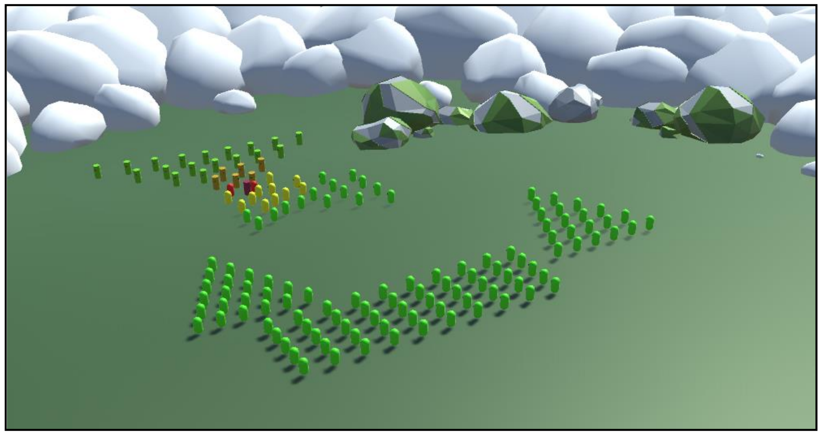
# Advanced Technologies Report: Can AI Sub-Commanders control Custom Formations for RTS Games

**Oscar Wilkinson**
**Department of Computer Science and Creative Technologies**
University of the West of England
Coldharbour Lane
Bristol, UK
Oscar2.Wilkinson@live.uwe.ac.uk
21007294



## Abstract

Can multiple AI sub-commanders control a custom formation system instead of a player and still be effective enough for a videogame or tool?

## Author Keywords

Unity; AI; Pathfinding; Formation

## Introduction

Real Time Strategy (RTS) games have varying methods of dictating how units or troops move and operate within the virtual environment. Most of the gameplay is controlled by player input, with troops being directly told where they should relocate to, or how they should attack an enemy structure or base.

This poses the possible skill or entertainment barrier within these styles of video games. If a player cannot micro-manage quick enough, then they can be at a major disadvantage compared to some players who can input actions much faster. It may be adequate to provide managers or systems that accomplish those inputs to make the skill floor lower and accessibility to the game/genre higher.

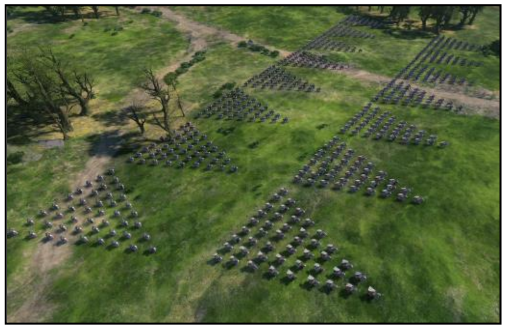


**Figure 01:** Total War Warhammer 3 example formation (Creative Assembly, 2022)

**Figure 02:** StarCraft 2 (Blizzard Entertainment, 2010)

To assist with this question, producing a system that removes a level of micro-management by using AI sub-commanders will help to see what benefits and drawbacks are apparent. The system chosen will be a custom formation system that attacks a base with opposing units in its path to evaluate its adaptability and range of utility.

## Background Research

Existing video games such as the Total War Warhammer games (Creative Assembly, 2016) have similar concepts for treating battle positioning and grouping of troops. There are a variety of unit types with their own strengths and weaknesses and can be formed in preset formations **(Fig 01)** or altered slightly to produce a more complex or bespoke layout, but cannot be saved for future playthroughs. The player dirrectly controls where certain groups of troops will position and when to attack, so all micro and macro is under the players direct influence.

In an extreme scenario in which micro-management is a large factor in who is succeding or failing is StarCraft 2 (Blizzard Entertainment, 2010) which does not necesserily involve as much battle structure or formations but still utilizes troops to fight and destroy **(Fig 02)**. The heavy reliance on micro inputs create gameplay scenarios in which the player rapidly controls multiple troops across an entire map, this level of Actions per Minute (APM) can be a skill barrier between levels of play, an example of 50APM may be a casual player, whereas cases of over 300APM is achieved by professionals.

