Hierarchical Plan-Space Planning for Multi-Unit Maneuvers

William van der Sterren
William.van.der.Sterren@cgf-ai.com

3rd International Planning in Games Workshop, ICAPS, Rome, Italy, June 10, 2013
Contents

- Speaker background
- Problem: Combat Maneuvers in Games/Simulations
- My Failures and Rejected Ideas

- Hierarchical Plan-Space Planning
  - Plan, Tasks, Planner Loop, Methods, Plan-Space
  - Efficient Planning
- Implementation, Performance and Stats

- Conclusions & Observations

Slides available as www.cgf-ai.com/docs/pg2013.pdf
William van der Sterren

- contributed AI to Killzone (PlayStation 2, 2004)
- spoken at Game Developer Conference 2001, 2005
  - procedural tactics, tactical pathfinding, squad behavior, terrain analysis
- contributed to AI Wisdom / Game Programming Gems books
- AIIDE program committee member since 2008

- experimenting with planners since 2007, resulting in the PlannedAssault "testbed" and mission generator

Disclaimer:
All game footage shown is from the ArmA series of games by Bohemia Interactive Studios. This work is not affiliated or endorsed in any way by Bohemia Interactive Studios.
Problem: Planning Combat Maneuvers

"Our plan:
We'll clear objective Z, with A, B, C, D and E platoons forming up and launching a two pronged simultaneous attack. Afterward, we'll regroup at objective Z.
B platoon will transport A and C to their form up areas. A and C platoons will attack across the northern bridge. D and E platoons will attack across the southern bridge.
Fire support is provided by batteries H and J and gunships W. Batteries H and J will fire smoke screens to cover the bridge crossings. W flight will be on call."
ArmA series (by Bohemia Interactive Studios, Czech Republic):
- FPS/combat sim, capable of large maneuvers (100+ soldiers & vehicles)
- shares technology with the VBS-2 simulator (by Bohemia Simulations)
- basic individual AI, and (AI) groups take orders from player / mission script
- simple mission scripts (text files) can be created with built-in editor

PlannedAssault uses a planner to generate large/complex ArmA missions
Problem Characteristics

- Complete plan required
- Combinatorics
- Plan quality matters
- Action ordering and synchronization
- Non-trivial sub-problems

Outside scope

- Plan repair / replanning
- Min-max (we lack exact intel on the opponent's composition, positions and intentions)
Problem: Complete Plan Required

Computer Scenario generation / Automated order generation

- All (playable) / computer units need to have their orders before hand
- Orders need to work out in a 1:1 FPS simulation (no abstraction)
- Brief human players, brief back commander, or explain planning failure
Problem: Combinatorics

State-space:
- 10 units, with each on average 5 actions each per plan
- 22 unit actions/operators
- 45000+ action locations (of 25x25m in a 8x6km battlefield)

Computation time
- "budget" to evaluate 1000 to 10000 plans

22^{10^5}

10^4
Problem: Plan Quality Matters

Same plan, different quality!

- Armor (E) prefers not to attack through towns or across bridges

The good news: We don't need a perfect plan. A good plan is good enough!
Problem: Action Ordering & Synchronization

A maneuver requires some ordered, synchronized actions:

- Synchronized start of attack
  - mutual support, mass firepower, exploit surprise

- Ordered "time separated" actions: fire support
  - Don't want aircraft to fly into mortar rounds near the objective

- Delayed actions
  - Don't want ambulance or anti-air units to arrive in a form-up area before the combat units do
Problem: Non-Trivial Sub-Problems

Maneuver planning also requires

- (Tactical) Path-finding (clearance, flanking, concealment, turn radii)
- Transportation decisions
- Resource allocation

To be integrated/interfaced with planning!

Resource Allocation problem:
given expected vehicle (red) and infantry (orange) approaches, how to best allocate defenders to sectors, given their capabilities in engaging vehicles and infantry?

Tactical path-finding:
A helicopter section flying an ingress path that uses terrain masking.
# My Failures and Rejected Ideas

Approach, as considered for a multi-unit planning problem

<table>
<thead>
<tr>
<th>Issues</th>
<th>&quot;Game Industry&quot; techniques</th>
<th>&quot;Academic&quot; planning techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(MC) Brute force</td>
<td>STRIPS</td>
</tr>
<tr>
<td></td>
<td>GOAP / A*</td>
<td>Behaviour Trees</td>
</tr>
<tr>
<td>Incomplete Plan</td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>Plan may contain actions which cannot be explained</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Picks first/any plan instead of &quot;best&quot;</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Synchronizes actions</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Cannot explore multiple alternatives</td>
<td>X</td>
<td>x</td>
</tr>
<tr>
<td>Drowns in combinations</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Cannot bring all facts into planner</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Didn't discover this approach until this workshop. See “Solving Planning and Scheduling Problems in Network based Operations”, Christophe Guettier
Contents

