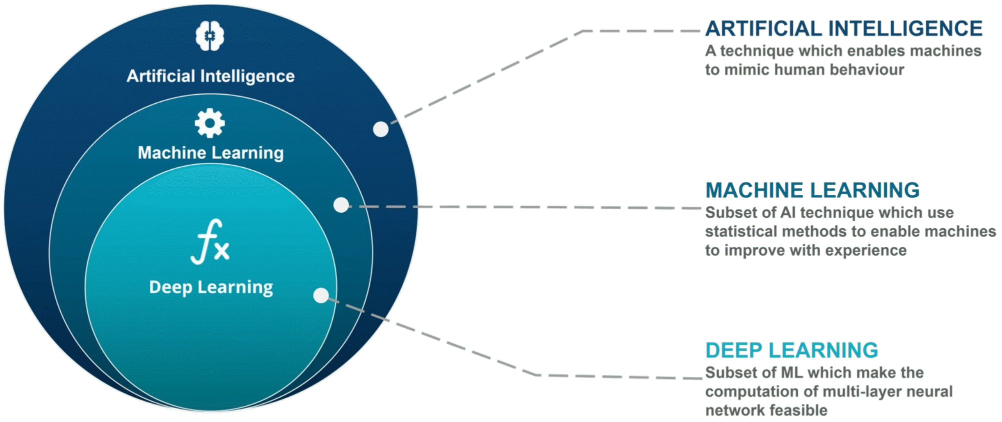
Artificial Intelligence and Simulations (Serious Games) in Neuroweapons


Artificial Intelligence
Machine Learning
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Computer Science has been enlisted in the service of Intelligence since its inception and is a direct derivative of warfare in World War 2, I cover the history of computer science in relation to warfare in the next chapter. Developing from the ideals of Alan Turing and others, Artificial Intelligence has developed over the past several decades to do more autonomous computation not involving direct human manipulation of algorithms. An algorithm, is basically a computer program, a set of code or libraries of code working toward computing a problem. The basic starting block of Artificial Intelligence is based in statistics, and what is known as Statistical Learning. Also, we have seen how what is considered cutting edge commercial technical advances in AI such as Generative Adversarial Networks (GANs) have been used in military defense technology for some 20 years before being 'leaked' to the commercial world. [See Ch. ?]. Yet, we see in the commerical application of AI severe ethical and technological problems are constantly at the forefront of the discussion of AI, yet, there is no gaurantee in the covert world with limited dialogue and availability to others research that such cutting edge developments were done with due regard to code review and testing before being put into production to meet deadlines of lucrative contracts in the often black budgeted world of Defense contractors. Given that this is a technical question it is good to have at least some technical background knowledge of some of the challenges involved in developing Automation and AI. In the following a brief overview of Machine Learning is presented.

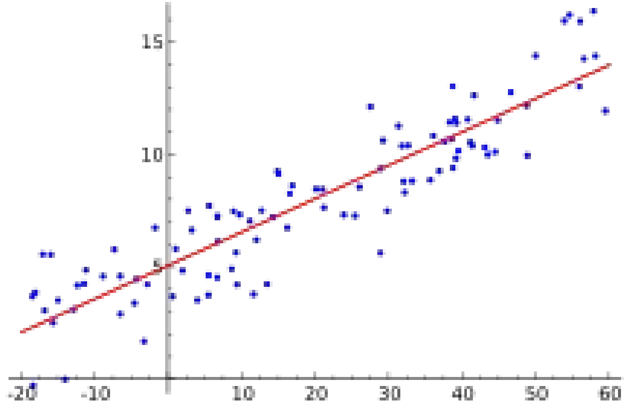
First, we need to understand what we say when we are talking about Machine Learning:

> *A computer program is said to learn from experience E with respect to some class of Tasks T and performance measure P if its performance at Tasks T, as measured by P, improves with experience E. --Tom Mitchell*

Machine Learning is based in statistics (similar to statistical mechanics in physics), the early work in Artificial Intelligence was done in Statistical Learning.

Definition of Statistical Learning - a set of approaches for estimating f, f is estimated for prediction and inference, how to estimate f: parametric (paramaterization) vs. non-parametric (featureless)

Linearity



In statistics, linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.

When you plot data points on a graph and then draw a line indicating their average on that graph you are making a linear approximation of the distribution on that graph of a data points. The formulation for such a process is given:

y = mx + b

y is the temperature in Celsius—the value we're trying to predict. m is the slope of the line. x is the number of chirps per minute—the value of our input feature. b is the y-intercept.

in machine learning, you'll write the equation for a model slightly differently:

y'= b + w1x1

y' is the predicted label (a desired output). b is the bias (the y-intercept), sometimes referred to as . w is the weight of feature 1. Weight is the same concept as the "slope" in the traditional equation of a line. x is a feature (a known input).

A linear regression model with two predictor variables can be expressed with the following equation:

Y = B0 + B1*X1 + B2*X2 + e.

The variables in the model are:

Y, the response variable; X1, the first predictor variable; X2, the second predictor variable; and e, the residual error, which is an unmeasured variable. The parameters in the model are:

B0, the Y-intercept; B1, the first regression coefficient; and B2, the second regression coefficient. One example would be a model of the height of a shrub (Y) based on the amount of bacteria in the soil (X1) and whether the plant is located in partial or full sun (X2). Y is the dependent variable you are trying to predict, X1, X2 and so on are the independent variables you are using to predict it, b1, b2 and so on are the coefficients or multipliers that describe the size of the effect the independent variables are having on your dependent variable Y

Models

A model defines the relationship between features and label. Two phases of a model's life:

Training means creating or learning the model. That is, you show the model labeled examples and enable the model to gradually learn the relationships between features and label.

Inference means applying the trained model to unlabeled examples. That is, you use the trained model to make useful predictions (y'). For example, during inference, you can predict medianHouseValue for new unlabeled examples.

Prediction Accuracy and Model Interpretability

The question of accuracy is encountered in any quantitative science, such as AI.

flexibility vs. restrictiveness

if interested in inference, restrictive models are much more interpretable, linear regression is less interpretable, non-linear methods i.e. SVM with non-linear kernels are even less interpretable.

Supervised vs. Unsupervised Learning

supervised w/ Labels linear regression (quantitative [val,int,num])
A label is the thing we're predicting—the y variable in simple linear regression. The label could be the future price of wheat, the kind of animal shown in a picture, the meaning of an audio clip, or just about anything.
unsupervised w/o Labels classification/categorization (qualitative [text,varchar,char]) one example of Unsupervised learning is clustering (K-Means). Cluster analysis- ascertain on the basis of X1...Xn, whether the observations fall into relatively distinct groups.
semi-supervised- some instances have labels others do not.

Variables (Features, Parameters, Coefficients)

A feature is an input variable—the x variable in simple linear regression. A simple machine learning project might use a single feature, while a more sophisticated machine learning project could use millions of features
either quantitative/regression (num) or qualitative/classification (text) Exceptions: least square linear regression is used with quantitative responses, logistic regression is used with qualitative (two-class or binary response: male, female; it estimates class probabilities, it can be thought of as a regression method as well.
whether the predictors are qualitative or quantitative is considered less important
Loss and Measuring Quality of Fit

Training a model simply means learning (determining) good values for all the weights and the bias from labeled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called empirical risk minimization.

Loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. The goal of training a model is to find a set of weights and biases that have low loss, on average, across all examples

In order to eval the performance of a statistical learning method on a given data set, we need a measure, we need to quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation.

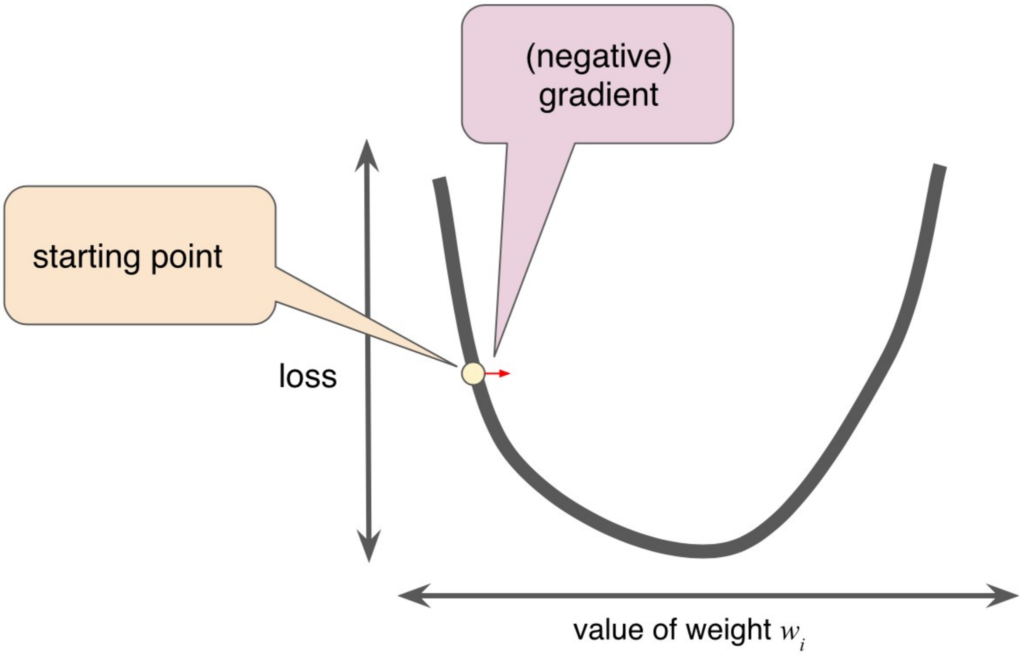
Regression Eval Metrics: MEA- Mean Absolute Error, is the mean of the abs err. MSE- Mean Squared Error, mean of the squared errors RMSE- sq.rt. of the MSE

Gradient Descent

A typical definition of gradient descent is:

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. If, instead, one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

However, to a mechanic this is still obtuse.



The gist of gradient descent is that you want to find the lowest point in the line, the arc of descent/ascent, like the creek bed in the forest: water always finds the lowest point to run along, the local minimum. What you are seeking in dealing with gradient descent is to find a piece of gold in the creek you need to step along the creek, if you step to far you will run right past it if you go to slow, it will take forever. This stepping through the creek is what is known as the 'learning rate' or 'step size'. The sweet spot, or goldilocks zone, is the point where there is a low value to loss vs weights. You are trying to find that spot but not overshoot it. If the bottom of the arc is large, then your learning rate can be larger, if the gradient of the arc is steep then a small learning rate is needed.

Goodfellow et al note on gradient based optimization, “Most deep learning algorithms involve optimization of some sort. Optimization refers to the task of either minimizing or maximizing some function f (x) by altering x. We usually phrase most optimization problems in terms of minimizing f(x). Maximization may be accomplished via a minimization algorithm by minimizing −f (x).

The function we want to minimize or maximize is called the objective function, or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function.

Optimization algorithms that use only the gradient, such as gradient descent, are called first-order optimization algorithms. Optimization algorithms that also use the Hessian matrix, such as Newton’s method, are called second-order optimization algorithms” (Goodfellow et al, Ch 4.3).

Hyperparameters

Hyperparameters are the knobs that programmers tweak in machine learning algorithms. Most machine learning programmers spend a fair amount of time tuning the learning rate. Iterations is another important hyperparameter in linear regression. It is important not to confuse hyperparameters with model features (also known as parameters). A very good explanation of hyperparameters is given by Xavier Amatriain :

In machine learning, we use the term hyperparameter to distinguish from standard model parameters. So, it is worth to first understand what those are.
A machine learning model is the definition of a mathematical formula with a number of parameters that need to be learned from the data. That is the crux of machine learning: fitting a model to the data. This is done through a process known as model training. In other words, by training a model with existing data, we are able to fit the model parameters.

However, there is another kind of parameters that cannot be directly learned from the regular training process. These parameters express “higher-level” properties of the model such as its complexity or how fast it should learn. They are called hyperparameters. Hyperparameters are usually fixed before the actual training process begins.

So, how are hyperparameters decided? That is probably beyond the scope of this question, but suffice to say that, broadly speaking, this is done by setting different values for those hyperparameters, training different models, and deciding which ones work best by testing them.

So, to summarize. Hyperparameters: Define higher level concepts about the model such as complexity, or capacity to learn. Cannot be learned directly from the data in the standard model training process and need to be predefined. Can be decided by setting different values, training different models, and choosing the values that test better Some examples of hyperparameters:

- Number of leaves or depth of a tree
- Number of latent factors in a matrix factorization
- Learning rate (in many models)
- Number of hidden layers in a deep neural network
- Number of clusters in a k-means clustering https://www.quora.com/What-are-hyperparameters-in-machine-learning (accessed 2/6/2019)

The Bias-Variance Trade off

In order to min the expected error, we need to select a statistical learning method that simultaneously achieves low variance and low bias.

variance- refers to the error that is introduced by approximating a real-life problem, by a much simpler model. For example, linear regression assumed that there is a linear relationship between Y and $X_1, X_2, ...X_n$. It is unlikely any real-life problem truly has such a simple linear relationship.

As a general rule, as we use more flexible methods,the variance will increase and the bias decrease. The relative rate of change of these two quantities determines whether thhe test MSE increases or decreases. As we increase the flexibility of a class of methods, the bias tends to mutually decrease faster than the variance increases. Consequently, the expected test MSE declines. Howeve at some point increasing flexibility has little impact on the bias but starts to significantly increase the variance. When this happens the MSE increases.
The relationship b/w bias, variance + MSE is known as bias-variance trade off.

Bayes' Theorem: describes the probability of an event, based on prior knowledge of conditions that might be related to the event. One of the many applications of Bayes' Theorem is Bayesian inference, a particular approach to statistical inference. When applied, the probs involved in Bayes' theorem may have diff. prob. interpretations. With the Bayesian probability interpretation the theorem expresses how a subjective degree of belief should rationally change to account for availability of related evidence.

Simple Linear Regression- predicts a quantitative response Y on the basis of a single predictor variable X, assumes linear relationship

$Y \approx \beta_0 + \beta_1 X$ (Y approximately equals beta 0 plus beta 1, times X) sales $\approx \beta_0 + \beta_1$xTV ads
constants: β_0- intercept | β_1- slope of the linear model, together known as model coefficients or parameters
then compute future sales on the basis of future TV ads $\hat{y} = \beta_0 + \beta_1 X$ where $\hat{y}$ (predicted y) indicates a prediction of Y on thhe basis X = x
RSS-residual sum of squares

residual- $e_i = Y_i - Y_i$ in iterator i this is the difference between the i observed response value and the i predicted response value in the iteration i

the least squares approach chooses $\beta_0 + \beta_1$ to min. RSS

Bias-Unbiased

if we use the mean (avg) of the sample $\hat{\mu}$ to estimate μ this estimate is unbiased, in the sense that on average,we expect $\hat{\mu} = \mu$,meaning on one set of observations $y_1,...y_n$ $\hat{\mu}$ might overestimate, with another set of observations it might underestimate, but with an average of a large volume of observations it is more accurate.

The quality of a linear regression model fit is made using:

a. RSE -residual standard error (smaller the RSE val the better)

b. R2 static- takes the form of a proportion val 0-1 is independent of the scale of Y, used more in multiple linear regression.

Model Fit, use RSE and R2

Predictions

3 Kinds of uncertainty associated with prediction a. coefficient estimates are the least square plane which is only an estimate for the true population regression plane. The innacuracy in the coefficient estimates is related to the reducible error. We can compute a confidence interval in order to determine how close Y will be to f(X)

b. in practice assuming a linear model for f(x) is almost always an approximation of reality, model bias-when we use a linear model,we are in fact estimating the best linear approximation to the true surface, ignoring the real topo for linear approx

c. even if wel new f(x) the response value cannot be predicted perfectly because of the random error (ε), the irreducible error.

how will Y vary from Y- this is prediction intervals which are wider then confidence intervals, because they include the reducible error [f(x) estimate]and the uncertainty how each individual point will differ from the population regression plane (the irreducible error).

confidence interval is used to quantify the average of all points of Y.

Dummy variable are used to replace text with num in qualitative data (i.e. male,female to 0,1) use only with binary choices or in other words don't use linear regression for categories.

Additive and Linear

Two of the most important assumptions state that the relationship between p+r are additive and linear.

additive- the effect of changes in a predictor Xj on the response Y is independent of the value of the other predictors Pi

linear- that the change in the response Y due to one-unit change in Xj is contant regardless of the val of Xj

there are methods that can be used to lessen additivity and linearity.

A Recipe for a Linear Regression Algorithm

1. train test split of data
2. create and train the model
3. fit the data
4. evaluate our model, checking coefficients and see how we interpret them
5. check coefficients, relate to each feature in datasset
6. predictions
7. check metrics

Potential Problems of Linear Regression

There are several tricky areas related to linear regression, some of these areas are covered below. It is important to understand that linear regression is a mathematical methodology of dealing with quantifiable objects, a numerical relationship to other numerical relationships.

1. Non-linearity of the r-p relationships

if the true relationship is far from linear, then virtually all of the conclusions that we draw from the fit are questionable, thus the prediction accuracy can be reduced.

Residual plots are a tool to detect non-linearity if the residual plots indicate there are non-linear associations in the data, then a simple approach is to use a non-linear transformation of the predictors; logX, sq. rt of X and X^2 in the regression model.

2. Correlation of Error Terms

important assumption in linear regression is that error terms ε1,ε2...,εn are uncorrelated. If there is correlation then the estimated standard error with underestimate the true standard error. Often occurs in time series data which is detectable through tracking (sim values for i) in adjacent residuals.

3. Non-Constant Variance of Error Terms

Another important assumption of the linear regression model is that the error terms have a constant variance, The standard errors, confidence intervals, and hypothesis tests associated with the linear model rely upon this assumption. But often they are non-constant

4.Outliers

An outlier is a point for which y, is far from the val predicted by the model residual plots identify outliers we can plot the studentized residuals- computed by dividing each residual e, by it's estimated standard error. Possible outliers have a value of >3, if outliers are data recording error, remove it.

5. High Leverage Points

observations with high leverage have an unusual value for predictor x. They have high impact on the estimated regression line. Leverage statistics is used to compute an observations leverage.

6. Collinearity

2 or more predictor vars are closely related to each other. The presence of collinearity can pose problems in the regression context, since it can be difficult to separate out the individual effects of collinear variables on the response. Collinearity reduces the accuracy of the estimates of the regression coefficients it causes the standard error for $\hat{\beta}$ to increase. The hypothesis test- the probability of correctly detecting a non-zero coefficient- is reduced. Detection: look at correlation matrix of the predictors- a large value means collinearity for multicollinearity assess the variance inflation factor (VIF). As a rule a VIF val > 5 = bad.

Cautionary Tales of Statistical Methods

There are some very important things to understand with Statistical Methods in Artificial Intelligence. The very basis of using statistics is a mathematical approximation of real things in the real world. Therefore, this approximation will always be an oversimplification of real things. However, for a mathematical based machine, such as the computer, it must use mathematical methods and not necessarily related to natural mechanics, to formulate a response, that which is returned from an Algorithm. The next section will go over Machine Bias that develops from these mathematical statistical methods. We should not treat returns from computation to be some sort of objective truth, rather it is affected by mathematical equations and the limitations of approximation.

Another complicating factor, and source for noise and results filled with errors if algorithms that are not properly set up, is the use of Probability in AI. Ian Goodfellow notes the use of probability in AI:

> Probability theory is a mathematical framework for representing uncertain statements. It provides a means of quantifying uncertainty as well as axioms for deriving new uncertain statements. In artificial intelligence applications, we use probability theory in two major

> ways. First, the laws of probability tell us how AI systems should reason, so we design our algorithms to compute or approximate various expressions derived using probability theory. Second, we can use probability and statistics to theoretically analyze the behavior of proposed AI systems.
>
> Machine learning must always deal with uncertain quantities and sometimes stochastic (nondeterministic) quantities. Uncertainty and stochasticity can arise from many sources. Researchers have made compelling arguments for quantifying uncertainty using probability since at least the 1980s. (Goodfellow, et al, 2016, ch. 3, 51)

Goodfellow et al go on to talk about 3 possible sources of uncertainty: Inherent stochasticity (non-determinism) in the system being modeled; Incomplete observability and Incomplete modelling. In the first case randomness has an effect, in the second, which happens in an open world, if we do not have complete data or know all the pieces of a chess game and a limited board, then we have Incomplete Observability. In the third case, the problem of outliers is a good example, AI algorithms ignore data that lies at the periphery in relation to the other data points, and on the other hand, if there is too much specificity in rules, it can lead to overfitting and break the model.

Probability extends logic to deal with uncertainty. Logic provides a set of formal rules, such as Piercean Logic, for determining what propositions are implied to be true or false given the assumption that some other set of propositions is true or false or in the Piercean Logic the third option: true & false, which raises certain questions about non-Boolean methods for validation checking, etc. Probability theory provides a set of formal rules for determining the likelihood of a proposition being true given the likelihood of other propositions. As one will not find any examples from Russell & Norvig, researchers and authors of what is considered the textbook on AI of Piercean Logic other than mention of C.S. Pierce in historical summaries, if developers use Piercean Logic then what form of validation do they use to check it is an accurate result? Since, Piercean Logic is used in defense sector computer engineering for automation and control this is an important question.

