



Goal-Driven Agent Behavior

Artificial Intelligence for Interactive Media and Games

Professor Charles Rich
 Computer Science Department
 rich@wpi.edu

[Based on Buckland, Chapter 9 and lecture by Robin Burke]

CS/IMGD 4100 (B 16)

1

Mon, Nov 21		Natural Language and Dialog	
Tue, Nov 22		Natural Language and Dialog	
Wed, Nov 23		Thanksgiving Break	
Mon, Nov 28	Chapter 9	Goal-Driven Behavior	
Tues, Nov 29	Chapter 9	Goal-Driven Behavior	
Weds, Nov 30			9- Steal Health [5%]
Thu, Dec 1	Chapter 9	Goal-Driven Behavior	
Fri, Dec 2		Brainstorming: Raven Bot Strategy	
Sun, Dec 4			10 - Bot Design [3%]
Mon, Dec 5	Chapter 10	Fuzzy Logic	
Tue, Dec 6		Research: Narrative Planning / Course Eval	
Wed, Dec 7			11 - AI Middleware [10%]
Thu, Dec 8		Presentations: AI Middleware	
Fri, Dec 9		Presentations: AI Middleware	
Sun, Dec 11			(Due 6pm!) 12 - Tournament Bot [10%]
Mon, Dec 12		Raven Tournament (GH 012 !)	
Tue, Dec 13		Guest Speaker: Damian Isla	
Thu, Dec 15		Final Exam [30%]	

CS/IMGD 4100 (B 16)

2

Goals and Planning in AI

- What is a **goal**?
 - intuitive and cognitively motivated concept
 - an abstraction (mental state) that guides behavior
 - often formalized as a **partial** description of a **desired** state of the world

Goal (Mental State)	Desired World State
go to the cinema	I am at the cinema
attack (given bot)	I am firing on the bot

Goals and Planning in AI

- Desired world **state**
 - is this the same notion of “state” as in state machines approach to AI?
 - no, states in FSM are part of mental states of agent
 - states in FSM more analogous to (can be used like) goals
 - FSM states have some similar implementation features similar to goals (see later)

Goals and Planning in AI

- Degrees of formalization of **goal**

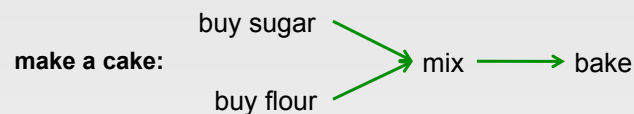
1. just the *name* of the goal, e.g., GoToCinema
2. code/procedure to **test** if world is in desired state (goal succeeded) or not (goal failed), e.g., test location
3. *declarative*/logical representation (very difficult in general)



Goals and Planning in AI

- What is a *plan* ?

1. a *sequence* of *actions* to achieve a *goal*, e.g.,
leave the house: [walk to closet, open closet door, remove coat from coat hook, ...]
 – *sequence*: totally ordered
 – *action*: directly executable by agent (changes world state)
 – *goal*: desired world state
2. a *partially ordered set* of actions, e.g.,



Goals and Planning in AI

- What is *planning* ?
 - given a **goal**
 - construct a **plan** to change current (or given) **world state** into desired world state
 - usually involves search
 - in space of possible plans
 - multiple solutions possible
 - plan may fail
 - if world changes due to other factors than own actions (e.g., other agents)
 - poor world model underlying plan
 - **example:** path planning
 - given current and desired location
 - find sequence of movements from here to there

Goals and Planning in AI

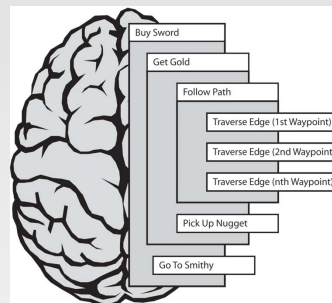
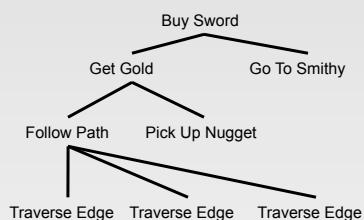
- What is *re-planning* ?
 - when the current plan for a goal fails
 - you executed all the actions in the plan
 - but the world is not in the desired state ☹️
 - > assumes you have some test for success/failure
 - or some planned action is not executable (blocked)
 - > e.g., cannot open door (because locked)
 - > assumes actions have some test for blocked
 - > could be a faulty plan or world changed unexpectedly
 - need to construct **another plan** for same goal
 - starting with current world state
 - and maybe other constraints based on current failure