- Speaker background
- Problem: Combat Maneuvers in Games/Simulations
- My Failures and Rejected Ideas

- Hierarchical Plan-Space Planning
  - Plan, Tasks, Planner Loop, Methods, Plan-Space
  - Efficient Planning
- Implementation, Performance and Stats

- Conclusions & Observations
Hierarchical Plan-Space Planning

Concepts

- Search in Plan-Space, not in State-Space
- Plan = Hierarchy of tasks
- Tasks describe activity, ordering, inputs & outputs, and costs
- Methods create more detailed plans by decomposing one task
- A* main loop, refining least-cost plan first
- Heuristics to estimate task and plan costs
- Advisors to generate, filter, sort and select best options for tasks
State-Space planning vs ...

- Forward (shown) or backward search in state-space
Plan-Space search doesn't consider unit actions until it is very clear what to do. Higher level (abstract) choices typically have few options.
Plan

Bonus! In addition to containing all unit actions and its scheduling, the plan includes the 'meta' information enabling explanation of the plan.
## Tasks

<table>
<thead>
<tr>
<th>scope</th>
<th>task examples</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>mission</td>
<td>mission</td>
<td>task force</td>
</tr>
<tr>
<td>objective</td>
<td>clear, occupy, defend</td>
<td></td>
</tr>
<tr>
<td>team</td>
<td>move, form up, attack, air land, defend, counter-attack, para drop, wait, casualty care</td>
<td>company</td>
</tr>
<tr>
<td>tactic</td>
<td>formation ground attack, planned fire support, smoke screen</td>
<td></td>
</tr>
<tr>
<td>units</td>
<td>transported move, defend sector</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>defend, guard, attack, hide, move, wait, air ingress, air egress, mount, dismount, load, unload, ride, para jump, fire artillery mission, close-air-support, casualty care</td>
<td>platoon / squad</td>
</tr>
</tbody>
</table>

- Scope matches the chain of command, unit size and decision types.
- Tasks are decomposed into sub-tasks of lower scope.
Task types, states and relations

- Parent task is decomposed into child tasks
- Tasks may have predecessors and successors
- Tasks can be compound or primitive (unit tasks or operators)
- Tasks have start times and duration (we'll get to that later)
Task Examples

```plaintext
# A ParachuteTask is given:
# - a start state, indicating the unit's initial loaded state
# - a target state, indicating the unit's target state
# - a description of the drop zone
# - the passengers which are to be parachuted

class ParachuteTask < UnitTask
  is_primitive
  has scope: unit
  has input: start_state, type: unit
  has input: target_state, type: unit
  has input: drop, type: string
  has input: passengers, type: units

def compute_expected_costs(context):
  end
end

# A RideTransportTask is given:
# - a start state, indicating the unit's mounted initial state (position)
# - an end state, indicating the unit's mounted end state (position)

class RideTransportTask < UnitTask
  is Primitive
  has scope: unit
  has input: transporter, type: unit
  has input: start_state, type: unit
  has input: end_state, type: unit

def compute_expected_costs(context):
  end
end

# An AttackAfterFormUpTeamTask is given:
# - an objective
# - a start state (as unit states preceding the form-up & attack)
# - an avenue of approach to use for this attack
# - the objective area indicating what terrain to attack & from where
# - the assembly area for pre-attack form-up
# - per unit assembly positions in the form-up area
# - an end state, as unit states after the attack in the objective area

class AttackAfterFormUpTeamTask < Task
  has scope: team
  has input: start_state, type: units
  has input: objective, type: objective
  has input: avenue of approach, type: avenue_of_approach
  has input: objective_area, type: area
  has input: assembly_area, type: area
  has input: assembled_state, type: units
  has input: end_state, type: units

def compute_expected_costs(context):
  end
end
```
Task Inputs and outputs

- To parametrize tasks (inputs)
- To share results (outputs connected to other inputs/outputs)
- To acts as variables that need to be grounded
- To take in hints

- Typical Input/output types:
  - unit, objective, area, avenue of approach, landing zone, path, ...
  - "Unit" ranges from a 2 man sniper team to a 13 man squad, from a single vehicle to a platoon of 5 tanks, 4 helicopters, ...
Task Input/Output Example

PDDL style indirect linking of output and input:
location(lz_A);
location(lz_B);
near(LZ, lz_A);
near(LZ, lz_B);
unload(C, A, lz_A);
unload(C, B, lz_B);
attack_from(A, lz_A);
attack_from(B, lz_B);
not_equal(lz_A, lz_B);
...

