

Disclaimer: I use these notes as a guide rather than a comprehensive coverage of the topic. They are neither a substitute for attending the lectures nor for reading the assigned material.

Plans are worthless, but planning is everything.

There is a very great distinction because when you are planning for an emergency you must start with this one thing: **the very definition of "emergency" is that it is unexpected, therefore it is not going to happen the way you are planning.**



Dwight D. Eisenhower

- <https://www.gdcvault.com/play/1024912/Beyond-Killzone-Creating-New-AI>

Announcements

- Exam topics
 - World representation: Grid, Mesh, etc.
 - Navigation: Local (Steering) & Global (Search)
 - Decision making: Dtree, FSM, Btree, Rules, A* & HTN Planning
- Questions:
 - Compare/contrast approaches; pros & cons
 - Understand when and why it is appropriate to use different techniques
- **Question:** Which of the following are true about X and Y? (Circle all that apply)
 - X executes faster than Y
 - Y can do everything X can do
 - X & Y both rely on Z
 - There is nothing X does better than Y
- **Question:** How can the use of X reduce computation time in Y?

N-1: Production/rule systems

- Based on the current game state, activate a set of rules, pick from those based on some heuristic (e.g. best matches conditions)
- Pros:
 - Don't need to specify response to every contingency
 - Can respond to novel conditions
- Cons:
 - Hard to author robust rule systems
 - Emergence vs. over-engineering
 - Hard to debug
- Use in Games Industry, e.g. Environment-sensitive dialog generation (dynamic dialog gen)
 - Overwatch: https://www.youtube.com/watch?v=yqIKRL_5f6o&t=13s&spfreload=10
 - 2Bots1Wrench (GDC2012): <https://www.youtube.com/watch?v=j4elu6LxdZg>
 - L4D2: <https://www.youtube.com/watch?v=T5-2EnX5-K0>
- See also:
 - Inform 7 rbs for interactive fiction
 - Eg rules chart: <https://emshort.files.wordpress.com/2009/01/rules-chart4.pdf>
 - Online Colossal Cave Adventure: <https://www.amc.com/shows/halt-and-catch-fire/exclusives/colossal-cave-adventure>

Rule Matching

Query

```
{ who: nick, concept: onHit, curMap:circus, health: 0.66, nearAllies: 2, hitBy: zombieclown }
```

PASS Rule 1: { who = nick, concept = onHit } → *"ouch!"*

FAIL Rule 2: { who = nick, concept = onReload } → *"changing clips!"*

FAIL Rule 3: { who = nick, concept = onHit, health < 0.3 } → *"aaargh I'm dying!"*

PASS Rule 4: { who = nick, concept = onHit, nearAllies > 1 } → *"ow help!"*

PASS Rule 5: { who = nick, concept = onHit, curMap = circus } → *"This circus sucks!"*

PASS Rule 6: { who = nick, concept = onHit, hitBy = zombieclown } → *"Stupid clown!"*

PASS Rule 7: { who = nick, concept = onHit, hitBy = zombieclown, curMap = circus }
→ *"I hate circus clowns!"*



Turn the Environment into Facts



Tag = "soda_can"



Tag = "barrel"



Tag = "bird"

Turn the Environment into Facts



Tag = "soda_can"



Tag = "barrel"



Tag = "bird"



Agent Knowledge Base

Current Decision Making Problems

- **Shallowness & Realism**
 - All of these techniques have the agent take **next** “best” move, but **don’t look further** into the future than the next state. Agents ought to be **motivated by goals**
- **Adaptability**
 - Each technique is more general/adaptive than the last (Rule Systems > Behavior Trees > FSMs > Decision Tree). But can still **struggle to perform well in unanticipated** situations.
- **Heavy Design Burden**
 - All three of these techniques require a heavy authoring burden on designers (FSMs: states and transitions, B trees: nodes and structure, Rules: all conditions and each individual rule)

Decision Making: Planning

2019-10-16

With extra thanks to Dana Nau, Hector Munoz-Avila, and Mark Riedl

Decision Making

- Classic AI:
 - making the optimal choice of action (given what is known or is knowable at the time) that maximizes the chance of achieving a goal or receiving a reward (or minimizes penalty/cost)
- Game AI:
 - choosing the right goal/behavior/animation to support the experience
- Today: One of closest overlaps

What is Planning?



- Finding a **sequence of actions** to achieve a **goal**
 - Problems requiring **deliberate** thought ahead of time and a sequence of actions
- Basic planning comes down to **search**:
“behavior planning using A^* ”
 - Given: state of the world, a goal, and models of actions
 - Find: sequence of actions (a plan) that achieves the goal
- Need to find appropriate heuristic

Some Examples

Which of the following problems can be modeled as AI planning problems?

- **Route search:** Find a route from GT to New Orleans
- **Project management:** Construct a project plan for organizing an event (e.g., Music Midtown)
- **Military operations:** Develop an air campaign
- **Information gathering:** Find and reserve an airline ticket to travel from Hartsfield to Miami
- **Game playing:** plan the behavior of a computer controlled player
- **Resource control:** Plan the stops of several of elevators in a skyscraper building

Answer: ALL!

Classes of General-Purpose Planners

General purpose planners can be classified according to the space where the search is performed:

- Action/Behavior Planning with A* {
- state
 - plan
 - Hierarchical Tasks
 - Disjunctive plans
 - SAT
- We are going to discuss these forms

“A planning system tells the A.I. what its goals and actions are, and lets the A.I. decide how to sequence actions to satisfy goals. [...] Planning is a formalized process of searching for sequence of actions to satisfy a goal” – Orkin

ACTION/BEHAVIOR PLANNING WITH A*

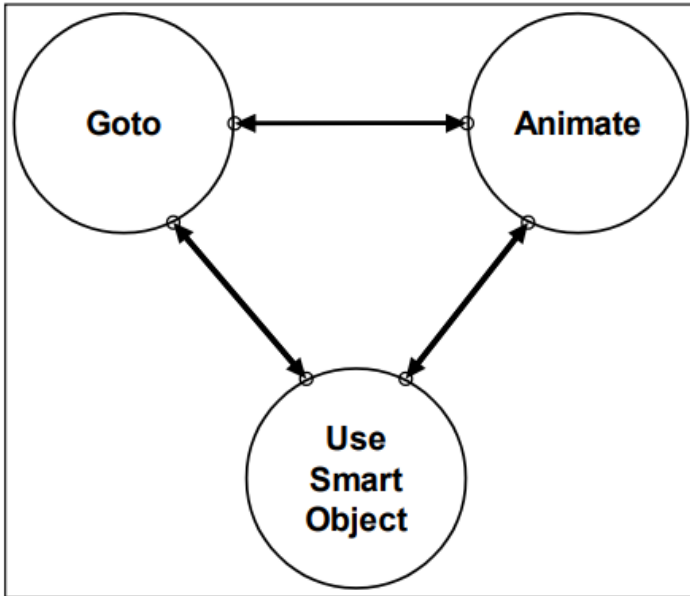
Action/Behavior Planning vs. Path planning

- Same algorithm, different action representation
 - Pathplanning:
 - Cells / Waypoints / Path nodes
 - Node links
 - Action/Behavior planning with A*:
 - Goals
 - Actions, with
 - Action name
 - Precondition
 - Effect (add/remove)
- As with all things in AI, how we represent the problem and the attributes of the problem is going to be key.
- In planning systems, agents have goals and a set of actions. Agent decides how to apply those actions to achieve goals.

Three States and a Plan: The A.I. of F.E.A.R.