Goals and Planning in AI

- Alternative to searching for plans ?
 - search can be expensive and error-prone
 - predefine specific plans for particular goals
 - quickly look up plan for goal
 - may be more than one choice (need to decide)
 - can be “manual” or cached from previous (e.g., offline) searches
 - already “knowing” a lot of plans for commonly occurring goals in a domain makes you an “expert”

Hierarchical Plans

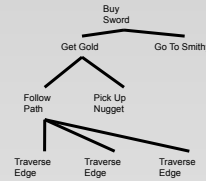
- tree of goals and **actions** (aka “atomic” or “primitive” goals)
- child/parent relationship called “subgoal” or “step”
- actions appear only at leaves
- all internal nodes are (“composite” / “abstract” / “nonprimitive”) goals
- subgoals at each level may be totally or partially ordered
- decomposition can be via planning (search) or predefined



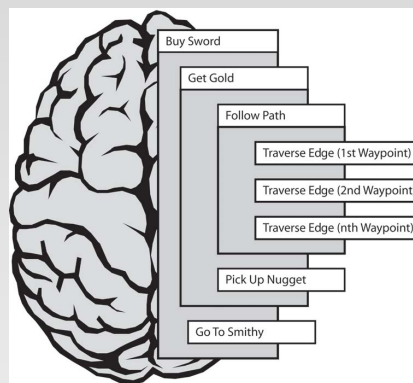
Hierarchical Plans

- when *fully* expanded (“decomposed”)
 - all leaves (“fringe” of tree) are actions
 - fringe constitutes a sequential or partially ordered plan

- often expanded *incrementally* (“lazy”)
 - some leaf nodes are not actions
 - not “directly executable” by agent
 - what is directly executable depends on level of modeling
 - not efficient or effective to expand goal nodes before they are “live”, because
 - will have more information later
 - e.g., to choose between alternative decompositions



Hierarchical Plans



Hierarchical Plans

▪ Hierarchical Task Networks (HTN's)

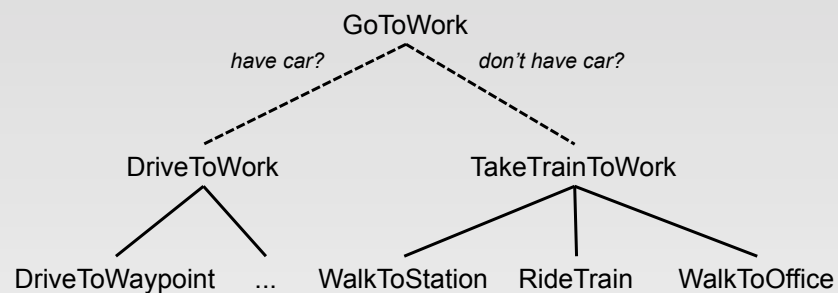
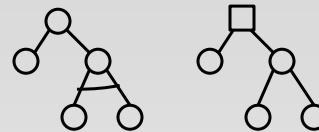
- AI term for predefined library of hierarchical plans
- the library usually implemented using a declarative representation

– e.g., ANSI/CEA-2018 (<http://ce.org/cea-2018>)

