Planning and Execution Control Architecture for Infantry Serious Gaming

Alexandre Menif, Christophe Guettier[1] and Tristan Cazenave[2]
[1]SAGEM, 27, Rue Leblanc, 75012 Paris, France
[2]LAMSADE, Université Paris-Dauphine, Paris, France
{alexandre.menif, christophe.guettier}@sagem.com
cazenave@lamsade.dauphine.fr

Abstract

Serious gaming is developing among all modern armies for teaching and training as well as for developing new concepts of engagement. To reach a realistic level of simulation, online planning techniques provide an expressive and constructive approach to define basic tactical activities. To achieve a mission goal, a virtual soldier must follow a short-term plan that can be quickly reprocessed in order to follow changes in the environment, orders or situation awareness. This paper presents a planning and execution control architecture for simulating the behaviour of virtual infantry soldier. The planning approach relies on the frequent generation of short plans using a Hierarchical Task Networks approach. Execution control handles synchronisations of soldiers and trigger replanning whenever action cannot be executed in simulation. Applied to two generic types of action, preliminary results show that response times match the level of reactivity needed for serious gaming.

Introduction

Automatic planning has always given major challenges in defence domains. Many problem models and search techniques have been considered at strategic, operative or tactical command levels. However, simulation of low level tactics and basic soldier behaviours refer in general to action scripting. In order to match the expected level of realism required by modern armies, this practice tends to become a tremendous and time-consuming engineering activity.

In video game, however, planning techniques have become more and more popular for a decade. The experience of planning in video games has started with FEAR, a first person shooter issued in 2005, which implements an algorithm called GOAP (Goal Oriented Action Planning) to provide an efficient coordinated behaviour to the enemies. Although GOAP was inspired on STRIPS, one of the oldest algorithm in actions planning, this experience has successfully illustrated the possibility to compute short linear plans for a few coordinated agents in real time (Orkin 2006).

Planning techniques provide an expressive way to model tactical behaviour, by developing a compositional approach to basic activities. Several dedicated planning domains can be developed according to the type of mission, environment or military action. Using search techniques, automatic plan generation can be used at different hierarchical levels to simulate both the command chain and soldier activities.

The paper exhibits a planning and execution control architecture that integrates mission management along the command-chain, automatic tactical sequence generation and execution control. The architecture makes use of Hierarchical Task Networks (HTN) representations, that facilitate command-chain modelling, automatic goal breakdown as well as sequential task synthesis. To match mission execution tempo and contingencies, the planning functionality can generate very short plans over small horizons. Once a plan is generated, each action is tentatively executed by the execution controler which interacts with the simulated environment. Execution control also handles synchronisations of soldiers and trigger replanning whenever action cannot be executed.

Applied to two generic types of action, preliminary results show that response times match the level of reactivity needed for serious gaming.

First section focuses on the relation between serious gaming and military requirements for infantry warfare at tactical level. The second section describes a software planning architecture intended to meet those requirements. The next section presents the modelling of a planning domain for infantry tactical behaviours and provide details about a new implementation of SHOP2. Finally the last parts are dedicated to the state of the art and conclusion.

Soldier Level Serious Gaming

Rather than using old fashion tactical simulation, serious gaming is a very promising approach to teach, train and develop new concepts of engagement. Serious gaming reuses video game design and offers immersive and interactive environment to armies end-users. To be effective in professional tasks, these environments necessitate realistic and intelligent behaviour for simulated soldiers.

Different requirements stress both agents behaviour as well as the architectural design of such serious gaming:

- Teaching: Basic infantry scenarii must be easy to program, and soldier behaviour must be realistic enough to stimulate the end-user. Basic action sequences can be analysable to give explanations during after action review.

- Training: Such training stresses coordination, orders and reporting capabilities of the end-user. The serious game must be able to handle a large set of soldiers, and to construct complex situation. Again, in spite of their complexity, user performances can be analysed during after action review.
- Concept development: the development of new operational concept or the integration of new systems need doctrinal evolution. On one hand, serious gaming can help understanding the impact of new tactics, operational techniques and procedures. On the other hand, it provides a strong support to evaluate new system performances and man-machine interactions.

Planning and Execution Control Architecture

Figure 1 gives an overview of the Planning and Execution Control Architecture. It features two different levels of planning functionalities. Mission management solves medium and long term planning from brigade down to the platoon level. Below this level, squads and soldier sequence of actions are generated by a dedicated HTN planner. This core component drives the quality of virtual soldier behaviour and is detailed in a dedicated section.

The planning architecture is meant to be integrated in a real time simulator, which means that it is not possible to spend much processing power in planning procedures that may alter frame rate, gameplay and graphical rendering. The architecture must scale up to several dozens of entities to be managed in parallel, potentially leading to many planning queries to be conducted simultaneously (see figure 2).