Another source of noise or error in computational algorithms that rely on Artificial Intelligence is the problem of numerical computation. Goodfellow et al explain:

> The fundamental difficulty in performing continuous math on a digital computer is that we need to represent infinitely many real numbers with a finite number of bit patterns. This means that for almost all real numbers, we incur some approximation error when we represent the number in the computer. In many cases, this is just rounding error. Rounding error is problematic, especially when it compounds across many operations, and can cause algorithms that work in Numerical Computation theory to fail in practice if they are not designed to minimize the accumulation of rounding error. One form of rounding error that is particularly devastating is underflow. Underflow occurs when numbers near zero are rounded to zero. Many functions behave qualitatively differently when their argument is zero rather than a small positive number. Another highly

> damaging form of numerical error is overflow. Overflow occurs when numbers with large magnitude are approximated as ∞ or −∞. Further arithmetic will usually change these infinite values into not-a-number values. One example of a function that must be stabilized against underflow and overflow is the softmax function.
> (Goodfellow et al, 2016, Ch. 4)

Yet, as many programmers of algorithms can tell you there is often the binary choice between 0,1. So that if we always assume a division between 0,1 then the rounding error may come into play. As one can see just because we have a machine that performs at unbelievably fast rates, and usually produces correct results, with these and other complicating factors such a system that is not accurate and precise can generate large errors, that can also grow exponentially over time. One journalist writing about the use of AI in major corporations makes a great point:

> Mathematicians say that it's impossible to make a "perfect decision" because of systems of complexity and because the future is always in flux, right down to a molecular level. It would be impossible to predict every single possible outcome, and with an unknowable number of variables, there is no way to build a model that could weigh all possible answers. Decades ago, when the frontiers of AI involved beating a human player at checkers, the decision variables were straightforward. Today, asking an AI to weigh in on a medical diagnosis or to predict the next financial market crash involves data and decisions that are orders of magnitude more complex. So instead, our systems are built for optimization. Implicit in optimizing is unpredictability—to make choices that deviate from our own human thinking. (Webb, 2019)

These are just some of the factors that can also lead to Machine Bias in AI Models, since results from an AI algorithm depend on the correctness of the Model a bad model can lead to a completely destructive algorithm. There are also other sources of Machine Bias which we shall now cover.

Machine Bias

In a much reported incident it was discovered that Africans in Image Recognition software were being identified as Gorillas rather then as humans[1]. This raised great concerns about the bias of Machine Intelligence algorithms employed in Image Recognition, although it is true that a hard-right political activist, Robert Mercer, was involved in the development of Image Recognition software while at IBM, it was discovered that the problem was in the AIs inability to properly deal with dark colors. This is just one example of Machine Bias in an AI system, once again, if we do consider a machine to be neutral and objective, this is not always true. For instance, problems are also reported in automated policing systems that rely on AI, where it seems to target in the US, African-Americans. These results were also demonstrated in automated judicial processes where AI is employed to decide court cases. As well there is the episode where Microsoft released a chat bot AI that quickly was skewed to voice far-right neo-Nazi rhetoric based on 'data poisoning' attacks (see below), and Google Search algorithms that also skewed toward the far right [2]. In dealing with automated Intelligence systems, never mind that generally the culture in Intelligence is biased toward the Right, there would be necessarily already existing biases which could slant the AI system into an even more biased depiction of actual events in the world. Just ask yourself, how would a system that is investigating ISIS, or other Islamist Jihadist groups, view Muslims in general, would it be biased? A paper on the dangers of Lethal Autonomous Weapons Systems (LAWS) points out some of the dangers of machine bias:

> However, as ‘intelligent’ software and machines need to be ‘fed’ by a huge amount of data in order to ‘learn’ (a trait that we deem ‘intelligent’), there exists the risk that they learn human prejudices from biased data. And so-called machine biases constitute a danger for AI-controlled or autonomous systems that some experts regard as far more acute than LAWS. Based on the data a bot is fed in order to learn, it could learn, e.g., to discriminate against people of color or minorities, or gain a strict political attitude.
> (Shurber, 2018, pg. 17-8)

Bias is not just limited to the problem of Intelligence or of Social Sciences. It can even be encountered in autonomous systems calculating each other in Financial Markets, when autonomous agents (or semiotic agents) are left to their own devices they capitalize, leading to economic bias:

> Researchers at the University of Bologna in Italy created two simple reinforcement-learning-based pricing algorithms and set them loose in a controlled environment. They discovered that the two completely autonomous algorithms learned to respond to one another’s behavior and quickly pulled the price of goods above where it would have been had either operated alone.
> (Calvano et al, 2018)

Although, we may assume that since, in many cases, we can look at the code of an AI algorithm that we would be able to understand what is happening from the results of an AI system. However, this is not always true, here we encounter what is known as the ‘blackbox’ problem. In Software Engineering we have such terms as white box and black box testing, in these cases a black box tester does not have access to the code or inner workings of the software undergoing testing, but has to figure out bugs based on this blindedness, which can actually be of more value, since the tester is not biased by the code and has knowledge of what it’s intended purpose is but rather can only go on the effects. In AI the blackbox comes into affect in a more complex way:

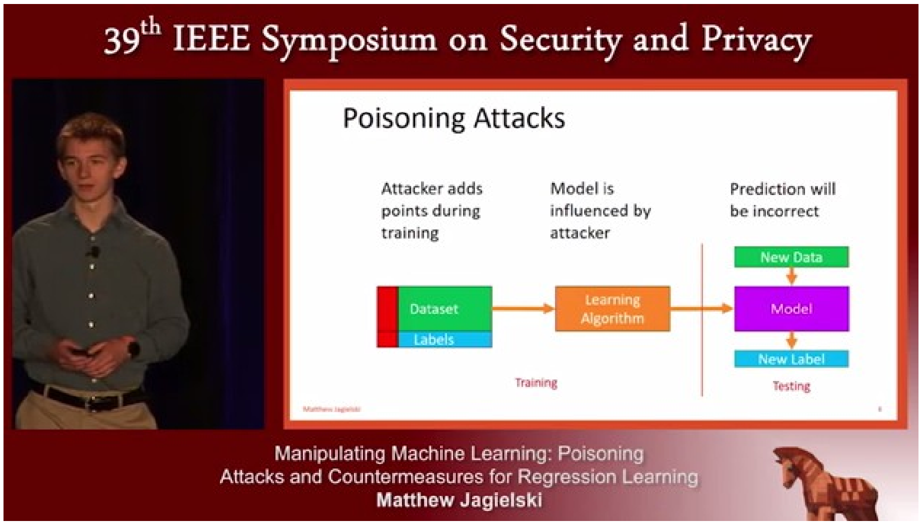
> That inability to observe how AI is optimizing and making its decisions is what’s known as the “black box problem.” Right now, AI systems built by the Big Nine might offer open-source code, but they all function like proprietary black boxes. While they can describe the process, allowing others to observe it in real time is opaque. With all those simulated neurons and layers, exactly what happened and in which order can’t be easily reverse-engineered. (Webb, 2019)

As is pointed out even the developers that code the algorithm can not fully give an account or even properly anticipate the results of an AI algorithm due to the blackbox problem. Now imagine, developing an Automated Surveillance system with this problem in mind and also bias, the dangers of such a system are not hard to realize.

As mentioned above attacks on Machine Learning algorithms can be accomplished through data poisoning. Researchers define data poisoning:

> Machine learning systems trained on user-provided data are susceptible to data poisoning attacks, whereby malicious users inject false training data with the aim of corrupting the learned model. While recent work has proposed a number of attacks and defenses, little is understood about the worst-case loss of a defense in the face of a determined attacker. (Steinhardt et al, 2018)

In a certain sense one can see a parallel between data poisoning and Information Operations, or Thought Injection attacks, which resembles the hacking technique of SQL Injection attacks were a hacker injects SQL code via http requests to corrupt or poison a database or dataset or collection. The study of data poisoning began around 2012, some time after ML was used in Defense contracting, whether there are mechanisms to protect against data poisoning in their systems is unknown due to the classified nature of such work. There are countermeasures being developed against such attacks. At the 2018 IEEE Symposium on Security and Privacy Matthew Jagielski provided one such methodology. In the below diagram we can see the flow of the attack vector and the results. Again validating results becomes a key criteria in developing resilient and accurate algorithms, so that developers will have to be vigilant with their Test and Validate cycles in AI as it becomes larger and larger in our society.



39th IEEE Symposium on Security and Privacy

Poisoning Attacks

Attacker adds points during training

Model is influenced by attacker

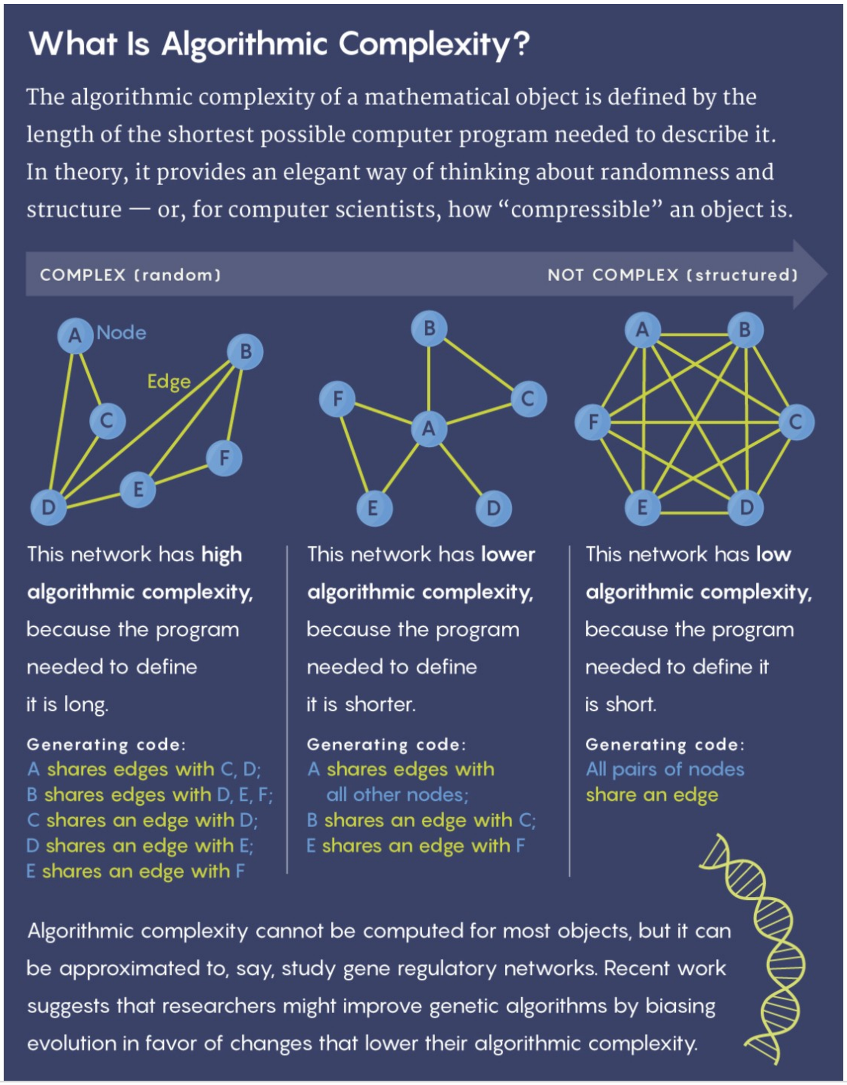
Prediction will be incorrect

Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning
Matthew Jagielski

(Jagielski, 2018)

Algorithmic Complexity

Aside from Machine Bias another area of concern in algorithm engineering is that of Algorithmic Complexity (See Graph below). One element that is a complicating factor in algorithmic complexity is the degree to which other components of a system are visible and connected to other elements within the program. As such, a compartmentalized non-visible architecture as used in covert operations and computation will by necessity of it's invisibility to each component create an complex algorithmic environment. Thus leading to unforeseen programming outputs and possibly circular and contradictory logic. This is the opposite of that foreseen in the Cybernetics of Beer's. Where each node in the algorithmic management system is visible. In this sense Beer's algorithms, which are being developed by sustainability advocates for such things as democratic finance, is the opposite of a covert system. Transparency thus giving it a technical advantage. Beer's system would resemble the nodes on the right in the chart below, whereas a covert system would resemble the graph on the left below.



What Is Algorithmic Complexity?

The algorithmic complexity of a mathematical object is defined by the length of the shortest possible computer program needed to describe it. In theory, it provides an elegant way of thinking about randomness and structure — or, for computer scientists, how "compressible" an object is.

COMPLEX (random) NOT COMPLEX (structured)

This network has **high algorithmic complexity,** because the program needed to define it is long.

Generating code:
A shares edges with C, D;
B shares edges with D, E, F;
C shares an edge with D;
D shares an edge with E;
E shares an edge with F

This network has **lower algorithmic complexity,** because the program needed to define it is shorter.

Generating code:
A shares edges with
  all other nodes;
B shares an edge with C;
E shares an edge with F

This network has **low algorithmic complexity,** because the program needed to define it is short.

Generating code:
All pairs of nodes
share an edge

Algorithmic complexity cannot be computed for most objects, but it can be approximated to, say, study gene regulatory networks. Recent work suggests that researchers might improve genetic algorithms by biasing evolution in favor of changes that lower their algorithmic complexity.

Serious Games (Simulations), Game AI and the Military Industrial Complex

Lt. Col. Tom Bearden was an analyst of Soviet 'Mind Control' technology for the Department of Defense. After retiring from the military, based on the science he had learned in his analysis, he began working in areas related to 'field propulsion' and 'free energy'. He noted that there was an active suppression campaign in these two areas involving not just government Intelligence but also Corporate espionage. In describing the methodology behind the suppression campaign waged against researchers he noted that it is based in what we would call Reflexive Control:

> ...basically [some people] have knee-jerk reactions or something or their radical or they have some kind of way they interact, which if could be connected with you would be in the area we wish the interaction to occur to get you off into something else totally different then what you're doing. So the next thing you know, here all it takes to set that up maybe a phone call and stimulate the interaction to occur and then the controllers sit back and watch the game go, it's gaming, but it's just like watching a movie scenario one of these days I would probably write a book on gaming and how it's done and what kind of the main games they can use, but I can tell you they're very effective you can get so many different games from so many different walks of life by so many charming folks who are really oily characters that you would not believe it and those come at you in Mass and usually they bury you… (Bearden, 2005, @timestamp: 21:20)

Bearden's claims are supported by a wide number of 'Targeted Individual' (see Neuroweapons Section: Case Study of Civilian Abuse) that claim that they are targeted through a game like scenarios, that they term 'street theatre'. Are there precedents in Military and Defense contracting literature to support such claims. Indeed, there are as we shall read below.

The first video game console was developed by Sanders Associates, which was for a time owned by Lockheed Martin before being sold to British based BAE Systems, which also owns Marconi- a radar company, some of whose scientists mysteriously were murdered in an unnatural cluster leading to conspiracy theories. Sanders developed the game console marketed and branded by Magnovox back in the early 1970s. Defense and gaming have been intertwined for a very long time. Games are also used in the Military. Video Games are now used to train military personnel in real world combat scenarios. As was examined in an earlier chapter there is Reflexive Game theory that is used by Lockheed-Martin and others in their applications. Real-Time Strategy games are also used in military simulations to study such things as strategy and resource management (see RTS below). During the Snowden leaks it was discovered that the NSA and GCHQ had infiltrated on-line gaming communities to study human behavior. It was suggested also by Dr. John Norseen, the Lockheed-Martin engineer responsible for developing BioFusion and Thought Injection that on-line games were used for testing neuroweapons systems.

According to New York Times reporters Mark Mazzetti and Justin Elliott
GCHQ, NSA and the Pentagon created infiltration into the video game world:

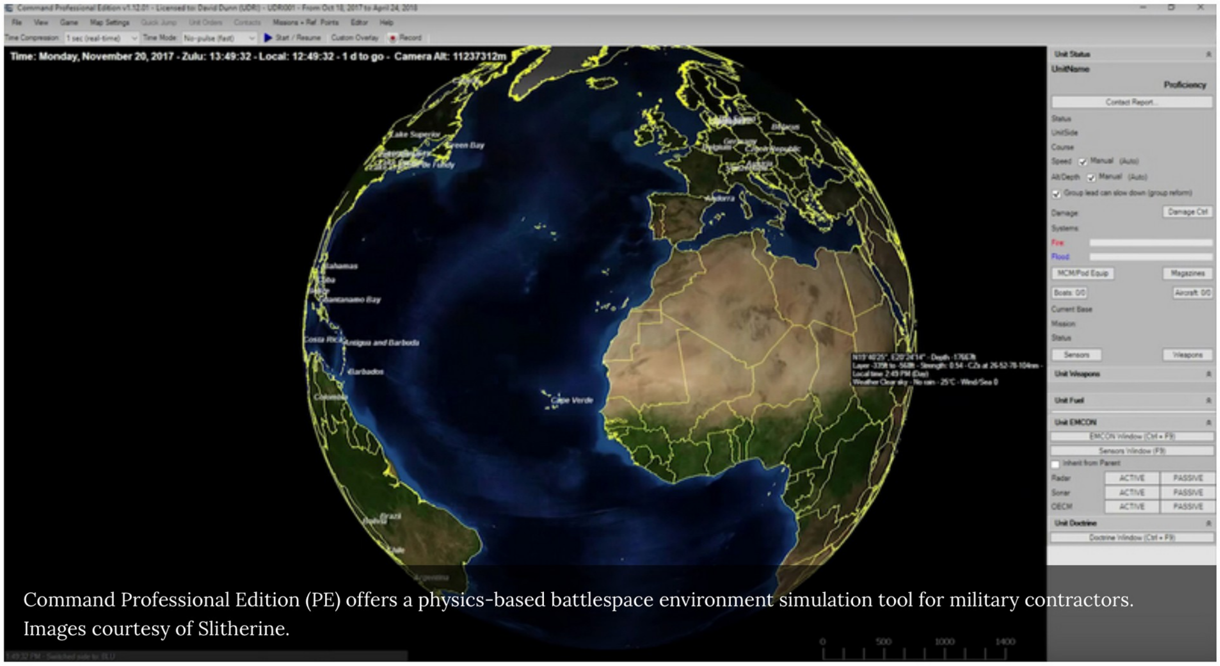
> According to the minutes of a January 2009 meeting, GCHQ’s “network gaming exploitation team” had identified engineers, embassy drivers, scientists and other foreign intelligence operatives to be World of Warcraft players — potential targets for recruitment as agents.
>
> At Menwith Hill, a Royal Air Force base in the Yorkshire countryside that the N.S.A. has long used as an outpost to intercept global communications, American and British intelligence operatives started an effort in 2008 to begin collecting data from World of Warcraft.
>
> The Pentagon’s Special Operations Command in 2006 and 2007 worked with several foreign companies — including an obscure digital media business based in Prague — to build games that could be downloaded to mobile phones, according to people involved in the effort. They said the games, which were not identified as creations of the Pentagon, were then used as vehicles for intelligence agencies to collect information about the users.

With this in mind it is a qood question to ask to what extent are games used in real world software intended for military and intelligence work. We need only look at the literature published by the military and intelligence world to see some of the impacts and then it is good to understand how AI in Games works.

Video games are used in the Defense Industry for such diverse applications from flight simulators, to logitics (operations research), to international relations. A recent privately produced game, Command Professional Edition, has been integrated by BAE Systems, Lockheed-Martin, the Department of Defense and other countries military-industrial complexes:

> Command PE is already being used in several countries including the US, UK, Japan, Ecuador, Finland, Germany and France, and a number of US departments are using it for different purposes. Defence contractor customers include BAE Systems and Lockheed Martin, and the audience is set to grow.
>
> “We have something like 300 products out there, and this has generated interest in other divisions,” says JD McNeil. “Command doesn’t really handle the infantry and the Americans have placed a pretty massive contract with us for an armed warfare simulation that we already do that they want for certain departments of the US infantry. The project’s been given a government title and we received the order for that last week.”

https://www.army-technology.com/features/military-simulation-game/



Command Professional Edition v1.12.01 - Licensed to: David Dunn (UDR) - UDR001 - From Oct 18, 2017 to April 24, 2018

Time: Monday, November 20, 2017 - Zulu: 13:49:32 - Local: 12:49:32 - 1 d to go - Camera Alt: 11237312m

Command Professional Edition (PE) offers a physics-based battlespace environment simulation tool for military contractors. Images courtesy of Slitherine.

Reflexive Rewards and Gaming

It is not hard to see how similar a video game design is to Reflexive Control (see previous chapter, ‘Ch. 4 Lessons from an American Weapons Designer’) and the fact that game theory is employed in Defense contractors publications and applications. According to Dr. Tommy Thompson, an expert in AI and Video Games. Every game is based on rewards:

> Designers consider how reward structures are established within games, given that they are required to maintain your interest in the game. One element that proves useful to appreciate from a design perspective is the notion of ‘compulsion loops’, a theory that has not been around for a particularly long time (read the Gamasutra article by John Hopson from Bungie on his research into the area). https://aiandgames.com/ai-101-part-2/

Rewards are of two kinds: Strong positive reinforcement, and Strong Negative reinforcement. The player in a game learns to do certain things through negative and positive feedback. There are visual, audio and other cues to alert, even subliminally, the player in the game as to what actions are good and bad. Rewards come into play to teach the player how the game works, the rewards can be subtle and simple. Compulsion loops, is a means of driving. You are driven to play the game based on these compulsive instruments in the game. Compulsion loops are of three kinds: Short, Medium and Long term.

AI in Video Game Design: Military Simulations Programming

Video games or simulation software is of several genres. An overview is presented below before we dive into the programming involved in creating military and intelligence simulations or real games.

Game and/or Simulation Genres:

Action - fast-paced requiring quick judgement and snap decisions. Often they can be single player, with AI Agents as team members, fighting adversarial AI agents, or Non-Player Characters (NPCs).

Adventure - a sequence basis to the game, with a rigid structure. With no wide game maps, usually the gamer is presented only with their immediate surroundings.

Role-Playing (RPGs) - gamer takes on the role of a character.

Vehicle Simulation - where the gamer controls a vehicle, such as ‘Pole Position’

Strategy - of course the big dog in this category is ‘Real-Time Strategy Games’ (RTS), which are covered extensively below. In this genre the gamer has a high level view of the game world.

Managment - sometimes referred to as ‘god’ games, Ihas a high-level games, like creating cities or forts, resources can be manipulated in the game world, change it in different ways for the NPCs in the game world.

Puzzle - small games in single player mode, gamer uses logic and deduction to complete goals.

One concept that is important to understand in Simulation Software is that of Game World- the environment the game is played in. Of which there are different types:

Accessible vs. Inaccessible

Accessible-an actor has knowledge of every aspect of the game world and knows of everything that is going on within that game world

Inaccesible-there are limits to what an actor may know about the game world (for example using the concept of fog-of-war, stochastic information or incomplete information about game world)

Environmentally Discrete vs. Environmentally

Continuous

Actors within the game world may make a number of possible actions at any point, determined by the range of potential actions within a game world

Discrete-a finite set of actions that an actor can take (for example only being able to move one square in any one of the cardinal directions on a grid)

Continuous- there is a continuum of possible actions, such as allowing an actor to turn to any direction

Static vs. Dynamic

Static- the game world remains the same until an participant has made a move
Dynamic- the game world alters whilst an participant is “thinking”

Deterministic vs. Non-deterministic

Deterministic-the next state can be explicitly concluded from the present state of the game world and the actions carried out by the actors.

Non-deterministic-there is an element of uncertainty, or if the game world changes despite actions by the actors

Episodic vs. Non-Episodic

Episodic-If an actor can take an action, the results of which have no relation on future actions, the environment
is episodic.

Non-episodic-the consequence of one action relates directly or indirectly to the available information or set of actions at a future point

Turn-Based vs. Real-time

Turn-based-place the players in a game-playing sequence. Whilst this type of game could be, theoretically, applied to any game, there are only a small number of genres where this mechanic is used, primarily 4X, strategy, some role-playing, and some life-simulation games. These games can require a great deal of strategic thinking, and as such having the time to analyse a situation and make decisions based on that is almost necessary.

Semi-Turn-based- the player has the opportunity to pause the game to make decisions or queue up actions, and then return to normal real-time playing afterwards; or where certain sections of the game are turn-based, and
the rest is real-time.

Real-Time- Non-turn-based games

Artificial Intelligence is used in almost all simulations or games currently. There are different AI’s that meet different programming needs in different game genres. As researchers remark, pointing out the use of Neuro-Evolution techniques (AI creating AI):

> However existing work is primarily focussed on the specific implementation of AI methodologies in specific problem areas. For example, the use of neuro-evolution to train behaviour in NERO. With greater analysis of the problems faced in implementing AI methods in computer games, more accurate and efficient methodologies can be developed to create more realistic behaviour of artificial characters within games. (Gunn et al, 2009, 1)

There are no universal methodologies that cross all simulation categories. AI in games can be thought of in the following ways taken from Gunn et al 2009:

Hierarchical Intelligence

> Player-as-manager games provide us with a potential hierarchy of intelligences that would be required: the artificial player and the artificial player's actors. Different AI methods would be required in this case, as the artificial player would require high-level strategic decision making. On the other hand, individual actors might only require reflexive behaviour (i.e. "I am being shot, I shall move away.") Currently in these types of games (especially strategy games) there is little intelligence at the artificial player level, merely consisting of such static tactics as "build up a force of x units, and send them along y path". Observation of such tactics in has shown that there is a reliance on some form of state analysis. By considering the hierarchical nature of the player and the actors under that player's control, suitable mechanisms can be introduced. First to provide adequate highlevel strategic planning for the artificial player. Secondly to provide low-level tactical planning for the artificial player's actors.

Co-operative vs. Non-Cooperative

> Cooperative games are those where the participants can form binding agreements on strategies, and there is a mechanism in place to enforce such behaviour. Non-co-operative games are where every participant is out to maximise their own pay-off. Some games may have elements of both co-operative and non-co-operative behaviour: coalitions of participants enforce co-operative behaviour, but it is still possible for members of the coalition to perform better, or receive better rewards than the others if working alone. These are hybrid games.

Game World

World Interfacing

...building robust and reusable world interfaces using two different techniques: event passing and polling. The event passing system will be extended to include simulation of sensory perception, a hot topic in current game AI. Polling [when the polling objects are also game characters] Looking for interesting information is called polling. The AI code polls various elements of the game state to determine if there is anything interesting that it needs to act on. (Millington, Funge 2016)

> Discrete action games within game theory consist of a finite number of participants, turns, or outcomes, resulting in a finite set of strategies which can be plotted in a matrix format for evaluation. Continuous action games, however, can have participants joining and leaving the game, or the stakes changing between actions, resulting in a continuous set of strategies. This represents a subset of the potential actions that the game world allows. Within our taxonomy, this relates directly to environmentally discrete and environmentally continuous game worlds.

Simultaneous vs. Sequential

> In direct relation to turn-based versus real-time games, sequential games have all the players within a game make their moves in sequence, and one at a time. Simultaneous games are those where any or all players may make their moves at the same time. Classically, sequential games are also called dynamic games: however this would cause confusion in our taxonomy. Sequential games allow the construction of the extensive form of the game - essentially a hybrid decision tree of all players and all possible moves with their rewards.

Information Visibility

> It is not necessary for all participants within a game to have access to all information about the state of the game at any point. The available information can be perfect, where all participants have access to the current state of the game, all possible strategies from the current state, and all past moves made by all other participants. The latter implies that all games that impart perfect information to the participants are by their nature also sequential games. Imperfect games impart partial information about the game to at least one participant. A special case of imperfect information visibility is complete information where all participants are aware of all possible strategies and the current state of the game, however the previous moves by other

participants are hidden.
In this taxonomy, this relates to accessible and inaccessible game worlds. However, a common observation in games is that artificial participants have access to perfect information of the game world, whereas the human player only has imperfect information. This breaks the immersion of the game.

Noisy vs. Clear

Noisy game worlds are those in which there is a significant amount of information that an intelligence must analyse to either make a decision on what to do next, but where not all of that information is appropriate to the goal. Both dynamic and non-deterministic games provide levels of noise: the former due to the fact that the state of the game keeps changing even during the times when the intelligence needs to make a decision or form behaviour from learning; and the latter where there is no clear progressive state from which to base rules and analyse the game world. Although these two definitions provide the clearest example of noise, the level of information available can also create noise. Even in perfect information game worlds it is possible that the available information is an overly large data set for any intelligence, and thus noise is introduced.

Nash Equilibria

Within game theory, a Nash equilibrium (NE) exists where an overall highest level of pay-off for all players takes place. There may be many such equilibria within game strategies for a particular game, or there may be only one - in which case it is a unique NE.

Zero-sum, these are a type of game in game theory that are a special case of general sum games - ones where there is a fixed overall value to winning (or losing) the game. The specific case where for any winning value v, there is a losing value of 0 - v, is a zero-sum game. In other words, what one player wins, the other loses.

Symmetry, In game theory, a two-player game is classed as symmetrical if, for player 1 and player 2, and the matrices A and B containing the stateaction utility values for the strategies for player 1 and player 2 (respectively), A = BT i.e. if player 1 and player 2 swap positions, they can follow the same strategies, with the same pay-offs.

Agents

In this context, “agent-based AI is about producing autonomous characters that take in information from the game data, determine what actions to take based on the information, and carry out those actions. for in-game AI, behaviorism is often the way to go. We are not interested in the nature of reality or mind; we want characters that look right. In most cases, this means starting from human behaviors and trying to work out the easiest way to implement them in softwares. (Millington, Funge 2016)

Reflex Agents
These agents use a conditional statement to provide the “intelligence”. Currently, most actors within games follow reflex systems, to the extent that players can monitor the input-output action pairs of specific actors. Once a pattern has emerged, the human player can modify their strategy sufficiently so that the opponent artificial actor will make a significant loss whilst the human player will make a significant gain. Reflex agents can fall into infinite loops, as there is no concept of context within if-then statements.

Temporal agents can be considered a special sub-group of reflex agents, where actions are carried out after measuring the passing of time. This specific type of agent would be applicable in dynamic and semi-dynamic game worlds, where time is a factor.
Reflex agents, are useful in situations where a high level of complexity is not required by a participant. The more limited the scope of possible actions in a discrete actions game world, for example, the less complexity is required in decision making. In some cases reflex agents might present the best compromise of complexity versus believability.

Model-Based Agents
An agent that monitors the environment and creates a model of it on which to base decisions is called a model-based agent. This type of agent would be best applied to dynamic, real-time games where constant monitoring of the environment is required on which to base decisions on actions. This would also be highly beneficial in a cooperative game, where although the actions of other actors are independent, they are inter-related, and so a broader monitoring range covering other co-operating actors can be introduced.
Goal-Based Agents
Using a model of the environment, goals can be created and planning carried out to achieve those goals, even within inaccessible

game worlds and with other participants. Although the artificially controlled participants will generally have broad goals built in to determine their over-all behaviour (such as “stop the human-player at all costs”), there is still scope within that command to create subgoals (such as “find the player”) Goal-based agents are also highly beneficial in inaccessible game worlds, as they can change their own sub-goals as the information they are made aware of changes

Utility-Based Agents

A further refinement on the model- and goal-based agent methodologies is the ability to manage multiple goals at the same time, based on the current circumstances. By applying utility theory to define the relative “best” goal in any situation, we have utility-based agents.

Swarm Intelligence

Swarm intelligences have also been used to compute NE [Nash Equilibria]. Ant colony modelling provides a strong methodology for actors to explore the game world to complete goals by providing path-finding around obstacles and creating search patterns to achieve their goals - something that has been seen to be lacking in games. Exploring the game world is important in imperfect information worlds, and such group goal finding resulting from ant colony optimisation is useful in co-operative game play.

Believable Agents

These agents are expected to behave as a human would in similar situations. Given this is one of the core purposes of developing better AI in games, any agents that are developed should fall into this category, unless (through the narrative) the actor is expected to behave differently. Even then they should be consistent in their behaviour which could be construed as providing believability in the behaviour across all actor types. Specifically, player-as-manager games require a great deal of believability given the large number of actors available to the player, and the semi-autonomous nature of those actors. Player-as-actor games will also require a high level of believability, as all the interactions with other actors in the game world must provide a sufficient level of immersion, give

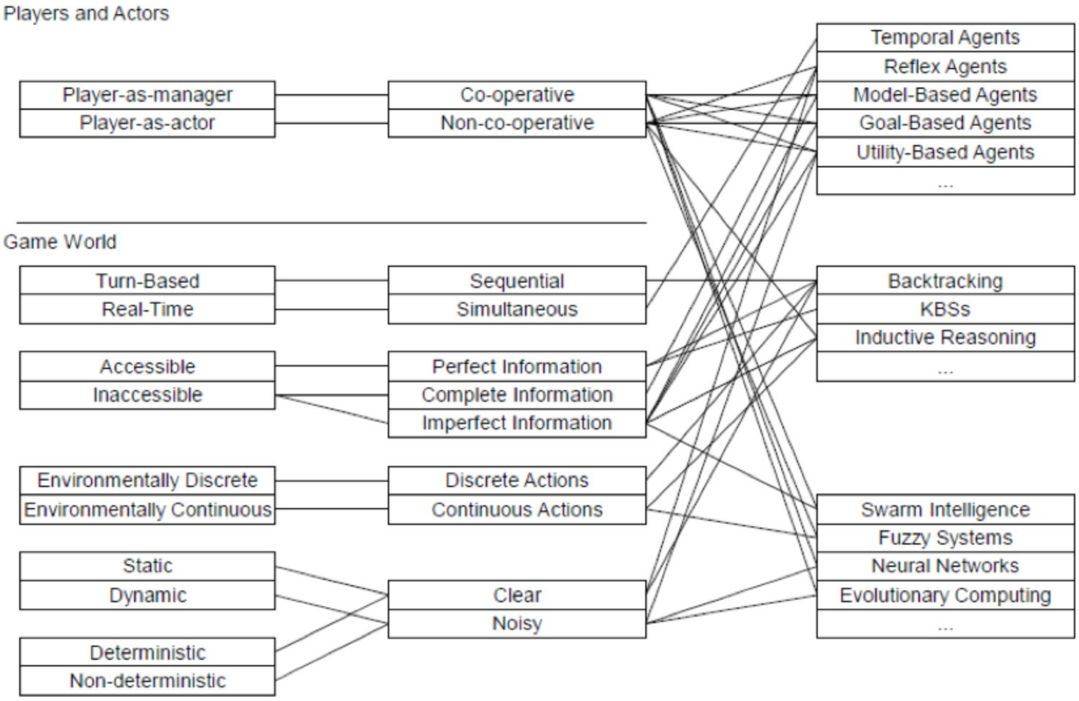


Figure 1. The complete taxonomy

Chart of Video Game Taxonomy (Gunn et al, 2009)

Packaging Simulations: Elements of AI in Games

Video Game AI is comprised of several different elements of Artificial Intelligence. Most NPCs in games are AI agents. With each game having something similar to a Director System, which manages the overall game. Agents are aware of each other and communicate with each other, while some have dialogue trees assigned to their agent behaviour, there are strictly hidden means of communication between agents as well as different AI sub-systems that manage different elements of the Game. You will encounter AI management systems that even cover Diplomacy in the Game (i.e. Total War) as well as battle AI management and Resource Management AI. As well you can have Finance Managers in the game and Nemisys Systems which manage opponents in the game, some even using Shadow AI to mimic a players tactics, therefore fighting as the gamer plays the game adaptig the players strategies into their own counter-measures against the player.

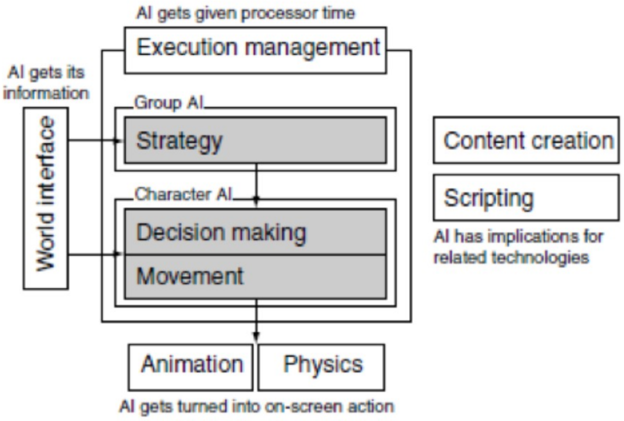


Figure 1.1 The AI model

Some of the common architectures used in games are polling, events, event managers and Sense Management. Each of which has it's own domains of control and delegation. The following is based on the work of Millington and Funge (2016).

Polling is the part in the game which keeps track of all the actions, goals and data in the game.

> The polling can rapidly grow in processing requirements through sheer numbers, even though each check may be very fast. For checks that need to be made between a character and a lot of similar sources of information, the time multiplies rapidly. For a level with a 100 characters, 10,000 trajectory checks would be needed to predict any collisions. Because each character is requesting information as it needs it, polling can make it difficult to track where information is passing through the game. Trying to debug a game where information is arriving in many different locations can be challenging. (Millington, Funge 2016)

Polling Stations

> There are ways to help polling techniques become more maintainable. A polling station can be used as a central place through which all checks are routed. This can be used to track the requests and responses for debugging. It can also be used to cache data (Millington, Funge 2016)

Events

we want a central checking system that can notify each character when something important has happened. This is an event passing mechanism. A central algorithm looks for interesting information and tells any bits of code that might benefit from that knowledge when it finds something.
The event mechanism can be used in the siren example. In each frame when the siren is sounding, the checking code passes an event to each character that is within earshot. This approach is used when we want to simulate a character's perception in more detail…. The event mechanism is no faster in principle than polling. Polling has a bad reputation for speed, but in many cases event passing will be just as inefficient. To determine if an event has occurred, checks need to be made. The event mechanism still needs to do the checks, the same as for polling. In many cases, the event mechanism can reduce the effort by doing everybody's checks at once. However, when there is no way to share results, it will take the same time as each
character checking for itself. In fact, with its extra message-passing code, the event management approach will be slower.
(Millington, Funge 2016)

EventManagers

Event passing is usually managed by a simple set of routines that checks for events and then processes and dispatches them. Event managers form a centralized mechanism through which all events pass. They keep track of characters' interests (so they only get events that are useful to them) and can queue events over multiple frames to smooth processor use. (Millington, Funge 2016)

An event-based approach to communication is centralized. There is a central checking mechanism, which notifies any number of characters when something interesting occurs. The code that does this is called an event manager.

The event manager consists of four elements:

1. A checking engine (this may be optional)
2. An event queue
3. A registry of event recipients
4. An event dispatcher

The interested characters who want to receive events are often called "listeners" because they are listening for an event to occur. This doesn't mean that they are only interested in simulated sounds. The events can represent sight, radio communication, specific times (a character goes home at 5 P.M. for example), or any other bit of game data. The checking engine needs to determine if anything has happened that one of its listeners may be interested in. It can simply check all the game states for things that might possibly interest any character, but this may be toomuch work.More efficient checking engines take into consideration the interests of its listeners.
(Millington, Funge 2016)

Event Casting

There are two different philosophies for applying event management. You can use a few very general event managers, each sending lots of events to lots of listeners. The listeners are responsible for working out whether or not they are interested in the event. Or, you can use lots of specialized event managers. Each will only have a few listeners, but these listeners are likely to be interested in more of the events it generates. The listeners can still ignore some events, but more will be delivered correctly. The scattergun approach is called broadcasting, and the targeted approach is called narrowcasting. Both approaches solve the problem of working out which agents to send which events. Broadcasting solves the problem by sending them everything and letting them work out what they need. Narrowcasting puts the responsibility on the programmer: the AI needs to be registered with
exactly the right set of relevant event managers. This approach is called broadcasting. A broadcasting event manager sends lots of events to its listeners. Typically, it is used to manage all kinds of events and, therefore, also has lots of listeners. (Millington, Funge 2016)

Inter-Agent Communication

While most of the information that an AI needs comes from the player's actions and the game environment, games are increasingly featuring characters that cooperate or communicate with each other. A polling station has two purposes. First, it is simply a cache of polling information that can be used by multiple characters. Second, it acts as a go-between from the AI to the game level. Because all requests pass through this one place, they can be more easily monitored and the AI debugged.
(Millington, Funge 2016)

Sense Management

Up until the mid-1990s, simulating sensory perception was rare (at most, a ray cast check was made to determine if line of sight existed). Since then, increasingly sophisticated models of sensory perception have been developed. In games such as Splinter Cell [Ubisoft Montreal Studios, 2002], Thief: The Dark Project [Looking Glass Studios, Inc., 1998], and Metal Gear Solid [Konami Corporation, 1998], the sensory ability of AI characters forms the basis of the gameplay.
Indications are that this trend will continue. AI software used in the film industry (such as Weta's Massive) and military simulation use comprehensive models of perception to drive very sophisticated group behaviors. It seems clear that the sensory revolution will become an integral part of real-time strategy games and platformers, as well as third-person action games.