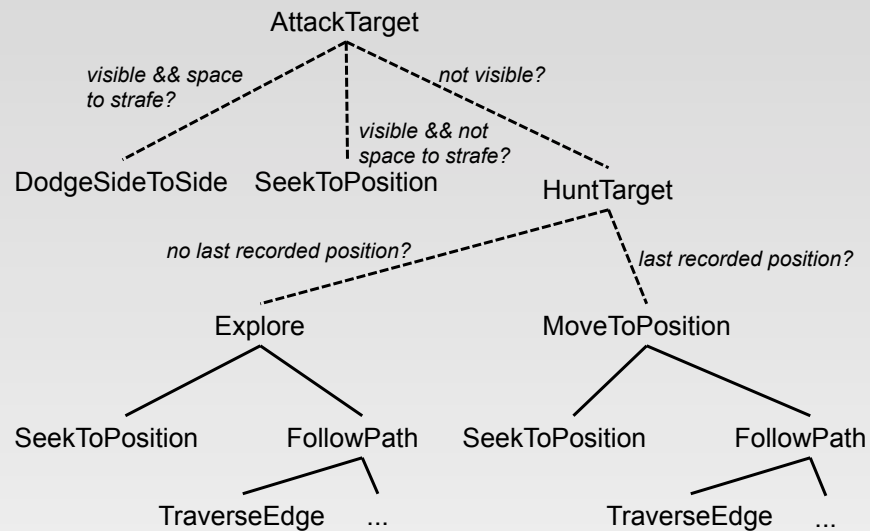
```
<task name="Buy Sword">
  <subtask task="Get Gold" .../>
  ...
</task>
```

Hierarchical Plans

▪ "And/Or Tree"



HTN in Raven



Goal/Behavior Trees

- What Buckland describes in Chapter 9 is essentially a
 - procedural implementation of
 - hierarchical task networks (and/or trees)
 - with totally ordered subgoals
- This technique is becoming popular in AI game dev community under the title of “behavior trees”
 - see <http://aigamedev.com/open/article/behavior-trees-part1>

Goal/Behavior Tree Execution Issues

- choosing a **toplevel** goal (goal arbitration)
- choosing among **alternative decompositions** of a goal (into subgoals and actions)
- **sequencing** of subgoals/actions
- **monitoring** of goal completion/failure
- **re-planning** after failure

Goal Tree Implementation

- *Design Pattern: Same base class used both for composite and atomic goals (actions)*
- Atomic goals (4) currently in Raven
 - Wander, SeekToPosition, TraverseEdge, DodgeSideToSide
- Composite goals (7) currently in Raven
 - *Think*: special root node (discuss later)
 - *Toplevel goals*: GetItem(*), AttackTarget, Explore
 - *Intermediate goals*: MoveToPosition, FollowPath, HuntTarget
- **Demo**

Key Properties of a Goal

- **Status** (enum)
 - *inactive* – waiting (e.g., due to predecessors not completed); default initial status
 - *active* – can be processed on next update
 - *completed* – will be removed on next update
 - *failed* – will be re-planned or removed on next update
- **Subgoals** (std::list<Goal>)
 - for composite goals only
 - in order of required execution

Key Methods of a Goal

- Activate
- Process
- Terminate
- HandleMessage

Goal::Activate

- Analogous to State::Enter
- contains initialization code (see Terminate)
- for atomic steering goals (e.g., Wander), turns on steering behavior
- for composite goals, **chooses subgoals** (decomposition method)
- may be called multiple times for **re-planning**
- set status to 'active'
 - unless cannot decompose (e.g., target no longer exists)
 - then status set to 'completed', so goal removed

Goal::Process

- analogous to State::Execute
- always starts with ActivateIfInactive()
 - gives Activate method a chance to re-plan
- for composite goals calls ProcessSubgoals
- returns goal status

Goal::Terminate

- analogous to State::Exit
- cleanup code before goal destroyed
- for atomic steering goals, turns off steering behavior

Goal::HandleMessage

- analogous to State::OnMessage
- for composite goals, check if handled by first subgoal; otherwise handle self
- messages only used in goal code for asynchronous (cf. time slicing) notification from path finder
 - Msg_PathReady
 - Msg_NoPathAvailablehandled by MoveToPosition and Explore

Code Walk

- Start at AbstRaven_Bot “brain”
- Goal_Composite::ProcessSubgoals
- Atomic Goals
 - Wander
 - TraverseEdge
- Composite Goals
 - FollowPath (TraverseEdge subgoals)
 - MoveToPosition (FollowPath subgoal)
 - AttackTarget

Goal Arbitration

- Six toplevel (“strategy”) goals
 - Explore
 - AttackTarget
 - GetItem
 - health
 - rocket launcher
 - shotgun
 - railgun
- How does bot decide which to pursue at any given moment? (Only one at a time)