No!

- Planner has to resolve location from $43000^2$ possible positions.
- Notation doesn't handle vectors (of objects) easily (PDDL 3.1 might)
- Instead...

TeamAirLand - CAB
inputs:
objective: Z
landing_zone: [...] 
start state: [A in C, B in C, C@1(1,8)]

outputs:
end state: [A@?, B@? C@?]

TeamAttack - AB
inputs:
objective: Z
start state: [A@?, B@?]

outputs:
end state: [A@?, B@?]
Plan-Space

refine
ClearObjective
into two alternative plans

refine lowest-cost plan first:
two alternative plans to move units A, B and C

alternatives
refinement

Plan space stats:
mean 74, median 18, max 1460 plans.
See 'Performance'.
Planner main-loop (A*)

loop
  current = get most promising plan from open list
  break if current.complete? or current.null?
  pick t = current.task_to_detail
  for every method m that applies to task t
    alternatives = m.generate(current, t)
    for every a in alternatives
      plan = clone current
      // refine using method m and alternative a
      m.refine(plan, t, a)
      compute plan's cost
      add plan to open list

A* here is best-first, expanding to all neighboring plans.
Planner methods

Planner methods detail a plan by picking a single abstract task and decomposing it into sub-tasks and/or setting its output values. Planner methods add an explanation to the task, for inclusion in the plan's briefing.

Four elements

- $\text{apply}(t)$ test to indicate tasks it knows to decompose
- $\text{generate}(t)$ function to generating a list of alternatives to implement task $t$
- $\text{apply}(t, a)$ function to decompose task $t$ according to alternative $a$
- $\text{troubleShoot}(t)$ function to explain why it failed to decompose task $t$
### Planner Methods examples

<table>
<thead>
<tr>
<th>scope</th>
<th>planner method responsibility</th>
<th>task examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>mission</td>
<td>arrange objectives, allocate units to objectives</td>
<td>mission</td>
</tr>
<tr>
<td>objective</td>
<td>define team activities, organize combat and support units in teams</td>
<td>clear, occupy, defend</td>
</tr>
<tr>
<td>team</td>
<td>execute tasks as a team, distributing the work according to roles</td>
<td>move, form up, attack, air land, defend, counter-attack, para drop</td>
</tr>
<tr>
<td>tactics</td>
<td>synchronize tactical moves between multiple units</td>
<td>formation ground attack, planned fire support, smoke screen</td>
</tr>
<tr>
<td>units</td>
<td>arrange co-operation between complementary units</td>
<td>transported move</td>
</tr>
<tr>
<td>unit</td>
<td>define end-state</td>
<td>defend sector, guard, close-air-support</td>
</tr>
</tbody>
</table>

Planner methods detail a plan by taking a single task from the plan which is decomposing into sub-tasks or given output values. Designer control / doctrine via catalog of planner methods.
Task Decomposition Example

Refine the A and C TeamFormationAttack task, using the A and B UnitAttack end states to determine TeamFormationAttack's end state.
Efficient Planning

- Identify most promising plan to expand
- Decompose high-level tasks first
- Support Middle-out planning
- Control how (many) alternatives are generated (advisors)
- Short-cuts, Heuristics, Tricks

```
loop
    current = get most promising plan from open list
    break if current.complete? or current.null?
    pick t = current.task_to_detail
    for every method m that applies to task t
        alternatives = m.generate(current, t)
        for every a in alternatives
            plan = clone current
            // refine using method m and alternative a
            m.refine(plan, t, a)
            compute plan's cost
            add plan to open list
```
Plan Costs, to Identify Most Promising Plan

Combat maneuvers: plan duration is a good 'costs' indicator

- Shorter movements to attack typically have less risk
- The sooner defenders are in position, the better

So, the most promising plan is

- the plan with shortest duration on the open list

Plan duration =

- latest task end-time - earliest task start-time, with tasks partially ordered according to task precedence
Task duration

Task duration =

- Task's duration, if task primitive and all task inputs available
- Max (sub-tasks' end-time) - Min (sub-tasks' start-time), if task decomposed into sub-tasks
- Lower-bound estimate, otherwise

- Every task (type) is able to provide a lower-bound estimate of itself (heuristic!)
Estimating duration of a compound task

