DAC619 AE1 Report

Q12975371

James Johnson

2019

Contents

[Chosen Algorithm: Goal Driven AI 2](#_Toc26517657)

[Design 4](#_Toc26517658)

[Goals 5](#_Toc26517659)

[Evaluators 7](#_Toc26517660)

[Data Evaluators 10](#_Toc26517661)

[Additional Helpers 11](#_Toc26517662)

[Testing Plan 12](#_Toc26517663)

[Test Data 13](#_Toc26517664)

[Critical Evaluation 20](#_Toc26517665)

[Bibliography 21](#_Toc26517666)

[Appendices 22](#_Toc26517667)

# Chosen Algorithm: Goal Driven AI

For this game type I’ve chosen to make use of Goal Driven AI to handle the bot’s behaviour. The reasons for this are many. First of all, Goal Driven AI allows us to give the bots an ability to logically reason and weigh up a decision based on a series of factors expressed as a simple equation which leads to its utility. After having weighed up these decisions it can then select the most desirable of these to then put into action and fulfil. It does either by selecting an Atomic or Composite Goal. Atomic Goals allow it to do one action whilst Composite Goals allow it to create more expressible plans which can add sub goals that lead to the overall behaviour decided upon. This allows the design of the AI to be easier when creating and avoids the stiffness given by that of State Machines which can only have fixed transitions from state to state instead of comprising new strategies as it evaluates them. (Buckland, M. 2004)

For a multiplayer game in the modern age player’s look more and more to have an experience with AI that mimics good logical reasoning and doesn’t simply perform an action because it’s reacted to some state or some binary choice. It instead uses a series of grey values taken from itself and the world state to help inform a decision to an action. Much like a decision to risk running to a medic across a field, instead of a simple binary choice which leaves little room for changes we can instead utilise a more averaged and approximated gauging of the decision by weighing it’s utility and applying this to our overall decision on which outcome would be best for us at the present moment. The affordance of these utilities allows a way to give better expression and lifelike choice to a bot which would help increase the interest in a multiplayer game without the error prone nature of a burgeoning behaviour tree with changes added causing limitations upon design with ideas being locked for fear of restricting the tree. (Merrill, B. 2013)

Another positive to its usage is that of performance and scalability. For a multiplayer game iteration time can be a rapid concern as behaviours should be tried, tested, tweaked and discarded or kept as fast as possible with player testing informing what works and what doesn’t. Designers need to be able to communicate with AI developers about what’s needed and using utility and goals helps to remove abstract concepts and allows for more in-game statements to be referenced directly. And with the use of utility values, we can quickly tweak and add or remove scorers of behaviours to get the results we desire without overly complicating or restructuring vast tree’s or remodelling complex state graphs as theirs nothing to break and more can be added or removed as needed. And because of this, if the evaluations are kept fairly simple it can be far more performant to make decisions as deep recursive/looped decisions don’t need to be made. (Graham, D. 2013)

Curves are another useful addition as they allow for better prioritising of desires and can even be used to help add character and personality to an AI. Through the use of curves decisions can be tweaked to allow items to pull on a bot as it nears it or help a bot to understand not to grab ammo whilst it still has above a certain threshold. These utility values can then be used to search up other agents’ scorers and given this bot’s values it can attempt to estimate what the other agent might do in its scenario thereby giving it the ability to have some fore-planning and strategy with relative ease which make them far more emergent than the limited world of FSM’s and the deep nested sub trees of complex actions in Behaviour Tree’s. (Buckland, M. 2004)

With these points in mind however, goal driven utility AI does have it’s share of negatives. Given that it’s a very designer orientated approach to AI it can be let down by the use of incorrect data or scoring from its designers which means that it’s fuzzy-ish approach to decision making can lend itself to making dumb or strange decisions with harder ways to debug it due to the values being updated and changed constantly. Its better performance can quickly be offset with repeated use of intensive operations on its data to determine scoring and can lead to very bad performance in unchecked conditions. Finally, for it to make correct decisions it needs constant adjustment and tweaking through playtests which can lead to long cycles of tweaking. Whilst it allows for rapid iteration this is both a gift and a curse. (Rasmussen, J. 2016)

For these reasons it made sense to me to use a Goal Driven, Utility based approach in the creation of the AI for this project as it will make scalability of the overall behaviours easier to manage and control and will help provide a more logically reasoned multiplayer bot and whilst it can lead to complex and constant adjustment of scorers it feels like it leads itself better to a design approached bot.

