Katana - Utility AI for football

Abhikalp Unakal

1 Introduction

The primary objective of this research paper is to implement a utility-based AI that can play football, run simulations and be able to discern how changing the objective functions, and the utility parameters can tweak the player behaviors.

The second objective of this project is to explore utility functions in the context of a game of football and how they can control player characteristics.

A third nuanced yet more practical objective is to demonstrate a novel solution I came up with that lets you exploit preexisting behavior trees and still be able to have all the power of utility AI's without spending upwards of 75USD that some plugins charge you.

2 The beautiful game – football

When a few army men in the British regiment kicked a round ball made of leather for the first time in history, little did they know that the game they had just invented would go on to change the world. and boy! change the world it did.

From an army regiment's pastime to the most popular sport on earth, Football has seen quite several "Firsts". The first official fixture was played in 1982. The first football club ever formed was Sheffield FC. The first world cup won by Uruguay was in 1930, and the first footballer to be traded in the "millions" of money category was the British footballer Trevor Francis in 1979. Fast forward a few years and hundreds of millions of fans, a Diego Maradona handball dubbed the "The Hand of God" later the world was hooked on to the beautiful game.

The Icing on the cake: FIFA the video game – The best-selling sports game series in the history of video games and a huge motivation for this project.

3 Dissecting the game dynamics - football

Now, football is a complicated game. The game is won by the team that scores the most goals, but there are quite a few ways a team can achieve this feat. A team can either choose to "park the bus" - defend deep and hit on the counter - or play "tiki-taka" - keep possession and create spaces. Some teams rely on set pieces, while others look to create chances to score from the wings.

The earlier formations were top heavy and fielded up to 7 attackers at a time, as the game modernized, and rules changed, teams became more balanced. From Herbert Chapman's "W-M" formation, to Portuguese tactician Jose Mourinho's fabled 4-2-3-1, or Sir Alex Ferguson's 4-4-2, there are many ways to win a game. Each formation requires

different tasks and actions from the players implementing it, and it is generally true that a great team made of inferior players will win more times than an average team made of great players. It can be concluded that each player must be in perfect sync with the team to produce optimal results.

The game dynamics lend themselves for a perfect use case in utility based AI simulation strategies, this is because at any given point of time a player needs to be able to make quick decisions between a wide variety of competing actions – should I pass ? with what force ? should I dribble and push ahead ? should I be defensive ?? In the real world of course, this happens in a continuous manner and there are 11 players on each side, and they try to score goals whilst simultaneously making it harder for the opponent to score. But as the saying goes in video games is like solving hard problems repeatedly with the only difference being, you are solving them 60 times a second.

In each game we need to identify the key elements that are in play and we need to keep track of **goal boundaries, team half vs opponent half pitch, field boundaries, closest team member, openings in opponent formation, ball progression, current score, weather conditions**(weather might affect visibility and velocity pitch conditions)**, distance to goal, goal angle, watch team body language, call for passes, lofted shots, lob shots, through balls, pincer moves, misdirection, kick force, playbook formations,** there's just a huge number of things for a player to keep track of at given point of time.

However, if we want to be able to simulate this then things start escalating quickly because we need to drill down to the essence of this inherent dynamism present in the game. We need to be able to break things down into concepts and blocks that will have commonalities across most of the players. A realization at this point for the technically avid might be that simulating football essentially boils down to a whole lot of complex geometry and physics. But if have any hope of maintaining sanity then we can make things simpler and create layers of abstraction for ourselves. What it essentially boils down to is a whole bunch of **vectors! vectors and vectors!**

So, we need to think in terms of vectors the angles and distance to intended attack or defend target. Once we do that, we can start going up the ladder of abstractions and start thinking about formations and flow fields and the rest. But the core AI can be demonstrated without adding in the infinite labyrinth of formations once we get this working we can move on to formations as being part of the utility functions.

4 Contextualizing the gameplay dynamics - experimental setup

Having talked about the beauty and the dynamics of the game, we now have some grasp of the complexities we are trying to deal with here. The next task we have is to create an environment an experimental setup if you will that will allow us to run simulations. To do this we are going to drill down to the most basic of all elements.