Jeff Orkin

Monolith Productions / M.I.T. Media Lab, Cognitive Machines Group
<http://www.jorkin.com>



If the audience of GDC was polled to list the most common A.I. techniques applied to games, undoubtedly the top two answers would be A* and Finite State Machines (FSMs). Nearly every game that exhibits any A.I. at all uses some form of an FSM to control character behavior, and A* to plan paths. *F.E.A.R.* uses these techniques too, but in unconventional ways. The FSM for characters in *F.E.A.R.* has only three states, and we use A* to plan sequences of actions as well as to plan paths. This paper focuses on applying planning in practice, using *F.E.A.R.* as a case study. The emphasis is demonstrating how the planning system improved the process of developing character behaviors for *F.E.A.R.*

We wanted *F.E.A.R.* to be an over-the-top action movie experience, with combat as intense as multiplayer against a team of experienced humans. A.I. take cover, blind fire, dive through windows, flush out the player with grenades, communicate with teammates, and more. So it seems counter-intuitive that our state machine would have only three states.



This is one of the main takeaways of this paper: It's **not that any particular behavior in F.E.A.R. could not be implemented with existing techniques.** Instead, it is the **complexity of the combination** and interaction of all of the behaviors that becomes **unmanageable.**



General Purpose vs. Domain-Specific

Planning: find a sequence of actions to achieve a goal

General purpose: symbolic descriptions of the problems and the domain. The plan generation algorithm the same

Advantage: - opportunity to have clear semantics

Disadvantage: - symbolic description requirement

Domain Specific: The plan generation algorithm depends on the particular domain

Advantage: - can be very efficient

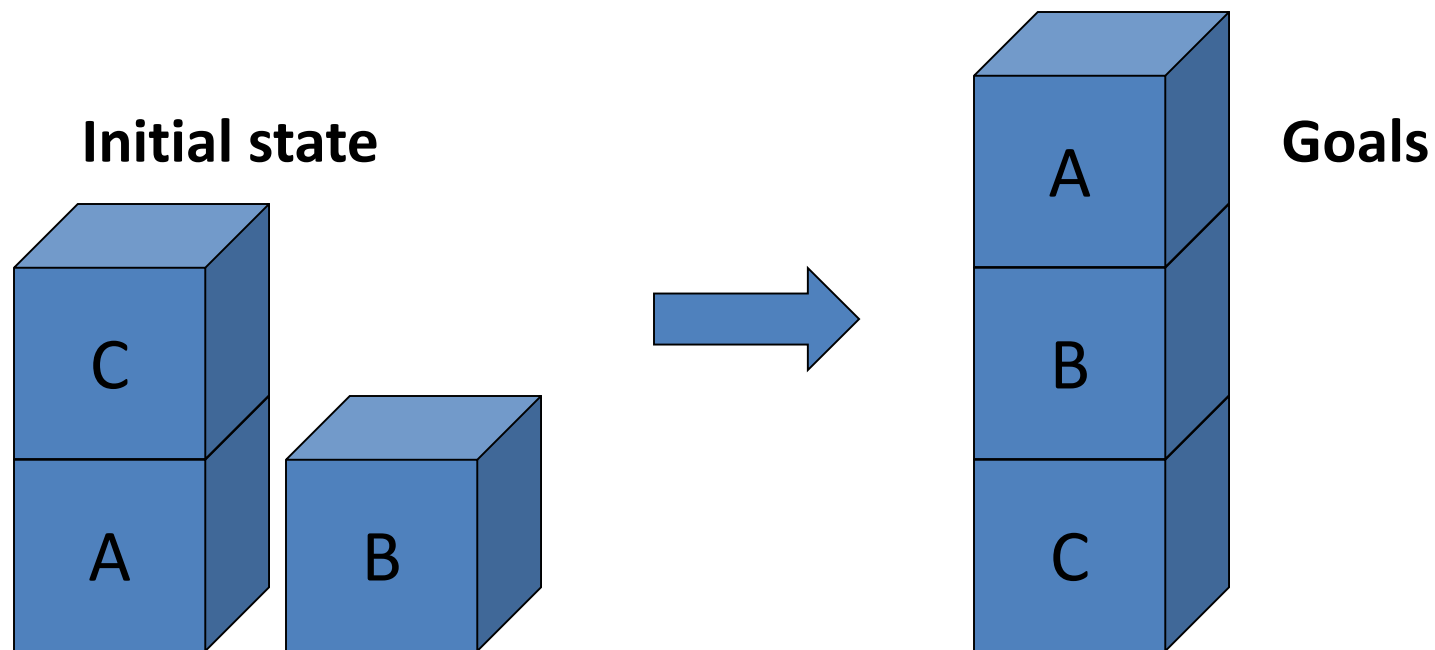
Disadvantage: - lack of clear semantics

- knowledge-engineering for plan generation

STRIPS (Fikes & Nilsson)

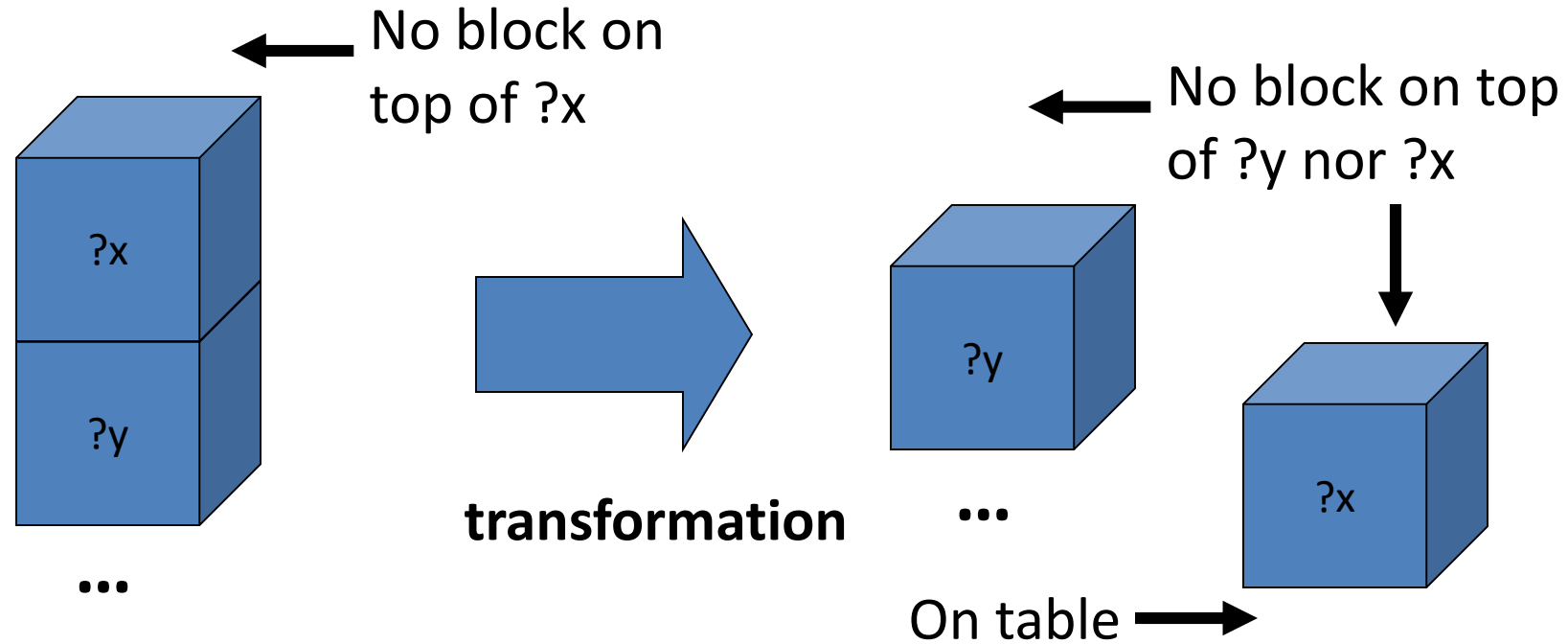
- States
 - at(plane1, Atlanta)
- Goals
 - A particular state, or part of a particular state
- Actions (“operators”)
 - Action Schema

General-Purpose Planning: State & Goals



- **Initial state:** (on A Table) (on C A) (on B Table) (clear B) (clear C)
- **Goals:** (on C Table) (on B C) (on A B) (clear A)

General-Purpose Planning: Operators



Operator: (Unstack ?x)

- **Preconditions:** (on ?x ?y) (clear ?x)
- **Effects:**
 - **Add:** (on ?x table) (clear ?y)
 - **Delete:** (on ?x ?y)

STRIPS actions

State: airport(LAX), airport(ATL), at(plane1, ATL),
at(plane2, LAX), path(ATL, LAX), ~at(plane1, LAX), ...