# Design

The following section will detail and go over my design and implementation of a goal driven AI solution.

When implementing a goal driven AI algorithm, I decided to research into relevant solutions others had created. Having looked into about a dozen implementations I decided to then create my own with a basis on an implement created by Mat Buckland as shown in Programming Game AI by Example. With this in mind, I then first broke down what it was I wanted my AI to do into a series of composite and atomic goals along with deciding what type of decisions I wanted to evaluate. This is shown in the table below.

***Figure 1: Goal Table of all possible goal’s an AI can take along with the evaluators it uses for scoring decisions.***

|  |  |  |
| --- | --- | --- |
| **Composite Goals** | **Atomic Goals** | **Evaluators** |
| Goal\_GetFlag | Goal\_AttackEnemy | Evaluator\_AttackEnemy |
| Goal\_ScoreFlag | Goal\_CollectFlag | Evaluator\_GetPowerup |
| Goal\_Heal | Goal\_DropObject | Evaluator\_UsePowerup |
|  | Goal\_FindFlag | Evaluator\_Search |
|  | Goal\_GetItem | Evaluator\_GetEnemyFlag |
|  | Goal\_Globals | Evaluator\_RetrieveLostFlag |
|  | Goal\_MoveToBase | Evaluator\_ScoreEnemyFlag |
|  | Goal\_Search | Evaluator\_ScoreFriendlyFlag |
|  | Goal\_UseItem | Evaluator\_Heal |

After having broken down my AI into what their actions and decisions were, I then drafted up a rough overview of what each goal would propose to do and what data each evaluator would make use of and the equations they would use.

## Goals

Goals are the specific actions that the AI will take when selecting them based upon their evaluators. Goals are split into two types: Composite goals are goals which can contain extra sub goals as steps to carry out and fulfil. The composite goal reports itself as active and completes only when all children have completed and fails if any 1 child fails. If a goal is failed the ***GoalManager*** goes back to the default ***Goal\_Search***. Detailed below is a brief overview of the various goal’s behaviour and why some decisions were made in regard to behaviour.

#### GoalManager

The overarching manager of all goals for an AI agent. Its job is to process the currently running goal and the global goal at specified tick intervals and swap out the current goal with a new one when prompted.

#### IGoal

Interface for all goal classes to implement.

#### CompositeGoal

Abstract base class for all composite goal types providing sub goal processing and handling logic.

#### Goal\_GetFlag

Composite Goal who’s aim is to first find the location of the flag given to it through the use of sub goal ***Goal\_FindFlag*** and then to progress on and attempt to collect the flag when found with ***Goal\_CollectFlag***. The goal fails if either sub-goal is not met.

#### Goal\_ScoreFlag

Composite Goal who’s aim is to move back toward the friendly base along the best path through ***Goal\_MoveToBase*** and then drop the flag into the base with ***Goal\_DropObject***. The goal fails if either sub-goal is not met.

#### Goal\_Heal

Given that many team based CTF games have health pickups that are used immediately I decided to mimic this behaviour with the design of my bot’s AI.

Composite Goal who’s aim is to first move toward the health kit spawner and if it’s there attempt to collect the item with ***Goal\_GetItem.*** And then, if it’s there, attempt to use the health item. The goal fails if either sub goal is not met.

#### Goal\_AttackEnemy

Atomic Goal whose purpose is to move towards and attack the targeted enemy when in range. If target is lost, then it will attempt to select another through the targeting system.

#### Goal\_CollectFlag

Atomic Goal whose purpose is to move toward and collect a flag if one is in view.

#### Goal\_DropObject

Atomic Goal whose purpose is to drop an item in the current location if the agent has it.

#### Goal\_FindFlag

Atomic Goal whose purpose is to head toward the last known position of the flag.