Both planners takes inputs from a situation awareness and threat assessment components that gathers and fuses data from the simulation environment. The execution control component processes the realisation of a sequence of actions in the simulator kernel. Whenever a basic sequence of action cannot be correctly executed in the simulation environment, a replanning event is triggered.

Mission manager, situation awareness, threat assessment and execution control modules are fundamental software components for the architecture design, however their detailed design are out of the scope of this paper.

Mission manager

The mission manager combines mission planning and scheduling as well as plan decomposition for lower units. Given initial conditions (on both friendly and enemy units), mission planning and scheduling (P&S) defines the course of actions to reach mission objectives. P&S takes in account terrain structure, unit capabilities and their coordination :

- Terrain representation involve axis of advances, observation points, covers and concealments.
- Capabilities refer to mobility, engagement, communication and observations.
- Coordination of actions is needed in time and space for self protection, or to reach an expected effect.

The problem solved by the mission manager P&S is to find, for each unit, a course of actions and movements (e.g.

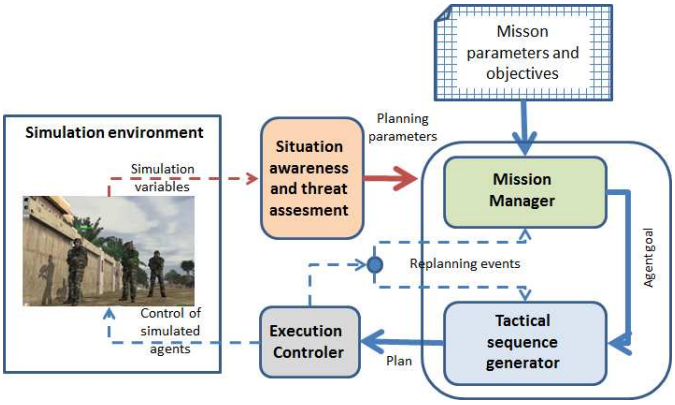

Figure 1: Global planning architecture for agent gaming

	squad	platoon	company	battalion
timeline	$<$ second	$<$ 5 mins	5 to 15 mins	$>$ hour
units	10	4	16	70

Figure 2: This table gives the timeliness with respect to the number of units to plan for

a plan) with an associated schedule from the initial position to the mission objective. The mission planning problem can be modelled and solved following a complete constraint programming approach (Guettier 2007).

The mission manager must also handle plan breakdown along the tactical command chain. Each hierarchical level defines an operational order (OPORD) using its own mission goal. The OPORD tasks / organises units of a lower echelon and allocates resources. This part is not automated yet, since many constructive parameters have to be taken in account simultaneously (organisational, logistics, assessment of enemy situation and associated course of action, complex coordination procedures...). However, it is possible to automatically structure the different OPORD by using the outcome of both mission planner and units organisation.

When major changes occur during mission execution, global replanning might be necessary. This has two impacts:

- A new problem instance is provided to the planner. Then, plan repair or local search can be used to find a plan that cope with the new situation.
- OPORD are updated, using so-called FRAGmentary Orders (FRAGO).

Execution Controller

The execution controller executes action provided by the sequence generator. In several ways it interacts with the simulation environment by controlling the virtual soldier. For a given action, it will set the virtual soldier mobility, orientation, and posture. Through the simulator, the controller can request a waypoint to reach, assess visibility of a line-of-sight, or find threatening objects within its field of view. To evaluate the success of an action, the execution controller uses the following logic:

- Preconditions are verified. Their scope are mobility (way points), previous action termination, observation results, or event occurring in the environment.
- Request to the simulator succeeds (for instance, the path finder has been able to find a waypoint).
- Time / space coordination can be met. These coordination can be either defined by the mission planner or by collaborative type of actions (for instance a mutual protection or a synchronised mobility action).

Whenever an action cannot be successfully achieved, a replanning event is triggered.

Situation awareness and threat assessment

Situation awareness is maintained by a Red Force Tracker (RFT) directly inspired from target acquisition and tracking systems (Sella & al. 2011). The RFT service tackles observation reports, target association, short term position estimate, long term enemy course of action prediction. Data fusion algorithms such as Kalman Filtering (KF), Interactive Multiple Models, or Multiple Hypothesis Tracking are integrated to associate observations, remove inconsistencies, and to manage an active list of tracks. Delayed reporting and tracking are also simulated such that two units does not necessarily have the same immediate awareness (note that in real life, this knowledge is actuated by issuing an OPORD through the chain of command).

Based on these outputs, whenever the enemy situation changes, each soldier or squad evaluates its own threat level in order to potentially recompute a new plan sequence.

