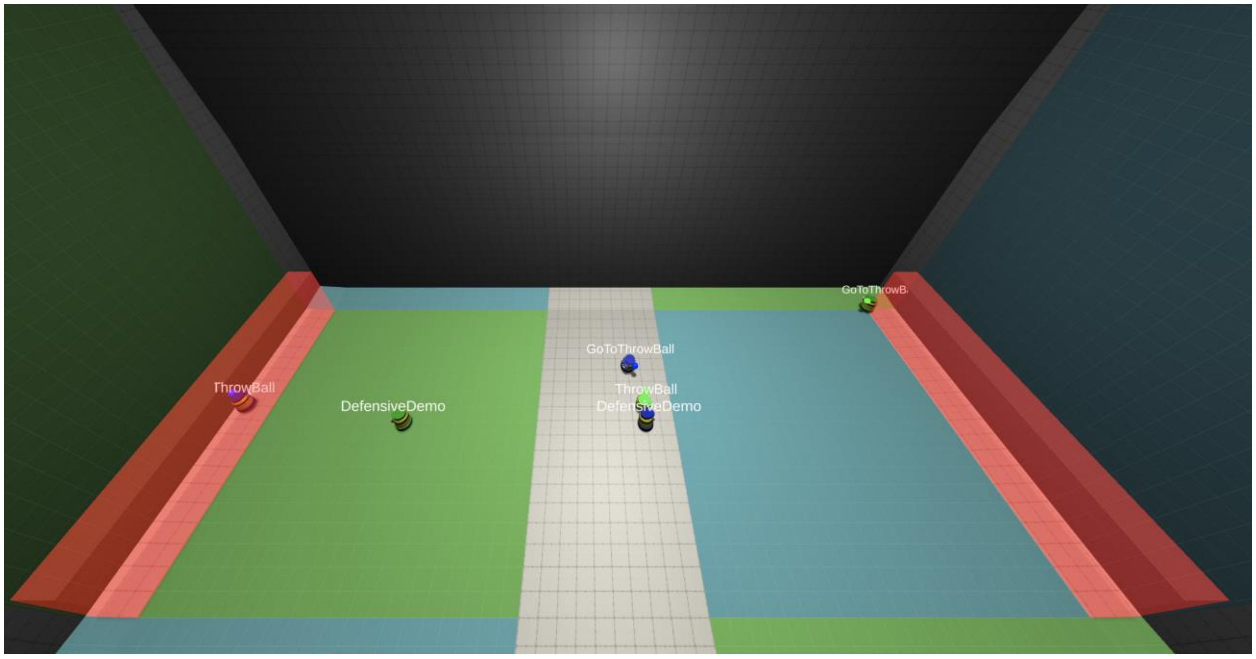
Homework 6 – Prison Dodgeball



In this assignment you will be implementing a Finite State Machine (FSM) to control each of a team of agents playing a game of Prison Dodgeball (description below). Your FSM must be able to reliably beat the provided demonstration opponent as well as undisclosed opponents to receive full points.

There will also be a class tournament for those that wish to participate. This tournament will be optional but will involve an extra credit opportunity. Details of the tournament are yet to be determined but will likely start with a round robin qualifier by randomly assigned groups. The best of each group advances to a single elimination tournament bracket, seeded by round robin group points (3 points for win, 1 point for tie, no points for loss).

An example FSM is provided that is minimally effective, but does demonstrate most all of the method calls you might be interested in. You can use the code as a starting point, though we recommend starting a state machine from scratch (the individual states) and building up a bit at a time so you fully understand your code. You may also find it useful to remove capabilities from your opponent as you are testing so that you can better observe your code in action.

Your FSM should result in reasonable behavior for a video game. For instance, agents shouldn't get stuck standing still in an unresponsive state, glitch back and forth, etc. They can however be a bit goofy (e.g. two teammates going after the same ball, not throwing at what appears to be the best target, etc.).

