# My first Unreal Editor Soccer AI Simulation

**posted in** Gamieon's Journal

Published September 16, 2014

Though I'm still new to the Unreal Editor and behavior trees, I wanted to create a primitive soccer simulation for a game I'm prototyping. You can get the code in its current form at:

## https://github.com/Gamieon/UBattleSoccerPrototype

[color=#b22222]

## Getting Started

[/color]

The first part of my journey was learning how "Blueprints" work in the Unreal Editor. I consider a blueprint to be a graphical representation of code. By graphical I mean both pixels-on-the-screen and boxes-connected-to-other-boxes-by-pointy-arrows. You can learn more about how they work at:

https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html

The second part of my journey was learning how "Behavior Trees" work. Buried in Google search results full of complicated papers and tutorials that blew my mind, I managed to find this little gem:

http://www.indiedb.com/groups/indievault/tutorials/game-ai-behavior-tree

That article clicked with me and I felt like I understood the basics after just one read.

## [color=#b22222]Setting the Rules[/color]

My first step into creating a soccer simulation was to establish a purpose and basic rules for the simulation:

1. The objective is to get the soccer ball into the enemy goal.
2. Each team has eleven players on the field at a time.
3. Each player is confined to an area on the field which I call "action zone."
4. Each player's purpose is to contribute to the objective for their team.
5. A player's only interaction with the soccer ball is to kick it into the goal or to a teammate.

Seen here is the reference I chose for assigning field positions for a soccer game

http://www.sportspectator.com/fancentral/soccer/soccer_diagram.gif

# [color=#b22222]Entities[/color]

With the rules established, I made a list of the different entities on the field that need to be tracked by a single player:

- Ball
- Enemy goal
- Teammates
- Enemy players

Note the absence of the role of the friendly goal. After several failed attempts at developing the behavior tree, I decided to create a tree that was to be used by every player; everyone from the goalie to the offensive players. I'm not going for perfect; I'm going for simple until I get better at this. Since none of my tree logic factors in the friendly goal, I'm not counting it as an entity here.

[color=#b22222]
## Binary Decision tree
[/color]

I'm new to behavior trees but not to basic problem solving. After several failed attempts, I came up with a simple decision tree that I could apply to every player on the field:

[code=nocode:0]- Is the ball in a goal? Yes - Is it in your goal? Yes A. Celebrate No B. Shrug No - Do you possess the ball? Yes - Are you close to the goal? Yes - Is there an obstacle in the kick line? No C. Take a shot Yes - Is it an enemy? Yes - Is another nearby teammate not close to any enemies and unobstructed? Yes D. Pass the ball toward the teammate No E. Run around enemy toward goal No F. Run around obstacle toward goal No - Is another teammate in front of you closer to the goal, not close to any enemies and unobstructed? Yes G. Pass the ball toward the player No H. Run toward goal No - Does a player on your team have possession of the ball? Yes - Are there enemies within a close distance to the player? Yes I. Pursue enemy closest to player No J. Run in parallel direction to friendly possessor up to the goal No - Does an enemy possess the ball? Yes K. Pursue enemy No - Is an enemy in the line between you and the ball? Yes L. Pursue obstructing enemy No M. Pursue ball

Notice how the farther down in the tree you go, the farther away you are from the purpose of the simulation which is to get the ball into the enemy goal. Here's a summary view of it in reverse order; note how it generally follows the progression of a soccer game:

- Go after the ball
- Pursue the enemy who possesses the ball
- Have the team maintain ball control and work with teammates to get the ball to the goal
- Kick the ball into the goal
- Celebrate the goal

[color=#b22222]
## What Does "Pursue" Mean?
[/color]

When a player pursues the ball, all they're doing is running to it. Once the player overlaps the ball's collision sphere, the game assigns them possession and the ball will always be in front of their feet no matter how they turn or run. The only way that a player can lose possession is if they or an enemy player kicks the ball away.

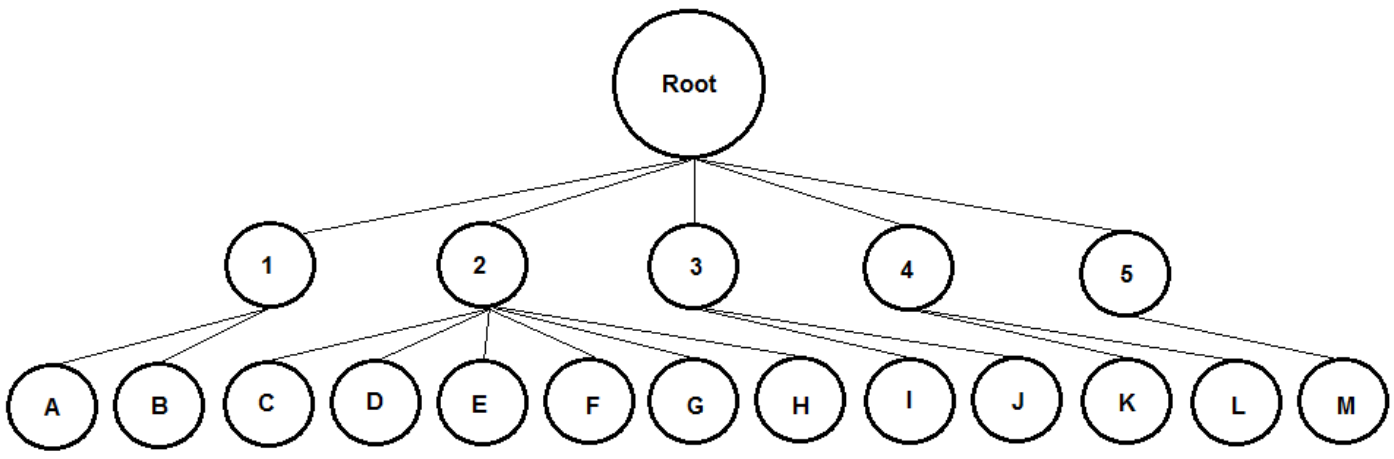
When a player pursues an enemy, all they do is run up to them and kick the ball away if they have possession. As I plan for this game to involve on-field fighting ala League of Legends or Diablo 3, I'm purposefully not doing anything more with player pursuits until I design the battle system.

# [color=#b22222]Leaf actions (tasks)[/color]

Once I had a decision tree I was content with, I turned my attention to the leaf actions in the tree and grouped them by purpose:

[code=nocode:0]1. GoalFanfare (Is the ball in a goal?) A. Celebrate B. Shrug2. ScoreGoal (Do you possess the ball?) C. Take a shot D. Pass the ball toward the teammate E. Run around enemy toward goal F. Run around obstacle toward goal G. Pass the ball toward the player H. Run to goal3. DefendPossessor (Does a teammate possess the ball?) I. Pursue enemy closest to player J. Run in parallel direction to friendly possessor4

possess the ball?) I. Pursue enemy closest to player J. Run in parallel direction to friendly possessor K. AttackPossessor (Does an enemy possess the ball?) K. Pursue enemy possessor L. Pursue obstructing enemy5. Pursue the ball (Nobody possesses the ball) M. Pursue ball
This list helped define the general shape and traversal of my behavior tree:




**The behavior tree would have a root node with five children below the root. Each child node would have one or more children of its own called "leaf" nodes since they themselves have no children. The tree traversal would start at the root node, then go through the child nodes from left to right until it finds one that is true. From there all of the leaf nodes (which from hereon I'll call "tasks") for that child are traversed from left to right until an action is decided on and performed by the player.**

## [color=#b22222]Splting up Tasks[/color]

Now that my behavior tree prototype was done, I had to make some decisions: Should I split any tasks into new subtrees with their own tasks? What functions do I need to write? Would I share data between tasks?