Cost (under)estimate for: TransportedMove(A,B,C) to X:
- Path duration(C, A) +
- Path duration(A, B) +
- Path duration(B, X) +
- 2 \times load time +
- 1 \times unload time,
while assuming transport C moves at maximum speed.
Decompose 'higher scope' tasks first

Benefits of decomposing 'higher scope' tasks first
- Higher level decisions have more impact on plan duration
- Plan failure typically discovered sooner
- Combinatoric problems typically occurs at units level (e.g. TransportedMove)
Middle-out planning

Expand the plan from a critical decision backwards and forward

- Military practice when planning, for instance, air landings
- Enabled by linking output and inputs
Control how (many) alternatives are generated

Unit position (45000 values) being a variable for most actions
- avoiding a theorem prover to ground unbound variables

Instead, handle alternative generation explicitly
- In planner methods' "generate" function

<table>
<thead>
<tr>
<th>Alternative selection</th>
<th>Where</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>generate()</td>
<td>Don't allow a transporter to pick up more than two passenger units</td>
</tr>
<tr>
<td>best ( n ) based on utility / quality</td>
<td>In advisors invoked by generate()</td>
<td>Advisors: attack, air land, defend, counter-attacks, battle positions, ...</td>
</tr>
</tbody>
</table>

- Good news: customer requests typically involve additional constraints, so fewer options to consider (but more logic to write).
**Advisor Example: AttackAide**

A three team joint attack with main and flanking elements

1. Generate FUPs for each team

2. Reject FUP combinations when paths to FUP cross

3. Generate LMM, MMR and LMR attack variants from FUPs into objective

4. Evaluate attack variants and pick highest quality few

- Reducing 3000+ FUP and flank combinations to the best few, possibly none.
Advisor: Picking the Form-Up Area (1 or 0 results)

- Influence mapping to select form-up positions.
- Either we find a form-up area or we don't.
- Something to keep outside the planner.
Short-cuts, Heuristics, Hacks

Read as much into player input as possible
- fewer choices for the planner, more control for player

Revert to sub-optimal solutions if optimal solutions take too long
- Defensive sector allocation blew up to 16.4M combinations from 0.19M combinations, when going to 10 units from 8
- Switched to a simulated annealing approach with 1000 iterations

Duration/Cost estimation
- Don't let the duration of "irrelevant" tasks hide the differentiating duration of key tasks. Ignore, for example:
  - Aircraft returning to base
  - Defensive Counter-attack duration
Implementation

PlannedAssault.com is a web application based on a planner

- Up since April 2009
- First in the Cloud, then on local 2.6GHz E8400 w 8GB, x64 Linux
- Scaled from 8 units, 4x3km terrain to 10 units 8x6km terrain while keeping job duration similar

Planner also integrated with commercial VBS-2 instructor support tool

Software

- Written in Ruby, running single-process on a JVM (Jruby)
- Code size: 27KLOC, with only 7KLOC for planner, tasks and methods.

"Combat proven": 72000+ plans, 18000+ missions
Performance (100 missions = 200 plans)

- Performance is dominated by 'advisors' performing terrain analysis, tactical path-finding, evaluating avenues of attack, **not** by the planning algorithm.

- Duration excludes game specific back-end translating plan to game input.
Plan-space doesn't grow much beyond the #steps (left)

More units doesn't lead to dramatically more planner steps (right), with 75% of the 10-units plans found in 80 steps or less.
Contents

- Speaker background
- Problem: Combat Maneuvers in Games/Simulations
- My Failures and Rejected Ideas

- Hierarchical Plan-Space Planning
  - Plan, Tasks, Planner Loop, Methods, Plan-Space
  - Efficient Planning

- Implementation, Performance and Stats

- Conclusions & Observations
Conclusions

Planning multi-unit maneuvers is feasible
- Uses "unorthodox" plan-space planning tailored to the domain
  - Hierarchy, doctrine, convention to use duration as costs
- Uses explicit control of plan alternatives being considered
- A* again a powerful and versatile tool
- Planning is a small part of the overall solution (code, performance)

Future work
- Speeding up advisors
- Planning from an in-battle situation (fewer choices, more analysis)
- Offering more/easier designer control (no separate "PDDL" file)
Observations (from a game AI perspective)

There is a place for planners in games
- When behaviors become too rich/long-term for Behavior Trees...
  - Multi-unit, cause & effect in the world, story/mission generation
  - Cloud computing may help here...

Games won't interface with an existing planner
- Either the planner is (re)implemented in the game, or no planner
- Planner small compared to game (state) logic and heuristics
- Subsets of planning problem are best solved outside planner

I don't see much use for planners which don't natively handle cost/utility