#### Goal\_GetItem

Atomic Goal whose purpose is to head toward the spawner/last known position of the item and attempt to collect it if it exists.

#### Goal\_Globals

Atomic Goal whose purpose is to act as the brain and evaluator for all AI decisions. It will run through each evaluation and pick the goal based on the highest desirability trait.

#### Goal\_MoveToBase

Atomic Goal whose purpose is to move towards the base given until it’s reached the location.

#### Goal\_Search

Atomic Goal whose purpose is to provide fallback idle/flee behaviour and move to random important locations around the map to help place itself into a scenario where it’s utility will help it select a goal.

#### Goal\_UseItem

Atomic Goal whose purpose is to use the item selected if the agent has it.

## Evaluators

Evaluators are the objects responsible for calculating the weight of a decision and returning the goal to use if that decision is selected. As the brain runs through them it finds the evaluator with the highest score and selects it’s goal. Each evaluator is weighted by a tweaker value whose purpose is to strengthen or soften the equation. Detailed below is a brief overview of each evaluators purpose and the data/equation it uses.

#### Evaluator\_AttackEnemy

Evaluator to determine whether or not to attack an enemy.

Attacking an enemy is based upon several factors which are:

* Tweaker Value = k
* Agent’s Current Health = h
* Distance to Selected Target = d
* Agent’s Current Total Strength = s

It makes sense that when an agent is feeling healthy, is strong (based upon it’s powerup status) and is close to the target that it should want to attack them. Below is the equation used to determine the utility of attacking an enemy. Because the normalised score of distance returns 0 the closer we are to an object we invert it’s value by 1 to find it’s true utility for the equation.

#### Evaluator\_GetPowerup

Evaluator to determine whether or not to grab a powerup.

Grabbing a power up is based upon:

* Tweaker Value = k
* Distance to Object = d

This decision is based solely upon the distance the agent is from the object. The equation for this is as follows: (We invert distance to find its true utility as it returns 0 the closer we are to the object.)

#### Evaluator\_UsePowerup

Evaluator to determine when to use a powerup held by an agent.

Using a powerup is based upon:

* Tweaker Value = k
* Distance of Closest Enemy = d
* Number of Enemies in Sight = n

When determining to use a powerup it made sense to factor in that we should only consider using it if we were close to an enemy and it should be more desirable to do so if there are multiple enemies around to maximise effectiveness. The equation for this is:

#### Evaluator\_Search

Evaluator to determine whether or not to search around the world space.

Searching the world space is a fallback option for fleeing/idle behaviour. As such there’s no equation required and is just a constant of 0.05.

#### Evaluator\_GetEnemyFlag

Evaluator to determine whether or not to grab the enemy team’s flag.

Grabbing the enemy teams flag is based on:

* Tweaker Value = k
* Agent’s Current Health = h
* Distance to Last Known Flag Position = d

When deciding whether to grab the flag health plays an important role in the decision. The distance to the flag also factors in as there may be other more pressing things to do if it’s far away. The equation for this is:

#### Evaluator\_RetrieveLostFlag

Evaluator to determine whether or not to retrieve the current team’s flag.

Retrieving the team’s flag is based on:

* Tweaker Value = k
* Agent’s Current Health = h
* Distance to Flag = d
* Flag Distance from Base = b

When deciding whether to grab the team’s flag or not there’s a couple of factors I thought were important to take into consideration. The agent’s health plays an important role as low health is less likely to want to grab it along with how far we are from the flag as from previous evaluations. But another important factor is how far the flag is from home, the further it is away from our base means it’s more likely the other team has it and thus means it’s more desirable to grab it from them. The equation for this is below:

#### Evaluator\_ScoreEnemyFlag

Evaluator to determine whether or not to take the currently held enemy flag back to base and score with it.

Scoring with the flag is evaluated on:

* Tweaker Value = k
* Number of Enemies Around = n

When attempting to score with the flag it’s logical that the number of enemies around the agent would play a role in whether they would want to trek back or not. The equation for this is below: (We invert the number of enemies as the close we get to 0 the more desirable the number of enemies becomes.)

#### Evaluator\_ScoreFriendlyFlag

Evaluator to determine whether or not to take the currently held team flag back to base and score with it.