(All other things that are true or non-true)

Fly (?p, ?from, ?to)

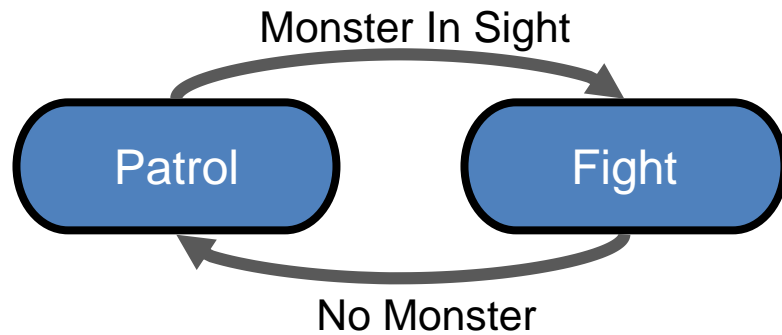
Precondition: at(?p, ?from), plane(?p), airport(?from),
airport(?to), path(?from, ?to), ?from \neq ?to

Effect: at(?p, ?to), \neg at(?p, ?from)

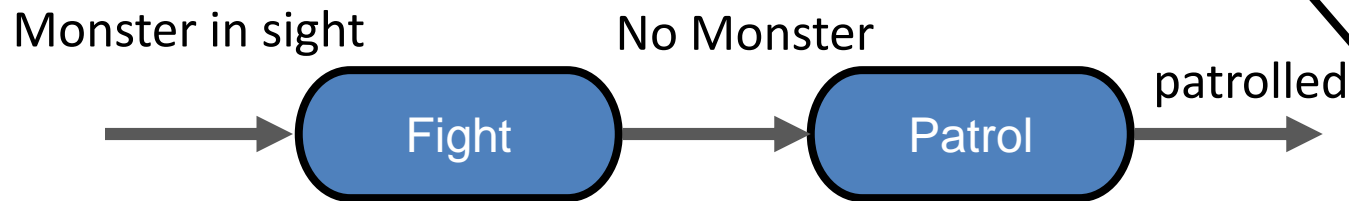
*Also a call to the game engine to play animation or run a function

FSM vs A* Planning

FSM:



A resulting plan:



Planning Operators

•Patrol

- Preconditions:
No Monster
- Effects:
patrolled

•Fight

- Preconditions:
Monster in sight
- Effects:
No Monster

Neither is more powerful than the other

But Planning Gives More Flexibility

- “Separates implementation from data” --- Orkin

reasoning

knowledge

Planning Operators

- **Patrol**

- Preconditions:

- No Monster

- Effects:

- patrolled

- **Fight**

- Preconditions:

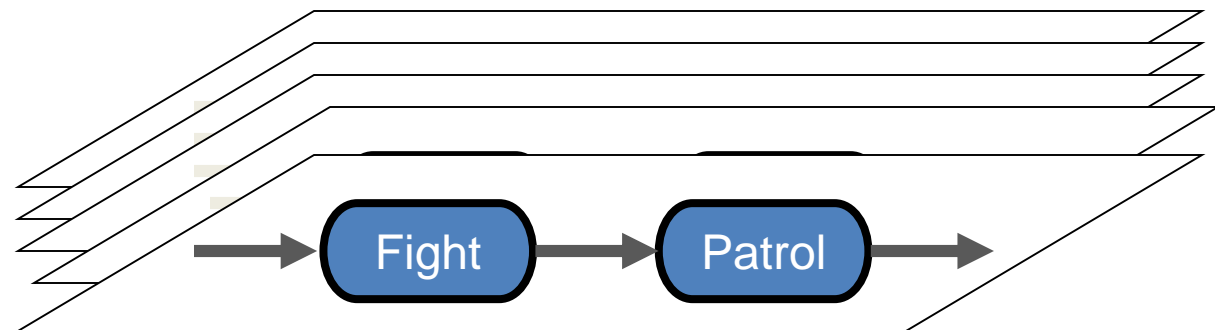
- Monster in sight

- Effects:

- No Monster

- ...

Many potential plans:



If conditions in the state change making the current plan unfeasible: replan!

FSMs vs. Planning

- FSMs tell agents how to behave in every situation
- In planning systems, agents have goals and a set of actions. Agent decides how to apply those actions to goals.
- With planning, “easy” to add goals and actions

But... Does Classical Planning Work for Games?

F.E.A.R. not!



Planning in F.E.A.R.

- Agents need to autonomously use environment to satisfy their goals
- Agent will not do anything without a goal
- Agent types defined by what actions are available to them

Benefits of Planning in F.E.A.R.

- Decoupled goals and actions
 - Each character has own Action Set
 - Allows for late additions in character types
 - Allows for shared information between goals
- Layered behaviors
 - Agents should always try to stay covered.
 - Agents should never leave cover unless threatened and other cover is available
 - Agents should fire from cover as best they can
- Dynamic problem solving
 - Replanning gives the AI the power to adjust to new scenarios
 - AI records obstacles in memory and uses that knowledge later during replanning

Layered Behaviors

- Basic Goal: KillEnemy
 - Satisfied by *Attack* action
- Additional Goal: Dodge
 - Satisfied by *DodgeShuffle* or *DodgeRoll*
- Goals and actions for melee attacks, taking cover, etc.
- With planning, easy to add goals and actions

F.E.A.R. Differences from STRIPS

- Action costs
 - Actions have non-uniform costs in FEAR; FEAR uses A^* to navigate the state-space.
- World state representation
 - The world state representation only allows for the planner to consider one enemy and one weapon at a time.
 - Therefore, systems outside of the planner choose which weapon and which enemy to deal with, and the planner only considers those.

Other Games with Planning

- Empire: Total War
- Fallout 3
- Killzone series

Initial state: gunForSale, ammoForSale, possumAlive, ~gunLoaded, ~hasFood, ~hasGun,
~criminal, ~hasAmmo, ~rich, smellsFunny

Goal state: rich, hasFood

Action: RobBank

PRE: ~rich, hasGun, gunLoaded

EFFECT: rich, criminal

Action: ShootPossum

PRE: ~hasFood, hasGun, gunLoaded, possumAlive

EFFECT: hasFood, ~gunLoaded, ~possumAlive

Action: LoadGun

PRE: hasGun, hasAmmo, ~gunLoaded

EFFECT: gunLoaded, ~has Ammo

Action: BuyGun

PRE: gunForSale, ~hasGun, ~criminal

EFFECT: ~gunForSale, hasGun

Action: BuyAmmo

PRE: ammoForSale, ~hasAmmo

EFFECT: ~ammoForSale, hasAmmo

Action: TakeBath

PRE: smellsFunny

EFFECT: ~smellsFunny

Action: PlayInMud

PRE: ~smellsFunny

EFFECT: smellsFunny

Forward Planning

- State-space search
- Start with initial state
- Applicable actions are those whose preconditions are satisfied by current state.
- Goal test
- Optional action cost
- Any complete graph search algorithm (e.g. A*)

Backward Planning

- State-space search
- Benefit: only consider relevant actions
- Actions must be consistent
- Graph search algorithm

Good, Bad, and Ugly

- Forward chaining

Good, Bad, and Ugly

- Forward chaining
 - Irrelevant actions cause high branching factor

Good, Bad, and Ugly

- Forward chaining
 - Irrelevant actions cause high branching factor
- Backward chaining

Good, Bad, and Ugly

- Forward chaining
 - Irrelevant actions cause high branching factor
- Backward chaining
 - Practical branching factor can be much lower because it only considers necessary actions
 - Total ordering susceptible to long backtracks when effects negate earlier decisions
- Start thinking: More informed? Total order?