Assessment

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

1.) Implement a FSM compatible with the Prison Dodgeball Game that can reliably beat opponent AI, "Glass Joe", with various team sizes (1-5) and number of balls (1-4). Your team should win at least 2/3rds of the time. (85%)
2.) Reliably beat undisclosed AI opponent(s) 2/3rds of the time. (15%)

In preparation for the tournament, your submission must follow the rules of the game. This is usually enforced by the game engine, but if we become aware of a solution that is clearly breaking the game rules then we may exclude it from the tournament. If you are uncertain about a particular strategy you can inquire in a private post on the class newsgroup. If you find an exploit and want to make sure it isn't used against you, then let the instructional staff know. We may patch the game before the tournament.

Also, be aware that your tournament submission must run from a compiled build. So you cannot use the UnityEditor namespace.

Ballistic Trajectory Prediction

In HW5, you developed algorithms for ballistic projectile trajectory solving and shot selection. You should utilize that work to improve your AI agent's effectiveness. You will be submitting ThrowMethods.cs and ShotSelection.cs in addition to your MinionStateMachine.cs. The trajectory related files can be improved from your original HW5 submission. The autograder will not directly call them. Only your MinionStateMachine.cs file will call those methods. So it's ok to change method signatures and types if you feel you need to.

If you are not happy with your implementation in HW5, you can use the default implementation for both, which is the same that Glass Joe uses.

Prison Dodgeball

If you aren't familiar with Prison Dodgeball, here is a description.

The game consists of two teams opposing one another. The team size is variable, but the number of members should be equal between the two teams. Each team has one side of a symmetrical playing area (e.g. a gym). Sometimes there is also a neutral area in the middle of the two sides. Both teams are allowed in the optional neutral area, but only members of a team are allowed in the team area that they are assigned. There are also two gutters along the long axis and to the sides of the playing area. This is so prisoners can exit the field of play to go to and from prisons at either end of the playing field. The prisons are at the far ends of the playing area.

The objective of the game is to throw dodgeballs at opponents to tag them (without a bounce on the playing field or walls). If an opponent is tagged, they must go to the prison behind their opponents playing area. A player in prison must wait for a teammate to throw a ball to them for a rescue. The ball must be caught without bouncing first. Prisoners are free to move back and forth in the prison boundaries but cannot leave until rescued. Also, the prisoner gets to keep the ball. Prisoners are not supposed to interact with a neutral ball, but it's not uncommon to see kids throwing/kicking a neutral ball back to their non-prisoner teammates.

A rescued prisoner can immediately leave the prison but only by way of one of the gutters. The freed prisoner cannot be tagged while in the gutter until they pass the line that transitions from the opponents' area to the neutral area. If there is no neutral area, then use the transition line that divides the two team areas. Rescued prisoners that don't leave the prison/gutter in a timely fashion can be sent back to prison by the referee/gym instructor.

There are some number of dodgeballs as well. These are placed uniformly around the playing area and may appear in the neutral area or team areas.

Players have assigned starting locations. Often this is from the end line of each team's playing area. Players can also be from assigned individual starting positions. When play starts, players can rush to collect a ball if they choose or focus on evasion.

Gameplay progresses until one team has all players in prison. Once that happens, the other team wins. There is an optional time limit. If time runs out there is a tie.

Special circumstances

A dodgeball that bounces off an opponent and hits a second opponent without first touching the playing field tags both players out. This chain can go further as well. A dodgeball that hits a held dodgeball of an opponent (and no body part) is considered a deflection. In this case, the opponent does not go to jail.

In the normal version of prison dodgeball, a player that catches a throw from the opposing teams causes the thrower to be sent to jail.

Potentially, two teams can reach the point where there is one free player on either team. Furthermore, each could throw their ball at each other and simultaneously send each other to jail. If it is clear that one was tagged before the other, then there is clearly a game winning tag. However, if it cannot be determined the result is a tie.

A player that goes into the opposing team's side, the prison containing the opposing team prisoners, and possibly also the gutters is considered to be out and sent to prison. Balls stuck in an area that cannot be reached are dealt with in various ways. For instance, prisoners may be required to place errant balls on the line between the prison and the opponents' playing area.