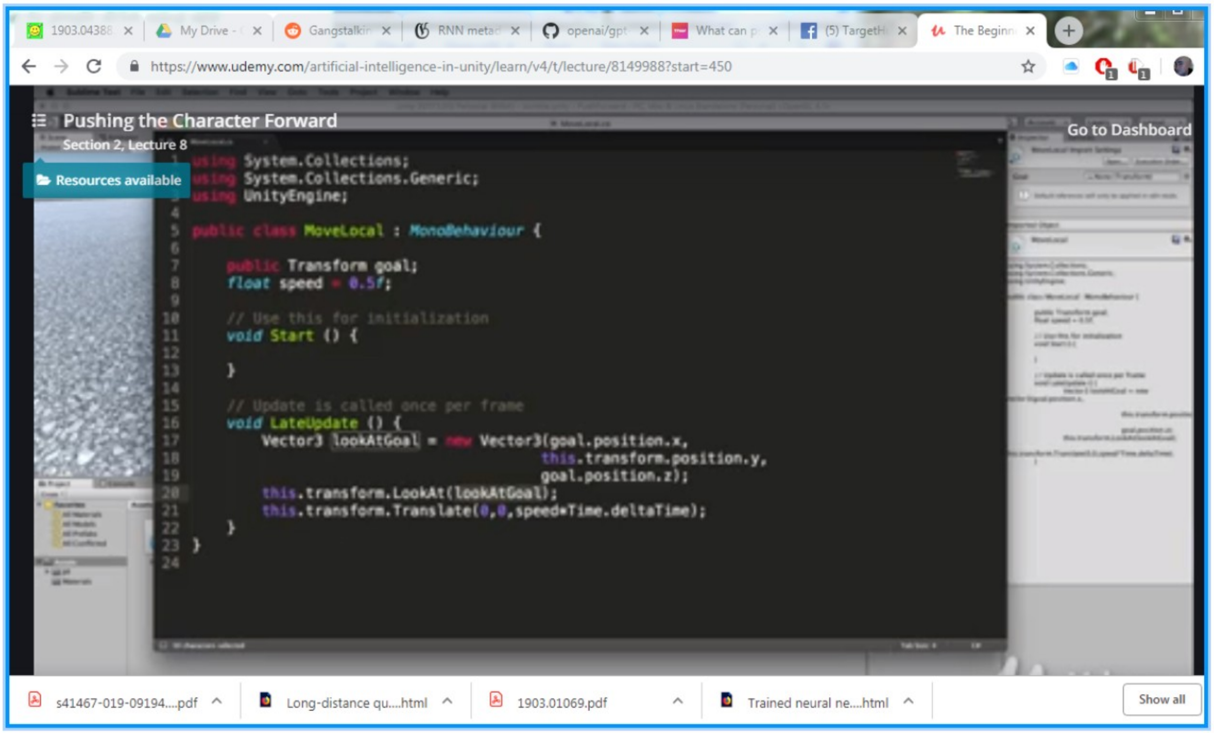
> A more sophisticated approach uses event managers or polling stations to only grant access to the information that a real person in the game environment might know. At the final extreme, there are sense managers distributing information based on a physical simulation of the world.Even in a game with sophisticated sense management, it makes sense to use a blended approach. Internal knowledge is always available, but external knowledge can be accessed in any of the following three ways: direct access to information, notification only of selected information, and perception simulation.
> We will exclusively use an event-based model for our sense management tools. Knowledge from the game state is introduced into the sense manager, and those characters who are capable of perceiving it will be notified. They can then take any appropriate action, such as storing it for later use or acting immediately.
> (Millington, Funge 2016)

Overview of search

Most of this is managed either through Finite State Machines, Markov Chains, Behaviour Trees, all of which are outgrowths of what is known as Graph Searching. Graph searching is a means of searching interconnected nodes in a matrix or network. Some early examples of Graph Search are Depth First Search, and Breadth First Search. Where one searches the graph tree horizontally first, as in Breadth First, the other searching through one depth to the next as in Depth First. In the early years of AI at the Stanford Research Institute, which was a CIA contractor that in the early days conducted research in Remote Viewing as well as AI and Robotics, for their robotic research came up with A* search, which examines edge weights and costs to give a heuristic approach to AI search, a heuristic is a rule of thumb, an approximate solution that might work in many situations but is unlikely to work in all.
. The main examination in determining a heuristic approach is how well the particularly path or trajectory of a search result achieves the goal, which is to win the game. Another area that SRI was involved in was developing Planning systems in Simulations, known as STRIPS, which we shall cover in the Goal Oriented Action Planning section.

Moving around

All games utilize the foundational mechanics in Games for moving characters around. Since it is based on statistical learning, you see a usually non-human nature to their movements. For instance, one thing a game character must do to target the opposing player is look at the player, automatically zeroing in on the player. This is known as the LookAt() method in movement libraries and is a fairly universal component of video games. Some Targeted Individuals report that routinely they are targeted through involuntary staring on the part of hypnotized people around them. That no matter how obscure their location that people entering their area LookAt() them.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MoveLocal : MonoBehaviour {