Goal Evaluators

- List of evaluators stored in “brain” (Goal_Think)
 - One for each toplevel goal
- **CalculateDesirability** method
 - returns value between 0 and 1 (inclusive)
 - evaluated on every update for each goal
 - allows “opportunistic” behavior
 - highest value becomes current goal
 - replaces current goal if different, even if not completed!
 - uses “helper functions”
 - static methods in Raven_Feature
 - each “extracts” useful features from game state
 - features combined with weights to compute desirability
- **Demo**

Feature Extractors (0,1)

- Health(pBot)
 - normalize health range to (0,1)
- DistanceToItem(pBot, int ItemType)
 - to nearest item of given type
 - if none, return 1
- IndividualWeaponStrength(pBot, int WeaponType)
 - how much ammo bot has for given weapon type
 - relative to max amount it can carry (return 1)
- TotalWeaponStrength(pBot)
 - combination of three individual weapon strengths

GetHealthGoal_Evaluator

$$Desirability_{health} = k \times \left(\frac{1 - Health}{DistToHealth} \right)$$

- the *farther away* health pack is, the *less* desirable
 - cannot divide by zero, since triggered if inside bounding radius (and thus doesn't exist any more)
- the *less* healthy, the *more* desirable
 - if at max health, desirability is zero
- k is source-level "tweak factor"



GetWeaponGoal_Evaluator

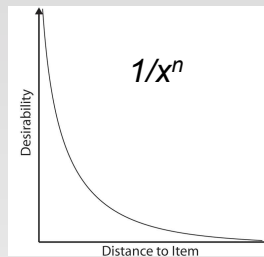
$$Desirability_{weapon} = k \times \left(\frac{Health \times (1 - WeaponStrength)}{DistToWeapon} \right)$$

- the *farther away* weapon is, the *less* desirable
- the *less* healthy, the *less* desirable to get weapon
- the *more* ammo it has, the *less* desirable
- k is source-level "tweak factor"



Non-Linear Functions

$$Desirability_{weapon} = k \times \left(\frac{Health \times (1 - WeaponStrength)}{DistToWeapon^2} \right)$$



- relative “pull” of weapon becomes much stronger as you get closer

AttackTargetGoal_Evaluator

$$Desirability_{attack} = k \times TotalWeaponStrength \times Health$$

- the *stronger* you feel, the *more* desirable to attack
 - health
 - total weapon strength
- k is source-level “tweak factor”

ExploreGoal_Evaluator

- returns fixed value of 0.05
- last resort

Bot “Personalities”

- e.g., cautious versus aggressive
- **Each bot** has Lua parameter file with additional tweak (“bias”) factors for each toplevel goal
- Easy to multiply in at end of desirability calculation

```

Bot_HealthGoalTweaker = 1.2
Bot_ShotgunGoalTweaker = 0.6
Bot_RailgunGoalTweaker = 0.5
Bot_RocketLauncherTweaker = 0.6
Bot_ExploreTweaker = 0.2
Bot_AggroGoalTweaker = 0.8

```

(Note inconsistent naming in Burke code ☹)

Homework #9 – Due Weds Midnight

- Adding a new goal, StealHealth, with associated evaluator
- Your bot should collect a health pack even if it doesn't need it, when there is a nearby opponent who does need it
- Detailed instructions online
- Familiarize you with goal code for **brainstorming Friday** and tournament

Architecture Extensions / Applications

- Player Possession
- Interruptions
- Special Path Obstacles
- Command Queuing
- Scripting

Player Possession

- Player “possesses” bot
 - right click once to select
 - right click again to possess
 - sets isPossessed() flag
- Right click on map to indicate destination
 - adds MoveToPosition goal to brain
 - invokes path planner in Activate method
 - other goal arbitration turned off

Interruptions

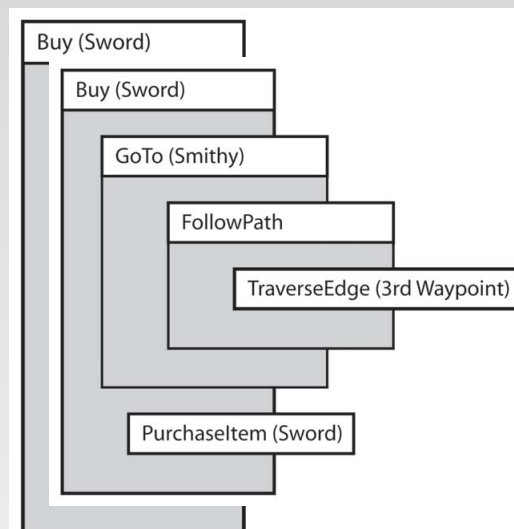
- Toplevel goal arbitration (desirability evaluation)
 - “throws away” the current goal when a “better” (higher scoring) goal is detected
 - a “one-track mind”
 - you might return to the first goal when the new goal is done (or before)---it all depends on the desirability evaluation at each tick
 - but there is no memory of previous goal (or its state information)
 - e.g., AttackTarget, GetHealth, AttackTarget
 - is this good or bad?
 - depends on what?