Heuristics

- $f() = g() + h()$
- $g()$ is sum of action costs, which can be arbitrary
- How do you estimate the distance to the goal?

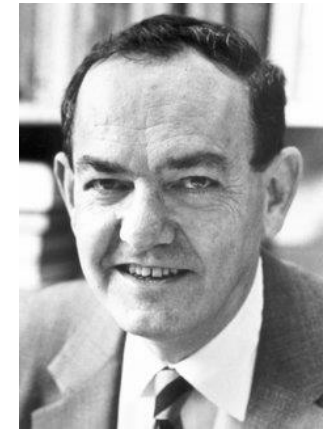
Heuristics and AI

"Heuristic Search Hypothesis. The solutions to problems are represented as symbol structures. A physical symbol system exercises its intelligence in problem solving by search-that is, by generating and progressively modifying symbol structures until it produces a solution structure." – Alan Newell

"A physical symbol system has the necessary and sufficient means for general intelligent action."
— Allen Newell and Herbert A. Simon



https://amturing.acm.org/award_winner/newell_3167755.cfm



<https://www.nobelprize.org/prizes/economic-sciences/1978/simon/biographical/>

Heuristics

- Informs decision into which node (state) to expand
 - A function that estimates how close a state is to a goal
 - Encapsulates domain knowledge for a particular search problem
- Admissible heuristics allow for A*
 - Relax the planning problem (no deletes) and use subgoal independence assumption
 - Inadmissible heuristics break optimality because good plans get left on the frontier
 - A* expands mainly toward the goal, but maintains alternatives in frontiers just in case
 - As heuristics get closer to the true cost, fewer nodes are expanded but usually more work is done per node to compute the heuristic itself

- Heuristic cost should never overestimate the actual cost of a node. This is an “admissible” heuristic
 - i.e. it must be “optimistic” so that we never overlook a node that is actually good
 - A* is optimally efficient: given the information in h , no other optimal search method can expand fewer nodes
 - Memory is a problem for the A* algorithms. Consider IDA*, SMA*
- A problem with less restrictions on its operators is called a relaxed problem
 - The optimal solution of the original problem is also a solution to the relaxed problem and must therefore be at least as expensive as the optimal solution to the relaxed problem

On memory

- In spite of its optimality and completeness, A* still has problems:
 - For most problems, the number of nodes on the frontier of the search space is still exponential in the length of the solution.
 - That is, the search tree can still grow to be as "bushy" as in Breadth-first Search. So there can be problems with respect to the amount of memory needed to run A* search.
- General drawbacks of heuristic (informed) search include the following:
 - need to keep everything on the frontier in memory
 - need to be able to do cost comparisons quickly — if the heuristic function is extremely complex, the search will not be fast.
 - need to choose good heuristics — this is not easy for many problems!
- When at a loss for a good heuristic function, consider a relaxed version of the problem.
 - An exact solution to a relaxed problem might be a good heuristic for the real problem.
 - For example, consider a sliding tile puzzle. One relaxed version of the puzzle is one in which the tiles can simply be picked up and put into place. Therefore, one possible heuristic is to count the number of tiles out of position, since simply placing them would solve the relaxed problem.

Heuristic Search Planning

- Computes heuristic values for each precondition based on graph analysis
 - Benefit: Only do it once as pre-computation step
- Heuristic
 1. Cost of action is *maximum* over costs of preconditions (admissible, but not informed)
 2. Cost of action is *sum* over costs of preconditions (informed, but not admissible)

Benefits of A* Planning

- Decouple goals and actions
 - Can create new character types (mimes vs. mutants)
 - State machines become unmanageable by design team
- Dynamic problem solving
 - Ability to re-plan when failure occurs
- Potentially more realistic combat behavior (goal oriented actions)
- Make it harder to learn how to defeat enemy tactics
- Emergent solutions
- Unanticipated situations can be realistically handled
- Less authoring



Potential Drawbacks of A* Planning

- Slower responses / longer computation
- Less reliable patterns of behavior
- Removal of authorial intent
- Less understandable/explainable than BTree

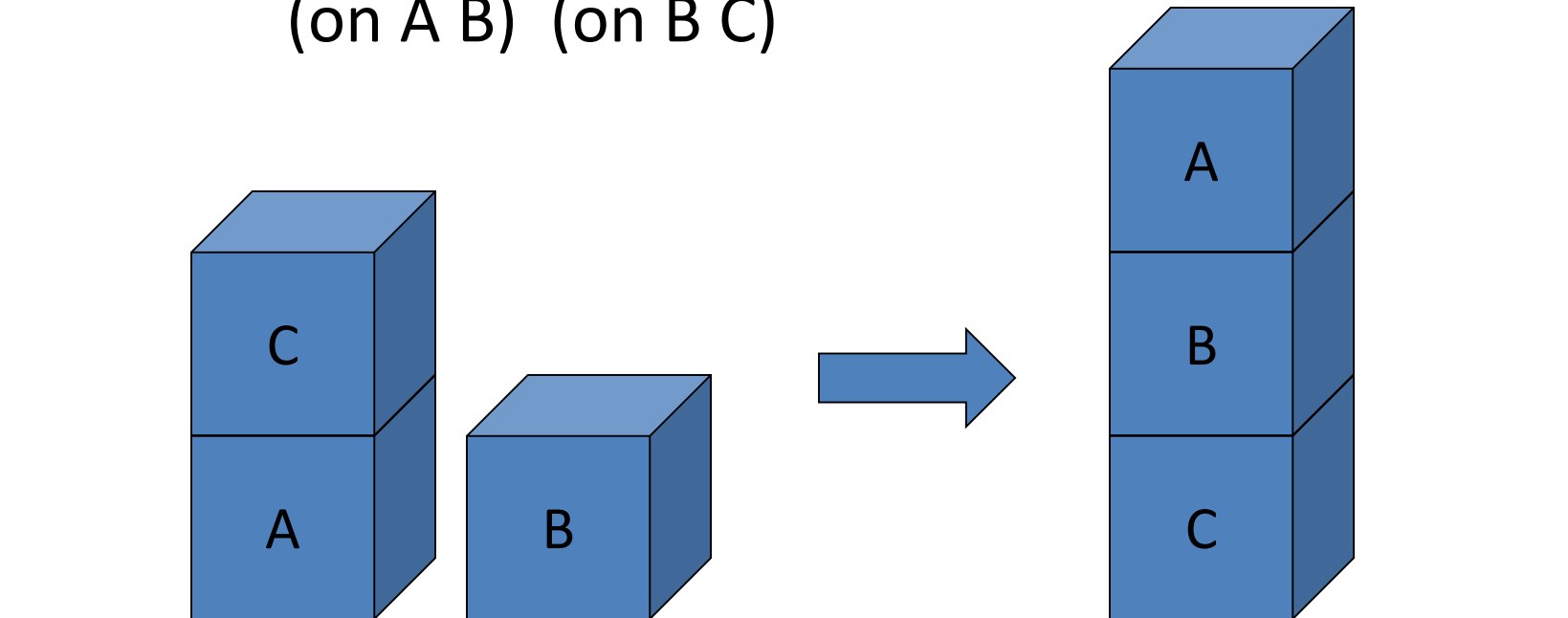
PARTIAL ORDER PLANNING

Partial-Order Planning

- Avoid total ordering (previous examples)
- Partial ordering treats every precondition as a sub-problem to be solved independently
- Reconcile solutions to sub-problems when they interact with each other
- Don't commit to any ordering before strictly necessary
 - Least-commitment planning

