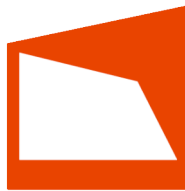


3 States and a Plan: The AI of F.E.A.R.



MONOLITH

Jeff Orkin
Monolith Productions/
MIT Media Lab

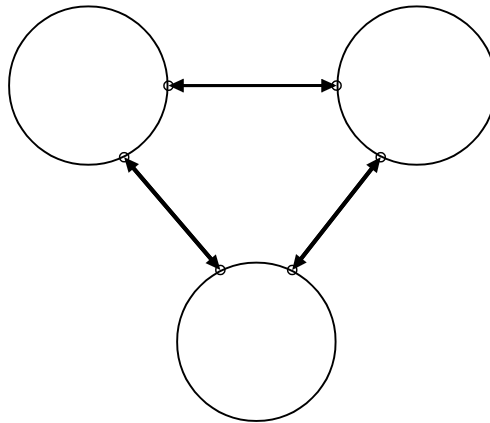
GameDevelopers
Conference



We wanted FEAR to be an over the top action movie experience, with combat as intense as multiplayer against a team of experienced humans.

AI work in squads to take cover, lay suppression fire, blind fire, dive thru windows to safety, flush out player w/ grenades, etc.

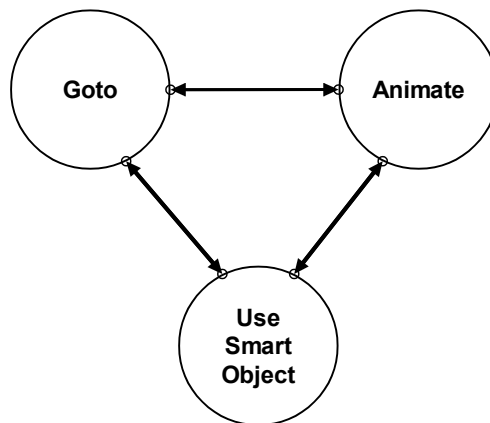
FSM: 3 States



GameDevelopers
Conference

So it seems counter-intuitive that our finite state machine would have only 3 states.

FSM: 3 States

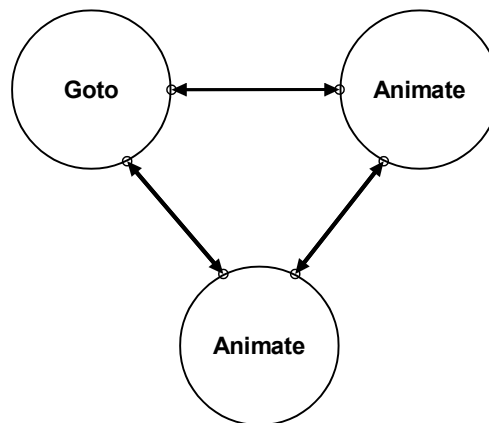


GameDevelopers
Conference

These are the only states in our Finite State Machine: Goto, Animate, and UseSmartObject.

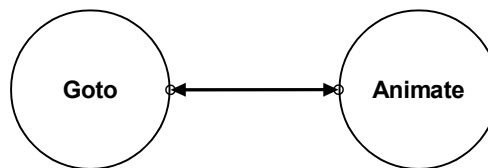
UseSmartObject is just a fancy animate state, specific to our systems at Monolith. Rather than explicitly telling the state to play a particular animation, we give it a game database record for a set of animations. This is really an implementation detail...

FSM: 3 States



...and for the purposes of this talk, we can just consider it to be the same as animate.

FSM: 2 States!



So what we are really talking about is a FSM with only two states! Goto and Animate.

As much as we like to pat ourselves on the back, and talk about how smart our AI are, the reality is that all AI ever do is move around and play animations! Think about it. An AI going for cover is just moving to some position, then playing a duck or lean animation. An AI attacking just loops a firing animation. Sure, the animation system does additional work on keyframes, such as playing footstep sounds, creating weapon effects, or telling the weapon to fire a bullet, but as far as the AI's behavior and decision making goes, he's just moving around or playing an animation.

And in fact, moving is performed by playing an animation! And various animations (like recoils, jumps, and falls) may move the character. So the only difference between Goto and animate is that Goto is playing an animation while heading towards some specific destination, while Animate just plays the animation, which may have a side effect of moving the character some arbitrary distance.

The hard part of modeling character behavior is determining when to transition between these two states, and what parameters to set. Which destination should we go to? How fast should we go? Where is the destination? Which animation should I play? Should the animation

FSM: Transition Logic

```
void StateAttack::Update()  
{  
    //...  
    if( Ammo == 0 )  
    {  
        pState = Reload(bCrouch);  
        return;  
    }  
    //...  
}
```

GameDevelopers
Conference

The logic determining when to transition from one state to another, and which parameters to specify have to live somewhere. This logic may be written directly into the code of the states themselves, or may be external in some kind of table, or script file, or may be represented visually in some kind of FSM modeling tool. However the logic is specified, in all of these cases it has to be specified manually by a programmer or designer.

For FEAR, this is where planning comes in. We decided to move that logic into a planning system, rather than embedding it in the FSM as games typically have in the past. As you will see in this talk, a planning system gives AI the knowledge they need to be able to make their own decisions about when to transition from one state to another. This relieves the programmer or designer of a burden that gets bigger with each generation of games.

Shogo, 1998



GameDevelopers
Conference

Let's look at the recent history of game AI. In the early generations of shooters, players were happy if the AI noticed them and started attacking.

No One Lives Forever, 2000



GameDevelopers
Conference

After a few years, players wanted AI that could use the environment to their advantage, so we start seeing AI that can take cover, and even flip over furniture to create their own cover.

In NOLF and NOLF2, AI pop randomly in and out of cover, like a shooting gallery.

F.E.A.R., 2005

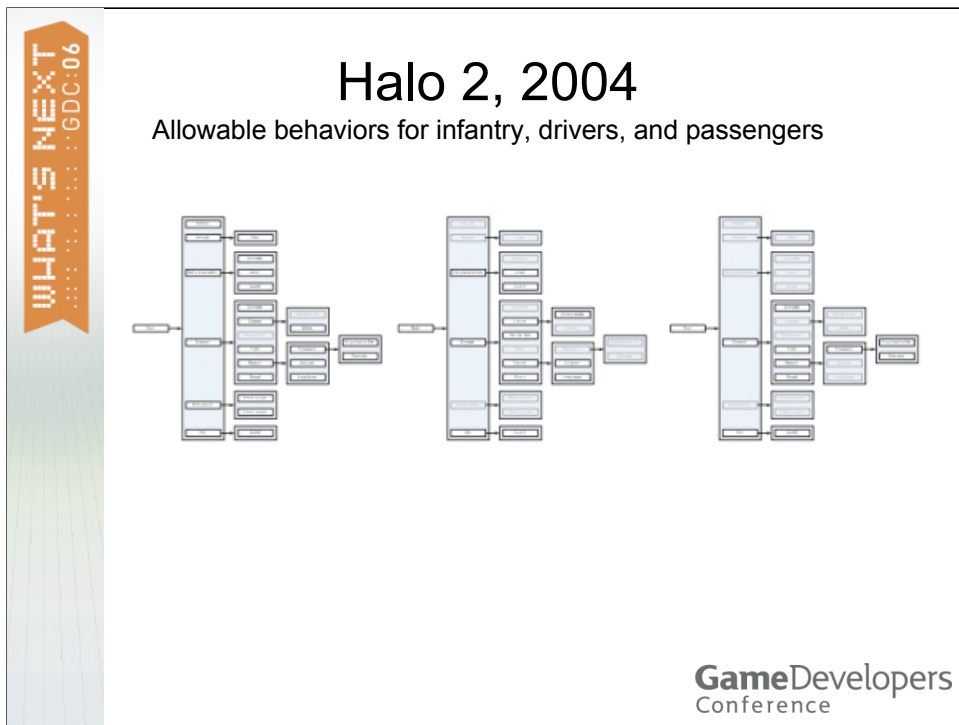


GameDevelopers
Conference

Today, players expect more realism, to complement the realism of the real-time lighting and physics environments. If the simulation of the AI behavior is not as realistic as the lighting and physics simulation, it can be jarring.

In FEAR AI use cover more tactically, coordinating with a squad to lay suppression fire while others advance. AI only leave cover when threatened, and blind fire if they have no better position.

With each layer of realism, the behavior gets more and more complex. The complexity required today for AAA titles is getting unmanageable. At last year's GDC, Damian Isla gave a great talk about managing complexity in the Halo 2 AI. This talk could be thought of as a variation on the same theme.



Here is a diagram from Damian's paper, and without even reading any of the text, it is obvious that things are getting quite complex. These characters have many interacting behaviors.

So, this diagram is evidence that complexity is a big problem for all of us. This rapidly growing complexity is a problem for all game AI developers, and introducing real-time planning was our attempt at solving the problem.

This is one of the main takeaways from this talk: It's not that any particular behavior in FEAR 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.

FOR THIS CLASS WE CAN SUMMARIZE THIS WHOLE TALK BY SAYING THAT ORKIN GENERATES THESE HIERARCHIES "ON THE FLY" BY REASONING ABOUT THE RELATIONSHIP BETWEEN A CHARACTER'S POSSIBLE ACTIONS AND THE CURRENT ENVIRONMENT. -CR

FSM vs Planning

FSM

- How

Planning

- What

Let's compare finite state machines to planning. A finite state machine tells an AI exactly how to accomplish his goals. A planning system tells the AI what his goals and actions are, and lets the AI decide how to sequence actions to satisfy goals.