Scoring with the friendly flag is the same as scoring with an enemy flag.

#### Evaluator\_Heal

Evaluator to determine whether or not to go and heal up with a health kit.

Deciding whether or not to heal is based upon:

* Tweaker Value = k
* Agent’s Remaining Health = h
* Distance from Health Spawner = d

When deciding whether or not to heal I wanted it to be inversely proportional from the amount of health the agent has remaining to how close they to the health spawner. The equation for this is below:

## Data Evaluators

Data Evaluators are used for getting data about the world and the agent in a normalised range of 0-1 to make the evaluation equations easier. Below the are the various data evaluators required for getting information about the game.

#### Health

Returns the given agent’s health as a normalised value between 0 – 1. 1 is desirable with 0 being least desirable.

#### DistanceToPosition

Returns the given position from the agent as a normalised value between 0 – 1. 0 meaning most desirable with 1 being least desirable.

#### DistanceToObject

Same as DistanceToPosition just with a GameObject overload.

#### Strength

Returns the current agent’s strength as a normalised value between 0 – 1. 0 being least desirable and 1 being most desirable. Strength is created as a factor of the agent’s current attack damage from the maximum attack damage they can deliver.

#### NumberOfEnemiesInSight

Returns the number of enemies in the players sight range as a normalised value between 0 – 1. 0 being most desirable and 1 being least desirable.

## Additional Helpers

Along with the main algorithm logic I also designed some helper classes and logic to help aid with behaviours.

#### Targeting System

When creating the AI for this project I want to try and give them the ability to have short term memory of perceived objects. Rather than go too complex into this area as the gameplay’s limiting by default, I plan to implement a basic targeting system that when polled looks at the enemies within the agent’s sight radius and then selects the closest one as a target. If this enemy has left the agent’s sight, then it would still perceive for a fixed amount of time “in memory” before promptly forgetting about them.

#### Helpers

To help aid with the evaluators and with various checks/debugging around the world I plan to create a class to help normalise ranges and clamp them along with shorten distance checks and log formatted debug values as they’ll prove to be useful, these will all be contained within helpers.

#### WorldManager

When creating the AI for this I relayed my knowledge of other common CTF games I’ve played in the past and so I wanted to help recreate the logic needed for the agents to access this information. For instance, in most CTF games the current flag position is commonly shown as a waypoint to players on a minimap or on their screen as well as teammates calling out where they saw it and so I want the AI to have access to the last spotted position of the flag from any of their teammates.

Along with this, players also commonly know where items spawn and so I want the AI to have access to spawner knowledge as well as easy access to their teammates. All of this and more data can be found in WorldManager which will provide an easy interface for all world state values which I feel most experienced players would know in a game.

#### Regulator

When polling the AI for a desirability trait or for updating their logic there’s really no need to poll it as much as the user’s framerate as it’s overkill so I plan to implement a regulator class which would only allow the GoalManager to process a goal at a defined interval.

#### Constants

I aim to add additional constants to the constants file to aid with easy debugging and tweaking of common values such as agent memory length and regulator speed.

# Testing Plan

For the testing of this project I will make use of a black box test plan in which I will be testing each goal as a separate piece and ensuring it works along with ensuring that each evaluator calculation returns the appropriate values, then I will test them together with one another. The only data not included in the test plan is that of value tweaking with the evaluators for desirability as the iteration count is likely to be high and unnecessary to document so evaluators will be deemed as working based on returning the correct value with a fixed set of numbers. This will be detailed within the test plan below.