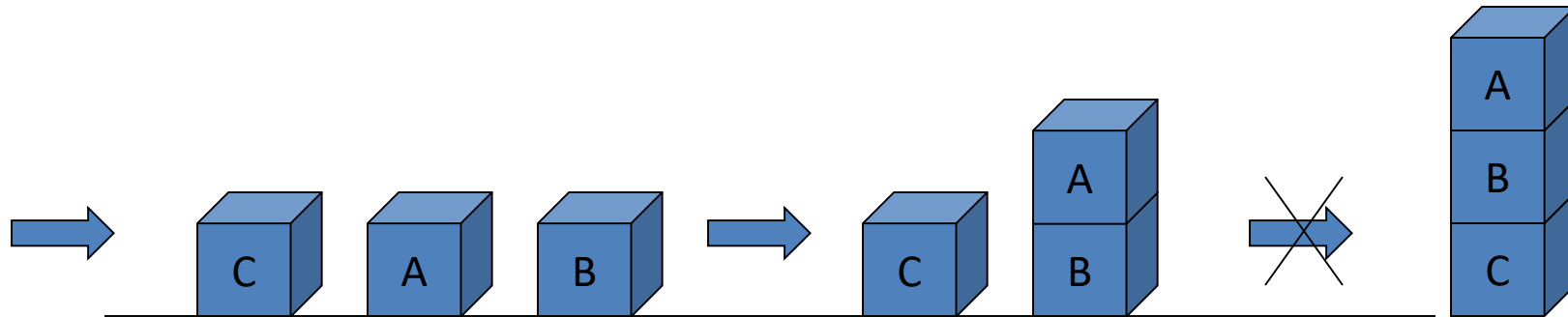
Why Plan-Space Planning?

- 1. Motivation: “Sussman Anomaly”
 - Two subgoals to achieve:
(on A B) (on B C)



Why Plan-Space Planning?

- Problem of state-space search:
 - Try (on A B) first:
 - put C on the Table, then put A on B



- Accidentally wind up with A on B when B is still on the Table
- We can not get B on C without taking A off B
- Try to solve the first subgoal first appears to be mistaken

Partial-Order Planning

- Plan-search rather than state-search
- Plans are made up of:
 - Actions used
 - Ordering constraints
 - Causal links
 - Open preconditions

Partial-Order Planning

- Conflict
 - An action C conflicts with $A \overset{p}{\rightarrow} B$ if C has $\neg p$ as an effect AND C can occur between A and B
- Consistent Plans
 - No cycles in ordering constraints
 - No conflicts with causal links
- Solution
 - Consistent plan with no open preconditions

POP Algorithm

- Start with initial plan [Start, Finish] where $\text{Start} < \text{Finish}$.
- Arbitrarily pick one open precondition p
- Generate successor plans for every possible consistent way of selecting an action A that achieves p
- Add new causal link to plan, and resolve conflicts (if necessary)

POP Heuristics

- Less understanding of how to create accurate heuristics for POP than total-order planning.
- Obvious heuristic: number of open preconditions
- Most-constrained-variable

```
agenda = { make_empty_plan(init, goal) }
```

```
current = pop(agenda)
```

```
WHILE agenda not empty and current has flaws DO:
```

```
    flaw = pick_flaw(current)
```

```
    IF flaw isa open condition flaw DO:
```

```
        FOREACH op in library that has an effect that unifies with o.c. DO:
```

```
            successors += make_new_plan_from_new(...)
```

```
        FOREACH op in current that is before and has an effect that unifies with o.c.
```

```
DO:
```

```
    successors += make_new_plan_reuse(...)
```

```
    IF a condition in init unifies with o.c. DO:
```

```
        successors += make_new_plan_from_init(...)
```

```
    IF a condition is negative and CWA applies DO:
```

```
        successors += make_new_plan_from_cwa(...)
```

```
    ELSE IF flaw isa causal threat flaw DO:
```

```
        successors += make_new_plan_promote(...)
```

```
        successors += make_new_plan_demote(...)
```

```
    agenda = agenda + successors
```

```
    current = pop(agenda)
```

