UNIVERSITÉ Concordia UNIVERSITY

COMP 476

Advanced Game Development

Session 6
Learning Algorithms

(Reading: AI for G, Millington § 7.1-7.3, 7.4-7.8)

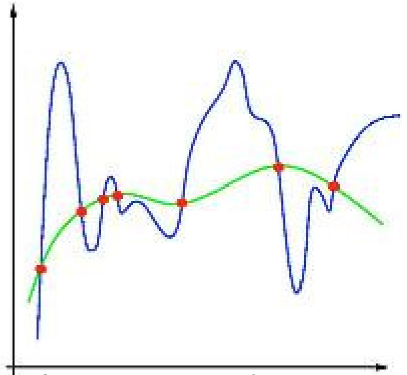
Lecture Overview

- ❑ Types of Learning
- ❑ Parameter Modification
- ❑ Action Prediction
- ❑ Decision Learning

Learning

- **Purpose/benefits of Learning**
  - ***More believable*** **(less predictable) characters**
  - ***Less work to program*** **decisions for all the possible situations**

- **Issues**
  - **Challenges using learning in games?**
  - ***Lack of control*** **over learnt behaviors**
  - ***Over-fitting or over-learning***
    - **performs poorly on new examples as it is too highly trained to noise instead of the actual relationship**



Types of Learning

- Offline
  - **Between levels of games**
  - **At the game development studio before the game is released**
    - **Most common approach**
    - ***Allows for testing* of the learnt behaviors**
- Online
  - **During the game itself**

Types of Learning

- **Intra-behavior learning: Adaptation of the behavior parameters**
  - ***Learning to target precisely***
  - **Learning *the best patrol routes* on the current level**
  - **Learning *good set of cover points* for the given room**
  - ***etc.***

Types of Learning

- ***Predicting the behavior of the enemy***
  - **Predicting the next move in a fight *based on the last few moves***
  - **Predicting the attack *based on the assembly of enemy troops***
  - **Predicting where players will be**
  - **Predicting what weapons players will use**
  - ***etc.***

Types of Learning

- Inter-behavior learning: ***Reacting to the behavior of the enemy***
  - **Making the most effective counter-move in a fight**
  - **Making the most effective reallocation of the troops**
  - **Making the best move in a board game based on all the pieces**
  - ***etc.***

Learning Algorithms

- Following are some of the learning algorithms used in games:
  - Parameter Modification
    - Hill Climbing
  - Action Prediction
    - N-Grams
  - Decision Learning
    - Naïve Bayes Classifiers
    - Decision Tree Learning
    - Reinforcement Learning
    - Artificial Neural Networks

(§7.2)

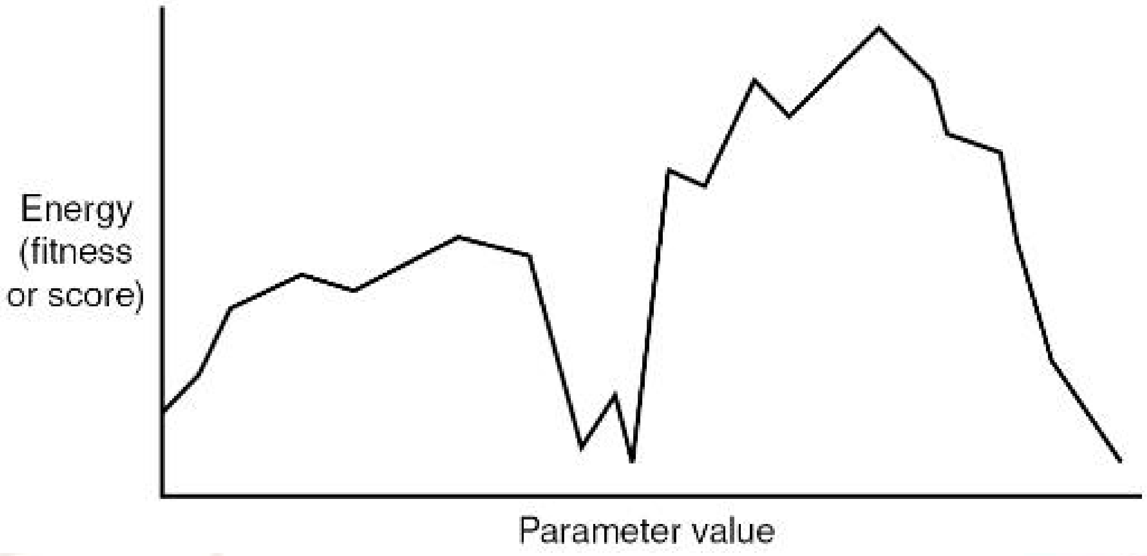
Parameter Modification

Parameter Modification

- The simplest learning algorithms are those *that calculate the value of one or more parameters*.
- *Numerical parameters* are those magic numbers that are used in steering calculations, cost functions for pathfinding, weights for blending tactical concerns, probabilities in decision making, and many other areas.
- These values can often have a large effect on the behavior of a character. A small change in a decision making probability, for example, can lead an AI into a very different style of play.
- Most commonly, this is done offline.

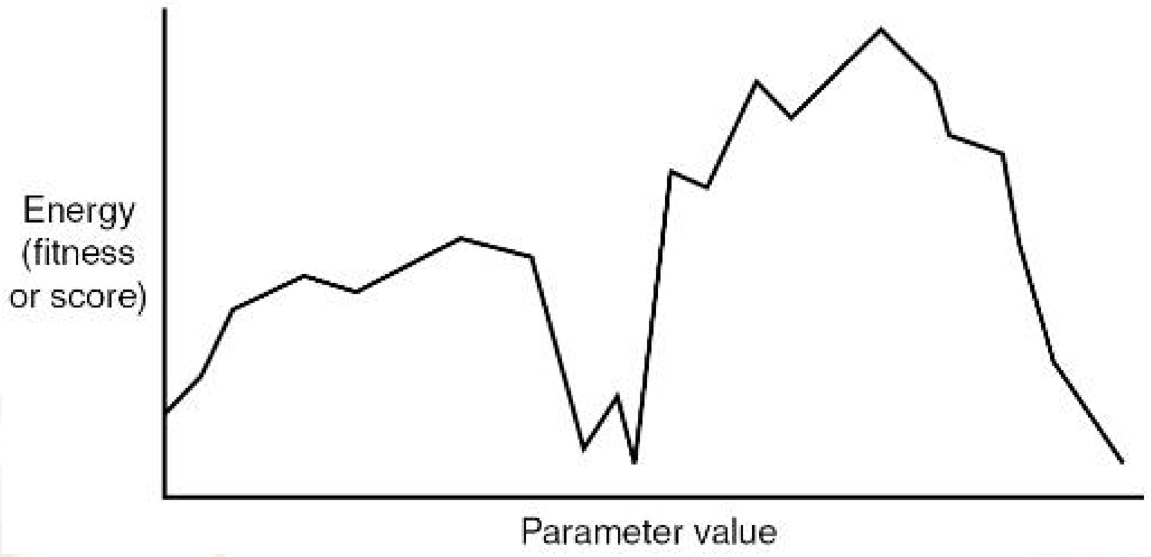
Parameter Landscape

- A common way of understanding the parameter learning is "fitness" or "energy" landscape.
- Imagine the value of the parameter as specifying a location. In the case of a single parameter this is *a location somewhere along a line*. For two parameters it is the location on a plane, *etc*.



Parameter Landscape

- For each location (*i.e.*, for each value of the parameter) there is some energy value. This energy value (often called a "fitness value" in some learning techniques) ***represents how good the value of the parameter is for the game***.
  - You can think of it as a score.



Parameter Adaptation

Hill Climbing

- The goal is to find the *parameter vector Ψ^* that results in the most optimal value* of the function:

$$\Psi^* = argmax_{\Psi}\ f(\Psi)$$

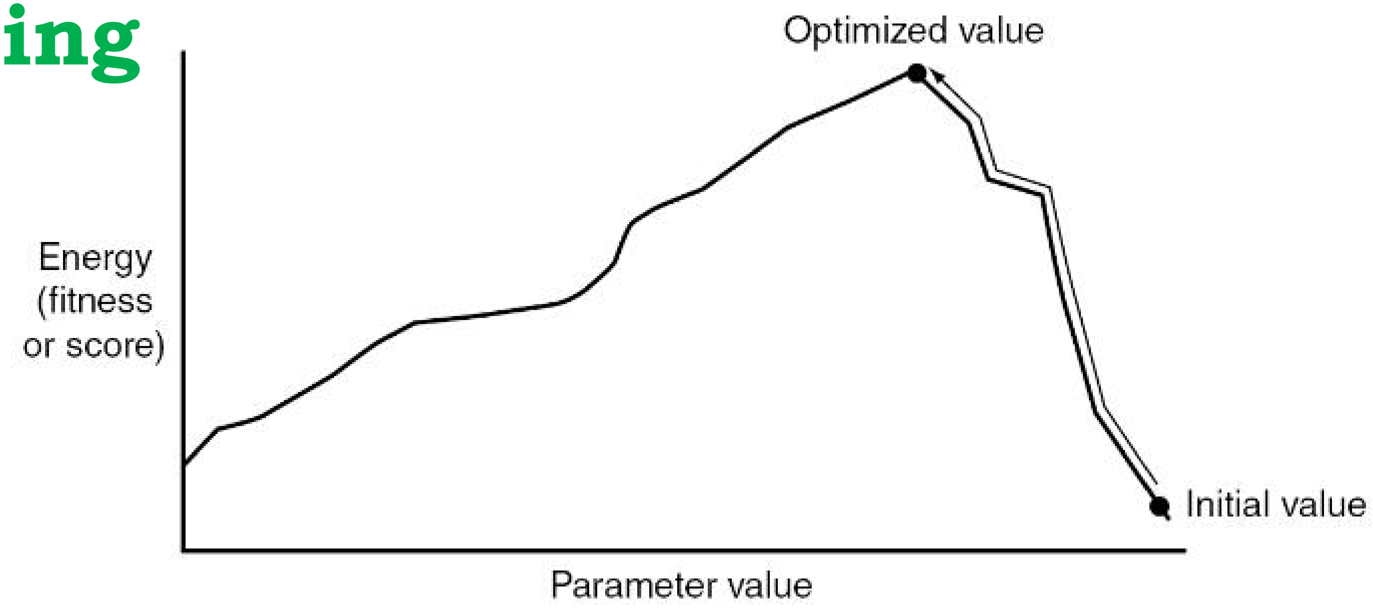
- General Approach:
  - Start with the **best guess** for the parameter vector Ψ
  - **Until no further improvement** try changing Ψ by small Δ in all directions and pick the best:

$$\Psi = \Psi + argmax_{\Delta}\ f(\Psi + \Delta)$$

- **Runs fast**; often gives very good results

Parameter Adaptation

Hill Climbing



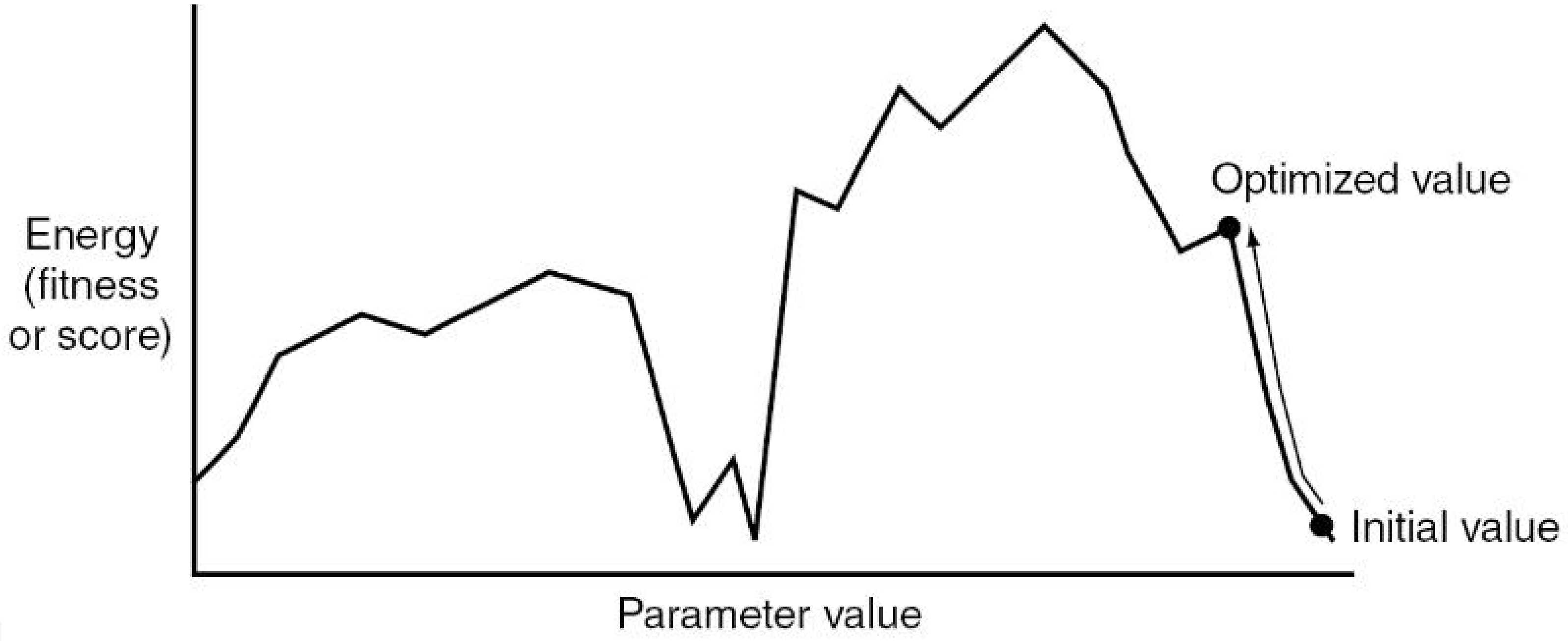
- **Equivalent formulation for differentiable functions (Steepest ascent):**
- ***Start with the best guess*** **for the parameter vector** X
- ***Until no further improvement*** **do**

$$X = X + \varepsilon f'(X)$$

Parameter Adaptation

Extensions to Basic Hill Climbing

- Basic Hill Climbing ***can get caught up on a local maximum*** while searching for a global maximum



- Various extensions to basic hill climbing try to solve this problem.

Parameter Adaptation

Extensions to Basic Hill Climbing

Several Extensions:

- **Momentum - If the search consistently improves in one direction, then it should continue in that direction for a little while, even when it seems that things aren't improving any more.**
- **Adaptive Resolution - Make long jumps early in the search (while still improving) and smaller jumps later on.**
- **Multiple Trials - Use multiple different start values distributed across the whole landscape.**
- **All such approaches improve the results but none guarantee that we get the best solution.**

(§7.3)

Action Prediction

Action Prediction

- ***Predict future action*** **of the player** **based on past actions and anything else relevant**
- **N-gram predictor**
  - **Very popular in all the combat (martial arts, boxing, swords, …) games (e.g., predicting next move)**
  - ***Often requires reduction in the power of AI to make it beatable***
  - **Extends to other predictions such as what weapons will be used or how the attacks will occur**

N-gram Predictor

- Maintain the probabilities of future (*e.g.*, the Nth) actions based on N-1 preceding observations
- For prediction, always return the most likely action ***based on last N-1 observations***

N-gram Predictor

- **Example of applying 3-gram predictor (N = 3):**
  - **training data (observed sequence of n=15 moves):**

    LRRLRLLLRRLRLRR

  - **learnt prediction table:**

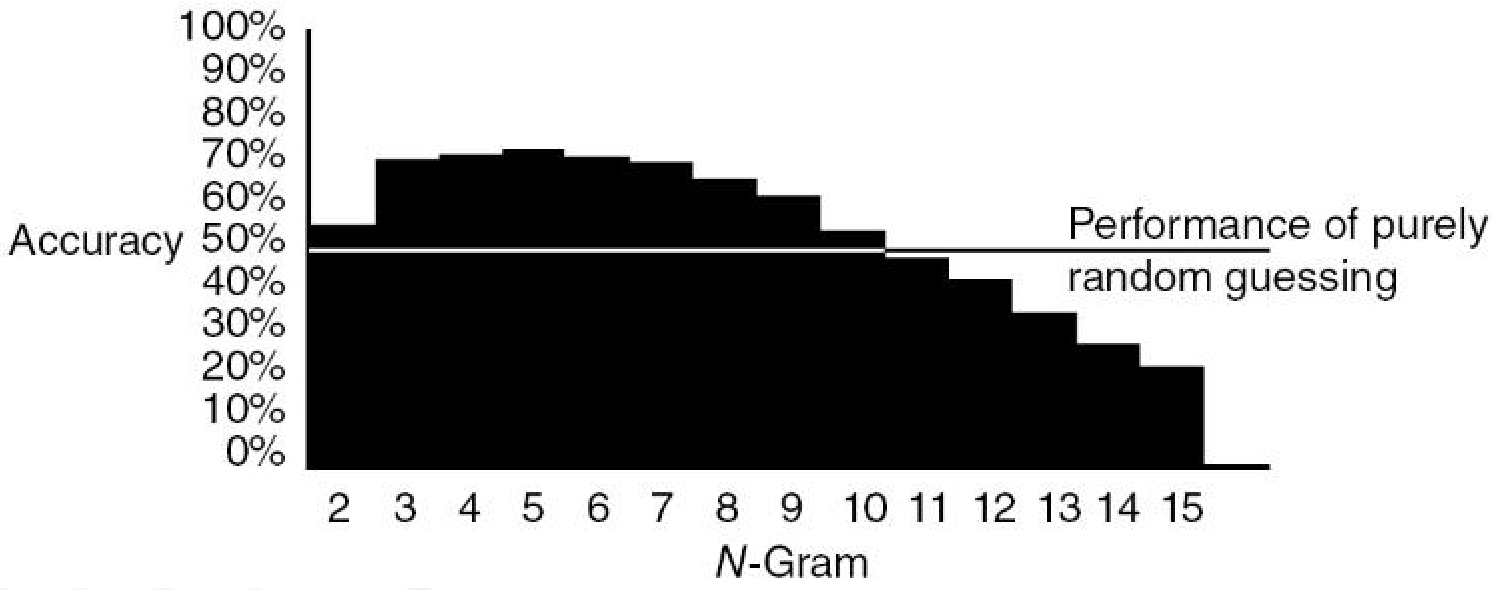
	..R	*..L*
LL	*1/2*	*1/2*
LR	*3/5*	*2/5*
RL	*3/4*	*1/4*
RR	*0/2*	*2/2*

Mathematically,
P(A=R|LL),
P(A=R|LR),
…

N-gram Predictor

Window Size

- **Accuracy of prediction** ***may reduce as value of N increases past some point***

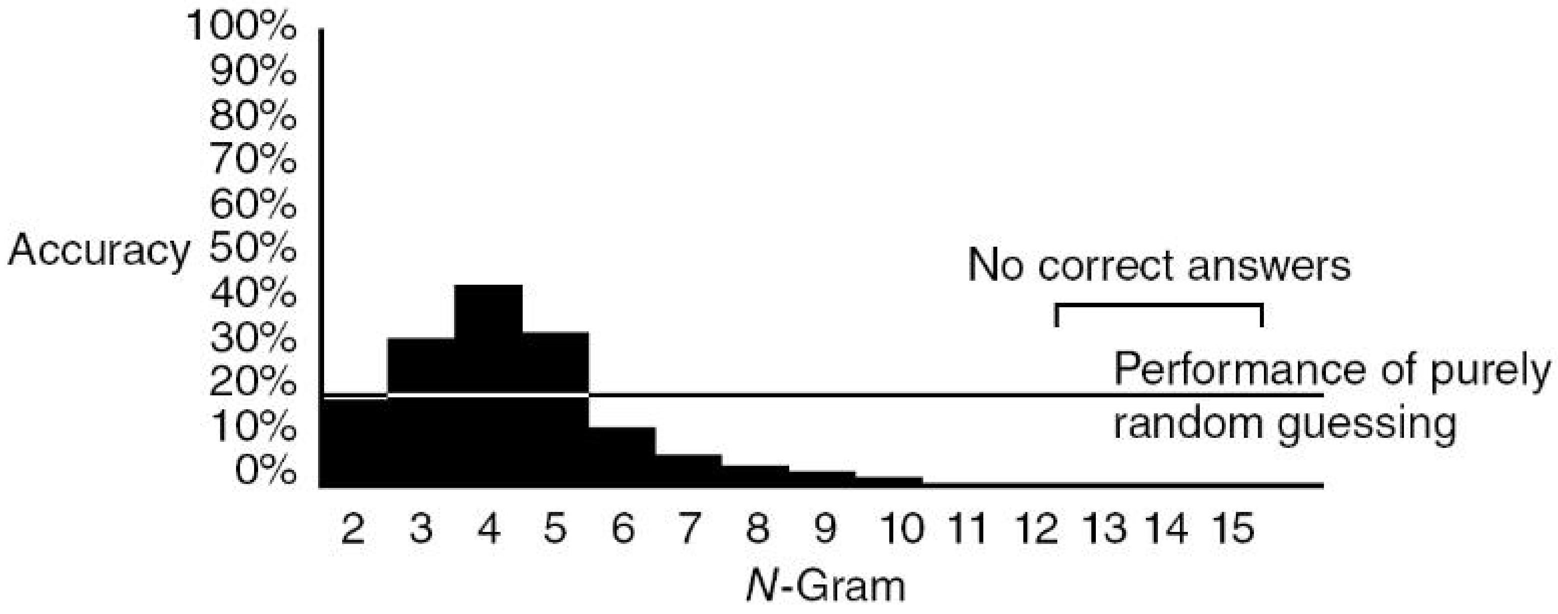


The accuracy of an N-Gram for different window sizes on a sequence of n=1,000 trials (for the left-or-right game)

N-gram Predictor

Window Size

- In predictions where there are more than two possible choices, the ***minimum window size needs to be increased a little***.



The predictive power in a five choice game

Hierarchical N-gram Predictor

- Game Performance in initial stages: what to do before the N-gram predictor is learnt?
  - Learn 1-gram, 2-gram, 3-gram, … predictors simultaneously
  - For prediction, pick *N-gram predictor* with *largest N and sufficient number of training samples* for the given input
  - If *none have sufficient samples*, then output random prediction

Hierarchical N-gram Predictor

- Training data (observed sequence of 15 moves):

LRRLRLLLRRLRLRR

1-gram Obs.	..R	..L	# of samples
—	8/15	7/15	15

2-gram Obs.	..R	..L	# of samples
L	5/7	2/7	7
R	3/7	4/7	7

3-gram Obs.	..R	..L	# of samples
LL	1/2	1/2	2
LR	3/5	2/5	5
RL	3/4	1/4	4
RR	0/2	2/2	2

Hierarchical N-gram Predictor

- What is the predicted next action for input=RRR
  - if we want at least 4 samples for prediction;
  - if we want at least 9 samples for prediction?

1-gram Obs.	..R	..L	# of samples
—	8/15	7/15	15

2-gram Obs.	..R	..L	# of samples
L	5/7	2/7	7
R	3/7	4/7	7

3-gram Obs.	..R	..L	# of samples
LL	1/2	1/2	2
LR	3/5	2/5	5
RL	3/4	1/4	4
RR	0/2	2/2	2

(§7.5-7.8)

Decision Learning

Decision Learning

- Previous learning approaches were in relatively restricted domains.
- How do we allow the AI to learn to make decisions?
- Simplification: we let the NPC choose from some set of behaviour options.
- To do this, we need to associate decisions with observations.

Supervised Learning

- Different approaches to learning can be classified by how much supervision is performed.
- **Supervision**: the amount of feedback provided by the learning algorithm
  - **Strong Supervision**: given *a set of correct answers* (typically a training set)
  - **Weak (or Semi-supervised)**: *some indication* is given *as to how good its action choices are*
  - **Unsupervised**: *no indication* is given as to how good its action choices are

Naïve Bayes Classifiers

- Scale easily for large # of input variables
- Very popular (and powerful) for machine learning problems
  - Good baseline for other learning techniques
- Assuming input variables $\{X_i = u_i\}$, $i = 1,\ldots,k$, the predicted action (from the set A of possible actions) will be:

(Hard to calculate)

$$A^{predict} = argmax_a\, P\{A = a \,|\, X_1 = u_1, \ldots, X_k = u_k\}$$

$$= \ldots$$

(Easy to calculate)

$$= argmax_a\, P\{A = a\} \prod_i P\{X_i = u_i \,|\, A = a\}$$

Naïve Bayes Classifiers

Example

- **Predicts whether the player will slow down (i.e., brake) based on the distance to obstacle corner and the speed of the car**
- **Training data (previously collected):**
- **Suppose our current input is:**

  Distance = far, Speed = slow
- **Will the character brake?**

brake?	distance	speed
Y	near	slow
Y	near	fast
N	far	fast
Y	far	fast
N	near	slow
Y	far	slow
Y	near	fast

Naïve Bayes Classifiers

Example

$$A^{predict} = argmax_a\, P\{A = a\} \prod_i P\{X_i = u_i \,|A = a\}$$

❑ Calculation

brake?	distance	speed
Y	near	slow
Y	near	fast
N	far	fast
Y	far	fast
N	near	slow
Y	far	slow
Y	near	fast

<u>{A = brake}:</u>

P{brake}=5/7;

P{Dist=far|brake}=2/5; P{Speed=slow|brake} = 2/5;

$$P\{A = a\}\Pi_i\, P\{X_i = u_i|A = a\} = \frac{5}{7} * \frac{2}{5} * \frac{2}{5} = \frac{\mathbf{4}}{\mathbf{35}};$$

<u>{A = not brake}:</u>

P{not brake}=2/7 ;

P{Dist=far|not brake}= 1/2 ;

P{Speed=slow|not brake} = 1/2 ;

$$P\{A = a\}\Pi_i\, P\{X_i = u_i|A = a\} = \frac{2}{7} * \frac{1}{2} * \frac{1}{2} = \frac{\mathbf{1}}{\mathbf{14}};$$

More likely to brake: 4/35 > 1/14

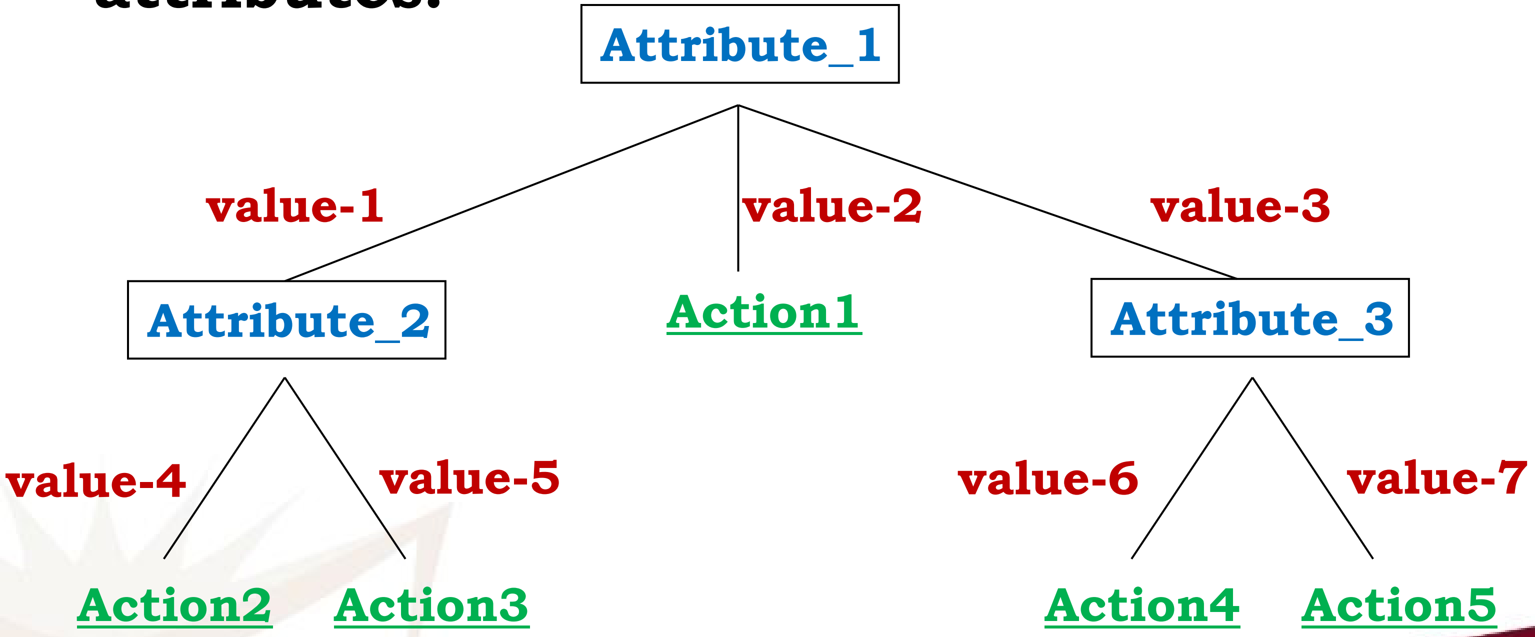
Université Concordia University

Decision Tree Learning

- **Decision trees can be efficiently learned: constructed dynamically from sets of observations and actions provided through strong supervision.**
- **The constructed trees can then be used in the normal way to make decisions during gameplay.**
- **There are *a range of different decision tree learning algorithms* used for classification, prediction, and statistical analysis.**
- **DTs can be used to learn about objects or other characters in game environment.**
- **Those used in game AI are typically based on Quinlan's ID3 algorithm.**

Decision Tree Learning

- The basic ID3 algorithm uses a set of observation–action examples. Example: *Hurt*, *In cover*, *With ammo*: Attack
- Observations in ID3 are usually called "attributes."



ID3

- The algorithm starts with a single leaf node in a decision tree and assigns the set of examples to the leaf node.
- It then splits the current node (initially the single start node) so that it divides the examples into two groups.
  - The division is chosen based on an attribute, and the division chosen is the one that is likely to produce the most efficient tree.
  - The division *does not consider the actions until all the actions in a node are the same*.

Decision Tree ID3 Example

Example Set (also the single start node):

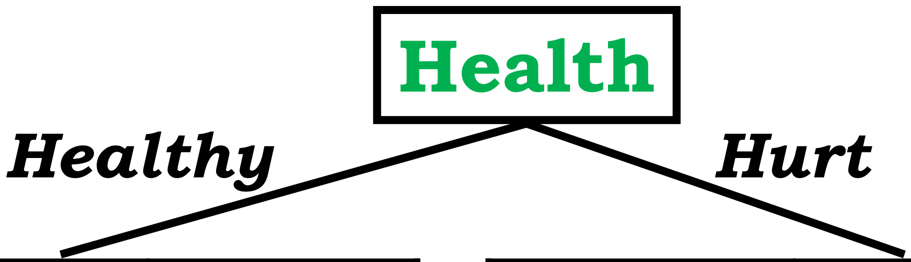
← Attributes →

Health	***Cover***	***Ammo***	***Action***
Hurt	In cover	With ammo	Attack
Healthy	In cover	Empty	Defend
Hurt	Exposed	With ammo	Defend
Healthy	In cover	With ammo	Attack

ID3

- When the division is made, each of the two new nodes is given the subset of examples that follow from the division, and the algorithm repeats for each of new nodes.
- This algorithm is recursive: starting from a single node it replaces them with decisions (observations) until the whole decision tree has been created.
- At each branch creation it divides up the set of examples among its daughters, until all the examples agree on the same action. At that point the action can be carried out; there is no need for further branches.

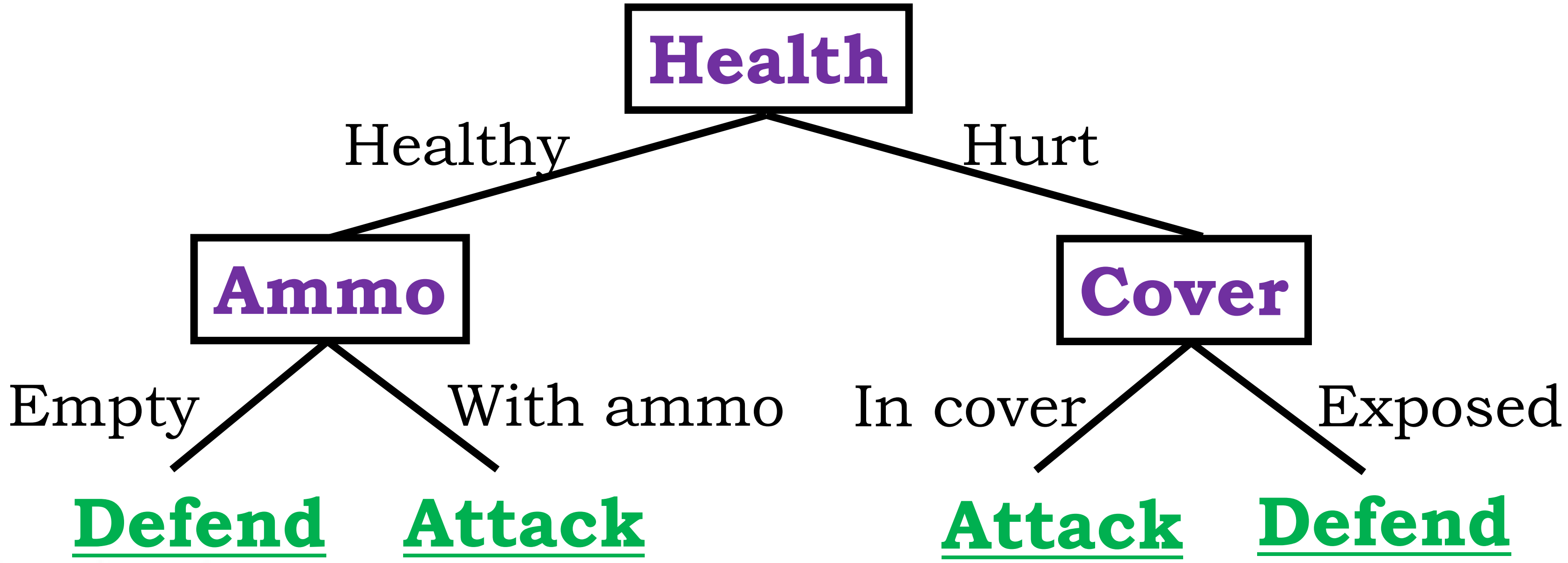
Decision Tree ID3 Example



Cover	Ammo	Action
In cover	Empty	Defend
In cover	With ammo	Attack

Cover	Ammo	Action
In cover	With ammo	Attack
Exposed	With ammo	Defend

Decision Tree ID3 Example



ID3: Entropy and Information Gain

- The split process looks at each attribute in turn (*i.e.*, each possible way to make a decision) and calculates the information gain for each possible division.
- The division with the highest information gain is chosen as the decision for this node.
- To do this calculation, ID3 uses the entropy of the actions in the set.
- Entropy is a measure of the information in a set of examples.
- Information gain is simply *the reduction in overall entropy*.

ID3: Entropy and Information Gain

- If our example set has n actions, where p_i is the proportion of each action in the example set, then (total) entropy can be computed as:

$$E = -\sum_{i=1..n} p_i \log_2 p_i$$

- Information gain G for each division is the reduction in entropy E_S from the current example set S to the entropies E_{S_j} of the daughter sets S_j :

$$G = E_S - \sum_j \frac{|S_j|}{|S|} * E_{S_j} \quad \text{where} \quad S = \bigcup_j S_j$$

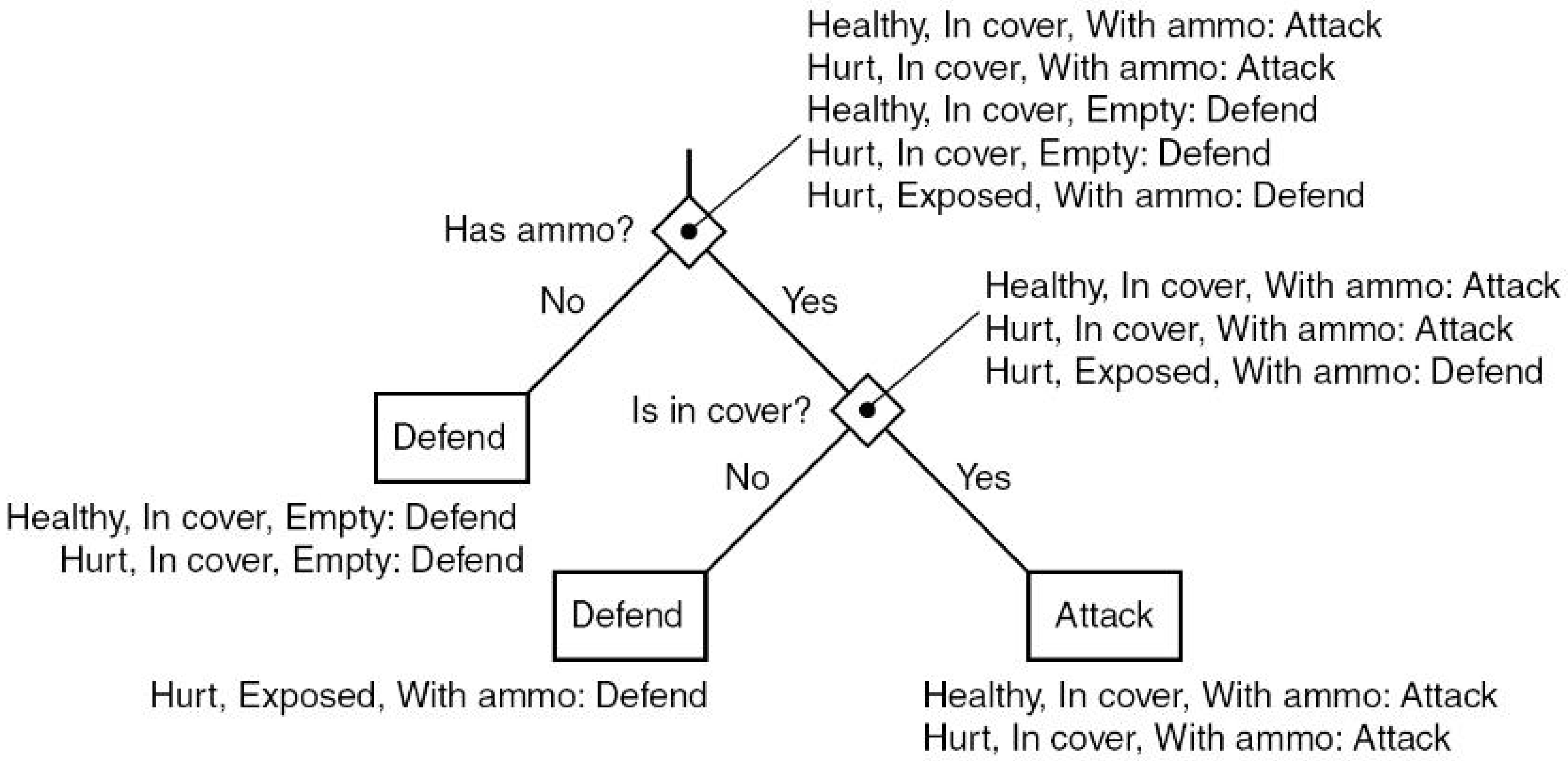
- Compute G for each attribute at a node; pick best attribute for division.

Incremental Decision Tree Learning

- ID3 is fine for creating a DT as part of pre-processing, but what if we want to dynamically update the DT as the game is running (e.g., online) as new examples become available?
- Running ID3 each time including new examples in the example set would be too slow. Adding the new examples simply as leaves of the tree would make the tree too tall, and slow to use for decision making.
- The simplest useful incremental algorithm is ID4. Initially, we start with the decision tree created by ID3.

ID4

- **Each node in the ID4 decision tree also keeps a record of all the examples that reach that node.**


Healthy, In cover, With ammo: Attack
Hurt, In cover, With ammo: Attack
Healthy, In cover, Empty: Defend
Hurt, In cover, Empty: Defend
Hurt, Exposed, With ammo: Defend
Has ammo?
No
Yes
Healthy, In cover, With ammo: Attack
Hurt, In cover, With ammo: Attack
Hurt, Exposed, With ammo: Defend
Defend
Is in cover?
No
Yes
Healthy, In cover, Empty: Defend
Hurt, In cover, Empty: Defend
Defend
Attack
Hurt, Exposed, With ammo: Defend
Healthy, In cover, With ammo: Attack
Hurt, In cover, With ammo: Attack

ID4

- When given a new example, each node will update itself according to one of the following:

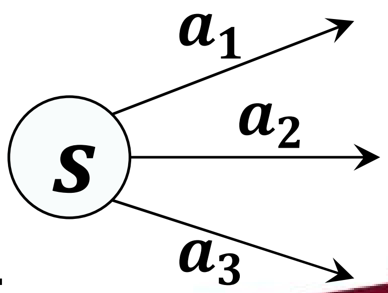
1. (node == terminal node) && (example has same action/class) ⟹ add example to list for node
2. (node == terminal node) && (example has different action/class) ⟹ make node a decision node and call ID3 to determine split.
3. (node != terminal node) ⟹ using information gain, determine best attribute to make decision on.
   a) if same attribute, add example to child node
   b) if different attribute, call ID3 for entire sub-tree (current node and descendants)

Reinforcement Learning

- **Reinforcement learning is the name given to a range of techniques for learning based on experience.**
- **In its most general form, a reinforcement learning algorithm has three components:**
  1. **an exploration strategy for trying out different actions in the game,**
  2. **a reinforcement function that gives feedback on how good each action is, and**
  3. **a learning rule that links the two together.**
- **In game applications, a good starting point is the Q-learning algorithm.**

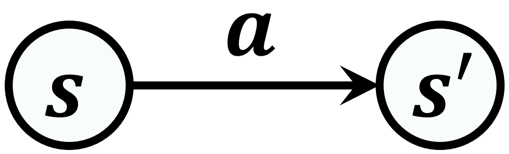
Q-Learning's World Representation

- Q-learning is known as a model-free algorithm because it doesn't try to build a model of how the world works. It simply treats everything as states, s. So, Q-learning treats the game world as a ***state machine***.
- Each state should encode all the relevant details about the character's environment and internal data; e.g., position, proximity of the enemy, health level, and so on.
- For each state, the algorithm needs to understand the actions that are available to it.



Q-Learning: Reinforcement

- After the character carries out one action in the current state, the reinforcement function should give it feedback. Feedback can be positive or negative and is ***often zero if there is no clear indication*** as to how good the action was.
- The four elements—the start state s, the action taken a, the reinforcement value r, and the resulting state s' — are called the **experience tuple**, often written as (s, a, r, s')

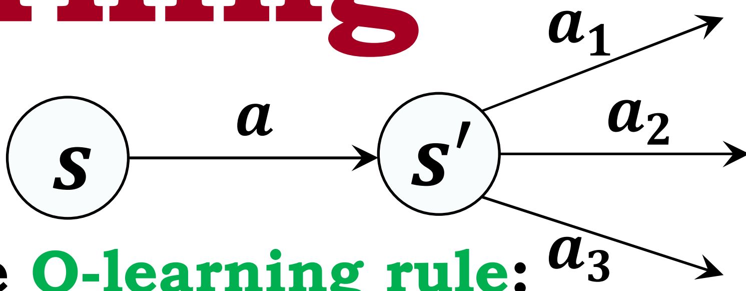


$$\begin{tikzcd} \boxed{s} \arrow[r, "a"] & \boxed{s'} \end{tikzcd}$$

Doing Learning

- Q-learning is named for the set of quality information (Q-values) it holds about each possible state and action, $Q(s, a)$. The algorithm keeps a value for every state and action it has tried.
- The Q-value represents how good it thinks that action a is to take when in that state s.
- The experience tuple (s, a, r, s') is split into two sections. The first two elements (the state and action) are used to look up a Q-value in the store. The second two elements (the reinforcement value and the new state) are used to update the Q-value based on how good the action was (r) and how good it will be in the next state.

Doing Learning



- The update is handled by the Q-learning rule:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'}(Q(s',a')))$$

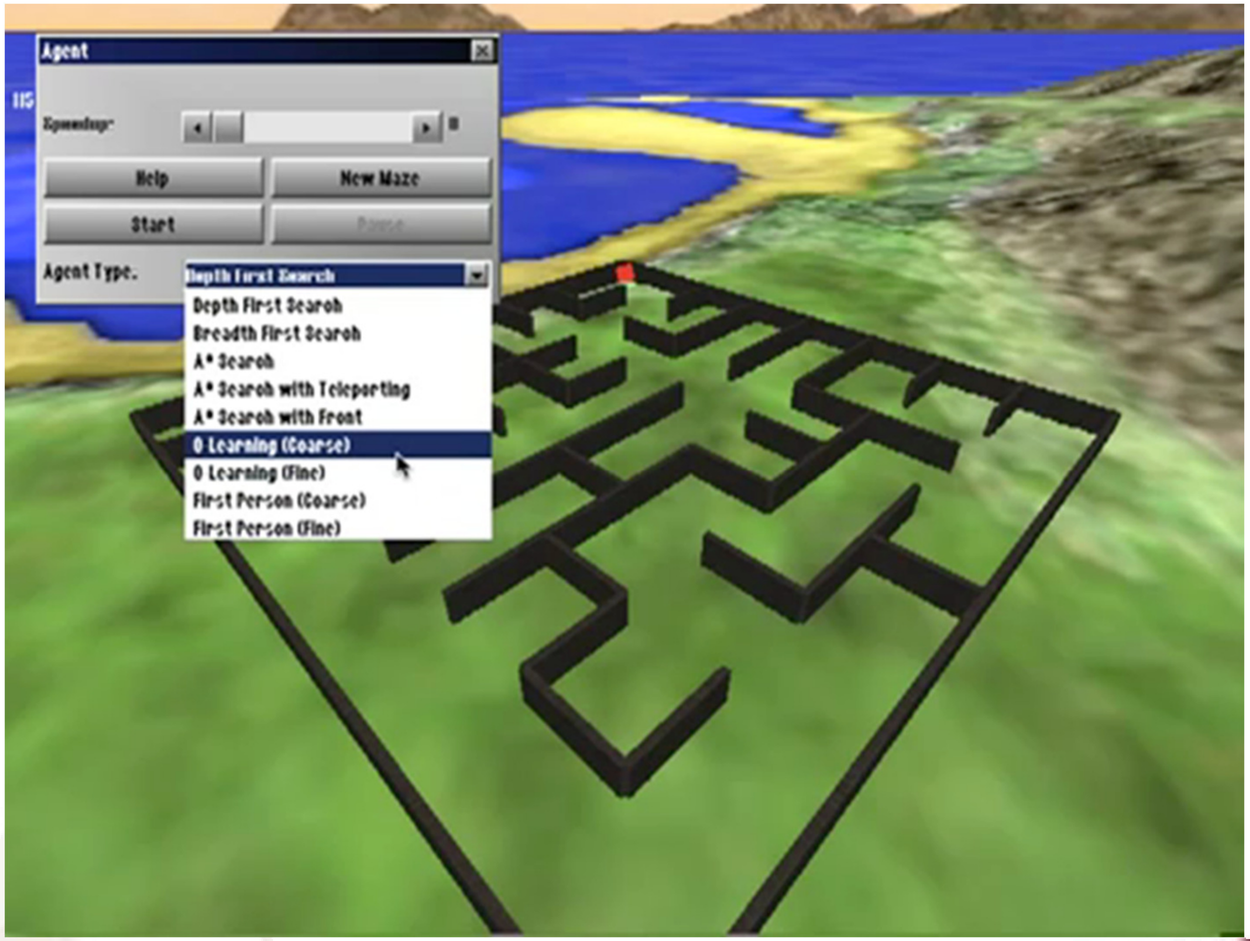
- where
  - α is the learning rate, in the range [0, 1], used to control the blend; and
  - γ is the discount rate, in the range [0, 1], controls how much the Q-value of the current state and action depends on the Q-value of the state it leads to. A very high discount will be a large attraction to good states, and a very low discount will only give value to states that are near to success.

Exploration Strategy

- Reinforcement learning systems also require an exploration strategy: a policy for selecting which actions to take in any given state.
- The basic Q-learning exploration strategy is partially random. Most of the time, the algorithm will select the action with the highest Q-value from the current state. The remainder, the algorithm will select a random action. The degree of randomness can be controlled by a parameter.
- Alternatively, we could incorporate the actions of a player, generating experience tuples based on their play.

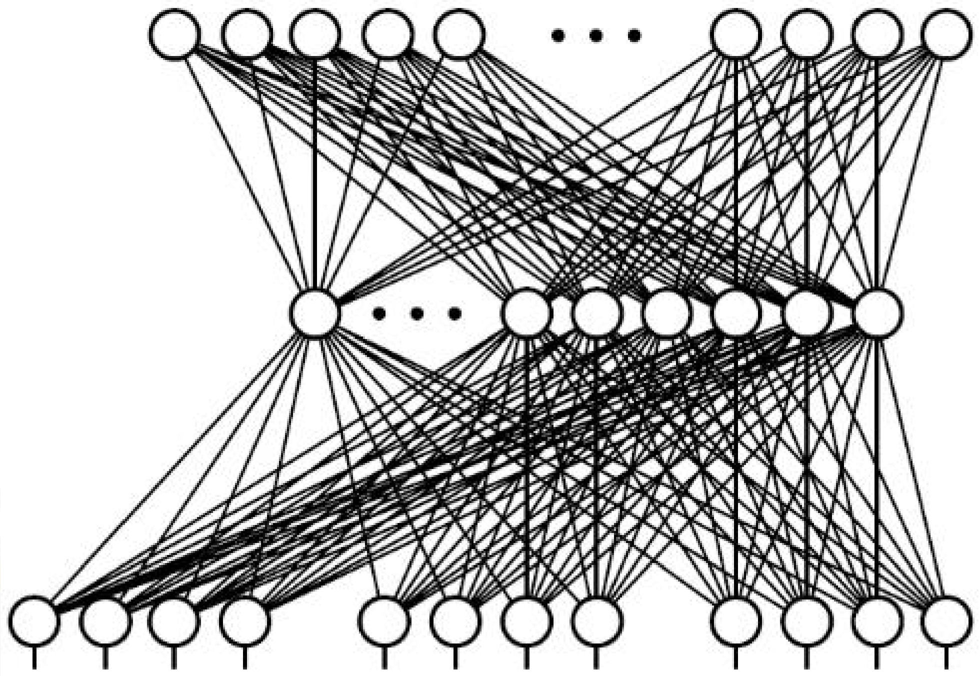
Q-Learning Demo

- (from www.opennero.org)



ANN

- There are many, many types of neural networks. So, we will just consider the most common form of ANN, the multi-layer perceptron (MLP).
- In an MLP network, perceptrons (the specific type of artificial neuron used) are arranged in layers, where each perceptron is connected to all those in the layers immediately in front of and behind it.



Used in: Black&White, Creatures

Multi-Layer Perceptron

- The multi-layer perceptron *takes inputs from all the nodes in the preceding layer* and *sends its single output value to all the nodes in the next layer*. Also known as a feedforward network
- The first layer (called the input layer) is provided input by the programmer
- The output from the last layer (called the output layer) is the output finally used to do something useful

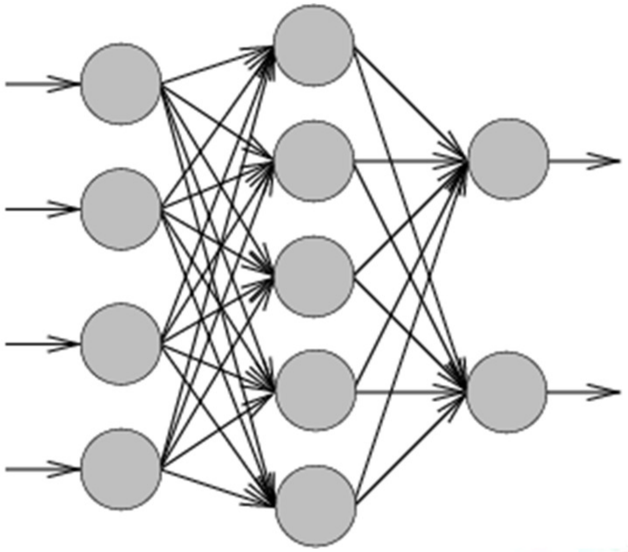
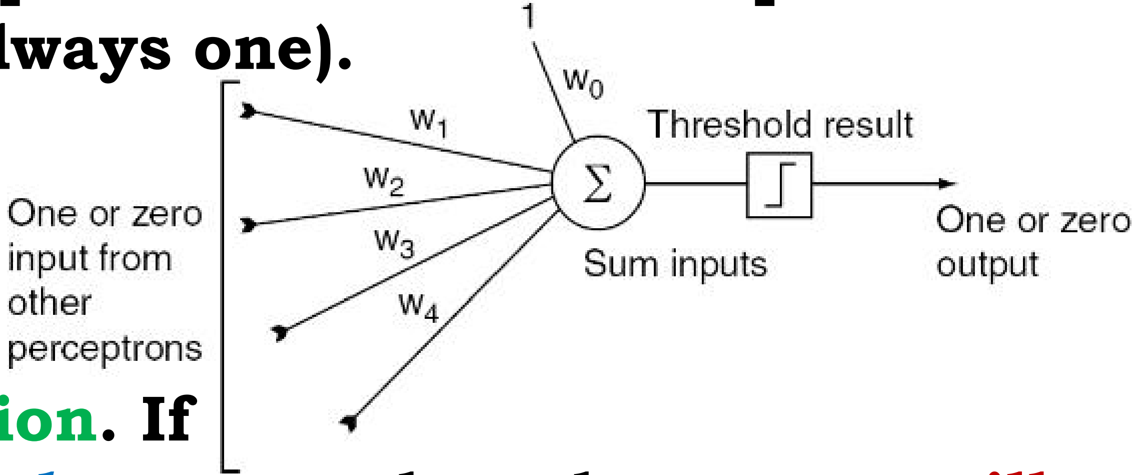


Image from publish.uwo.ca/~jmalczew/gida_7/Gilardi/Gilardi_Bengio.htm

Multi-Layer Perceptron

- **Each perceptron runs in parallel the same algorithm. The input values (we're assuming that they're zero or one here) are multiplied by the corresponding weight. An additional bias weight is added (it is equivalent to another input whose input value is always one).**



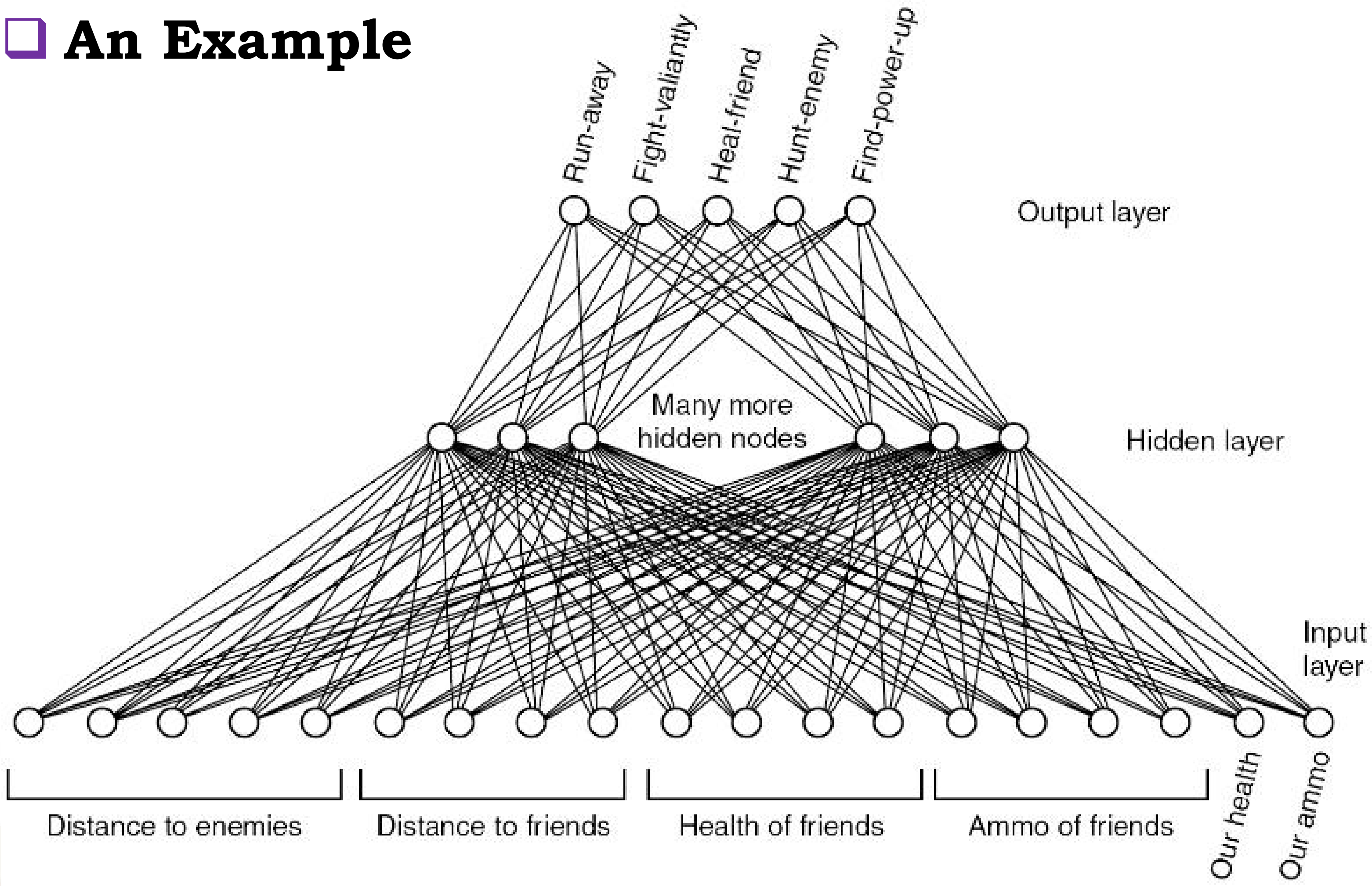
- **The final sum is then passed through a threshold function. If the *sum is less than zero*, then the neuron will be off (have a value of zero); *otherwise*, it will be on (have a value of one).**

Multi-Layer Perceptron

- To learn, the multi-layer perceptron network is put in a specific learning mode. Here another algorithm applies: the learning rule.
- Although the learning rule uses the original perceptron algorithm, it is more complex.
- The most common learning algorithm used in multi-layer perceptron networks is backpropagation. Where the network normally feeds forward, with each layer generating its output from the previous layer, backpropagation works in the opposite direction, working backward from the output. An error term guides the adjustment of the weights w_i.

Multi-Layer Perceptron

- An Example



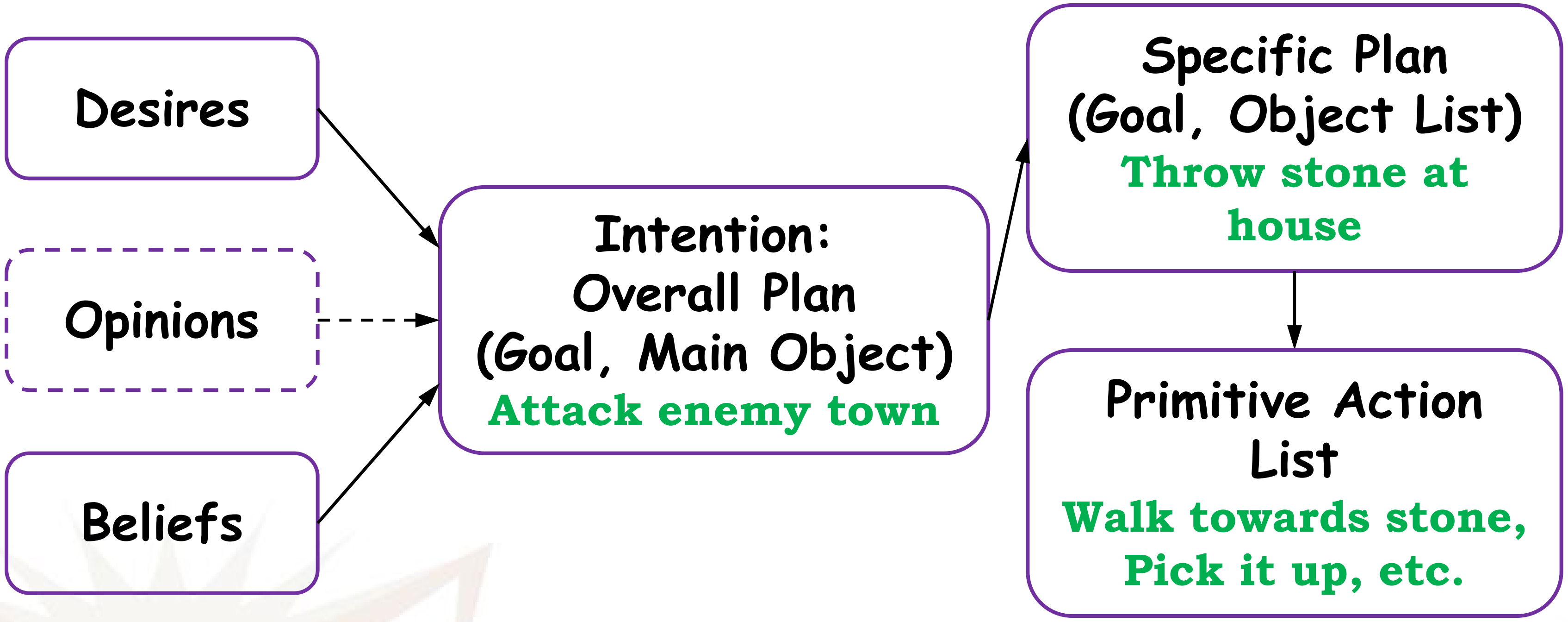
AI Game Programming Wisdom 1

Chapter 11.2

LEARNING in Black & White

Belief-Desire-Intention

- Frequently, autonomous NPCs (i.e., agents) are designed using a Belief-Desire-Intention (BDI) architecture, here augmented with Opinions.



Beliefs

- Beliefs are data structures storing information about individual objects. Symbolic attribute-value pairs are used to represent a creature's belief about any individual object.
- An example of this type of representation is:

  Strength of obstructions to walking:

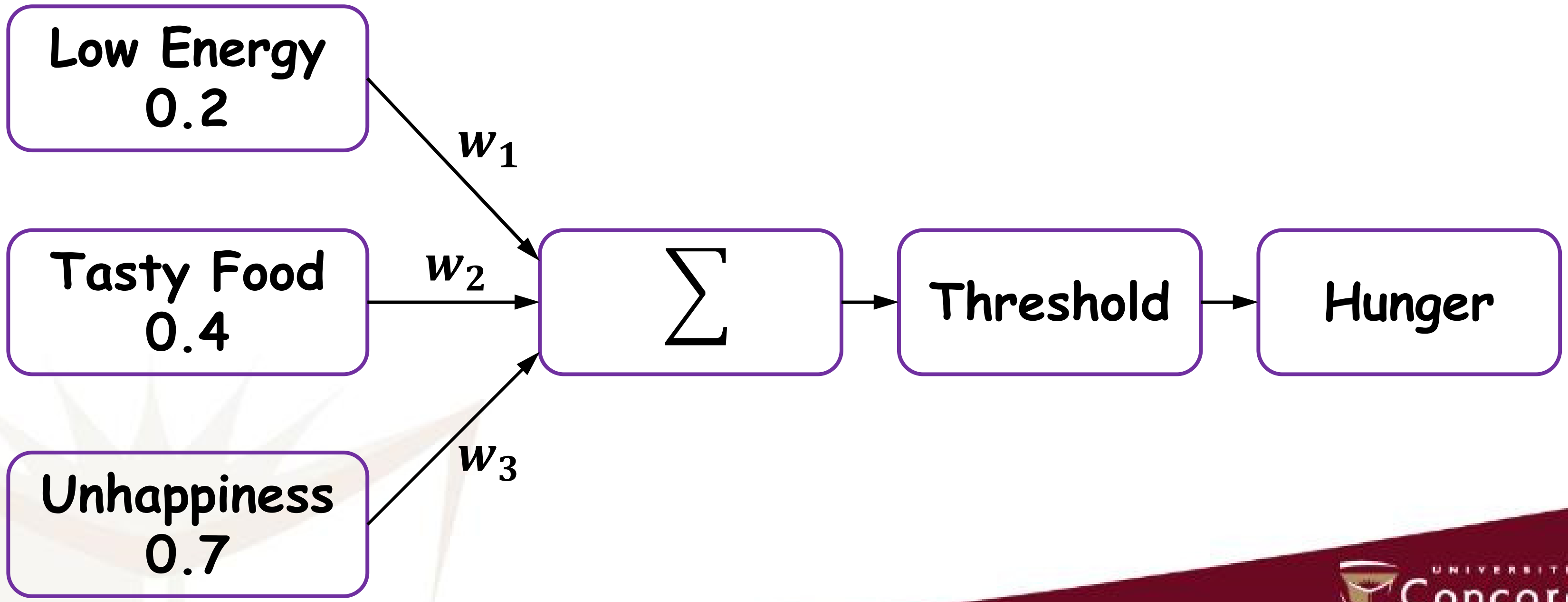
  object.man-made.fence->1.0

  object.natural.body-of-water.shallow-river->0.5

  object.natural.rock->0.1
- This method is used alongside with rules-based AI (situational calculus) to give creatures their basic intelligence about objects.

Desires

- Desires are goals, that will likely change with time, that the NPC attempts to satisfy.
- In Black & White, desires are modelled as a single layer perceptron (here is the Hunger desire):



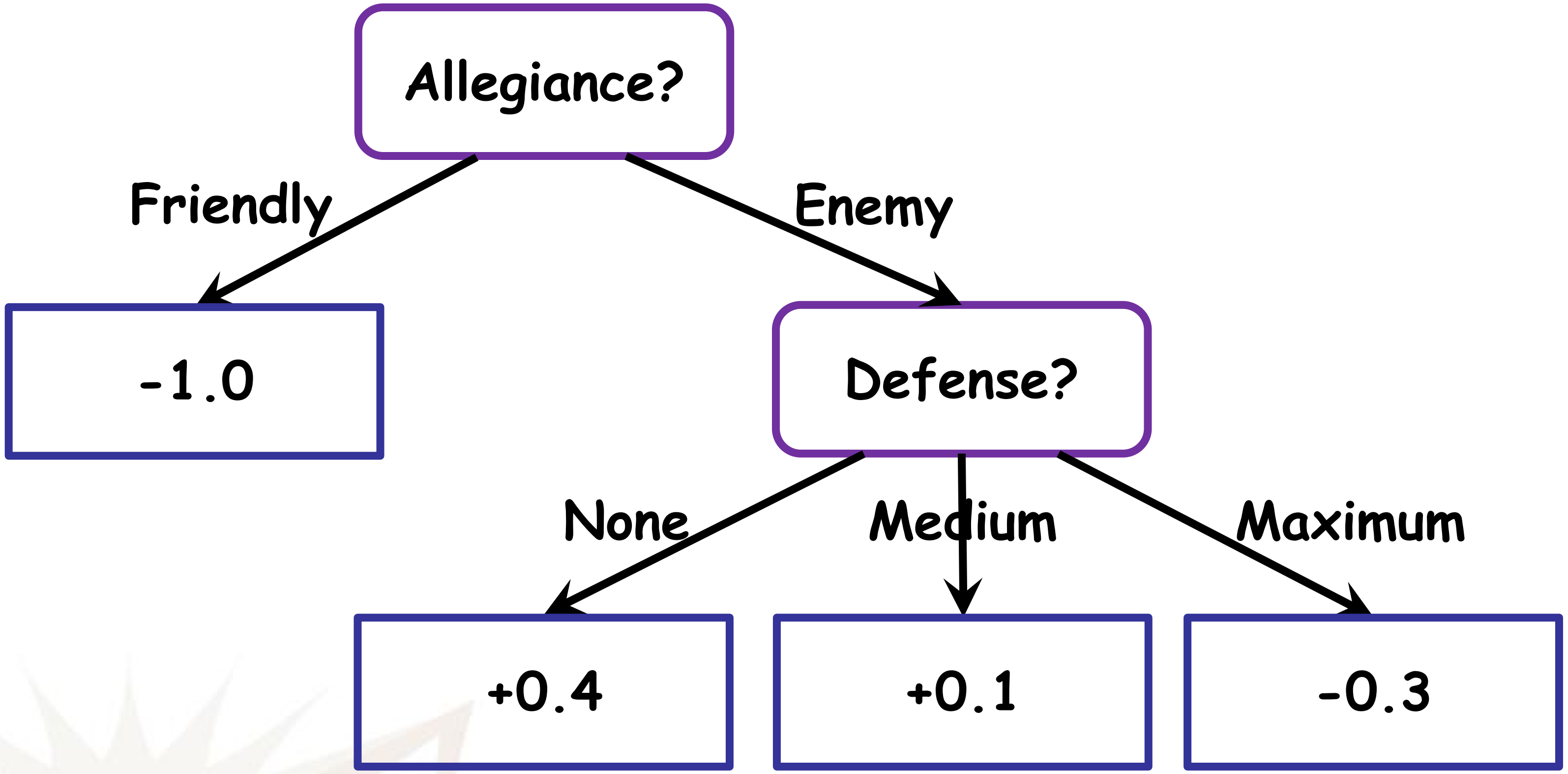
Opinions

- **In Black & White, Opinions are also used to determine Intentions. Each Desire (e.g. Attack Town) has an Opinion about it that expresses what types of objects (e.g., types of town) are best suited for satisfying this Desire.**
- **Black & White uses Decision Tree Learning for Opinions. Consider the following feedback.**

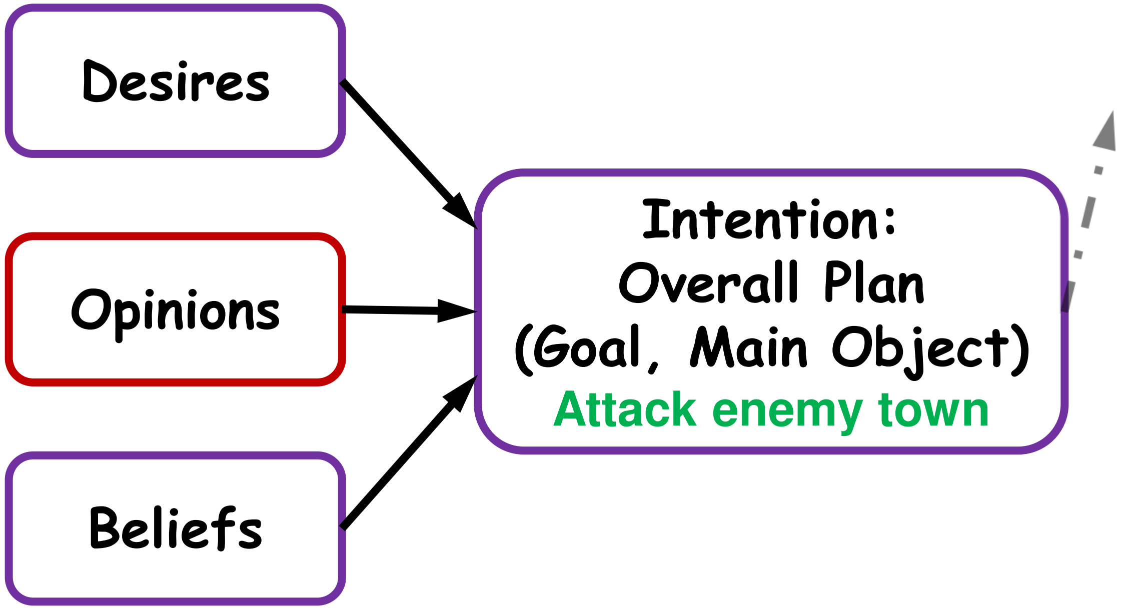
What Creature Attacked	Feedback from Player
Friendly town, weak defense, tribe Celtic	-1.0
Enemy town, weak defense, tribe Celtic	+0.4
Friendly town, strong defense, tribe Norse	-1.0
Enemy town, strong defense, tribe Norse	-0.2
Friendly town, weak defense, tribe Greek	-1.0

Opinions

- Example: Anger Opinion Decision Tree



Intentions



- Intentions are represented by plans. For each desire currently active, the NPC looks through all suitable beliefs and picks the one that it has the best opinion about.

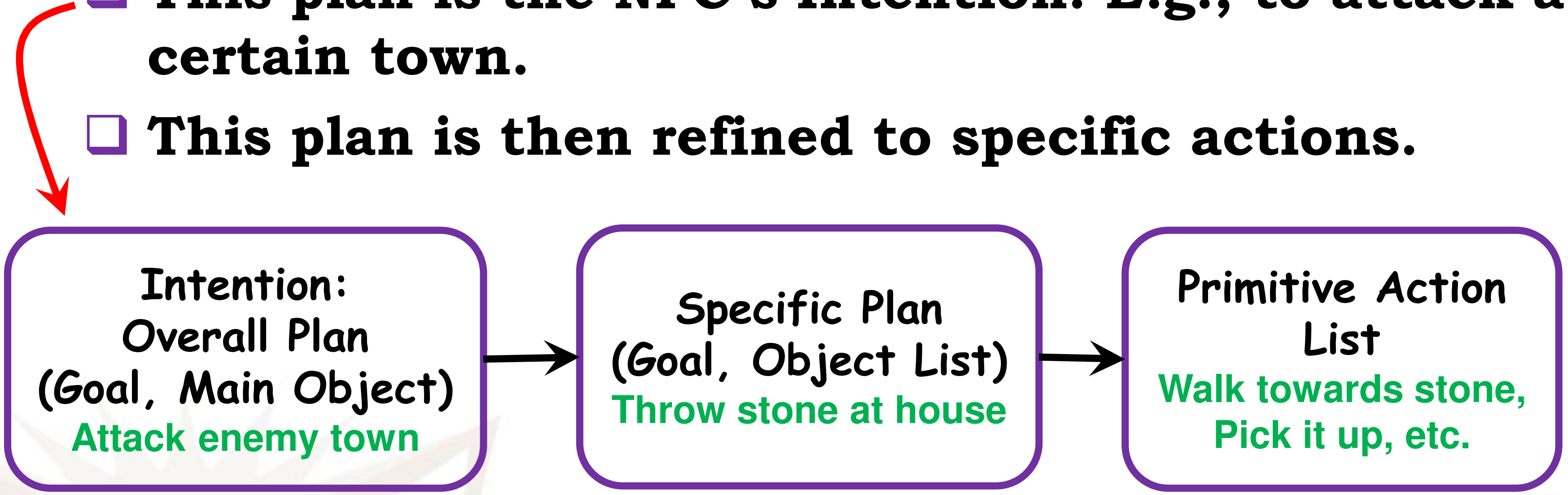
Intentions

- The NPC then forms a plan for each goal, then picking the plan that has the highest utility:

utility(desire, object) =

intensity(desire) * opinion(desire, object)

- This plan is the NPC's Intention. E.g., to attack a certain town.
- This plan is then refined to specific actions.


Intention:
Overall Plan
(Goal, Main Object)
Attack enemy town
Specific Plan
(Goal, Object List)
Throw stone at house
Primitive Action List
Walk towards stone,
Pick it up, etc.