## Test Data

The next section will show off the test plan with its relevant data filled. (Please look at Test Plan.xlsx included in folder for cleaner and easier to understand table.)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Feature** | **Test** | **Expected Result** | **Actual Result** | **Action** |
| 1 | Goal\_Search | Agent Enters and Processes Goal | Agent should pick random locations in the world and travel to them. Upon reaching them, it should then move to another etc. | Failed as it keeps picking the same position occasionally and will often get stuck in the same spot | Added simple recursive test to ensure currently selected position isn't the same as the one we're at. FIXED |
| 1.1 Retest |  | Agent Enters and Processes Goal | Agent should pick random locations in the world and travel to them. Upon reaching them, it should then move to another etc. | Agent picks random locations in the world and travels to them. Upon reaching them, it then moves to another etc. |  |
| 2 | Goal\_AttackEnemy | Agent Enters and Processes Goal | Agent should have a target and begin closing the distance between them and attacking when it's range. | Fails as the AI will sometimes just get stuck and stop where it is. | Agent loses track of target and then never actually re-assigns it so it gets stuck in goal. Fixed by making agent now re-assign target. |
| 2.1 Retest |  | Agent Enters and Processes Goal | Agent should have a target and begin closing the distance between them and attacking when it's range. | Agent gets a target and begins closing the distance between them and attacking when it's range. |  |
| 3 | Goal\_CollectFlag | Agent Enters and Processes Goal | Agent should move toward and collect the flag when in range of it and return complete when it has collected it. | Agent gets near flag but never does anything when in vicinity of it. | Seems that the collectable list in senses doesn't filter with the flag so the problem is fixed with a new approach in which we fix the basic sensing system by including flags in a collectable search. |
| 3.1 Retest |  | Agent Enters and Processes Goal | Agent should move toward and collect the flag when in range of it and return complete when it has collected it. | Works as intended and flag is collected upon reaching it and complete is returned. |  |
| 4 | Goal\_DropObject | Agent Enters and Processes Goal | Agent should drop given item from their inventory when in range and return complete when it's done this. | Agent drops item from their inventory if they have it and returns complete when it's done this. |  |
| 5 | Goal\_FindFlag | Agent Enters and Processes Goal | Agent should attempt to move toward either the enemy team's base or the last known flag position and then complete when it has reached there. | Agent successfully navigates towards the positions and returns complete upon arrival. |  |
| 6 | Goal\_GetItem | Agent Enters and Processes Goal | Agent should attempt to collect item when in range of it and if not, it should move closer. Upon collection it should return complete. | Agent fails to collect item as it doesn't seem to see it. | Seems that I used the wrong sight list by accident. Fixed by using correct sight list. |
| 6.1 Retest | Goal\_GetItem | Agent Enters and Processes Goal | Agent should attempt to collect item when in range of it and if not, it should move closer. Upon collection it should return complete. | Agent fails as it steals item from each other's inventory without the system ever updating this change. | Seems the bot sensing has an inherent bug in which it never actually checks to see if it can physically see the object before attempted to grab it. Fixed by adding in visual check to provided sensing system. |
| 6.2 Retest | Goal\_GetItem | Agent Enters and Processes Goal | Agent should attempt to collect item when in range of it and if not, it should move closer. Upon collection it should return complete. | Agent successfully navigates towards item and collects it. |  |
| 7 | Goal\_UseItem | Agent Enters and Processes Goal | Agent should attempt to use item if it has it in its inventory. Returns complete if it has it. | Agent attempts to use item if it has it in its inventory. Returns complete if it has it. |  |
| 8 | Goal\_MoveToBase | Agent Enters and Processes Goal | Agent should move toward friendly base and return complete when it has reached there. | Agent moves toward friendly base and returns complete when it's reached it. |  |
| 9 | Goal\_Heal | Agent Enters and Processes Sub Goals and Exits When Finished | Agent should process sub goals by first heading toward health kit spawn location and then grabbing it if it's there. | Agent fails as it stays in moving toward health kit goal | Seems there was a flaw in the sub goal processing of the composite goal abstract class. Fixed by allowing it to remove goals from the stack when they're complete. |