FSM vs Planning

FSM

- How
- Procedural

Planning

- What
- Declarative

Another way to say this is that finite state machines are procedural programming, while planning is declarative.

I'll explain later how we used these two systems together.

Motivation



GameDevelopers
Conference

My motivation for going down this path was at the beginning of preproduction for FEAR, when I was looking at all of the behaviors designers were asking for. I figured that there's only one of me, but there are lots of AI. If these AI are really so damn smart, I should be able to delegate some of my workload to these guys, we'll be in good shape. If we want squad behavior in addition to individual unit behavior, it's going to take more man-hours to develop. If they can figure out some things on their own, then we'll be all set!

Here's the Plan:

- STRIPS Planning Overview
- Planning in F.E.A.R.
- ~~Differences from STRIPS~~
- ~~Squad Behaviors & Communication~~
- Beyond F.E.A.R.

This talk focuses on demonstrating how planning improved the process of developing character behaviors for FEAR, how planning integrated with our tools, and how it supported higher level squad behaviors and communication.

We'll start with a brief overview of STRIPS planning. Then we'll show how we used planning in FEAR. We'll talk about how our planning system differs from STRIPS. Then we'll talk about how planning impacts squad behaviors. And we'll wrap up with some discussion of where we can go from here.

CR: NOT COVERING STRIPS DETAILED (TOO ACADEMIC)

NOT COVERING SQUAD BEH., SINCE ESSENTIALLY AD HOC
AND NOTHING TO DO WITH PLANNING

What is Planning?

- Planning is a formalized process of searching for sequence of actions to satisfy a goal.
- Process is called “Plan Formulation.”

STRIPS Planning



...in a nutshell

GameDevelopers
Conference

The planning system that we implemented for FEAR most closely resembles the STRIPS planning system from academia. Here is a very brief overview of STRIPS planning.

STRIPS Planning

STRIPS =
STanford **R**esearch **I**nstitute
Problem **S**olver

GameDevelopers
Conference

STRIPS was developed at Stanford in 1970, and the name is simply an acronym for the STanford Research Institute Problem Solver.

STRIPS Planning

- STRIPS Goal:
Desired state of the world to reach.
- STRIPS Actions:
 - Preconditions
 - Effects

STRIPS consists of goals and actions, where goals describe some desired state of the world to reach, and actions are defined in terms of preconditions and effects. An action may only execute if some preconditions are met, and each action changes the state of the world in some way.



A word about what we mean by “states”...

States: FSM



Attack



Search

GameDevelopers
Conference

When we talk about states in the context of a Finite State Machine, we're talking about procedural states, which update every frame, run logic, play animations, create effects, and fire weapons.

States: Planning

Represented as a logical sentence:

$$\text{AtLocation}(\text{Home}) \wedge \text{Wearing}(\text{Tie})$$

Represented as a vector:

$$(\text{AtLocation}, \text{Wearing}) = (\text{Home}, \text{Tie})$$


GameDevelopers
Conference

In planning, we represent the state of the world as a conjunction of literals. Another way to phrase it is to say that we represent the state of the world as an assignment to some set of variables that collectively describe the world at some point in time. In this slide we are representing the state where someone is at home and wearing a tie. We can represent it as a logical expression, or as a vector of variable values.

States: Planning

Example: Lemonade Stand

(weather, #lemons, \$\$) =

( ,  , ) or

( ,  , )

Desired (Goal, Final) State

Example: Lemonade Stand

(weather, #lemons, \$\$) =

(-- , -- , )

[slide added by C. Rich]

GameDevelopers
Conference

The desired final state, which is also a vector of variables, may have “don’t care” values. E.g., for the lemonade stand, all I really care about at the end is that I have a pile of money. I don’t really care how many lemons I have left or the weather, though obviously the number of lemons and the weather will affect my sales. Note there is a clue here: If all I care about is the amount of money, if I have some other action (other than selling lemonade) which can get me to that state (such as robbing a bank :-), then I should consider planning to do it.

-C. Rich

STRIPS Planning Example



Here is an example of how STRIPS works. Let's say that Alma is hungry.

STRIPS Planning Example



GameDevelopers
Conference

Alma could call Domino's and order a pizza to satisfy her hunger.

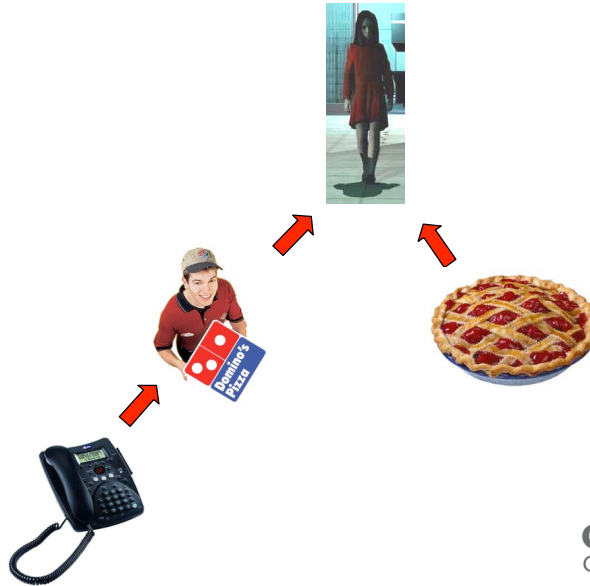
STRIPS Planning Example



GameDevelopers
Conference

But Alma can only order a pizza if she has the phone number for Domino's.

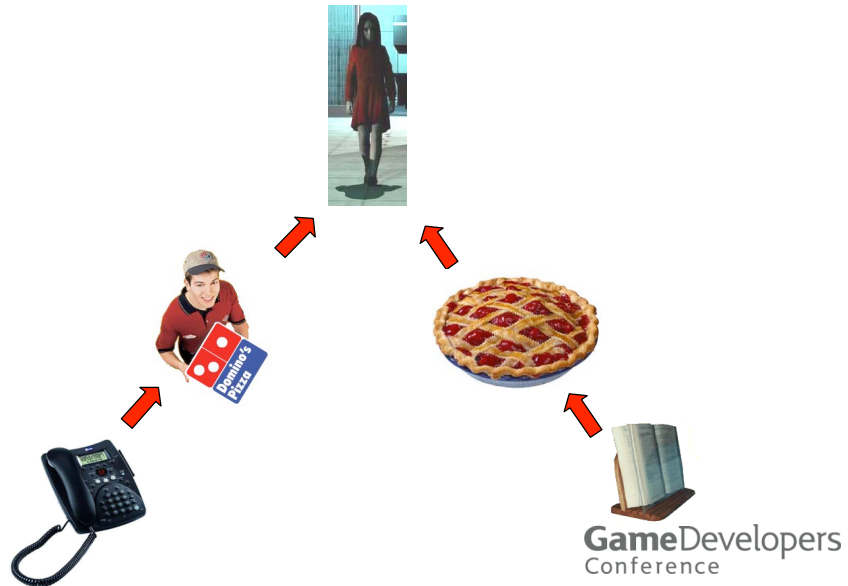
STRIPS Planning Example



GameDevelopers
Conference

Pizza is not her only option however. Alternatively, she could bake a pie.

STRIPS Planning Example

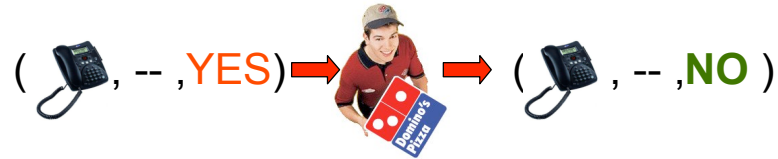


But she can only bake a pie if she has a recipe.

STRIPS Planning Example

State: (phone#, recipe, hungry?)

Goal: (-- , -- , **NO**)



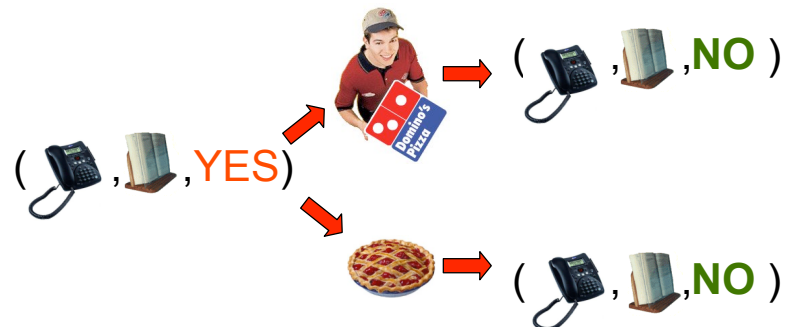
GameDevelopers
Conference

So, Alma's goal is to get to a state of the world where she is not hungry. She has two actions she can take to satisfy that goal: calling Domino's or baking a pie. If she is currently in a state of the world where she has the phone number for Domino's, then she can formulate the plan of calling Domino's to satisfy her hunger. Alternatively, if she is in the state of the world where she has a recipe, she can bake a pie.

STRIPS Planning Example

State: (phone#, recipe, hungry?)

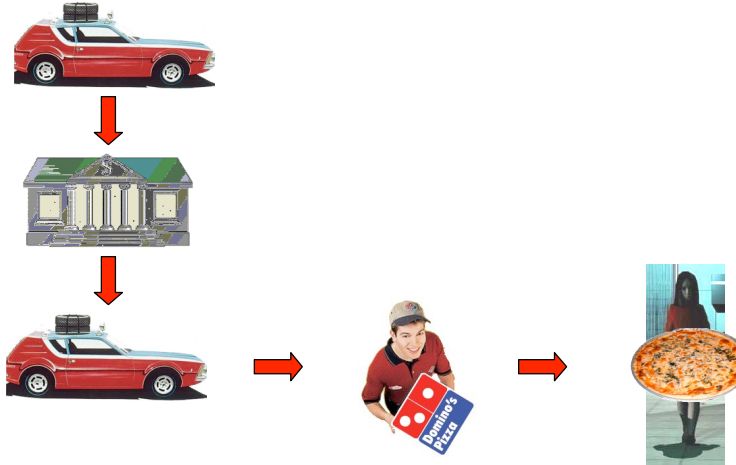
Goal: (-- , -- , **NO**)



GameDevelopers
Conference

If Alma is in the fortunate situation where she has both a phone number and a recipe, either plan is valid to satisfy the goal. We'll talk later about ways to force the planner to prefer one plan over another. If Alma has neither a phone number or a recipe, she is out of luck; there is no possible plan that can be formulated to satisfy her hunger.

STRIPS Planning Example



GameDevelopers
Conference


These examples show trivially simple plans with only one action, but in reality a plan may have an arbitrary number of actions, chained by preconditions and effects. For instance, if ordering pizza has a precondition that Alma has enough money, the satisfying plan may require first driving to the bank.

STRIPS Planning Example

State: (phone#, recipe, hungry?)



Action

Preconditions: ( , -- , --)

Effects:

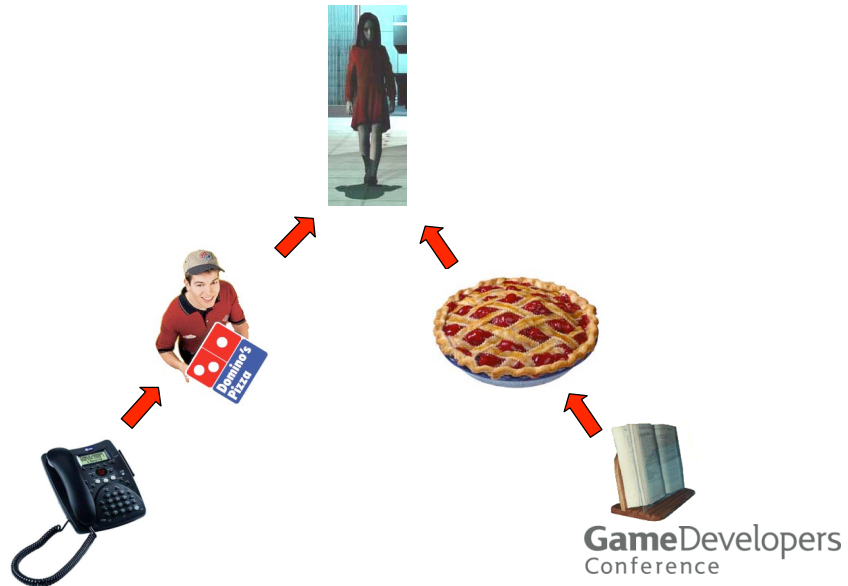
Delete List: Hungry(**YES**)

Add List: Hungry(**NO**)

GameDevelopers
Conference

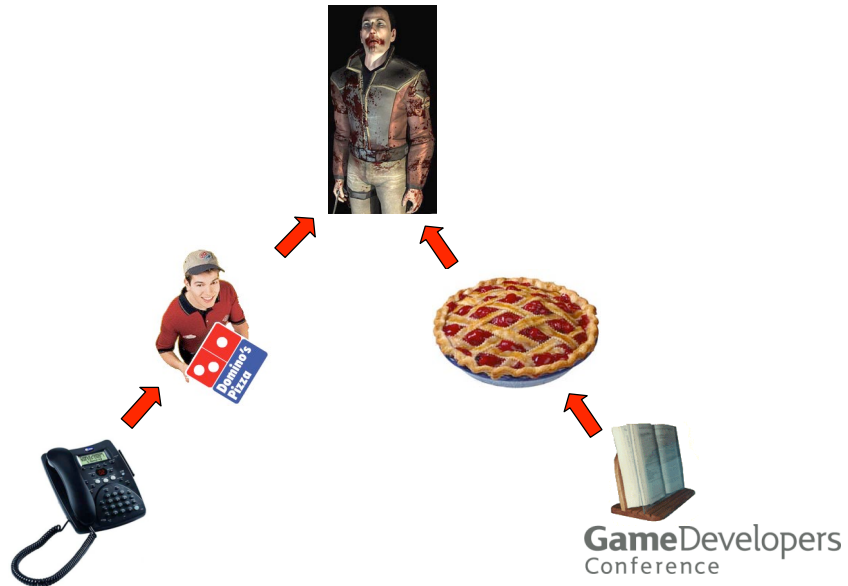
We have previously described what a Goal is in STRIPS. Now we will describe an Action. A STRIPS Action is defined by its Preconditions and Effects. The Preconditions are described in terms of the state of the world, just like we defined our Goal. The Effects are described with list of modifications to the state of the world. First the Delete List removes knowledge about the world, then the Add List adds new knowledge. So, the Order Pizza Action first removes knowledge that Alma is hungry, then adds knowledge that she is not hungry.

STRIPS Planning Example



Now, back to our original example, there are two possible plans for feeding Alma. But what if instead we are planning for Paxton Fettel, who is a cannibal?

STRIPS Planning Example



Neither of these plans will satisfy someone who only eats human flesh.

STRIPS Planning Example



GameDevelopers
Conference

We need a new Action to satisfy Fettel. The Eat Human action has a precondition that we have a fork and knife.

STRIPS Planning Example



So, we have three possible plans to satisfy hunger, but only a subset of these plans are available to each type of character. Only two are suitable for Alma, and one is suitable for Paxton Fettel. This is essentially what we did for FEAR.

STRIPS Planning Example



...but instead of planning ways to satisfy hunger, we were planning ways of eliminating threats. We can satisfy the goal of eliminating a threat by firing a gun at the threat, but the gun needs to be loaded, or we can use a melee attack, but we have to move close enough.

So now I've shown you another way to implement behaviors that we could have already implemented with a Finite State Machine. So what's the point?

Here's the Plan:

- STRIPS Planning Overview
- Planning in F.E.A.R.
- ~~Differences from STRIPS~~
- ~~Squad Behaviors & Communication~~
- Beyond F.E.A.R.

It's easiest to understand the benefits of a planning system by seeing how it helped the development of AI behaviors in FEAR. Before we look at the implementation details, and how our system differs from STRIPS, we'll look at some videos that illustrate how we used the system.

Design Philosophy

Designer's job is:

Create environments that allow AI to showcase their behaviors.

Designer's job is NOT:

Script behavior of individual AI.

GameDevelopers
Conference

The design philosophy at Monolith is that the designer's job is to create interesting spaces for combat, packed with opportunities for the AI to exploit. For example, spaces filled with furniture for cover, glass windows, and multiple entries for flanking.

Designers are not responsible for scripting the behavior of individuals, other than for story elements. This means that the AI need to autonomously use the environment to satisfy their goals.

NOTE CONTRAST WITH PHILOSOPHY OF HALO3 (DISCUSSION AT END?) -CR

Planning Video #1

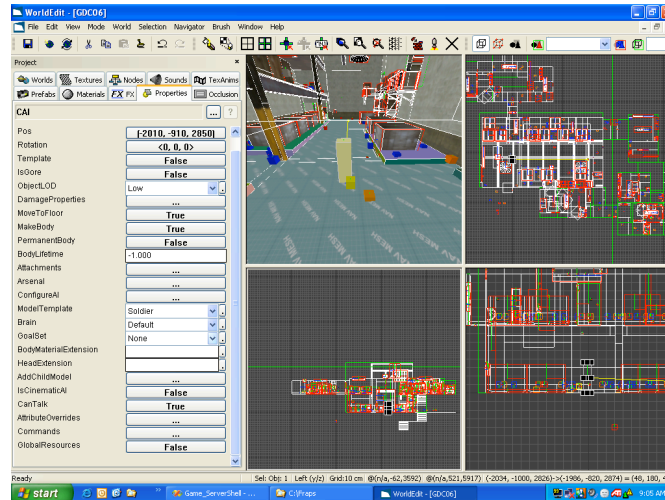


GameDevelopers
Conference

If we simply drop an AI into the world, and let him see the Player, the AI will do.... nothing. This is because we have not yet given him any Goals.

[Click for video]

Planning Video #1

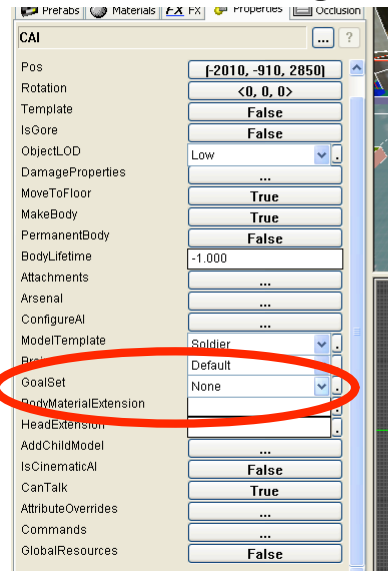


GameDevelopers
Conference

In our world editor, we assign a Goal Set to each AI. These Goals compete for activation, and the AI uses the Planner to try to satisfy the highest priority Goal.