To demonstrate the different behaviors, we will basically need 1 player that defends and another player that attacks on both sides of the team. An interesting addition to the setup however is the addition of corner blockers. This is because on several simulations I observed that players would sometimes chase the ball right into the corner and form sort of a grid lock that they procedurally will not be capable of breaking out of. So, adding in corner blockers

that ensures obtuse angles all around the field makes sure we sidestep this issue.

We will also need to have a goal scoring mechanic and the goal lines to detect when the ball crosses the line and it is considered a goal. We are going to start simple with 1 player on each end and then gradually build out functionality by adding in another player and see how the players interact.

Another key element we will have is the utility intent bars for each player, these are going to be the key element for this research project. Essentially there are 4 different intents **Attack, Defend, Dribble, Kick.** There will be more explanation and a detailed deep dive into each one of the intents in a later section.

To summarize the experimental setup is going to consist of these key elements.

- 2 Goalkeepers
- 2 Attackers
- 2 Goalposts
- 1 Scoring Mechanic
- 1 Football
- 1 Field boundary lines
- 1 Mid field line
- 4 Corner blockers
- 4 Intent indicators for utility
    - Attack
    - Defend
    - Dribble
    - Kick

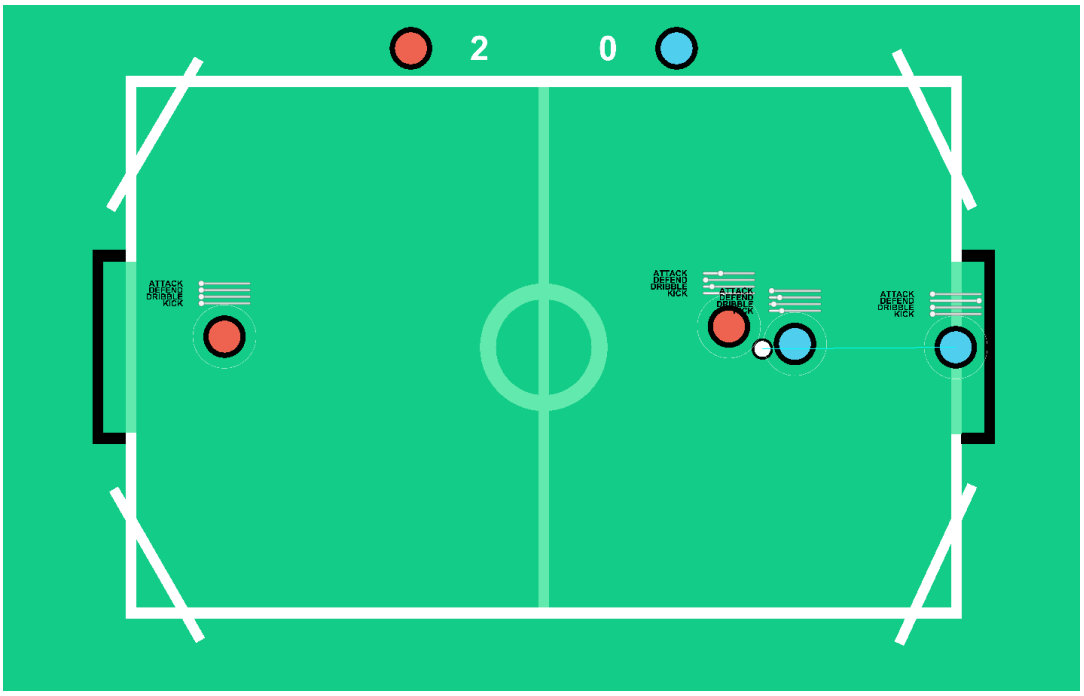


Figure 1 The experimental setup.

5 Setting the stage – What are Behavior Trees?

Behavior Trees (BT) were this way of thinking about NPC behavior that was originally invented by Damian Isla back in 2005 for use in the HALO 2 AI. It is an AI architecture that solves many of the drawbacks of finite state machines like difficulty in modeling the n^2 transitions across all the states in the worst case.

The simplest way to describe a BT would be that they consist of action/leaf nodes and control flow nodes that enable traversal across the tree.