Interruptions

- an alternative approach/mechanism
 - which can **co-exist** with toplevel arbitration
 - when a new goal becomes appropriate
 - as determined by some event or evaluation function
 - e.g., “incoming!”, or “gas tank low”
 - **push** it onto the front of the **lowest level** current subgoal list
 - when the this new goal completes, the original subgoals (and parents) will continue as before
 - the new goal will function as an **interruption**



Interruptions

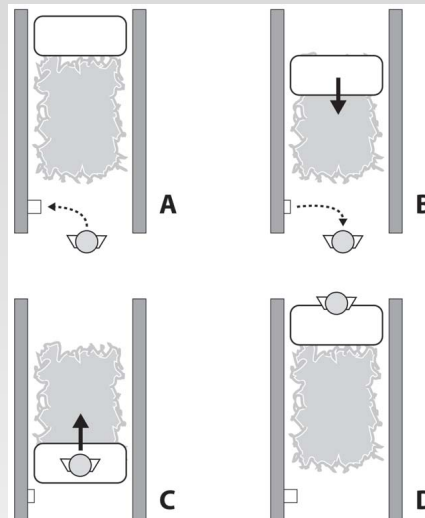


Interruptions

- But what if interruption has changed the world state enough to “break” the plan of the interrupted goal?
 - e.g., defending attacker has taken bot far from planned waypoint path
- Plans **already** need to have code to check for failure and trigger re-planning (recursively up the goal tree)
- **Conclusion:** Our bots are pretty simple and don't need interruptions, but a more “cognitively oriented” game might benefit

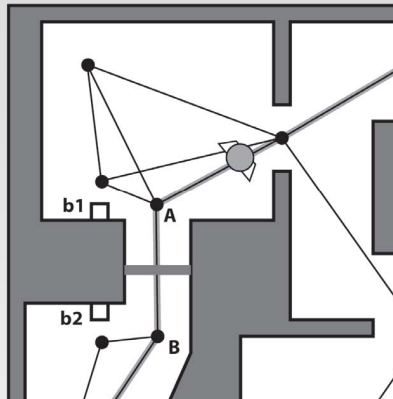
Special Path Obstacles

- bot calls the moving platform and rides it across the pit of fire...
- underlying path edge is specially marked
- FollowPath adds special subgoal instead of usual TraverseEdge



Special Path Obstacles

- Sliding door example in Raven
 - demo



Command Queuing

- How about letting the **player** put subgoals directly into the tree?
 - gives the player a way to “instruct and forget” an NPC
 - e.g., “attack this house, then take down the flag, then retreat to meeting area”
 - need some kind of user interface design
- Navigation waypoint example in Raven
 - holding down ‘Q’ key while clicking right
 - adds MovePosition goal to back of subgoal list (queue)
 - demo

Scripting

- How about exposing the subgoal lists to Lua scripting?

```
function AddGenie (...)  
  genie = CreateGenie(...)  
  genie:SayPhrase("Welcome...")  
  genie:SayPhrase("Follow me...three wishes...")  
  genie:LeadPlayerToPosition(...)  
  genie:VanishInPuffOfSmoke  
end
```

Scripting

- What do you need to do?
 - expose C methods in Lua to add subgoals to current goal
 - call appropriate Lua method from C Activate (planning) method of goal
 - optionally expose additional methods to create objects, etc.

The Road to Tournament

- *Fri, Dec 2:* Brainstorming Raven bot strategy
- *Sun, Dec 4:* Bot Design (HW #10) due
- *Sun, Dec 11:* Tournament bot (HW #12) due **6pm**
- *Mon, Dec 12:* Raven Tournament (**GH 012**)