| 9.1 Retest | Goal\_Heal | Agent Enters and Processes Sub Goals and Exits When Finished | Agent should process sub goals by first heading toward health kit spawn location and then grabbing it if it's there. | Agent processes sub goals and heads toward health kit and collects it if it's there. Successfully returns complete and failed when it's collected and when it's not there. |  |
| 10 | Goal\_GetFlag | Agent Enters and Processes Sub Goals and Exits When Finished | Agent should process sub goals by first heading toward las known location of flag and then grabbing it if it's there. | Agent successfully finds and grabs the flag when it's there and returns the correct states. |  |
| 11 | Goal\_ScoreFlag | Agent Enters and Processes Sub Goals and Exits When Finished | Agent should process sub goals by first returning to home base and then dropping the flag when in range. | Agent fails as it heads toward the wrong base. | Seems that I had the wrong base set for move to base when passing it from the evaluator. Fixed by using its own internal tracker of friendly base in the AgentData class |
| 11.1 Retest | Goal\_ScoreFlag | Agent Enters and Processes Sub Goals and Exits When Finished | Agent should process sub goals by first returning to home base and then dropping the flag when in range. | Agent succeeds and drops flag when in base. |  |
| 12 | Evaluator\_AttackEnemy | Fed a fixed set of values | Returns 0.354 with input data | Returns 0.354 |  |
| 13 | Evaluator\_GetPowerup | Fed a fixed set of values | Returns 0.629 with input data | Return 0.629 |  |
| 14 | Evaluator\_UsePowerup | Fed a fixed set of values | Returns 0.19 with input data | Return 0.19 |  |
| 15 | Evaluator\_Search | Fed a fixed set of values | Returns 0.52 with input data | Returns 0.52 |  |
| 16 | Evaluator\_GetEnemyFlag | Fed a fixed set of values | Returns 0.37 with input data | Returns incorrect result | Equation was done wrong. Fixed by wrapping 1-d inside of parenthesis to ensure it happens before the multiplication |
| 16.1 Retest | Evaluator\_GetEnemyFlag | Fed a fixed set of values | Returns 0.37 with input data | Returns 0.37 |  |
| 17 | Evaluator\_RetrieveLostFlag | Fed a fixed set of values | Returns 0.75 with input data | Returns 0.75 |  |
| 18 | Evaluator\_ScoreEnemyFlag | Fed a fixed set of values | Returns 0.2 with input data | Returns 0.2 |  |
| 19 | Evaluator\_ScoreFriendlyFlag | Fed a fixed set of values | Returns 0.2 with input data | Returns 0.2 |  |
| 20 | Evaluator\_Heal | Fed a fixed set of values | Returns 0.681 with input data | Returns incorrect result | Equation was done wrong. Fixed by wrapping 1-h inside of parenthesis. |
| 20.1 Retest | Evaluator\_Heal | Fed a fixed set of values | Returns 0.681 with input data | Returns 0.681 |  |
| 21 | Health | Fed a fixed set of values | Returns 0.5 with input data | Returns 0.5 |  |
| 22 | DistanceToPosition | Fed a fixed set of values | Returns 0.16 with input data | Returns 0.16 |  |
| 23 | DistanceToObject | Fed a fixed set of values | Returns 0.16 with input data | Returns 0.16 |  |
| 24 | Strength | Fed a fixed set of values | Returns 1 with input data | Returns 1 |  |
| 25 | NumberOfEnemiesInSight | Fed a fixed set of values | Returns 0.33 with input data | Returns 0.33 |  |
| 26 | Goal\_Globals | Evaluators selected correctly | Agent should loop through all available evaluators and select the one with the highest desirability | Agent successfully manages to select correct evaluator with highest desirability |  |
| 27 | Goal\_Globals | Goals changed correctly | Agent should change current goal to desired goal correctly | Fails as agent keeps changing its desired goal too frequently never giving it a chance to actually perform the one it's in. | Fixed: By simply adding a check to determine if the desired goal is the one, we are already in, if it is, there's no point switching it. |
| 27.1 Retest | Goal\_Globals | Goals changed correctly | Agent should change current goal to desired goal correctly | Agents change current goal to desired goal correctly and don't bother if we’re already in it. |  |
| 26 | Combined Test | Agents function with evaluators controlling goals | Agents should function independently with evaluators selecting goals based on desirability. Different goals should be selected based on what's happening around them. | Failed as AI lose track of agent data and results in null reference exception error from several goals. | Fixed: Removed teammates from collection when they've died and re-add them upon re-spawning. |
| 26.1 Retest | Combined Test | Agents function with evaluators controlling goals | Agents should function independently with evaluators selecting goals based on desirability. Different goals should be selected based on what's happening around them. | Agents successfully function with evaluators picking goals as they play. Values need tweaking as expected. |  |