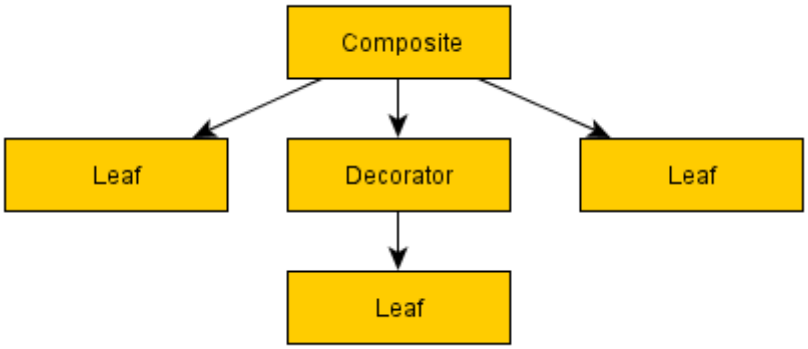


Figure 2 The General structure of a Behavior Tree [BT2].

Let us understand what each of these terms mean, and see a simple example of BT.

- **Leaf** nodes are the nodes where the actual action and task that an agent needs to perform get executed this is where we will be putting our player behaviors for example `move_left, move_right, kick_ball, do_dribble.`
- **Decorator** nodes are the nodes where we usually run some sort of simple logical checks or maybe invert the result of a child node or try to always set a node to fail for example `always_succeed, inverter.`
- **Composite** nodes are the control flow nodes where we usually decide how to evaluate the children or what is the next node that gets executed. Composite nodes come in several different flavors
    - **Selector :** returns true if any one of the child nodes are true. The children are executed in the order that they are declared in sort of like a priority list of actions that can be taken.
    - **Sequencer :** returns true only if all the children return true. These kinds of control flow nodes are useful to ensure that a preset list of tasks or actions get executed in a sequence.
    - **Parallel :** executes all the child nodes at the same time and then usually returns false if all of them are false, however we can set an execution policy depending on our preference.
    - **Random Selector :** selects a child node at random executes it and the status is whatever the child node returned

6 Exploring the waters – Are Behavior Trees going to cut it?

Having talked about the dynamics involved in a football game and a brief glance over behavior trees, let us actually consider a simple hypothetical scenario, build a behavior tree for our player and see what we can glean from this exercise.

To make our experiment easier let us assume that the player (blue) always aims at the center of the goal and the goalie has decided to go out for a stroll. Now given that the player has actions such as **Kick, Attack, Defend, Dribble** let us see if we can create a simple behavior tree that instructs the player what action to take at any given point in time. Our behavior tree is going to look something like the one shown below in Figure 3A.

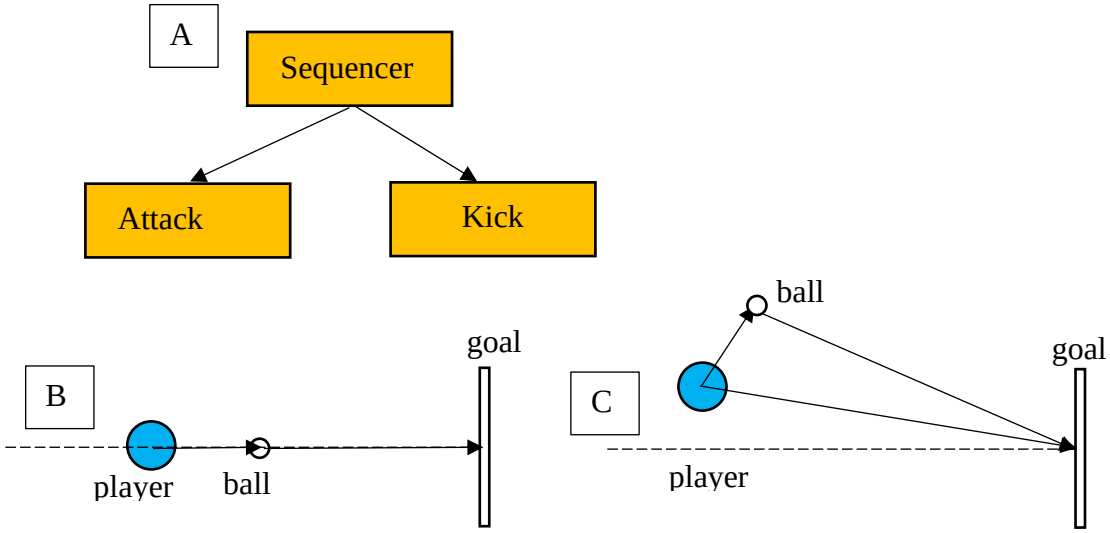


Figure 3 A Behavior Tree Exercise to illustrate some drawbacks. In Figure, [A] the behavior tree, [B] case 1 ball and player aligned with goal, [C] case 2 player and ball not aligned with goal.