I decided to start by breaking up the tasks by modularity. In Unreal Editor a task is a standalone function which you may create and use in more than one place in a behavior tree. Tasks have access to shared variables stored in a "blackboard" which any task can read or write from. I looked at what tasks I could possibly modularize:

[code=nocode:0]1. GoalFanfare (Is the ball in a goal?) A. Celebrate B. Shrug2. ScoreGoal (Do you possess the ball?) C. Take a shot D. ** Pass the ball ** E. ** Run to goal (use Unreal's internal pathfinding to avoid obstacles) ** F. ** Run to goal (use Unreal's internal pathfinding to avoid obstacles) ** G. ** Pass the ball ** H. ** Run to goal (use Unreal's internal pathfinding to avoid obstacles) **3. DefendPossessor (Does a teammate possess the ball?) I. ** Pursue enemy ** J. Run in parallel direction to friendly possessor4. AttackPossessor (Does an enemy possess the ball?) K. ** Pursue enemy ** L. ** Pursue enemy **5. Pursue the ball (Nobody possesses the ball) M. Pursue ball

I broke every task with ** in two: One is the modular task that can be used in multiple places in the tree, and the other is the task that calculates what data to give to that modular task. I changed my behavior tree to look like this:

[code=nocode:0]1. GoalFanfare (Is the ball in a goal?) A. Celebrate B. Shrug2. ScoreGoal (Do you possess the ball?) C. Take a shot D1. Find a nearby teammate we can pass to D2. ** Pass the ball ** E1. Determine if an enemy is obstructing our route to the goal E2. ** Run to goal ** F1. Determine if a non-enemy actor is obstructing our route to the goal F2. ** Run to goal ** G1. Determine if there is a player closer to the goal we can pass to G2. ** Pass the ball ** H. ** Run to goal **3. DefendPossessor (Does a teammate possess the ball?) I1. Determine if an enemy is near the possessor I2. ** Pursue enemy ** J. Run in parallel direction to friendly possessor4. GetPossession (Ball is still in play but nobody on our team possesses it) K1. Determine if an enemy possesses the ball K2. ** Pursue enemy ** L1. Determine if an enemy is in the way between you and the ball L2. ** Pursue enemy ** M. Pursue ball

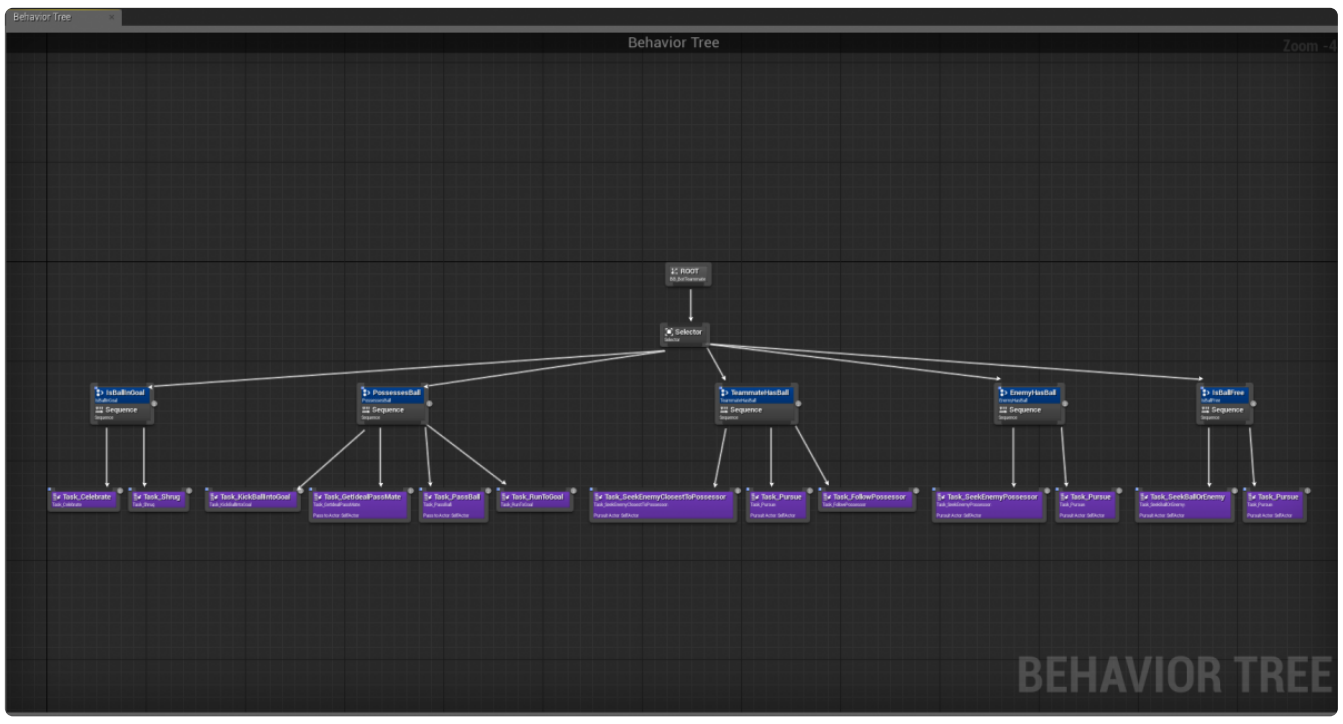
I needed only two Blackboard variables for passing data from one task to another:

- KickTargetActor (read by "Pass the ball" and "Take a shot")
- PursuitActor (read by "Pursue enemy" and "Pursue ball")

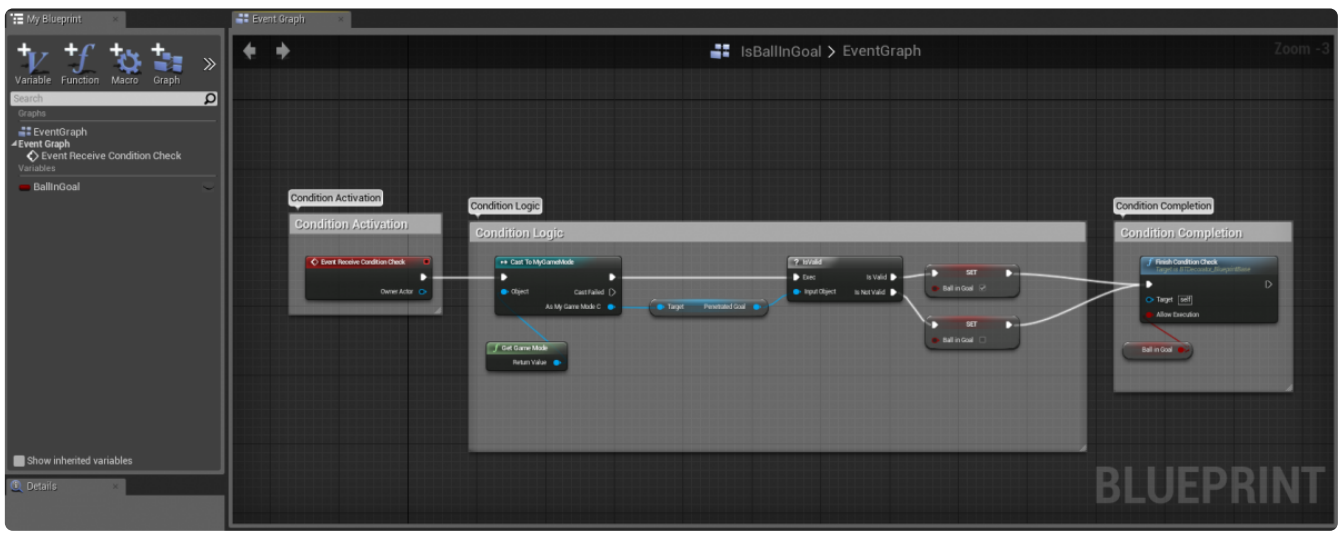
## [color=#b22222]Task Development[/color]

I won't bore you with how I developed the blueprint for each task here, but I will say that I tried to keep all of them small and modular where possible. I thought about posting screenshots but that would be like posting random snippets of code with no context. Instead you can see