← Insert sort

```
END WHILE
```

```
RETURN current or nil
```


Must read: <http://www.cs.umd.edu/~nau/papers/nau2013game.pdf>

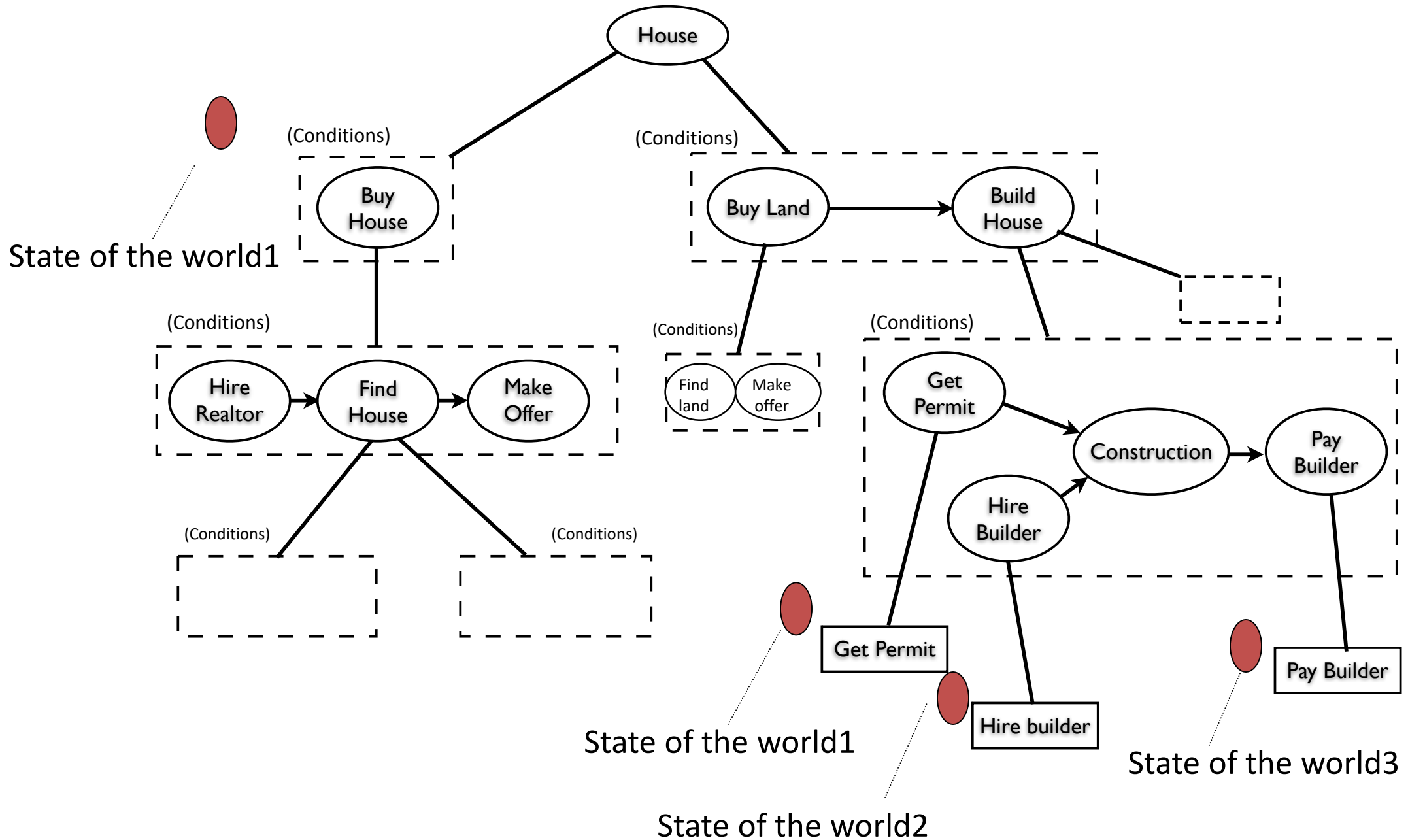
HTN PLANNING

Hierarchical Task Network (HTN) Planning

- Sometimes you know how to do things
- Example: going on a trip
 - Domain-independent planner: lots of combinations of vehicles and routes
 - Experienced human: a few recipes
 - Buy air plane ticket
 - Go from home to airport
 - Fly to other airport
 - Go from airport to destination
- Describe recipes as tasks that can be decomposed to sub-tasks (tasks == goals)

Hierarchical Task Network (HTN) Planning

- Hierarchical decomposition of plans
- Initial plan describes high-level actions
 - E.g. “Build House”, “Find Player”, etc
- Refine plans using action decompositions
- Process continues until the agent reaches primitive actions

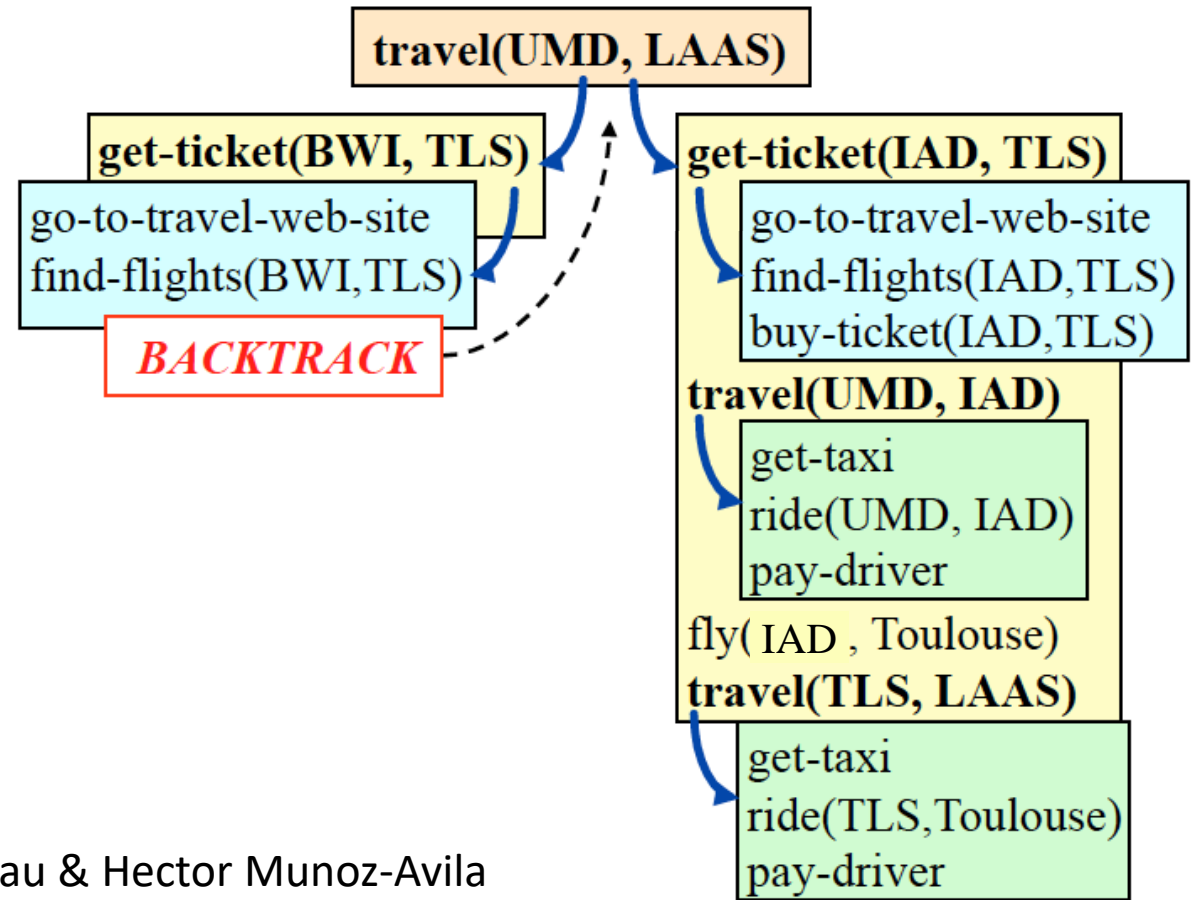
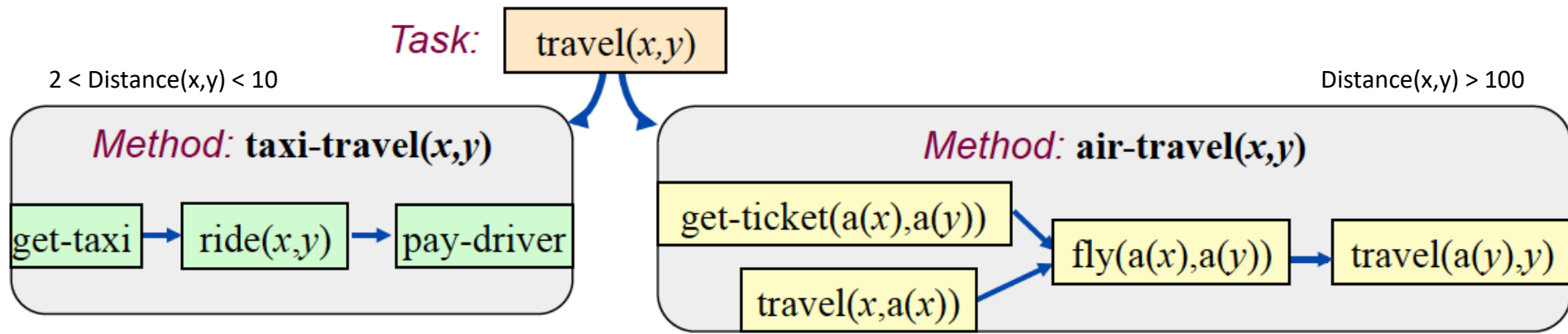


HTN Planning: Idea

- Each “task” segments the planning problem into smaller and smaller problems
- Only plan out the high-level plan at first
 - Replan if conditions break it
- Only plan out with primitive actions the current high-level task
- *Note:* can use heuristic A* planning algorithms discussed for primitive action planning

HTN Planner

- Given a task...
- Pick method with conditions that match the current world state (or pick randomly)
- Planning process
 - When you get to primitive, update state, repeat
 - Execute full plan (monitor world state)
- Can also create a partial plan
 - But early decisions can affect later conditions
- Replanning
 - If plan breaks, just pop up a level and re-decompose
 - Keep popping up decomposition fails
- SHOP2



