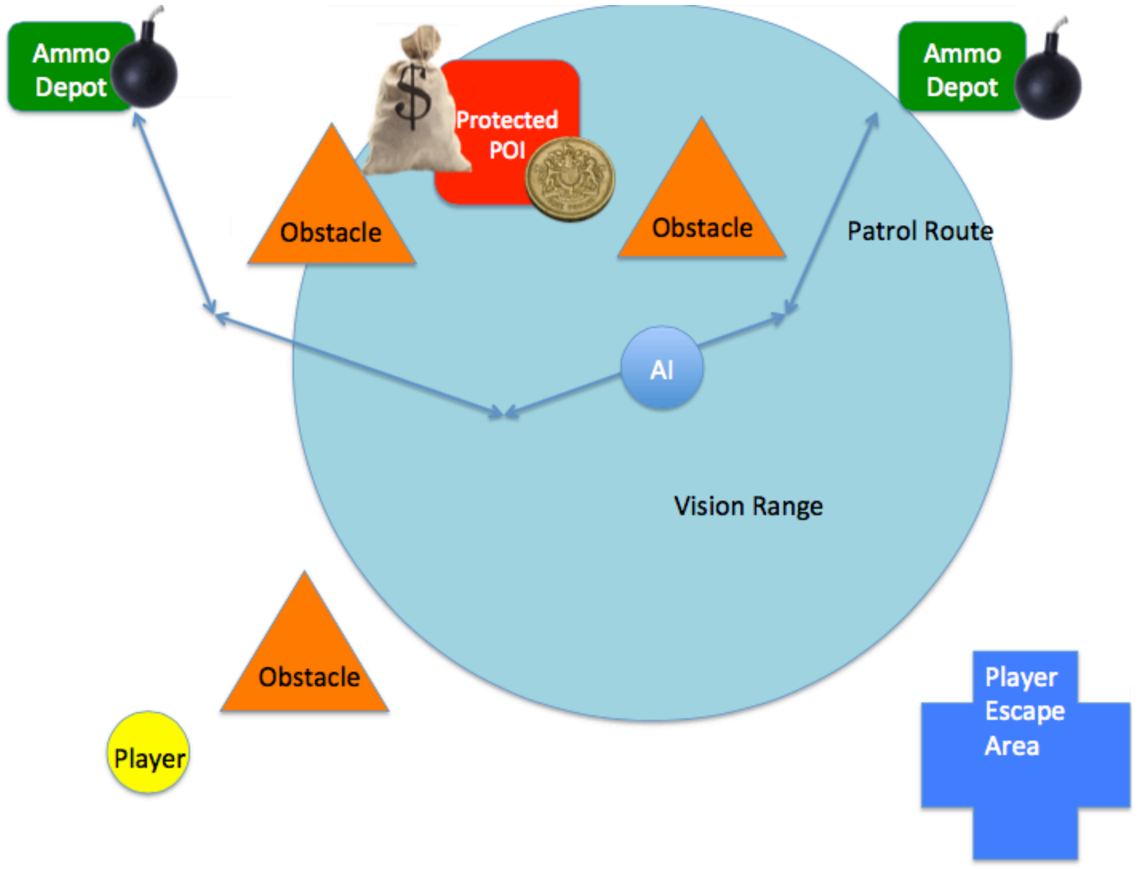
This is an *INDIVIDUAL* assignment.

Due
October 7th 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 integrate an AI-controlled non-player character (NPC) into one of your game feel gardens. This will create a simple game-like experience where the player attempts to steal something from the AI, and the AI tries to stop the player via different attack and chase behaviors. You will also introduce the concept of player health, dying, and respawning.

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.



Your AI will perform a few behaviors related to guarding and defending a point-of-interest (POI) in the game world. In this case, the AI is defending against the player's avatar reaching the POI.

If the player is too far away to be seen, the AI will perform a patrol around the POI. There should be at least three waypoints in the patrol.

If the player is seen (e.g. the player is close enough to the AI), then the AI will respond in one of two ways:

AI has ammo: If the AI has ammo, it will throw a projectile towards the *predicted* future position of the player. The prediction should be based on the player's current position and velocity. The projectile cannot be instantaneous like a laser but you can ignore (e.g. turn off) the effect of gravity. Therefore the projectile must have a velocity (speed and at least a 2D direction). The projectile must do damage to the player if there is a hit. (The projectile should have a short life cycle and be removed from the world based on criteria of your choosing.)

