FUPS – Specification

Contents

[Introduction 2](#_Toc456707518)

[Main idea 2](#_Toc456707519)

[High-Level AI 2](#_Toc456707520)

[Low-Level AI 2](#_Toc456707521)

[Partial planning in FUPS 3](#_Toc456707522)

[Task assignment 3](#_Toc456707523)

# Introduction

FUPS (**F**ett Lis **U**rban **P**atrol **S**cript) is a script-package designed to enhance AI on a mission basis for ArmA 3.

It aims at improving group behavior rather than unit behavior – for example: FUPS will improve positioning and collaboration of AI groups but won’t improve specific unit positioning, i.e. taking cover.

# Main idea

There are two layers of AI:

* High-Level AI
* Low-Level AI

## High-Level AI

High-Level AI develops a plan for a set of AI groups. I. e. the High-Level AI will tell groups who to attack, which position to defend. It will develop plans using partial planning algorithms relying on Low-Level AI perceptions. There is only one High-Level AI.

## Low-Level AI

The Low-Level AI will perform by the BDI-Agent principle with the extend that it has its desires set by the High-Level AI. Each AI group controlled by FUPS is a Low-Level AI / a BDI-Agent. These agents will use their receptors in order to generate beliefs which will be then used by the High-Level AI. Once they’re told what to do they’ll generate intentions to fulfill their desires.

# Partial planning in FUPS

Partial planning in FUPS will only use partial planning mechanics to describe the planning problem and its implementation. The algorithm will then be implemented application-oriented in order to evade problems like unification of logic formulas.

For each situation there are two type of goals that must be reached:

* Every enemy must be defeated
* Every assigned area must be defended, i.e. there is no enemy and at least one friendly inside the area

The planning algorithm will use this goals indirectly by stating:

* Every enemy must be attacked
* Every assigned area must have an arbitrary number of units defending it

## Task assignment

A more formal definition: Be the (obviously finite) set of all enemies. Be the (obviously finite) set of all areas to defend. Be the (obviously finite) set of all units. We will identify elements in , and with unique numbers in , i.e. .

The planning problem to solve in which a finite sequence of sets with . Such a sequence is called a *task assignment*.

A task assignment is *sound* iff .

A task assignment is *complete* iff .

Each sequence member holds a set of units that are assigned to an element of or . More formal: means that all units in are assigned to attack the enemy iff or to defend the area iff .

So what we’re looking for are complete and sound task assignments.

With no further specification this problem is hilariously easy to solve. Just take an element of successively and assign it to a member in . As pseudo-code:

Input: Set of units U, sequence of empty sets G.  
Output: Modified sequence G.

FOR 0 <= i < |G|:  
 IF !empty(U) THEN:  
 G(i) := { pop(U) };  
 END;  
END;

RETURN G;

This algorithm will result in a sound task assignment and in a complete if there are enough elements in .

But there are more things to consider. Units can only defend an area if they’re present in it. And units will only attack an enemy if they have a chance of victory. Now partial planning comes into place.

## Task assigments and partial planning

A complete and sound task assignment can be takes as planning goal by inductive definition:

1. , note that whether a unit can fight an enemy is predetermined
2. , where is a variable not present in
3. , where is a variable not present in

The initial state is .

The set of available operations has only three elements: