Cadeira De Introdução Investigação: Survey On Game Artificial Intelligence Techniques for Computer Games

Pedro Ricardo Varela Pereira[1]

[1] Instituto Superior Técnico
Av. Rovisco Pais 1049-001 Lisboa – Portugal
9 de Feveiro de 2007

prvp@mega.ist.utl.pt

Abstract. The computer game industry as grow in terms of diversity and importance over the last 20 years. As the graphic characteristic of a game, is no longer the selling point that it was a few years ago with the introduction of the first 3D games, developers must look for alternative game features. Game Artificial Intelligence is becoming a major game aspect and for developers to keep up with the demand of quality, alternative and innovating techniques must be explored. Over the years, research has been made by two major communities, the commercial and academic. The commercial community focus has been on developing artificial players to provide good game experiences despite how it is achieved. By opposition the academic focus has been on conceiving good models to mimic human behavior. This survey analysis techniques from both communities, providing a comparison on how the techniques can be applied to solve common AI computer games limitations. From the academic community three techniques are presented, an emotion based and two architectures, GRUE and MONAD. From the commercial community Orwellian State Machines, Goal Tree based techniques, Fuzzy Logic , Decision trees and State machines. An overview of other techniques is also presented such as Scripting and the Subsumption Model although others like neural networks are also referred. In the end, a global summary is made and the properties of a generic architecture for delivering efficient artificial game players is presented

Keywords: Games, Artificial Intelligence, GRUE, MONAD, Emotions, Orwellian State Machines, Goal Trees, Fuzzy Logic, Decision Trees, State Machines, Scripting, Brooks Subsumption Model, Neural Networks, A-Life, Generic.

1. Introduction

The objective of this Review

Before presenting any of the work, is important to clarify the main goal of this review. This review is intended to provide a knowledge base for my master's work. The main objective of this thesis is to deliver a generic architecture to provide artificial players for games. The first question is precisely what a generic architecture is. By definition, the "generic" word defines something that is applicable to a large set of entities. With this review, I aim to define what the existing "specific" architectures provide, in order to determine what can and cannot be applied in a generic architecture for games.

Off course, nothing completely general can be defined because there are always some compromises taken and some decisions must be made in order to achieve an architecture that can really be up to its goal of providing artificial players for games. In this review, the game genre is almost completely absent. This is because an independent review could be build for each of the specific game type genres. It was my decision to remove that part of the equation and present architectures from a neutral point of view were we are only looking at what is possible to do with them instead of analyzing the best use for them.

AI in Games

The role of artificial intelligence in games has always been problematic. Until very recently, the resources available for AI in games were very limited because other game areas were very resource consuming, specially the graphical part. Also, it was rare to see AI referred as a marketing sales point [6] [12] [17]. This lead to a secondary role of AI, that only really grew when the game industry started looking at alternative ways to promote games when the consumers asked for better games.

According to [12] and [17] there is a clear distinction between AI for games, and academic AI. The most important aspect that is relevant to this work, is simultaneously one of the few that is consensual between the game developer and academic community. That aspect is that AI in games is primarily concerned in bringing the illusion of life with the least resource consumption possible. This results in a common family of problems that emerge in games that have particular solving methods but have no regard to any "formal" definition of "agent". These families of problems include representation of movement, pathfinding, decision making and learning. As stated in [17], sometimes the AI can have access to privileged game information ignoring the game rules if that makes the agent more efficient in the game, improving the overall gaming experience. In the last years, there have been several approaches from both communities, and many games are making use of well accepted AI academic models [12] because there are enough resources to make them work in games.

However, the use of tweaks in game computer AI, always has been present, specially in a time where the game industry has grown into a multi billion dollars industry, and game publishers pressure the developers to present games in "*time to market*".

1.1. The information Taxonomy

Before presenting any information, it is important to organize information to allow a better understanding of the problem in analysis. One big problem in structuring information over this topic, is that usually the Game AI community doesn't focus the documentation about their systems from an academic point of view. This means that they just present a system from an implementational point of view to specifically highlight the game problem they wish to solve. This makes the task of trying to compare in the same document, well defined architectures and models mostly from the Academic Community to poorly connected schematics and descriptions from the game community a great challenge. As such, the taxonomy will reflect the origin, if it is from the academic or commercial industry, and the type, if it is a complete architecture, a model or a method. Because the aim is at presenting the ideas behind each technique the type of the technique becomes less important in this review.

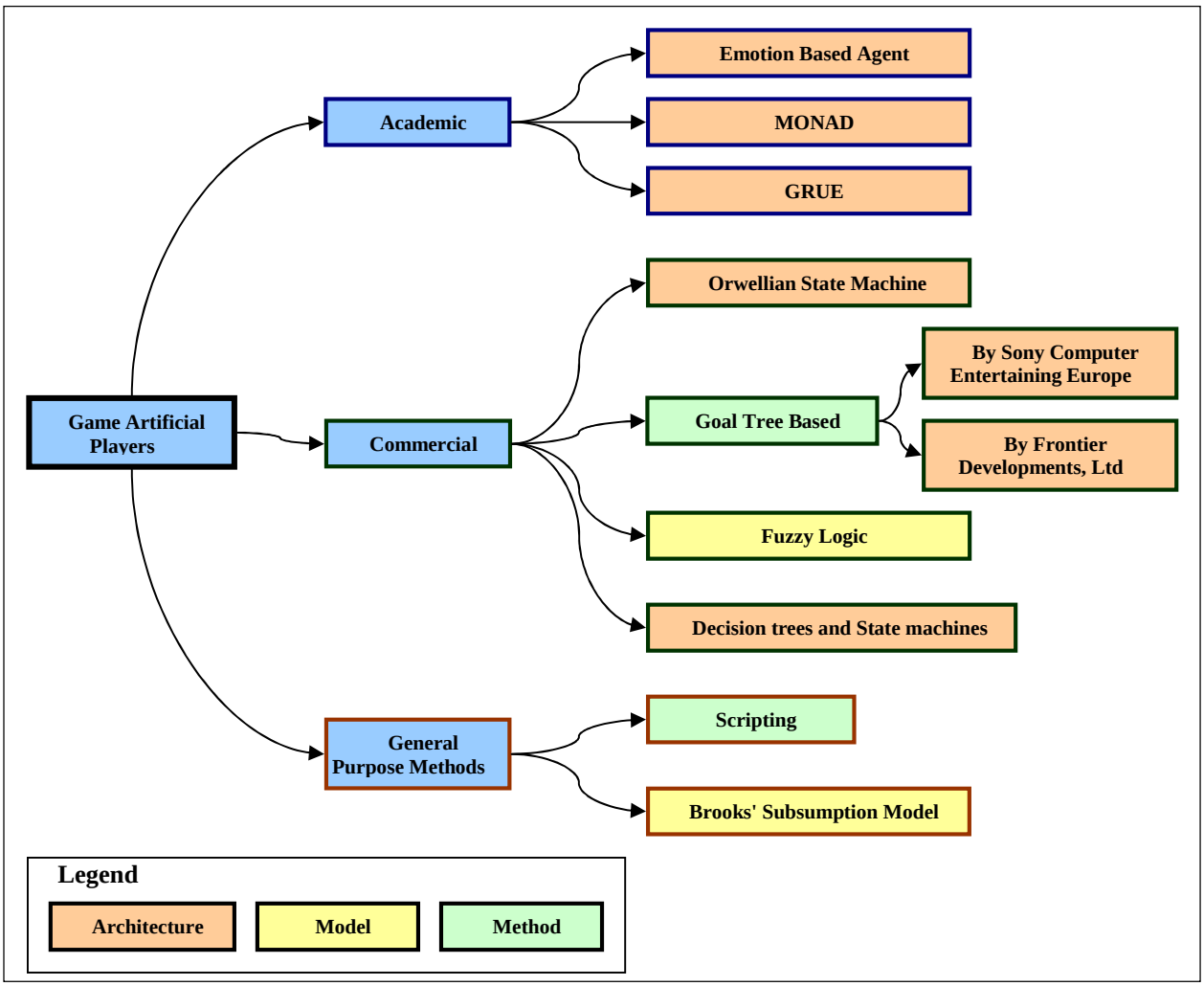


Figure 1 - The Taxonomy

In Figure 1 we can see the proposed taxonomy. To start, this work will present an overview on specific methods to achieve game artificial players from the academic and commercial communities followed by other general methods. Although the focus will be on the first ones, the general methods also are of great importance as they can be used as part of the other systems.

The emphasis of this work will be in present the general idea of each of the techniques highlighting the points that one developer may consider important when choosing a way of delivering an artificial player for a game. In fact, two roles that are implicitly present through this work are the role of “developer” as being the one that chooses and implements a software solution and the role of “content creator” that assumes a built system, providing data content to it.

1.2. Information topics

Now that I’ve presented a way to categorize the information, is important do decide what information will we gather. The review is about techniques but there are many ways to present a technique. For example, if one is presenting an architecture, as is well highlighted in[15], there are many ways to document and present one. What is expected in this review is to show the basic ideas behind the implementation from a higher level perspective. With this, it is possible to have an idea of the general capabilities of a system that applies that technique, and to provide comparison metrics with other different systems.

So, for each different system, the following information will be provided:

- Global Overview of the system: What are the system components, and how do they connect themselves? What were the desired capabilities of the system?
- Relevant distinct module information: What are the most notorious aspects of this system and what is their purpose?

1.3. The evaluation metrics

As is well patent in [16], a software architecture can be evaluated in many ways. However, traditional ways of evaluation are not well suited to the kind of results expected from this kind of review from an architectural point of view. In a quest for a general architecture for artificial players for games, there are a lot of aspects to consider ranging from raw physical resource management to decision making. As such, I will define a set of metrics derivate from metrics used in books such as [12] and [17], considering aspects presented in [7] and also in [5] instead of traditional ways.

The following metrics will be used to evaluate each of the systems presented in the following section:

1. Flexibility: How good is the system, when used in different applications? How does changing game type or mechanics affect the system?
2. Performance: How is the balance between capabilities and resources needs?
3. Decision Making: How good can the decisions be? How does it compare to what a human player would do?

For each system, I will present relevant aspects for each of the metrics, to derivate a final score ranging from 1 to 5 where higher is better. The score will be given amongst all systems meaning that a score of 5 for a metric, means that the particular system, amongst all others presented here, is excellent.

2. Techniques

In the next three sections I will give an overview of the main methods for making artificial players for games as presented in the taxonomy. The first will be dedicated to AI techniques from the academic community more specifically agent based intended to be applied in games. The second will describe several different ways used by the game AI community to deliver artificial players. The last section will present alternative techniques that do not fit in the previous sections but whose distinctness helps to have a wider view of how can an artificial player in games can be made.

2.1. Artificial Intelligence Techniques - Academic

In this section, I will present three generic, agent based techniques for agents. Although they are to be used in games, they try to never forget the idea of designing an agent, in an academic point of view.

To begin, I will present the MONAD and GRUE architectures that implement the Belief Desire Intention (BDI) mode[1]l. In this model we have Beliefs that represent some information the agent has about the world, Desires that represents some state of the world that the agent may want to accomplish and Intentions represents paths to accomplish some desire.

Finally I will present the architecture of an emotion based system that represents the increasingly importance of emotions in the task of bringing the computer systems closer to humans.

MONAD Architecture [2]

Global Overview Of The System

The main objective of the MONAD architecture is to integrate the well known BDI model in a multi agent architecture. To accomplish this, the authors start by representing desires and intentions in an acyclic connected Graph. They define two types of nodes in a graph and two types of relation between them.

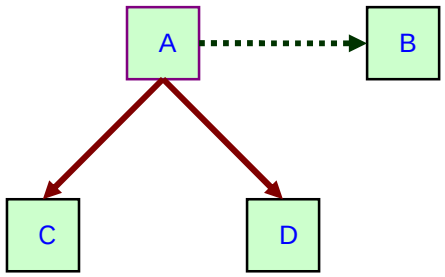


Figure 2 - Monad nodes and relations example.

The nodes can be:

- Leaf Nodes: Leaf nodes or atomic nodes correspond to Intentions that the agent can carry out. Correspond to the node set {C, D, B} of Figure 2.
- Non Leaf Nodes: Non leaf nodes correspond to nodes whose accomplishment depends on their children nodes. This means that the relations of these nodes behave similar to a “And-Or Graph” They correspond to the Desires in the BDI model. Node set {D} in the desire example of Figure 2.

The relations can be:

- Follow: Correspond to the dotted line on Figure 2 and are also referred as horizontal connections. They express a follow relation for example, in Figure 2, the B Intention is only triggered after the A desire is achieved.
- Decompose: Correspond to the hard lines in Figure 2 and are also referred as vertical connections. They express a decomposition relation. In Figure 2, the A desire is decomposed in {C, D} meaning that to achieve the A desire, one must first achieve de C and D intentions if the A node is an And node, or just C or D if it is a Or node.

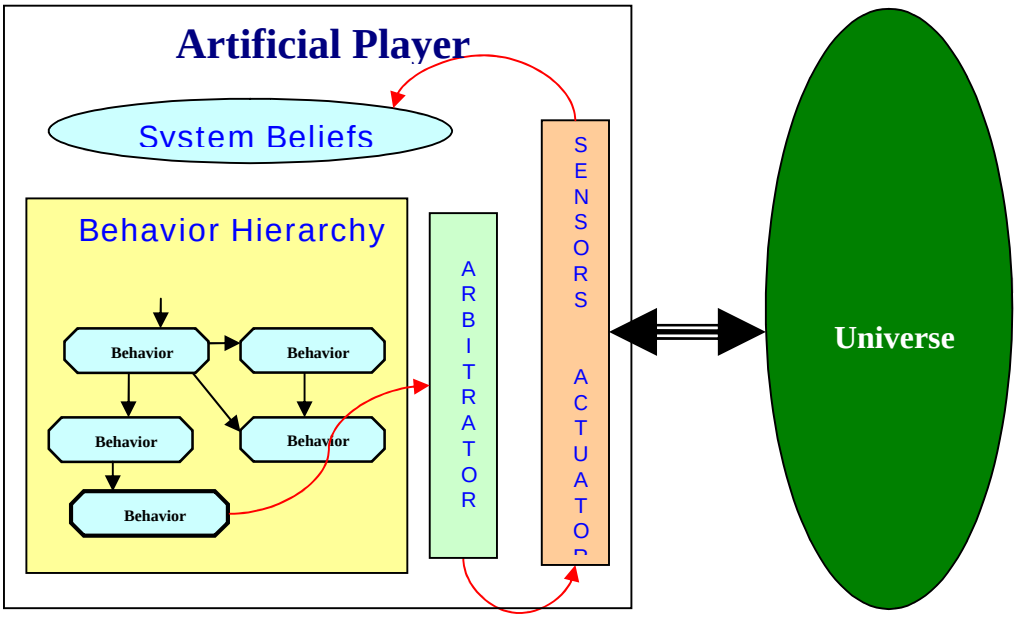


Figure 3 - The Monad Artificial Player

The global system can be seen in Figure 3.

- Beliefs – A belief of the system, is no more than some fact about the world that the system beliefs as true.
- Behavior Hierarchy – The behavior hierarchy explained above is where the Desires and Intentions of the BDI model are implemented.
- Sensors and Actuators – World interfaces where the sensors update internal beliefs and actuators carry out the actions of the system.
- Arbitrator – Entity responsible for coordination between agents or agent teams.

This architecture is specially intended to scenarios where we want a large number of agents cooperating. The DAG as a desire representation allows decomposing the agent actions, into sub actions, and integrating them into an implicit state machine build around the horizontal connections of the DAG. One particular usage scenario is First Person Shooters (FPS) games, where a group of agents is present in a scenario, and must search and accomplish some objective such as eliminating their opponents. With this architecture, is relative easy to build a cooperative agent team to accomplish some simple task because the agent states can be mapped into the horizontal connections, and the possible alternative ways to accomplish the task can be mapped into the vertical connections. The arbitrator is then capable of coordinate the behavior path chosen, amongst all team members.

Relevant Distinct Module Information

Two aspects of this architecture are very particular. To begin, the DAG representation of the BDI model is a somewhat poor implementation because for example, it is arguable that an atomic node is in fact an Intention of the BDI model. The other aspect is the role of the Arbitrator in the system. Traditionally an Arbitrator is an entity responsible for resolving conflicts between conflicting actions of the agent. In this case, the Arbitrator resolves conflicts between agents. As stated before, the architecture intends to provide a base for multi agent use scenarios where several agents cooperate to achieve their goal. However, if they all have the same DAG a problem arises. When facing the decision of choosing a “follow” relation, they will always choose the same. To solve this issue, each time the agents make a decision, they communicate between them what action was performed. When one agent has for example, two vertical connections in their DAG, they will choose the one that least agents took before. This allows the agents to work as a team and accomplish a somewhat broader view of their behavior hierarchy.

GRUE Architecture [3]

Global Overview of the System

The main purpose of this architecture is to allow the easy development of agents capable of actuation in very dynamic environments based on the BDI model. The main aspect of the architecture is that it does not assume the desires to be static and constant entities in the agent mind through his life in the world. Usually the agents are presented with desires when first entering the game even if the world changes. In this approach, instead, they are generated by the agent as an adapting step to a changing world. To achieve a goal, "Teleo Reactive Programs"[10] (TRP) are used to build plans of actions. A TRP is an entity that achieves some goal. To do that, it is built around a simple structure as can be seen in Figure 4

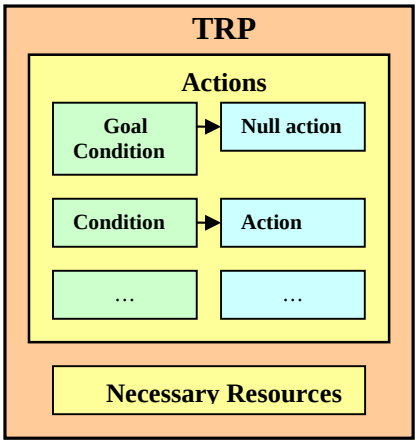


Figure 4 - TRP structure.

In this structure, we have a set of pairs of <condition, action>. Each time a condition evaluates to true, the respective action is performed. When a condition does not evaluates to true, then the bottom pair is evaluated and so on until reaching a true condition. The idea is that each lower level action, makes a part of a topmost condition true. When the goal is achieve, the null action is fired signaling that the TRP is no longer operating.

As a way to eliminate problems where various TRP´s are activated simultaneously and have conflicting conditions or effects, there is a notion inherent to the architecture that is the notion of resource. A resource is a representation of something the agent has in limited quantities and that can be shared. As such, each TRP define a set of resources that the agent must have to the TRP to be able to run. When executing TRP´s, each must acquire the necessary resources and only those that can do that can be activated.

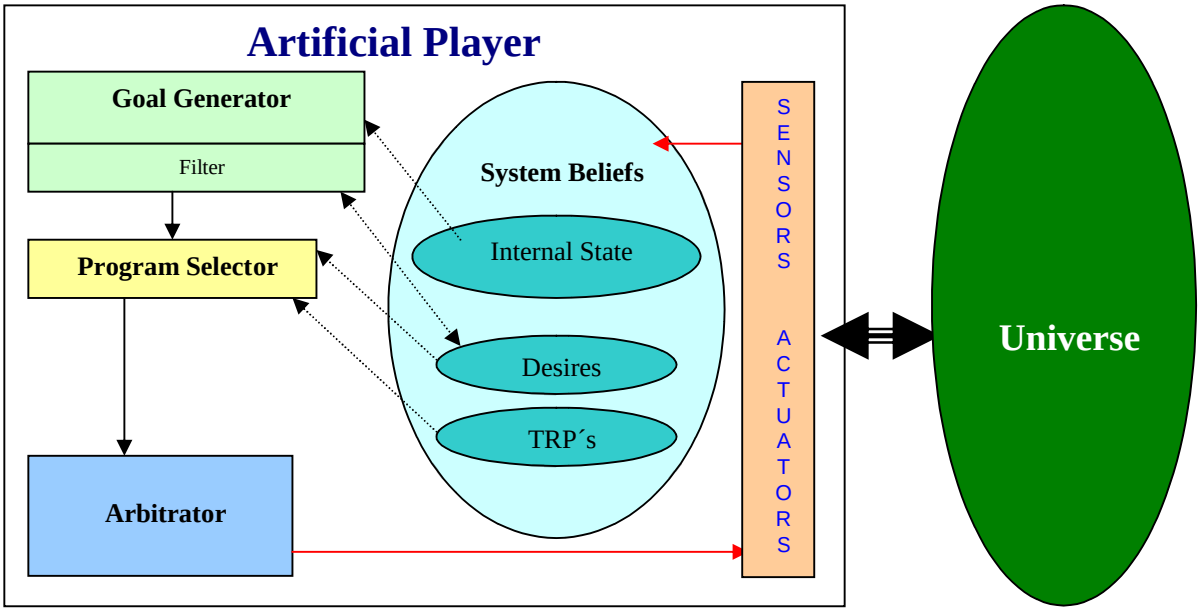


Figure 5 - The GRUE Artificial Player

As a global view, Figure 5 shows the architecture global view.

- Sensors and Actuators – World interfaces where the sensors update internal beliefs and actuators carry out the actions of the system.
- Goal Generator – Module responsible for managing the set of desires of the agent. In the presence of certain beliefs it builds the set of desires of the agent and adjusts their priority. It then passes those desires to a filter that discards the lower priority desires.
- Program Selector – This module defines a set of TRP´s for each goal such that the final condition of the last TRP accomplishes the goal. Its basically a planning module that aggregates TRP´s to achieve each goal. Each of these plans is referred as a task.
- Arbitrator – From the set of possible tasks, the arbitrator decides which to be put in execution in the current cycle. In practice, the higher priority tasks will be executed first. The number of tasks performed depends on the available resources. This is because the execution of TRP´s progressively consumes resources. If TRP´s are in conflict only the TRPs in the highest priority task will be activated.
- Beliefs – A belief of the system, is no more than some fact about the world that the system beliefs as true. These include the current desires and the TRP list.

Its main quality is its adaptability to the world since it is capable of generating desires and plans, in a very flexible way. However it requires great knowledge of what to expect in the world to be able to represent the primitives for the desire generation, and TRP execution.

Relevant Distinct Module Information

The three main aspects of this architecture are:

1. *No predetermine set of desires*
2. *Notion of resource*
3. *Notion of TRP*

The first aspect is most important when the agent is in a very complex and dynamic environment and must accomplish difficult task, to implement this in practice, the agent has a two layer planning process. In the first it decides what he can achieve and in the second how to achieve it. The notion of resource is important because it adds a way to allow the agent to do several tasks at the same time. The TRP is a basic action planning object that consumes resources and triggers actuators. A TRP is viewed by many as a small reactive agent based in the subsumption theory that satisfies some condition. This adds a third hidden layer to the two planning layers which allows for several ways of achieving a condition in the same TRP.

Emotion Based Agent Architecture- Kismet [23]

Global Overview Of The System

Kismet is a robot head that symbolizes a shift in the way humans look at autonomous agents. Robots have been seen as mere machines to accomplish unpleasant or impossible tasks such as space building [21], space exploration[22], bomb disposal and even nuclear wastes cleaning. These tasks require agents mainly because they require quick and precise answers and a human controller is not a feasible option. However, these tasks do not require the agent to "appear" nice to the human. They are machines with a concrete task, and that is how humans look at them. Kismet is an example of the "humanization" of robots where these cold machines, are given the ability to interact affectively with humans.

The Kismet project is a multi area project with ramification to robotics, sensor equipment, and emotion generation. To this review, the most relevant work is on the agent model and the inherent emotion generation.

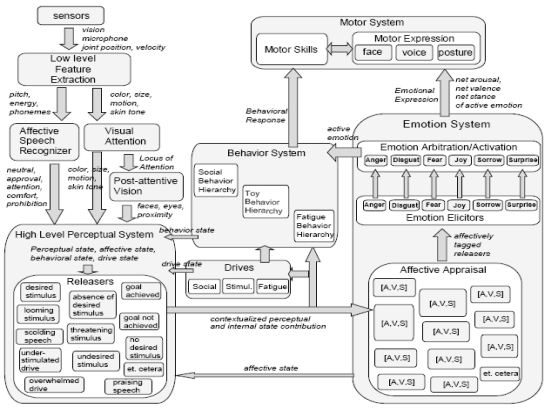


Figure 7 - The Kismet Architecture

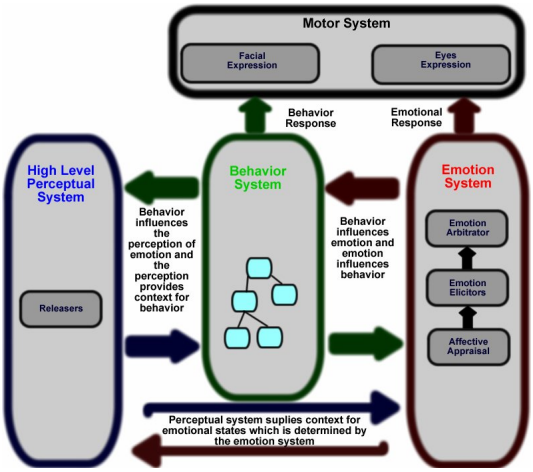


Figure 6 - The Kismet Simplified Architecture

In Figure 7 we can see the big picture of the agent model of Kismet. As we can see is a complex system and to describe it even briefly would require more than the length of this review. However if the picture is compressed we can end up with something like Figure 6. In this picture we see the role the emotion has on the behavior system. The behavior is directly influenced by the current emotions the agent has. As such, it really acts the way he feels like at the moment. This provides a much more richer experience to whom he is interacting with.

Relevant Distinct Module Information

This is a review about architectures for artificial players for games, so one may ask why is emotional agents important. To answer that, one as to look at current top games such as Oblivion IV and see the increasingly important role that human relations between human players and computer agents have. So, agent based systems such as Kismet that mimic human behavior in such an amazing way must point the way to the future in computer games. There is now a tremendous amount of games such as NeverWinters Nights 2 and Sims 2, where the emotional aspect of computer characters is in the core of the game.

General Capabilities

Of course, this architecture will not be evaluated as the rest. Kismet as stated before, is a multi area project and the entire system is dedicated to the simulation of one robotic head and not to an agent capable of acting in a real game or simulation. Even so, it represents a property that is emerging has one of the more important in current computer games and may well point the way to future computer games where we will have more human like characters.

2.2. Artificial Intelligence Techniques - Commercial

This section will present some of the methods commonly used in commercial games. To start, I will present the Orwellian State Machines architecture that has its bases on the Subsumption theory presented in section 2.3. Then, I will present two methods based on Goal Trees followed by several methods that combine decision trees and state machines.

Orwellian State Machine Architecture [11]

Global Overview Of The System

Orwellian State Machines (OSM) can be used in many aspects of an artificial game player. It can be used in the decision making process, but also in other game aspect such as the graphic representation. However, here the focus will be in the decision process.

The main idea behind the OSM model, is to provide a way to control the problems of the subsumption model. As stated in section 2.3, one of the main problems in this model, is the complexity it quickly achieves making it hard to use in complex environments or with complex tasks. The OSM is built around three basic concepts that together provide a way of controlling the emergence problem:

- Controller: A Finite State Machine that controls some "atomic" AI function that can communicate with the other members of the system (In [11] the "controller" is designated as "agent", but to prevent a misunderstanding of the term, it was renamed in this survey).
- Dispatcher: A dispatcher controls interactions between controllers and provides input to them.
- Collective: A collective is the topmost element of the hierarchy that groups controllers and dispatchers.

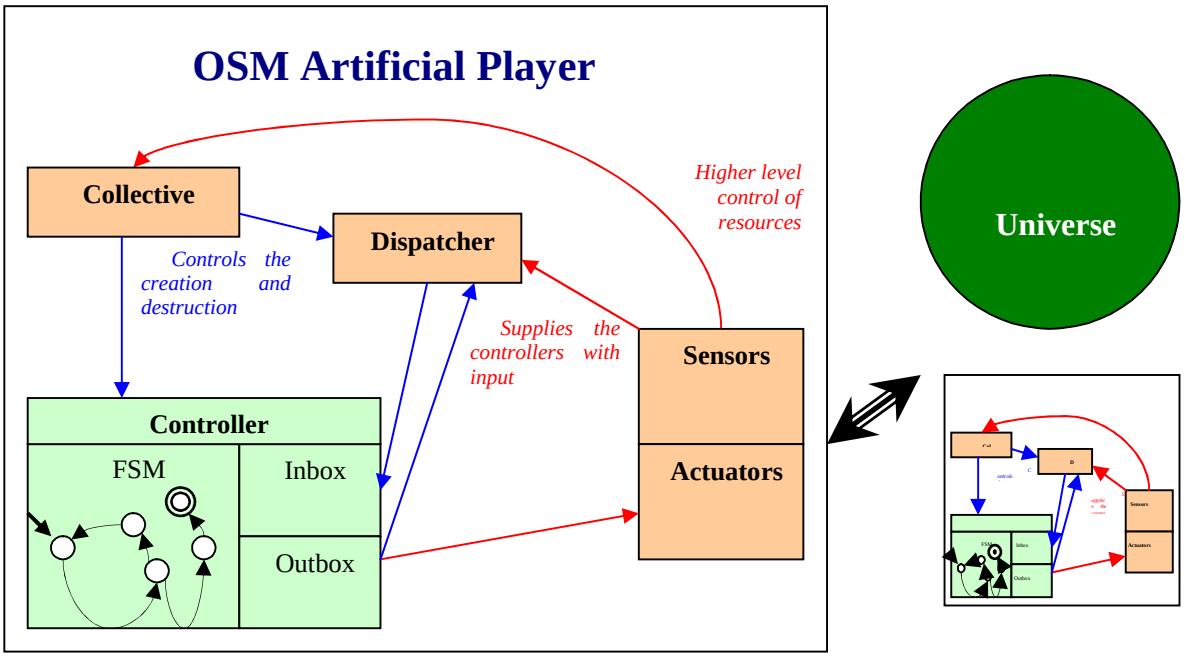


Figure 8 - The OSM Artificial Player

The first thing to notice is that in Figure 8, there is a change in the connection between the universe and the system. This is due to the nature of the OSM that is intended to be modular in such a way, that it can connect to a higher level OSM functioning as a Controller. In this sense,

the sensors would become the Inbox, and the actuators the Outbox. This allows the combination of several OSM, each specialized in a function, where the topmost OSM connects to the Universe and relies the inputs to the others.

- Collective – The collective manages controllers and dispatchers being able to coordinate the resources available.
- Dispatcher – Supplies input to the controllers based on sensor information, and reports supplied by each controller. It also manages resource allocation when multiple controllers ask for the same resource.
- Controller - The controller is the operating part of the system. It comprises all behavior associated with a particular agent responsibility in a finite state machine (FSM) [4] although the author says that is only a possible choice and that other methods could have been used.
- Sensors and Actuators - As said before, its an implicit part of the system that can have multiple roles depending of the connection of the agent directly to the universe, or as part of a higher level agent.

This architecture provides ways of building agents in a progressive way by adding new controllers and dispatchers to the collective to use. It allows the making of AI groups in a somewhat strange way, by combining several agents, into a collective, and assigning them a dispatcher to coordinate actions

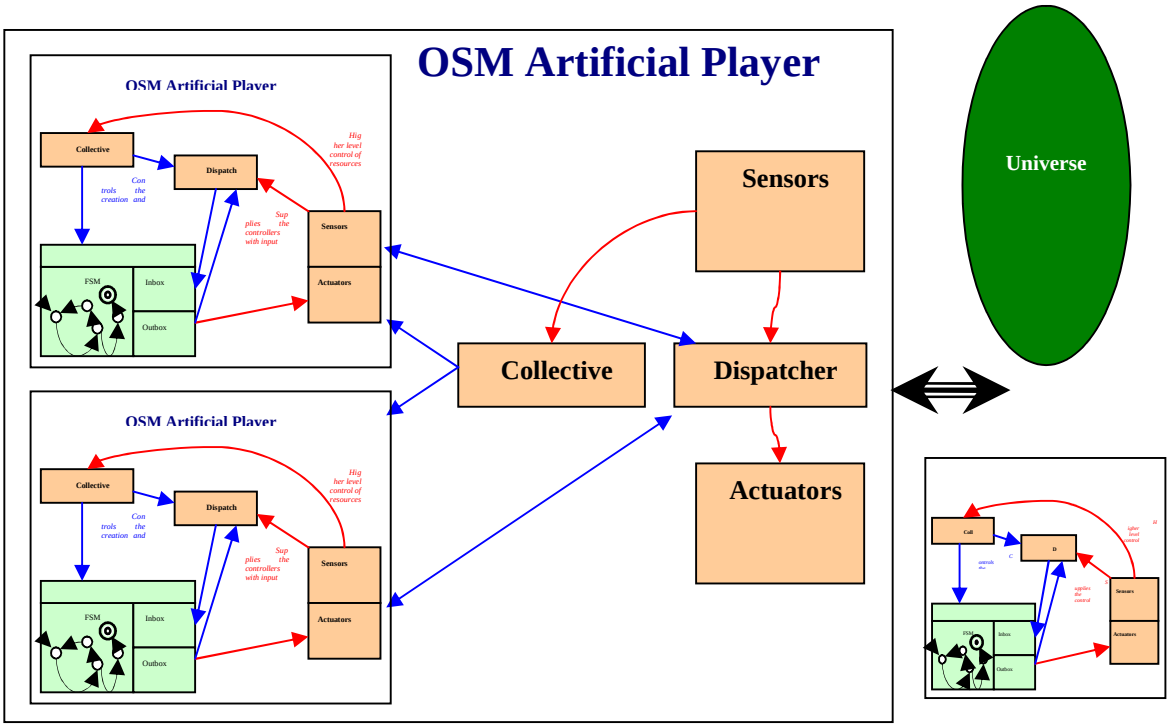


Figure 9 - Combining OSMs into a higher level OSM

In Figure 9 there is an example where several agents with this architecture are combined to work as a team. To the academic community this approach can be very controversial because from an agent point of view, for example, the combination at the decision level of agents, does not makes sense because it is limiting the autonomy of the agent at very low level. The agent can act, but his actions can be changed by others without it being able to do anything. However, in the light of the requirements of a game, it makes perfect sense as it facilitates the control of the content creator over the expected behavior of the artificial players.

Relevant Distinct Module Information

The main feature of this system is the segmentation of the global AI actions agents can have, into distinct modules denominated Controllers. The subsumption theory is then applied in each individual controller. A FSM models the pairs of rules / actions, and resources are managed by a mediator entity denominated as dispatcher. This allows controlling the rule set explosion that occurs in the basic model presented in section 2.3 by aggregating the rules in individual entities not aware of each other actions. In fact, the controllers can send messages between them but the author points that it is important that no controller knows who sent the message to disallow direct dependencies.

Also important, is the fact that this architecture can be applied to virtually all aspects of the artificial player in a game and not only the decision process. Combining OSM specialized in aspects as animation, movement and decision making, it is possible to build the entire representation of the player in the game.

Goal Trees

Usually in computer games, the AI developer, is focused in determining how an artificial player will react to events in the world[15]. Sometimes however, “reaction” is just not good enough to maintain the illusion of life. The human player must be made believe that their opponents are making actions because they want, and not just because some trigger fired. In some games it is imperative that computer players have goals. One good example is “The Sims” game, where the human player manages a set of computer characters each with its own goals in the game. The definition of goal is a somewhat confusing on in the game community and is used to express many different concepts such as desires or intentions. However, the ultimate objective is just to provide meaning to the artificial player actions giving it a sense and purpose in the game.

One way of accomplishing that, is with a goal trees approach that provides mechanisms for the content creator to decide what will the artificial player do. In this survey, two goal tree architectures are presented to capture the expressiveness of the goal tree based approaches.

Goal Tree Based Architecture - By Sony Computer Entertaining Europe[13]

Global Overview Of The System

Just like the previously presented OSM, this architecture can be used in several aspects of the artificial player in the game.

Some of the problems this architecture tries to solve are reusing behaviors, and the imposing of a control mechanism over the agent behaviors. The first problem occur when we have programmed behaviors that in essence are the same but have special nuances that forces the developer to repeat behavior code. The second problem occurs when the developer wants to impose courses of action, or prevent the agent of doing something. To address this, this architecture is build around the concept of goal. A goal is something the agent wishes to accomplish. It can be a pathfinding problem, a strategy problem or even a graphical representation problem, in this architecture it is all the same. A goal can in turn, be decomposed in sub goals and from these, emerges the tree topology that consists in a set of connected active goals.

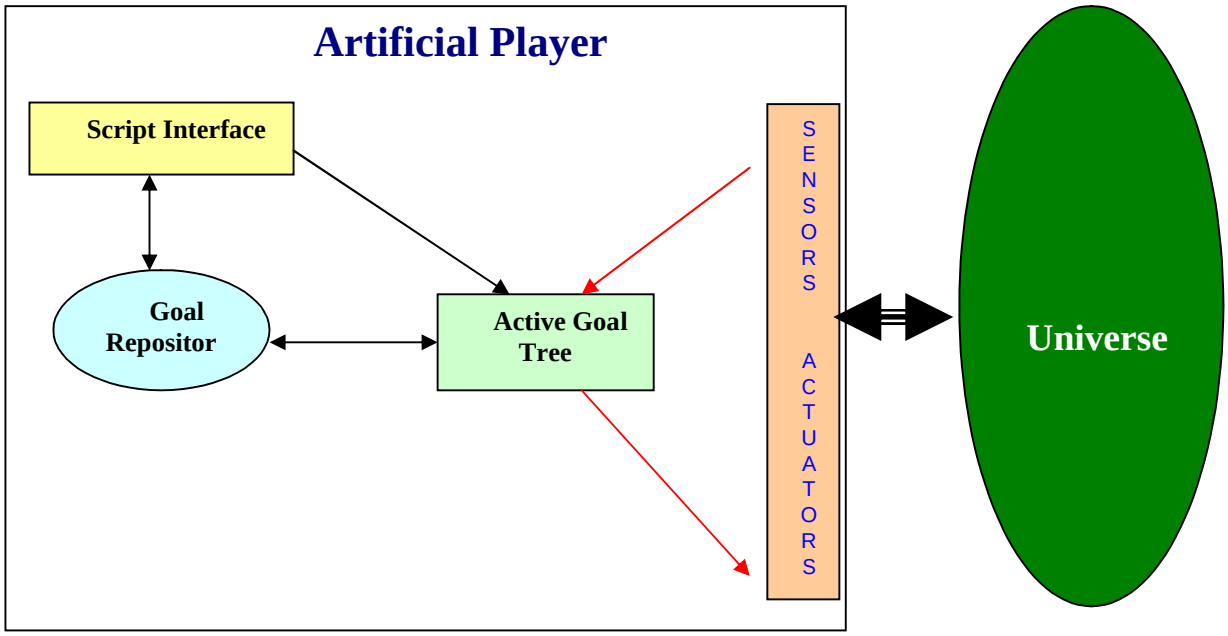


Figure 10 - Sony Goal Tree Based Artificial Player

As we can see in Figure 10 the actual system is quite simple. There is no explicit total tree representation, as the tree is built in runtime.

- Sensors and Actuators - The sensors build a set of parameters that are then given to the goals translating sensed information about the universe. The actuators consist in triggered actions of the agent over the universe.
- Active Goal Tree - Initially this goal tree has the default goal. Each time a goal is evaluated, if it is atomic (meaning no sub goals), the actions inside are carried out. If it has sub nodes, those are added to the active goal tree and evaluated also.
- Goal Repository - The complete set of goals the agent can carry out. Those can be primary goal or secondary goals. The difference is just that only the primary goals can be placed as root of the tree. Each goal carries the code for performing actions, and to select sub goals.
- Script Interface - The script interface provides a way to the developer to control the goal tree growth forcing goals to be added or removed. It also allows customizing the sub goal selection by supplying heuristics based on the sensor information.

This architecture provides a way of reusing behaviors for different agents by the addition or removal of goals from the goal repository. By using a dynamic tree of goals, it provides an easy way of providing adaptability properties to the agent.

Relevant Distinct Module Information

The most relevant aspect in this architecture is that it provides a unique interface for the “goal” concept encompassing all agent aspects in the game from decision making to the game animations. To improve performance and to better organize information, the goals can also be grouped into selection groups. This selection groups have a define hierarchy where the higher lever have more priority. In absence of alternative sub goals selection in a goal, the first higher level goal will be choose.

Also to note that the level to control is very large as the developer must design each goal individually. This is very useful in games as it gives the developer the power to program special cases inside each goal instead of modifying all the other so that the special case can be successfully solved. The script interface allows these rules to be passed into the goals in an easier way at runtime. Also, when designing similar agents, the removal of goals can force the system to behave differently delivering different agents from a game perspective.

Goal Trees for Multi Task Agents Architecture - By Frontier Developments, Ltd. [15]

Global Overview Of The System

This architecture is specially intended for complex environment where the agent has to have a quick response time. It consists in a mixture of the subsumption model with a more classic tree goal base representation. Each possible behavior is coded using the subsumption model. As such, each behavior has a set of pairs <condition, action> and when a behavior is activated, it returns the action associated with the first condition evaluated to true. The actions can in turn reference other behaviors forming an implicit tree of behaviors.

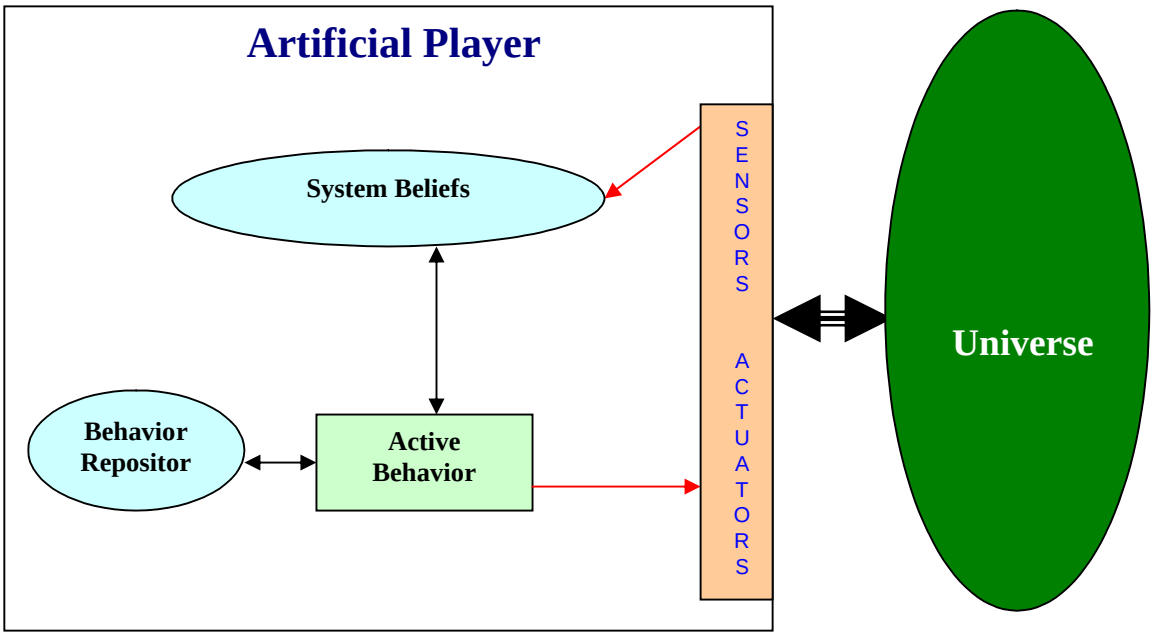


Figure 11 - The Frontier Developments Artificial Player

As we can see in Figure 11, the system is fairly simple, being composed by only a set of behaviors and one active element. This architecture is intended to use cases where we want fast agents, and the ability to easily reuse elements. It can be seen in a cut down version of the previous architecture but in fact, this one is more directed to agents performing simple tasks without the need of extreme ways of improving the decision process. Also, in this architecture there is the explicit notion of a belief. It has a fairly simple internal language that is used to represent knowledge about the world. This knowledge can then be used in the dynamic construction of the behavior tree through a query language based on the knowledge language.

Using this architecture, one can deliver a great number of agents in a system as they are very resource efficient. Also, there is the possibility to define behaviors in a data driven way, based on the internal beliefs of the agent.

Relevant Distinct Module Information

This architecture presents two very important concepts. First, it has an explicit knowledge representation of its perceptions. In previous game architectures, the beliefs were not even part of the model and were directly connected to the decision making modules. Of course, AI theorist can argue that an implicit belief is always there because the agent doesn't process raw data. However in practice that is not always true especially in commercial games where the main objective is always to provide performance and "shortcuts" in the decision process, and as such, can bypass the common AI model of sensor / actuator.

The other important concept is that the tree expansion is made using the knowledge base, using a unified notion of behavior. This makes behavior reuse possible because several equal behaviors can be activated with different parameters. With an optimized query language,

also allows the deliver of resource efficient agents able to perform various tasks in great numbers.

Fuzzy Logic Based Model [19]

Global Overview of the System

Before going in the details of this model, the main concept has to be better clarified. Fuzzy logic is a branch of mathematics that does not considers only the binary values TRUE or FALSE but also considers degrees of TRUE between the two traditional values. There are numerous debates over this theory and many ways to explain the concept. Of interest for this review, the primary idea to retrieve is that it is possible to associate degrees of truth to some events that allow to a more accurate control. The example presented in [18] is very elucidative of the concept of evaluating how true something is. The author presents a set of empiric rules that describe the driving behavior in the road. Then, it presents a set of action that can be made such as increasing speed or breaking.

- If *distance* is small and *distance delta* grows, maintain speed
- If *distance* is small and *distance delta* stable, slow down
- If *distance* is perfect and *distance delta* grows, speed up
- If *distance* is perfect and *distance delta* stable, maintain speed

The distance measures how far the drivers car is from the next car. The metrics is based on distance cars where perfect is 2 cars distance, small is 1 car and big is 3 cars. The distance delta measures the variation of the distance.

Figure 12 - The rules (adapted from [20])

In Figure 12 we can see the set of rules that translate empiric procedures to circulate in a highway.

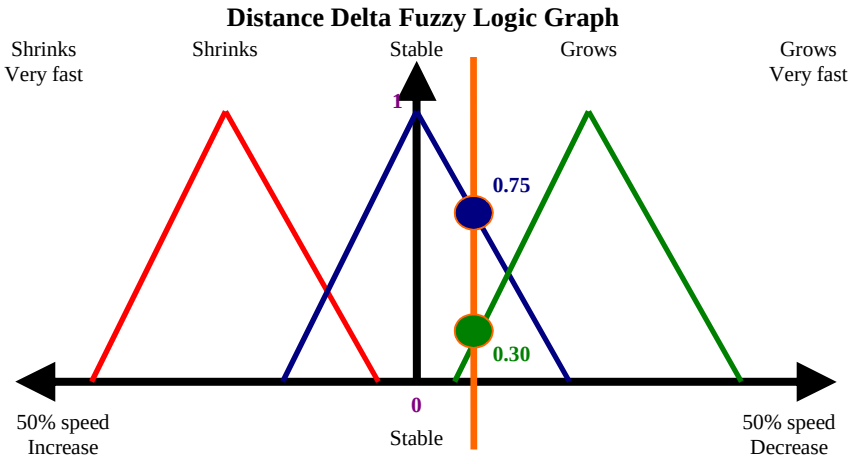


Figure 13 - The distance delta fuzzy logic graph for speed

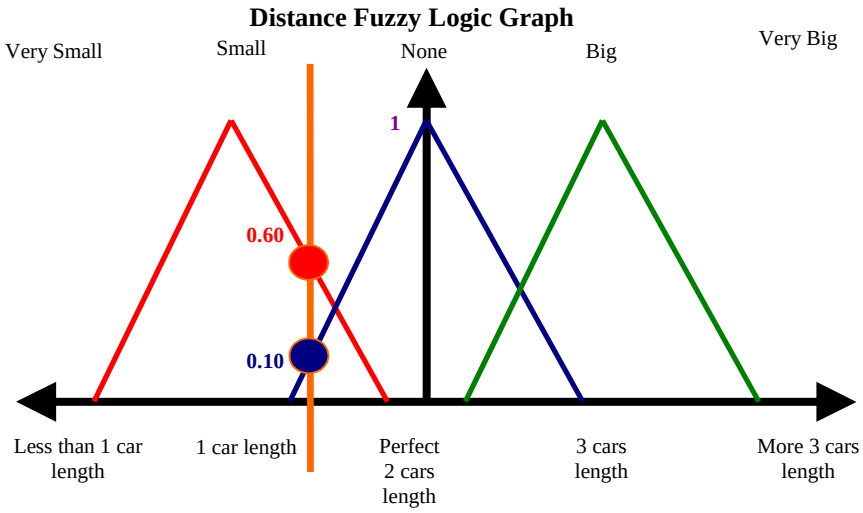


Figure 14 - The distance delta fuzzy logic graph for car distance

In Figure 13 and Figure 14 we can see the example. The graphs both have 3 colored lines that describe the evolution of the "truthness" of the upper level condition. For example, in Figure 13 we have 3 states respectively shrinks, stable and grows that correspond to the colors red, blue and green. To read the graph, one has to start from the "perfect" state where the statement "the distance delta {shrinks, stable, grows}" is true (i.e. is 1 on the yy axis). Then, when we move in the xx axis that statement lowers until it reaches 0 meaning that is a false statement.

When choosing an action to take such as brake, accelerate or maintain speed, one must evaluate the truthness of our set of rules. The orange line represents the current state of the system in both the distance and distance variations. This allows us to give values of truthness to each rule instead of just picking the first true one or none if all are false. To calculate these values one possible way it to based them on the lower value where both condition have "truth" larger than 0.

- If distance is small and distance delta grows, maintain speed
  (0.60 true) (0.30 true) — **0.30 "True"**
- If distance is small and distance delta stable, slow down
  (0.60 true) (0.75 true) — **0.60 "True"**
- If *distance* is perfect and *distance delta* grows, speed up
  (0.10 true) (0.30 true) — **0.10 "True"**
- If *distance* is perfect and *distance delta* stable, maintain speed
  (0.10 true) (0.75 true) — **0.10 "True"**

Figure 15 - The rule truthness

As we can see in Figure 15, by using this method for describing actions, the chosen rule would be the second rule as it is the "most true" one.

This model is based on the evaluation at each time, of the agent rules. Each rule corresponds to a particular behavior that has an associated agent action. As we can see in Figure 16, these behaviors are organized in sets, providing a parallel way of non conflicting behaviors to be activated at the same time.

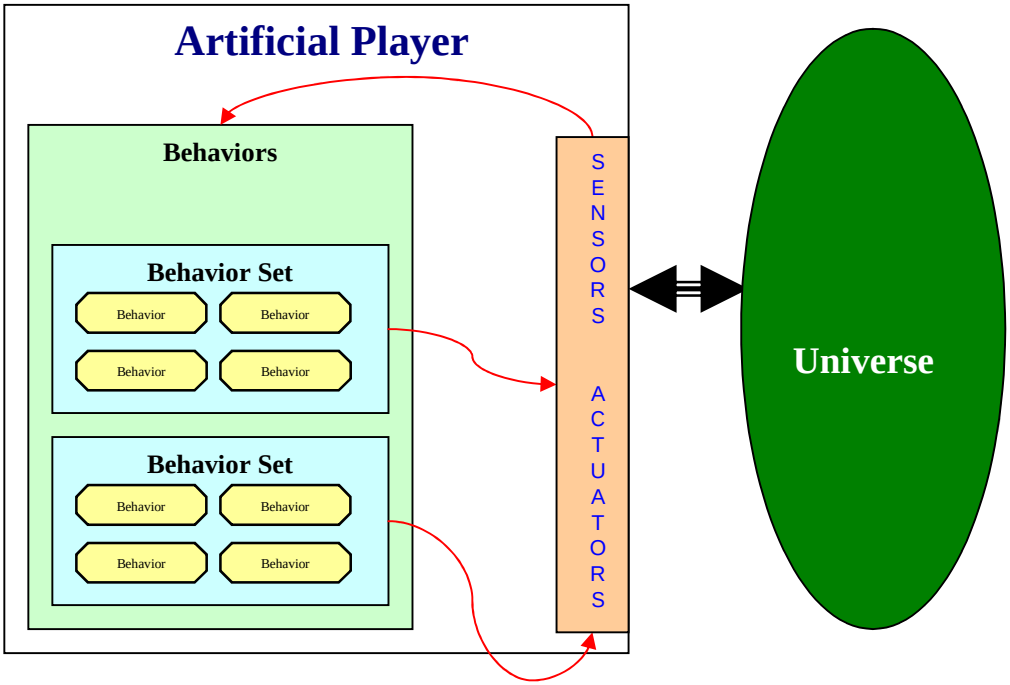


Figure 16 - The Fuzzy Logic Based Artificial Player

- Sensors and Actuators - The sensors consist in a set of a set of parameters that are given to the goals translating sensed information about the universe. The actuators consist in triggered actions of the agent over the universe.
- Behaviors - The set of behaviors the agent has. These are organized by sets that define the parallelization of multiple behaviors. Each behavior can have a set of rules that defines the rule activation properties.

The most distinct capability of this model is that it brings up the subsumption theory to a new level, where the rules are not only true or false but can have “middle” values. When designing content for a rule based architecture, one must always have in mind that at all times, at least one rule must evaluate to true meaning that the agent always knows what to do even if that means “no action”. In a system such as this, each rule has a real number attached and even multiple otherwise “true” rules can be ordered according to that value instead of the system builder having to think about that order.

Relevant Distinct Module Information

The system has an important feature that states that at each decision step, only one behavior at each behavior set can be triggered. This imposes that the behaviors that can be triggered simultaneously to be in other sets. However, each particular behavior can be in many sets. This allows the content designer to organize behavior in the best suited way for the system.

Decision trees and State machines based Architecture [14]

Global Overview Of The System

Amongst the most used techniques for deploying game AI players, have been the State machines[4] and decision trees. These are two formalisms that allow the definition of simple criteria for making a decision. In the previous architectures, several have used these models as base for their internal processes. One of the most useful combination of the two is called Hierarchical State Machines[14]. This is because it combines the notion of state, present in state machines, and the tree representation of behavior that allow a much easier representation of the artificial player for the content creator.

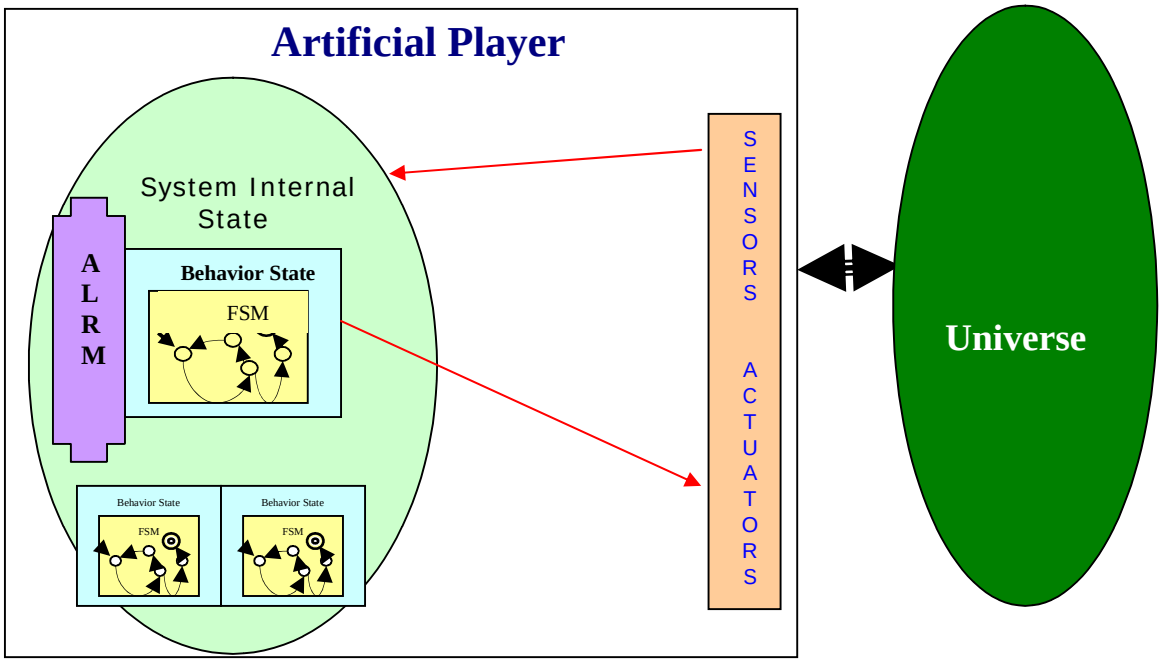


Figure 17 - Decision trees and State machines based Artificial Player

In Figure 17 , we can see the architecture components. A hierarchical FSM is just a state machine where each state is defined by other FSM. The system is built top down, starting by a base decision tree, that illustrates the complete base decision process, followed by the definition of a FSM of behaviors that accomplish each higher level state. It is also possible to define special entities denominated by “alarms” that trigger events whose treatment may force unconditional jumps to other higher level states.

The architecture allows the definition of rule based systems organized in hierarchies using human readable representations. It’s intended for tasks where the number of inputs is small and the notion of state is important.

Relevant Distinct Module Information

The most interesting property of this architecture, is in a system content creator view. Some AI tasks are sometimes trivial to think of, but difficult to implement. The way to start looking at those tasks, is through the definition of rules that state what should happen when you received some input from the world. However these rules can get human illegible very quickly and as such, a better way to represent them is required. The simplest way, is through the use of decision trees to represent decision points that are very human understandable. The model presented here, allows the definition of complex rule driven agents, using a human readable representation through the use of hierarchies of rules implicit in the FSM´s.

2.3. Alternative Techniques

This section is dedicated to alternative ways to build artificial players for games. Some could easily be fitted in the above sections, but it would not be right because their aim was to build artificial players for games, where the following methods are not. To start I will present Brooks Subsumption model. After, I will present the basics of scripting and a short summary of some of the techniques that due to space limitation did not appear in this overview.

Brooks' Subsumption Model [1]

Global Overview Of The System

This is one of the most well known reactive agent model. In a reactive approach the ultimate goal is to build complex behaviors from simpler ones. This architecture makes three assumptions that justify the core idea of the entire model:

- Intelligent Behavior can be obtained without the use of a specific knowledge representation as proposed by symbolic AI theories.
- Intelligent Behavior can the obtained without explicit reasoning mechanisms as proposed by symbolic AI theories.
- Intelligence is an emergent property of certain complex systems.

This assumptions function as safeguards, which allow us to refer to the behavior of agents with this model as intelligent and serve as building blocks for the subsumption model. Mr. Brooks suggests that an agent should be built incrementally, putting together smaller and simpler behaviors. The subsumption property emerges from the fact that several behaviors can be triggered simultaneously given a set of perceptions, and that those behaviors can be ordered has layers.

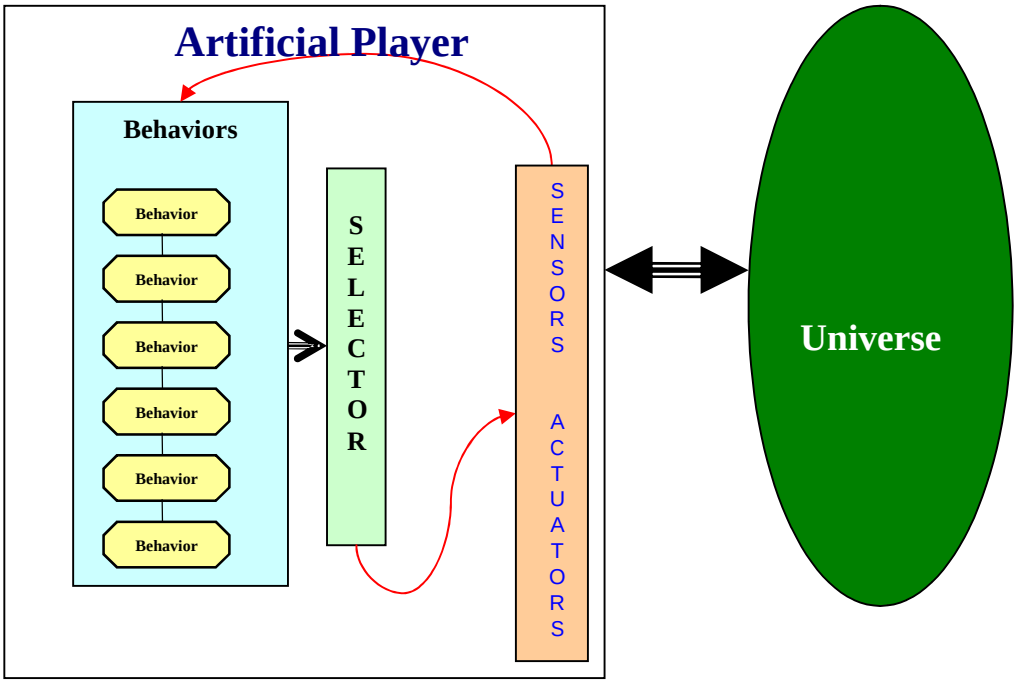
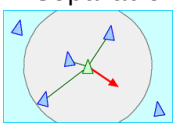


Figure 18 - Brooks' Subsumption model Artificial Player

Figure 18 shows the big picture of the agent model. As one can see the behaviors are defined in an hierarchical way being left up to the Selector the task of defining a rule that decides which actions to be taken given a set of triggered behaviors. This hierarchy also allows the interaction between behaviors making possible that the activation of one behavior, can allow or disallow the activation of other behaviors from lower layers. The Selector also has to make sure that no two, or more triggered behaviors are conflicting with each other.

This architecture allows the building of rich and complex behavior in agents. One of the strengths is that no set of complex behaviors are pre defined. That is to say that one can use a set of conditions and actions, which can even be contradicting between them, but that in a wider view, result in complex behavior. One of the most complete and yet simpler examples of the use of this architecture, is in the emulation of flock behavior. These are a perfect example, because they have a complex behavior, although only have a small set of rules in conflict between them. As we can see in the following figures, Craig Reynolds designed the Boids[9] behavior from three simple rules.

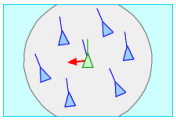
Separation Rule



The rule simply states that a Boid must avoid overcrowded areas. In Figure 19, the green Boid detects that it is near many boids and as such, tries to get away.

Figure 19 - Separation Rule

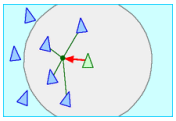
Alignment Rule



The rule states that a Boid must try to maintain its direction to the direction of the other Boids. In Figure 20, the green Boid calculates the direction of the nearest Boids, and concludes that we must change direction to be in the average value.

Figure 20 - Alignment Rule

Cohesion Rule



The rule states that a Boid must go to the most central position amongst the nearest Boids. In Figure 21 the green Boid tries to be on the centre of its neighbors.

Figure 21 - Cohesion Rule

As we can see, almost all rules have conflicting results. However, they can produce very realistic results. Nevertheless, the great strength that comes from this design is also one of the major drawbacks. When building agents to perform complex tasks, the set of rules can become very large and the interactions between them can become out of control very quickly. However, in complex and unknown worlds, where the agent must react quickly, this kind of architecture can be very useful.

Relevant Distinct Module Information

The notion of behavior in this perspective, can simply be a pair of <condition, action> that state that if the condition evaluates to True, then the action can be triggered. It is of course a very generic model that has been extended by many different people for several different proposes. This somewhat basic model remains one of the most popular amongst the current commercial games.

Scripting Based Architecture

Global Overview Of The System

Some of the techniques presented in this survey, already use scripting as tool, so why a section dedicated to the theme? Scripting is one of the most used techniques to deliver "intelligent" AI players in games. Game successes such as Farcry or Unreal make intensive use of scripting techniques, so in a survey about AI players, it has to be present.

One possible definition of what a script and a scripting language (from Wikipedia): [17]

> ***"Scripting languages*** *(commonly called* ***scripting programming languages*** *or* ***script languages****) are computer programming languages that are typically interpreted and can be typed directly from a keyboard. Thus,* ***scripts*** *are often distinguished from programs, because programs are converted permanently into binary executable files (i.e., zeros and ones) before they are run.* ***Scripts*** *remain in their original form and are interpreted command-by-command each time they are run"*

Other possible definition (from Free Range Network**)**

> *"****Scripting*** *is a form of computer program. But unlike traditional programming language, that manipulate the processes of the computer,* ***scripting*** *tends to involve a far smaller set of simple instructions. Many of these instructions will be related to accomplishing a specific purpose, such as controlling the process of connecting your computer to another computer via a modem. On the Internet* ***scripting languages*** *tend to be structured towards a specific task, and there are a number of different* ***scripting languages*** *in common use. (…)****Scripting systems*** *work dynamically to control a system, whereas making-up languages simply provide a constant, static scheme to control the display of information."*

Yet another (from Webopedia)[25]:

> *"Another term for macro or batch file, a* ***script*** *is a list of commands that can be executed without user interaction. A* ***script language*** *is a simple programming language with which you can write scripts."*

As is possible to see, the definitions are similar but if analyzed carefully, we reach the conclusion that they are not quite the same and thus, describe the entity differently.

In games these scripts can be used with many purposes such as triggering expected game events and control AI players minds.

What is of most interest in this review, is those who are intended to control players minds. In these kinds of languages, the more expressive the language, the more resources it needs and the more powerful may the action be. Once more, we have the balancing between computational resources, and decision power. This decision power originate from the possibility of the content creator, having tools to express complex conditions and functions that can describe effectively what the agent should do.

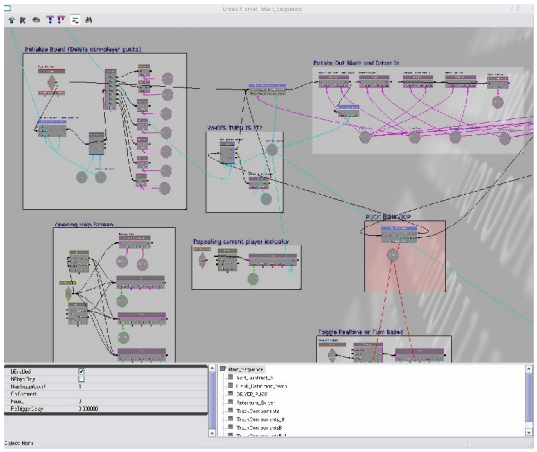


Figure 23 - The Unreal 3 Script GUI

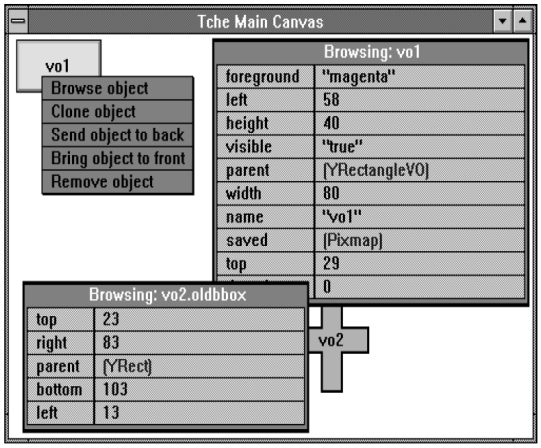


Figure 22 - A LUA Script GUI

Figure 22 shows an impressive example taken from the Unreal 3 framework, that shows that a scripting language can be integrated in a graphical representation allowing the content creator the possibility of delivering complex behaviors through the use of graphical symbols that are more human understandable. This graphics symbols are then evaluated and translated into the script language proving abstraction even over the script language being used. One of the most popular languages is LUA used as support for many systems. Figure 22 shows an example of a graphic system that builds interface objects by converting the graphical information into LUA scripts that when evaluated produce the expected result[26].

Other

Is impossible to present all possible ways of building artificial players for games. Even if space weren't a limitation, the evolution in the game AI area is so fast that any review wouldn't ever be completely current. As such, I will now briefly present two of also used methods for building AI that had to be removed from this review.

The first is the Neural Networks method[27]. A neural network is basically a simplified simulation of the human brain. We have a static or dynamic network of connected neurons trough a set of synapses. These synapses have weights that are used in the activation function of the neuron. The evolution of the system is made by a learning process by the adjustment of synapses weights. Connecting the sensors to the first level of neurons, and the final neurons to the actuators of a system, and we have the basic agent. Now it would be a matter of adjusting the topology of the network, and providing inputs and expected outputs to train the network. This of course, is no easy task and much research has, and is being made in the topic of neural net topology. One of the main problems in neural networks, is that it is very hard for a human to understand it, as it difficult to extrapolate a set of rules to express the current state of the network. The most famous game to use it was Black & White with very success. It was present in a form of a creature that the player could teach basic actions.

The second method is called Artificial Life or simply A-Life[23]. A-Life tries to simulate artificial life by giving each member of the simulation, unique traces that define it as an individual. Games such as SimCity, The Sims and Creatures, are well known example of the application of A-Life. On of the methods used to accomplish A-Life, is by the use of genetic algorithms. A Genetic algorithm[27] is a machine learning technique based on the natural evolution and selection. In simulations, each individual is given an unique piece of code that mimics DNA. The simulation world decides which individuals are viable and by sexual reproduction each individual may pass his set of DNA to its siblings. After many generations, only the more fitness ones prevail in the simulation.

3. Analysis

Before presenting the results, I must reiterate that the following classifications are not absolute in value. They only reflect the comparison between the present techniques. As such, they only reflect the comparison of the architectures amongst themselves and not to a specific domain problem. Ultimately, one could design a system where each module followed the presented techniques, however the goal of this review is not to present how they may be used as whole in a system but rather to identify their good and bad design options considering the needs of a generic artificial game player system.

As stated before each system will be analyzed according to three metrics:

1. Flexibility: How good is the system, when used in different applications? How does changing game type or mechanics affect the system?
2. Performance: How is the balance between capabilities and resources needs?
3. Decision Making: How good can the decisions be? How does it compare to what a human player would do?

The results of each metric will now be presented amongst all systems. In the end of the section a global table aggregating all results will help summarizing them.

Flexibility

Technique	Grade
MONAD Architecture	3
GRUE Architecture	5
Orwellian State Machine Architecture	3
Goal Tree Based Architecture	3
Goal Trees for Multi Task Agents Architecture	3
Fuzzy Logic model	4
Decision trees and State machines based Architecture	4
Brooks' Subsumption Model	2
Scripting	5

Table 1 - Flexibility analysis table

The main idea behind the MONAD architecture was to provide an easy way to design agent communities cooperating to achieve some goal. So, it presented a graph like representation of the behaviors each agent could have. To allow cooperation it was imposed that each player in order to cooperate to share the same graph of behaviors being this the first system limitation. In addition, one of the relations, the “follow” relation, imposed that once followed, never could the agent mind return to the origin behavior making this a major drawback when the agent is in

a highly dynamic world. The richness of expression originated from the representation is though weaken by the restrictions imposed in it. In comparison to other techniques that make use of graph like representation this is the only that make such strong limitations on its structure receiving as such, the 3 grade. The GRUE architecture in opposition doesn't have an explicit representation of the desires of the agent. It implements a BDI system using planners to generate both desires and intentions. The basic elements in the intentions are Teleo Reactive Programs that act as small reactive agents or pre-made plans, to achieve a goal. All this makes the system very flexible and although one can argue that it's no simple task to define planning elements to generate desires and intentions. In spite of that, with the proper tools and information about the game where the player is expected to play, the platform is sufficiently flexible allowing the content creator to build the desired player minds and as such, receiving a 5 grade. The Brooks Subsumption model tries to build complex behaviors using simpler ones. Those complex behaviors emerge from the interaction between the rules of simpler behaviors. However, making rules that allow the agent to perform well in complex games is a major task as it gets exponentially difficult to represent by rules the actions. Also, the emergence property is difficult to achieve and once achieved, can also be undesired in the game context as the content creator loses control over the expected behavior of the player in the game. For that, and because is very hard for a game content creator to debug the behavior, it receives the 2 grade. The Orwellian State Machines are based on the subsumption model, but focus on controlling the previously presented problems. Their objective is to provide an unique representation for all the player aspects in the game making them reusable. It builds finite state machines to determine possible courses of evolution and groups the behaviors in 3 basic entities, the collective, the dispatcher and the controller. Although this technique is more flexible from a game perspective than the basic subsumption model, it still remains a rule based system making harder to control when the task or world becomes more complex and as such receives the 3 grade. In a similar way, the Fuzzy Logic Model tries to make rules to represent the player actions. These rules aren't like the others because they consider degrees of truth making the process of evaluating rules a more complex and powerful one. As such, at any time the player must choose the action whose conditions have a higher degree of truth than the others, making this system very suited for games where is difficult to express exact certainly about conditions. However is still a rule based system with the same drawbacks as the others and as such, it only receives a 4 grade. Similar to the MONAD the Goal Tree Based Architecture also uses a graph-like representation of its goals. However, this representation isn't explicit like in the MONAD and the emphasis of the system is in building controllable and reusable goals. To that end all elements of the graph carries the same interface and the evolution of the system is marked by the existence of mechanisms to impose changes. These "hacks" allow the content creator a large degree of control over the player actions and responses in the world. In comparison to the other techniques it isn't such a positive aspect because it is arguable that the existence of these "hacks" makes the behaviors reusable as the game and tasks grow complex. As such, I believe the flexibility of this system is only comparable to the rule based systems and so, it receives a 3 grade. The second goal tree technique is the Goal Trees for Multi Task Agents Architecture. It distinguished itself for the ability to have an internal representation of the world. A belief is calculated and thought off information that exists for a purpose in the system. Because the player behaviors are based on beliefs, they become a more reusable option because of the new layer of information. However, the flexibility ends there, because the actions associated with each goal are hardcode making the adaptability of the agent, limited by the anticipation creativity of the content creator and as such, receiving only a 3 grade. The Decision Trees and State Machines architecture, makes use of finite state machines to determine the actions the player can make at any given time, and decision trees to decide which state machine to run. This makes this architecture, to have the representation potential of decision trees combined with the adaptability of a state machine. As a whole, they are a very complete solution not achieving grade 5 since they lack the ability to scale up gracefully as all state machines when the number of inputs and states grows. Finally, comes the Scripting method. It is difficult to evaluate scripting because it may be has different as its purpose. So, one must evaluate in accordance to common uses made, and against the other techniques presented.

A script can have an expressiveness equivalent to a programming language and some are even used as such. So, the flexibility grade has to be the best.

Performance

Technique	Grade
MONAD Architecture	5
GRUE Architecture	1
Orwellian State Machine Architecture	5
Goal Tree Based Architecture	3
Goal Trees for Multi Task Agents Architecture	4
Fuzzy Logic model	4
Decision trees and State machines based Architecture	5
Brooks' Subsumption Model	5
Scripting	*

Table 2 - Performance analysis table

In the MONAD architecture, at each cycle only a limited set of behaviors is evaluated because it only considers a subset of the total graph. The graph itself is static and the relations are lightweight because no complex decision takes place, only OR and AND checks are made because of the simple model implemented. From this, the only grade possible is the maximum, 5. In the other side of the spectrum, we have the GRUE architecture that must evaluate at each cycle, the current desires and intentions. This is a very resource intensive operation and if the game or the tasks of the player are complex, then the architecture may be easily considered inadequate. When compared with the other techniques this one receives the lowest grade, 1. In opposition, rule based systems require very little resources because they just need to evaluate conditions and trigger actions. In the Brooks Subsumption model, in most cases, only a subset of the rules are even evaluated at each cycle justifying the 5 grade. Similarly, The Orwellian State Machines, because they are basically a rule based system, also have a similar performance receiving the same 5 grade. The Fuzzy Logic technique, although it is also a rule based system, receives only a 4 because the evaluation of all rules is required at each cycle and because the process of rule evaluation is much heavier than the traditional. The Goal Tree based architecture had the goal of reusing and controlling the player goals. To that, it designed a script based interface trough which commands could be issued even during execution. This script interface enables the use of “shortcuts” during game to allow the content creator to have control over the artificial player actions forcing expected behaviors. Compared with the other techniques, this should in theory offer good performance. However the script interface requires that the agents actions depend on the current goal and in the “shortcuts” the content creator introduced. As such, this can be a heavy task and as so receives only a grade 3. Next we have the Goal Trees for Multi Task Agents, that is intended for complex environments were the player must have a low response time. To achieve that, it contains several goals, each with a set of rules. When the rule condition is satisfied, it triggers the associated action. This action can in turn spawn a new goal with a new set of rules. At each turn, the active goal is evaluated and builds up a tree of connected goals spawn by the rules that decide the player actions. The

existence of an explicit notion of system belief makes possible to abstract the complexity of the world to the system rule conditions. As such, because of these methods that allow the player to make decisions almost only with rule condition evaluation focusing in subsets of goals, it receives the 4 grade. Next we have the Decision Trees and State machine technique that bridges concepts of decision trees and state machines in one model. It has the performance of a state machine, and because the transitions between behavior states are triggered like events, it doesn't has a significant performance penalty receiving as such, a 5 grade. The last technique is the scripting technique. This technique is very hard to evaluate because it can be in some cases very efficient, and in others not, depending very hard of the scripting language implementation. A scripting language can be compiled, interpreted in addition to be able to represent many different aspects of an artificial player. Because of that, it doesn´t receives a grade in this metric.

Decision Making

Technique	Grade
MONAD Architecture	3
GRUE Architecture	5
Orwellian State Machine Architecture	2
Goal Tree Based Architecture	4
Goal Trees for Multi Task Agents Architecture	3
Fuzzy Logic model	3
Decision trees and State machines based Architecture	3
Brooks' Subsumption Model	3
Scripting	5

Table 3 - Decision Making analysis table

To evaluate the techniques in this metric, is important to analyze the degree of quality the artificial player can hope to achieve. Obviously, with a tremendous amount of work, probably even the most complex agent could be built using a rule based technique. In the limit, every technique here can be viewed as a rule based system. Off course, the abstractions allow a better representation which in course help the content creator to create more intelligent behaviors or ones that fit better in the game. So, all evaluations are made, having in mind the expected quality of decision one can expect from an artificial player using a specific technique, when implemented by a decent developer and content creator. The first technique is the MONAD that is based upon a graph representation of the desires. However the mechanism behind "how" desires manifest themselves as actions is hardcode in themselves. As such, the gain in using a graph becomes small if all behavior is modeled into hard coded desires. As such it receives only a 3 grade. The GRUE in opposition, allow a very powerful representation mechanism. The content creator can state the behavior at 3 levels of abstraction, the desire, the intention and the TRP layer. In the TRP level he states what the atomic actions are, and

how can the player achieve some basic goal. At the intention layer, he can state how the TRP will be combined to achieve some goal, and finally, at the desire level he can decide how do these goals are built, and which should the agent pursue first. Because of that richness it receives the 5 grade. Next we have the Brooks Subsumption technique. This architecture has the potential to mimic complex behaviors through the use of emergent behaviors. However, making those behaviors work is a complex task. Moreover, as the world becomes more dynamic and its interactions with the player increase, the task of modeling behaviors becomes evermore difficult. It is even arguable if those emergent behaviors are in fact desired and if in the game, they make any sense at all. In commercial computer games, this kind of systems is looked at with suspicious eyes because it is unpredictable and as such, can harm the gaming experience. Because all this, it receives only a 3 grade. Next comes the Orwellian State Machine technique. In terms of raw quality of decision, the upper limit is the model it is based upon, the subsumption model. However, because the OSM effort is in controlling the emergence capability, it loses some of its potential receiving only a 2 grade for that. The Fuzzy Logic technique is very hard to classify. To start, it has an immense potential in decision because it can represent uncertainty in its rules. This somewhat probabilistic approach to rule representation greatly increases the representation and provides means for higher level decision. However, it is still a rule based technique and as such, suffers from the inherent static nature. That is to say that adaptability to world is limited by the initial logic imprinted in its rules. When comparing with Goal Trees techniques for example, in the goals structures is possible to store information that will later change the decision. In rules, we don't have that possibility and the decision is always made by what we perceive immediately from the world. As such, I can not give any higher than a 3 grade. Entering the Goal Tree based techniques, we have the Goal Tree Based. As said earlier, it has a goal tree with each goal imprinted with shortcuts that can force the player to do actions that it would not do at the present goal. I believe that in terms of quality of decision, the existence of these shortcuts can be a very good thing because in most games, we don't want the artificial player to be autonomous, we want it to play the game against the player. One example of the importance of control, was in the Black & White game, were there was a "Creature" character that had a complex mind including a set of desires. One of these desires was the desire to "feed". Because the game world was populated with food for the creature, the content creators expected it to go hunt and feed from that food. Instead it tried to eat itself because it recognized itself as a food source. Sometimes the existence of "shortcuts" can focus the decision making in what matters, and instead of making a complete logical and accurate modeling of a problem, we end up with a simple "do that" that solves the problem. For that, it receives a 4 grade. The next goal tree technique is the Goal Trees for Multi Task Agents. I believe that this technique is similar in decision making, to a rule based system. In fact, I believe the gain from its tree like representation, doesn't add to the quality of its decision. As such, it receives only a 3 grade. By almost the same reasons, the technique of Decision Trees and State Machines receives the same 3 grade. Although it levels up the easiness to built a player for a game, it is still a rule based system with all the presented limitations. Finally, we have the scripting based technique. A script can be as expressive as a language can be. In fact, all the techniques presented here could be achieved by a scripting language and as such it must receive a 5 grade.

Now that all evaluations were presented, in we have a table with all results, plus an additional column with the main problem the technique tries to address.

	Flexibility	Performance	Decision Making	Focus on Problem
Brooks' Subsumption Architecture	2	5	3	Resource Effectiveness
MONAD Architecture	3	5	3	Resource Effectiveness
GRUE Architecture	5	1	5	Expressiveness
Orwellian State Machine Architecture	3	5	2	Resource Effectiveness
Goal Tree Based Architecture	3	3	4	Reusability
Fuzzy Logic Based Architecture	4	4	3	Resource Effectiveness
Goal Trees for Multi Task Agents Architecture	3	4	3	Reusability
Decision trees and State machines based Architecture	4	5	3	Resource Effectiveness
Scripting Based Architecture	5	*	5	Expressiveness

Table 4 - Global Metrics analysis table

A possible way to analyze these results, is to define a conceptual generic architecture, which in theory, could have the best qualities presented in each of the metrics.

To start, in the flexibility metric, the GRUE an Scripting are clear winners because they allow greater flexibility emerging from the fact that the content creator has the freedom to make the agent as adaptive has it can with little limitation from the system. In a generic architecture, the best maybe would be to have a goal description as GRUE, but described using a script language to allow a more flexible and easier way to the content creator to express himself and mimic complex behaviors into the agent mind.

In the performance metric, it appears the state machines based systems, are the clear winners. Having no planning abilities makes them the fitness to perform with the less resource consumption. In a generic architecture, it would be desired that the concept of agent state, be an explicit part of the whole with explicit relation to other parts of the systems. This could allow us to emerge a global FSM that implicitly translated the possible evolution of the system allowing a easier debug for the content creator, with a minimal performance impact by having to look only to the current state and possible follow ups.

In the last metric, the decision making, the winners are again the GRUE and script based architectures. Mainly due to the planning ability of the GRUE architecture and the expressiveness of a script based architecture. In a generic architecture, one of the possibilities could be to link this two characteristic by describing the planning elements in a script form.

Off course, the metrics aren't independent variables but they transpire important lessons about the conjunction of different systems and allow us to imagine how useful that conjunction would be when trying to build agents in them.

4. Conclusions

To conclude, I will revise the overall problems in making artificial players for games, and the analysis of how the presented architectures try to solve them.

The main problems in making artificial players are:

- How to make reusable artificial players (metric flexibility) - One can use the same data on several agents or on several different simulations by just changing the system sensors and actuators.
- How to make resource efficient artificial players (metric performance) - One can make scalable agents that are able to perform without maxing out the computational resources.
- How to make expressive artificial players (metric decision making) - The agent creator can express the agent goals and behaviors without system specific limitations.

Almost all of the presented architectures focus on one of these problems as presented in [14]. Is interesting however to note that results vary tremendously among all architectures without a clear pattern of methods arising. One could expect the FSM based architectures to always perform better, or the goal based to always have a greater decision making capability but that doesn't happen. There is no concrete solution to each of the problems, and it is not the method that solves each problem, but the conjunction of methods. In the previous section, the extrapolation of the Generic Architecture resulted in a goal based architecture with planning capabilities where the goals are implicit in a finite state machine with the agent mind loaded from a script. The pure analysis of the results produced that, and although it makes senses, we see that the systems that excel in one area, almost always are poor in the others. In a generic architecture, each peace of the system must be balanced to provide optimal results. As such I believe the more important conclusion to infer is that the key to a successful "generic" system is parameterization. If the content creator can use the same architecture for several types of agents, it must be allowed to tune it to the resources available. Ideally, one could tune all relevant aspects to each of the three main problems to better adjust the type and task of artificial player.

5. References

[1] Michael Wooldridge, An Introduction to MultiAgent Systems, John Wiley & Sons Ltd, 2002, paperback, 366 pages, ISBN 0-471-49691-X.

[2] Vu, T., Go, J., Kaminka, G., Veloso, M., Browning, B. "MONAD: A Flexible Architecture For Multi-Agent Control", Autonomous Agents and Multi Agent Systems 2003

[3] Gordon, E., and Logan, B., GRUE: A Goal Processing Architecture for Game Agents, Computer Science Technical Report No. NOTTCS-WP-2003-1, School of Computer Science and Information Technology, University of Nottingham, 2003.

[4] S. Rabin 2001. "Implementing a State Machine Language" AI Game Programming Wisdom (ed. S. Rabin), Charles River Media, Hingham, MA, pp. 314–3.

[5] Sweetser, Penelope, Current AI in games: A review , School of ITEE, University of Queensland, 2003.

[6] Michael Freed, Travis Bear,and others, Towards more human-like computer opponents, NASA Ames Research Center, 2000.

[7] Asda Sloman, "What sort of architecture is required for a humanlike agent?" in Wooldridge, M. and Rao, A. (ed.), Foundations of Rational Agency, Portland Oregon, 1999.

[8] Russel, Stuart J., and Peter Norvig. Artificial Intelligence: A Modern Approach. Englewood Cliffs, NJ: Prentice Hall, 1994.

[9] Internet site: http://www.red3d.com/cwr/boids/

[10] Nilsson, Nils J., Teleo-Reactive Programs for Agent Control, Journal of Artificial Intelligence Research, 1994.
[11] Borovikov, Igor. Orwellian State Machines. In: Rabin, S., AI GameProgramming Wisdom 3, Charles River Media , 2006.
[12] Johnson, Geraint. Goal Trees. In: Rabin, S., AI GameProgramming Wisdom 3, Charles River Media , 2006.
[13] Gordon, Elisabeth. A Goal-Based Multitasking Agent Architecture. In: Rabin, S., AI Game Programming Wisdom 3, Charles River Media , 2006.
[14] Ian Millington, Artificial Intelligence for Games, Morgan Kaufmann Publishers , 2006.
[15] P. Clements et al., Documenting Software Architectures:Views and Beyond, Addison-Wesley, 2002.
[16] Clements, R. Kazman, and M. Klein, Evaluating Software Architecture, Addison-Wesley, 2002.
[17] Wikipedia Site: http://en.wikipedia.org/wiki/Scripting_language
[18] McCuskey, Mason, "Fuzzy Logic for Video Games", GameProgramming Gems, Charles River Media, 2000.
[19] Alexandre, T. "An optimized fuzzy logic architecture for decision-making". In: Rabin, S., editor, AI Game Programming Wisdom. Charles River Media, 2002
[20] Breazeal, C. (2003). "Emotion and sociable humanoid robots". International Journal of Human Computer Interaction.
[21] Internet site: http://www.nasa.gov/mission_pages/station/structure/elements/mss.html
[22] J. Kurien, P. Nayak, and B. C.Williams, "Model-based autonomy for robust Mars operations," presented at the 1st Int. Conf. Mars Soc., Boulder, CO, 1998.
[23] Internet site: http://en.wikipedia.org/wiki/Artificial_life
[24] Internet site: http://www.fraw.org.uk/library/005/gn-irt/glossary.html#scripting_language
[25] Internet site:http://www.webopedia.com/TERM/S/script.html
[26] Carregal, André. "Tche - A visual Environment for the Lua language",1995
[27] Mitchell, Tom M., Machine Learning, Boston: WBC/McGraw-Hill, 1997
[28] Internet site: http://www.gamasutra.com/php-bin/news_index.php?story=12574