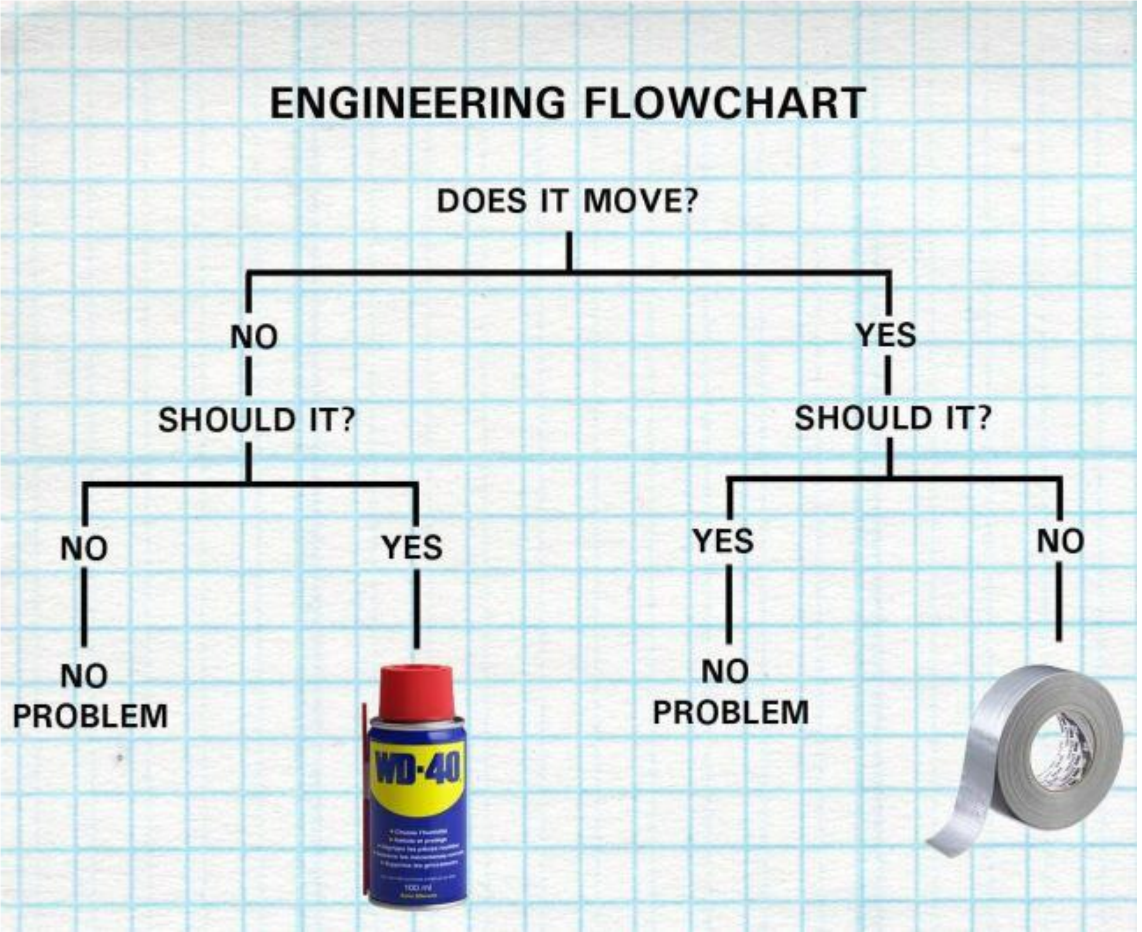
Disclaimer: I use these notes as a guide rather than a comprehensive coverage of the topic. They are neither a substitute for attending the lectures nor for reading the assigned material.



FSM Pros & Cons

Pro

- Ubiquitous (not only in digital games)
- Quick and simple to code
- (can be) Easy* to debug
- Very fast: Small computational overhead
- Intuitive
- Flexible
- Easy for designers without coding knowledge
- Non-deterministic FSM can make behavior unpredictable

Con

- When it fails, fails hard:
  - A transition from one state to another requires forethought (get stuck in a state or can't do the "correct" next action)
- Number of states can grow fast
  - Exponentially with number of events in world (multiple ways to react to same event given other variables): $s=2^e$
- Number of transitions/arcs can grow even faster: $a=s^2$
- Doesn't work with sequences of actions/memory

More problems with FSM

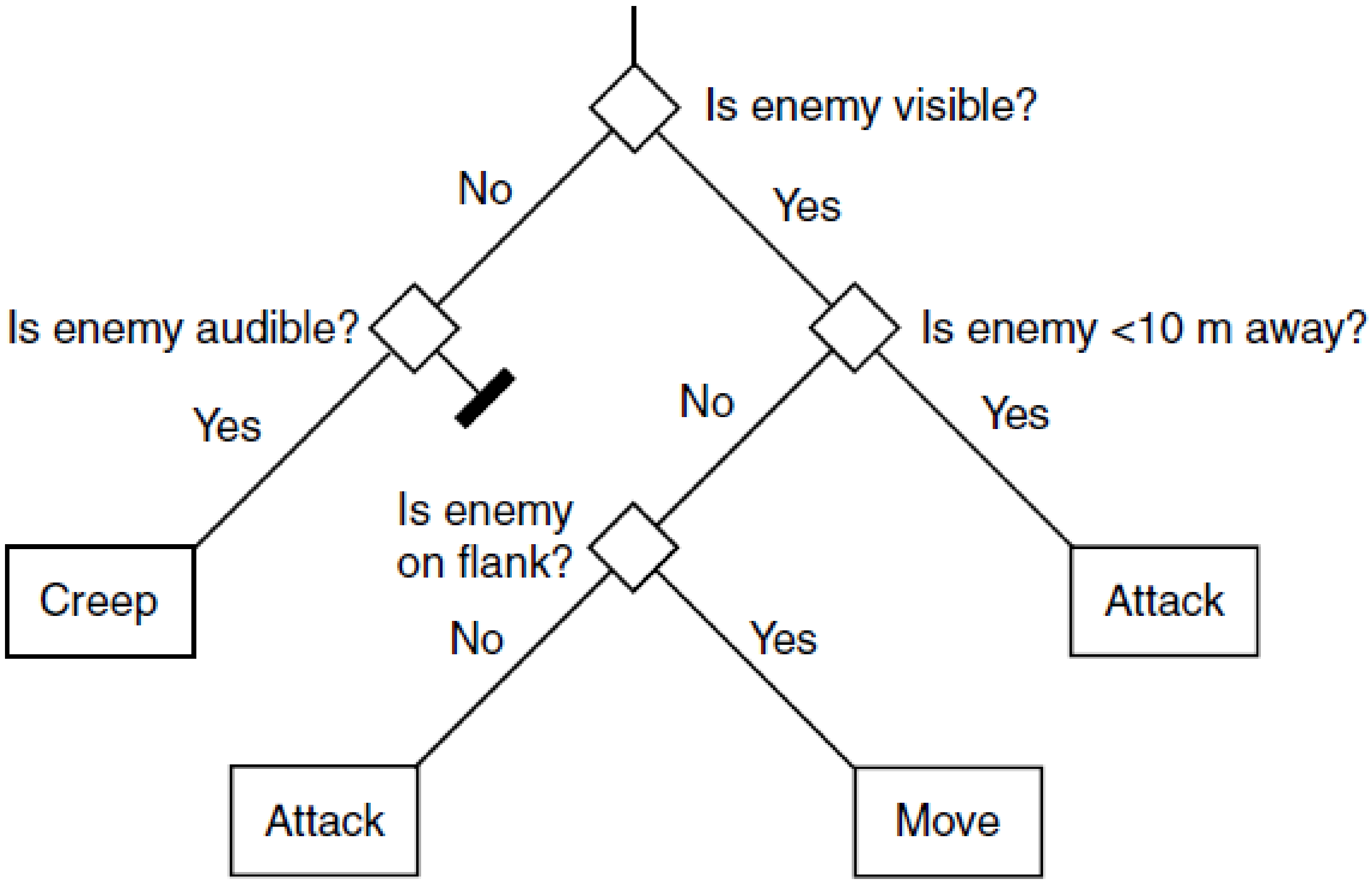
- Maintainability:
  - Addition/removal of state requires change of conditions of all other states that have transition to the new or old one. Susceptible to errors
- Scalability:
  - FSMs with many states lose readability, becoming rats nest.
- Reusability:
  - Coupling between states is strong; often impossible to use the same behavior in multiple projects
- Parallelism:
  - With a FSM, how do you run two different states at once?

Decision Making: (Decision & Behavior) Trees

2019-09-30

M&F Ch 5.2

Decision Trees



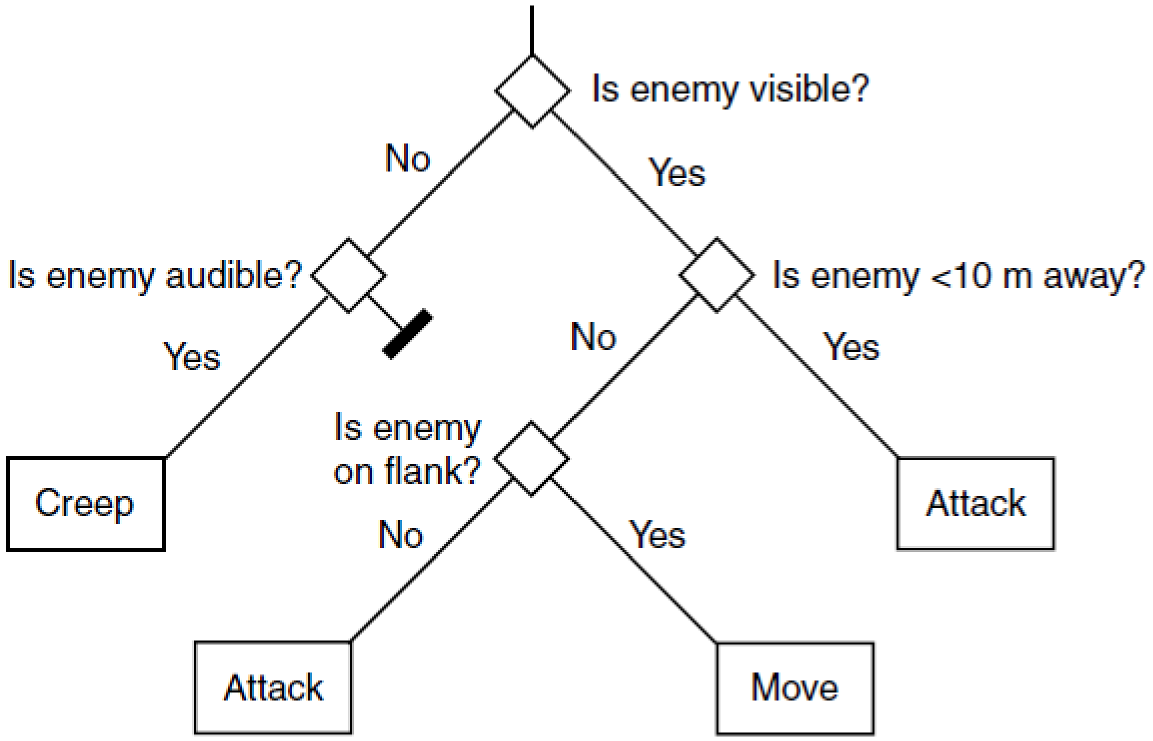
M&F 5.3

Decision Trees

- Fast, simple, easily implemented, easy to grok (simple ones)
- Modular & easy to create
- Simplest decision making technique
- Used extensively to control
  - Characters
  - In-game decision making (eg animation); complex strategic and tactical AI
- Can be learned (rare in games)
  - Learned tree still easy to grok: rules have straightforward interpretation
  - Can be robust in the presence of errors, missing data, and large numbers of attributes
  - Do not require long training times
- w/out learning, it's essentially a GUI (or fancy structure) for conditionals

D-Tree Structure

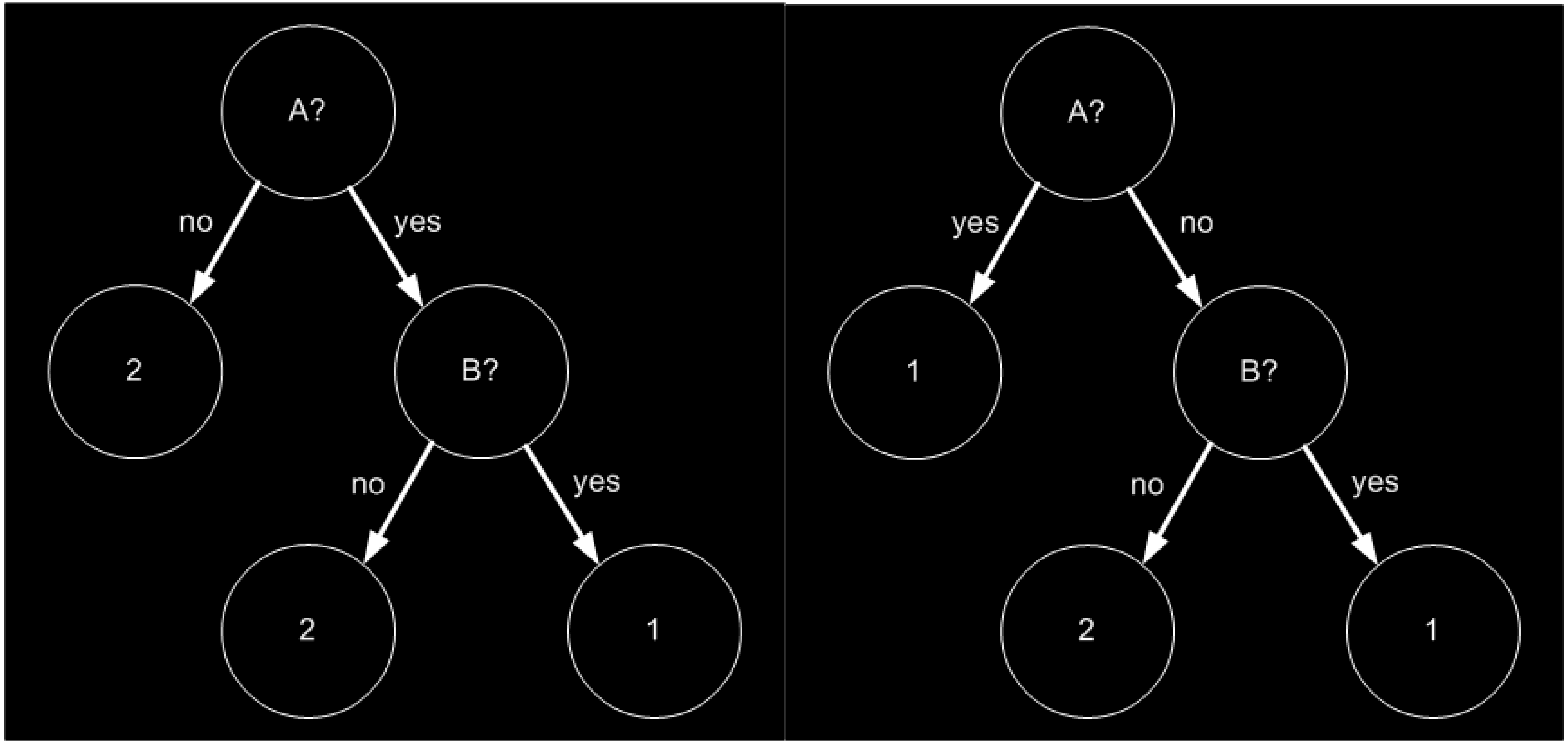
- Dtree made of connected decision points
  - root == starting decision
  - leaves == actions
- For each decision, one of 2+ options is selected
- Typically use global game state



Decisions

- Can be of multiple types
  - Boolean
  - Enumeration
  - Numeric range
  - etc.
- No explicit AND or OR, but representable
  - Tree structure represents combinations

AND / OR in D-Tree

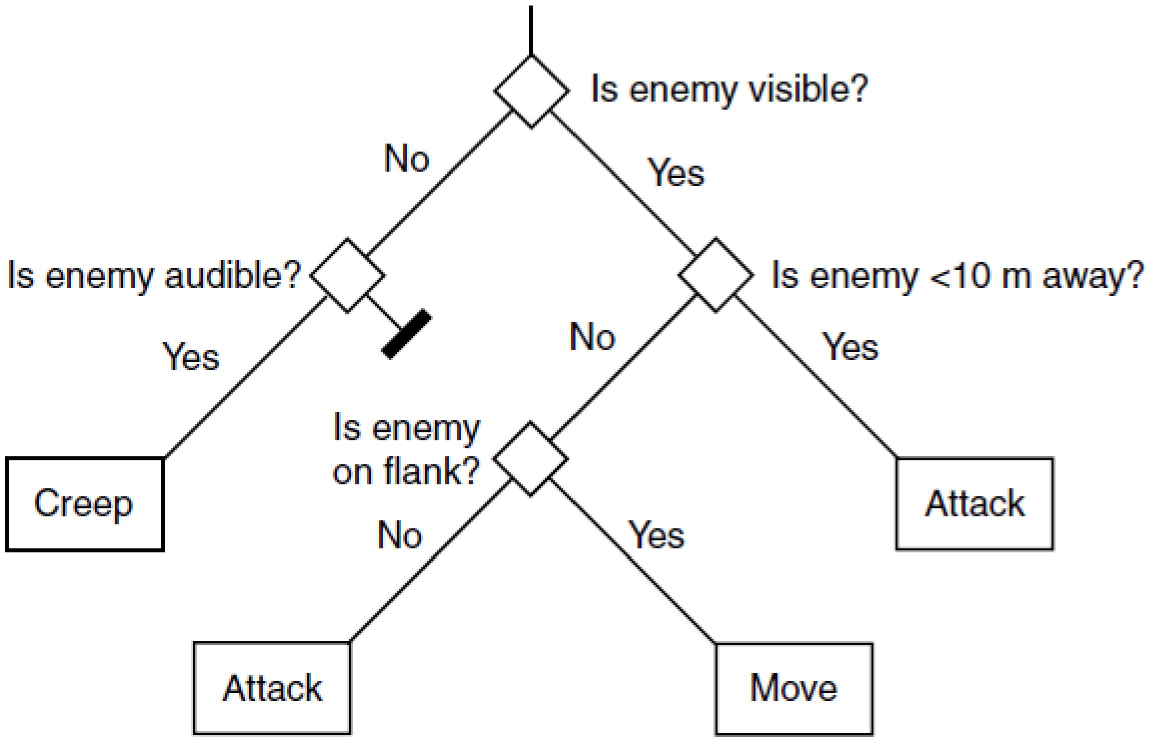


Can these be translated into rules? If so, how?

D-Tree Decisions

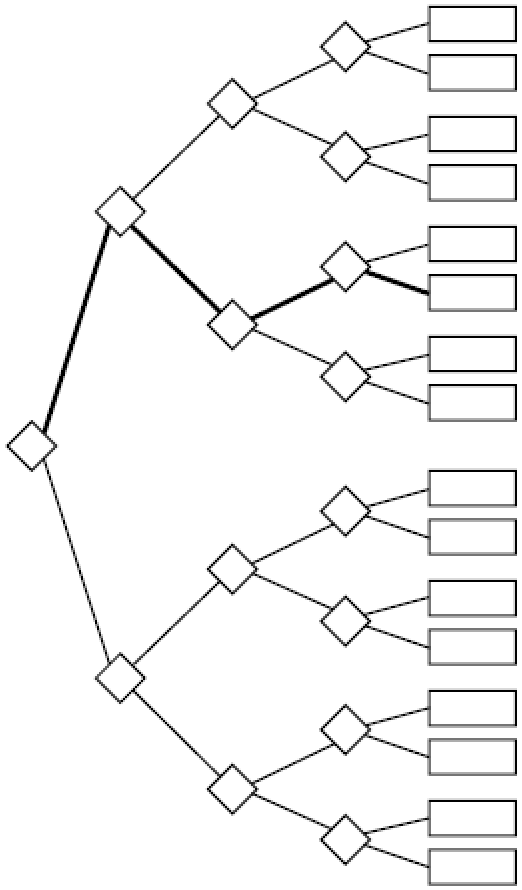
Enemy Visible OR Audible?
…or…
Enemy NOT Visible AND Audible?

- No explicit AND or OR, but representable
  - A AND B: serial TRUE decisions:
    - A?->TRUE->B?->TRUE
  - A OR B: TRUE if either of:
    - A TRUE (and B TRUE or FALSE)
    - A ?FALSE->B? TRUE
  - Tree structure represents combinations
  - **Lack of compound Boolean sentences is more a convention, as granularity of decisions has benefits for automated restructuring tree later**



Decision Complexity and Efficiency

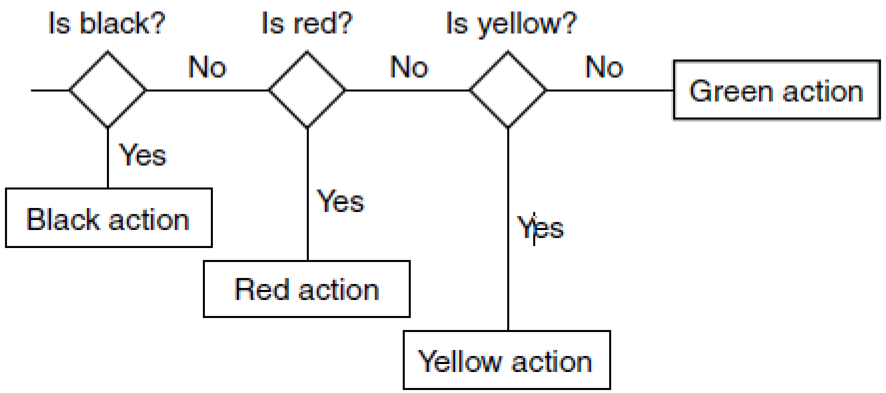
- Tree structure affords shared condition evaluation
  - Number of decisions in tree usually much smaller than number of decisions in tree
  - E.g. 15 different decisions w/ 16 actions, but only 4 considered
- This insight exploited by RETE (later)
- Must tree be binary?

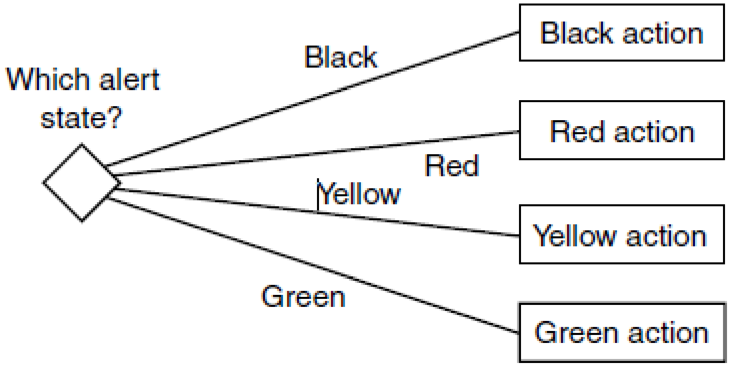


M&F 5.6

Branching

- N-ary trees
  - Usually ends up as if/then statements
  - Can be faster if using enums w/ array access
  - Speedup often marginal & not worth the effort
- Binary trees
  - Easier to optimize
  - ML techniques typically require binary trees
  - Can be a graph, **so long as it's a DAG**





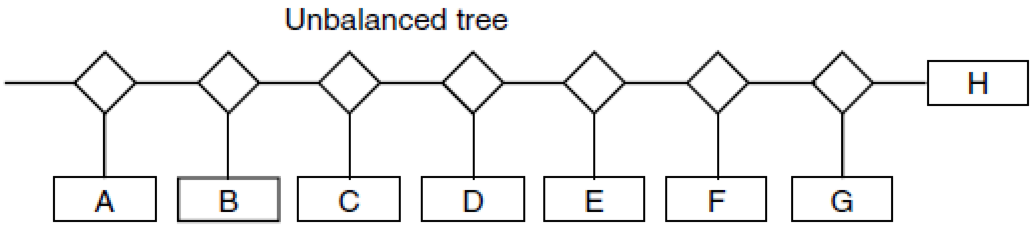
M&F Figs 5.7, 5.8

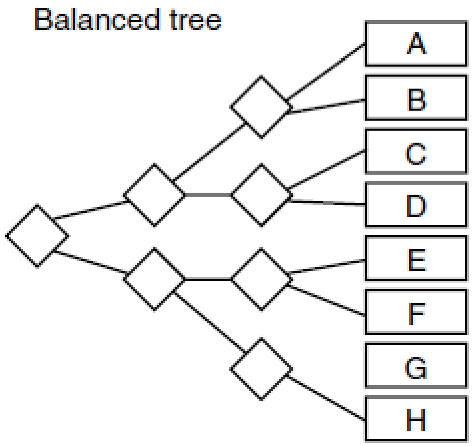
Knowledge Representation

- Typically work directly w/ primitive types
- Requires no translation of knowledge
  - Access game state directly
  - Since whole tree isn't evaluated, expensive to query knowledge can be *lazy/on-demand* for performance improvement (consider in comparison to rule based system)
  - Can cause HARD-TO-FIND bugs
    - Rare decisions → when do pop up, weird effects
    - Structure of game-state changes → breaks things
  - Cons avoidable w/ careful world interface
    - See Millington CH 10

Tree Balancing

- More balanced → faster (theory)
  - Balance ~= same number of leaves on each branch
  - O(N) vs O(Log2 N)
- Short path to likely action → faster (practice)
  - O(1)
  - Defer time consuming decisions 'til last
- Performance tuning
  - Dark art – since fast anyway, rarely important
  - Balance, but keep common paths short & bury long decisions





M&F Fig 5.9

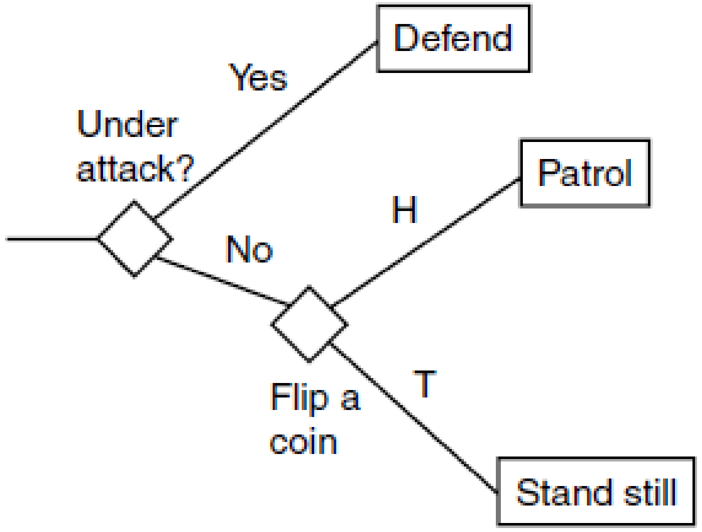
See M Ch 5.2

```
class DecisionTreeNode:
    def makeDecision() #recursively walk tree

class Action:
    def makeDecision():
        return this

class FloatDecision(Decision):
    minValue
    maxValue
    def getBranch():
        if max >= test >= min:
            return trueNode
        else:
            return falseNode
```

```
class Decision(DecisionTreeNode):
    trueNode
    falseNode
    testValue
    def getBranch()
    def makeDecision() :
        branch = getBranch() #runs test
        return branch.makeDecision() #recursive walk
```

Randomness

- Predictable == bad
- Can add a random decision node
  - random behavior choice adds unpredictability, interest, and variation
- Keep track of decision from last cycle
  - Random choice made at every frame can make unstable behavior
  - Add timeout so behavior can change
- See M 5.2.10 for implementation deets



M&F 5.12

D-Trees VS FSMs?

- Decision tree: same set of decisions is always used. Any action can be reached through the tree.
  - Root to leaf every time
- FSM: only transitions from the current state are considered. Not every action can be reached.
  - FSM update function called (each frame, or based on transition condition)
  - If transition “triggered”, schedule for “fire” the associated actions (onExit, transition action, onEnter

Learning Decision Trees

- Real power of D-trees comes from learning
- Problem: Construct a decision tree from examples of inputs and actions
- Sol'n: Quinlan's "Induction of Decision Trees"
  - ID3, C4.5, See5
    - http://en.wikipedia.org/wiki/ID3_algorithm
  - J48 (GPL java implementation)
    - http://www.opentox.org/dev/documentation/components/j48
    - See Weka (GNU GPL)

Andrew Ng – The State of AI (December 15, 2017)

- "99% of the economic value created by AI today is through one type of AI: which is learning a mapping A → B, or input to output maps"
  - Falls under category of supervised learning
- Other types (ordered falloff)
  - Transfer learning
  - Unsupervised learning
  - Reinforcement learning

Input	Output
Picture	Is it you? (0/1)
Loan application	Will the applicant repay the loan? (0/1)
Online: (Ad, User)	Will you click? (0/1)
Voice input	Text transcript
English	French
Car: image, radar/lidar	Positions of other cars

https://www.youtube.com/watch?v=NKpuX_yzdYs

Learning Decision Trees

- A simple technique whereby the computer learns to predict human decision-making
- Can also be used to learn to classify
  - A decision can be thought of as a classification problem
- An object or situation is described as a set of attributes
  - Attributes can have discrete or continuous values
- Predict an outcome (decision or classification)
  - Can be discrete (classification) or continuous (regression)
  - We assume positive (true) or negative (false)

Learned D-tree: how well do they work?

- Many case studies have shown that decision trees are at least as accurate as human experts.
  - study for diagnosing breast cancer had humans correctly classifying the examples 65% of the time; the decision tree classified 72% correct
  - British Petroleum designed a decision tree for gas-oil separation for offshore oil platforms that replaced an earlier rule-based expert system
  - Cessna designed an airplane flight controller using 90,000 examples and 20 attributes per example

https://www.cc.gatech.edu/~bboots3/CS4641-Fall2016/Lectures/Lecture2.pdf

Basic Concept

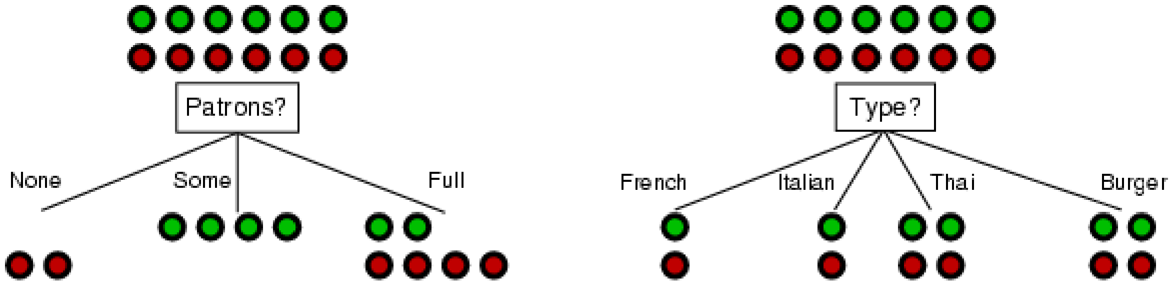
- Given the current set of decisions, what attribute can best split them?
- Choose the “best one” and create a new decision node
  - Best == most information gained == smallest entropy
  - Keeps tree small
- Good attributes make homogeneous sets
- Recursively go down each edge

Example

Example	Attributes										Target
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0-10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30-60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0-10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10-30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0-10	T
X_7	F	T	F	F	None	$	T	F	Burger	0-10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0-10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10-30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0-10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	10-60	T

Choosing an Attribute

- Idea: A good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”

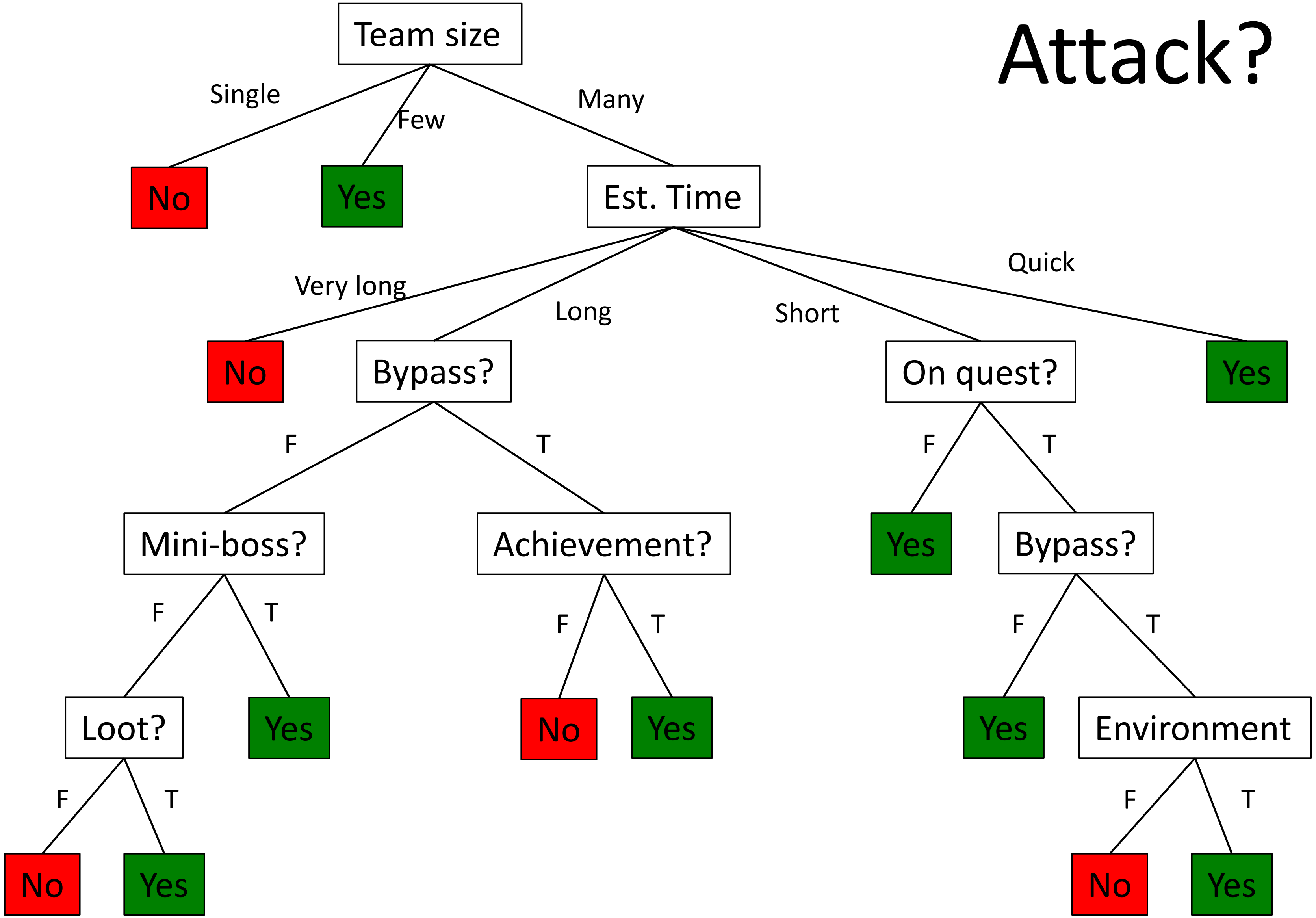


- *Patrons?* is a better choice

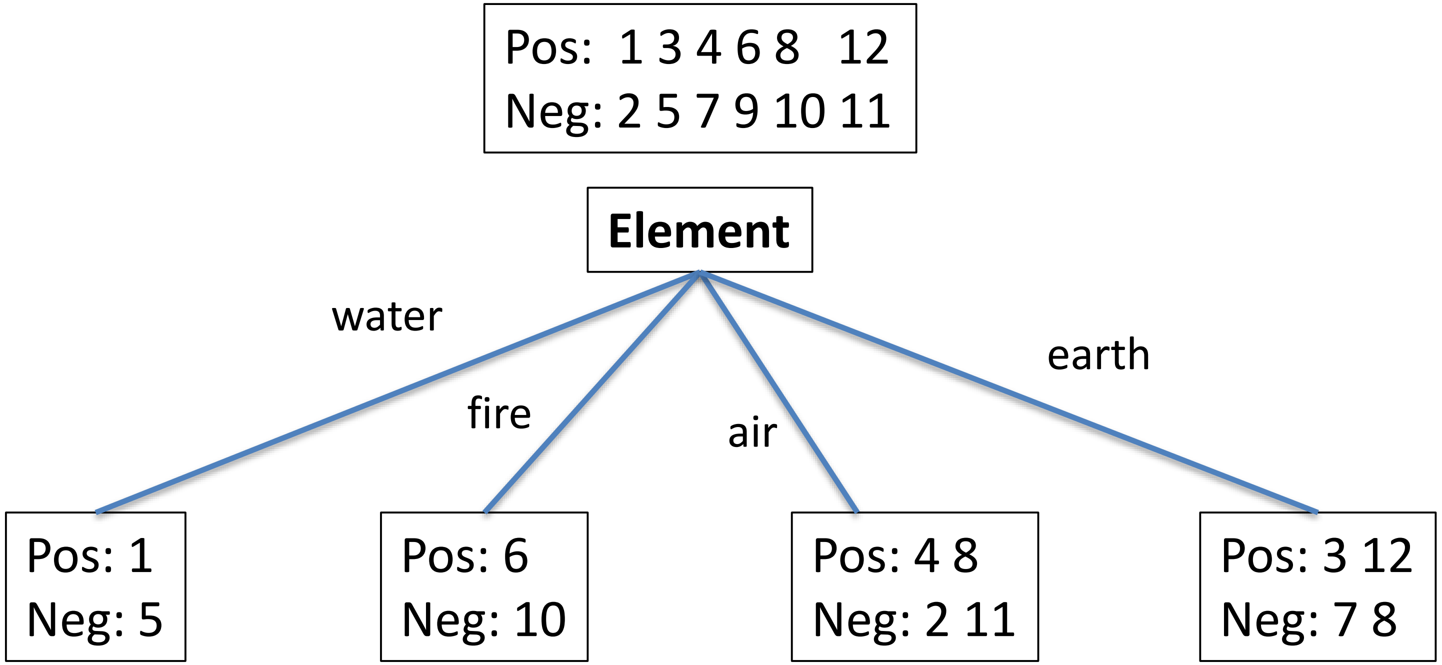
Attack?

- Attributes:
  - Bypass? Can be bypassed
  - Loot? Has valuable items/treasure
  - Achievement? Will unlock an achievement if you win
  - On Quest? You are on a quest
  - Experience. How much experience points you get
  - Environment. How favorable is the terrain?
  - Mini-boss? Is this a mini-boss, preventing further progress?
  - Element. The elemental properties (earth, air, fire, water)
  - Estimated Time. How long will this combat take (quick, short, long, very long)?
  - Team size. How many monsters in the team (none, small, large)?

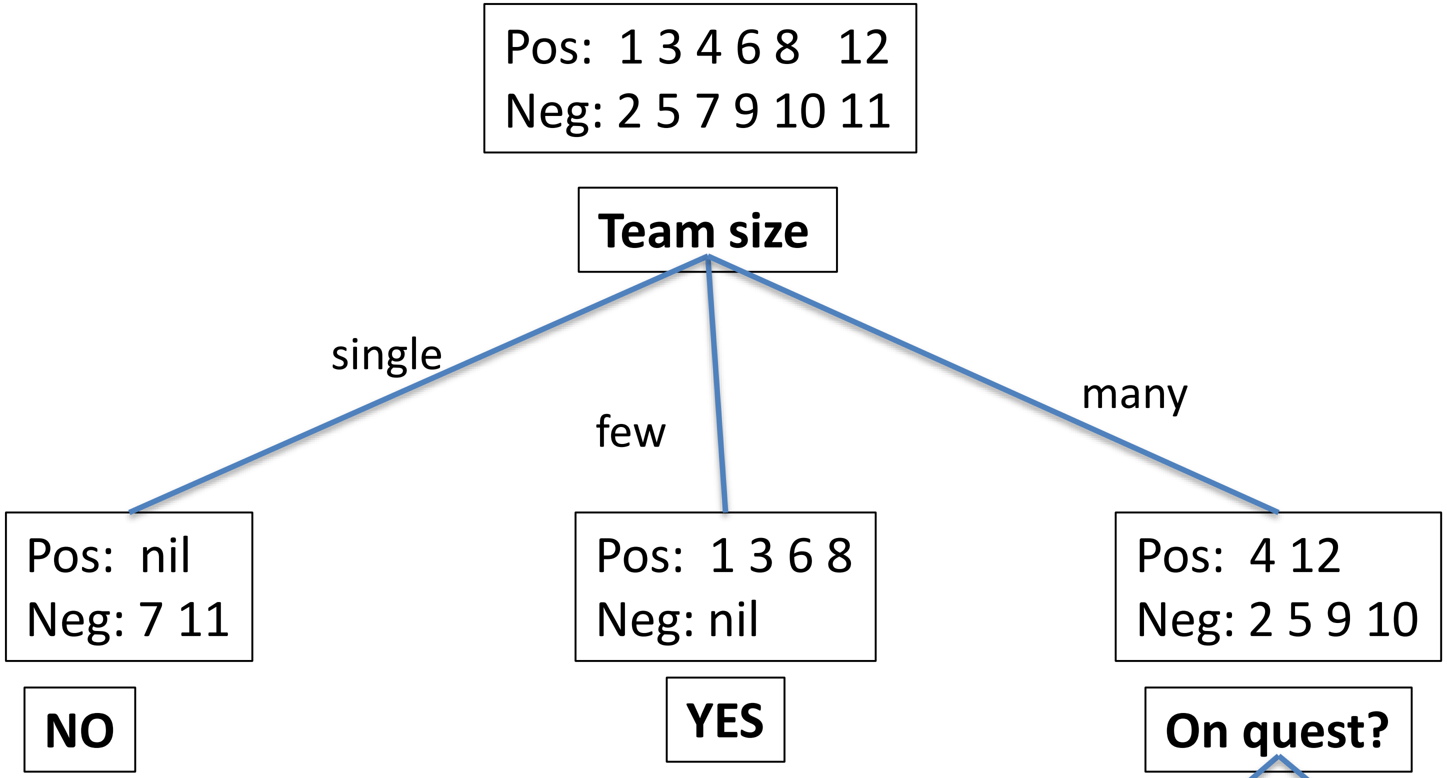
Attack?

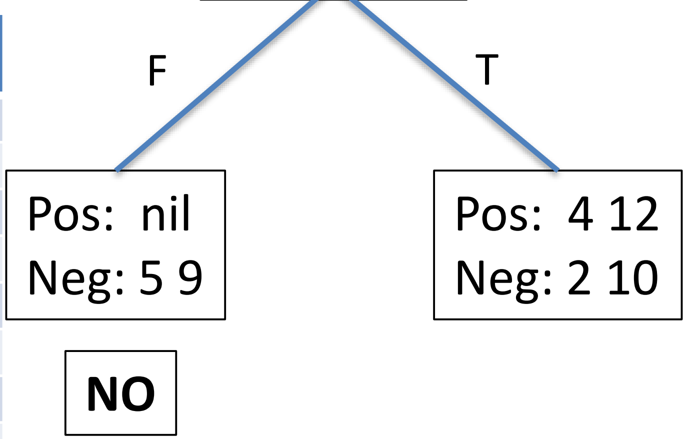

Team size
Single
Few
Many
No
Yes
Est. Time
Very long
Long
Short
Quick
No
Bypass?
On quest?
Yes
F
T
F
T
Mini-boss?
Achievement?
Yes
Bypass?
F
T
F
T
F
T
Loot?
Yes
No
Yes
Yes
Environment
F
T
F
T
No
Yes
No
Yes

#	Bypass?	Loot?	Achieve.	On quest	Team size	Exp.	Env.	Mini-Boss	Element	Est. Time	Attack?
1	T	F	F	T	few	Lot	Bad	T	water	quick	**Y**
2	T	F	F	T	many	Little	Bad	F	air	long	**N**
3	F	T	F	F	few	Little	Bad	F	earth	quick	**Y**
4	T	F	T	T	many	Little	Bad	F	air	med	**Y**
5	T	F	T	F	many	Lot	Bad	T	water	v. long	**N**
6	F	T	F	T	few	Med	Good	T	fire	quick	**Y**
7	F	T	F	F	single	Little	Good	F	earth	quick	**N**
8	F	F	F	T	few	Med	Good	T	air	quick	**Y**
9	F	T	T	F	many	Little	Good	F	earth	v. long	**N**
10	T	T	T	T	many	Lot	Bad	T	fire	med	**N**
11	F	F	F	F	single	Little	Bad	F	air	quick	**N**
12	T	T	T	T	many	Little	Bad	F	earth	long	**Y**



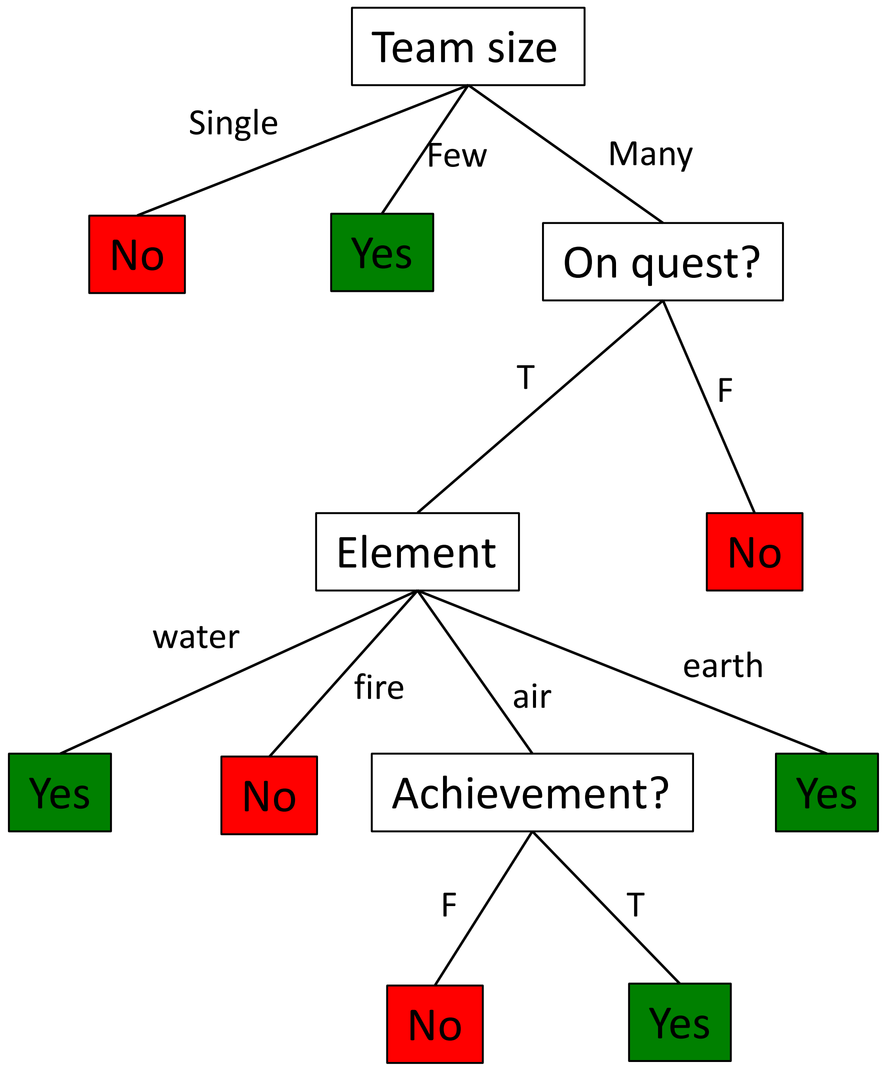
#	Bypass?	Loot?	Achieve.	On quest	Team size	Exp.	Env.	Mini-Boss	Element	Est. Time	Attack?
1	T	F	F	T	few	Lot	Bad	T	**water**	quick	**Y**
2	T	F	F	T	many	Little	Bad	F	**air**	long	**N**
3	F	T	F	F	few	Little	Bad	F	**earth**	quick	**Y**
4	T	F	T	T	many	Little	Bad	F	**air**	med	**Y**
5	T	F	T	F	many	Lot	Bad	T	**water**	v. long	**N**
6	F	T	F	T	few	Med	Good	T	**fire**	quick	**Y**
7	F	T	F	F	single	Little	Good	F	**earth**	quick	**N**
8	F	F	F	T	few	Med	Good	T	**air**	quick	**Y**
9	F	T	T	F	many	Little	Good	F	**earth**	v. long	**N**
10	T	T	T	T	many	Lot	Bad	T	**fire**	med	**N**
11	F	F	F	F	single	Little	Bad	F	**air**	quick	**N**
12	T	T	T	T	many	Little	Bad	F	**earth**	long	**Y**





#	Bypass?	Loot?	Achieve.	On quest	Team size	Exp.	Env.	Mini-Boss	Element	Est. Time	Attack?
1	T	F	F	T	**few**	Lot	Bad	T	water	quick	**Y**
2	T	F	F	T	**many**	Little	Bad	F	air	long	**N**
3	F	T	F	F	**few**	Little	Bad	F	earth	quick	**Y**
4	T	F	T	T	**many**	Little	Bad	F	air	med	**Y**
5	T	F	T	F	**many**	Lot	Bad	T	water	v. long	**N**
6	F	T	F	T	**few**	Med	Good	T	fire	quick	**Y**
7	F	T	F	F	**single**	Little	Good	F	earth	quick	**N**
8	F	F	F	T	**few**	Med	Good	T	air	quick	**Y**
9	F	T	T	F	**many**	Little	Good	F	earth	v. long	**N**
10	T	T	T	T	**many**	Lot	Bad	T	fire	med	**N**
11	F	F	F	F	**single**	Little	Bad	F	air	quick	**N**
12	T	T	T	T	**many**	Little	Bad	F	earth	long	**Y**

- Learned from the 12 examples
- Why doesn't it look like the previous tree?
  - Not enough examples
  - No reason to use environment or mini-boss
  - Hasn't seen all cases
- Learning is only as good as your training data
- Supervised learning
  - Training set
  - Test set



Which attribute to choose?

- The one that gives you the most information (aka the most diagnostic)
- Information theory
  - Answers the question: how much information does something contain?
  - Ask a question
  - Answer is information
  - Amount of information depends on how much you already knew (information gain)
- Example: flipping a coin

Entropy

- Measure of information in set of examples
  - That is, amount of agreement between examples
  - All examples are the same, E = 0
  - Even distributed and different, E = 1
- If there are n possible answers, $v_1 \ldots v_n$ and v_i has probability $P(v_i)$ of being the right answer, then the amount of information is:

$$H\left(P(v_1),\ldots,P(v_n)\right) = -\sum_{i=1}^{n} P(v_i)\log_2 P(v_i)$$

- For a training set:

  p = # of positive examples

  n = # of negative examples

$$H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Probability of a positive example

Probability of a negative example

- For our attack behavior
  - p = n = 6
  - H() = 1
  - Would not be 1 if training set weren't 50/50 yes/no, but the point is to arrange attributes to increase gain (decrease entropy)

Pos: 1 3 4 6 8 12
Neg: 2 5 7 9 10 11

Measuring attributes

- Remainer(A) is amount of entropy remaining after applying an attribute
  - If I use attribute A next, how much less entropy will I have?
  - Use this to compare attributes

Instances of the attribute

Positive examples for this answer to A

Negative examples for this answer to A

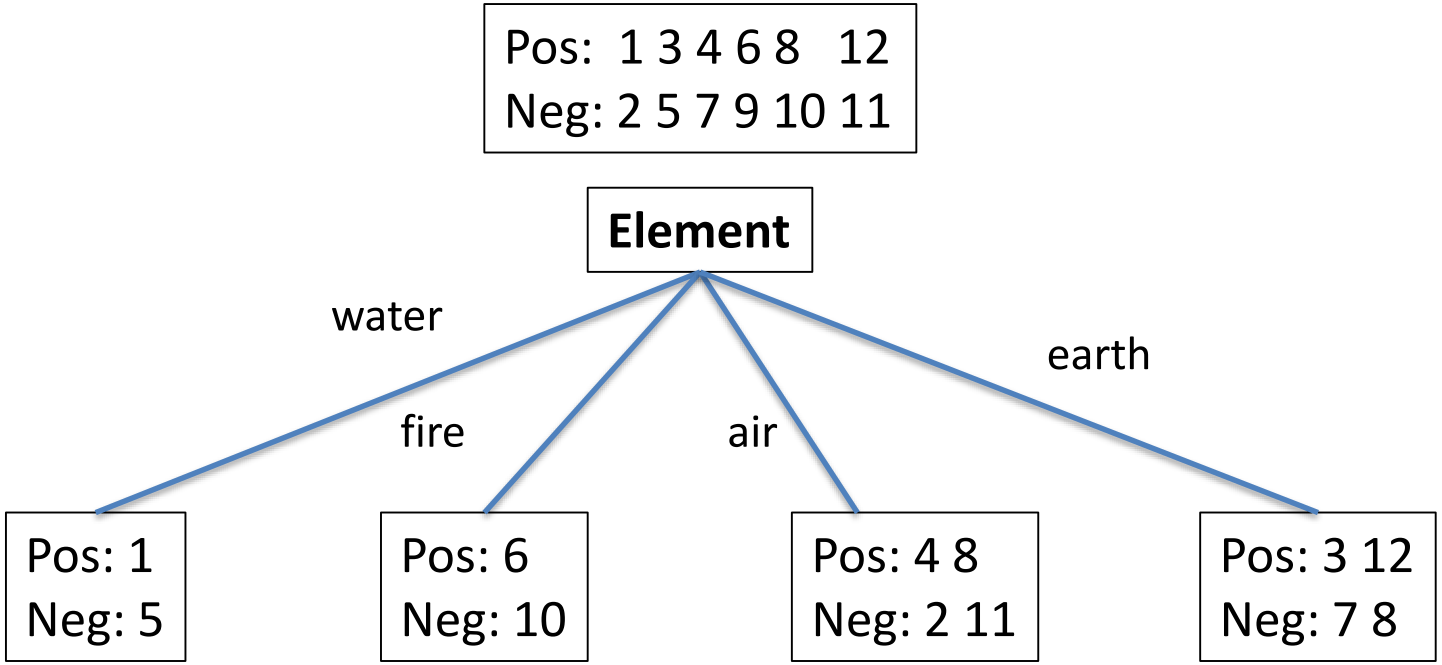
$$\text{Remainder(A)} = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} H\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

attribute

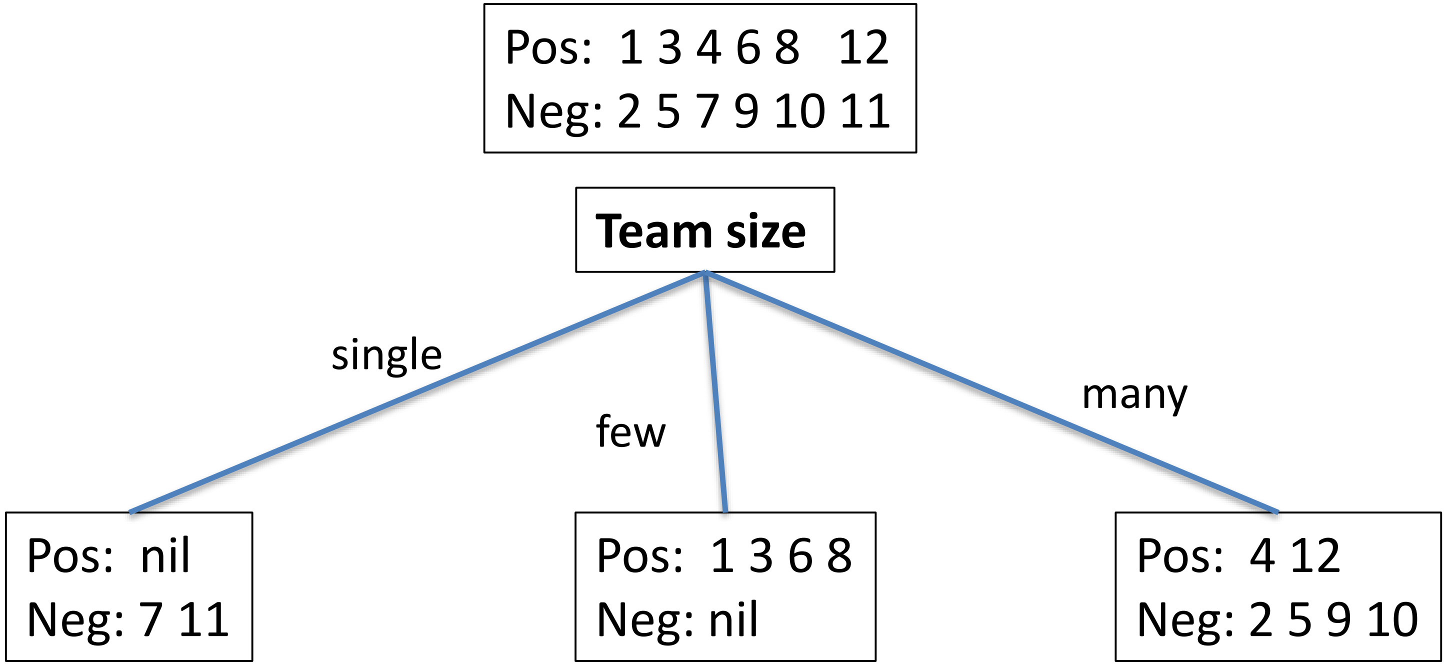
Different answers

Total answers

Examples classified by A



$$\text{Remainder(element)} = \underbrace{\frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right)}_{\text{water}} + \underbrace{\frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right)}_{\text{fire}} + \underbrace{\frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right)}_{\text{air}} + \underbrace{\frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right)}_{\text{earth}} = 1 \text{ bit}$$



$$\text{Remainder(teamsize)} = \frac{2}{12}\underbrace{I\left(\frac{0}{2},\frac{2}{2}\right)}_{\text{single}} + \frac{4}{12}\underbrace{I\left(\frac{4}{4},\frac{0}{4}\right)}_{\text{few}} + \frac{6}{12}\underbrace{I\left(\frac{2}{6},\frac{4}{6}\right)}_{\text{many}} \approx 0.459 \text{ bit}$$