VERY SIMILAR TO HALO3, EXCEPT AS WE SHALL SEE,
DECOMPOSITION HIERARCHIES ARE CREATED ON THE FLY. -CR

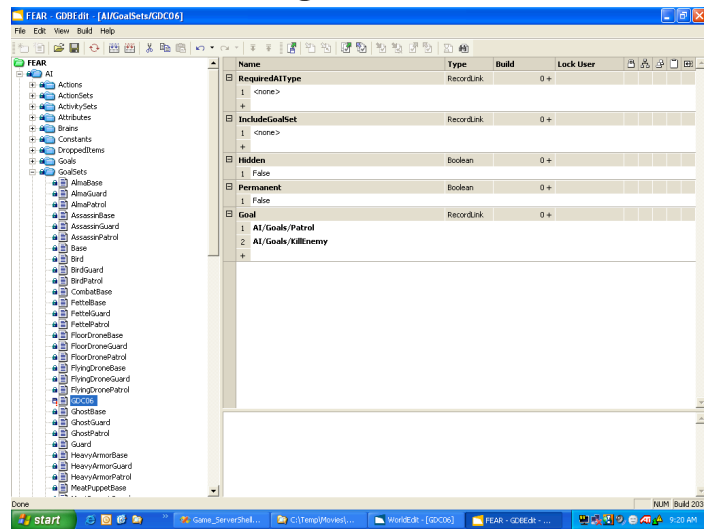
Planning Video #1



GameDevelopers
Conference

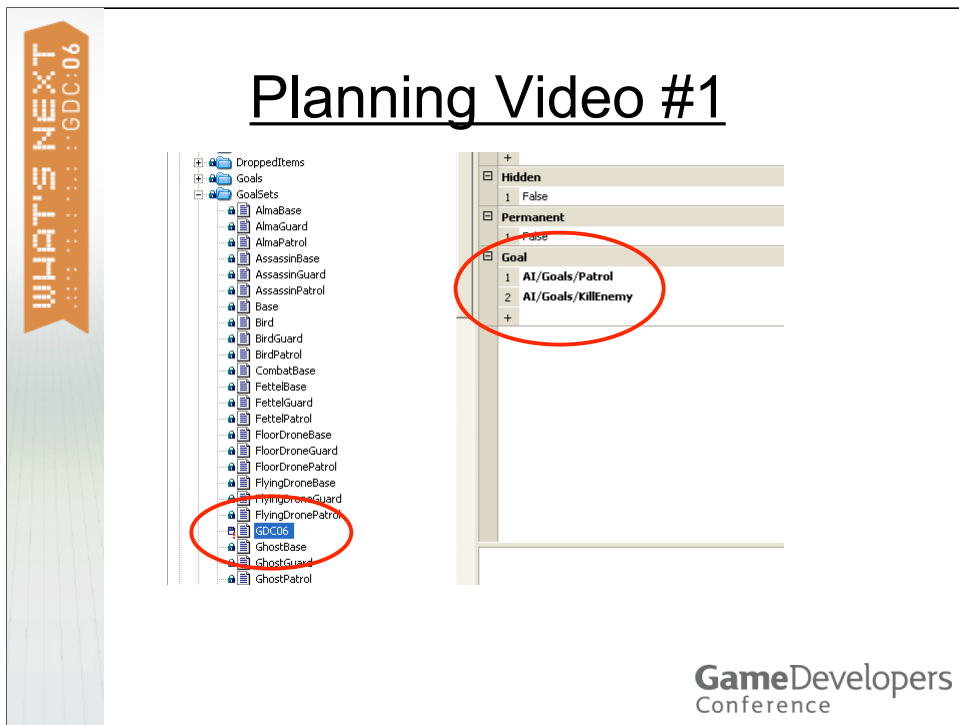
Zooming in to the AI properties, we see that by default a new AI has Goal Set None.

Planning Video #1



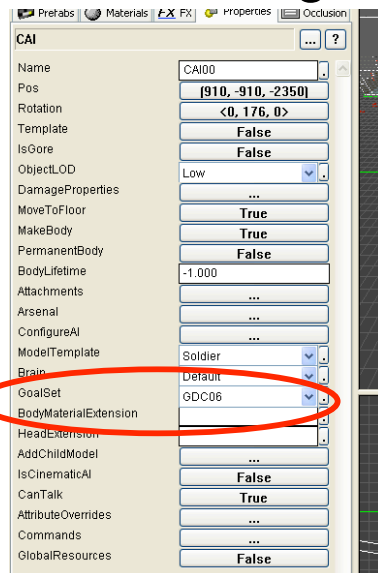
GameDevelopers
Conference

To rectify this, we can go into our Game Database editor and create a new Goal Set, which we'll call GDC06.



Zooming in we see that the new GDC06 Goal Set includes the Goals Patrol and KillEnemy.

Planning Video #1



GameDevelopers
Conference

So now we assign the AI the new GDC06 Goal Set in the world editor.

Planning Video #2



GameDevelopers
Conference

Now, when the soldier Patrols and when he sees us, his KillEnemy Goal takes priority, and he starts to attack.

JUST LIKE IN HALO3. -CR

[Click for video]

Planning Video #3



GameDevelopers
Conference

Now let's look at an Assassin in the exact same level, with the exact same Goal Set.

He's satisfying the Goals in different ways than the soldier. He patrols by running cloaked and climbing onto the wall, and attacks by lunging at the Player.

[Click for video]

Planning Video #4



GameDevelopers
Conference

Finally, let's look at a Rat with the exact same Goal Set in the same level.

The Rat patrols, and never attacks at all. What we're seeing here is three different characters with an identical Goal Set, but the Goal Set is only half of the picture. These characters each have a different Action Set, providing the Actions that the character may sequence to satisfy his Goals.

[Click for video]

Planning Video #4

Soldier

Action
1 AI/Actions/Attack
2 AI/Actions/AttackCrouch
3 AI/Actions/SuppressionFire
4 AI/Actions/SuppressionFireFromCover
5 AI/Actions/FlushOutWithGrenade
6 AI/Actions/AttackFromCover
7 AI/Actions/BlindFireFromCover
8 AI/Actions/AttackGrenadeFromCover
9 AI/Actions/AttackFromView
10 AI/Actions/DrawWeapon
11 AI/Actions/HolsterWeapon
12 AI/Actions/ReloadCrouch
13 AI/Actions/ReloadCovered
14 AI/Actions/InspectDisturbance
15 AI/Actions/LookAtDisturbance
16 AI/Actions/SurveyArea
17 AI/Actions/DodgeRoll
18 AI/Actions/DodgeShuffle
19 AI/Actions/DodgeCovered
20 AI/Actions/Uncover
21 AI/Actions/AttackMelee

Assassin

Action
1 AI/Actions/Attack
2 AI/Actions/InspectDisturbance
3 AI/Actions/LookAtDisturbance
4 AI/Actions/SurveyArea
5 AI/Actions/AttackMeleeUnloaked
6 AI/Actions/TraverseBlockedDoor
7 AI/Actions/UseSmartObjectNodeMounted
8 AI/Actions/MountNodeUnloaked
9 AI/Actions/DismountNodeUnloaked
10 AI/Actions/TraverseLinkUnloaked
11 AI/Actions/AttackFromAmbush
12 AI/Actions/DodgeRollParanoid
13 AI/Actions/AttackLungeUnloaked
14 AI/Actions/LopeToTargetUnloaked

Rat

Action
1 AI/Actions/Animate
2 AI/Actions/Idle
3 AI/Actions/GotoNode
4 AI/Actions/UseSmartObjectNode

GameDevelopers
Conference

So what we're seeing is that the Soldier, Assassin, and Rat have the exact same Goal Set, but very different Action Sets.

The Soldier's Action Set includes Actions for firing weapons, while the Assassin's Action Set has Lunges and Melee attacks. The Rat has no means of attacking at all, so he fails to formulate a valid plan to satisfy the KillEnemy Goal, and he falls back to the lower priority Patrol Goal.

Benefits of Planning

- 1. Decoupling Goals & Actions**
2. Layering Behaviors
3. Dynamic Problem Solving

So, the videos illustrate the first of three benefits of a planning system. The first benefit is the ability to decouple Goals and Actions, to allow different types of characters to satisfy Goals in different ways.

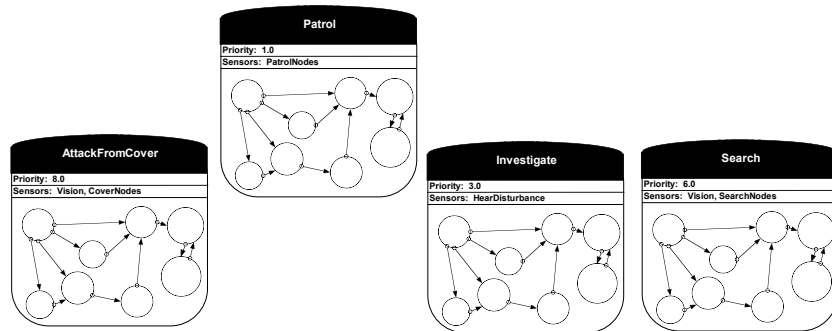
Decoupling Goals & Actions



GameDevelopers
Conference

In our previous generation of AI systems, we ran into the classic AI problem of the Mime and the Mutant. No One Lives Forever 2 has mimes, and Tron2.0 had mutant virus characters.

Decoupling Goals & Actions



GameDevelopers
Conference

Our previous AI system was used to develop both NOLF2 and TRON 2.0. Our AI system consisted of Goals that competed for activation, just like they do in FEAR, in the old, system these Goals each contained an embedded Finite State Machine. There was no way to separate the Goal from the plan used to achieve that Goal.

So if we wanted any variation between the behavior of a mime and the behavior of a mutant, or other character types, we had to add branches to the embedded state machines. Over the course of two years of development, these state machines become overly complex, bloated, and unmanageable.

Decoupling Goals & Actions



GameDevelopers
Conference

For example, we had out of shape policemen in NOLF2 who needed to stop and catch their breath every few seconds while chasing. This required a branch in the state machine for the Chase Goal to check if the character was out of breath, when only one type of character ever exhibits this behavior.

With a planning system, we can give each character their own Action Set, and in this case only the policemen would have the Action for catching their breath. This unique behavior would not add any unneeded complexity to other characters.

Decoupling Goals & Actions

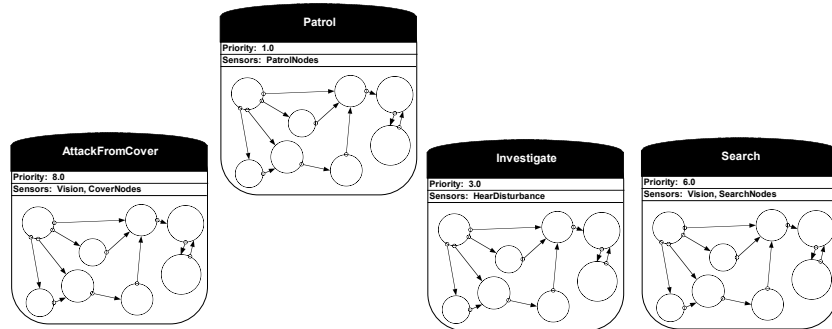


GameDevelopers
Conference

The modular nature of Goals and Actions benefited us on FEAR, when we decided to add a new enemy type late in development. We added flying Drones with a minimal amount of effort by combining Goals and Actions from characters we already had.

By combining the Ghosts Actions for flying movement with the Soldier's Actions for firing weapons and using tactical positions, we were able to create a unique new enemy type in a minimal amount of time, with very little new code.

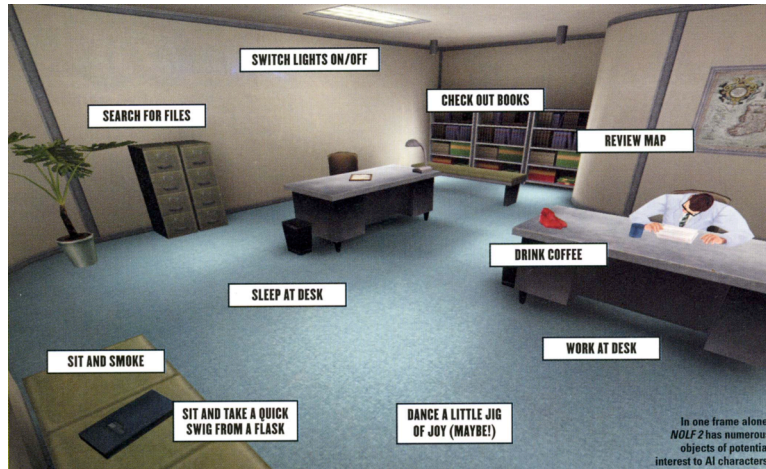
Decoupling Goals & Actions



GameDevelopers
Conference

And there's another good reason for decoupling Goals and Actions. In our previous system, Goals were self-contained black boxes, and did not share any information with each other. This can be problematic.

Decoupling Goals & Actions

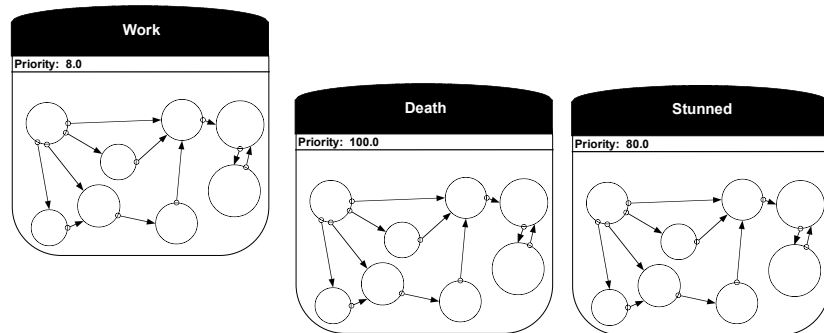


GameDevelopers
Conference

This is very similar to a problem we had in NOLF2. We had all of these objects that a character could interact with. For example someone could sit down at a desk and do some work. If you shoot or stun someone at a desk, you would expect him to slump over.

Instead what we were seeing is that the AI would finish his work, stand up, push in his chair, and then pass out and fall to the floor.

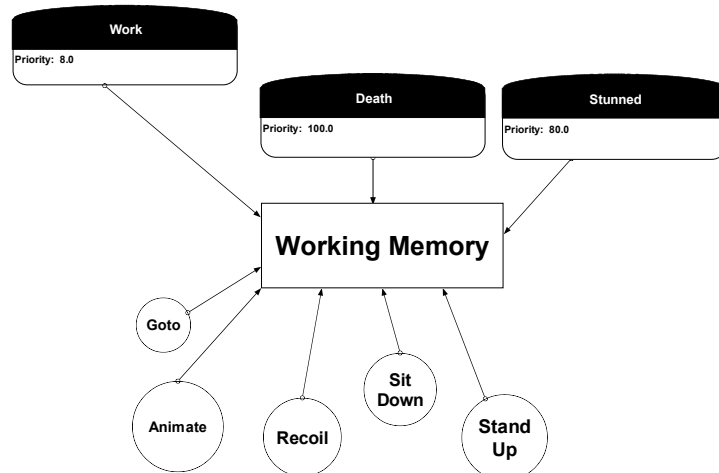
Decoupling Goals & Actions



GameDevelopers
Conference

The problem was that only the Work Goal knew that the AI was in a sitting posture, interacting with the desk. This was because there was no information sharing between Goals, so each Goal had to exit cleanly, and get the AI back into some default state where he could cleanly enter the next Goal.

Decoupling Goals & Actions



GameDevelopers
Conference

Decoupling the Goals and Actions forces them to share information through some external working space. Now all Goals and Actions have access to information including whether the AI is sitting or standing, and interacting with a desk or some other object. We can take this knowledge into account when we formulate a plan to satisfy the Death Goal, and slump over the desk as expected.

THIS IS REALLY OF FEATURE OF THE PRIORITIZED GOAL ARCHITECTURE, NOT PLANNING SPECIFICALLY. -CR

Benefits of Planning

1. Decoupling Goals & Actions
2. Layering Behaviors
3. Dynamic Problem Solving

The second benefit of the planning approach is facilitating layering of behaviors. We were able to evolve complex combat behavior over three years of development of FEAR by layering simple atomic actions.

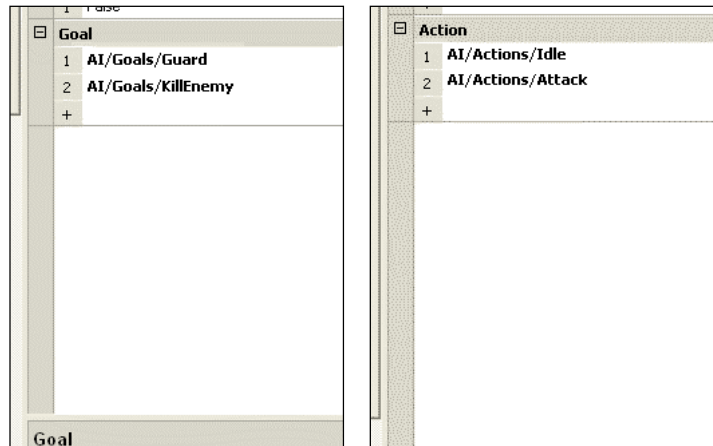
Layering Behaviors



GameDevelopers
Conference

You can think of the basic combat behavior in FEAR as a seven layer dip. We get deep, complex tactical behavior by layering a variety of simple atomic behaviors. We wanted the FEAR AI to do everything for a reason. This is in contrast to the NOLF2 AI, which would run to valid cover and then pop in and out randomly like a shooting gallery. FEAR AI always try to stay covered, never leave cover unless threatened and other cover is available, and fire from cover to the best of their ability.

Layering Behaviors



We start with the basics; the beans. AI fire their weapons when they detect the Player.

The things to note here are that we are simply adding goals and actions to our lists (goal sets and action sets). We do not need to specify anywhere which actions are associated with which goals, or which actions can chain with which other actions. The AI can figure this out for themselves based on preconditions and effects defined within the actions. Also, while it appears on this slide that there is some kind of one-to-on correspondence between goals and action, this is actually not the case as we'll see in future layers.

Layering Behaviors



GameDevelopers
Conference

We start with the basics; the beans. AI fire their weapons when they detect the Player.

Lighting effects have been disabled for this series of videos so that it's easier to focus on the AI behavior.

