EDUCATION

Attention, Students: Put Your Laptops Away

3:19

+ Queue

April 17, 2016 · 6:00 AM ET
Heard on Weekend Edition Sunday

https://www.npr.org/2016/04/17/474525392/attention-students-put-your-laptops-away

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.

- "Everything should be as simple as possible, but not simpler." – Einstein
- Occam (of Razor fame – parsimony, economy, succinctness in logic/problem-solving)
  - "Entities should not be multiplied more than necessary"
  - "Of two competing theories or explanations, all other things being equal, the simpler one is to be preferred."
- "All that is complex is not useful. All that is useful is simple." – Mikhail Kalashnikov (of AK-47 fame)

Announcements

- HW 5 Due Sunday night
- Fall recess, 10/14: no class
- Exam 1

behaviors themselves.

12.9.1 A "Soft-Coded" Behavior Tree

Here, we take as an example a "first-generation" behavior tree—essentially a decision tree with behavior states as its leaves, a very common design in modern games including *Crysis 2*.

Implementations have included the use of snippets of Lua to form flexible conditions at each node, or in *Crysis 2* a simple virtual machine executing a tree specified by XML [Martins 11]. However, in essence, the structure is a simple tree of if-else statements—the rest of the behavior tree architecture deriving from the requirement for rapid iteration, as a set of hardcoded C++ if-else clauses would bring AI development to a near halt.

We can now reevaluate that assumption. Under RCC++ such decision code can be thought of as "soft"-coded—we can change it at will while the game is running. So, assuming some nicely formatted code, let's consider its properties: being raw C++, it is obviously extremely fast, it will handle errors without crashing, it has easy access to all the state in our AI system, it can share sub-trees as function calls, it can use all our C++ math and utility routines, we can make use of our existing debugger, we can apply our existing profiler if we need to, you can see useful diffs from source control ... the list goes on. In a few lines of code you've implemented a behavior tree—in fact, an unusually fast and powerful behavior tree.

The main aspect for improvement is that of the interface for designers. Simple parameters may, of course, be exposed through XML or a GUI. In this author's experience more structural changes are best kept the responsibility of AI programmers, who often find a simple text specification better to work with than a graphical one. However, such if-else code may be very easily generated from a graphical representation should one be required, with the simple act of replacing the source file triggering runtime-compilation.

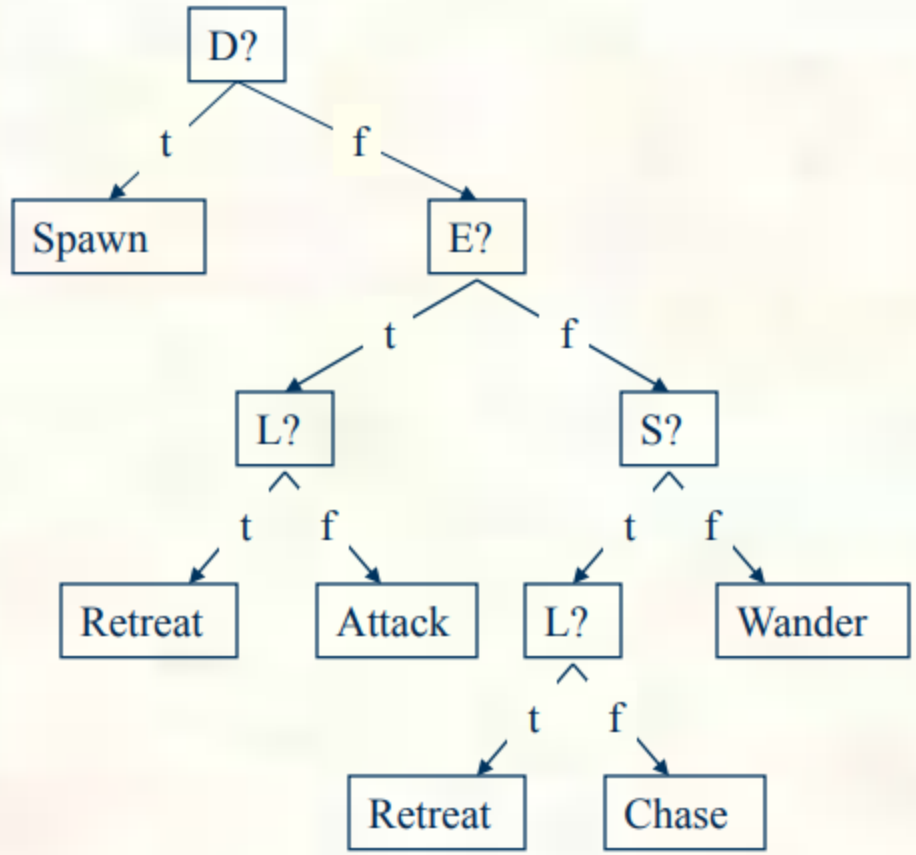
12.9.2 A Blackboard

Blackboards may be naturally represented in languages such as Lua or Python as a table of key-value pairs, comprising strings and dynamically typed values. While similar constructions are possible in C++, they represent a possible performance bottleneck. In an analogous approach to our behavior tree, we can represent our blackboards under RCC++ as simple structs—each key-value pair becoming a named member variable of appropriate type.

Reading and writing from the blackboard becomes as simple as passing its pointer to our sensory systems and to our RCC++ behavior tree. The main difficulty we must resolve is that of dependencies, as all the code that uses it must include the struct definition as a header,

Decision Tree for Quake

- Just one tree
- Attributes: Enemy=<t,f>
  Low=<t,f> Sound=<t,f>
  Death=<t,f>
- Actions: Attack, Retreat, Chase, Spawn, Wander
- Could add additional trees:
  - If I'm attacking, which weapon should I use?
  - If I'm wandering, which way should I go?
  - Can be thought of as just extending given tree (but easier to design)
  - Or, can share pieces of tree, such as a Retreat sub-tree



https://www.cs.utexas.edu/users/fussell/courses/cs378-spring2012/lectures/cs378-13.pdf

Sidebar – The Game Loop

The “**Game Loop**”

while game is running

process inputs

update game world

generate outputs

```
while( user doesn't exit )
  check for user input
  run AI
  move enemies
  resolve collisions
  draw graphics
  play sounds
end while
```

https://en.wikipedia.org/wiki/Game_programming

http://www.informit.com/articles/article.aspx?p=2167437&seqNum=3

What are production/rule systems?

- Based on the current game **state**, activate a set of **rules**, pick/**arbitrate** from those based on some heuristic (e.g. best matches conditions)
- What were these traditionally called?
- Use in Games Industry includes Environment-sensitive dialog generation (dynamic dialog gen)

- 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

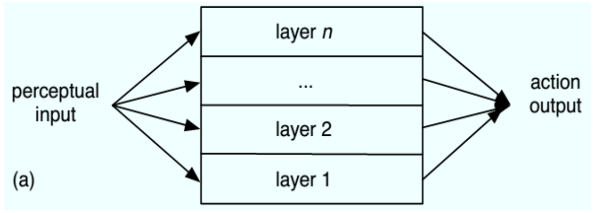
What is Planning?

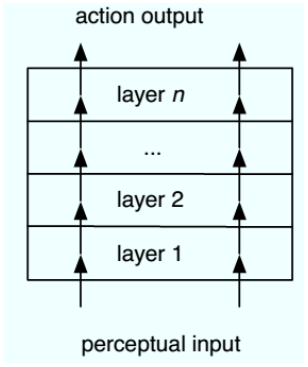
- Finding a **sequence of actions** to achieve a **goal** or **accomplish a task**
  - Problems requiring **deliberate** thought ahead of time and a sequence of actions
- Basic planning comes down to **search**:
  - "behavior planning using A*"
    - Given: state of the world, a goal, and models of actions
    - Find: sequence of actions (a plan) that achieves the goal
  - hierarchical task network planning
    - Given: state of world, a list of tasks, and models of actions and methods
    - Find: a sequence of actions (a plan) that achieves the tasks by recusively reducing them into appropriate sub-tasks supported by primitive actions

Decision Making: Reactions vs Anticipation

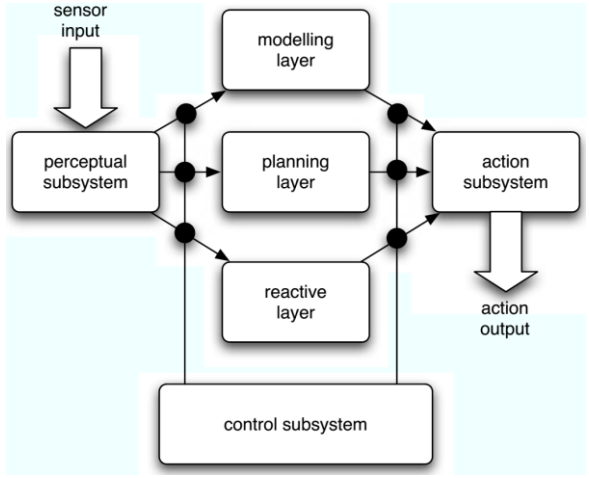
- Shallowness & Realism
  - All reactive techniques have the agent take **next** "best"/prescribed move, but **don't look further** into the future than the next state. Agents ought to be **motivated by goals** and able to consider the effects of actions
- Adaptability
  - Each technique is more general/adaptive than the last (Planning > Rule Systems > Behavior Trees > FSMs > Decision Tree). To **perform *well* in unanticipated** situations is challenging
- Design Burden
  - All decision making techniques impart an authoring burden on designers (FSMs: states and transitions, B trees: nodes and structure, Rules: all conditions and each individual rule, Planning: state rep and methods/operators).
  - Knowledge engineering is a "hot potato" (can't eat it, no free lunch)

Decision Making: Architectures…

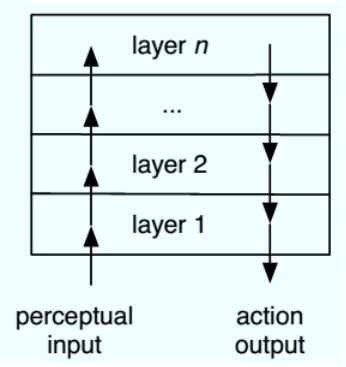




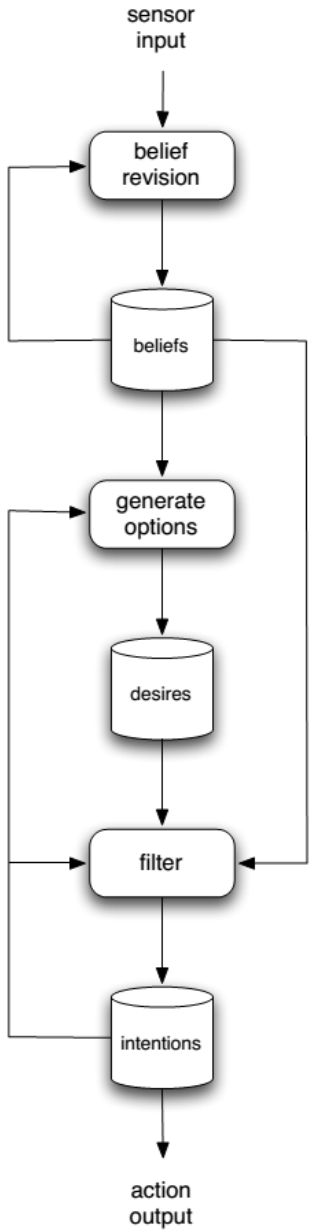
One-pass



Brooks Subsumption Arch.



Two-pass



BDI

See Multiagent Systems 2nd ed, Gerhard Weiss

Decision Making: Rule-Based Systems

2019-10-07 (and 09)

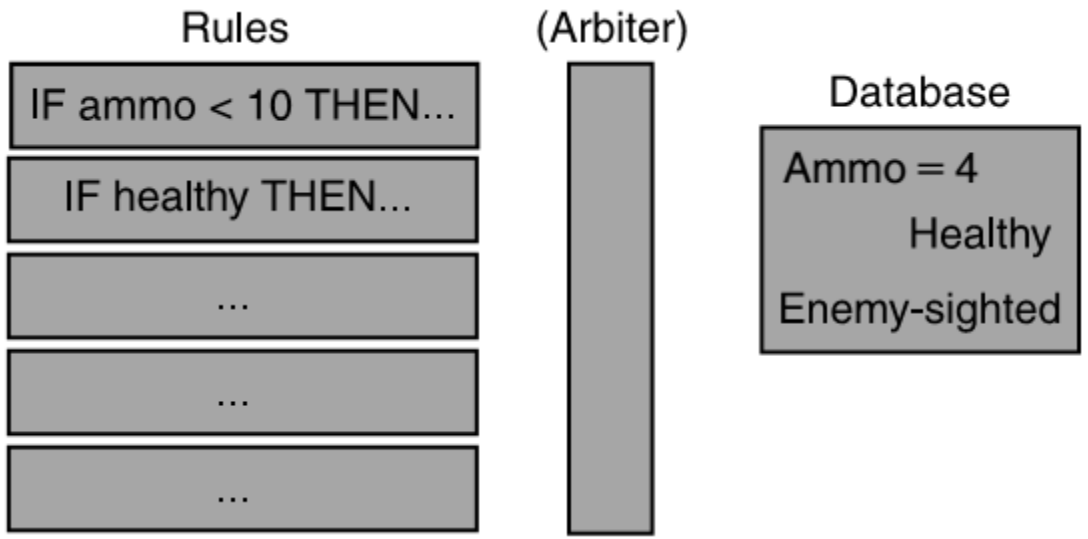
See M&F 5.8

Background

- Symbolic AI, “Expert Systems”
- Vanguard of AI research 70s + early 80s
  - By mid 80’s, 2/3rd of Fortune 500 applied them to daily activities
- Used in some games, but not as common as FSMs or decision trees
  - Reputation for inefficiency + challenge to impl.
  - Similar behaviors achievable using Dtree/FSMs
- More robust than decision trees when worlds are unpredictable – improvisation possible
  - Recall, decision trees can be turned in to rules
- A form of reactive planning

Production/Rule System

- Consists of :
  - Facts (database of knowledge, “working memory”, stores state)
  - Rules (if/then constructs, with Boolean ops)
  - May also include arbiter / conflict resolution scheme
- Like a FSM, but triggers based on DB/effects are more general
- Basic process / idea:
  - Match: facts to if-part of rules (“pattern matching”)
    - Rules with matching if’s become activated (“**triggered**”)
  - Arbitrate: Choose an active rule to “**fire**”
    - Can make change to facts or to world
  - Repeat



Millington Figure 5.46

If enemiesInSight > 0 and patrolling THEN
    remove(patrolling)
    add(attackNearest)

Comments

- It is like writing a program and then allowing the computer to decide which functions to call and when
- Forward vs. Backward chaining
  - B: theorem proving + planning
  - Authors never saw backward in games
- DB **rewriting rules** vs **Condition-Action Rules**
  - Rewrite Rules can change DB (+/- facts)
  - Typically only for AI specific knowledge (e.g. patrol)
  - Bias in GAI for condition-action rules (no rewrites)

DATABASE:
1500 kg fuel remaining
100 km from base
enemies sighted: Enemy 42, Enemy 21
currently patroling

(DB rewrite) RULE:
IF number of sighted enemies > 0
and currently patroling
THEN
remove(currently patroling)
add(attack first sighted enemy)

Declarative / Short-term Knowledge

- Stateable facts about the world
- (*attribute value*)
  - (Captain-weapon rifle)
- *value* can be nested knowledge
  - (Captain-weapon (rifle (ammo 36)))

DK: Facts

- Health(captain, 51)
- Health(Johnson, 38)
- Health(Sale, 42)
- Health(Whisker, 15)
- Holding(whisker, radio)
- Weapon(whisker, rifle)
- Weapon(johnson, pistol)
- Ammo(whisker, 36)

- Whisker
  - Health: 51
  - Holding: radio

- (captain
  (weapon (rifle (ammo 36) (clips 2))
  (health 51)
  (position [21, 46, 92])
)

  (radio (held-by whisker))

Procedural / Long-term Knowledge

- Knowledge about how we *do* the things we do
- IF (some facts about the world) THEN do (some action)

PK: Rules

- The If-clause contains items of data to match joined by any Boolean operators

- IF whisker's health < 15 AND Whisker holding radio
  THEN Whisker: Radio-call "help!" on radio

- IF whisker's health = 0 AND whisker holding radio
  THEN
  - Remove(whisker holding radio)
  - Add(radio on ground)

- IF ?anyone health < 15
  THEN …

Components

- Declarative / short-term knowledge (facts/KB)
- Procedural / long-term knowledge (actions)
- Selection knowledge (conditions, arbiter)
- Arbiter
  - First applicable (FIFO on input order)
  - Least recently used (LIFO on use order)
  - Random
  - Priority / Most specific conditions
  - Dynamic Priority system

Unification

- Binding of vars in logical statements
  - Same problem as in Planning
  - (?persn health 0-15) AND (Radio (heldby ?persn))
- Allows rules to match in many situations
  - See Russell & Norvig, Millington 5.8.7
- On complexity:
  - N is number of items in DB,
  - M is number of clauses in a pattern to match…
  - O(nm), or maybe O(m $\log_2$ n), but generally O(n^m)

Simple Algorithm

```
def ruleBasedIteration( database, rules ):
    for rule in rules:
        bindings = []
        if rule.ifClause.matches( database, bindings ):
            rule.action( bindings ) #fire rule
            return #exit; we're done for this iteration
    #if we get here, there's no match; can do default
    #or do nothing
    return
```

```
class Rule:
    ifClause
    def action(bindings)

class Match:
    def matches( DB, bindings)
```

Performance

- 90%+ goes to matching the rule conditions against working memory
- Naïve approach of evaluating all rules each cycle is too slow
  - May be necessary to support rules of arbitrary complexity (ie function calls allowed in condition)
- Note that
  - Number of changes to working memory on any given cycle is small
  - Rule conditions can be associated with changes → event driven

RETE

- AI industry standard for rule matching
- Rule patterns represented in DAG
  - Pattern nodes, Join nodes, Rule Nodes
- Each path represents set of patterns for one rule
  - Fast matching (share evaluation)
  - Graceful updates (add/remove facts)
  - Determines which rules are active (all)
  - Millington & Funge cover very well

Warning re IP

“You should also be careful of proprietary algorithms because many are patented. Just because an algorithm is published doesn’t mean it isn’t patented. You could end up having to pay a license fee for your implementation, even if you wrote the source code from scratch. We’re not lawyers, so we’d advise you to see an intellectual property attorney if you have any doubts.” – M&F

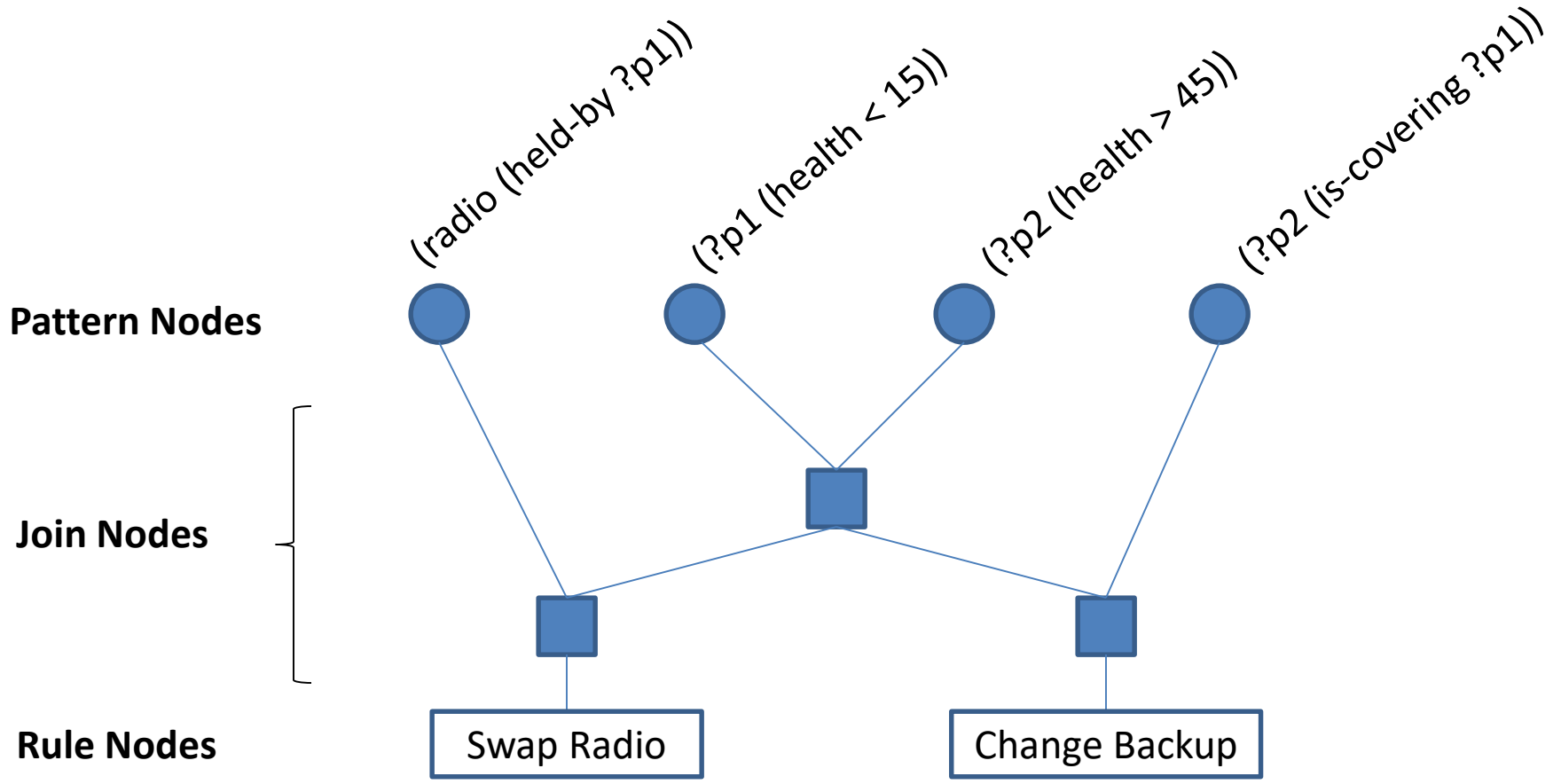
Rete Example

Swap Radio Rule:

```
IF
  (?p1 (health < 15)) &&
  (?p2 (health > 45)) &&
  (radio (held-by ?p1))
THEN
  remove(radio (held-by ?p1))
  add(radio (held-by ?p2)
```

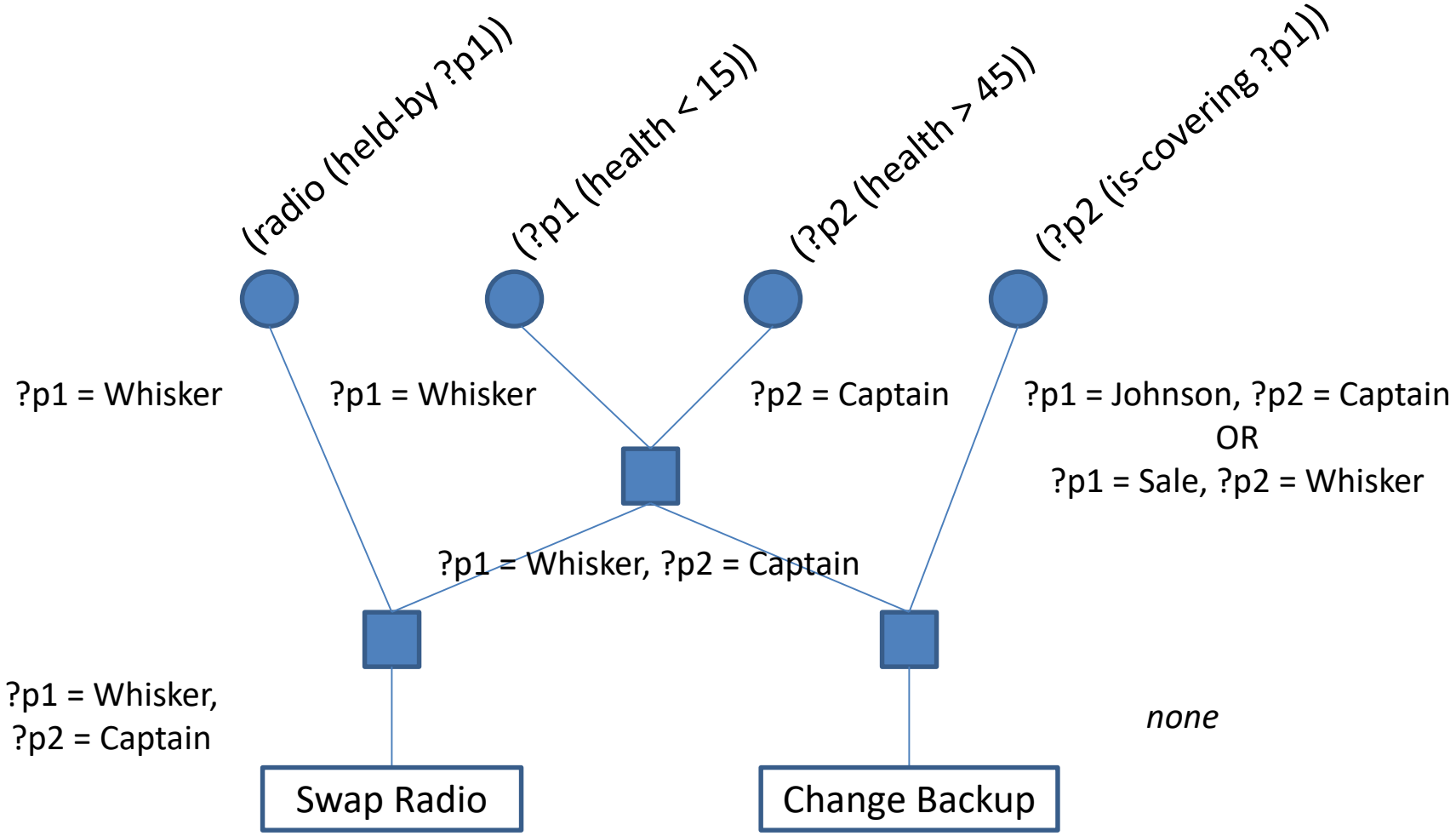
Change Backup Rule:

```
IF
  (?p1 (health < 15)) &&
  (?p2 (health > 45)) &&
  (?p2 (is-covering ?p1))
THEN
  remove(?p2 (is-covering ?p1))
  add(?p1 (is-covering ?p2))
```


(radio (held-by ?p1))
(?p1 (health < 15))
(?p2 (health > 45))
(?p2 (is-covering ?p1))
Pattern Nodes
Join Nodes
Rule Nodes
Swap Radio
Change Backup

M&F Fig 5.48

(Captain (health 57) (is-covering Johnson))
(Johnson (health 38))
(Sale (health 42))
(Whisker (health 15) (is-covering Sale))
(Radio (held-by Whisker))



Pattern Nodes

- Database fed into top of network
- Pattern nodes find matches in database and pass them down to join nodes
  - When wildcards are used, variable bindings are also passed down

Pattern Nodes

- Pattern nodes keep record of matching facts for incremental updating
- Find *all* matches instead of *any* match
  - …and all variable-bindings
- E.g.
  - ?person1 could be Whisker or Captain
  - Not at the same time, but we pass both since we don't know which is useful

Join Nodes

- Make sure that both inputs have matched and any variables agree
- When variable-bindings are used, join nodes identify all acceptable combinations of bindings
- Not necessarily AND
  - AND and XOR need extra support for unification

Rule Nodes

- All rules that receive input at bottom of network are triggered
- Arbiter determines which triggered rule goes on to fire

Updating the Network

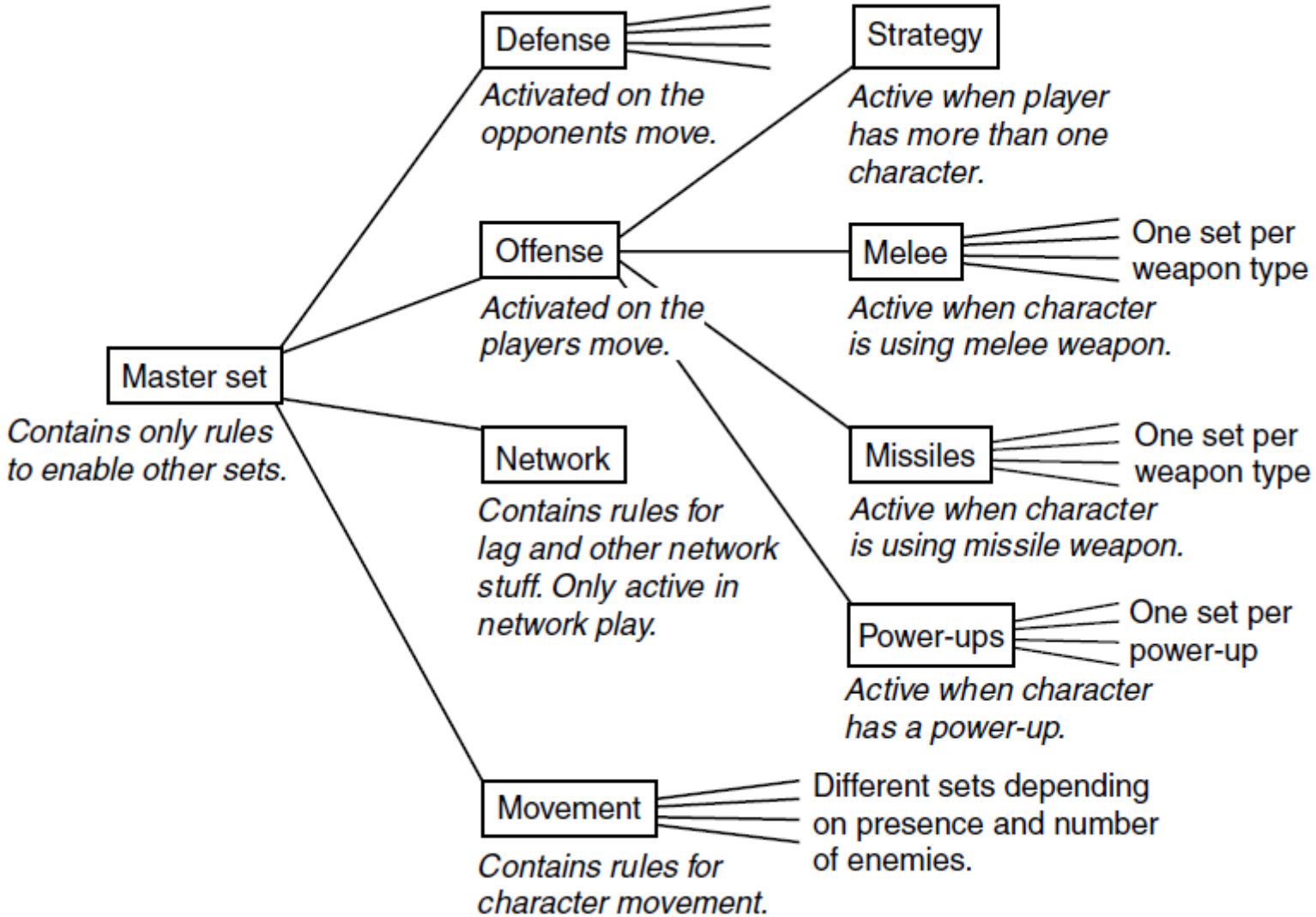
- Could re-run each time with new database
  - But usually, data changes minimally between iterations
- Nodes store data, so only need to process changes to database.
  - Only update nodes that need it! Need remove/add.
  - Effects are handled by walking down the network
- Removing facts from database:
  - Request sent to pattern nodes
  - If node has stored match, remove it and pass request down.
  - Adding is basically the same.
- Adding is basically the same, looking for new matches.
  - Most efficient to handle removes before adds.
  - After updates, arbiter still decides among triggered rules

Rete Efficiency

- O(*nmp*) time efficiency
  - n = # rules
  - m = # clauses per rule
  - p = # facts in database
- Unifying wildcards can take over if wildcard matches are large
- More memory usage → faster performance

Large Rule Sets

- Series of 2D turn-based war games
  - Large rule set
  - Each game in series required addition of many new rules: new features, player requests, bug fixes
  - Eventually, even RETE barfs
- Solution?
  - Group rules, and make activation hierarchy
  - Only rules in active sets are triggered
  - Disabled rules have no chance to trigger
- See “agenda groups” in Drools


Master set
Contains only rules to enable other sets.
Defense
Activated on the opponents move.
Offense
Activated on the players move.
Network
Contains rules for lag and other network stuff. Only active in network play.
Movement
Contains rules for character movement.
Different sets depending on presence and number of enemies.
Strategy
Active when player has more than one character.
Melee
One set per weapon type
Active when character is using melee weapon.
Missiles
One set per weapon type
Active when character is using missile weapon.
Power-ups
One set per power-up
Active when character has a power-up.

M&F 5.53

Justification in Expert Systems

- Common extension is audit trail
- Capture rule firing information
  - The rule that fired
  - The data that the rule matched
  - Time stamp
- This information can be recursive
- Useful for debugging and justifying behavior

Rules: Advantages & Disadvantages

- Pros
  - First technique we've seen that allows for improvisation / response to novel conditions
  - Another way to achieve reactive behaviors
  - Don't need to specify response to every contingency
- Cons
  - Less designer control – possible to have unanticipated reactions
  - Difficult to author and debug robust rule systems
  - Difficult to make sequences of behaviors to achieve a goal
  - Reputation for inefficiency + challenge to impl.
  - Similar behaviors achievable using Dtree/FSMs

Resources

- A full source code implementation is provided on the M&F website, with lots of comments
- Jess
  - http://www.jessrules.com/
- Drools (/OptaPlanner)
  - http://www.jboss.org/drools/
  - http://www.optaplanner.org/
  - http://www.javacodegeeks.com/2013/04/life-beyond-rete-r-i-p-rete-2013.html
- Aima-java, under FOL (see Unifier.java)
  - https://code.google.com/p/aima-java/

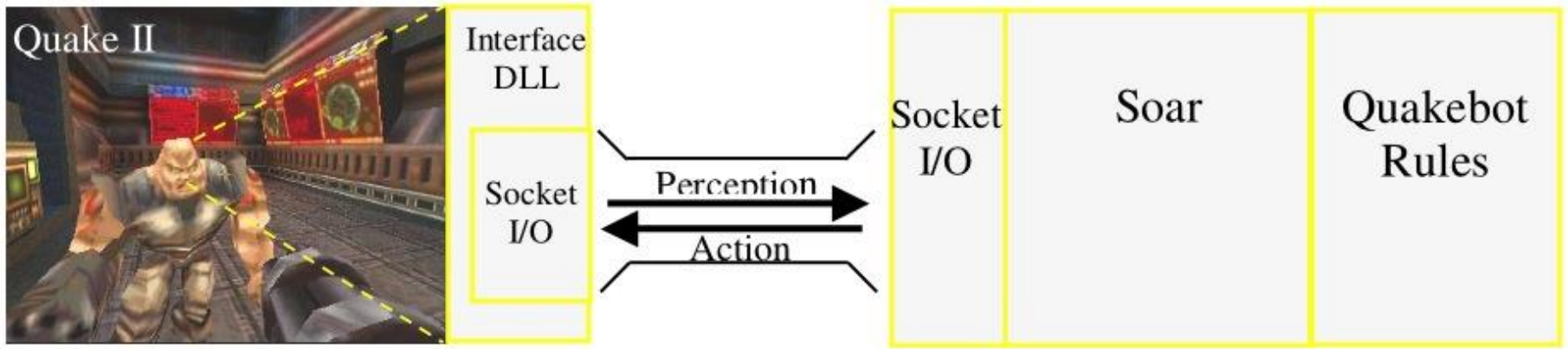
Jess

```
(defrule change-backup
        (< (health ?person1) 15)
        (> (health ?person2) 45)
        ?cover <- (is-covering (?person2 ?person1))
        ==>
        (//make a call to java//)
        (retract ?cover)
        (add (is-covering (?person1 ?person2)))
)
```

Soar

- A production system based on a theory of human cognition
- Production system with fancy arbitration
  - If two rules are active, Soar breaks the tie by firing more rules to figure out which is better
  - Forward mental simulation

```
sp {hello-world
    (state <s> ^type state)
    -->
    (write |Hello World|)
    (halt)}
```



Soar

- Newell, Laird, & Rosenbloom (CMU)
- Represents Newell's *Unified Theory of Cognition*
- Several decades in development
- Used in academic and military applications
- Previous Cognitive Psychology use
- Largest system: 8,000 rules

Decision Making:
(Blackboard) Communication

2019-10-09

M&F 5.9

DM: Communication. Why?

- Lens: Multi-agent system
  - Collection of collaborative agents
  - Communicate & cooperate
  - Retain autonomy
  - Need for negotiation / mutually acceptable agreements (cooperative problem solving)
- Reasoning decomposition: distributed expertise
  - Problems too large for single / centralized agent
  - Reactive agents rarely communicate / collaborate
  - Problem independence, partial result sharing
- Hope: Sum greater than parts

An **agent** is a **computational entity** such as a software program or a robot that is **situated in some environment** and that to some extent is able to **act autonomously** in order to achieve its **design objectives**. As **interacting entities**, agents do not simply exchange data but are **actively engaged in cooperative and competitive scenarios**. They may **communicate** on the basis of semantically rich languages, and they **achieve agreements** and **make decisions** on the basis of processes such as negotiation, argumentation, voting, auctioning, and coalition formation. As intelligent entities, agents **act flexibly, that is, both reactively and deliberatively**, in a variety of environmental circumstances on the basis of processes such as planning, learning, and constraint satisfaction. As **autonomous entities**, agents have **far-reaching control over their behavior** within the frame of their objectives**, possess decision authority** in a wide variety of circumstances, and are **able to handle complex and unforeseen situations on their own** and without the intervention of humans or other systems. And as entities situated in some environment, **agents perceive their environment** at least partially and **act upon their environment without being in full control of it**.

See Multiagent Systems 2nd ed, Gerhard Weiss

Distributed Decision Making: A Recipe

1. Decompose the task
2. Allocate subtasks to “experts”
3. Await task accomplishment
4. Synthesize & Arbitrate results

Information sharing needed for most/all!

Communication Types

- Point to Point
  - Experts directly communicate w/eachother
  - Where have we seen this?
- Broadcast
  - Send information to group of experts
  - Talk about today.
- Mediated
  - Experts go through facilitator/arbitrator

Communication Mediums

- Firm software interfaces
- Databases
- Protocol layers (e.g.: TCP/IP + JSON)
- Hierarchies (hybrids)
- Pub/Sub services

BLACKBOARD ARCHITECTURES

Daniel D. Corkill, 1991

Blackboard systems are not new technology. The first blackboard system, the Hearsay-II speech understanding system [1], was developed nearly twenty years ago. While the basic features of Hearsay-II remain in today's blackboard systems, numerous advances and enhancements have been made as a result of experience gained in using blackboard systems in widely varying application areas. **Unlike most AI problem-solving techniques that implement formal models, the blackboard approach was designed as a means for dealing with ill-defined, complex applications.** Unconstrained by formal requirements, researchers and developers have had considerable flexibility in inventing and applying advanced techniques to blackboard architectures. However, **the lack of formal specifications has also contributed to confusion about blackboard systems and their proper place in the AI problem-solving toolkit.** This article describes the characteristics and potential of blackboard systems. I'll discuss what a blackboard system is (and is not) and why the use of blackboard-based problem solving is only now emerging from the academic and research laboratory. Finally, I'll discuss whether you should consider using the blackboard approach for your applications and how to get started using a blackboard approach.

Corkill, Daniel D. "Blackboard systems." *AI expert* 6.9 (1991): 40-47.

Blackboards

- Isn’t a decision making algorithm
- Architecture / coord. mechanism / pattern
- Problem: Multiple decision making systems (experts). How to communicate (share data)?

On simplicity

- M&F: “Decision trees, state machines, and blackboard architectures have all been used to control steering behaviors. Blackboard architectures, in particular, **are suited to cooperating steering behaviors**; each behavior is an expert that can read (from the blackboard) what other behaviors would like to do before having its own say.”