Differences in Minion Prison Dodgeball

It’s not possible to catch opponents’ throws (unless there is a bug in the code). Minions play with a neutral area. Gutters and prisons have a sloped floor to return the balls. Minions play with a time limit (varies). Game rules block minions from entering an area that they are not allowed, so no penalties are necessary.

Rescued minions that refuse to leave the prison or gutter will be returned to prisoner status after 10 seconds without return to the neutral area.

Homework Resources

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

This project contains scene PrisonBall which you can open and play for a demonstration.

The file you will be working with is:
Assets/Scripts/GameAIStudentWork/MinionStateMachine.cs

Note that you will also submit ThrowMethods.cs and ShotSelection.cs (even if you plan to use the default “Glass Joe” implementation)

This file includes a demo implementation of a state machine. You should replace the implementation of the states with your own. However, you will likely want to keep most of the MinionStateMachine implementation. This class inherits from FiniteStateMachine and does most of the work of running the individual states. You can instantiate and AddStates() to MinionStateMachine in the method Start().

Each State you implement has and Enter(), Exit(), and Update() method that you can implement. When the FSM runs a state, it first calls Enter(). Then it calls Update() until a transition occurs. Lastly Exit() is called after a transition. States can initiate a transition by returning a DeferredStateTransition to another state from Update(). If the state doesn’t want to transition yet it should return null. A DeferredStateTransition is just an object that notifies the FSM what state you want to transition to, passes any parameters the new state needs, and optionally allows the transition to immediately update before the next simulation frame (default is to **not** immediately update).

There is a special global state to handle wildcard transitions. Also, there is a simple “TeamShare” implementation that allows for coordination between your team. It is somewhat like a *Blackboard Architecture*. It currently provides a way to track teammates and dodgeballs. But it could also be extended by you to track responsibilities, avoid redundant behavior, etc. Be aware that with the TeamShare, you should always check it for null before accessing. Also, check any values you store in TeamShare for null as well before using them. In all cases of a null (or otherwise invalid data), have safe fallback behavior.

Each State has access to a variety of information available including the parent FSM (MinionFSM), the Minion that it is attached to, the game manager (Mgr), the assigned Team, and a TeamData object shared amongst the team. The Minion and Mgr will probably be used a lot in your states.

Here are useful features of these objects:

<u>Minion</u>

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.

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). If there is something that is public but not mentioned above, just ask about it on Newsgroup.

PrisonDodgeBallManager

There are a variety of markers you can access for determining positions (gutter, middle of team area, etc.). Markers are just positions that you can use for path planning. You can create your own positions stored in Vector3 variables, possibly offset from the known markers.

TeamSize
BallsPerTeam
IsGameOver
IsTie
IsWinner(Team)
MatchTimeRemSec
bool FindClosestNonPrisonerOpponentIndex(Vector3 myPos, Team myTeam, out int foundIndex) – Find a minion worth throwing at
bool GetOpponentInfo(Team myTeam, int index, out OpponentInfo oi) – Get details about opponent
public bool GetAllOpponentInfo(Team myTeam, ref OpponentInfo[] oppInfo)
public bool GetDodgeballInfo(Team myTeam, int ballIndex, out DodgeballInfo di, bool determineRegion)
public bool GetAllDodgeballInfo(Team myTeam, ref DodgeballInfo[] dodgeballInfo, bool determineRegion)

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 Unity game state managers. **If you want to use a particular data source but are unsure, just ask on Ed.**

Throwing

See HW5 regarding the logic of throwing. Use the provided default implementations if you did not develop code you are happy with (same as Glass Joe).

Testing

There are a few resources to aid with testing.

There are *MinionTestThrowScenario* and *AdvancedTestThrowScenario* scenes. This can help with debugging throwing with a real minion. It requires working with MinionThrowTester.cs.

<u>Unit/Integration Testing</u>

You can find unit/integration tests under Assets/Scripts/GameAIStudentWork/PlayModeTests. There is a test that can pit two agents head-to-head over an arbitrary number of matches in various configurations.

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

Submission

Submit file *MinionStateMachine.cs, ThrowMethods.cs,* and *ShotSelection.cs*

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

Tournament entry will be a separate submission (probably via Canvas and not via Gradescope!). Final tournament details are TBA.