If we use the above logic the player will attack or in other words go towards the ball, and then kick the ball on reaching it, consider the naïve case 1 in Figure 3B, the player and ball are both aligned with the goal and this strategy will work. However consider case 2 in figure 3C the ball and player aren't aligned so maybe the player might have to now dribble first align the ball by kicking it in the right direction and then start attacking towards the ball, in a straight line towards the goal.

This was just a toy example but you get the gist, if we encounter situations where sometimes we might have to do action A and then action B, but if the situation was slightly different we need to do action B and then action A, if we have to use a plain behavior tree then since the BT is so hard baked, we either need to create monstrous BT's to account for every single possibility which is just not feasible, or we need something better – something slightly more dynamic in the way we prioritize our actions in the child nodes. That something is a utility function that will let us prioritize actions based on utilities. Let us take a look at these.

7 The Build-Up – What is Utility Theory?

Consider having multiple nodes at a given level and you need to be able to pick the best one amongst them currently based on the constraints of the current game state *Utility theory* is simply the process of measuring the worthiness or score of an action.. The actions that any agent or player might need to take at a given point of time in a complex game like football are never clear 0/1 black and white binary decisions they are instead a spectrum that can sometimes be higher or sometimes be lower [UT2].

The hard part is determining what are the parameters that need to be considered to score a certain task and to ensure that we can reliably compare different scores from multiple tasks reliably. What this means is that we need to be able to identify various information or input sources and be able to combine them in some manner to make a final decision on the action that we need the player to execute at any given instance.

Let us consider a toy example again of a mouse trying to grab some cheese, but at the same time there is a cat that patrols the corridor leading up to the room of cheese. Let us consider the utility below that would result in a decision of whether the mouse runs towards the room of cheese or waits in its hole.

$$Hunger = e^{(Time\_since\_last\_meal)} \quad (1)$$

$$Danger = (Time\_since\_last\_patrol)/(Distance\_from\_cat)^2 \quad (2)$$

$$Utility = (Hunger*Max\_run\_speed) - (Danger*Stay\_intention) \quad (3)$$

We are going to assume that we normalize values for all entities involved. The reason for this is simple we cannot compare apples to oranges – we need some way to be able to standardize the scores so we can compare them.

An elegant way to think about how utility theory can work for us to think in terms of needs and wants. What we need vs what we want, in the above example the mouse needs to satisfy hunger i.e. it needs cheese, but what it wants is to also be safe. So, as its need to eat goes up so will its desire to be safe start mattering lesser as it would die without the food anyways. This simple concept of being able to in some way evaluate or score an action before actually deciding to commit to an action is very powerful and can add a layer of expressiveness and dynamism to an existing concept like BT.

8 Diving into the deep – Utility functions for football

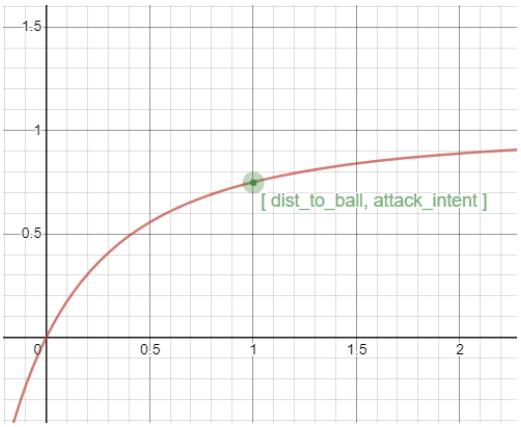
We have seen how powerful the concept utility theory can be, and we have also talked a bit about the dynamics of football and how everything essentially boils down to vectors. Now we are ready to take a deep dive and start designing Utility functions for our player.

As mentioned before in the experimental setup phase each player is going to have 4 different intentions at any given point of time – **Attack, Defend, Dribble, Kick.** To ensure there is some clarity on what these terms mean and the context that I am using them for, I am going to define what each of them does and which actions they are responsible for.