On reusability

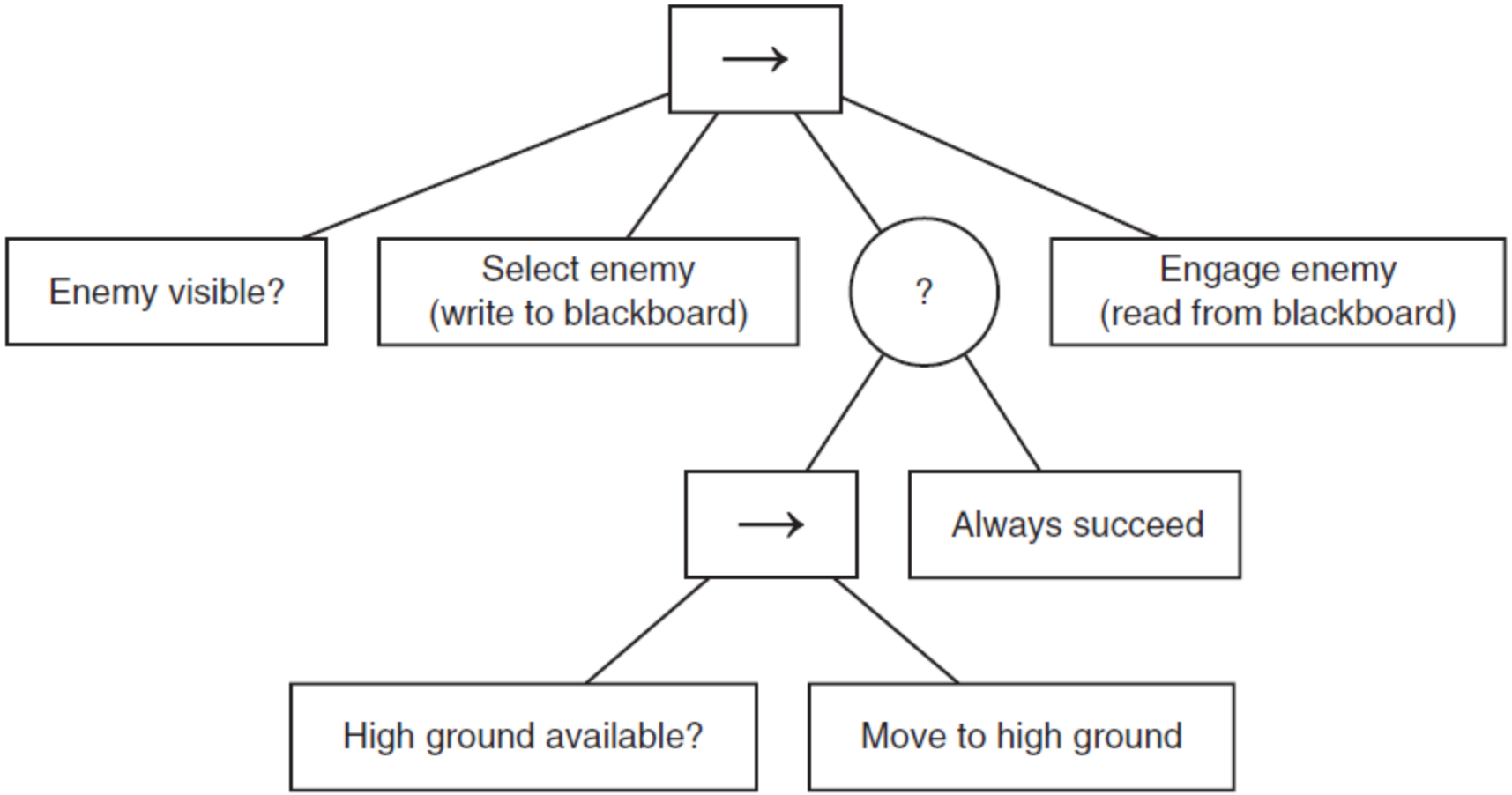
- M&F: “The most sensible approach is to **decouple the data** that behaviors need from the tasks themselves. We will do this by using an **external data store** for all the data that the behavior tree needs. We’ll call this data store a blackboard. […] For now it is simply important to know that **the blackboard can store any kind of data** and that **interested tasks can query it** for the data they need. Using this external blackboard, we can write tasks that are still independent of one another but can **communicate when needed**.”

Example

```
class MoveTo (Task):
        # The blackboard we're using
        blackboard

        def run():
                target = blackboard.get('target')
                if target:
                        character = blackboard.get('character')
                        steering.arrive(character, target)
                        return True
                else:
                        return False
```

