DAC619 AE1 Report

Q12975371

James Johnson

2019

Contents

[Chosen Algorithm: Goal Driven AI 2](#_Toc26501969)

[Design 4](#_Toc26501970)

[Goals 5](#_Toc26501971)

[Evaluators 6](#_Toc26501972)

[Bibliography 7](#_Toc26501973)

[Appendices 8](#_Toc26501974)

# 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

The following section will offer a brief overview of the designed goals and their functionality intent for the AI.

#### 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

H

#### Goal\_CollectFlag

H

#### Goal\_DropObject

H

#### Goal\_FindFlag

H

#### Goal\_GetItem

H

#### Goal\_Globals

H

#### Goal\_MoveToBase

H

#### Goal\_Search

H

#### Goal\_UseItem

H

## Evaluators

H

#### Evaluator\_AttackEnemy

H

#### Evaluator\_GetPowerup

H

#### Evaluator\_UsePowerup

H

#### Evaluator Search

H

#### Evaluator\_GetEnemyFlag

H

#### Evaluator\_RetrieveLostFlag

H

#### Evaluator\_ScoreEnemyFlag

H

#### Evaluator\_ScoreFriendlyFlag

H

#### Evaluator\_Heal

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