- **Attack:** this will refer to the behavior where the player chases after the ball.
- **Defend:** this will refer to how defensive goalkeeper like behavior our player will exhibit.
- **Dribble:** this will refer to the behavior where our player aligns so as to put the ball and the goal in a straight like – think of it like course correction and will evaluate where to intercept the ball.
- **Kick:** this will refer to the behavior of kicking the ball with a set amount of force.

Let's get right down to it and design our first utility function that controls the players Attack intent, the distance value was normalized wrt a *max_distance* threshold, and the resulting Attack intent score was also normalized to be within [0,1]. For all the graphs that follow we are interested only values in the [0,1] window on both axes. All the graphs below have associated interactive demos – links are provided in the last section of references.

8.1 Attack Intent



$$A(x) = \left(1 - \frac{1}{(1+x)^2}\right) \cdot + c \qquad (4)$$

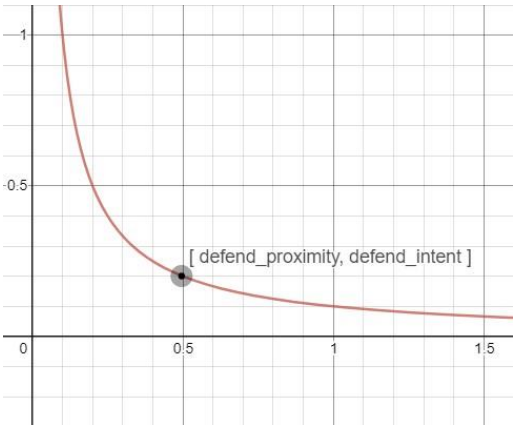
Figure 4 Utility Function for the Attack intention, and equation (4).

x Over here represents the normalized distance to ball from the player. And *c* is some arbitrary constant to make shape our curve according to player aggressiveness.

The intuition that this curve gives us is that if the player is farther away from the ball his intent to attack is a lot higher, than when compared to if the player was closer to the ball. This makes sense if you think about it – if we already have the ball we don't really need to chase after or attack after the ball anymore. But if the ball is further away from us then we definitely might want to chase after the ball.

Observing numerous matches where the players try and intercept the ball and chase after it the players usually slow down as they approach the ball, this utility function demonstrates similar characteristics in that sense.

8.2 Defend Intent

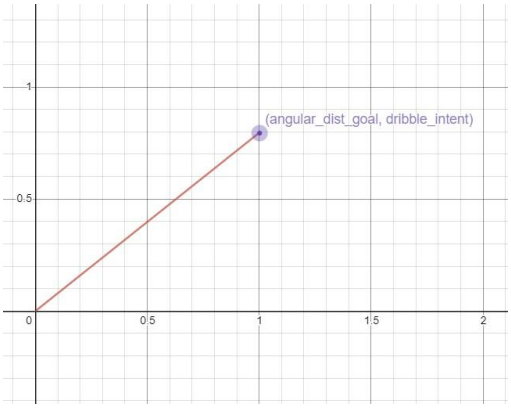


$$B(x) = \frac{d}{x} \cdot a \qquad (5)$$

Figure 5 Utility Function for the Defend intention, and equation (5).

d Over here represents the defense activate radius - this is the radius beyond which there is the literally no need to act defensive because the ball is outside your danger zone. The defend radius is measured with respect to your own team goal. *a* Over here represents the aggression value of your defend behavior. As before all values are again normalized and clamped between the ranges [0,1]. *x* Over here represents the distance to ball with respect to the player. The curve is again nonlinear exponential if you observe a few football matches you'll notice that the plays per minute (PPM) are higher closer to the goal and hence the defensiveness intent needs to be significantly higher – an exponential curve models this well.

8.3 Dribble Intent



$$D(x) = a \cdot \frac{x}{\pi} \ \{0 \leq x \leq 1\} \qquad (6)$$

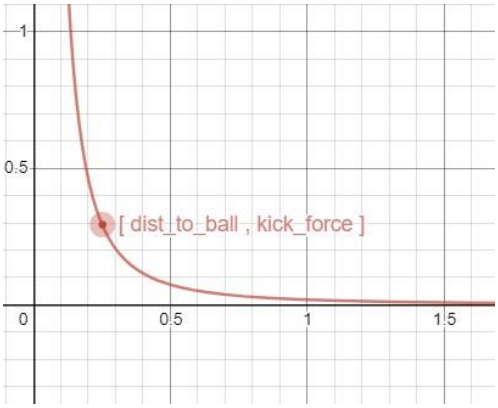
Figure 6 Utility Function for the Dribble intention, and equation (6).