A real life example in which battalions require specific formations to effectively fight opposing forces is the colonial war. Such as the use of musket formations that allowed for lines of defences and firing lines that were cycled between troops to keep a constant firing rate and strong defence (Keegan, 2014).

## Implementation

### *Tasks*

| No. | Description | Necessity |
|---|---|---|
| **1** | Generate formations that consist of multiple groups. | Required |
| **2** | Have groups be of varied sizes and shapes | Required |
| **3** | Have groups moves together towards a goal with obstacles to block the path. | Required |
| **4** | Have an interface that creates formations with editable values. | Should be implemented |
| **5** | Have enemy units to block their path. | Should be implemented |
| **6** | Save every created formation and be able to load them back in. | Should be implemented |
| **7** | Groups assist other groups. | Should be implemented |
| **8** | Randomize the map. | Could be implemented |

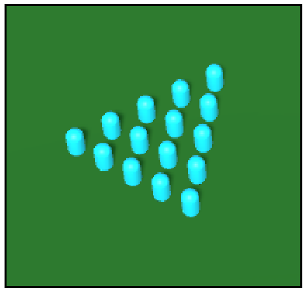


**Figure 03:** Triangle Formation

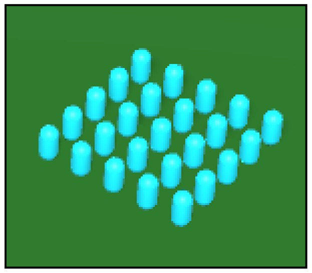


**Figure 04:** Box Formation

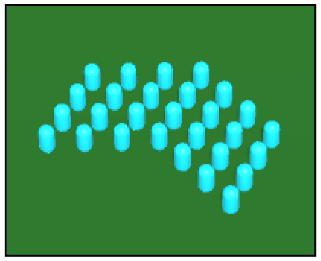


**Figure 05:** Arrow Formation

### *Group Movement.*

To create a formation, a proper structure must be made to control the flow of the system. In a full formation there are multiple groups, acting as AI sub-commanders, that each contain a grid of units that function as the individual troops in a battle, each with their own behaviour controlling them.

The units depend on the groups movements which function as a singular body. Formations are given a leader group which decides how the overall formation will move, the group closest to the goal at the very start is crowned as the leader. Every other group will base their movement from the leader group by being an offset distance away from them to keep the intended structure. Whenever a leader group dies, then furthest ahead group inherits that trait, maintaining a chain of hierarchy.

### *Map Influences with Pathfinding and Obstacles*

The leader will move to the intended destination by utilizing the Unity NavMeshAgent pathfinding, which can be influenced by NavMeshObstacles to dictate the path of movement the formation will take. Whenever a group within a formation cannot pass an obstacle, or is far behind its intended position, then all groups will halt their movement and wait for the group to catch up and reposition itself.

### *Formation Patterns.*

Groups control where their units are positioned based on their formation pattern and a collection of variables associated with the pattern type. The three patterns within this system are:

- Triangle Formations **(Fig 03)**, which create a triangular shape to pierce though groups.

- Box Formations **(Fig 04)**, which create a rectangle of size X and Y to create a solid block of units.

- Arrow Formations **(Fig 05)**, which create a chevron shape to pierce through groups.

Each formation type has:

- A noise value, adding a slight random shift in the position to provide more natural patterns, or have less predictable positions.

- An Nth shift, controlling how much each row is shifted from each other.

- An even shift, controlling how much each second row is shifted.

- A hollow toggle, emptying out the inner units.

- A supplier toggle, dictating whether the units are used to fill in gaps in other groups.

## *Formation Creator and Editor.*

Formations can be created within a scene that holds a user interface environment in which formation types can be dragged and dropped into a box to dictate its layout and how many groups there are **(Fig 06)**. Each group can be interacted with to start editing the values behind their structure. Once an adequate formation has been created, it can be previewed in a 3D example, positioning the units where they would be if used in a simulation to see if any alterations need to be made before wanting to use it.