SHOP2

```
(:method
  ; head
  (transport-person ?p ?c2)

  ; precondition
  (and
    (at ?p ?c1)
    (aircraft ?a)
    (at ?a ?c3)
    (different ?c1 ?c3))

  ; subtasks
  (:ordered
    (move-aircraft ?a ?c1)
    (board ?p ?a ?c1)
    (move-aircraft ?a ?c2)
    (debark ?p ?a ?c2)))
```

*primitive actions have
preconditions and effects



Killzone 2

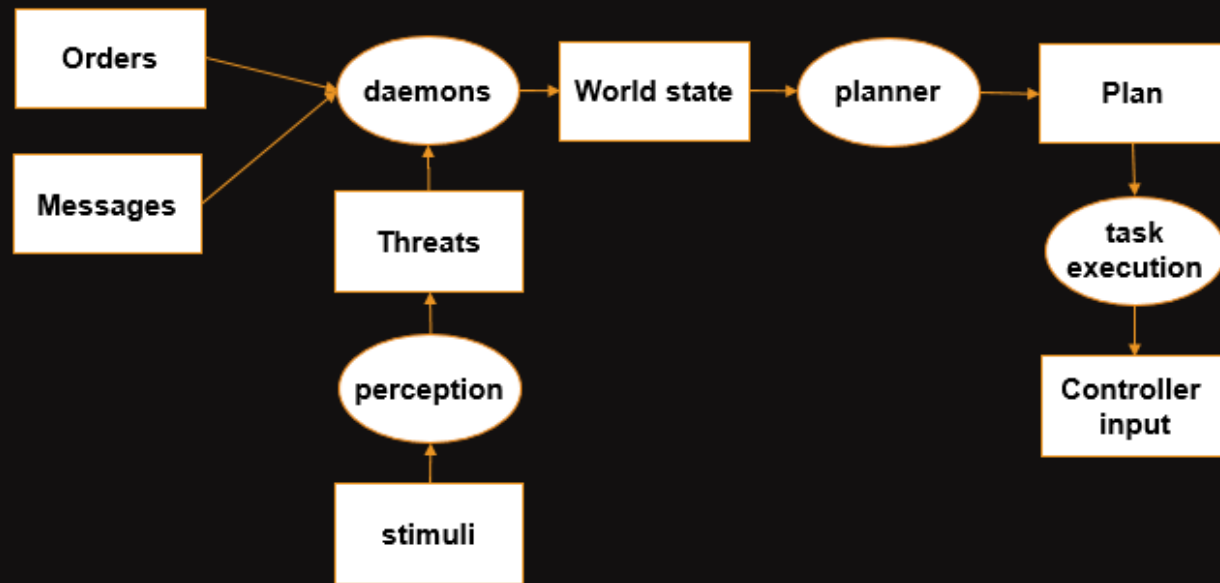
Developer Guerrilla Games **Publisher** Sony Computer Entertainment **Release Date** February 27, 2009

Platforms PlayStation 3

9.4 **Review** >



Individual AI



Game AI Conference, Paris, June 2009

KILLZONE™

KILLZONE 2



- Special-purpose HTN planner for planning at the squad level
 - » Method and operator syntax similar to SHOP's and SHOP2's
 - » Quickly generates a linear plan that would work if nothing interferes
 - » Replan several times per second as the world changes
- **Why it worked:**
 - » Very different objective from a bridge tournament
 - » Don't *want* to look for the best possible play
 - » Need actions that appear believable and consistent to human users
 - » Need them very quickly

Individual AI : HTN Planner

```
(:method (select_weapon_and_attack_as_turret ?inp_threat)
  ( branch_use_bullets // Only use bullets against humanoids and turrets.
    (and (or (threat ?inp_threat humanoid) (threat ?inp_threat turret) )
      (distance_to_threat ?inp_threat ?threat_distance)
      (call lt ?threat_distance @weapon_bullet_max_range) )
    ( attack_as_turret_using_weapon_pref ?inp_threat wp_bullets)
  )
  ( branch_use_rockets // Don't use rockets against humanoids and turrets.
    (and (not (threat ?inp_threat humanoid)) (not (threat ?inp_threat turret))
      (distance_to_threat ?inp_threat ?threat_distance)
      (call lt ?threat_distance @weapon_rocket_max_range) )
    (attack_as_turret_using_weapon_pref ?inp_threat wp_rockets)
  )
)
```

Individual AI : Plan monitoring

Plan fails when current task fails

Abort current plan preemptively when

- Better plan available
- Current plan no longer feasible

So, we keep planning @5hz, but:

- Prevent twitchy behavior
- Prevent unnecessary checks (optimizations)
- Combine planning and monitoring using **continue branches**
 - Branch with “continue” as only task in plan
 - When encountered during planning, keep current plan.