The Dribble intent is basically a measure of how far out your current velocity is from the vector that is the direction of the ball with respect to your opponent's goal the one where you need to kick the ball to score.

Think of this intention as a desire to course correct if the players current direction of heading is not aligned towards the goal, this usually happens in a game of football where the players will dribble the ball to a point and then circle back and align themselves in a straight line towards their target which could be either the goal, or towards a team mate if they are aiming for a pass. In our case since we have just 2 players on each team the player will be aiming towards the goal.

This is again normalized to a value in the range [0,1] however this is normalized with respect to the angular displacement error between your current heading and your desired heading. Hence the normalization occurs with respect to the maximum angular displacement of π radians. So a linear curve models this particular dribble intent fairly well, as we need to impart as little angular displacement on the ball as possible.

8.4 Kick Intent



$$K(x) = \frac{1}{x^2} \cdot a \cdot b \qquad (7)$$

Figure 7 Utility Function for the Kick intention, and equation (7).

The Kick intent is again a nonlinear curve that basically depends on how far away the ball is from you. *x* Over here denotes the normalized distance of the ball wrt to the player, *a* here represents the aggression valued [0,1] of the player. *b* represents the ball progression as a normalized value from [0,1] ranging from 0 at your goal to 1 at the opponent's goal.

The Kick behavior is similar to the attack curve in the sense that it still grow proportionally to the square of the distance, but the subtle difference between these 2 being that as the player nears the ball and the ball falls within the kick radius of the player his intention to kick goes further up.

Also, if you notice the parameter *b* here (ball progression) will skew the player to start kicking more as he gets closer to the goal and this is exactly what happens in a real football match. Players are much more likely to take chances and shoot for the goal if they are nearing the opponent's goal. However, the distance is squared as in the case of Attack because the drop off and surge needs to be high and low within a short span of time.

This is to ensure that no particular action has an influence for a longer period than is

absolutely necessary. The *a* (aggression) parameter again makes things interesting as it acts a multiplier that amps up and down a players kick force and this is done by make the Kick intent rank higher in comparison to others by means of the aggression multiplier.

All these curves were made using *Desmos* as a graphing tool and these curves were arrived after several iterations, the really fun part is I've made these interactive graphs available on *Desmos* - the links to each of these interactive graphs can be found in the references section of this paper.

9 The Reveal – Katana

We have seen how the dynamics of football can be modeled using *utility functions* and we have also seen how behavior trees work, ideally we would like to know how we can combine these concepts together and create a functioning Utility AI.

```
1  tree("root")
2      fallback
3          sequence
4              scorer [determine best]
5              node_1
6              node_2
7
```

Figure 8 The Scorer.

Behavior Trees by their design lend themselves to extension pretty well, so the novel method that I created is what I call Katana. The ingenuity of this approach is that you don't need to create a custom new utility selector that enables you to select the best action.

The way this works is simple – we use a block that is structured as above – the sequence node as the parent and the first node is the scorer node, following that are the child nodes that we want to be able to dynamically score and select from, additionally each node has a *taskid* of their own – we will see why in a minute.

The scorer node works as follows – we first create corresponding scoring functions for each of the nodes that come after the scorer – the scoring functions evaluate each action and return a value in the range of [0,1]. The execution policy. We can define a function that describes the combination policy for the scorer node. The combination policy that I have chosen is *max_combine* which picks the max scoring node.

Once each scoring function returns a score – we pick the maximum amongst these and we then set the corresponding node's *taskid* as the best one so far. Post this when the BT ticks and progresses on to the child nodes – we check if the best *taskid* matches the current node's *taskid* and only they match we execute the current node. Irrespective of if the current node is a match we always set the current node execution status to true – this is because the parent node for all the children nodes is a sequencer which will succeed only if every child succeeds.

10 The big picture – How to go about building Utility AI ?

The entire project was built using unity, and I used C# to write the scripts – the behavior tree functionality itself was implemented using a plugin called BT. Right now, there are two different behaviors that we can assign a player. Either a chaser or a goalie. And the players try and score goals against each other.