Tactical sequence generator

Each hierarchical unit uses planning for its level and then assigns orders for its sub-level units. Units coordination is verified by planning at the upper level, so that each level can behave more autonomously within its own action area. It also results in time / space synchronisations, enforced by the execution controller.

In combat simulation, the tactical environment is constantly evolving, so that plans can be quickly invalidated. Replanning is necessary to comply with the up to date situation awareness. Relying on a planning domain where actions would be interleaved between hierarchical levels would be hard to manage. Indeed this would likely produce complex and expensive plans that would be compromised by the first unexpected event.

Instead, the proposed approach is closed to the Command Hierarchies concept (Pittman 2008). This results in a much more flexible way of planning: not only the global plan search complexity would be reduced, but an unexpected event would only affect a tiny sub-part of the overall plan and not trigger a re-planning for the whole operation. Another aspect is that low-levels units would probably be much more subjected to planning events than high-levels ones. For example, the detection of obstacles, traps or enemies would surely alter the way a fireteam had planned to conduct its mission, but not necessarily the global manoeuvre of a platoon.

```
; monitor problem definition
(defproblem problem monitor
  ; this line describe the environment as a set of predicates
  ((down soldier1) (detected soldier1 target sector))
  ; the task assign to the soldier
  ((monitor soldier1 sector)))

; a computed plan for the monitor problem
(:task !report soldier1 target)
(:task !stand-up soldier1)
(:task !use-weapon soldier1 target)

; patrol problem definition
(defproblem problem patrol
  ((sector front) (sector sector1) (unsafe sector1) (front fireteam soldier2)
   (back fireteam soldier1) (covering soldier1 front) (covering soldier2 sect
   (up soldier1) (up soldier2))
  ((patrol fireteam)))

; a computed plan for the patrol problem
(:task !cover soldier2 sector1)
(:task !cover soldier1 front)
(:task !find-cover-point soldier1 sector1)
(:task !go-to-cover-point soldier1 sector1)
(:task !pass soldier1 soldier2)
(:task !cover soldier1 sector1)
```

Figure 3: Planning request and plan answer for monitoring actions

Infantry domain modelling

It sounds crucial that the lower a unit is in the command chain, the simpler its behaviour should be defined. For example, a single soldier would probably be constantly planning to adapt his activity to a constantly changing environment, so his behaviour should be extremely cheap to plan. Therefore, our requirement for a planning domain relies on two aspects: actions at one hierarchical level should only focus on its level and the synchronization of the direct sub level, and at the bottom of the hierarchy, planning should be extremely simple. With those rules in mind, we have started to design a simple planning domain that could match our expectation, using SHOP formalism (Nau & al. 2003).

The planning domain (see appendix) describes a set of very simple actions for monitoring and patrolling tasks, carried out by one soldier or a two-soldiers fireteam. The ”monitor” action illustrates how planning would interact with events from the simulator. Here the simulator would request planning for the ”monitor” task each time an event modifies the situation (in this case, simply either there is an enemy which is detected or not). Then, the virtual soldier has a few possible behaviours. He may or may not engage the target, according to its ability to do it or if it is allowed to do it. The ”patrol” action is collaborative (it involves two soldiers from a fireteam) and handles the coordination between the two soldiers (so one soldier only moves if it is covered by the other), and will assign very simple tasks to each soldier agent. Figure 3 representes two typical queries for both examples (planning requests for actions ”monitor” and ”patrol”) as they would be issued by the simulated environment, and the replies from the planning system: two plans as sequences of tasks.

Search Algorithm

In order to constitute a benchmark of popular planning algorithms, a first implementation of SHOP2 is under development. The C++ programming language is used as it is

the one employed in the targeted simulator product. At its current state, the planner implements the main core functionalities of the original one from Nau (Nau & al. 2003), alongside with a parser component that is able to read planning domains and problems written with a subset of the formalism described for SHOP.

On different situations, all the previously detailed examples (figures 3) are computed in less than a millisecond with our current implementation of SHOP2. Even if actual planning domains would probably be more sophisticated, this is the target time we will have to achieve for such basic level hierarchical units.

Conclusion

This paper proposed a way to apply planning techniques from the video games experience on basic military units behaviour to professional simulation tools. The main objective is to demonstrate that tactical behaviours for low level entities of military hierarchy (individual soldier, fireteam, squad...) could be encoded and executed with low computational power through two main considerations.

On one hand, hierarchical planners seem adapted to model complex behaviour for coordinated units. They affords a segregation of the chain of command from the local virtual agent behaviour, alongside with a convenient expression of action sequences. On the other hand, a reactive architecture, aware of any triggered events from the simulated environment, can restrict the time horizon for planning request considerably, and thus reduces the need for complicated planning processes.