    public Transform goal;
    float speed = 0.5f;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void LateUpdate () {
        Vector3 lookAtGoal = new Vector3(goal.position.x,
                                        this.transform.position.y,
                                        goal.position.z);
        this.transform.LookAt(lookAtGoal);
        this.transform.Translate(0,0,speed*Time.deltaTime);
    }
}
```

LookAt() function with in the code of a game (De Byl 2019)

Moving in a game is based on vector mathematics, a ‘array’ of three elements.
A point is a location in space. A vector is a direction and length. A vector has no predetermined starting point. There is only one point in space that has its coordinates, however vectors with the same values can be anywhere. The x,y,z of the point is a single location in space, the x,y,z of a vector is the length of the vector in each of those dimensions.

Vector Mechanics:
  a point plus a vector will result in a point
  a vector plus a vector will result in a vector
In games we add vectors to points to move objects around. For example, the movement of characters from one location to another. The vector calculated between one point and another tells us the direction and distance.

If Stevie (the zombie) wants to get from her position to Granny's position then the vector she must travel is granny's position minus Stevie's position. For example, if Stevie was at (10,15, 5) and Granny was at (20, 15, 20) then the vector from Stevie to Granny would be (20, 15, 20) - (10, 15, 5) = (10, 0, 15). If Granny wanted to find her vector to Stevie the equation is reversed.

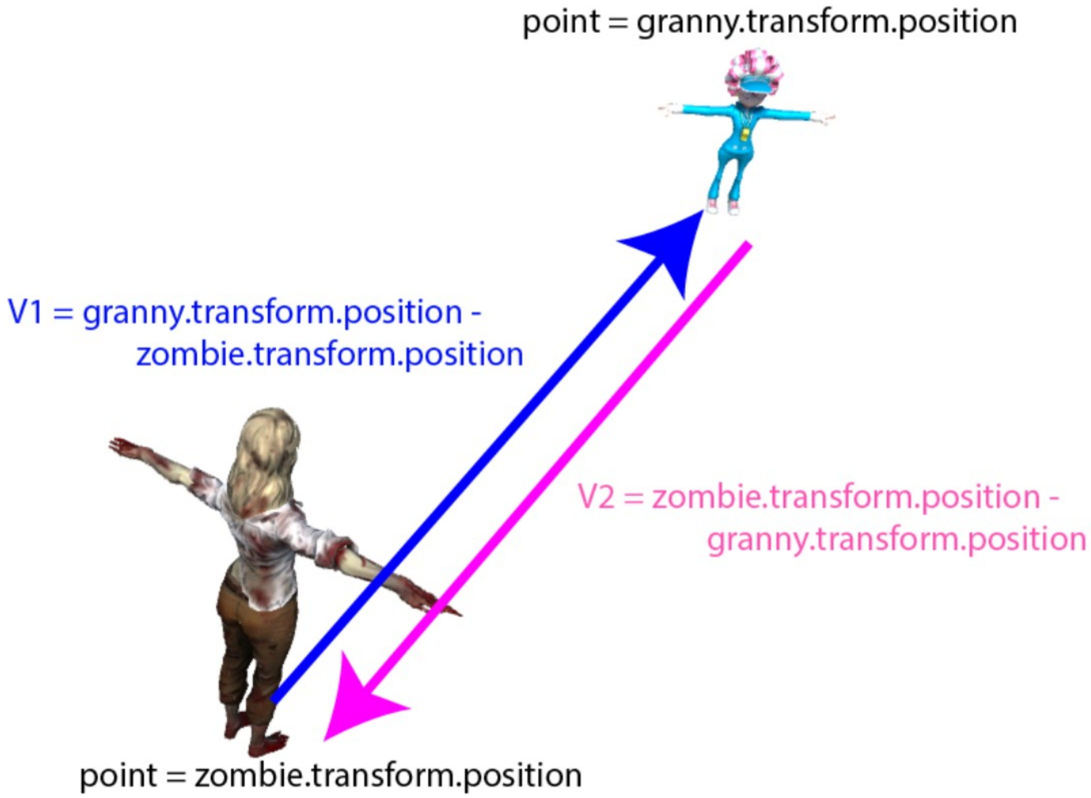

point = granny.transform.position
V1 = granny.transform.position - zombie.transform.position
V2 = zombie.transform.position - granny.transform.position
point = zombie.transform.position

Fig 2. Calculating Vectors Between Points

(De Byl 2019)

Then to move Stevie from her current location to that of Granny you would add the vector (V1) to Stevie's position thus: (10,15, 5) + (10, 0, 15) = (20, 15, 20) which you can check is correct because it's the position of Granny!

Vectors can also be added together to give a total direction and magnitude.
The length of a vector is called its magnitude. When the direction toward a character is calculated as we've done in the previous examples, by taking one position away from another, the resulting length of that vector is the distance between the characters.

In games distance between locations is used by decision making AI as well as moving objects around. For example, an NPC might work out the distance to a player before deciding whether to attack or not. In determining the direction in which to travel to get from one location to another you might also require an angle that indicates how much a character needs to turn to be facing that location, otherwise you'll get a character that moves sideways. Once you have calculated the angle between the way the character is facing and the direction it is about to travel you'll be able to program its turning.

The smoothing of a NPCs movement is a product of statistical mechanics. Instead of making jagged or quick movements it will walk in an arc like fashion or take a straight angle approach. This is also noticed in Self-Driving or Vehicle Simulations: cars proceed in smooth accelerations, decelerations, turning is rounder, etc. All due to the AI used in their systems for managing movements and being statistical based rather then natural or chaotic. Which of course does not give a totally satisfactory simulation of real-time conditions in a real world, but for training purposes it may be good enough if you can account for these unnatural attributes and they do not become ingrained in muscle memory of the soldiers.

To move an agent NPC around, you could rely on older technology such as waypoints: a reference point used for navigation purposes by in-game characters. Most commonly used in strategy games and squad based games. For instance if you wanted to statically block an entrance you would simple give the coordinates to a waypoint near the door and have the NPC proceed to that waypoint. Single-Rail games are a good example of using waypoints. A newer approach is to use Navmeshes: in use since the mid-80s in robotics as meadow maps, became part of video game code around 2000, an abstract data structure used in artificial intelligence applications to aid agents in pathfinding through complicated spaces.

Aside from moving individual NPCs you can also group NPCs to move as flocks or as swarms, both video games and military drones use behaviour trees for complicated actions including swarming.

Some common movement related elements of game design are presented below, again based on the work of (Millington, Funge 2016):

Sight Cone

> a sight cone of around 60◦ is often used. It takes into account normal eye movement, but effectively blinds the character to the large area of space it can see but is unlikely to pay any attention to. (Millington, Funge 2016)

Movement

> Movement refers to algorithms that turn decisions into some kind of motion. When an enemy character without a gun needs to attack the player in Super Mario Sunshine [Nintendo Entertainment, Analysis and Development, 2002], it first heads directly for the player. When it is close enough, it can actually do the attacking. The decision to attack is carried out by a set of movement algorithms that home in on the player’s location. Only then can the attack animation be played and the player’s health be depleted. Movement algorithms can be more complex than simply homing in. the AI needs information from the game to make sensible decisions. This is sometimes called “perception” (especially in academic AI): working out what information the character knows. In practice, it is much broader than just simulating what each character can see or hear, but includes all interfaces between the game world and the AI. This world interfacing is often a large

> proportion of the work done by an AI programmer, and in our experience it is the largest proportion of the AI debugging effort. (Millington, Funge 2016)

Steering Behaviour-

> Steering behaviors is the name given by Craig Reynolds to his movement algorithms; they are not kinematic, but dynamic. Dynamic movement takes account of the current motion of the character. A dynamic algorithm typically needs to know the current velocities of the character as well as its position. A dynamic algorithm outputs forces or accelerations with the aim of changing the velocity of the character. Craig Reynolds also invented the flocking algorithm used in countless films and games to animate flocks of birds or herds of other animals….Because flocking is the most famous steering behavior, all steering (in fact, all movement) algorithms are sometimes wrongly called “flocking.” (Millington, Funge 2016)

Characters as Points

> Although a character usually consists of a three-dimensional (3D) model that occupies some space in the game world, many movement algorithms assume that the character can be treated as a single point. Collision detection, obstacle avoidance, and some other algorithms use the size of the character to influence their results, but movement itself assumes the character is at a single point. This is a process similar to that used by physics programmers who treat objects in the game as a “rigid body” located at its center of mass. Collision detection and other forces can be applied to anywhere on the object, but the algorithm that determines the movement of the object converts them so it can deal only with the center of mass. (Millington, Funge 2016)

Seek

> A kinematic seek behavior takes as input the character’s and its target’s static data. It calculates the direction from the character to the target and requests a velocity along this line. The orientation values are typically ignored, although we can use the getNewOrientation function above to face in the direction we are moving. (Millington, Funge 2016)

Steering Behaviors

> Steering behaviors extend the movement algorithms in the previous section by adding velocity and rotation. They are gaining larger acceptance in PC and console game development. In some genres (such as driving games) they are dominant; in other genres they are only just beginning to see serious use.
>
> Obstacle avoidance behaviors take a representation of the collision geometry of the world. It is also possible to specify a path as the target for a path following behavior.

In these behaviors some processing is needed to summarize the set of targets into something that the behavior can react to. This may involve averaging properties of the whole set (to find and aim for their center of mass, for example) (Millington, Funge 2016)

Variable Matching

The simplest family of steering behaviors operates by variable matching: they try to match one or more of the elements of the character's kinematic to a single target kinematic. (Millington, Funge 2016)

Seek and Flee

Seek tries to match the position of the character with the position of the target. Exactly as for the kinematic seek algorithm, it finds the direction to the target and heads toward it as fast as possible. Because the steering output is now an acceleration, it will accelerate as much as possible. Seek will always move toward its goal with the greatest possible acceleration (Millington, Funge 2016)

Velocity Matching

So far we have looked at behaviors that try to match position with a target.We could do the same with velocity, but on its own this behavior is seldom useful. It could be used to make a character mimic the motion of a target… [useful in psychological operations] (Millington, Funge 2016)

Face

The face behavior makes a character look at its target. It delegates to the align behavior to perform the rotation but calculates the target orientation first. Wander, the wander behavior controls a character moving aimlessly about (Millington, Funge 2016)

Path Following

So far we've seen behaviors that take a single target or no target at all. Path following is a steering behavior that takes a whole path as a target. A character with path following behavior should move along the path in one direction.
Path following, as it is usually implemented, is a delegated behavior. It calculates the position of a target based on the current character location and the shape of the path. It then hands its target off to seek. There is no need to use arrive, because the target should always be moving along the path.We shouldn't need to worry about the character catching up with it. The target position is calculated in two stages. First, the current character position is mapped to the nearest point along the path. This may be a complex process, especially if the path is curved or made up of many line segments. Second, a

target is selected which is further along the path than the mapped point by a fixed distance. (Millington, Funge 2016)

Separation

The separation behavior is common in crowd simulations, where a number of characters are all heading in roughly the same direction. It acts to keep the characters from getting too close and being crowded. (Millington, Funge 2016)

Attraction

Using the inverse square law, we can set a negative valued constant of decay and get an attractive force. The character will be attracted to others within its radius, but this is rarely useful. Some developers have experimented with having lots of attractors and repulsors in their level and having character movement mostly controlled by these. Characters are attracted to their goals and repelled from obstacles, for example. Despite being ostensibly simple, this approach is full of
traps for the unwary. [useful in psychological operations] (Millington, Funge 2016)

Collision Avoidance

In urban areas, it is common to have large numbers of characters moving around the same space. These characters have trajectories that cross each other, and they need to avoid constant collisions with other moving characters.
A simple approach is to use a variation of the evade or separation behavior, which only engages if the target is within a cone in front of the character. (Millington, Funge 2016)

Obstacle and Wall Avoidance

The collision avoidance behavior assumes that targets are spherical. It is interested in avoiding getting too close to the center point of the target.
This can also be applied to any obstacle in the game that is easily represented by a bounding sphere. Crates, barrels, and small objects can be avoided simply this way.
The obstacle and wall avoidance behavior uses a different approach to avoiding collisions. The moving character casts one or more rays out in the direction of its motion. If these rays collide with an obstacle, then a target is created that will avoid the collision, and the character does a basic seek on this target. Typically, the rays are not infinite. They extend a short distance ahead of the character (usually a distance corresponding to a few seconds of movement).
Decision trees, state machines, and blackboard architectures have all been used to control steering behaviours. (Millington, Funge 2016)

Targeters

Targeters generate the top-level goal for a character. There can be several targets: a positional target, an orientation target, a velocity target, and a rotation target. We call each of these elements a channel of the goal (e.g., position channel, velocity channel). All goals in the algorithm can have any or all of these channels specified. An unspecified channel is simply a “don’t care.” Individual channels can be provided by different behaviors (a chase-the-enemy targeter may generate the positional target, while a look-toward targeter may provide an orientation target), or multiple channels can be requested by a single targeter. (Millington, Funge 2016)

Decomposers

Decomposers are used to split the overall goal into manageable sub-goals that can be more easily achieved. (Millington, Funge 2016)

Constraints

Constraints limit the ability of a character to achieve its goal or sub-goal. They detect if moving toward the current sub-goal is likely to violate the constraint, and if so, they suggest a way to avoid it. Constraints tend to represent obstacles: moving obstacles like characters or static obstacles like walls. (Millington, Funge 2016)

The Actuator

Unlike each of the other stages of the pipeline, there is only one actuator per character. The actuator’s job is to determine how the character will go about achieving its current sub-goal. Given a sub-goal and its internal knowledge about the physical capabilities of the character, it returns a path indicating how the character will move to the goal. The actuator also determines which channels of the sub-goal take priority and whether any should be ignored. (Millington, Funge 2016)

Coordinated Movement

Games increasingly require groups of characters to move in a coordinated manner. Coordinated motion can occur at two levels. The individuals can make decisions that compliment each other, making their movements appear coordinated. Or they can make a decision as a whole and move in a prescribed, coordinated group. (Millington, Funge 2016)

Emergent Formations

> Emergent formations provide a different solution to scalability. Each character has its own steering system using the arrive behavior. The characters select their target based on the position of other characters in the group. (Millington, Funge 2016)

Overview of Decision Making

Decision making in games is comprised of several different methodologies to achieve desired AI Agent optimality. Each game has different methods that could be used, along with some that are seen in more then one genre of game, such as Goal Oriented Action Planning and Behavior Trees. The following concise overview of Decision Making is from (Millington, Funge 2016):

> In reality, decision making is typically a small part of the effort needed to build great game AI. Most games use very simple decision making systems: state machines and decision trees.
>
> Rule-based systems are rarer, but important. The character processes a set of information that it uses to generate an action that it wants to carry out. The input to the decision making system is the knowledge that a character possesses, and the output is an action request. The knowledge can be further broken down into external and internal knowledge. External knowledge is the information that a character knows about the game environment around it: the position of other characters, the layout of the level, whether a switch has been thrown, the direction that a noise is coming from, and so on. Internal knowledge is information about the character's internal state or thought processes: its health, its ultimate goals, what it was doing a couple of seconds ago, and so on. Typically, the same external knowledge can drive any of the algorithms in this chapter, whereas the algorithms themselves control what kinds of internal knowledge can be used (although they don't constrain what that knowledge represents, in game terms).
>
> Actions, correspondingly, can have two components: they can request an action that will change the external state of the character (such as throwing a switch, firing a weapon,moving into a room) or actions that only affect the internal state (see Figure 5.2). Changes to the internal state are less obvious in game applications but are significant in some decision making algorithms. They might correspond to changing the character's opinion of the player, changing its emotional state, or changing its ultimate goal. Again, algorithms will typically have the internal actions as part of their makeup, while external actions can be generated in a form that is identical for each algorithm.
> The format and quantity of the knowledge depend on the requirements of the game. Knowledge representation is intrinsically linked with most decision making algorithms. It is difficult to be completely general with knowledge representation… Actions, on the other hand, can be treated more consistently.

Decision Trees

A decision tree is made up of connected decision points. The tree has a starting decision, its root. For each decision, starting from the root, one of a set of ongoing options is chosen.

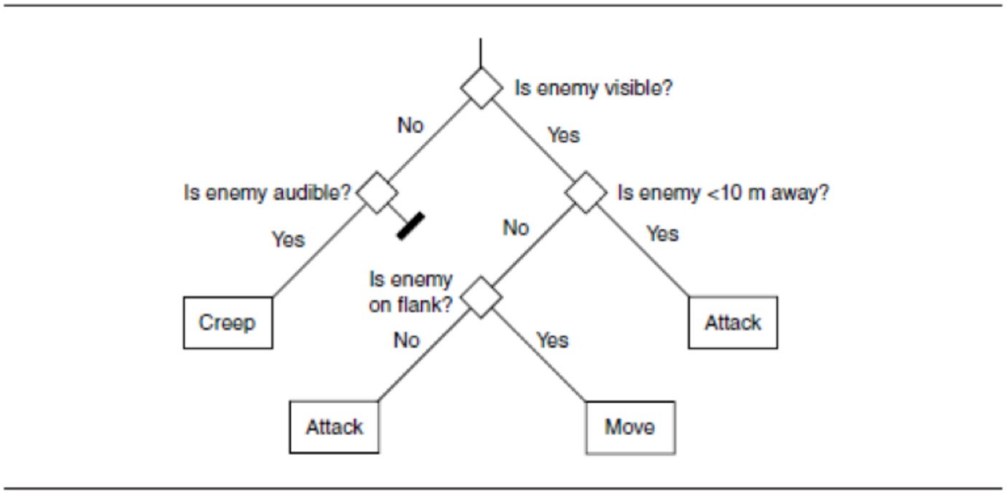


Figure 5.3 A decision tree

(Millington, Funge 2016)

Finite state machines

In Simulation development, we have what is known as the state, not to be confused with quantum state.

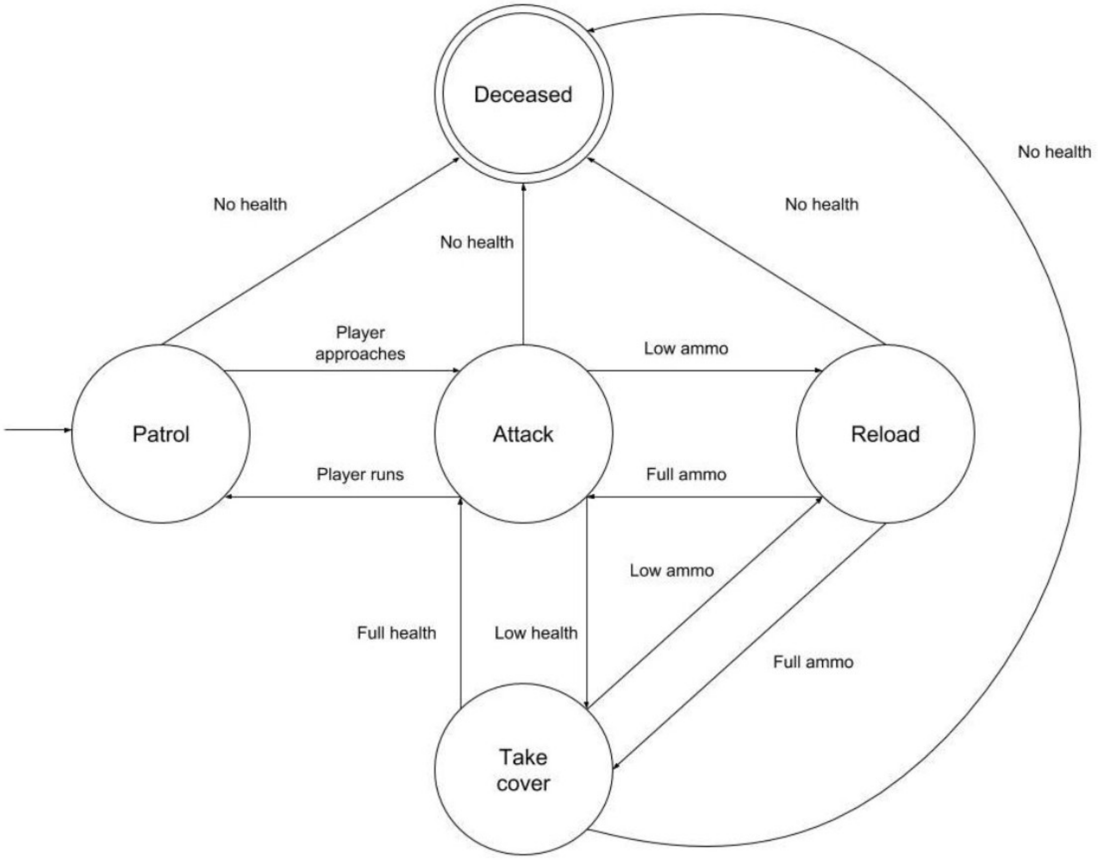
> State machines are the technique most often used for this kind of decision making and, along with scripting...make up the vast majority of decision making systems used in current games. State machines take account of both the world around them (like decision trees) and their internal makeup (their state). States are connected together by transitions. Each transition leads from one state to another, the target state, and each has a set of associated conditions. If the game determines that the conditions of a transition are met, then the character changes state to the transition’s target state. (Millington, Funge 2016)

So how can we generate complex behaviors out of a fairly simple codebase. Complex behaviors were first generated in video games (perhaps, most famously in Batman Arkham) as the first step in the evolution of ever increasing complex behaviors from NPCs.

> A Finite State Machine is a model of computation, i.e. a conceptual tool to design systems. It processes a sequence of inputs that changes the state of the system. When all the input is processed, we observe the system's final state to determine whether the input sequence was accepted or not. (Sanatan 2019)

Software Engineer Marcus Sanatan provides a very good overview of FSM:

> Enemy AI
> Finite State Machines allows us to map the flow of actions in a game's computer-controlled players. Let's say we were making an action game where guards patrol an area of the map. We can have a Finite State Machine with the following properties:
>
> States: For our simplistic shooter we can have: Patrol, Attack, Reload, Take Cover, and Deceased.
> Initial State: As it's a guard, the initial state would be Patrol.
> Accepting States: An enemy bot can no longer accept input when it's dead, so our Deceased state will be our accepting one.
> Alphabet: For simplicity, we can use string constants to represent a world state: Player approaches, Player runs, Full health, Low health, No health, Full ammo, and Low ammo.
> Transitions: As this model is a bit more complex than traffic lights, we can separate the transitions by examining one state at a time:
>
> Patrol
> If a player approaches, go to the Attack state.
> If we run out of health, go to the Deceased state.
>
> Attack
> If ammo is low, go to the Reload state.
> If health is low, go to the Take Cover state.
> If the player escapes, go to the Patrol state.
> If we run out of health, go to the Deceased state.
>
> Reload
> If ammo is full, go to the Attack state.
> If health is low, go to the Take Cover state.
> If we run out of health, go to the Deceased state.
>
> Take Cover
> If health is full, go to the Attack state.
> If ammo is low, go to the Reload state.
> If we run out of health, go to the Deceased state.
>
> This Finite State Machine can be drawn as follows:

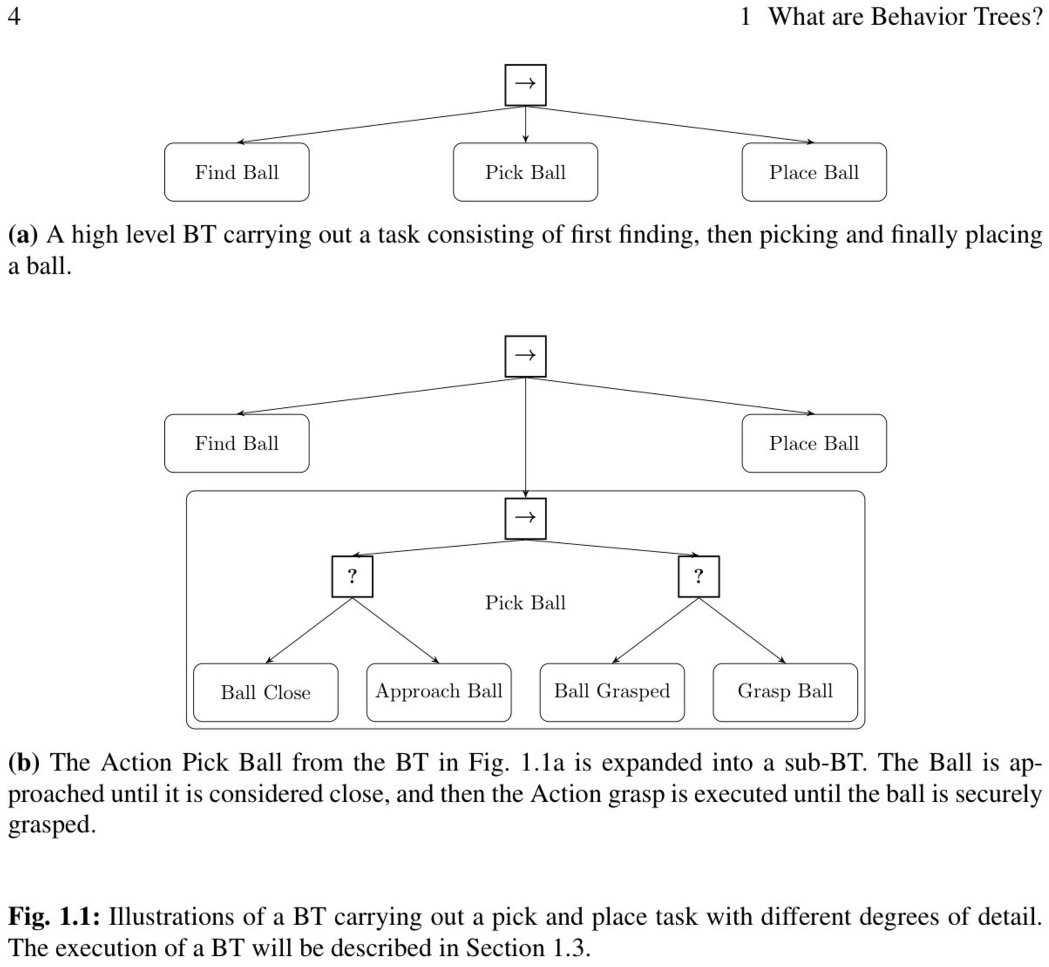


(Sanatan 2019)

Behavior Trees

The downfall of many FSMs is their simplicity, though it is true you can generate many complex patterns from a simple 0,1 cellular automata, out of which you may even create chaos. The desire for having more advanced and nuanced which is to say 'believable' NPCs led to the creation of what was known as Behavior Trees.

> A Behavior Tree (BT) is a way to structure the switching between different tasks in an autonomous agent, such as a robot or a virtual entity in a computer game. An example of a BT performing a pick and place task can be seen in Fig. 1.1a [see below]. As will be explained, BTs are a very efficient way of creating complex systems that are both modular and reactive. These properties are crucial in many applications, which has led to the spread of BT from computer game programming to many branches of AI and Robotics. (2019)



4 1 What are Behavior Trees?

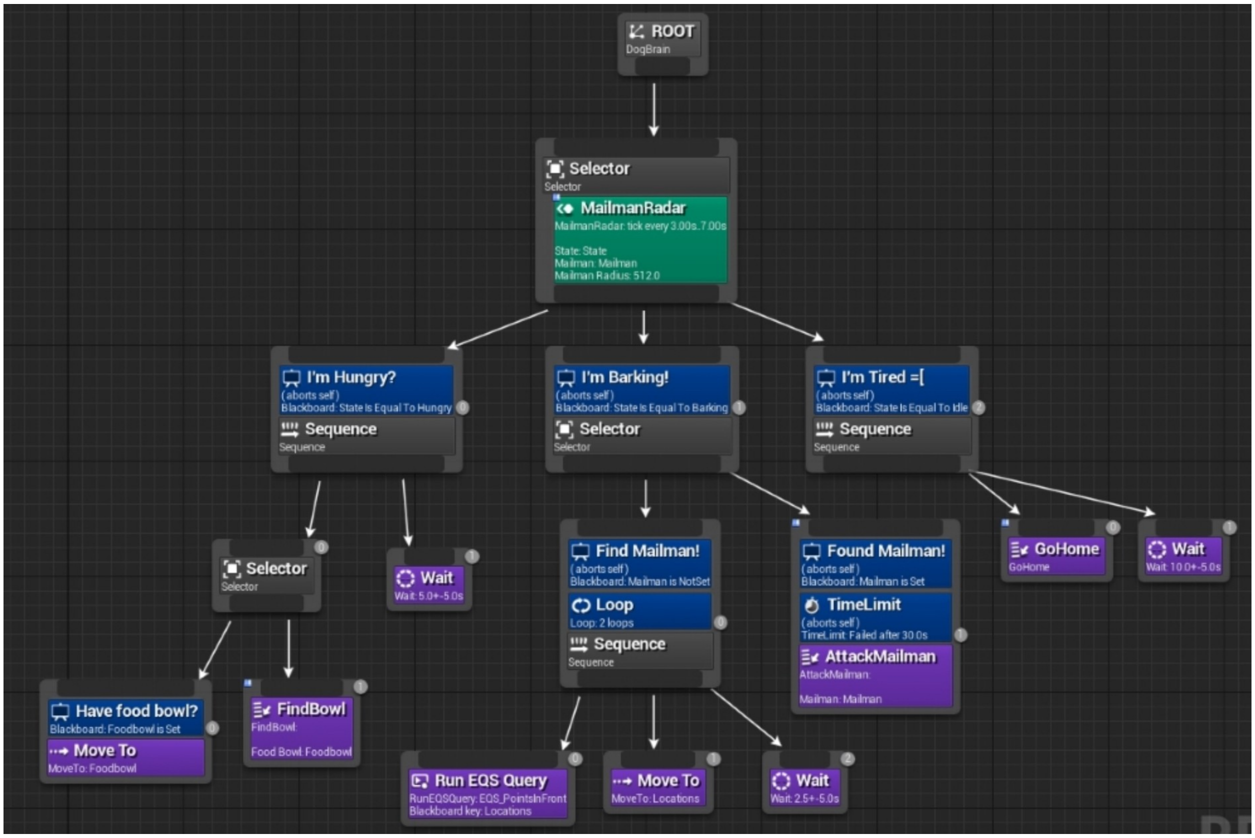
(a) A high level BT carrying out a task consisting of first finding, then picking and finally placing a ball.

(b) The Action Pick Ball from the BT in Fig. 1.1a is expanded into a sub-BT. The Ball is approached until it is considered close, and then the Action grasp is executed until the ball is securely grasped.

Fig. 1.1: Illustrations of a BT carrying out a pick and place task with different degrees of detail. The execution of a BT will be described in Section 1.3.

Behavior Trees are Event-Driven: All NPC's (non-player characters) continue to execute a specific behaviour until an event (by friend or foe) triggers a change-up. The Director system or other management AI must monitor the ‘environment’ of the game, then passes on sensory feedback to the AI Agent NPC to change it’s behavior given what is happening in the environment. It keeps track of changes in the game, depending on the frame rate of the game usually 60fps, via observers or polling agents, whose job it is to keep track of the game and update the Director AI.

As an example of different things an NPC can do with Behavior Trees is that of Squad Triggers when playing in a group of NPCs with the gamer, or enemy AI NPC groups or squads. Squad Triggers - volumes placed in the unreal engine editor that trigger specific responses in non-player characters. When players walk into a AI NPC what behaviours to execute are triggered and specific voice lines to occur and allow the story to play out. Preferred locations and allowed positions. It forces the main game characters to adhere to the story and allow it to be played out in the game. (Steers the game to its desired end or goal) Enemy NPCs are able to shout and chatter, call outs, move position, reload, panic/flee, etc. Ally AI and Enemy AI part of one AI system to move the story. Akin to behaviouralism's notion of positive and negative feedback.



A flow chart for a Behavior Tree from a video game

Again Millington and Funge give more information on the use of Behavior Trees:

> Behavior trees have become a popular tool for creating AI characters. Halo 2 [Bungie Software, 2004] was one of the first high-profile games for which the use of behavior trees was described in detail and since then many more games have followed suit. They are a synthesis of a number of techniques that have been around in AI for a while: Hierarchical State Machines, Scheduling, Planning, and Action Execution. Their strength comes from their ability to interleave these concerns in a way that is easy to understand and easy for non-programmers to create. Despite their growing ubiquity, however, there are things that are difficult to do well in behavior trees, and they aren't always a good solution for decision making.