## a screenshot of the final behavior tree here.



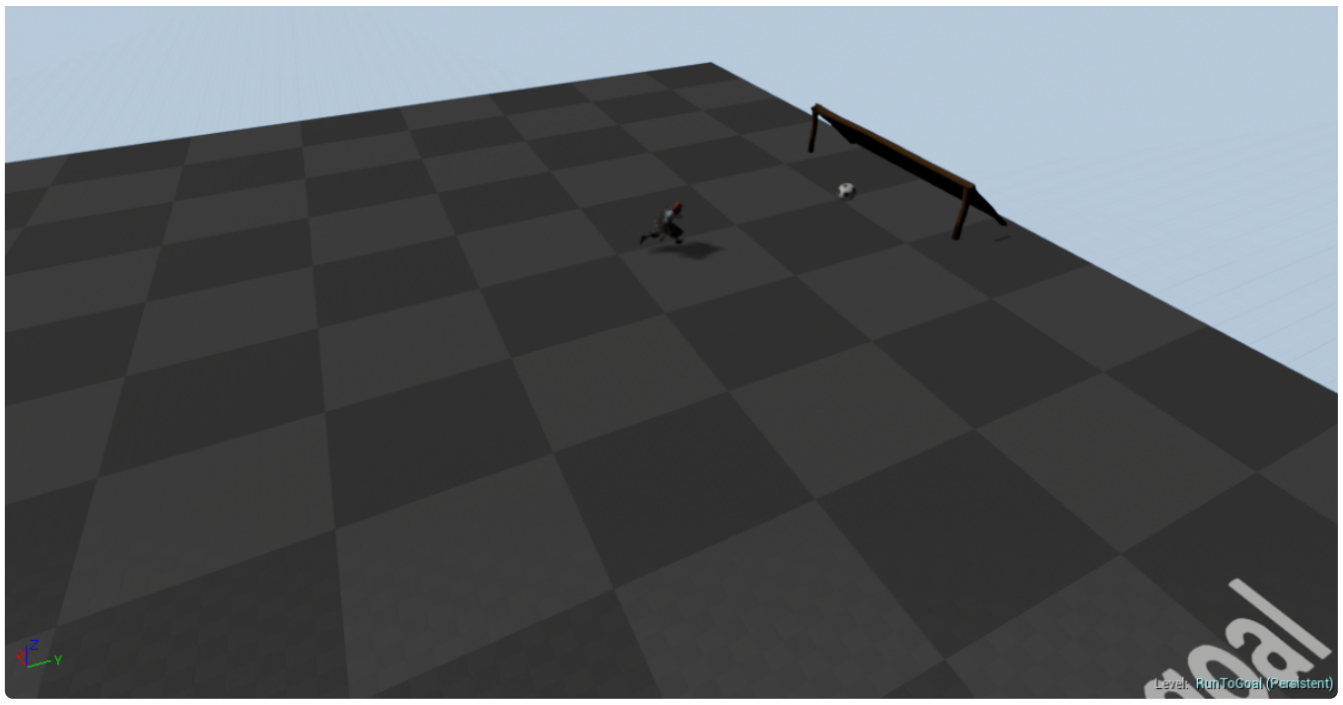

You may notice that the child nodes have "blue things" on them. Those are Unreal Editor-specific elements called "decorators." You can use them as a switch to tell the traversal whether it should iterate through the tasks for that child node. Here's the complete blueprint for my decorator that informs the traversal whether the soccer ball is in a goal:

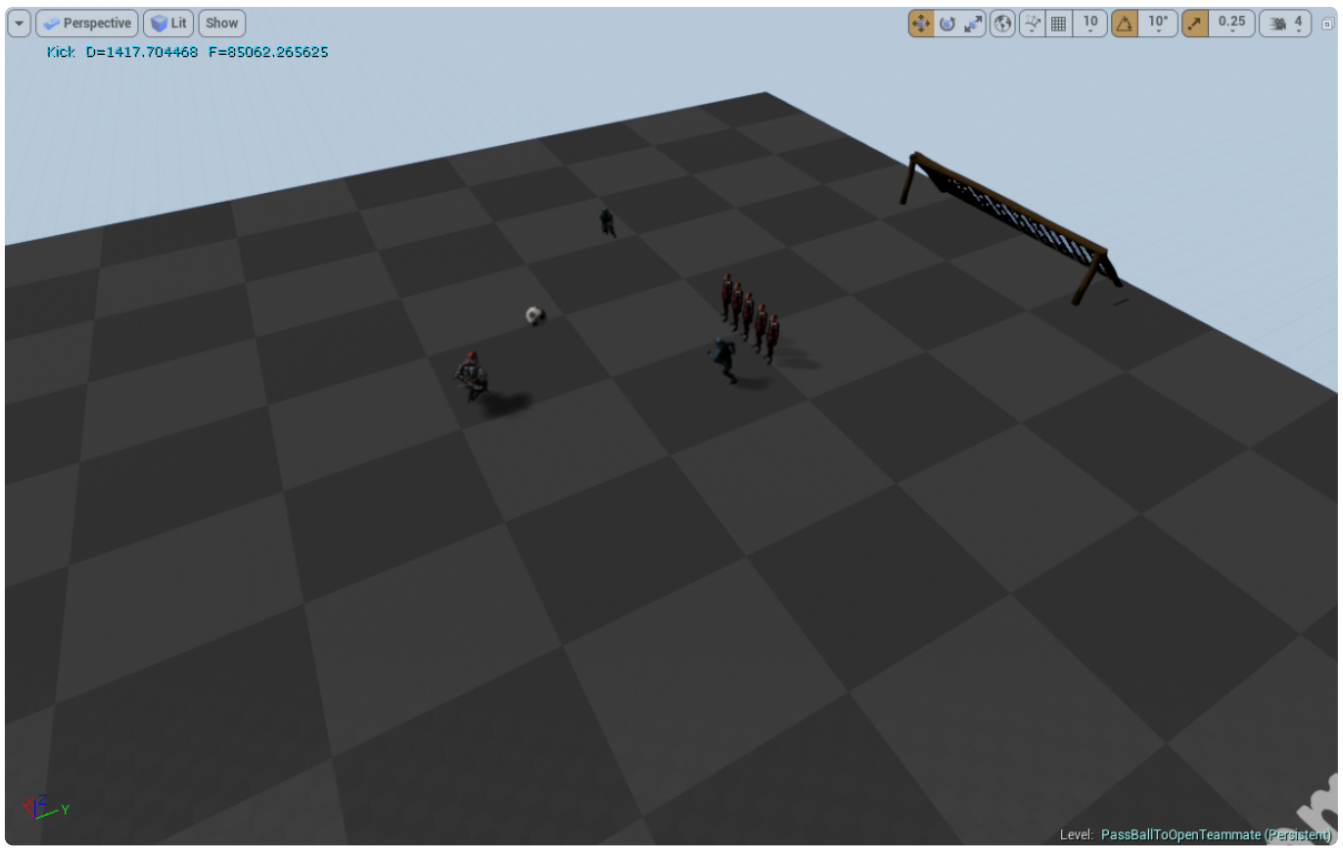


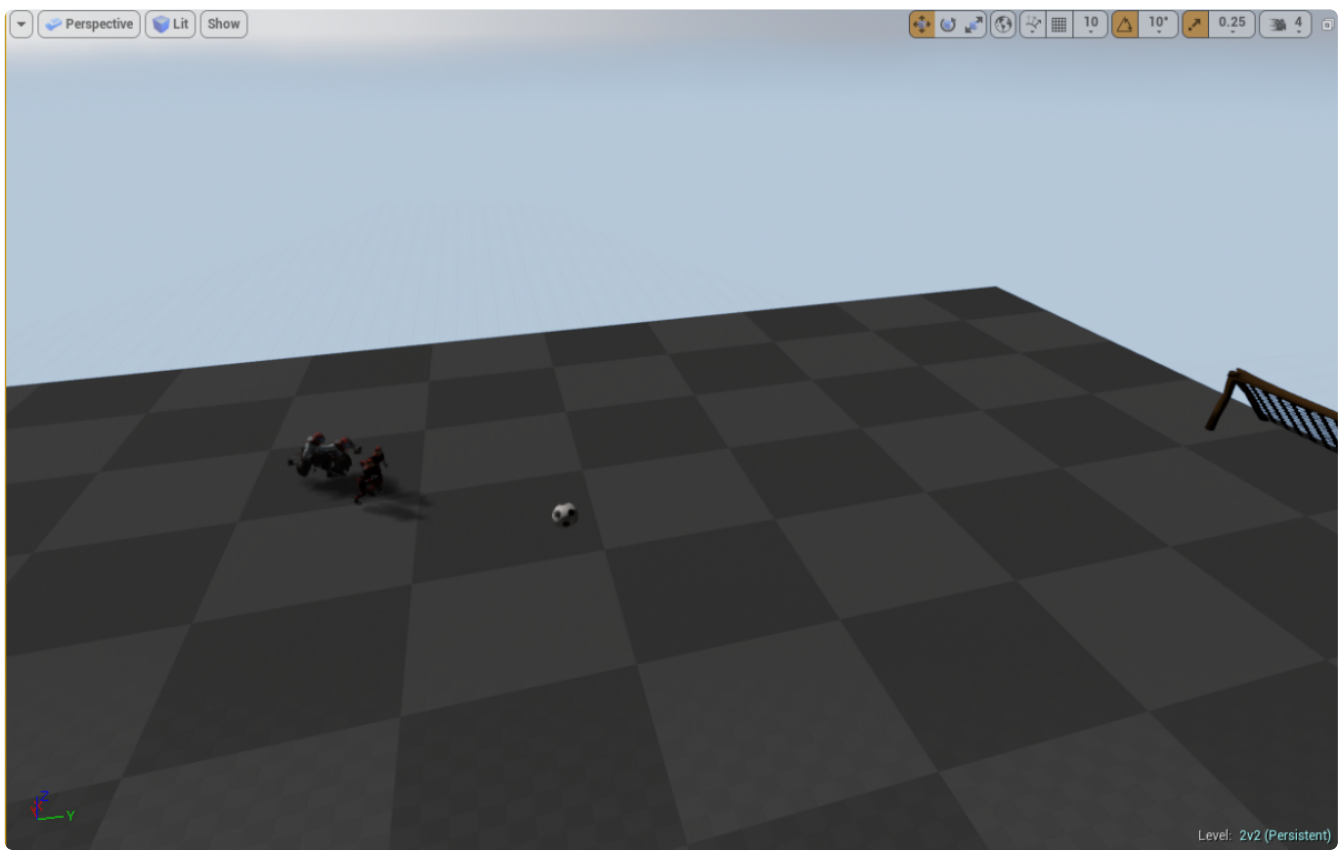

## [color=#b22222]Task Testing[/color]

While developing tasks I bounced back and forth between blueprinting and testing to make sure all my changes worked. It was after all the tasks were written that I got the idea to write manual "Unit Tests." I would create one scene for each behavior tree traversal to verify it worked through manual observation. This definitely helped because two of the "Unit Tests" revealed bugs that would have been much harder to pin down in a full simulation. Here are some poorly lit screenshots of:

1. A player kicking the ball into the net
2. A player passing to a teammate that is farther away from a row of enemies than another teammate
3. A 2v2 clump of players pursuing the ball




I'm aware that in Test-Driven development one is supposed to write unit tests first rather than later...so to all you developers I've offended out there by writing them last: Deal with it! smile.png

## [color=#b22222]Final Product[/color]

Here are some screens and a video of more test scenes and a single team on a field. The screens go in progression from the start of the simulation to the end:

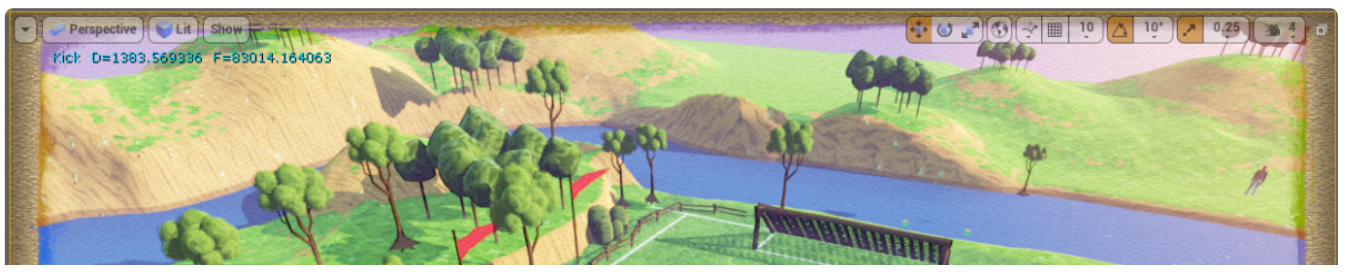
[media]


"Battle Soccer(?)" Unreal Editor AI Pro...

[/media]


Perspective
Lit
Show
Level: Stylized_Kingdom_forts (Persistent)


Perspective
Lit
Show
Kick: D=1114.805054 F=66888.304688
Kick: D=1244.687256 F=74681.226563
Kick: D=1266.532471 F=75991.945313
Kick: D=1283.171143 F=76990.265625
Kick: D=1371.706299 F=82302.382813
Kick: D=1345.075195 F=80704.507813
Kick: D=753.733215 F=45223.992188
Level: Stylized_Kingdom_forts (Persistent)


Perspective
Lit
Show
Kick: D=1383.569336 F=83014.164063


[color=#b22222]

## Implementation concerns

[/color]

- In the game the tree traversal always goes through all child nodes instead of stopping at the first one whose decorator returns true. I need to fix that.
- I created a class inherited from GameMode as a hub for global functions and variables (such as the soccer ball). I suspect I'm not supposed to do this but I don't know a better place for them.
- In each task I cast the owning actor input variable to an AI controller, then call GetControlledPawn, then cast that result to a BotTeammate to get the actual character on the field. It seems like a lot of work to get access to that character class...
- I really wanted the human characters and bots to have identical interfaces so I wouldn't have to do "if (bot) {} else if (player) {}" branches, but haven't figured out how yet.
- The editor recently acquired an incessant desire to rebuild the navigation area every time I start a simulation. It happens after a fresh launch of the editor after I start modifying blueprints.
- This: https://answers.unrealengine.com/questions/49111/bug-trigger-volume-causes-trace-hit-with-no-collis.html . I resolved my issues by having traces only look for pawns. If I ever decide to add static obstacles to the field I'll have to come back to this.

[color=#b22222]

## What's next?

[/color]

Now that I'm satisfied I can develop a basic soccer simulation, the next step is to start designing the game as a whole. As I wrote previously, I intend to have opposing players fight each other for ball possession. Fighting may include melee attacks, ranged attacks, and magic. It may involve launching attacks from a distance, or even team tactics where one player freezes an opponent before the other launches a localized lightning storm at them.

There's also the matter of letting players accumulate resources to spend on armor, weapon and skill upgrades for themselves and their bot teammates. At first I was thinking a Warcraft 3-like deal where you build structures and kill creeps...but decided it would make more sense to have creeps on the field and possibly special item shops (concession stands) on team sidelines. Regular shops could appear in-between matches. Should I have player leveling? Should I allow for instant action matches where all players start at the same level and no items for a competition of pure skill?

There's even the matter of configuring the bot teammates: Team captains could arrange the bot positions in defensive or offensive formations as well as define the aggression level for each bot. Perhaps a team captain would prefer that human players pursue the ball while bots do nothing but attack enemy players for example.

I should probably find some people to help me design all this; preferably those who have a lot of experience with MOBA's and balancing.

Previous Entry
My First Excursion Into the Unreal Editor

Next Entry
How I sometimes fix crashes running my Unreal 4 game in DebugGame mode