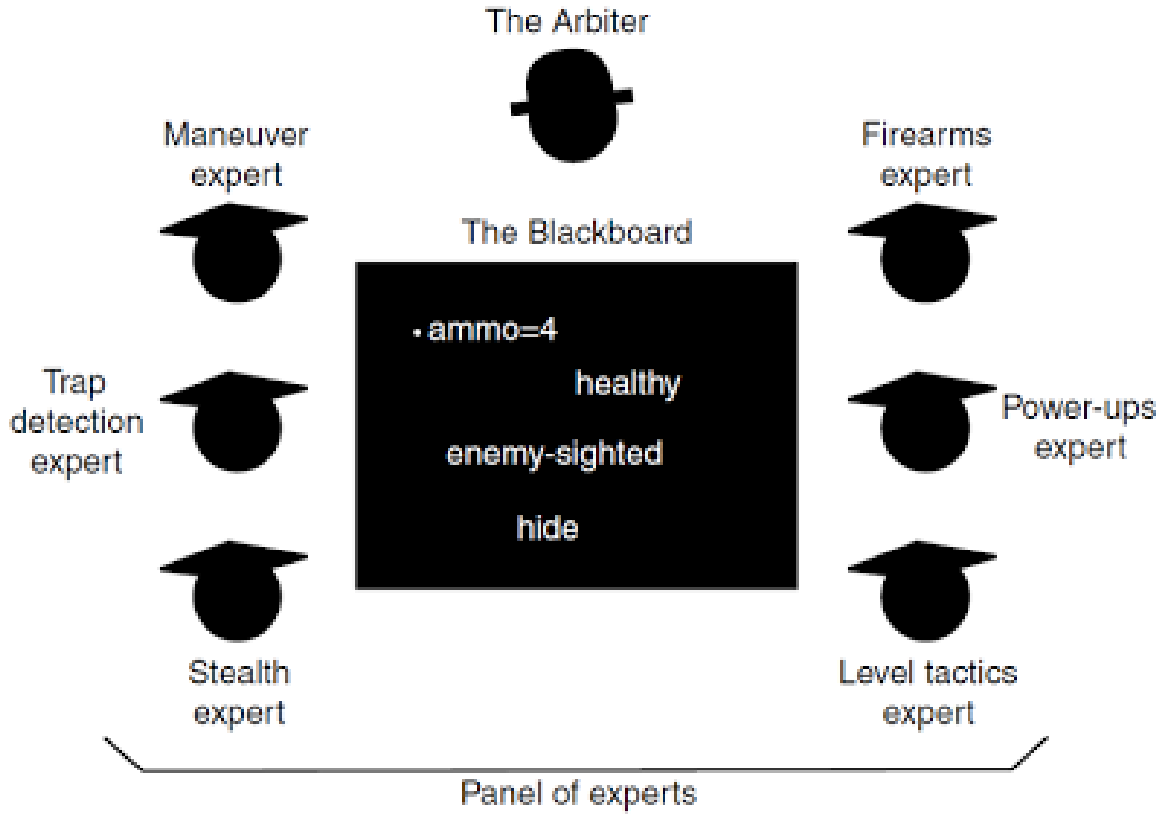
M&F 5.4


→
Enemy visible?
Select enemy
(write to blackboard)
?
Engage enemy
(read from blackboard)
→
Always succeed
High ground available?
Move to high ground

M&F Fig 5.36

Basic BB Architecture

- 3 main parts:
  - Experts
  - BB
  - Arbiter
- Other:
  - Action history
  - Scheduled Actions



Millington & Funge, Figure 5.54

BB Data Format

- Often uses application-specific organization
- Highly domain-dependent
  - 3D locations, maneuver (steering) info
  - FOL strings (flat, hierarchical)
  - Polymorphic data types
- Three typical features:
  - Value (e.g. 3)
  - Type (e.g. float)
  - Semantic Information (e.g. lives remaining)

Information on the BB

- Shared data
- Present task of each expert
- Current state of solution
- Intermediate results
- Next subproblems to be solved
- Requests for help
- Action scheduling

- E.g. Steering:
  - 3D locations
  - combinations of maneuvers
  - animations.
- E.g. Decision making:
  - game state
  - position of enemies or resources
  - internal state of a character
  - targets
  - strategies

BB Arbiter in Control

- Advertises next problems to be solved
- Checks on progress of experts
- Assign pending problems
- Monitor change
  - Polling vs Observer patterns
  - Can notify experts of relevant changes

BB Experts in Control

- On each decision making cycle
  - Experts look at BB, indicate interest (e.g. numeric **insistence value**)
  - Arbiter selects an expert to have control
  - Expert does work and/or modifies blackboard
  - Expert relinquishes control

BB Uses

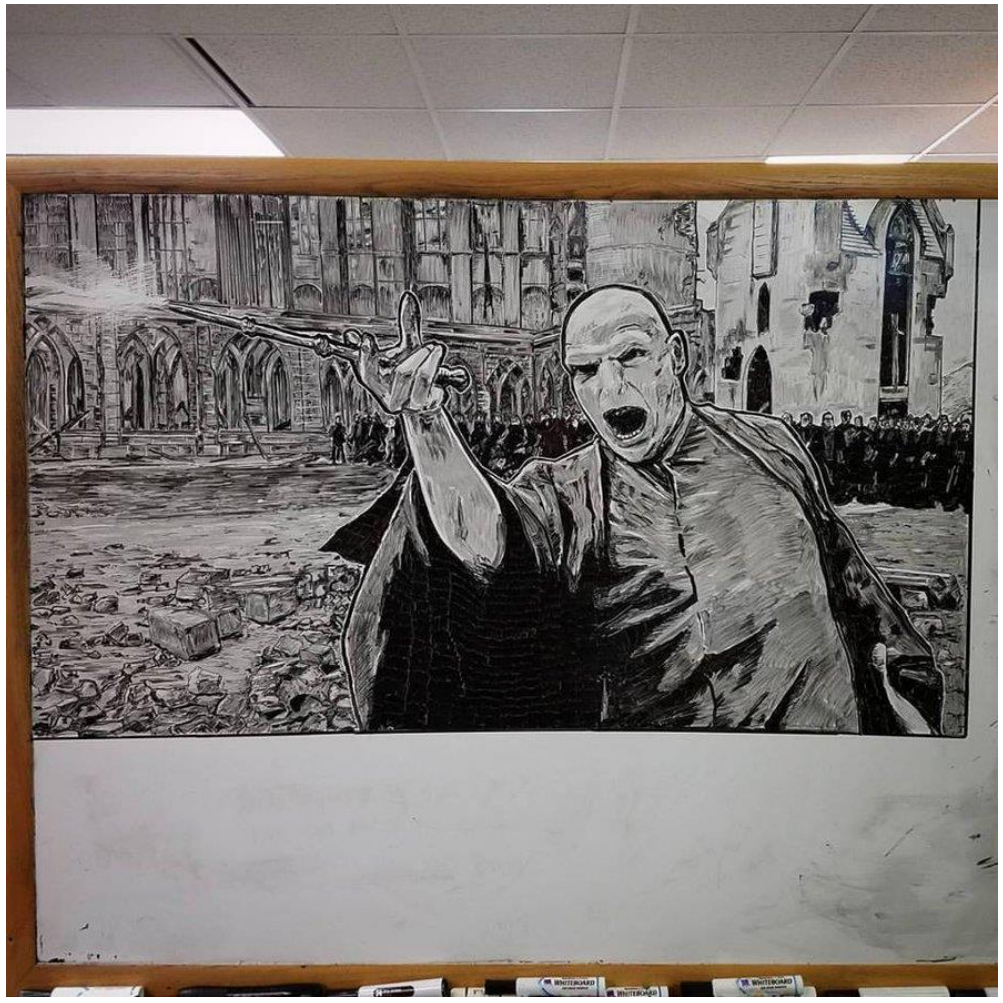
- Conflict detection
  - Task level
  - (incompatible) solution level
  - Action level
    - Potential actions, along with a set of agreement flags
- Task sharing
- Result / information sharing
  - Includes both partial and complete results

Is a BB?

- RBS?
  - Experts: rules
  - BB: Facts DB
  - Arbiter: which rule(s) to fire
- FSMs?
  - Subset of RBS
  - Experts: transitions (rewrite state)
  - BB: current state + related info
  - Arbiter: which transition(s) to fire

BB Pros and Cons

- Pro:
  - Flexible, allowing for comm. + coop.; (*n* bb's)
  - Independent of cooperation strategy
  - Does not restrict internal structure of agent
- Con
  - Management code
  - Complicated data structures
  - Centralized structure (single point of failure)
  - System bottleneck
- Have a bad rep among game+academic AI. But they're used anyway, and "shall not be named"



Work whiteboard Voldemort

by spinester

Traditional Art / Drawings / Portraits & Figures ©2016-2018 spinester

#dryerase #blackandwhite #drawing #dryeraseboard #harrypotter #voldemort #whiteboard #workinprogress (show more)

I love the Potter series. Had to do a whiteboard from it at some point and finally got around to it.
Black dry erase markers on whiteboard.