Individual AI : Random Numbers



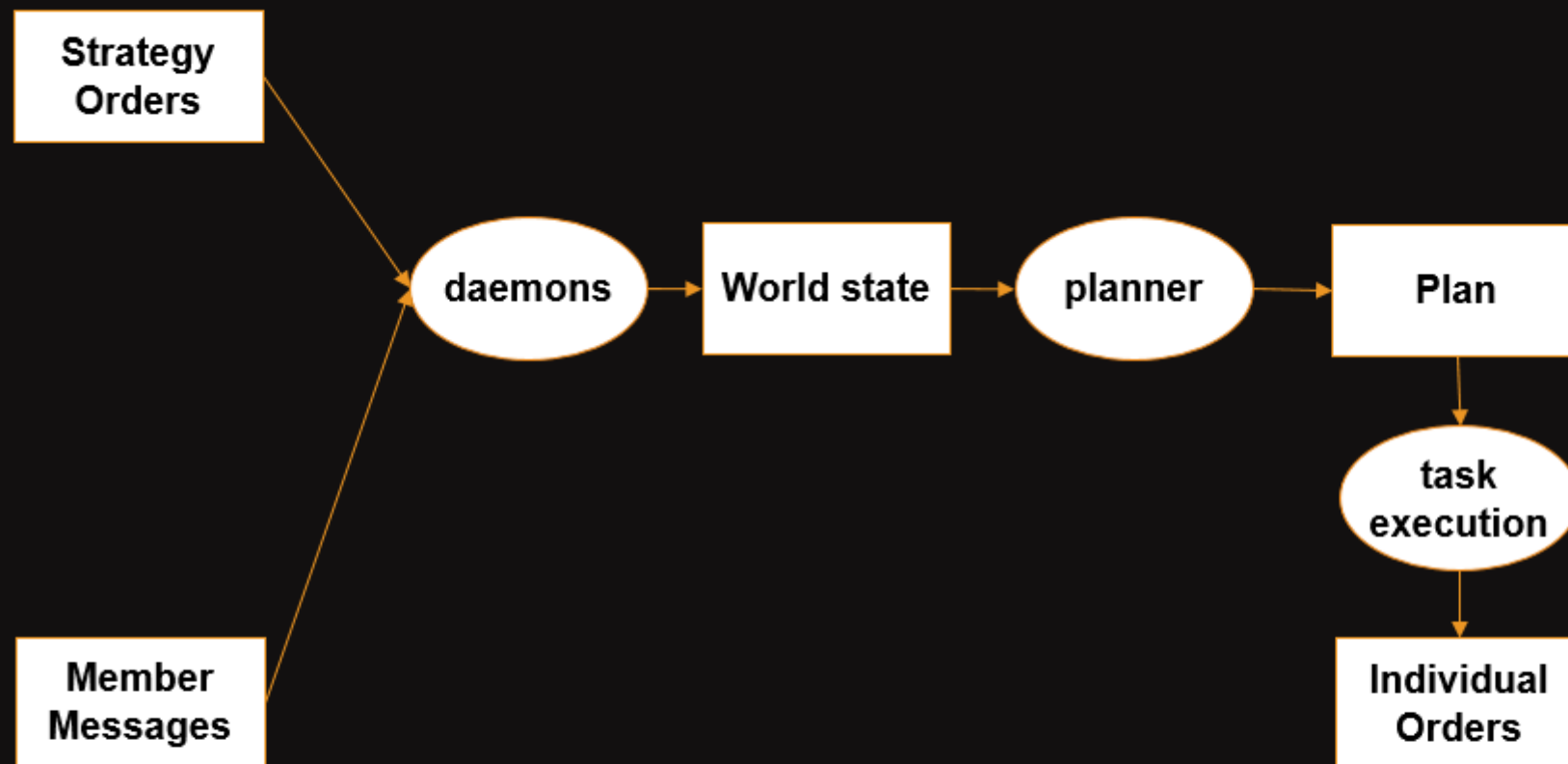
Individual bot domain

- 360 methods
- 1048 branches
- 138 behaviors
- 147 continue branches

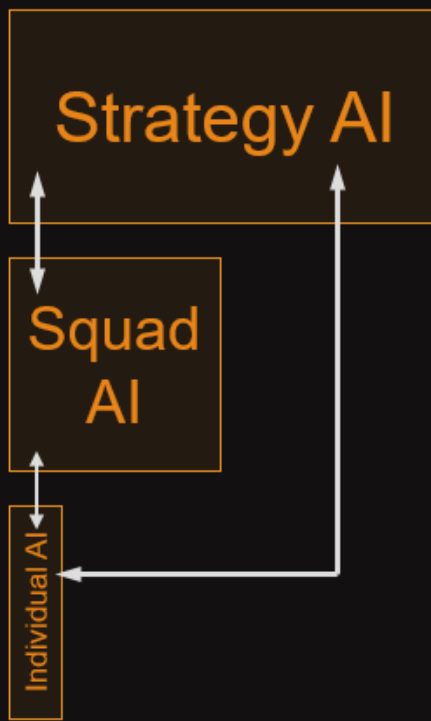
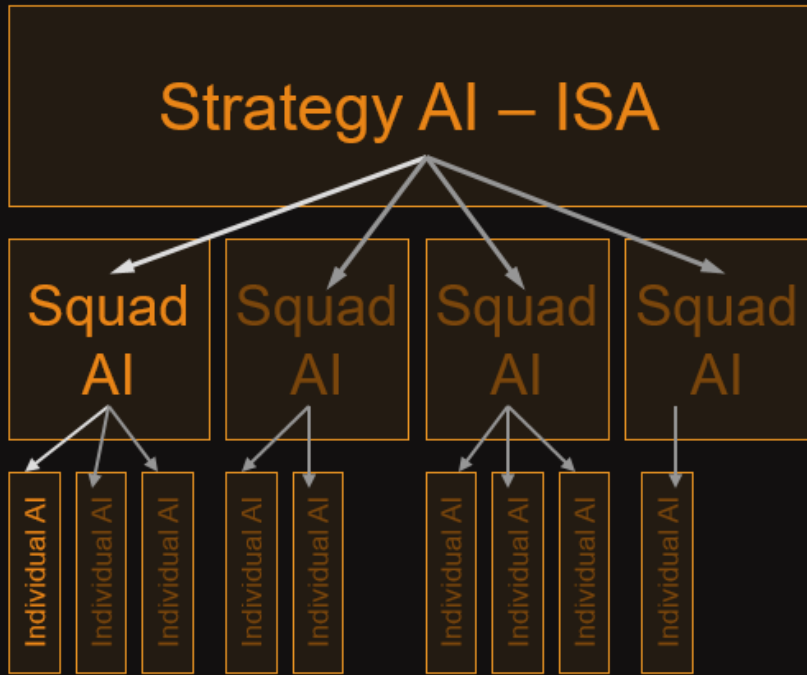
During multiplayer game (14 bots / max. 10 turrets / max. 6 drones / squads)

- Approx. 500 plans generated per second
- Approx. 8000 decompositions per second
- Avg. 15 decompositions per planning.
- Approx 24000 branch evaluations per second.

Squad AI



Architecture



- Strategy to Squad: Orders: Defend, Advance, ...
- Squad to Strategy: Feedback: Order failed
- Squad to Individual: Orders: MoveTo
- Individual to Squad: Combat information
- Strategy to Individual: Orders: Assassination target
- Individual to Strategy: Request reassignment

Given state s , Tasks T , Domain D

Let P = empty plan

Let $T_0 = \{t \in T \mid \text{no task comes before } t\}$

Loop

 If T_0 is empty, return P

 Pick any $t \in T_0$

 If t is primitive

 Modify s according to effects

 Add t to P

 Update T by removing t

$T_0 = \{t \in T \mid \text{no task comes before } t\}$

 Else

 Let M = a method for t with true preconditions in state s

 If M is empty return FAIL

 Modify T : remove t , add subtasks of M (note order constraints)

 If M has subtasks

$T_0 = \{t \in \text{subtasks} \mid \text{no task comes before } t\}$

 Else

$T_0 = \{t \in T \mid \text{no task comes before } t\}$

Repeat

HTN vs. A* Planning

- What are the advantages or disadvantages of HTN planning?
A* planning? ~~Partial-order planning?~~
 - Search time?
 - Authoring time?
 - Optimality?
 - Novelty/Predictability?
 - Partial solutions?

- HTN: Takes advantage of knowledge of how things are done. Often is much faster, but decomposition could slow it down depending on your level of granularity. Less/no improvisation
- A*: Opportunistic discovery and creativity. Harder for player to learn/anticipate.
- POP: Efficient handling of semi-decomposable problems

Planning Under Uncertainty

- What if actions can fail?

Planning Under Uncertainty

- What do you do if you end up in a state you do not desire?

Planning Under Uncertainty

- What do you do if you end up in a state you do not desire?
 - Replan
 - Create a policy

Planning and Games – Future

- Plan recognition
- Story generation
- Where else?

Reactive Planning

- Real-time decision making by performing one action every instant
- Instead of focusing on state, focus on action
- Examples
 - State-action table
 - Universal plan
 - Behavior trees
 - Rule systems

Resources

- F.E.A.R AI: https://www.youtube.com/watch?v=rf2T_j-FIDE
- Dana Nau HTN and games presentation
 - <http://www.cs.umd.edu/~nau/papers/nau2013game.pdf>
- Killzone 2 AI:
 - <https://www.youtube.com/watch?v=7oWKCLdsGTE>
 - <http://www.ign.com/boards/threads/killzone-2-enemy-a-i-is-it-up-there-with-fear-as-1.177634641/>
- Killzone 3:
 - http://www.gameapro.com/GameAIPro/GameAIPro_Chapter29_Hierarchical_AI_for_Multiplayer_Bots_in_Killzone_3.pdf

Resources

- Planning in modern games:
 - <http://aigamedev.com/open/review/planning-in-games/>
 - Nau HTN planning in Killzone: <http://www.cs.umd.edu/~nau/papers/nau2013game.pdf>
 - G.O.A.P: <http://web.media.mit.edu/~jorkin/goap.html>
 - Workshop at ICAPS 2013: <http://icaps13.icaps-conference.org/technical-program/workshop-program/planning-in-games/>
 - The AI of F.E.A.R.: http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf
- SHOP, JSHOP, SHOP2, JSHOP2, Pyhop (HTN planners)
 - <http://www.cs.umd.edu/projects/shop/>
 - <https://bitbucket.org/dananau/pyhop/src/default/>
- Scala impl. of partial-order planning
 - <https://github.com/boyangli/Scalpo>
- Other planners:
 - http://www.cs.cmu.edu/~jcl/compileplan/compiling_planner.html
- Facing your F.E.A.R. lecture: https://www.youtube.com/watch?v=rf2T_j-FIDE