AI does not have ammo: If the AI is out of ammo, then it will instead head to a point somewhere between the player and the POI and attempt to block the player from reaching the POI while simultaneously performing a melee attack. The melee attack must do damage to the player.

If the player reaches the POI without dying, then the AI switches to a chase mode and attacks whenever close enough. (Imagine that the player has stolen something of value upon reaching the POI and the AI now wants it back. You don't have to show a stealing animation, but do play a success event sound.) The AI should be implemented with the navmesh and A* features of Unity in order to chase the player to any location in the game world. You can assume that the AI always knows where the player is without necessarily seeing the player because of a magical homing signal in the stolen goods.

Alternatively, if the player leaves sight of the AI and has *not* touched the POI, then the AI will do one of the following:

AI has ammo: If the AI has ammo, it will return to following its patrol waypoints.

AI does not have ammo: If the AI does not have ammo, it will first visit an ammo depot to reload, then return to its patrol waypoints.

The player must die from sufficient attack from the AI. In this case, the level will reload automatically. If the player successfully reaches the POI AND then makes it to a Player Escape Area, also reload the level.

To meet the basic assignment requirements, you don't need AI character animation for the projectile toss or the melee attack. However, you need to play an appropriate sound to indicate when the attacks are occurring. Also if not using RAIN AI, the current AI state should be displayed on the GUI.

Your environment should have a variety of obstacles/paths and not just be a large open field.

Additionally, you must tweak the various parameters of your AI and level such that the grader can achieve all outcomes without extreme effort. That means that the grader should be able to reach the POI and the Player Escape Area. Additionally, the grader should be able to easily witness successful attacks from the AI. Lastly, the player should be able to get sufficiently far from the AI and become hidden behind various obstacles such that use of navmesh/A* can be observed navigating around them.

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.

   (5 points)

2) Your NPC must follow a path of at least three (3) waypoints when in a patrol/guard state.

   (10 points)

3) Your NPC has a ranged attack that involves position prediction based on the position and velocity of the player. The projectile must have a velocity and not be an instantaneous hit.

   (20 points)

4) When out of ammo, your NPC will reload ammo by visiting an ammo depot if the player leaves visible range (and the player hasn't touched the POI).

   (10 points)

5) Your NPC has an out of ammo behavior of trying to stay between the player and the POI, and executing a melee attack when the player is close enough.

   (20 points)

6) Your NPC uses Unity's navmesh and built-in A* support to track down the player in the *recover-stolen-item* state. (NPC tracks player after player touches POI.) Additionally the level provides suitable obstacles/corridors to demonstrate A*'s effectiveness.

   (20 points)

7) Your player can die. Upon death the player respawns and level resets/reloads. Also, your player can successfully visit the POI (e.g. steal item from AI) and then make it to the Player Escape Area. Upon success, the level restarts.

   (10 points)

8) Sound effects are utilized to denote seeing the player, attacks, taking damage, and reaching the POI and Player Escape Area. Also, there should be a visual indication of health remaining for the player.

   If **not** using RAIN AI, the HUD clearly displays the current AI state.

   (5 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.

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 player's predicted position. The projectile is thrown with a parabolic arc. The AI may adjust a variable (but bounded) throwing force and/or a variable launch angle. If the AI determines the player is out of range, then the throw is not made. The AI should be able to account for relative elevation changes with its toss and your level should provide an opportunity to assess this. 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. Feel free to use area damage (like an explosion) when the ground is hit.

Example projectiles: Grenade, bomb, rotten tomato, etc.

New Animations for AI

Your AI has an animation for picking up projectiles. Also the AI is shown carrying the projectile in hand. Additionally, your AI has an animation for throwing the projectile. Lastly, your AI has a melee attack animation. You may need to take advantage of Mecanim's animation layers and avatar masks.

Tips

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

Try changing the NavMeshController's default vehicle properties to match your root motion performance envelope. Then start tweaking things slowly to reduce jittery movement. Finally, add filtering on your mecanim inputs using Lerp() to smooth out the last bit of jitter.

You should be aware of the benefits of the atan**2**() function for calculating headings (for projectiles perhaps).

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, 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 the following procedural approach:

```
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 FixedUpdate ()
{

        //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 has 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. (E.g. "The best way to see A* in use is to first touch the POI then use the short cut that requires jumping over the ditch as the AI can't jump. You should then see the AI navigate through the hedge maze to get to your position.")
    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).