[Click for video]

Layering Behaviors

Goal	
1	AI/Goals/Guard
2	AI/Goals/KillEnemy
3	AI/Goals/Dodge
+	

Action	
1	AI/Actions/Idle
2	AI/Actions/Attack
3	AI/Actions/DodgeShuffle
4	AI/Actions/DodgeRoll
+	



We want the AI to value his life, so we add the Guacamole. The AI dodges when a gun is aimed at him. The new additions are highlighted in blue. We add a new Dodge goal, and two possible actions for satisfying that goal.

Layering Behaviors

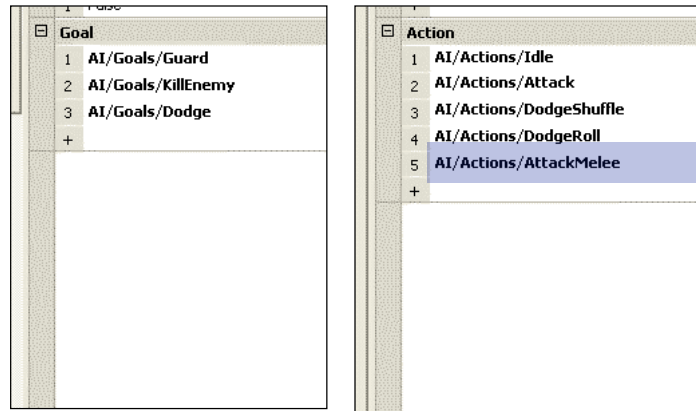


GameDevelopers
Conference

We want the AI to value his life, so we add the Guacamole. The AI dodges when a gun is aimed at him.

[Click for video]

Layering Behaviors



Next we add the Sour Cream. When the Player gets close enough, use melee attacks instead of wasting bullets. We do not need to add any additional goals here. We only add a new action for melee attacks as an alternative way of satisfying the existing KillEnemy goal.

Layering Behaviors

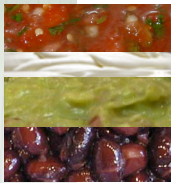
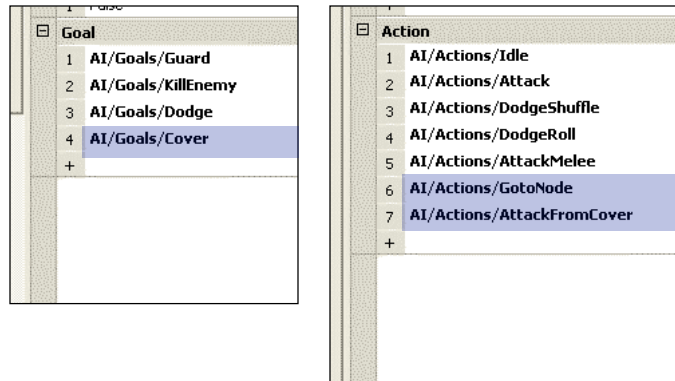


GameDevelopers
Conference

Next we add the Sour Cream. When the Player gets close enough, use melee attacks instead of wasting bullets.

[Click for video]

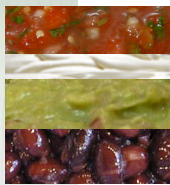
Layering Behaviors



If AI really value their lives, they won't just dodge, they'll take cover.
This is the salsa.

We add the Cover goal, which makes an AI who is not currently in cover get to cover. He gets there with the GotoNode action, and then attacks in a context-sensitive manner with the AttackFromCover action.

Layering Behaviors

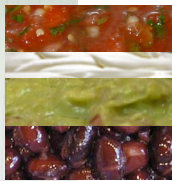
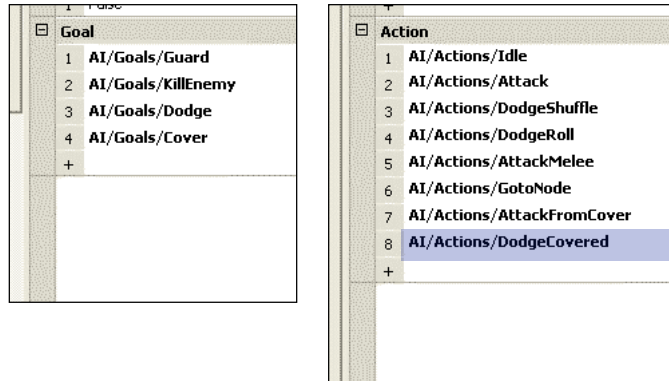


GameDevelopers
Conference

If AI really value their lives, they won't just dodge, they'll take cover.
This is the salsa.

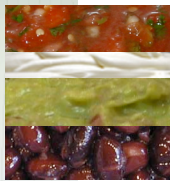
[Click for video]

Layering Behaviors



We previously added the ability to dodge, but have not provided any way to dodge in the context of taking cover. We can add another context-sensitive Dodge Action to handle this, still as part of the salsa layer.

Layering Behaviors



GameDevelopers
Conference

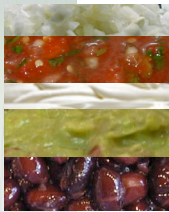
We previously added the ability to dodge, but have not provided any way to dodge in the context of taking cover. We can add another Dodge Action to handle this.

[Click for video]

Layering Behaviors

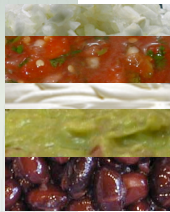
Goal	
1	AI/Goals/Guard
2	AI/Goals/KillEnemy
3	AI/Goals/Dodge
4	AI/Goals/Cover
+	

Action	
1	AI/Actions/Idle
2	AI/Actions/Attack
3	AI/Actions/DodgeShuffle
4	AI/Actions/DodgeRoll
5	AI/Actions/AttackMelee
6	AI/Actions/GotoNode
7	AI/Actions/AttackFromCover
8	AI/Actions/DodgeCovered
9	AI/Actions/BlindFireFromCover
+	



Dodging is not always enough, so we add the Onions; blind firing. If the AI gets shot in cover, he blind fires for a while to make himself harder to hit. This is an alternative way to satisfy the kill enemy goal.

Layering Behaviors



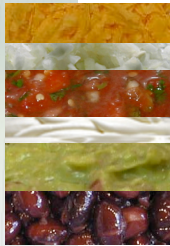
GameDevelopers
Conference

Dodging is not always enough, so we add the Onions; blind firing. If the AI gets shot in cover, he blind fires for a while to make himself harder to hit.

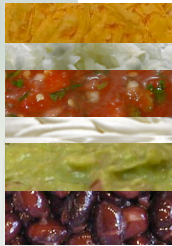
[Click for video]

Layering Behaviors

Goal	Action
1 AI/Goals/Guard	1 AI/Actions/Idle
2 AI/Goals/KillEnemy	2 AI/Actions/Attack
3 AI/Goals/Dodge	3 AI/Actions/DodgeShuffle
4 AI/Goals/Cover	4 AI/Actions/DodgeRoll
5 AI/Goals/Ambush	5 AI/Actions/AttackMelee
+	6 AI/Actions/GotoNode
	7 AI/Actions/AttackFromCover
	8 AI/Actions/DodgeCovered
	9 AI/Actions/BlindFireFromCover
	+



The cheese is where things really get exciting. We give the AI the ability to reposition when his current cover position is invalidated. The Ambush action basically lets the AI hide somewhere when cover is not working out. It just chooses a different type of node for the AI to go to, so it can be satisfied with our existing GotoNode action.



Layering Behaviors



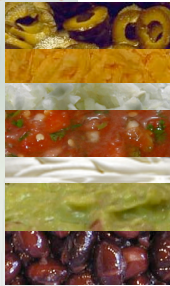
GameDevelopers
Conference

The cheese is where things really get exciting. We give the AI the ability to reposition when his current cover position is invalidated.

Here we see the effects of level designers adding additional cover nodes, so that the AI moves around behind the box, and the Ambush goal, which gets the AI to leave cover when he's taking fire, and try to hide until cover seems valid again.

[Click for video]

Layering Behaviors



The final layer is the olives, which really bring out the flavor. For this layer, we add dialogue that lets the Player know what the AI is thinking, and allows the AI to communicate with squad members. We'll discuss this a little later.

Layering Behaviors

Kill Enemy

Attack Ranged

GameDevelopers
Conference

So the point I'm trying to make is that with a planning system, we can just toss in Goals and Actions...