Most of the experimental setup was described in a previous section. The scoring utility functions themselves can be found in the code listings section below – it includes code that is used to score each action, the scorer logic itself and the main setup. I have strived to keep all the non-essentials to the basic – like shifting into 2D instead of 3D, all players, goals and the ball are 2D rigid bodies and the ball has linear drag associated with it to simulate friction between the ball and the field.

The entire process can be broken down into these following phases

1. Create Experimental setup and add controllers to agents.
2. Figure out actions for the agents.
3. Create helper functions that form bigger actions.
4. Figure out inputs for your utility math.
5. Brainstorm and come up with Utility functions.
6. Create scorers for each separate utility that you wish to emulate.
7. Figure out how you would like to combine scores.
8. Create nodes with desired functionality.
9. Test check results against human and iterate from 2 onwards.

11 Curtains drop - Conclusion

Utility Theory being applied and put to use in behavior trees is a very neat solution that creates expressive dynamic AI's and demonstrating it's use in a football environment just shows the power that such an architecture wields. At its core *Utility Theory* is a way for us to score a set of actions and then combine the scores to select the most appropriate action at any given instance.

Some key takeaways from this research paper is that *Behavior Trees* are for more extensible that you would think at first glance. You might want to use some sort of graphical tool like *Desmos* to model your utility functions. Evaluating actions before executing them is a powerful concept that can be applied in many other contexts as well. This project mainly focuses on the dynamics between an attacker a goalkeeper and trying to score vs defend goals, however the framework described in this paper is pretty extensible so adding in a 3rd behavior for let's say the mid fielder isn't much of a hassle.

12 Show me the code – Code Listings

Listing 1 Function to score the Attack Intent

```
public float score_attack ()
{
    float score = (1 - (1 / (distance*distance)));
    return score*attack_multiplier;
}
```

Listing 2 Function to score the Attack Intent

```
public float score_defend()
{
    float score = 0.0f;
    if((pos_ball-pos_team_goal).magnitude < defend_distance)
    {
        score = defend_distance/
                (pos_ball - pos_team_goal).magnitude;
    }
    return score*defend_multiplier;
}
```

Listing 3 Function to score the Attack Intent

```
public float score_dribble()
{
    float angle_current_desired = Mathf.Acos(
        Vector2.Dot(dir_me_to_ball, dir_ball_to_goal));
    // normalize between [0,180] degrees OR [0,PI] radians
    float score = aggression*angle_current_desired/Mathf.PI;
    return score*dribble_multiplier;
}
```

Listing 4 Function to score the Attack Intent

```
public float score_kick()
{
    float ball_progression = dist_ball_to_team_goal /
        (dist_ball_to_team_goal + dist_ball_to_opp_goal);
    float score = (1/(dist_me_to_ball* dist_me_to_ball)) *
        (aggression) * (ball_progression);
    return score*kick_multiplier;
}
```

13 Gratitude is important - References

No great project ever got done alone – and I am grateful to the below people for their research and thoughts which helped me implement my project. The last section here also includes links to the interactive graphs that I build and used to create the utility functions.

13.1 Chapter in multiauthored book

[BT3] The Behavior Tree Starter Kit – Alex J. Champandard and Philip Dunstan http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter06_The_Behavior_Tree_Starter_Kit.pdf.

[UT2] Building Utility Decisions into Your Existing Behavior Tree - Bill Merrill http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter10_Building_Utility_Decisions_into_Your_Existing_Behavior_Tree.pdf.

13.2 Electronic Research Paper

[NN1] Adding Neural Network Controllers to Behavior Trees without Destroying Performance Guarantees - Christopher Iliffe Sprague, Petter Ögren https://arxiv.org/abs/1809.10283.

13.3 Online Documents

[BT1] GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI – Damian Isla https://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php?page=2

[BT2] Behavior trees for AI: How they work https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

[UT1] Are Behavior Trees a Thing of the Past? - Jakob Rasmussen https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php.

13.4 Interactive graphs

[DZ1] Attack utility – Abhikalp Unakal
https://www.desmos.com/calculator/7oqxlfufyi

[DZ2] Defend utility – Abhikalp Unakal
https://www.desmos.com/calculator/ewt0ryened

[DZ3] Dribble utility – Abhikalp Unakal
https://www.desmos.com/calculator/doa2hrmcdm

[DZ3] Kick utility – Abhikalp Unakal
https://www.desmos.com/calculator/cpfuyb2h2a