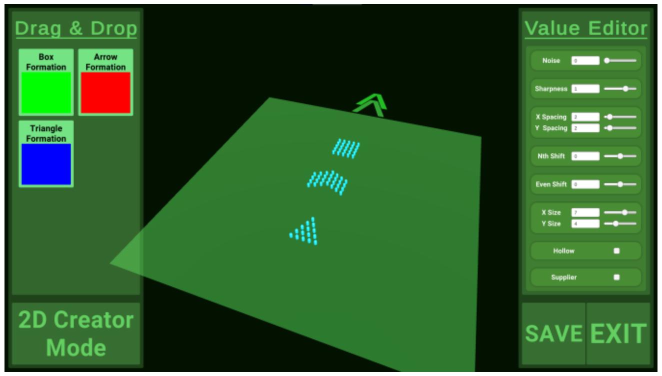


**Figure 06:** Formation creator and editor scene.

These formation plans can then be saved into a JSON file which allows for it to be used again after the application closes.

Once a formation has been created and saved, it is then open to two possibilities, the first is editing the structure again in the same manner as creating them, the second is starting a simulation based on the layout.

## *Unit Movement and Combat.*

Although each group is an independent body, every unit within the group move independently from it. By also using NavMeshAgents they move to where their position is within the group and have their own functionality. Units can be in one of a few states **(Shown in Fig 07)**:

- Stationary (Blue), holding their position in a formation that is not moving.
- Moving (Green), holding their position in a moving formation.
- Detected (Yellow), discovering a unit of the opposing type.
- Attacking (Red), damaging a unit of the opposing type.

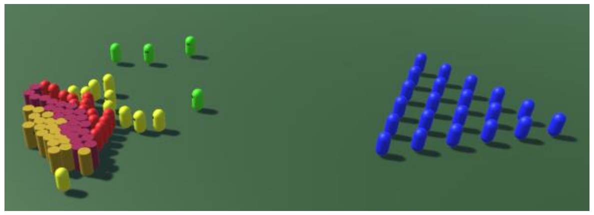


**Figure 07:** Units with coloured meshes.

Whenever a unit detects an enemy, it communicates to the group that an enemy is nearby, causing the entire formation to storm the opposing group and attack them, until either group has no more units to fight.

Due to their movements being based on NavMeshAgents, it allows for them to avoid other Agents and Obstacles to better improve their path and arrive to their destination faster. If a group is battling another group and feels overpowered by their number of units, then the group will request backup from another group that is not fighting and is closest to the position of the average enemy.

During a fight, if a group loses a unit, it will then reorder the formation of each unit to construct a stronger structure and request for a supplying unit to fill in the gap created at the back of the group to solidify the entire pattern. This supply request will grab the get the closest group that is enabled as a supplying group and inform it to change groups, also making the selected group to lose a unit. This process will repeat in a supply chain until it reaches the end of the formation. Doing this allows for minimal time between weak structures, allowing for more optimised fighting to simulate more strategic behaviours.

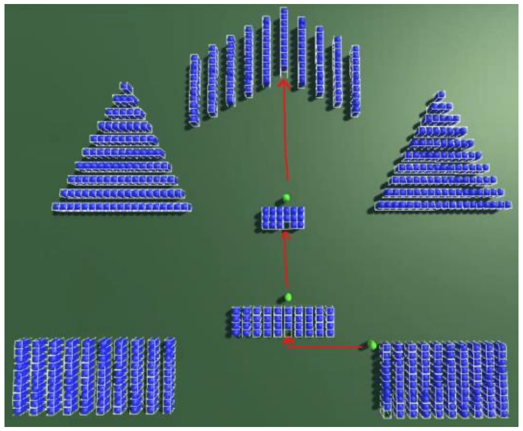


**Figure 08:** Unit supply chain.

***Map with Fog of War Quads.***

The map in which the simulation occurs has obstacles and enemies which are predetermined; however, a selection of maps can be created beforehand and be added to a selection of available maps for one to be selected randomly between each simulation.