- Not done yet
- Need to measure information **gained** by an attribute

$$\text{Gain}(A) = H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{remainder}(A)$$

- Pick the biggest
- Example:
  - Gain(element) = H(½,½) $- \left(\frac{2}{12} H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} H\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} H\left(\frac{2}{4}, \frac{2}{4}\right)\right)$

    = 0 bits

  - Gain(teamsize) = H(½,½) $- \left(\frac{2}{12} H\left(\frac{0}{2}, \frac{2}{2}\right) + \frac{4}{12} H\left(\frac{4}{4}, \frac{0}{4}\right) + \frac{6}{12} H\left(\frac{2}{6}, \frac{4}{6}\right)\right)$

    ≈ 0.541 bits

Pos: 1 3 4 6 8 12
Neg: 2 5 7 9 10 11

Team size

Many

Pos: 4 12
Neg: 2 5 9 10

On Quest

F

Pos: nil
Neg: 5 9

T

Pos: 4 12
Neg: 2 10

teamsize=many, onquest=T

teamsize=many, onquest=F

Gain(quest) = $H\left(\frac{2}{12},\frac{4}{12}\right)-\left[\frac{2}{12}H\left(\frac{0}{2},\frac{2}{2}\right)+\frac{4}{12}H\left(\frac{2}{4},\frac{2}{4}\right)\right]$

no

yes

= 0.959 – [0 + (4/12)(1)]

≈ 0.626 bits

Decision-tree-learning (examples, attributes, default)

IF examples is empty THEN RETURN default

ELSE IF all examples have same classification THEN RETURN classification

ELSE IF attributes is empty RETURN majority-value(examples)

ELSE

    best = choose(attributes, example) ← Where gain happens

    tree = new decision tree with best as root

    m = majority-value(examples)

    FOREACH answer v_i of best DO

        $examples_i$ = {elements of examples with best=v_i}

        $subtree_i$ = decision-tree-learning($examples_i$, attributes-{best}, m)

        add a branch to tree based on v_i and $subtree_i$

    RETURN tree

How many hypotheses?

- How many distinct trees?
  - N attributes

    = # of boolean functions

    = # of distinct truth tables with 2^n rows

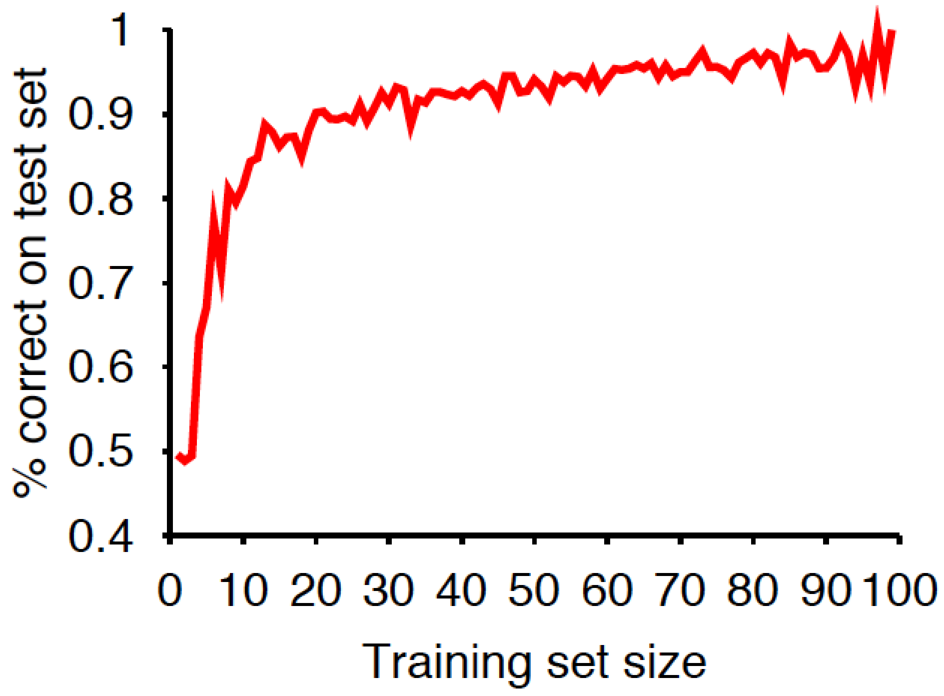
    = 2^2^n
  - With 6 attributes: > 18 quintillion possible trees
    - 18,446,744,073,709,551,616

How do we assess?

- How do we know hypothesis ≈ true decision function?
- A learning algorithm is good if it produces hypotheses that do a good job of predicting decisions/classifications from unseen examples

1. Collect a large set of examples (with answers)
2. Divide into training set and test set
3. Use training set to produce hypothesis h
4. Apply h to test set (w/o answers)
   - Measure % examples that are correctly classified
5. Repeat 2-4 for different sizes of training sets, randomly selecting examples for training and test
   - Vary size of training set m
   - Vary which m examples are training

- Plot a learning curve
  - % correct on test set, as a function of training set size



- As training set grows, prediction quality should increase
  - Called a “happy graph”
  - There is a pattern in the data AND the algorithm is picking it up!

Noise

- Suppose 2 or more examples with same description (Same assignment of attributes) have different answers
- Examples: on two identical* situations, I do two different things
- You can’t have a consistent hypothesis (it must contradict at least one example)
- Report majority classification or report probability

Overfitting

- Learn a hypothesis that is consistent using irrelevant attributes
  - Coincidental circumstances result in spurious distinctions among examples
  - Why does this happen?
    - You gave a bunch of attributes because you didn't know what would be important
    - If you knew which attributes were important, you might not have had to do learning in the first place
- Example: Day, month, or color of die in predicting a die roll
  - As long as no two examples are identical, we can find an exact hypothesis
  - Should be random 1-6, but if I roll once every day and each day results in a different number, the learning algorithm will conclude that day determines the roll
- **Applies to all learning algorithms**

Black and White

http://www.ign.com/games/black-and-white

Black and White

- Creature must learn what to do in different situations
- Player can reward or punish the creature
  - Tells the creature whether they made the right choice of action or not
- Creature learns to predict the feedback it will receive from the player

Example	**Attributes**			**Target**
	Allegiance	Defense	Tribe	Feedback
D1	Friendly	Weak	Celtic	-1.0
D2	Enemy	Weak	Celtic	0.4
D3	Friendly	Strong	Norse	-1.0
D4	Enemy	Strong	Norse	-0.2
D5	Friendly	Weak	Greek	-1.0
D6	Enemy	Medium	Greek	0.2
D7	Enemy	Strong	Greek	-0.4
D8	Enemy	Medium	Aztec	0.0
D9	Friendly	Weak	Aztec	-1.0

Continuous DTs must discretize the variables by deciding where to split the continuous range.

No Free Lunch

- ID3
  - Must discretize continuous attributes
  - Offline only (online = adjust to new examples)
  - Too inefficient with many examples
- Incremental methods (C4.5, See5, ITT, etc)
  - Starts with a d-tree
  - Each node holds examples that reach that node
  - Any node can update self given new example
  - Can be unstable (new trees every cycle; rare in practice)

But first…

- “What Makes Good AI – Game Maker’s Toolkit”
  - https://www.youtube.com/watch?v=9bbhJi0NBkk&t=0s
  - https://www.patreon.com/GameMakersToolkit
  - React/adapt to the player – no learning required (authoring is)
  - Communicate what you’re thinking
  - Illusion of intelligence; more health & aggression can be a proxy for smarts
  - Predictability is (usually) a good thing
    - Too much NPC stupidity can ruin an otherwise good game

Next Class

- More decision making!
  - Behavior trees
  - Production / Rule Based systems
  - Fuzzy logic + probability
  - Planning

BEHAVIOR TREES (M CH. 5.4)

Next Class

- More decision making!
  - Behavior trees
  - Production / Rule Based systems
  - Fuzzy logic + probability
  - Planning