# Critical Evaluation

When it comes to implementation of the AI, I feel that it went well overall. I managed to implement most of everything I wanted albeit I had to scrap some as the gameplay just didn’t allow for the complex behaviours I wanted to model happen.

My design of the goal driven utility AI seems to be a good fit overall. It’s major strengths lie in it’s modularity of atomic and composite goals and it really allows itself to create complex behaviours from simple building blocks and the stack like nature of the sub goal processing means that a new goal can come in from an external source and it would simply perform the new steps given before popping back up to where it was previously and resuming. The brain goal that allows it to think through evaluations is a strong and safe system that doesn’t cause everything to become bogged down and adding new behaviours and evaluators is as simple as creating them by themselves and adding them to a list. There’s no major reconstruction or additional parts to the system that needlessly complicate it.

With that said, there are some issues with the system overall. The first and major issue is that of my evaluators. To prevent an AI from doing certain behaviours constant checks are performed which are intensive and run over a larger period of frames they could have halting effects. Also, there’s no use of dual utility reasoning and so an AI can tend to oscillate between multiple goals if it finds it’s needs all closely matched. A good solution to this problem would be the implementation of a bucket or double weighting system to evaluation checks (Graham, D. 2013)

Along with this, the AI doesn’t take any real complicated decisions with each other, I wanted to model more complex squad dynamics but ran out of time to implement a messaging system capable of allowing them to interface with one another and do behaviours I planned such as swap a flag with a healthier teammate or defending a friendly who’s bearing a flag. This coupled with some flawed equations for the evaluators has led to an ok AI in my personal preference.

I feel that the structure of the algorithm is sound but the design of the AI itself is weak and could do with further adjustments such as the ones I just stated above along with some other additions such as factoring in score to the evaluators so that a losing team goes on the offensive more as well as using character bias’s to allow individual traits to shine through. I also had a difficult time utilising curves in my equations to help adjust parameters due to my lack of math skills and so greater work could be placed there to really change up the way utilities are calculated and give the AI better decision making.

Overall, I feel it went well and I learnt a lot, but there’s a still a lot more that could be done to make it markedly better than it is right now.

# Bibliography

Buttice, C. (2019). *Finite State Machine: How It Has Affected Your Gaming For Over 40 Years*. [online] Techopedia.com. Available at: https://www.techopedia.com/finite-state-machine-how-it-has-affected-your-gaming-for-over-40-years/2/33996 [Accessed 1 Dec. 2019].

Buckland, M. (2004). *AI game programming by example*. 2nd ed. Plano, Tex.: Wordware, pp.44-56.

Champandard, A. and Dunstan, P. (2013). Game AI Pro: The Behaviour Tree Starter Kit. Boca Raton: CRC Press, pp.73-91.

Day, J. (2016). *Game AI: Finite State Machines*. [online] Game Development. Available at: https://www.gamedevelopment.blog/game-ai-finite-state-machines/ [Accessed 1 Dec. 2019].

Graham, D. (2013). *Game AI pro: An Introduction to Utility Theory*. Boca Raton: CRC Press, pp.113-126.

Merrill, B. (2013). *Game AI Pro: Building Utility Decision into Your Existing Behaviour Tree*. Boca Raton: CRC Press, pp.127-136.

Millington, I. and Funge, J. (2009). *Artificial Intelligence for Games*, 2nd ed. CRC Press, pp.125-155.

Rasmussen, J. (2016). *Are Behavior Trees a Thing of the Past?*. [online] Gamasutra.com. Available at: https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are\_Behavior\_Trees\_a\_Thing\_of\_the\_Past.php [Accessed 1 Dec. 2019].

Russell, S. and Norvig, P. (2009). *Artificial intelligence: A Modern Approach*. Reading: MA: Prentice Hall, pp.480-509.

Verma, E. (2016). *Finite State machine : history definition Model example | Engineer's Portal.* [online] Er.yuvayana.org. Available at: https://er.yuvayana.org/finite-state-machine-history-definition-model-example/ [Accessed 1 Dec. 2019].

# Appendices

H