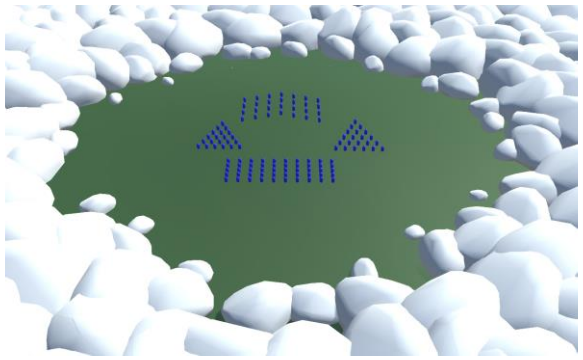


**Figure 09:** Fog of War around formation.

These maps also have a Fog of War (FoW) visualisation that shows what the units can see and creates a better setting for its applicable use cases. The FoW is controlled by a quadrant based spawning system. An area that starts from ($X_1$, $Y_1$) and ends at ($X_2$, $Y_2$) has a grid count of ($X_3$, $Y_3$) and instantiates multiple clouds per grid separation with a random spread, size, and rotation. Then every frame the FoW manager detects what grids all the players groups are located in and for every cloud within X grids distance will interpolate their size based on a min distance, a max distance and the distance from the closest group. This is utilized instead of having each cloud calculate their own distance and scale because it drastically decreases the processing load on the fog system as less distance calculation checks are accomplished.

## Outcomes

All tasks proposed were completed in the implementation process, including the tasks with low necessity concern, so the project outcome can be stated as a success, resulting in the following outcomes.

By separating out groups into their own sub-commanders they were easily given identical functionality with levels of permissions and hierarchy to their order, so any new feature can be implemented into one class with minimal external factors to consider.

The utilization of NavMeshAgents allowed for well optimised pathfinding and obstacle avoidance which accurately suited the desired usage and performance. Although, there are scenarios in which NavMeshAgents would consider the unit in front of them in the formation to be an obstacle and path around them halting their movement considerably, when they should disregard them as an interference.

More importantly, the custom made and altered formations function appropriately to the setting and gameplay loop which allowed for a unique simulative experience for RTS movement and fighting.

## Evaluation

As shown with its adaptability, AI sub-commanders can be suitable for applications within games to improve the accessibility and casual playability of RTS games.

The combined communication between leaders, to other groups, to individual units constructs an effective hierarchical structure which has high adaptability and opportunities for additional content.

There are a few possible avenues of future improvements and features that can assist this system further. Units can be given a variety of types for more advanced movement and behavioural architecture around fighting and countering against certain formation structures. Troops can be sent to scout out areas to see into the FoW and provide the player more information about the enemy’s whereabouts, proposing a more complicated strategic option of flanking enemies to gain more advantageous positions and attacking structures.

## References

[1] Technologies, U. (n.d.). Unity - Scripting API: NavMesh. [online] docs.unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/AI.NavMesh.html.
[Accessed 28 Apr. 2024]

[2] Technologies, U. (n.d.). Unity - Scripting API: NavMeshAgent. [online] docs.unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html.
[Accessed 28 Apr. 2024]

[3] Technologies, U. (n.d.). Unity - Scripting API: NavMeshObstacle. [online] docs.unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/AI.NavMeshObstacle.html.
[Accessed 28 Apr. 2024]

[4] Blizzard Entertainment (2010) StarCraft 2. [Video game].
[Accessed 28 Apr. 2024]

[5] Keegan, J. (2014). The face of battle : a study of Agincourt, Waterloo and the Somme. London: The Bodley Head.
[Accessed 28 Apr. 2024]

[6] Total War: WARHAMMER III. (n.d.). Total War: WARHAMMER III - Home. [online] Available at: https://warhammer3.totalwar.com/
[Accessed 28 Apr. 2024]