The dedicated planner fits easily with other components (long term mission planning, situation awareness and execution control). In terms of software engineering, the approach provides a strong gain compared to scripting methods.

References

Guettier C. Solving Planning and Scheduling Problems in Network based Operations. Proceedings of CP'07, USA. 2007.

Sella, G.; Cherrier, O.; Guettier, C. & Yelloz, J. Development and Experimentation of Collaborative Red Force Tracking in Service Oriented Architecture for Tactical Networking Systems Proceedings of MILCOM'11, USA. 2011

Orkin, J.. Three states and a plan: the AI of FEAR. In Game Developers Conference. 2006.

Nau, D. S.; Au, T. C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D. & Yaman, F. SHOP2: An HTN planning system. J. Artif. Intell. Res. (JAIR), 20, 379-404. 2003.

Pittman, D. Command Hierarchies Using Goal-Oriented Action Planning, AI Game Wisdom 4, Charles River Media (2008), pages 383 to 391. 2008.

Planning domain

```
(defdomain monitor (
  (:operator (!use-weapon ?soldier ?target) ((up ?soldier)) () ())

  (:operator (!watch ?soldier ?area) ((up ?soldier)) () ())

  (:operator (!report ?soldier ?target) () () ((can-engage ?soldier ?target)))

  (:operator (!bend-down ?soldier)
             ((up ?soldier))
             ((down ?soldier)))

  (:operator (!stand-up ?soldier)
             ((down ?soldier))
             ((down ?soldier))
             ((up ?soldier)))

  (:operator (!cover ?soldier1 ?sector1)
             (and (sector ?sector2) (covering ?soldier1 ?sector2))
             ((covering ?soldier1 ?sector2))
             ((covering ?soldier1 ?sector1)))

  (:operator (!follow ?soldier1 ?soldier2) () () ())

  (:operator (!go-to-waypoint ?soldier1) ((have-waypoint ?soldier1)) () ())

  (:operator (!find-cover-point ?soldier1 ?sector1)
             ()
             ()
             ((have-cover-point ?soldier1 ?sector1)))

  (:operator (!go-to-cover-point ?soldier1 ?sector1)
             ((have-cover-point ?soldier1 ?sector1))
             ()
             ((at-cover-point ?soldier1 ?sector1)))

  (:operator (!pass ?soldier1 ?soldier2)
             (and (front ?fireteam ?soldier2) (back ?fireteam ?soldier1))
             ((front ?fireteam ?soldier2) (back ?fireteam ?soldier1))
             ((front ?fireteam ?soldier1) (back ?fireteam ?soldier2)))

  (:method (use-weapon ?soldier ?target)
           ; soldier cannot use weapon if down
           ((down ?soldier))
           ((!stand-up ?soldier) (!use-weapon ?soldier ?target))
           ; soldier already up
           ()
           ((!use-weapon ?soldier ?target)))

  (:method (watch ?soldier ?area)
           ; soldier cannot watch if down
           ((down ?soldier))
           ((!stand-up ?soldier) (!watch ?soldier ?area))
           ; soldier already up
           ()
           ((!watch ?soldier ?area)))

  (:method (engage ?soldier ?target)
           ; soldier must protect itself
           ((under-fire ?soldier))
           ((!bend-down ?soldier))
           ; target is neither hostile nor can be engaged
           ; according to rule of engagement
           ((not (hostile ?target)) (not (can-engage ?soldier ?target)))
           ((!report ?soldier ?target) (use-weapon ?soldier ?target))
           ; else engage target
           ()
           ((use-weapon ?soldier ?target)))

  (:method (monitor ?soldier ?area)
           ; a target is detected while monitoring
           (and (detected ?soldier ?target ?area))
           ((engage ?soldier ?target))
           ; else, keep watching
           ()
           ((watch ?soldier ?area)))

  (:method (patrol ?fireteam)
           ; if a target is detected in a given sector, engage it
           ((detected ?target ?sector1) (covering ?soldier1 ?sector1))
           ((engage ?soldier1 ?target))
           ; if there is an unsafe sector, use parrot-like move
           ((sector ?sector1) (unsafe ?sector1)
            (front ?fireteam ?soldier1) (back ?fireteam ?soldier2))
           ((!cover ?soldier1 ?sector1) (!cover ?soldier2 front)
            (!find-cover-point ?soldier2 ?sector1)
            (!go-to-cover-point ?soldier2 ?sector1)
            (!pass ?soldier2 ?soldier1) (!cover ?soldier2 ?sector1))
           ; else progress normally
           ((have-waypoint ?soldier1) (front ?fireteam ?soldier1)
            (back ?fireteam ?soldier2))
           ((!cover ?soldier1 front) (!cover ?soldier2 far)
            (!follow soldier2 soldier1) (!go-to-waypoint ?soldier1)))))
```