# Inspiration

Unexpected results in games are sometimes the most entertaining moments. Unexpectedness can emerge from complex interactions between simple elements such as physics simulations. Directly designing unexpectedness has the drawback of non-replayability, when the interactions are not free enough to have emerging unforeseen results.

Artificial intelligence (AI) in games typically contains the case of unexpected behavior in arbitrary situations. Ranging from role-playing games (ex. The Elder Scrolls series, GTA, etc.) to simulations (The Sims, etc.) to strategy (Warcraft, Command & Conquer, Total War series, Civilization, etc.), artificially intelligent agents (even if simple) can be a source of fun because of the uncanniness of their actions. Sometimes, they can surprise with clever solutions or skillful displays, other times they can do goofy things that a real player might never achieve (such as combining unexpected animations or sounds, or the like).

But AI in games is rarely truly intelligent, because the designs often specify behaviors that the agents need to follow in order to fit the other design decisions, such as story or game mechanics. Academic AI, as opposed to game AI, is focused towards creating more “truly” intelligent behavior, maybe in some cases one could call it more human-like. Examples of this could be search engines, advertisement systems that learn from the preferences of users, mechanical robots that can navigate, etc. However, this is rarely used for entertainment purposes, even though the potential is there (ex. Physical movement simulations, chat-bots, etc.). One of the sources of inspiration for this project is the unexpected fun of “serious” systems.

# Self-improving artificial intelligence

A self-improving ultra-intelligent agent has been said to be “the last invention that man need ever make” (Good, 1965). Self-improvement can be a very broad notion, but in essence, it means that the agent should be able to perform better on arbitrary tasks based on self-evaluation and adaptation. By self-improving, in addition to choosing the best course of action for different situations, the agent can modify its internal structure to better suit the needs of its functioning and to better fulfill its goals.

An analysis of self-improving systems by Stephen Omohundro, though abstract and somewhat futuristic, describes how a self-improving system will converge towards predictable behaviors founded on theories of microeconomics (Omohundro, 2008, p. 4). It also describes 5 stages of technology as *inert, reactive, adaptive, deliberative and self-improving*, relating to the way systems interact with their surrounding medium (Omohundro, 2008, p. 5). The article describes many generalities about self-improving systems, which can be used to form a list of requirements for a concrete implementation.

## Utility function

Omohundro’s article (Omohundro, 2008) demonstrates the need that an intelligent system must have a goal, and it formalizes this goal as a *utility function U*, which returns a score of how efficient an action (or series of actions) is towards achieving that goal (Omohundro, 2008, p. 8). The utility function, in the most generalized case, is defined on the domain of all possible histories of the universe, *S*, and uses a vector *P* to hold the probability that each history will occur. Even though this sounds very over-the-top, S is referring to the universe that the agent is located in (and interacts with). For example, a chess-playing machine would choose U to be the total number of chess games that it wins, S is the total amount of possible games that it can play. The system would therefore choose to pursue the scenario from S which is most likely, according to P, to achieve the goal of winning. P represents the system’s beliefs about the likelihood of each universe history, and at any moment in time, the system would have a subset of S which is compatible with the system’s current knowledge and actions that can lead to those situations (Omohundro, 2008).

## Set of actions

A fundamental requirement for an intelligent system is to have a set of actions that it can perform freely; it needs to use them most efficiently to fulfill its goals. For a self-improving system, those actions could even be modified and improved. The level of the actions can be very simple, such as activating a motor, to more elaborate, such as traveling the shortest path towards a destination. The way the actions are defined will affect how they interact with the environment and their effect as perceived by the agent. If an action does not have a perceivable effect in the environment, the agent will have no reason to perform it and thus it will end up unused/discarded.

## Perception

Another required component of such a system is perception. The information the agent has about the environment is what will determine the accuracy of its estimations about the utility of its actions and about the environment. The perception system must also contain information about the agent’s own state, such as health and other status parameters. This data will be used by multiple actions and goals, so it must be centralized in a different system than actions or goals.

For every unknown in the environment, the system will have to compute a probability, thus rendering the estimation a little more unreliable with each unknown factor. For simple scenarios, it could be possible to describe all parameters of the environment and have the agent make purely rational choices, but in most cases it is unfeasible from either computational or memory reasons to do so, therefore approximations must be used instead. It was shown in (Omohundro, 2008) that probabilities can be used for unknown factors in such deliberations.

# Game-oriented system

A simple way to test if such a system ­­­works is in a controlled environment such as a game, with few parameters, and having agents with few basic actions they can perform. The goal that the agents need to achieve should be something well defined according to the game universe. The proof that the system is working is shown if agents can achieve some degree of success based on the utility function, and can learn to use the correct actions, incorporate new actions in their behavior, or discard inefficient actions.

After the system is shown to work in a simple environment, a more elaborate design can be made to test it in more broad situations.

# Overview of game AI architectures

Based on this article (Mark, 2012), there are several examples of AI systems implemented in games. The purpose of a game-oriented system is entertainment, which is different from academic AI; the agent is not supposed to be better than the player, surpassing their skills and defeating them easily. They are supposed to provide a fun challenge, seem “intelligent” but not overly so.

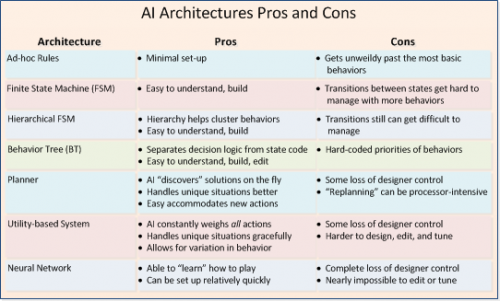


Figure 1 Table showing pros and cons of different architectures as described in (Mark, 2012)

The most common systems are linear, and as described by (Omohundro, 2008) could be categorized as *reactive* rather than intelligent. Basic rule systems and finite state machines would fall under this category. More complex systems such as behavior trees would split decision logic from actions, which means they would deliberate about situations and act accordingly, which makes them *deliberative* though still predictable. Finally, planner or utility based systems can have very intricate mechanisms of adapting the weights of actions usefulness, so they would be able to incorporate new actions as well as change their behavior after trial and error, which would make them more genuinely intelligent, but harder to stick to the game design. Neural networks are not described thoroughly in the article, but most likely would be very adaptive and unpredictable, and if used correctly might provide the most rational behavior.

## Goal Oriented Action Planning

A system very similar in concept to the requirements taken from (Omohundro, 2008) is Goal Oriented Action Planning (GOAP) (Orkin, Goal Oriented Action Planning (GOAP) , 2011), a variation on the STRIPS planning system (Nilsson & Fikes, 1971) developed as a standard academic AI at Stanford University in 1971, first used in the game F.E.A.R. (2005), but also used in other major titles such as Fallout 3, Empire: Total War, Deus Ex: Human Revolution, etc.

The GOAP system is based on the idea that the agent must use its actions in order to fulfill its goals, and the designer need only provide those actions and specify their effects (Orkin, 2006).

In F.E.A.R., the system is set up in terms of goals and actions, therefore agents can have common goals and actions but not behave the same way.

# Implementation

This chapter will guide the reader through the process of implementing the system, roughly, based on redesigns and reiterations.

## Required systems

Based on (Omohundro, 2008) and (Orkin, Goal Oriented Action Planning (GOAP) , 2011), the requirements for this system are as following:

* Basic actions
* Goals
* Perception

Making those systems separate has the benefit of being able to alter them independently for purposes of optimization, not depending on each other means they can be used in other contexts or generalized, and subsystems or techniques can be implemented inside each system that would make them more dynamic and/or adaptive. A self-improving mechanism can then be implemented by creating a separate virtual model of the entire system or each subsystem which would allow the agent to modify the structure of the systems while being able to know/learn the effects of such modifications.

## The connection between goals and actions

The agent’s actions will need to have specific effects described in some data structure, while goals have requirements that need to be fulfilled, which should be described in the same structure. This way, the agent will know what actions to pursue in order to fulfill its goals, without trial and error for every single action. Of course, trial and error can work by automatically updating this information in the structure after every performance of the action.

In (F.E.A.R., 2005, p. 12), the requirement/effect structure is four bytes, representing basic data such as a node in the navigational path, a vehicle type, a weapon type and a Boolean specifying if the target is dead. Depending on the complexity of the awareness system, this structure can get very complex; if intended for general use, it would most likely require all parameters of perception regardless if the action is manipulating them in any way. For example, an eating action would not require to know about the state of enemies around the agent, an attack state would not require info about food, but a defensive action would require both sets of data, so the simplest way to implement is to always use the full set of data, if its size is reasonable.

# Later implementation step: prediction and learning phases

The system was made so that the cycle of decision making, performing actions, and reevaluation is at the core. The decision making step involves predicting the next most useful action, performing actions is the part where the action is actually performed without the learning system interfering, and reevaluation is when the system analyzes the environment, finds the changes in the world, and interprets them in order to make better predictions in the future.

The scheme looks like this:

The perform action part is typical to what would be done outside intelligent systems. Each action will have functionality and effects, though the effects are not directly specified.

The perception system has functions which specify how to read the parameters of the environment. This system is the representation of the exterior world, the agent’s beliefs about how the world is around him. The perception also remembers the recent past so that it can evaluate the changes that are occurring.

Learning the action’s effects and the prediction system are the most troublesome; these are the real learning part of the system. The learning system saves the information about each action’s previous runs, while the prediction system uses that in order to evaluate which of those effects is most useful.

## Prediction

The prediction system must use the goal system to calculate the utility of each action, and then choose the highest scoring action to execute. If actions had clear effects this would be easy to do, because it would basically be any type of pathfinding algorithm, trying to find the best route to achieve ideal efficiency. But the actions have unknown effects, so they must be learned by experiment.

Calculating the utility function can be split into steps, as shown in the figure below:

First it takes the information about how the action influences the environment (in the form of a list of probabilities or probability curves). It passes it through each goal, calculating the utility of the new world state that would emerge if that change was added to the existing state of the world. Finally, add the utility values from each goal and finding the final sum, which is the total utility of that action.

For each action

The calculation of utilities is something that depends on the goals’ setup of the utility function. If the goals have simple utility functions, their calculation can be simplified even more through mathematical tricks.

The prediction system finds how each state influences the world from each action’s memory system, where the actions try to remember how the previous attempts affected the environment. The basic idea is based on causality: if a world state is similar to an older world state, and the same action is performed, it is most likely that the effects will also be similar, so their prediction can be used in decision making.

### Multiple actions

The prediction system is not very clever if it can only see into the future one single step. In order for it to see multiple steps, it needs to figure out the utility functions of the future actions and incorporate those into the next action’s utility. However, as demonstrated in (Omohundro, 2008), it is rarely the same if a rewarding action is performed in the distant future or in the near future, for example a person might have a greater utility for money received now rather than 10 years, depending on the amount. In this case, a way to decrease the influence of utility of distant future actions is presented by using a “discount factor” which decreases the result of the utility function at a specific rate (Omohundro, 2008). This can be implemented in various ways, but it is an important aspect in deciding future behaviors.

## Learning

To find how actions influence the world, the previous times actions were performed are the perfect way to find out if nothing else is known about the actions or their effects. As Immanuel Kant once wrote in his Critique of Pure Reason, “All our knowledge begins with the senses, proceeds then to the understanding, and ends with reason”. Therefore, it is justified to say that empiricism is something AI can also embrace.

### Purpose of the learning system

The requirement of the action memory system is to remember how actions affect the environment, so the agent can understand how to use that to its advantage, in the end. To know how an action affects the environment is not simply remembering the effects by themselves, because context of the world can sometimes affect the results of an action greatly. For example, making a fire might not be very successful following the same techniques while being out in the rain, in sunny weather or in a cave. In each case, something is different in the environment but not in the actions being performed.

From here on, a list of all the perception system’s variables, which describe the state in which the agent thinks the world is, will be called a world state, even though it is not a full description of the world, but an incomplete, perceived one.

An action can have an in-/direct effect on any of the parameters in the world state, as far as the agent can tell. There is no possible way of knowing what the action will affect if the insides of the actions are not known, and also if the state of the world is not fully understood. Therefore, every action must look for effects upon any single parameter of the world state, each time it evaluates itself, just in case there might be a positive correlation.

### Dealing with too much information

Remembering all the previous times the action was performed, along with all the parameters of the world, and possibly along with the other actions that were performed in that sequence, might be the most certain way to evaluate the best possible course of action, and calculate the highest possible utility that can be achieved among the possible actions. However, this is not feasible for memory and/or computational reasons, as the amount of data would increase exponentially with every addition of a parameter, or an action, or a goal. Therefore, a way to compress or abstract this information is needed.

There is research that focuses on learning mechanisms for data compression (Pezzulo, Butz, Sigaud, & Baldassare, 2009). That is not the focus of the current system.

Based on (Omohundro, 2008), the unknown data about the world state in general, and about the actions in particular, can be represented as probabilities, and the effects of actions on the world state are just that.

Certain knowledge can be achieved by finding the actual outcome of actions, by looking at the world state before and after the action has been executed, or after a specific time. From this certain knowledge, it can be said that those outcomes can (or could) happen, and the likelihood of them happening again is higher than the likelihood of completely different outcomes to occur in the future. Even if the outcome depends on a rule such as every time a button is pressed, a counter would increase, it still means that the outcome will be most likely semantically congruent with the previous outcomes, such as falling under a similar range of values.

Statistics shows us ways that samples of data can be interpreted to predict the behavior of the population they belong to; notions such as Gaussian distribution and standard deviation can be implemented in the learning system so that the predicted values fall under less unexpected boundaries.

The Fourier Transform can be used to determine the shape of the curve, towards which a specific parameter would likely tend, so that from fewer attempts the agent can make better guesses.

### Example scenario

In a possible scenario, an agent would have parameter P1 and P2 in their perception, action A1 and A2 as their possible actions, and would have to determine if the actions influence either parameter.

After a few attempts, it will be apparent that some actions have some influence on the world state. The learning system must start to assign values to specify the probabilities that each outcome is likely to result from an action. But the world state might influence the outcome of these actions, so the probability that an outcome would happen depends on the world state the action is performed in.

If A1 is executed after A2, it might seem that the effects of A2 depend on the fact that A1 was performed before it. However, it is clear that if the full state of the universe is identical, the results of an action must be the same; so unless there is a lot of data missing from the perception system about what A1 influenced in the world, A2 could simply look at the world state instead of A1, limiting the amount of data there needs to be computed.

### Requirements

Learning which values are affected by each action requires a data structure that remembers all the parameters that can be affected by the action in every context. For this, a table of previous actions can be used for remembering how the action affected the world state in the past, along with the state of the world before the action was executed (to be able to find if that influenced the effect in some way).

### Requirement for utility function

When calculating the utility function, the complexities of the curves that might result from the probability calculations should be taken into account; for example: imagine an action that has a small chance of generating a very negative response in a specific parameter, a zero chance of generating a zero-response, and a larger chance of generating a small but positive response. The average of the products of the three pairs of values - probabilities and chances - might be a null result. If a simplistic algorithm would look at this, it might not attempt performing the action since the effect on the world state is zero; but in reality, the effect cannot even be zero!

If the result from utility function in the very negative case is extremely bad, and the result from the positive case is not good enough to account for that risk, it might not be a good idea to treat that action as neutral; similarly, if the risk of the negative case occurring only results in a slight loss in utility, but the positive case is a very useful plus for utility, it would be better to perform that action instead of looking for more monotonous solutions. This brings us to the following requirement.

#### Calculation order problem

To compute utility function calculations based on probable effects, the probabilities need to be fed to the system after computing the utilities of the effects alone, and not before – this way, even if risky situations are neutralized by small chances of them happening, their value will be adjusted to the utility instead of the parameter response. This would make the calculation valid for all utility functions.

#### Workaround for calculation problem

A formalized way to describe the previous problem is to show what the effect of an action is in terms of individual possibilities and outcomes:

where is the total effect of an action (a world state) and to are the particular results which will occur with probability to . If then, the utility function for that action can be written as:

where is the utility of a world state and the total utility is the sum of the utilities of possible world states multiplied by the occurrence chance. If one was to simplify this and be able to compute the utility function of the total effect of an action, , the utility function must satisfy rules that allow it to form the following equation from the previous:

Provided that the utility function satisfies the following two rules, the calculation problem above can be solved easily:

And

A simple function that fulfills those requirements while still rewarding an agent for modifying parameters in a specific way (thus fulfilling goals) is:

where is any scalar (real number).

# Introduction (WIP)

This project is a learning system which uses pre-defined actions in the best way it can find in order to fulfill a specific predetermined goal, implemented in a game. An arbitrary but main requirement of the system is that each module/subsystem must be kept separate from the others, for ease of optimization, generalization, improvement, adaptation and reusability. Proof that the system is working is shown if agents can achieve some degree of success based on its goal, and can learn to use the correct actions, incorporate new actions in their behavior, or discard inefficient actions.

# Bibliography

F.E.A.R. (2005). Monolith Productions/VU Games.

Good, I. (1965). Speculations concerning the first ultraintelligent machine. *Advances in Computers*, 31-88.

Mark, D. (2012, 11 1). *AI Architectures: A Culinary Guide (GDMag Article)*. Retrieved from intrinsicalgorithm.com: http://intrinsicalgorithm.com/IAonAI/2012/11/ai-architectures-a-culinary-guide-gdmag-article/

Nilsson, N., & Fikes, R. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 189-208.

Omohundro, S. (2008). *The Nature of Self-Improving Artiﬁcial Intelligence.* Palo Alto, California: Self Aware Systems.

Orkin, J. (2006). Three States and a Plan: The A.I. of F.E.A.R. *GDC 2006.* Monolith Productions / M.I.T. Media Lab. Retrieved from http://web.media.mit.edu/~jorkin/gdc2006\_orkin\_jeff\_fear.pdf

Orkin, J. (2011). *Goal Oriented Action Planning (GOAP)* . Retrieved from Jeff Orkin: http://web.media.mit.edu/~jorkin/goap.html

Pezzulo, G., Butz, M., Sigaud, O., & Baldassare, G. (2009). *Anticipatory Behavior in Adaptive Learning Systems.* Berlin: Springer.