

# Design Goals for Autonomous Synthetic Characters

John Laird  
Artificial Intelligence Lab  
University of Michigan  
1101 Beal Ave.  
Ann Arbor, MI 48109-2110  
laird@umich.edu

## 1. Introduction

Synthetic characters in current computer games have limited reasoning abilities and fall short of human players in many dimensions, often struggling to achieve even a modicum of artificial intelligence. However, progress is being made. Characters in recent games, such as Half-Life, have moved to include limited forms of cooperation, communication, and reasoning. Our work in developing a bot for Quake using the Soar architecture tries to push even further to human-like behavior. However, as of yet there is not a clear set of evaluation criteria for comparing various approaches. In this paper I attempt to explore the design goals inherent to building synthetic autonomous characters for computer games. Many of my observations are obvious and have been made before, but bringing them together in a structured form should make it easier to analyze future systems. Some of the observations may be wrong given my modest experience in building AI systems for computer games. So at the least I hope that this paper stimulates discusses and refinement of these ideas.

In analyzing a synthetic character, its structure can be decomposed into three components, the underlying architecture, the knowledge or program that generates behavior within the architecture, and the interface of the architecture with the game environment. The architecture provides the set of fixed mechanisms, such as control structures, memories, and primitive actions, which together define a language for expressing knowledge that controls the character. Different architectures support different computational frameworks, such as scripting languages, finite-state machines, neural nets, genetic algorithms, and rule-based systems. Usually the architecture is designed to support the encoding of knowledge for many different tasks, not just a specific character, although there may be certain task-specific architectural primitives that are included to improve efficiency or expressibility. However, for games it is not uncommon for the character architecture to be specific to a class of characters, where the only knowledge is a set of parameters, such as strength, endurance, aggressiveness, that are set by the game designer. Architectures have their own “complexity profile,” being able to perform some calculations directly and very efficiently, while others require interpretation and significantly more computational resources. Architectures have limits to the knowledge they can encode, so that certain relations or computations can be impossible to express.

My interest is mainly in the relationship between different classes of architectures and the performance they support as well as their associated development and maintenance costs. This relationship is necessarily indirect because it is the combination of architecture and knowledge (program) that produces behavior, determines performance, development times, and so on. However, if I look only at the combination of architecture and knowledge, it is difficult to extract any general lessons that can be carried over to other problems, tasks, characters, ...

In the remainder of this paper I attempt to explicate in more details the dimensions that are important for evaluating both the performance of the characters and the development cycle.

## 2. Performance

The capabilities and performance required of an autonomous character can be broken down into two subcategories: behavioral capabilities and resource requirements. The first is essentially the space of behaviors that the character can produce, while the second refers to the computational resources (CPU and memory) required to generate that behavior.

### 2.1 Behavioral Capabilities

The behavioral capabilities of an agent cover its interaction with the world and the other characters. For example, in a fighting games behavioral capabilities include the set of possible attacks and defenses that can be used, the game conditions in which they are used, and how they are organized into sequences of behavior. Thus, the variety and complexity of the agent's behavioral capabilities determine the gameplay experience for the user. These capabilities are determined both by the underlying architecture, which determines the total space of possible behaviors, and the knowledge encoded for a character, which determines the specific behaviors that the character will use and how they combine.

Before attempting to create a comprehensive list of desirable behaviors, it is worthwhile asking what is our goal as developers for these characters. The most obvious and vague goal is to provide great gameplay. Some developers and reviewers have suggested that the best games are those that support multi-player interactions among human players. This leads to the conclusion that the Holy Grail for character designers should be to produce characters whose behavior is indistinguishable from human behavior. This is not a foregone conclusion because part of the attraction of human multi-player games is the broader social milieu in which the game takes place – you are playing against friends. However given the popularity of anonymous networked games, it is clear that people enjoy human-to-human games without the background social aspects. Even though my group is attempting to create human-like characters, for many games, non-human synthetic characters are more important than synthetic human characters, where “life-like” or “believable” behavior is more important than human-like behavior (see Bryan Loyall's thesis, which includes an analysis of the requirements for believability and the illusion of life).

To complicate the discussion further, many single player games use synthetic characters to further the story, or to play roles that must be modulated by the overall game play – the characters are more like actors in a real-time play with a director giving them instructions as the action unfolds. Pure autonomy takes second place to performing specific, scripted behaviors at specific times or in specific circumstances. This is just the tip of the iceberg in terms of customizing a synthetic character for a specific game or genre.

In this section I will concentrate on the characteristics of behaviors that seem to be generally useful for many games, with an emphasis on human-like behavior. Of course the point is for the observed behavior to be human-like and it really is immaterial how the underlying implementation produces that behavior. In games, the most profitable approach is usually to find computationally inexpensive approximations, or as I like to say, “Cheat without getting caught.” However, if we look out a few years, I expect that the approximations will have to get more and more detailed to produce more robust and believable behavior. Each of the following capabilities defines a dimension of variability for creating synthetic characters and research into any these characteristics would be useful. An important reference for modeling human behavior is “Modeling Human and Organizational Behavior: Applications to Military Simulations,” National Research Council, 1998. Unfortunately this comes off as a laundry list of capabilities. I need to add more structure to this list.

1. Human sensing. The ability of a character to sense things in the environment should be similar to the abilities of a human in the game. The character should not have superhuman abilities, such as seeing through walls (unless the character is Superman) and should not have subhuman abilities (being unable to hear someone running up from behind in a quiet room). Matching human sensing goes a

long way to providing human-like behavior, with many of the complaints about characters in current games coming from super or subhuman character sensing. Superhuman sensing is often used to make up for weaknesses in the reasoning abilities of a character. Unfortunately, realistic models of sensing can be very difficult and computationally expensive to implement. For example, a player should be able to hide in a dark area of a game, so the sensing model must be able to detect light levels. However, if the player is in a dark area but is in front of a bright hallway, the player will be backlit and an outline of the player will be clearly visible. Similarly, it can be very difficult for a human player to identify whether another player is friend or foe at large distances and few if any synthetic characters have trouble with this. An underlying reason for the difficulty is that the internal game data structures are organized for fast display of a scene and rely on the human visual system to pick out relevant objects and their relations. It is infeasible to simulate the generation and processing of an image for a synthetic character, so that character sensing must rely on models applied to internal symbolic information.

2. Human actions. A character should be able to perform actions in the environment that correspond roughly to what a human could do in a similar environment. For example, the Quake II DLL allows for instantaneous turning which is not possible for a human player. As with all aspects we are discussing, this is tempered by the overall gameplay and the roles defined for the characters by the game designer. Just because there is water, it is not necessary for a synthetic character to be able to drink the water. Conversely, supernatural actions are appropriate in many fantasy games.
3. Human-level reaction times. Humans do not respond instantaneously to changes in their environment, nor do they take arbitrarily long to respond (unless distracted). There are good psychological models of human reaction times in sensing and action in human-computer interaction. For example, it takes approximately 250 milliseconds to respond to a change on the screen.
4. Spatial reasoning. Many games are only 2D Cartesian grids where spatial reasoning is straightforward. However, the topology of a game from the standpoint of a character can be complex as obstacle, shortcuts, and a third dimension are added. Usually, the internal game representation can be used by the character so that no additional model needs to be created internal to the character; however, 2D & 3D spatial reasoning routines often need to be developed that compute relations such as line-of-sight or shortest path. Often synthetic characters can bypass the need for spatial reasoning by having important points (nodes) pre-computed and included in the environmental map. For example, most level-designers for games like Quake annotate the level with nodes for where the characters should travel. This simplifies the development of the characters, but also makes them less general, as they are unable to easily respond to changes in the world.
5. Memory. Most current characters can be described as, "out of sight, out of mind." As soon as an enemy is out of sensor range, it is forgotten. If a discarded weapon is by-passed during a firefight, it is not remembered for the future. This might work for one month-olds, but it leads to very unhuman-like behavior. Thus, complex characters need to maintain memories of the world and have a model of how the world changes over time. In Quake II, there are spawn points for weapons, health, and armor. Whenever an item is picked up, a new one will be created automatically in 30 seconds. Similarly, whenever a weapon is dropped (such as when a player is killed) it will disappear in 30 seconds if it is not picked up. Just as humans quickly learn these aspects of the game and use them in their strategies, so must human-like synthetic characters.
6. Common Sense Reasoning. This is the most dreaded of all classes of reasoning in AI because it is completely ill-defined. Using the knowledge about memory described above could be classified as common-sense reasoning, although it is sort of Quake common-sense reasoning. Other types of common-sense reasoning include knowing that doors can be opened and closed, locks have keys, and that if you jump off a ledge you will fall. The only viable approach seems to be to determine what knowledge is necessary for the tactics you wish to encode in your character and then encode the necessary common-sense knowledge to support those tactics. This is insufficient if you wish to develop characters that develop their own tactics. A related capability is broad, but shallow behavior.

