DAC619 AE1 Report

Q12975371

James Johnson

2019

Contents

[Chosen Algorithm: Goal Driven AI 2](#_Toc26502869)

[Design 4](#_Toc26502870)

[Goals 5](#_Toc26502871)

[Evaluators 7](#_Toc26502872)

[Data Evaluators 8](#_Toc26502873)

[Additional Helpers 8](#_Toc26502874)

[Testing Plan 9](#_Toc26502875)

[Test Data 9](#_Toc26502876)

[Critical Evaluation 10](#_Toc26502877)

[Bibliography 11](#_Toc26502878)

[Appendices 12](#_Toc26502879)

# 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 reasons 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. Detailed below is a brief overview of the various goal’s behaviour and why some decisions were made in regard to behaviour.

#### 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.

#### TeamScore

Returns the current agent’s team score as a normalised value between 0-1. 0 being least desirable to 1 being most desirable.

#### 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

H

## Additional Helpers

H

#### Targeting System

H

#### Helpers

H

#### WorldManager

H

# Testing Plan

H

## Test Data

H

# Critical Evaluation

H

# 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