Layering Behaviors

Kill Enemy

Attack Melee

Attack Ranged

GameDevelopers
Conference

Layering Behaviors

Kill Enemy

Dodge

Dodge Roll

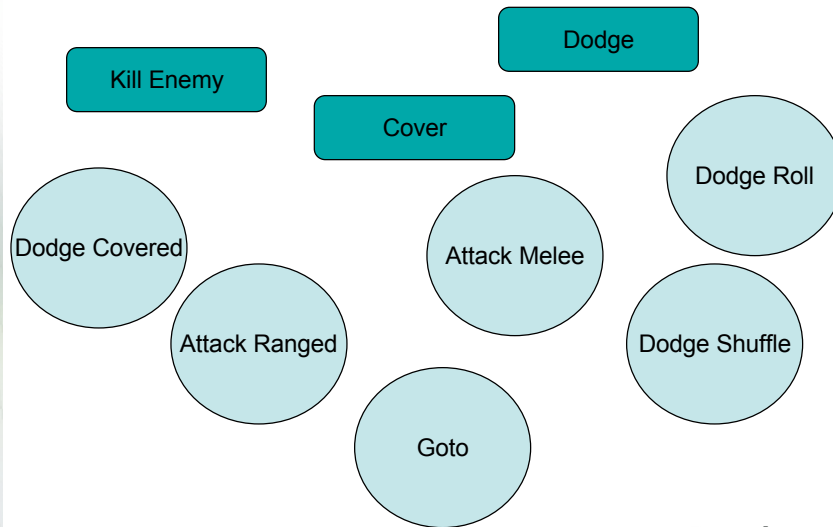
Attack Melee

Attack Ranged

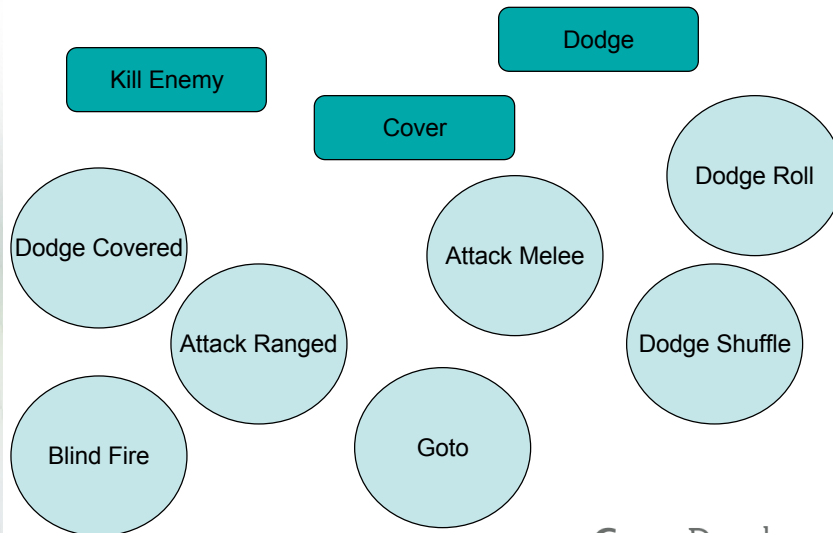
Dodge Shuffle

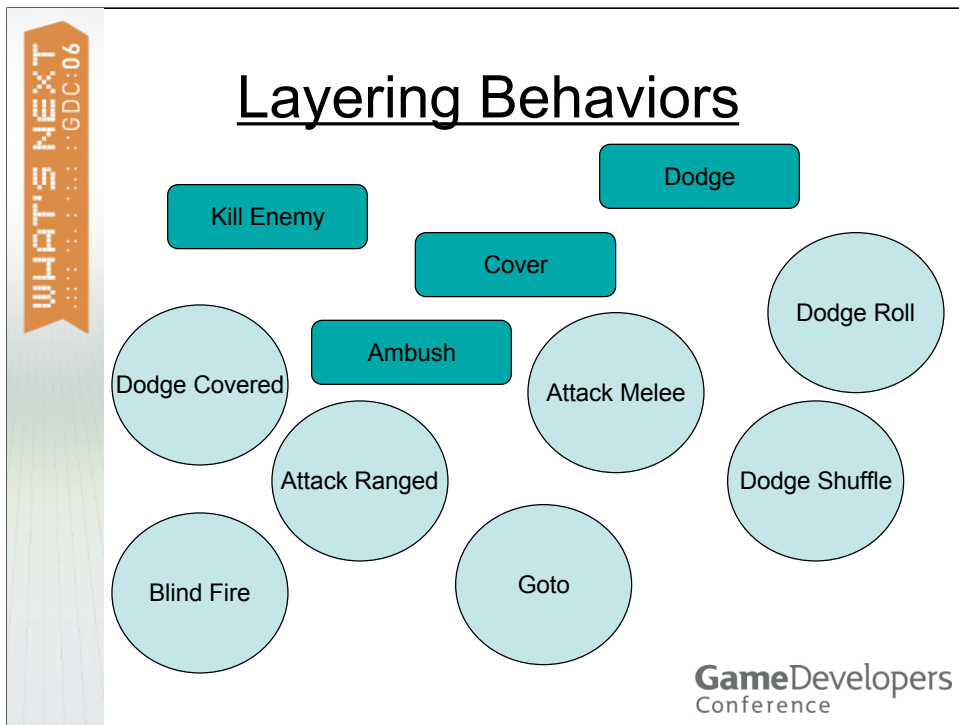
GameDevelopers
Conference

Layering Behaviors



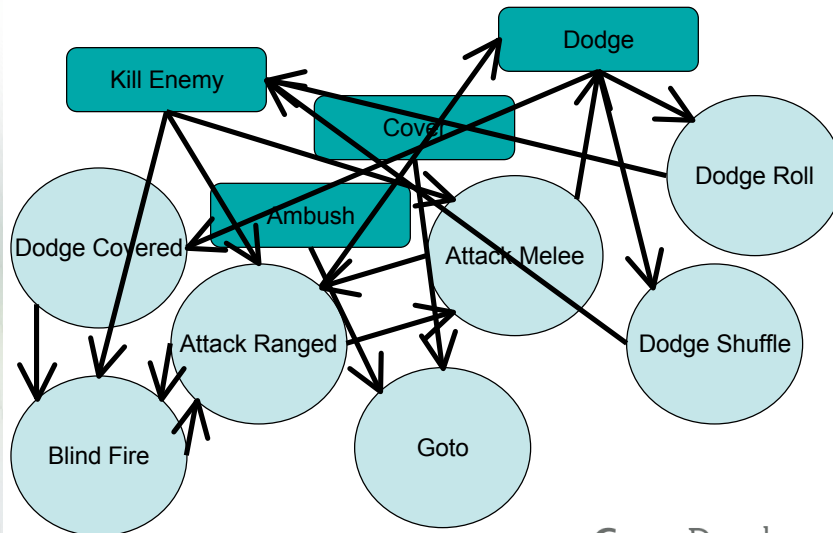
Layering Behaviors





...But we never have to manually specify the transitions. We never have to specify which actions are associated with which goals, which actions can transition to which other actions, or which goals can transition to other goals.

Layering Behaviors



GameDevelopers
Conference

...But we never have to do this. This is where things in an FSM get really hairy. In the planning system, the AI figures the dependencies out at run-time based on the Goal state and the preconditions and effects of Actions.

Layering Behaviors



GameDevelopers
Conference

Late in development of NOLF2, we added the requirement that AI would turn on lights whenever entering a dark room. In our old system, this required us to revisit the state machine inside every goal and figure out how to insert this new state. This was both a headache, and a risky thing to do so late in development. In the FEAR planning system, adding this behavior would have been much easier, as we could have just added a TurnOnLights action, and added a LightsOn precondition to the Goto action. This would affect every Goal that was satisfied by using the Goto action.

Benefits of Planning

1. Decoupling Goals & Actions
2. Layering Behaviors
3. **Dynamic Problem Solving**

Finally, the third benefit of a planning system is the dynamic problem solving ability that replanning gives the AI.

Dynamic Problem Solving



GameDevelopers
Conference

Here we see a patrolling AI who walks in the door, sees the player and starts firing.

[Click for video]

Dynamic Problem Solving



GameDevelopers
Conference

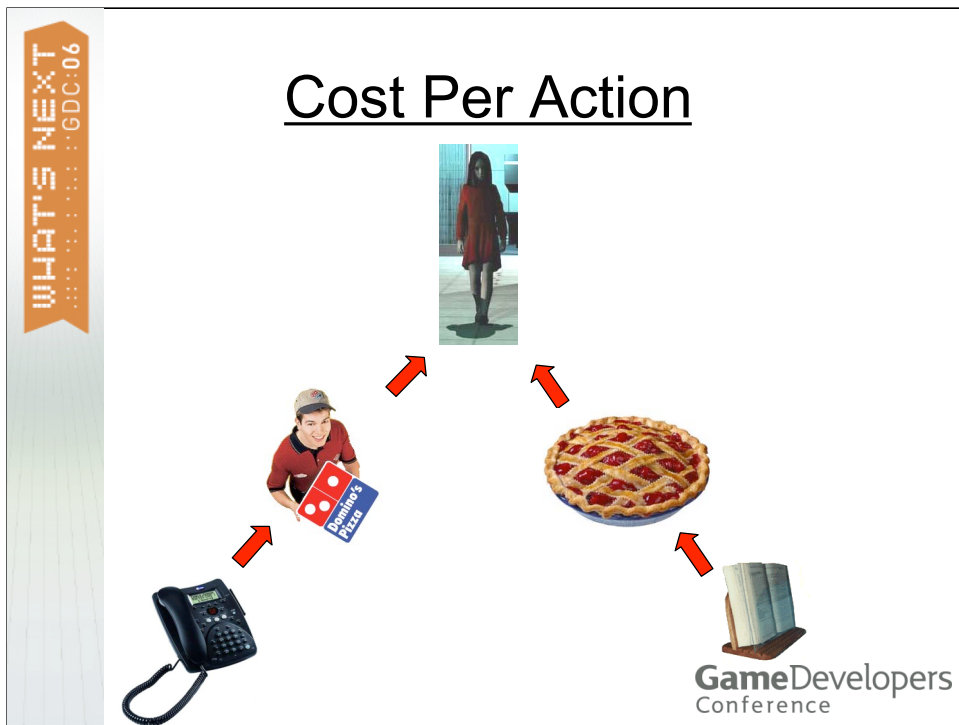
If we run this again, but this time physically hold the door shut with our body, we see the AI try to open the door and fail. He then replans and decides to kick the door. When this fails, he replans again and decides to dive through the window and ends up close enough to use a melee attack! We can watch the AI in the security camera.