7. Goals. Characters must have a purpose, some goal they are trying to achieve. Often there will be multiple goals, and the character must decide which one to pursue in a given situation, or better yet, how to select actions to pursue multiple goals. The goals should drive the actions of the character.
8. Tactics. A character should have a variety of tactics or methods that can be applied to achieving goals.
9. Planning. Planning provides the ability to try out actions internal, discover consequences, avoid death and destruction. Many of the benefits of planning can be compiled into a knowledge base when you know the world and goals beforehand, but it can simplify the construction of a character if it can plan on its own.
10. Communication and coordination. For many games, the underlying goals for the characters require that they cooperate with other characters, and possibly the human player. The characters need to communicate in realistic ways and coordinate their behavior as would humans – well, but not necessarily perfectly when there is uncertainty or limited communication.
11. Learning. For most games, learning can be avoided. It is only an issue for characters that have prolonged interactions with the human players. Even in those cases, what is more important than learning is the appearance of learning. Learning is sometimes difficult to implement and can lead to unexpected and undesirable behavior unless carefully controlled. Instead, it is safer and usually easier to implement different levels of behavior available for a character to use as it experiences the world. However, very simple learning is often important to have the characters adapt to the player, such as gathering data on the types of tactics the human uses and adjusting the synthetic characters' responses accordingly.
12. Unpredictable behavior. As with all of the capabilities covered in this section, non-determinism itself is less important than the illusion of unpredictability. It also depends on context. When there is only one right thing to do, then being predictable is fine. However, when there are multiple actions that can be performed, all with similar value, a random selection can eliminate predictability. However, if a character has a sufficiently broad and rich set of fine-grained responses, its behavior may be very difficult to predict. One thing to keep in mind is that although much of human behavior appears unpredictable, when it is analyzed carefully, there are significant regularities in human responses to specific situations.
13. Personality. Personality can be thought of as what distinguishes one character from another above and beyond gross characteristics such as physical build and general mental capability. It establishes a “style” that influences many activities. What underlies the style? One theory (Barrick and Mount, 1991) is that there are five factors: openness, conscientiousness, extroversion, agreeableness, and neuroticism that determine overall personality. However, mapping this onto the behavior of a synthetic character is clearly an art.
14. Emotions. Unfortunately, there are no comprehensive computational models of how emotions impact behavior. What are the triggers for anger? How does anger impact other behaviors? However, as with personality, the expression and influence of emotion may be critical to creating the illusion of human behavior.
15. Physiological Stressors. In addition to emotions, there are other physiological changes that happen to people that in turn impact their behavior. In computer games, there is often a collective component of health, although the level of health rarely changes the behavior of a character (synthetic or human) except when it goes to 0 (and the player dies). Other stressors include fatigue (which negatively impacts reaction time and error rate), heat, chemicals, radiation, ...

## 2.2 Computational Resources

Clearly less is better; although with the continued exponential speed improvements of computers and decreases in memory costs, there are more resources becoming available for AIs in computer games.

1. CPU: As the underlying architecture and knowledge increases in complexity, the amount of processing required to produce behavior invariably increases. At one extreme, behavior can be produced by the execution of raw C code or via indexing into a lookup table to find a predefined

response. At the other extreme, behavior generation might require planning across multiple entities that must be spread across multiple cycles of the game, where a cycle, or frame, consists of an update of the drawing system (20-60 frames/sec.). As the number of entities being simulated increases, the amount of CPU available for an individual entity decreases, although every entity usually will not be activated every cycle. Thus, games that have large numbers of entities are forced to have very simple behaviors at the individual entity level. However, overall performance can be improved by creating groups of entities, such as military platoons or companies, where more resources can be applied to planning and executing the behavior of the groups. AI systems can expect between 5-10% of the CPU, although this depends on the type of game. Turn-based strategy games can use significantly more than real-time shooters.

2. Memory: There are three types of memory costs.
  - a) Runtime-Architecture: this is the amount of storage required to hold the code that executes the AI architecture. For purely compiled languages, this could be zero – there is no runtime interpretation. The runtime architecture is a fixed cost independent of amount of knowledge and runtime state. It may or may not be shared across multiple entities. In Soar, for example, the cost of the architecture is approximately 1Mbyte, which can be partially shared across multiple entities.
  - b) Long-term behavioral knowledge/program: This is the storage required to hold the behavioral knowledge, tactics, etc. At one extreme it might be only a few parameters that determine special characteristics of each entity, such as its distance before sensing, type of attack, etc. In this case, the code that interprets these parameters can be thought of as the architecture because it is shared across all entities. For In Soar the cost is approximately 1Kbyte/rule. It is not possible to share this across active entities.
  - c) Runtime state: This is the amount of storage required to hold the internal, dynamic data structures used by the behavior knowledge. This includes sensory knowledge and internal state, such as map information. At one extreme this might include only the entity's current position, its current health, and strength. At the other extreme, it might include a set of plans being executed, goals being attempted, and internal map data.

### 3. Development

The viability of a given approach to synthetic character design rests not only with its performance capabilities, but also with the costs associated with its development. If a given approach is too hard or too expensive to use, then it won't be used. I'm more in the dark on this dimension, and less seems to be written on it. There are many anecdotal stories of how AI technology was attempted for a specific game but was later dumped because of development problems. What I have learned is that a critical part of the development process for many games is the separation of content development by game designers and underlying game engines by game programmers. This distinction is essentially the knowledge/architecture distinction I made earlier. The importance of this distinction is that the architecture must support a level of behavior specification that is easy enough for game designers to use. Many game designers are not programmers, so that the language they use must be easy to learn and use. The need for straightforward specification of behavior will often be at odds with the ability to specify complex behavior. This can be overcome in part by having complex behaviors as primitives available for the game designer to select or merely provide some parameters for. For example, the game programmer can create very complex path finding code that the game designer need know nothing about except that it works. The game designer need only specify the situations in which it is used. This approach can lead to a three-layer system, where there is a set of underlying game architectures that are used by the game programmer to create a set of behaviors available to the game designer. The game program creates a language that the game designer then uses to specify behaviors built out of the complex primitives.