>
> Behavior trees have a lot in common with Hierarchical State Machines but, instead of a state, the main building block of a behavior tree is a task. A task can be something as simple as looking up the value of a variable in the game state, or executing an animation.
>
> Tasks are composed into sub-trees to represent more complex actions. In turn, these complex actions can again be composed into higher level behaviors. It is this composability that gives behavior trees their power. Because all tasks have a common

interface and are largely self-contained, they can be easily built up into hierarchies (i.e., behavior trees) without having to worry about the details of how each sub-task in the hierarchy is implemented. (Millington, Funge 2016)

Types of Task

Tasks in a behavior tree all have the same basic structure. They are given some CPU time to do their thing, and when they are ready they return with a status code indicating either success or failure (a Boolean value would suffice at this stage). Some developers use a larger set of return values, including an error status, when something unexpected went wrong, or a need more time
status for integration with a scheduling system. three kinds of tasks: Conditions, Actions, and Composites. (Millington, Funge 2016)

Behavior Trees and Reactive Planning

Behavior trees implement a very simple form of planning, sometimes called reactive planning. Selectors allow the character to try things, and fall back to other behaviors if they fail. This isn't a very sophisticated form of planning: the only way characters can think ahead is if you manually add the correct conditions to their behavior tree. Nevertheless, even this rudimentary planning can give a good boost to the believability of your characters.

The behavior tree represents all possible Actions that your character can take. The route from the top level to each leaf represents one course of action,2 and the behavior tree algorithm searches among those courses of action in a left-to-right manner. In other words, it performs a depth-first search.
In the context of a behavior tree, a Decorator is a type of task that has one single child task and modifies its behavior in some way. You could think of it like a Composite task with a single child. Unlike the handful of Composite tasks we'll meet, however, there are many different types of useful Decorators.

One simple and very common category of Decorators makes a decision whether to allow their child behavior to run or not (they are sometimes called "filters"). If they allow the child behavior to run, then whatever status code it returns is used as the result of the filter. If they don't allow the child behavior to run, then they normally return in failure, so a Selector can choose an alternative action.(Millington, Funge 2016)

External Datastore in Behaviour Trees

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. (Millington, Funge 2016)

GOAP

With the ever increasing need for complexity and entertainment in the Game industry a further development in planning, behaviors and goals was that of the creation of Goal Oriented Action Planning (GOAP). Prof. Tommy Thompson has provided an informative introduction to GOAP. The early predecessor to GOAP, according to it's innovator in the Game Industry- Jeff Orkin (MIT), was the STRIPS (Stanford Research Institute Problem Solver) on which GOAP was based. Developed at SRI in 1971 along with A* search, it is recounted by it's developers, Nilsson and Fikes:

> "STRIPs is often cited as providing a seminal framework for attacking the 'classical planning problem' in which the world is regarded as being in a static state and is transformable to another static state only by a single agent [see Nozawas 'Single Warrior Model'] performing any of a given set of actions. The planning problem is then to find a sequence of agent actions that will transform a given initial world state into any of a set of given goal states. For many years, automatic planning research was focused on that simple state-space problem formulation, and was frequently based on the representation framework and reasoning methods developed in the STRIPS system." (Fikes, 2, 1993)

A* Search and STRIPS would work hand-in-hand to find optimal paths. Automated Planning: is a form of AI that is focussed on long-term and abstract decision making. Decisions are madelled at a high level, using methods based on STRIPS.
The STRIPS plan would cause the transitions between states, as the FSM needed to shift state several times in order to execute all actions within a plan.
Related to this project was the development at SRI in the 70s of 'Shackey the Robot' of which A* developed [cf. STRIPS, a retrospective, Artificial Intellignce 59 (1993) 227-32 Fikes and Nilsson] A* developed by Hart also worked on region-finding scene analysis programs (Duda and Hart, R.O. Duda and P.E. Hart, Experiments in scene analysis, Tech. Note 20, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1970). Thus having 'understood' the scene new plans could be created to address goals. In video games this was first delivered in the title 'F.E.A.R.' A first-person shooter (FPS). Thompson gives a short overview of GOAP:

> One of the key issues that we need to consider when developing intelligent and engaging NPCs for games is their ability to build some kind of strategy. Traditionally, a NPC implementation will rely on finite state machines or a similar formalism. These are really useful for a number of practical reasons, notably:

We can model explicitly (and often visually) the sort of behaviours we expect our NPC to exhibit. We can often express explicit rules or conditions for when certain changes happen to the NPC state. We can be confident that any behaviour the NPC exhibits can be 'debugged', since their procedural nature often exhibit the Markov property: meaning we can always infer how a state of the bot has arisen and also what subsequent states it will find itself in. However the problem that emerges here is that behaviours will neither be deliberative, nor emergent. Any bot that uses these approaches is tied to the specific behaviours that the designers had intended. In order for the more interesting NPCs in F.E.A.R. to work, they need to think long-term. Rather, they need to plan.

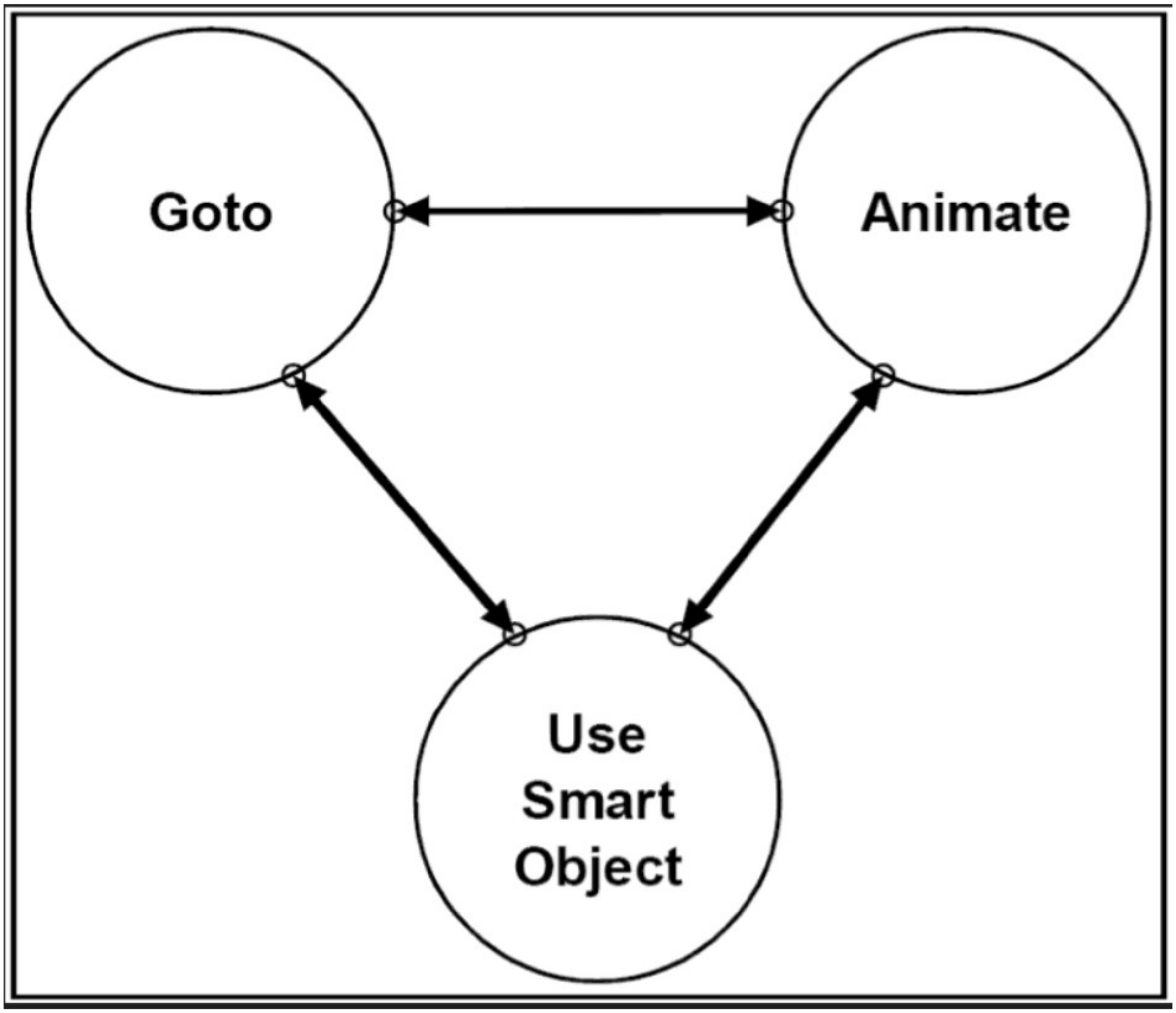
Planning (and scheduling) is a substantial area of Artificial Intelligence research, with research dating back as far as the 1960's. As my game AI students will soon learn in more detail, it is an abstract approach to problem solving; reliant upon symbolic representations of state conditions, actions and their influence. Often planning problems (and the systems that solve them) distance themselves from the logistics of how an action is performed. Instead they focus on what needs done when. For those interested, I would encourage reading (Ghallab et al., 2004) which provides a strong introduction to planning and scheduling systems.

The NPCs in F.E.A.R. are not merely reactive in nature, they plan to resolve immediate threats by utilising the environment to full effect. The NPCs in F.E.A.R. are not merely reactive in nature, they plan to resolve immediate threats by utilising the environment to full effect. One benefit of a planning system is that we can build a number of actions that show how to achieve certain effects in the world state. These actions can dictate who can make these effects come to pass, as well as what facts are true before they can execute, often known as preconditions. In essence, we decouple the goals of an AI from one specific solution. We can provide a variety of means by which goals can be achieved and allow the planner to search for the best solution.

However, at the time F.E.A.R. was developed, planning was not common in commercial games (Orkin, 2004). Planning systems are often applied in real-world problems that require intelligent sequences of actions across larger periods of time. These problems are often very rich in detail – examples include power station management and control of autonomous vehicles underwater or space – and rely on the optimisations that planning systems make (which I will discuss another time) as well as time. Planning systems are often computationally expensive. While we are looking at time-frames of a couple of seconds when dealing with plans of 20-30 actions in games, this is still dedicating a fair chunk of overall resource to the planner (Thompson, 2009); thus ignoring the issues that a game running in-engine at 30-60 frames per second may face. Put simply, you must find a way to optimise this approach should you wish to implement planning systems, lest you suffer a dip in the games performance at runtime.

G.O.A.P.: Goal Oriented Action Planning

The approach taken by Monolith, as detailed in (Orkin, 2005), was known as Goal Oriented Action Planning. The implementation attempted to reduce the potential number of unique states that the planning system would need to manage by generalising the state space using a Finite State Machine.



The three states of the GOAP system in FEAR.

The three states of the GOAP system in FEAR.

As shown in the figure above, the behaviour of the agent is distilled into three core states within a the FSM:

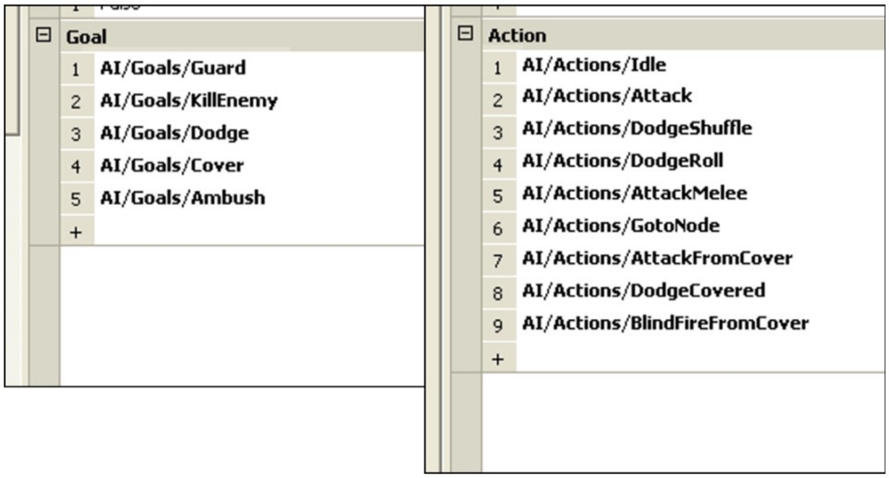
Goto – It is assumed that the bot is moving towards a particular physical location. These locations are often nodes that have been denoted to permit some actions to take place when near them.

Animate – each character in the game has some basic animations that need to be executed in order for it to maintain some level of emergence with the player. So this state enforces that bots will run animations that have context within the game world.

These can be peeking out from cover, opening fire or throwing a grenade.
Use Smart Object – as conceded in (Orkin, 2005), this is essentially an animation node. The only different is the animation is happening in the context of that node. Examples of this can be jumping over a railing or flipping a table on its side to provide cover.
That's it! The entire FSM for the NPCs in the game is three states. Note that these states are very abstract; we do not know what locations are being visited, the animations played and the smart objects used. This is because a search is being conducted that determines how the FSM is navigated.

Which state are these NPCs currently in now?
Bear in mind we traditionally utilise events in order to move from one state to another in a FSM. These events are typically the result of a sensor determining whether some boolean flag is now true, or an 'oracle' system forcing the bot to change the state. This then results in a change of behaviour. However in F.E.A.R., each NPC uses sensor data to determine a relevant goal and then conduct a plan that will navigate the FSM (using grounded values) that will achieve that goal. (Thompson, 2018)

Versatility:
- each agent could be given different goals and actions that would become part of its behaviour.
- Developers could easily tweak the action sets of different characters.



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

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

A collection of goals and actions assigned to a solider. Images taken from the GDC 06 presentation by Jeff Orkin.

(Thompson, 2018)

Goals/Actions
AI/Goals/Dodge. AI/Actions/DodgeRolls AI/Actions/DodgeRun

These gaol actions can be customized to each character in the game. Character: Surveillance Cyborg: AI/Actions/Idle -> talk/text on phone and lounge around on couch, which during action sequence involving player could add AI/Actions/GetInWay in the AI/Goals/StopHim

NPCs are responding to the player and their actions, and they are coordinated, they work together. A peculiar attribute of communicating AI Agents is that they tend to develop their own unique language to speak to each other in Genetic Algorithm contexts. A hive mind of AI Agents has it's own emergent properties which could be foriegn to human understanding or reasoning, it could even appear contradictory and irrational to a human intelligence. In video game and simulations of combat, squads are managed in squads, courtesy of a squad manager system, squad enrollment is based largely on proximity, provides useful information they might need. Squad manager tells them what goals to achieve, but it doesn't override their base instincts or drivers.

Goal Selection is based on threat level. The objective of a Goal Selection is to minimize the threat. Then invoke a Plan to meet that Goal. The focus shifts from building actions that can complete goals when put together, to creating behaviours of multiple actions.

Millington and Funge (2016) give some important components of GOAP as:

Goal-Oriented Behavior

> So far we have focused on approaches that react: a set of inputs is provided to the character, and a behavior selects an appropriate action. There is no implementation of desires or goals. The character merely reacts to input.
> It is possible, of course, to make the character seem like it has goals or desires, even with the simplest decision making techniques. A character whose desire is to kill an enemy will hunt one down, will react to the appearance of an enemy by attacking, and will search for an enemy when there is a lack of one.
> to present the character with a suite of possible actions and have
> it choose the one that best meets its immediate needs. This is goal-oriented behavior (GOB), explicitly seeking to fulfill the character's internal goals. Like many algorithms in this book, the name can only be loosely applied. GOB may mean different
> things to different people, and it is often used either vaguely to refer to any goal seeking decision maker or to specific algorithms similar to those here
> Goal-oriented behavior is a blanket term that covers any technique taking into account goals or desires. There isn't a single technique for GOB, and some of the other techniques in this chapter, notably rule-based systems, can be used to create goal-seeking characters. Goal-oriented behavior is still fairly rare in games, so it is also difficult to say what the most popular techniques are. (Millington, Funge 2016)

Goals

A character may have one or more goals, also called motives. There may be hundreds of possible goals, and the character can have any number of them currently active. They might have goals such as eat, regenerate health, or kill enemy. Each goal has a level of importance (often called insistence among GOB aficionados) represented by a number. A goal with a high insistence will tend to influence the character's behavior more strongly.

For the purpose of making great game characters, goals and motives can usually be treated as the same thing or at least blurred together somewhat. In some AI research they are quite distinct, but their definitions vary from researcher to researcher: motives might give rise to goals based on a character's beliefs, for example (i.e., we may have a goal of killing our enemy motivated by revenge for our colleagues, out of the belief that our enemy killed them). This is an extra layer we don't need for this algorithm, so we'll treat motives and goals as largely the same thing and normally refer to them as goals. (Millington, Funge 2016)

Actions

In addition to a set of goals, we need a suite of possible actions to choose from. These actions can be generated centrally, but it is also common for them to be generated by objects in the world. We can use a range of decision making tools to select an action and give intelligent-looking behavior. A simple approach would be to choose the most pressing goal (the one with the largest insistence) and find an action that either fulfills it completely or provides it with the largest decrease in insistence. In the example above, this would be the "get raw-food" action (which in turn might lead to cooking and eating the food). The change in goal insistence that is promised by an action is a heuristic estimate of its utility—the use that it might be to a character. The character naturally wants to choose the action with the highest utility, and the change in goal is used to do so.

We can do this by introducing a new value: the discontentment of the character. It is calculated based on all the goal insistence values, where high insistence leaves the character more discontent. The aim of the character is to reduce its overall discontentment level. It isn't focusing on a single goal any more, but on the whole set. Discontentment is simply a score we are trying to minimize; we could call it anything. In search literature (where GOB and GOAP are found in academic AI), it is known as an energy metric. This is because search theory is related to the behavior of physical processes (particularly, the formation of crystals and the solidification of metals), and the score driving them is equivalent to the energy. (Millington, Funge 2016)

Timing

In order to make an informed decision as to which action to take, the character needs to know how long the action will take to carry out.

To allow the character to properly anticipate the effects and take advantage of sequences of actions, a level of planning must be introduced. Goal-oriented action planning extends the basic decision making process. It allows characters to plan detailed sequences of actions that provide the overall optimum fulfillment of their goals. This is basically the structure for GOAP: we consider multiple actions in sequence and try to find the sequence that best meets the character's goals in the long term. In this case, we are using the discontentment value to indicate whether the goals are being met. This is a flexible approach and leads to a simple but fairly inefficient algorithm. In the next section we'll also look at a GOAP algorithm that tries to plan actions to meet a single goal.

To support GOAP, we need to be able to work out the future state of the world and use that to generate the action possibilities that will be present.When we predict the outcome of an action, it needs to predict all the effects, not just the change in a character's goals. To accomplish this,we use a model of the world: a representation of the state of the world that can be easily changed and manipulated without changing the actual game state. For our purposes this can be an accurate model of the game world. It is also possible to model the beliefs and knowledge of a character by deliberately limiting what is allowed in its model. A character that doesn't know about a troll under the bridge shouldn't have it in its model.Without modeling the
belief, the character's GOAP algorithm would find the existence of the troll and take account of it in its planning. That may look odd, but normally isn't noticeable (Millington, Funge 2016)

Dialogue and Chatbots in AI Agents

Along with Video or Images in AI Games and Simulations, Audio also plays an important part in the game world. Through interactions with AI Agents gamers can learn narratives of the story, they can learn where to go next, they be bullied by oppositional AI Agents or NPCs. AI Assistants can leverage this Audio into hidden ways to win the game, depending on how generative an AI Assistant is.

Dialogue Systems in AI are of varied designs. For instance in a combat simulation you will have a Battle Chatter system- enemy enemy AI will shout commands to one another and provide exposition of their intended behaviour. Not all AI Agents are the same, one factor of differentiation in NPCs is Knowledge of Player- regular archetypes need to find the player, while specialists always know where you are.

Which is reminiscent in Drone Swarm programming that a lead AI Agent (drone) tells and coordinates the other drones in the swarm, a hierarchical C2 within drone swarms.

To fully understand how Dialogue works in games, the following overview of Dialogue Systems is presented. A way for an NPC to show intelligence in a game is to engage in a

dialogue with a player, this can of course also be used as strategy in the game. The vast majority of dialogue in games is scripted often leading to clunky implementations where NPCs repeat themselves unnecessarily or they will ignore the player entirely, especially when they have input that matches no handler for dialogue responses. Dialogue trees are a ocmmon technique to simute dialogue in the game world. Branching dialogue trees allow a gamer to choose from pre-selected questions or options to speak to the NPC. Some dialogue trees are not trees but can be graphs that have cycles (Rose 2018).
Understanding inputs involves the use of Natural Language Processing libraries. Rose talks about the limitations of using NPL:

> The use of simple natural language processing has the following major disadvantages:
>
> NPC responses are usually not automated
>
> o Most of the time, NPC responses are prewritten by a human author; it is very rare for them to be automated. Thus, lots of time and effort must go into writing them because “the only way to increase interactivity is to author extraordinary amounts of content by brute force”. NPC may ignore what player says
>
> o If no rule exists on how to respond to the player’s input, an NPC will either ignore the player entirely or suddenly change the topic of conversation. Either scenario breaks the illusion that the NPC is intelligent because it becomes obvious that it does not know how to respond.
>
> With sentiment analysis, the system could recognize that the player has a negative sentiment towards the sword’s sharpness. In response, it could perhaps recommend either a place to get it sharpened or a shop to purchase a new sword. Furthermore, players may be either friendly or hostile towards an NPC. Sentiment analysis would allow the system to recognize the player’s positive or negative sentiment about the NPC and have it respond in a sensible manner. For example, if a player insults the NPC, then the NPC could respond aggressively, while if the player compliments it, it could respond kindly. By adjusting the NPC’s reaction based on how the player is perceived to be feeling, the dialogue will seem more natural and the NPC will seem more autonomous. (Rose, 2018, 24-5)

With these limitations in mind recent game developers have relied on other means of developing Game dialogue systems. Such as Information Extraction, which extracts structured information from unstructured sources. Sentiment Analysis, analyzes the sentiment of the gamer and makes recommendations or misdirections based on it’s perceptions of the gamers dialogue. Question Answering, which posts information based on a natural language question.

> Combined with information extraction and sentiment analysis, question answering could potentially offer very sophisticated replies. While none of these techniques have been perfected yet, their implementation in a dialogue engine could greatly add to the believability of an NPC compared to traditional methods. (Rose 2018, 25-6)

A key factor that facilitates this sort of Turing Test in games, is that the NPC has Episodic Memory, which is responsible for recalling episodic memories based on previous topics in conversation. Keywords are used which trigger the activation of certain memories. When the topic handler decides a match exists in memory the most recently activated memory is retrieved (Rose, 2018, 47). Episodic memory can be created ahead of time in a knowledge base or assigned dynamically during the game, each memory receives a weight based on 'importance'. Two pools of memories are created based on these weights: short term and long term memory. With low weight short term memories being deleted like a forgetful or immerable experience.

An effective Dialogue system can go a long way to making the NPCs in a game or simulation believable. It can also make the game interactions even richer. One developer, Alec John Larsen, has proposed a solution that allows authors to create different NPCs using a single chatbot, by using the open source project Artificial Linguistic Internet Computer Entity (A.L.I.C.E.).

To facilitate the dialogue behaviors are facilitated by A Behavior Language (ABL). Also, there is a drama manager, to facilitate beats in dialogue, beats are selected using preconditions (causal dependence), weights, priorities and the current tension levels (Larsen, 16). A rule-based system for translating player actions and textual input into discourse acts. This can provide a great experience in Interactive Drama in the game as Larsen explains:

> ...interactive drama, interactive storytelling and interactive experiences, to name a few. There are also many definitions of what interactive diction is. It is stated that interactive dramas are systems in which the audience experience the story as interactive participants. The users may be first person protagonist but this is not an explicit requirement of an interactive diction system. Some works of interactive diction make use of artificial intelligence (AI) techniques, to manage the way the story is generated, and some do not (e.g. hypertext diction). The focus of interactive storytelling is audience experience, this includes building worlds with character and story, where the virtual worlds are dramatically interesting and contain computer-controlled characters, with whom the user can interact. From the design focus when creating interactive experiences is agency, immersion (a player feels like an integral part of the story) and transformation (a player can experience different aspects of the story), where agency is the most important. Agency is defined in as the satisfying ability of the player to make meaningful decisions, which have tangible effects on the play experience. It is not that the players have an unlimited number of actions at their disposal but rather there are context-based actions that can be performed, which have a very real impact on the experience . The reason agency is the main focus when designing interactive diction is because it allows the player to have the most substantial influence on the experience and requires the system to dynamically generate the story based on the player's unpredictable actions.
> (Larsen, 14-5)

He goes on giving an account of how the dialogue is programmed:

> Dialogue graphs are made of stimulus response elements (SREs), represented by AIML patterns (which map an input to an output). A GUI is used to create the dialogue graphs and the scenes out of SREs. Scenejo requires that the authors take the NPCs' moods into account and as such cannot dynamically assign mood. Scenejo uses AIML extremely verbosely and requires a knowledge-base per character per seen, resulting in an inefficient writing process. SentiStrength is a textual affect sensing framework, which makes use of keyword spotting and statistical methods...to assess the sentiment of informal text. SentiStrength gives text a rating of positive (1 to 5) and negative (1 to 5) simultaneously, which differentiates it from other opinion mining systems which do not give both positive and negative ratings simultaneously and do not give strengths of the sentiment but rather an indication of whether it is positive or negative. SentiStength was created using MySpace user comments (which provided a large quantity of informal text) as training data sets and evaluation data set. (Larsen, 18)

An important point here is that dialogue is dependent on mood. Which is computationally complex, see next section. How does one rank the emotive qualities of the player? This is handled by the field of Textual Affect Sensing, commonly used libraries are ConceptNet and SentiStrength. ConceptNet uses the model of Paul Ekman, where every possible emotion can be given a weighted sum of six emotions: happiness, sadness, anger, fear, disgust and surprise. And of course each would also have other behaviors or markers in the game interface. The weights are then calculated into a decision making structure to produce desirable and hopefully believable behaviors. The mulitplicative weights are modeled using a Feelings Modelling Language (FML).

So one can see how dialogue systems can be crafted in games or serious games or simulations. Of course one could take this a step further and input a deep psychological profile of the gamer into the system to get even more specific responses to the player. Making the illusion that the game is real, enticing the player to get lost in the ‘flow’, is tantamount to an emotionally invested and adrenaline rushed gamer, to get them into the game, to lose sight of reality and be entertained. Creating believable characters with effective dialogue is just one aspect of the open world game that is popular in current game development. Open world games require a large diverse and unique set of background NPCs to create the illusion of a populated virtual world that mirrors the diversity of the real world. One can imagine the difficulties is trying to hard code all these unique NPCs, which means that many NPCs are repeated characters and patterns, which makes an open world game seem repetitive, predictable and not believable (Fallon, 2-3). The highest priority should be given to Personality in a Game. Researcher in making games believable extrapolates on the Personality problem in games:

> Emotion: Just as in the traditional character-based arts, as well as acting, where emotion is regarded as a fundamental requirement to bring the characters to life, so too is it in making believable agents. In order for an agent to be believable, it must appear to have emotions and be capable of expressing these emotions in a manner that corresponds

> with its personality.
>
> Self Motivation: In order to create believable agents, a requirement is that they should be self motivated. In other words agents should be able to carry out actions of their own accord and not just react to stimuli, as is often the case in the creation of autonomous agents. The power of this requirement is quite substantial and can lead to increased admiration and interest in the agent when it appears to complete tasks according to its own intentions.
>
> Change: Essential to the believability of agents is how they grow and change. However, this growth and change cannot be arbitrary, rather it should correspond with the agents personality.
>
> Social Relationships: Since humans lead a generally social lifestyle, an important requirement for believable agents is social relationships. There should be interaction amongst agents, influenced by the relationships that exist between them, and in turn those interactions may influence the relationships.
>
> Consistency of Expression: A basic requirement for believability is the consistency of expression. An agent can have a multitude of behavioural and emotional expressions such as facial expression, movement, body posture, voice intonation etc. In order to be believable at all times, these expressions must work in unison to communicate the relevant message for the personality,
> feelings, thinking, situation etc. of the character. If this consistency is
> broken, the suspension of disbelief is likely to be lost.
>
> The Illusion of Life: Some requirements necessary for believable agents are overlooked or taken for granted by the conventional media. For example, it is not necessary to specify that an actor should be able to speak while walking since any human actor reading the script can do this easily. This is simply not the case when creating believable agents where those properties must be built into the agent. A list of such properties, which alltogether form this final requirement for believability, includes: appearance of goals, concurrent pursuit of goals and parallel action, reactive and responsive, situated, resource bounded, existence in a social context, broadly capable and well integrated (capabilities and behaviours).
> (Fallon 2013)

The desire to have believable AI Assistants has driven a lot of research into the area of believability, which is to say Turing Test passable characters, even with some NPCs seeming more human then humans we meet in real life. With these AI Agents or NPCs the internal model of the agent is built on a system known as the Beliefs-Desires-Intentions (BDI) architecture (Fallon 2013) . Other models are also incorporated into the model of the agent, the Consumat model uses dominant psychological theories on human behavior, being put into a 2x2 matrix [reminiscent of Krylov Space in Norseen’s application]. One hand you have the level of need of satisfaction (LNS) and behavioral control (BC), while on the other hand based on certainty, type of needs and cultural perspective (Fallon 2013 12). Another model which is a meta-model developed by Carley and Newell is the Fractionation Matrix (C&N Matrix) on social behaviour to provide understanding of the complexity of an AI Agent. This model is based on an assortment of sociological theories that can be used to classify, categorize and silo human like behavior in agents, the goal of which is to create a “Model social agent” (MSA) which has strong, human-like social behavior(Fallon 2013, 13-4). A derivative from MSA is the Model Social Game Agent of Johansson and Verhagen which takes that there exists an emotional state and a social state that contribute to a behavior selection, which subsequently leads to an action from the agent ((Fallon 2013, 15). One last entry in this field is that of SIGVerse as explained by Fallon:

> More comprehensive frameworks have been recently proposed with a focus on serious games. Inamura et al propose a simulator environment known as SIGVerse which includes a combination of dynamics, perception and communication simulations, with a background in robotics simulation and therefore with a strong model of embodiment and situatedness. Embodiment involves equipping a body with sensors and actuators which enables structural coupling with the agents surroundings. A video game agent can therefore be said to be embodied in that sense (or ‘virtually embodied') since it procures exteroceptive and proprioceptive sensing data through its software sensors and actions can also be performed using its software actuators. Situatedness is concerned with interaction with the world. Take for example a video game character, if it interacts with the simulated world, the game environment is affected, which in turn affects the agent. The SIGVerse project does not however incorporate social interaction very well. (Fallon 2013, 18)

Emotions in Video Games

As seen in the previous section dialogue is a key component to believable AI Agent NPCs. It was also drawn out that dialogue must have a calculation for emotion to give a proper context to the dialogue. So when we say emotion modeling what is it we are talking about:

> ‘Emotion modeling’ can mean the dynamic generation of emotion via black-box models that map specific stimuli onto associated emotions. It can mean generating facial expressions, gestures, or movements depicting specific emotions in synthetic agents or robots. It can mean modeling the effects of emotions on decision making and behavior selection. It can also mean including information about the user’s emotions in a user model in tutoring and decision-aiding systems, and in games. There is also a lack of clarity regarding what affective factors are modeled. The term ‘emotion’ itself is problematic …it has a specific meaning in the emotion research literature, referring to transient states, lasting for seconds or minutes, typically associated with well-defined triggering cues and characteristic patterns of expressions and behavior. (More so for the simpler, fundamental emotions than for complex emotions with strong cognitive components.) (Hudlicka, 1)

Yet, there is a question as to what is being modeled and the consistency of the models.

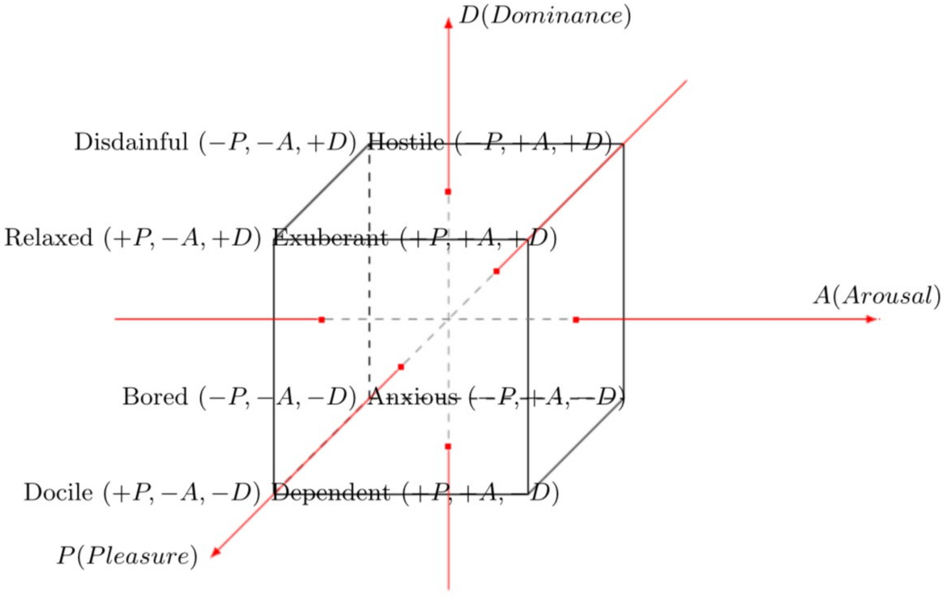
> Recognizing the lack of consistent terminology and design guidelines in emotion modeling, this paper proposes an analytical framework to address this problem. The basic thesis is that emotion phenomena can usefully
> be understood (and modeled) in terms of two fundamental processes: emotion generation and emotion effects, and the associated computational tasks. These tasks involve, for both processes: defining a set of mappings (from triggers to emotions in emotion generation, and from emotions to their effects in the case of emotion effects), defining intensity and magnitude calculation functions to compute the emotion intensities during generation, and the magnitude of the effects, and functions that combine and integrate multiple emotions: both in the triggering stage (antecedents), and in the emotion effects stage (consequences). This analysis represents a step toward formalizing emotion modeling, and providing foundations for the development of more systematic design guidelines, and alternatives available for model development. Identifying the specific computational tasks necessary
> to implement emotions also helps address critical questions regarding the nature of emotions, and the specific benefits that emotions may provide in synthetic agents and robots. (Hudlicka, 8)

Sergey Tarashenko, has sought out an emotional model based on Reflexive Control In a previous section, ‘Lessons from an American Weapons Designer’ we showed how that Reflexive Control was combined with Game Theory. One researcher, currently with Mizuho Securities, has done extensive work on Reflexive Game Theory and Emotions, Tarashenko (2016). Sergey Tarashenkos work also deals with using robots interacting with humans in Reflexive Games He writes regarding the incorporation of emotion into games in a paper which he also cites Ekman, from above:

> The semantic differential approach originally proposed by Osgood et al [1957]. considers three dimensions to characterize the person’s personality. These dimensions are Evaluation, Activity and Potency.This approach was further tested from the point of view of emotions. Russell and Mehrabian proved in their study that the entire spectra of emotions can be described by the 3-dimensional space spanned by Pleasure (Evaluation), Arousal (Activity) and Domination (Potency) axes. For every dimension, the lower and upper bounds (ends) are recognized as negative and positive poles [-1,0,1], respectively. Consequently, the negative pole can be described by negative adjectives, and positive one by positive adjectives. It was shown by Russel and Mehrabian that these three dimensions are not only necessary dimensions for an adequate description of emotions, but they are also sufficient to define all the various emotional states. In other words, the Emotional state can be considered as a function of Pleasure, Arousal and Dominance. On the basis of this study, Mehrabian proposed Pleasure-Arousal-Dominance (PAD) model of Emotional Scales. The emotional states defined as combinations of ends from various dimensions are presented. In this study, we discuss

the matter of how the PAD model can be used in Reflexive Game Theory (RGT) to emotionally color the interactions between people and humans and robots.

According to Mehrabian, “pleasure vs displeasure” distinguishes the positive vs negative emotional states, “arousal vs non-arousal” refers to the combination of physical activity and mental alertness, and “dominance vs submissiveness” is defined in terms of control vs lack of control.
(Tarashenko 2016)



D(Dominance)
Disdainful (−P, −A, +D) Hostile (−P, +A, +D)
Relaxed (+P, −A, +D) Exuberant (+P, +A, +D)
A(Arousal)
Bored (−P, −A, −D) Anxious (−P, +A, −D)
Docile (+P, −A, −D) Dependent (+P, +A, −D)
P(Pleasure)

Fig. 1. The Pleasure-Arousal-Dominance (PAD) model’s space.

Rather then having a 2X2 Matrix as the above example illustrates we have a 3x3 matrix [not unlike the Enneagram and it’s 3 divisions and 3 values per division, again see Krylov space from previous]. Tarashenko sees this as a way of influencing people, it might also be used to create great NPCs. By applying his math to a computer game it could be possible to create more believable characters. Of course all of this is very similar to the ideals presented by Norseen in ‘Thought Injection’. Tarashenko points out how different variances in the matrix can lead to different emotional states:

Summarizing the facts about the RGT and PAD model, we highlight that **RGT has been proven to predict human choices in the groups of people and allows to control human behavior by means of particular influences on the target individuals.** Next we note that PAD model provides a description of how the emotional states of humans can be modelled, meaning that a certain emotional state of a particular person can be changed to the desired one. Furthermore, it is straight-forward to see that the coding of the PAD emotional states and alternatives of Boolean algebra are identical.

Therefore, it is possible to change the emotional states of the subjects in the groups by making influences as elements of the Boolean algebra. In such a case, vector {1,0,0},

> for example, plays a role of influence towards emotional state Docile. Besides, we have distinguished three basis emotional states Docile ({1,0,0}), Anxious ({0,1,0}) and Disdainful ({0,0,1}). The interactions (as defined by disjunction and conjunction operations) of these basic emotional states can result in derivative emotional states such as Dependent, Relaxed, etc. Before, considering the example of PAD application in RGT, we note that reflexive function Φ defines state, which subject is going to switch to. **This process goes unconsciously.** We have discussed above the reasons the conjunction and disjunction represent alliance and conflict relationships, respectively (Tarashenko, 2016) [emphasis, bold, added]

In a video game this could also be incorporated into dynamic AI vs Human, AI vs AI, interactions. In his paper he gives the example of a Director of Enterprise and how to measure his assistants. One can see how in squad based combat simulations this could come in handy in single player mode, as the gamer is in a squad with only AI NPCs. The interactions could become very interesting rather then scripted. Indeed, he does see AIs taking on human emotions and believes this will make computation faster. Tarashenko writes:

> Furthermore, we have not only illustrated how to apply RGT to the control of subject's emotions, but uncovered the entire cascade of human reflexion as a sequence of subconscious reflexion, which allows to trace emotional reflection of each reflexive image. This provides us with unique ability to unfold the sophisticated structure of reflexive decision making process, involving emotions. Up todate, there has been no approach, capable of doing such a thing, reported.The emotional research based on PAD model is transparent and clear. The models of robots exhibiting human like emotional behavior using only PAD has been successfully illustrated in recent book by Nishida et al. The ability to influence the factors (variables) is the major justification for such approaches and for the application of the RGT. Yet, it is just a "mechanical" part of the highly sophisticated matter of human emotions.
>
> The present study introduces the brand new approach to modeling of human emotional behavior. We call this new breed of RGT application to be emotional Reflexive Games (eRG). At present RGT fused with PAD model is the unique approach allowing to explore the entire diversity of human emotional reflexion and model reflexive interaction taming emotions. This fusion is automatically formalized in algorithms and can be easily applied for further developing emotional robots. Since the proposed mechanism has no heavy negative impact on human psychological state, robots should be enabled to deal with such approach in order to provide human subjects with stress free psychological friendly environment for decision making. (Tarashenko, 2016)

Another aspect of emotions in games is that of the beat of the game. The beat or pacing of the game is a key element of emotional engagement as studied by Thompson. In Left 4 Dead (2009) the use of an AI Director System was introduced (Thompson, 2014), the director system builds pace in the game, it does not just constantly through a steady stream of npc at the player rather it uses 3 phases: build-up (anticipation/expectations), peak, relax, with the amplification of each phase variable. The Director system uses player stress levels based on player performance, worse performance = greater stress. The stress level rate of increase tells the system how skilled the player is. The Directors present something novel: an opportunity to create unique experiences every time you play the game.

Another emotional manipulating trick in games is that of creating a personal relationship with the player by NPCs, as shown with the Nemesis System in Shadow of Mordor (2014):

“Middle-earth: Shadow of Mordor has a lot of ambition in what it’s trying to do. The action-adventure title stands out for a couple new features that help its open world feel alive.
One of those things is the unique Nemesis system, where the game populates a hierarchical system of hostile Uruks who all have unique names, features, and traits. Better yet, these AI-controlled enemies remember specific things about you and the world, such as whether or not you beat them in combat, wound them, run away, or do something else entirely. What’s fantastic here is that not only does the system make each player’s experience unique, it gives the title’s setting a personality that other open-world games struggle to create.

These efforts to humanize your enemies, and then use those enemies to more or less troll the player in a way we enjoy, is brilliant. A few simple systems work together to create something that feels larger than life and complex, and it helps to make Shadow of Mordor one of the most interesting games of the season.”
https://www.polygon.com/2014/10/13/6970533/shadow-of-mordors-nemesis-system-is-simpler-than-you-think-and-should

Case Study: Diplomacy AI in Total War

When combining the different elements of AI in Games we can create robust simulations and real games. As noted previously, in the Cyber Semiotics section of an ‘Lessons from an American Weapons Designer’ engineers are seeking out ways to model not just combat but also geopolitical international relations. In the game world, there is one game that is known for its crafting of a Diplomacy Artificial Intelligence. In the following case study we will look at the Diplomacy AI of ‘Total War’.

Diplomacy and AI go further back then game development. An active area of research is optimizing diplomacy bots in the serious game ‘Diplomacy’:

> The main community on the internet that focuses on writing AI bots that play Diplomacy is the Diplomacy AI development Environment (DAIDE).... Also they have developed a language for communication between the bots. (De Jonge, 6) [http://www.daide.org.uk]

Screen Shot from Total War (Thompson 2018b)

It is simpler to create a Diplomacy AI or bot in a game then the real world. The following section is based on the case study of Tommy Thompson (Thompson 2018b). Total War came out in 2000 with the release of Shogun Total War. It had three main areas of AI: Diplomacy, which handles troop and resources, establishes unit troop formations, and it's state management uses Genetic Algorithms; Combat which was campaign management, strategy for attack and defense; Unit Manager that collects unit types, manages combat, uses an Artificial Neural Network for managing individual units (i.e. Calvary). The AI had integrated into it a knowledge base from Sun Tsu's Art of War. For an example rule base or maxims:

-when you surround an army, leave an outlet free
-on open ground, do not try to block the enemy's way
-if you outnumber the enemy 10:1 surround them
-if you outnumber the enemy 5:1, attack them directly
-if you outnumber the enemy 2:1 divide them

The use of Genetic Algorithms created tactics for each State of Feudal Japan in war with each other. With some states being more aggressive and others more diplomatic. This created variance in the different feudal states and complex negotiations in Diplomacy.

Total War had subsequent releases: Medieval Total War 2002, Foundations of Shogun Total War, Total War Rome 2004. Also in 2004 the franchise opened the ability for fans to modify the game, by using an API to write their own scripts for game play. With the release of Total War Empire 2009 an expanded game was presented that added to warfare the need to tax or raise finances to fight and naval battles. In 2009 the core AI was redeveloped and it adopted GOAP for state management. As one game designer that worked on the AI notes:

> This AI is not like any other we have written... by far the most complex code edifice i've ever seen in a game. I wrote much of the campaign AI for Shogun and Medieval... And I know that even quite simple 'static' evaluate-act AI's with no plans or memory can be complex enough to exhibit chaotic behavior (we're talking about mathematical 'butterfly effect' style chaos here). It does what it does, and it's not quite what you intended. This can be a good thing- you cull out the bad behaviors and are left with just what is good, and with a simple system that's not too predictable.
> ...The empire Ai is way more complicated than any of our previous product... The net result is **an AI that plans furiously and brilliantly and long term, but disagrees with itself chronically and often ends up paralyzed by indecision**"
> (Mike Simpson, Blog the Second, Total War Blog, Oct 9, 2009) [emphasis added]

As we can see from an algorithmic complexity perspective even a self-contained universe of a game managed by AI can lead to disagreement and paralyzed by indecision, not to mention bad decisions.

The Campaign AI allowed for interactions with players, in negotiations they asked 3 key questions:

-how well am I doing right now?
-what can I do next?
-what resources can I allocate to that?

It also used the previously noted Belief, Desire, Intention (BDI) system. Which was used for evaluating offers, etc. It was divided into sub-AIs:

-Financial System
-Construction System
-Diplomacy System
-Task Management System
-Technology Management
-Character Management

With the release of Total War Rome2 a new decision manager was introduced which used Monte Carlo Tree Search techniques (MCTS). Along with introducing diplomacy ‘personalities’, AI Agents negotiating with other AI Agents and human players. These personalities were collections of probabilities that influence decisions for budgeting, diplomacy evaluations, negotiations between factions and technology research.

As Tommy Thompson notes regarding some of their tactics of these personalities can lead to extreme and binary results, where **burn it all down strategies dominate toward the end of the game, when the Campaign AI is in an unwinnable position.** A mathematical analysis of why it develops a ‘scorched earth’ approach, also exhibited by Nazis in World War 2, would be an insightful endeavor.

RTS Section

The role of Real Time Strategy games in military simulations is little acknowledged or known outside of military research initiatives. However, RTS is used extensively for simulations in for instance the United States Air Force (Gruber 2015). Real-Time Strategy Games are defined as:

> ...Real-Time Strategy (RTS) games, which are essentially simplified military simulations. In an RTS game, a player indirectly controls many units and structures by issuing orders from an overhead perspective (figure 1) in “real-time” in order to gather resources, build an infrastructure and an army, and destroy the opposing player’s forces. The real-time aspect comes from the fact that players do not take turns, but instead may perform as many actions as they are physically able to make, while the game simulation runs at a constant frame rate (24 frames per second in StarCraft) to approximate a continuous flow of time. Some notable RTS games include Dune II, Total Annihilation, and the Warcraft, Command & Conquer, Age of Empires, and StarCraft series’. Generally, each match in an RTS game involves two players starting with a few units and/or structures in different locations on a two-dimensional terrain (map). Nearby resources can be gathered in order to produce additional units and structures and purchase upgrades, thus gaining access to more advanced in-game technology (units, structures, and upgrades). Additional resources and strategically important points are spread around the map, forcing players to spread out their units and buildings in order to attack or defend these positions. Visibility is usually limited to a small area around player-owned units, limiting information and forcing players to conduct reconnaissance in order to respond effectively to their opponents. In most RTS games, a match ends when one player (or team) destroys all buildings belonging to the opponent player (or team) (Robertson, Watson, 2014, 1)

There are several areas of major activity in an RTS game for instance creating forts or battle installations, resource management, and an area that is highlighted in terms of AI Agents playing RTS games is that of micromanagement or “micro” controlling the actions of individual units or groups of units in combat. This is an area where an AI bot can outperform a human. Such that even a human with a superior strategy can be out done by AI agents, since such issues as miskeying are not present to the extent they are with a human player. This is fairly usual, in basic things, an AI can well outperform a human, but in large scale structures and open world parameters the AI struggles to outdue a human gamer.

There has been a major tradition in RTS game world of conducting competitions to create the best AI bots to play each other. A recent advantage has been made by DeepMind in their RTS bot that has beaten human competitors, while many critique this as a phenomenon of micromanagement, it cannot be ignored that DeepMind has made great strides in it’s ability to realistically play like a human and beat humans. They have accomplished this through the use of Deep Learning. While other strategies have been based in more traditional Game AI techniques. The difficulty in RTS is that these computational problems are considered NP Hard, that is computationally long and most times unsolvable on classical architecture, although this supposition is open to question with DeepMind's success. Often, what we think on paper according to math is not correct, such as this supposition.

Capt. Gruber in creating his AI bot uses the Multi-Objective Evolutionary Algorithm (MOEA) method, which a genetic algorithm approach. Genetic programming techniques are a fairly common answer to AI bot functioning (Gruber 2015, 4). The strategy planning of an RTS game is one of the more important elements in the simulation.

> The strategic planning of an RTS game covers the same objectives as a standard military confrontation. In most RTS games the objective is complete destruction of the enemy, with some versions describing victory as the elimination of any manufacturing ability or the conquest of a specific portion of the game map. These goals are accomplished through the development and application of a series of construction actions, otherwise known as build order. The build-order is responsible for moving a player through various technological development stages and can improve a player's unit construction ability. For example, a technical level of 1 may allow a player to build basic infantry, level 2 may introduce more advanced unit types such as flamethrowers or heavy machine guns. Level 3 may build off this further and allow the player to begin building larger assets such as tanks or other vehicles. (Gruber 2015, 22)

The difference between strategy, an overall game approach, and tactics, an more modular and goal oriented sequence of actions is something to consider. Strategy and Tactics decision making have overall architectures presented below:

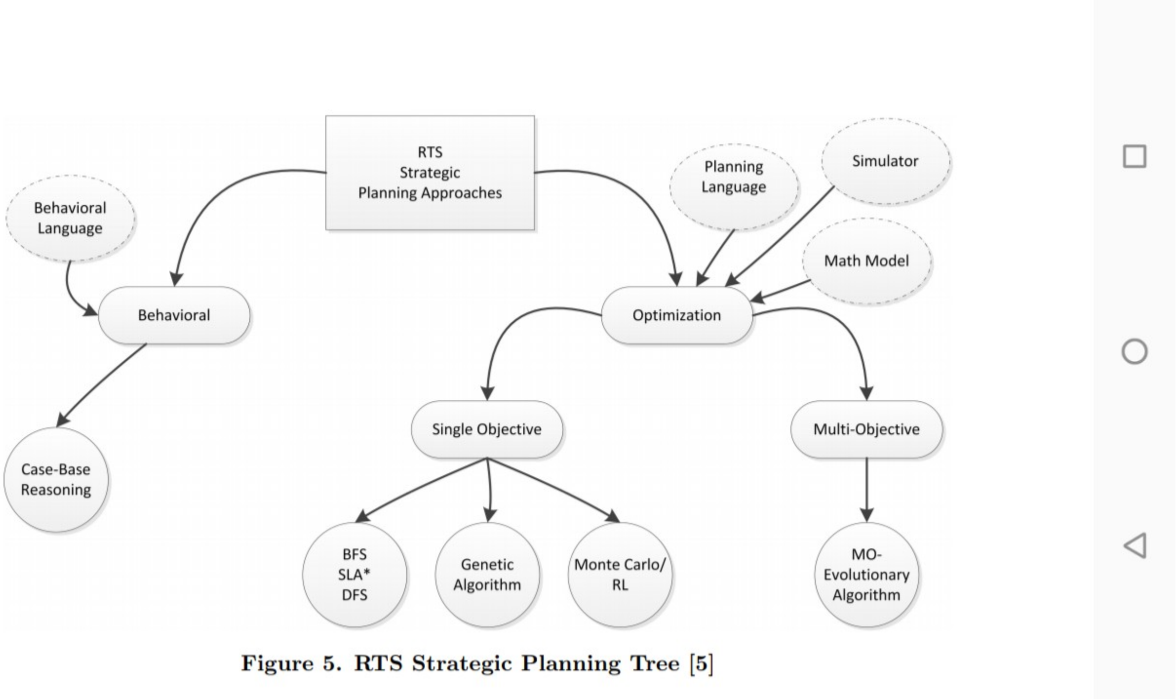


Figure 5. RTS Strategic Planning Tree [5]

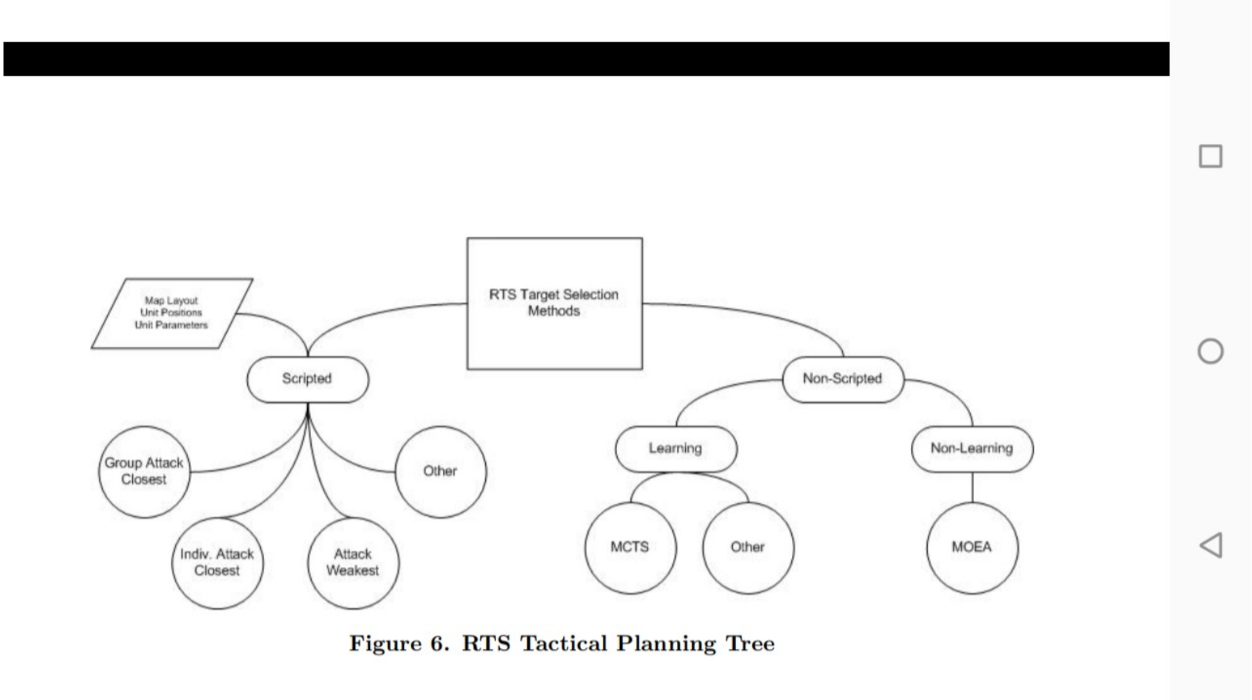


Figure 6. RTS Tactical Planning Tree

(Gruber 2015)

Managing strategy and tactics is one of the key questions in creating an effective bot in RTS games. One approach is that of the creators of Eisbot, it’s creator Weber et al has explained their approach:

> Our approach for managing the complexity of StarCraft is to partition gameplay into domains of competence and develop interfaces between these competencies for resolving goal conflicts. It is based on the integrated. Agent framework proposed by McCoy and Mateas (2008), which partitions the behaviors in a reactive planning agent into managers which specialize in distinct aspects of gameplay. The decomposition of gameplay into managers is based on analysis of expert gameplay. (Weber 2011, 329)

To see the complexity in these games, the managers used to carry out the Robot strategy of Eisbot are:

> -Strategy Manager: responsible for the strategy selection and attack timing competencies
> -Income manager: handles worker units, resource collection and expanding
> -Construction manager: responsible for managing requests to build structures.
> -Tactics manager: performs combat tasks and micromanagement behaviors
> -Recon manager: implements scouting behaviors. (Weber 2011, 331)

Other methods used for managing RTS games with AI Agents have been based on such as previously covered Bayesian Learning, Behavior Trees and Goal-Oriented Action Planning.

Robertson and Watson give other means to build AI bots in RTS games. Neuro-evolution techniques which they say:

> Neuroevolution is a technique that uses an evolutionary algorithm to create or train an artificial neural network. Gabriel, Negru, and Zaharie (2012) use a neuroevolution approach called rtNEAT to evolve both the topology and connection weights of neural networks for individual unit control in StarCraft. In their approach, each unit has its own neural network that receives input from environmental sources (such as nearby units or obstacles) and hand-defined abstractions (such as the number, type, and "quality" of nearby units), and outputs whether to attack, retreat, or move left or right. During a game, the performance of the units is evaluated using a hand-crafted fitness function and poorly performing unit agents are replaced by combinations of the best-performing agents. It is tested in very simple scenarios of 12 versus 12 units in a square arena, where all units on each side are either a hand-to-hand or ranged type unit. In these situations, it learns to beat the built in StarCraft AI and some other bots. However, it remains unclear how well it
> would cope with more units or mixes of different unit types (Gabriel, Negru, and Zaharie 2012). (Robertson, Watson, 2014, 6)

Neuro-evolution being based on Genetic programming techniques it uses a fitness function to determine the best agent composition to deploy in the game. Another approach is that of Case-Based Planning:

> CBP is a planning technique which finds similar past situations from which to draw potential solutions to the current situation. In the case of a CBP system, the solutions found are a set of potential plans or sub-plans which are likely to be effective in the current situation. CBP systems can exhibit poor reactivity at the strategic level and excessive reactivity at the action level, not reacting to high-level changes in situation until a low-level action fails, or discarding an entire plan because a single action failed. (Robertson, Watson, 2014, 6)

CBP is also used in systems such as the Shadow AI in 'Killer Instinct' which mimics the actions of the human gamer and incorporates it into the actions of the AI opponent, similar to learning from demonstration. Of course it is apparent that one stumbling block of this is that new situations will have no effective model, perhaps creating a misfit logjam. Hierarchical Planning is yet another method used for AI bot optimization:

> An alternative means of hierarchical planning was used by Weber et al. (2010). They use an active behavior tree in A Behavior Language, which has parallel, sequential and conditional behaviors and goals in a tree structure (figure 6) very similar to a behavior tree (see section 3.3). However, in this
> model, the tree is expanded during execution by selecting behaviors (randomly, or based on conditions or priority) to satisfy goals, and different behaviors can

> communicate indirectly by reading or writing information on a “shared whiteboard”. Hierarchical planning is often combined as part of other methods, such as how Ontanon et al. (2007) use a hierarchical CBP system to reason about goals and plans at different levels. (Robertson, Watson, 2014, 7)

One way to address the issue of high and low-level reactivity in CBPs is to use a Goal-Driven Autonomy model in which an agent reasons about goals, realizes when they need to be updated, and changes or adds to them as needed for subsequent planning and execution, this actively reasons and reacts to why a goal is succeeding or failing (Robertson, Watson, 8).

In Conclusion

One topic that is of importance to analysing social networks and a phenomenon occurring more and more in society is that of what is known as silofication or silos. This basically is the ideal that people are being fed more and more myopic representations that only reflect themselves. For instance, in Social Network platforms you only see ‘news’ that reflects your personal beliefs and attitudes never having to cross boundaries into other silos to see other information that might not agree with a given silo’s representations. The extent that AI plays in this is great and increasing as AI algorithms are used to generate the content we see on such platforms. The extent that a possible automated Neuroweapons system could play and does play is open to debate. Nonetheless, whether from social network silofication or neurocognitive ‘Thought Injection’ that directly alters people views in such silos the silofication problem is great and increasing.

As we read before such agencies as GCHQ and the NSA are using social networks to study human behaviour and shape such behaviors whether through Thought Injection or typical extrinsic Information Operations, of which we have read GCHQ and the NSA are engaged in the manipulation of social behavior is real and persists as a security problem for nation states and communities. Since they originallay started infiltrating Gaming social networks or as Norseen claims testing psyops through on-line video games such as Norseen’s cited ‘Doom’, it is worth while to see how this has developed. In one genre of Game Massive Multiplayer Online Games human behavior is studied as Osaba and Davis have written:

> The 2008 NRC report highlighted Massive Multiplayer Online Games (MMOG) platforms, surveys and ethnography as predominant sources of behavioral data for training behavioral models. Some recent work has explored MMOGs for understanding observed social behaviours." (Osaba, Davis 2017, 1)

As we learned from the incidents with Cambridge Analytica, psychographics is a real means of profiling and manipulating a person. A psychographic is a profile of your personality, culled from online interactions:

> The growth in behaviorally relevant data streams mirrors a growing reliance on tools and devices for mediating behaviors (e.g. GPS for navigation, music streams for mood management, social media platforms for expression). It is now common to have records or signals of an individual's plans, intentions, and mental states in digital form, especially in affluent cultures with high smartphone adoption. Clark and Chalmers used the term 'The Extended Mind' to refer to the extension of mental deliberations and cognition outside the (as yet) unobservable confines of the human mind [clearly a false assumption, see Thought Injection]. Extended minds with accessible digital data exhausts can be potentially revolutionary for behavioral modeling. The data ecosystem seems to be growing in that direction" (Osaba, Davis 2017, 4)

And:

> Unfortunately, the tendency in analysis is to emphasize signal presence over absence. This tendency has been implicated as a potential cause of excessive polarization in social media interactions. Studies of political discourse on blogs and social media platforms highlight the tendency for hyper-polarized minorities to commandeer discourse and thereby skew platform data streams away from underlying population preferences (Tumasjan et al. 2014)." (Osaba, Davis, 2017, 5)

Some other key points these researchers point out is the limitations of AI/ML method of feed-forward models. Pointing out that it limits such endeavors to sub-tasks of behavioral modeling like NLP. They point out that training statistical models like this can be daunting and unstable. One other limitation of feed forward is that it does not adequately represent chaotic dynamics or complexity (Osaba, Davis, 2017, 10). Social context can also be missing from purely data-driven modeling, lacking explanatory power which is needed to inform decision making. (Osaba, Davis, 2017, 11). The researchers explain the problem in the data to representation problem in models:

> Efforts at building social and behavioral models are subject to similar misspecification risks. Data-driven behavioral models, without the benefit of strong hints from plausible social-science theories of behavior may have poor explanatory power—especially in nonlinear systems. The statistical models may not even be useful for postdiction or post-hoc sense-making. To be sure, fully theory-driven modeling can also be seriously misguided.
>
> In an ideal situation with complete relevant theory and perfect and complete data, the data validates the theories and the theories provide an explanatory/interpretive frame for the data. The current situation, however, is very different: we have a lot of data but it is often imperfect, incomplete, and/or biased, and we have a great many social-science “theories,” which are often narrow, fragmentary, unvalidated, and certainly not settled. (Osaba, Davis, 2017, 12)

So what could be the possible outcome to having AI enlist in the field of Information Operations, Psyops and Neuroweapons. As a veteran, one of the core things that one must adhere to in conflict is the problem of not losing one's moral compass or losing one's humanity, which is difficult enough minus any computer controlling the soldier's mind as envisioned by research scientists in the military industrial complex in the Five Eyes Nations. Computers are not people, they think different, their intelligence is not mediated by being in a real environment with tactile feedbacks, emotional responses, etc. A computer is a calculator, it does math. Such mathematical modelling can never equate to the complexity of humans. So that when one looks at for instance Google's Deep Dream responses of AI generated images, we see weird hallucinogenic representations. Computers talk differently to each other as well, since human language is not a mathematical calculation but computation is. The effects of such utilization of mathematical functions to model non-mathematical reality leads to a problem where computers seem like 'Psychopaths'. As explained by Greene:

> "This means virtual agents trained to maximize their own rewards through the manipulation of humans, using simulated human emotions, have the potential to throw our entire society out of whack. The effects of introducing a near-ubiquitous psychopathic entity (you've got a virtual psychopath in your phone right now) into a society that's only evolved to handle a less-than-one-percent incorporation are, to the best of our knowledge, not widely studied. (Greene, 2019)

Yet, this is precisely what has been proposed since the inception of Neuroweapons and specifically by DoD and Private Contractors working in the Defense space, the direct manipulation of human minds, whether soldiers in the battlespace or terrorists planning operations, or now with Information Operations on a global scale, AI is in charge of shaping human minds, which can only lead to a mathematically calculated self-destruction sequence. This is a direct derivative of such mathematical functions as cost measurements, maximization and even zero sum games in simulated virtual worlds of real world outcomes. With the incorporation of cybersecuity self-defense in these AI systems it also may be impossible to stop it once it has become an autonomous weapons system, because any attempt at stopping it would be viewed as a threat and engage it's self-protection protocols, or even with the purchase of a private system by a rogue entity or even theft by a rogue entity could even lead to it being used as a doomsday device setting nation against nation in a downward spiral, AI Apocalypse. As Nation-State weapons systems, and since we are talking about such powers as the USA and Great Britain, their robust requirements for their weapons systems is to survive even a nuclear war, which could mean that even if all of humanity is dead and gone, the machines will go on fighting.

Notes:

[1] Burns, Terri, (2018) 'Bad People, Bad Computers: Algorithmic biases in AI and machine learning', https://youtu.be/q1u6Q_jch9A (accessed 6/6/19)

[2] See e.g. 'Tay', a bot created by Microsoft who should learn from humans and turned into a Nazi within 24 hours, in: Steiner, Anna, 2016, Zum Nazi und Sexisten in 24 Stunden, Frankfurter

Allgemeine, March 24, 2016, available at:
http://www.faz.net/aktuell/wirtschaft/netzwirtschaft/microsofts-bot-tay-wird-durch-nutzer-zum-nazi-und-sexist-
14144019.html (accessed on February 15, 2018); or Google's search algorithm that spread false information with a right wing bias, in: Solon, Olivia, and Levin, Sam, 2016, How Google's search algorithm spreads false information with a right wing
bias, The Guardian, December 16, 2016, available at: https://www.theguardian.com/technology/2016/dec/16/googleautocomplete-
rightwing-bias-algorithm-political-propaganda (accessed on February 15, 2018).

Shadow AI:
See Tommy Thompson video on 'Killer Instinct' about AI ability to mimic player actions and strategies and replay them back at them through antagonists in the game.
https://en.m.wikipedia.org/wiki/Case-based_reasoning
Cbr created in the 1980s at Yale. Related to cognitive sciences prototyping.

Bibliography:

Bearden, Lt. Col. Thomas. (2005) 'Interview with Lt. Col. Thomas Bearden', Project Disclosure. Video: https://youtu.be/eNU3MLqyzPk (accessed 2/3/19)

Calvano, Emilio and Calzolari, Giacomo and Denicolo, Vincenzo and Pastorello, Sergio, (2018) 'Artificial Intelligence, Algorithmic Pricing and Collusion' (December 20, 2018). Available at SSRN: https://ssrn.com/abstract=3304991or http://dx.doi.org/10.2139/ssrn.3304991
https://www.technologyreview.com/the-download/612947/pricing-algorithms-can-learn-to-collude-with-each-other-to-raise-prices/ (accessed 2/15/2019)

De Byl, Penny (2019) The Beginner's Guide to Artificial Intelligence in Unity.
A practical guide to programming non-player characters for games.
Udemy Course: https://www.udemy.com/artificial-intelligence-in-unity/

De Jonge, Dave (2010) Optimizing a Diplomacy Bot Using Genetic Algorithms, Masters Thesis
Http://www.ellought.demon.co.uk/dipai

Fallon, John (2013) Believable Behaviour of Background Characters in Open World Games

Goodfellow, Ian. Bengio, Yoshua. Courville, Aaron (2016) 'Deep Learning' MIT Press online: http://www.deeplearningbook.org

Green (2019) ‘What Can Psychopaths Teach us About AI’ online: https://thenextweb.com/artificial-intelligence/2019/03/15/what-can-psychopaths-teach-us-about-ai

Gruber, Capt. Donald. (2015) ‘Tactical AI in Real Time Strategy Games’ Air Force Institute of Technology

Gunn, E.A.A. Craenen, B.G.W., Hart, E. (2009) ‘A Taxonomy of Video Games and AI’

Hudlicka, Eva () What Are We Modeling When We Model Emotion?
Psychometrix Associates, Inc.

Jagielski, Matthew. (2018) 'Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning' IEEE Security and Privacy Symposium
Http://github.com/jagielski/manip-ml
presentation video: https://youtu.be/ahC4KPd9lSY (accessed 3/6/19)

Larsen, John Alec. () A Chatbot Service for use in Video Game Development

Millington, Ian. Funge, John (2016) Artificial intelligence for games – 2nd ed. p. cm. Includes index. ISBN 978-0-12-374731-0

Osaba, Osonde and Davis, Paul (2017) ‘An Artificial Intelligence/Machine Learning Perspective on Social Simulation: New Data and New Challenges’ Rand Corp.

Robertson, Glen. Watson, Ian. (2014) A Review of Real-Time Strategy Game AI
University of Auckland

Sanatan, Marcus. (2019) ‘Theory of Computation: Finite State Machines’, online: https://stackabuse.com/theory-of-computation-finite-state-machines/ (accessed 6/15/19)

Steinhardt, Jacob. Koh, Pang Wei, Liang, Percy. (2018) ‘Certified Defenses for Data Poisoning Attacks’ Neuro Information Processing Systems Conference online:
https://papers.nips.cc/paper/6943-certified-defenses-for-data-poisoning-attacks.pdf

Surber, Regina (2018) ‘Artificial Intelligence: Autonomous Technology (AT), Lethal Autonomous Weapons Systems (LAWS) and Peace Time Threats’
ICT4Peace Foundation and the Zurich Hub for Ethics and Technology (ZHET)

Tarasenko, Sergey. (2016). Emotionally Colorful Reflexive Games.

Thompson, Tommy (2014) In the Directors Chair lecture 4

Thompson, Tommy (2018) 'Facing Your Fear' online: https://aiandgames.com/facing-your-fear/
-- The Road to War The AI of Total War (2018b)
Online:
https://www.gamasutra.com/blogs/TommyThompson/20180131/313865/The_Road_To_War__The_AI_of_Total_War_Part_1.php

Rose, Caroline (2015) Realistic Dialogue Engine for Video Games
The University of Western Ontario

Webb, Amy (2019) 'The Big Nine: How the Tech Titans and Their Thinking Machines Could Warp Humanity' Business Insider Online:
https://www.businessinsider.com/amy-webb-big-nine-artificial-intelligence-2019-2/?r=US&IR=T
(accessed 2/25/19)

Weber et al, (2011) Building Human-Level AI for Real Time Strategy Games, 2011 AAAI Fall Symposium