[Click for video]

Planning Algorithm

[Slide added by C. Rich]

GameDevelopers
Conference

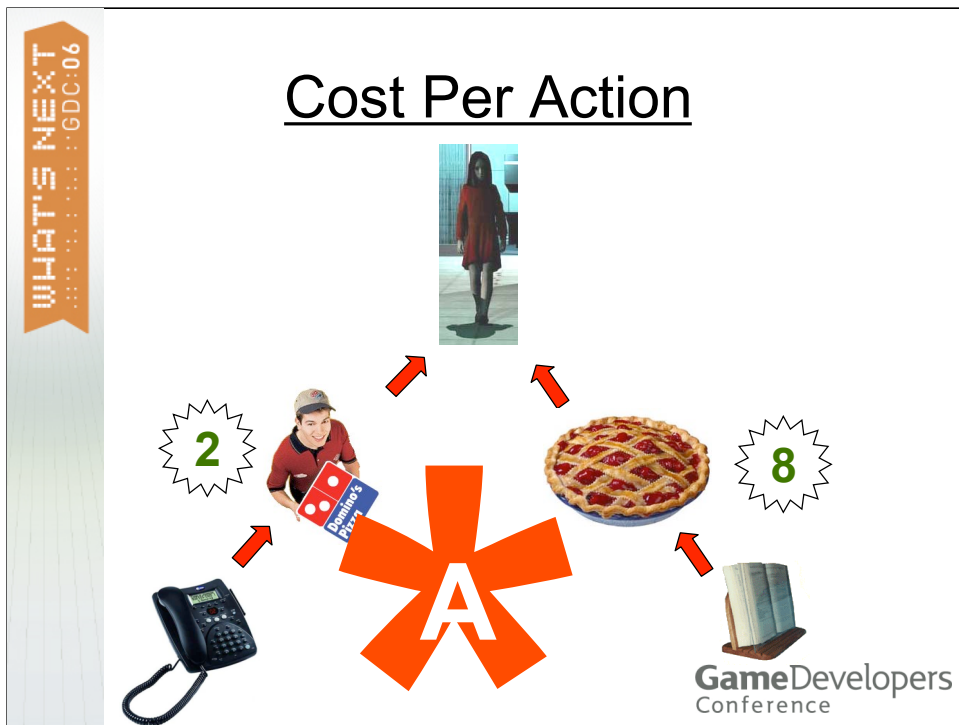


Remember we said earlier that if Alma has both the phone number and the recipe, either plan is valid to satisfy her hunger.

Cost Per Action



If we add a cost per action, we can force Alma to prefer one action over another. If she cannot satisfy the preconditions of ordering pizza, she can fall back to baking a pie.



This is where our old friend A^* comes in! Now that we have a cost metric, we can use this cost to guide our A^* search for the lowest cost sequence of actions to satisfy some goal.

Cost Per Action



GameDevelopers
Conference

Normally we think of A* as a means of finding a navigational path, and use it in this way in FEAR too, to find a path through the navigational mesh. But in fact, A* is really a general search algorithm.

Cost Per Action

A*	Navigation	Planning
Nodes:	NavMesh Polys	World States
Edges:	NavMesh Poly Edges	Actions
Goal:	NavMesh Poly	World State

GameDevelopers
Conference

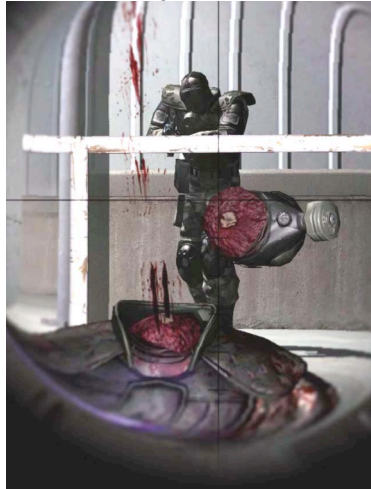
A* can be used to search for the shortest path through any graph of nodes connects by edges. In the case of navigation, it is intuitive to think of navigational mesh polygons as nodes, and the edges of the polygons as edges in the graph that connect that connect one node to another. In the case of planning, the nodes are states of the world, and we are searching to find a path to the goal state. The edges connecting different states of the world are the actions that lead the state of the world to change from one to another.

Here's the Plan:

- STRIPS Planning Overview
- Planning in F.E.A.R.
- ~~Differences from STRIPS~~
- ~~Squad Behaviors & Communication~~
- Beyond F.E.A.R.

Dialogue Integration

“What’s your status?”



GameDevelopers
Conference

For FEAR, the combat dialogue system was completely separate from the action planning system. We manually hooked dialogue lines into the code in various places. It took a lot of trial and error to get AI saying the right things at the right times. For example, in this screenshot where the AI’s head has completely disconnected, there is really no reason for his ally to ask him “What’s your status?” It’s pretty obvious what his status is. Situations like this are not always obvious when you are looking at C++ code trying to figure out where to insert dialogue lines.

If we want an order of magnitude more dialogue, we need a system managing dialogue that is as well structured and well informed as the system used to plan actions.

Unified Planning for Actions and Speech

OpenDoor -or- “Open the door!”

Precondition:

door is **closed**

Effect:

door is **open**



GameDevelopers
Conference

The complication comes when AI can accomplish the same effect by either acting or speaking.

For example, a soldier may decide to open the door himself, or order a squad member to open the door. In order to decide whether to speak to someone else or take a physical action yourself, we need to consider a number of factors when weighting the costs of these options. We need to consider the physical situation, like who is closer to the door, and the social situation, like who is the higher ranking member of the squad. A soldier does not usually shout orders to a superior.

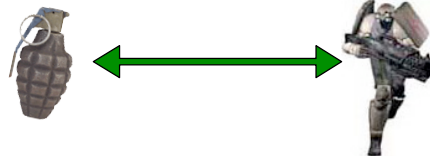
Unified Planning for Actions and Speech

“Look out! Grenade!”

Precondition:



Effect:

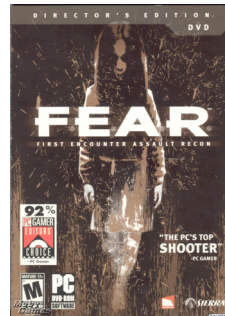
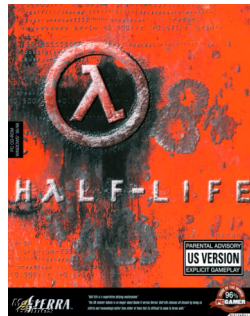


GameDevelopers
Conference

We are looking at how we can seamlessly integrate dialogue into the action planning system. We can think of dialogue lines serving the same purpose as actions in a plan, and we should be able to formalize the lines in the same way we did actions. For anything the AI says, there are preconditions for why the AI should say it, and effects that the AI expects saying the line to have on the world.

For example, if there is a grenade coming near an AI's ally, the AI expects that shouting “Look out! Grenade!” will have the effect of his ally getting some distance from the grenade.

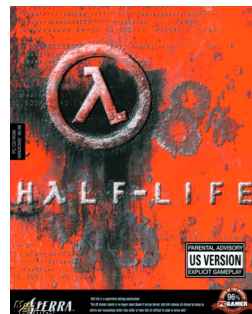
“... just like the marines in Half-Life 1”



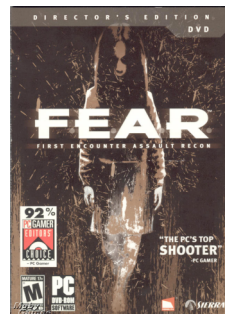
GameDevelopers
Conference

When FEAR shipped, it was great to see that the AI was well received. Many people commented that the soldier AI reminded them of the Marines in Half Life 1.

“... just like the marines in Half-Life 1”



1998



2005

GameDevelopers
Conference

Half Life 1 shipped in 1998, and FEAR shipped in 2005. It seems that we haven't made much progress in seven years, and what's worse, people seem happy about this!!

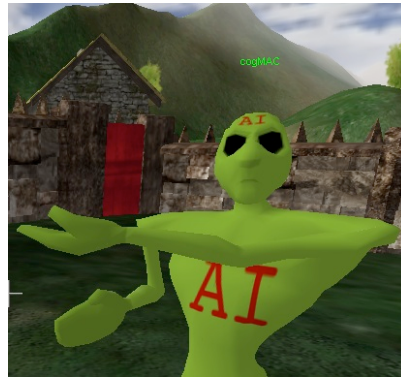
There has to be more we can do with game AI. My opinion is that we need an order of magnitude more interaction, cooperation, and communication between AI, and between AI and the player. To achieve this, we need to investigate new techniques.



If we look 20 years in the past, 20 years ago we were playing Super Mario Bros on the NES. Game have come a long way in 20 years, and today every game has incredible graphics. In the next 20 years, AI is going to be what differentiates one game from another.

MIT Media Lab: Cognitive Machines Group

<http://www.media.mit.edu/cogmac>



GameDevelopers
Conference

In the Cognitive Machines group we use robots and computer games as platforms for researching the use and understanding of language. Our goal is to create robots and characters that can use language to communicate the way people do.

At the Media Lab, we are supposed to be looking 20 years into the future.