Towards a Game Agent

Christoph Niederberger, Markus H. Gross

Computer Science Department
ETH Zurich, Switzerland
e-mail: {niederberger, grossm}@inf.ethz.ch
http://graphics.ethz.ch
# Table of Contents

1 Introduction .................................................. 5
   1.1 Purpose of this report .................................. 5
   1.2 Introduction .......................................... 5
   1.3 Overview ............................................. 6

2 Artificial Life .................................................. 7
   2.1 ALife - Definition .................................... 7
   2.2 ALife Fields .......................................... 8
   2.3 ALife History ....................................... 8

3 Artificial Intelligence ......................................... 11
   3.1 Overview ............................................. 11
   3.2 Adaptation and Learning .............................. 12
      3.2.1 Genetic and Evolutionary Algorithms ......... 12
      3.2.2 Simulated Annealing ............................ 13
      3.2.3 Neural Networks / Learning from Examples ... 13
      3.2.4 Reinforcement Learning ......................... 14
      3.2.5 Support Vector Machines ....................... 15
      3.2.6 Other Learning Techniques ...................... 15
   3.3 Problem Solving ....................................... 15
      3.3.1 Problem Formulation ........................... 16
      3.3.2 Acting ......................................... 16
      3.3.3 Searching .................................... 17
      3.3.4 Comparison of Search Strategies ............ 17
   3.4 Knowledge ............................................. 18
      3.4.1 Knowledge Representation ...................... 19
      3.4.2 Building a Knowledge Base .................... 19
   3.5 Planning ................................................ 20
   3.6 Uncertain Knowledge ................................ 22
   3.7 Communication ...................................... 23

4 Games and Entertainment ...................................... 25
   4.1 Games ................................................. 25
      4.1.1 Classical Games ................................ 26
      4.1.2 Pure Computer Games ......................... 26
   4.2 Game Characters ...................................... 28
   4.3 Game Development .................................. 29
   4.4 Cinematography ...................................... 29

5 Agents .......................................................... 31
   5.1 Definition ............................................. 31
   5.2 Environments .......................................... 32
   5.3 Autonomous Agents ................................ 33
      5.3.1 Simple Reactive Agents ....................... 34
      5.3.2 Goal-based Agents ............................. 35
      5.3.3 Utility-based Agents ......................... 36
   5.4 Game Agents .......................................... 36

6 Behavior Modeling ............................................. 39
1

Introduction

1.1 Purpose of this report

The objective of this report is to give the reader a survey on state-of-the-art techniques and academic research in the field of artificial life where the simulation of complex and emergent behavior is the central point of investigation. Furthermore, games, artificial intelligence, and the concept of agents are focussed to give a classification and comparison of modern techniques used to simulate and/or animate creatures and other life-like forms.

This report is not meant to be fully complete since some topics might remain unconsidered.

1.2 Introduction

Since the invention of computers, man has always tried to form any kind of “intelligence” within them. The field of artificial intelligence (AI) has produced several very interesting techniques such as classification, learning, problem solving, controlling of physical movements, searching in huge information spaces, modeling, and planning.

Artificial Life (ALife) is a very new field of research. It involves many different academic disciplines such as artificial intelligence, biology, genetics, sociology, and many more. It’s goal is to recreate behavioral characteristics of living systems from observations of natural life. Many research results from AI are used within ALife.

With today’s computer graphics it is also possible to render nearly real-world looking scenes which gives the entertainment industry new possibilities to produce entertaining medias such as movies and games. The former is a non-interactive form where the needs on computation time are negligible but the movements and graphical requirements are considered as the main problem and should be as perfect as possible. The latter is an interactive form of entertainment where the player(s) decisions control the progress of the plot. Because of the interactivity, computation time is the major hurdle where one has to find a balance between realism and reactivity.

With ALife and computer graphics together, it seems possible to create new types of agents, presented in a fashionable way using three dimensional computer graphics, which are autonomous and have intelligent characteristics. This process is very challenging because these agents should spawn characteristics not only in movement but also in cognition, reasoning, expression, emotion, motivation, learning, social behavior, and in many other areas.
1.3 Overview

In chapter 2, we will have a look at artificial life, its definition, and its applications. Then, in chapter 3, we will focus on artificial intelligence as a part of many artificial life applications. We will discuss adaptation and learning theories, consider problem solving, knowledge representation, planning with or without uncertainty, and communication.

Then, Chapter 4 overviews games as the goal platform, and especially the characters therein. Additionally, a short look on cinematography will conclude the chapter.

With the game characters in mind, we consider agents as the fundamental concept to build manifold characters and creatures. Therefore, chapter 5 first tries to find a definition of agents and their environment before presenting simple agent architectures which lead to a specific game agent discussion.

An extensive review of current work on behavior and character modeling can be found in chapter 6, which also covers cognitive modeling as a very recent research result. The following chapter deals with motion synthesis in general, which is discussed with respect to the presentation of behaviors in terms of expressive animation in chapter 8.

Finally, the last chapter discusses agent architectures with respect to implementation specific issues. Their requirements, resource consumption, and the three main parts of an agent are covered: Perception, inference, and acting.
Artificial Life

Artificial life (ALife) is a new and interesting field of research. It involves many academic disciplines such as artificial intelligence (AI), biology, genetics, psychology, and social sciences. ALife studies natural life by attempting to recreate behavioral characteristics of living systems.

2.1 ALife - Definition

Artificial life lacks a proper definition, because it covers many different disciplines and every researcher sees the field from her own perspective.

C.G. Langton provides several approaches to a definition: In [62] his definition remains short:

“The study of man-made systems that exhibit behaviors characteristic of natural living systems.”

He extends this definition in [63] to a futuristic outlook:

“Artificial Life is a field of study devoted to understanding life by attempting to abstract the fundamental dynamical principles underlying biological phenomena, and recreating these dynamics in other physical media — such as computers — making them accessible to new kinds of experimental manipulation and testing. (...) In addition to providing new ways to study the biological phenomena associated with life here on earth, life-as-we-know-it, Artificial Life allows us to extend our studies to the larger domain of "bio-logic" of possible life, life-as-it-could-be ...”

Others such as T.S. Ray [92] look from a different viewpoint and suggest a bottom-up approach, rather than top-down:

“Artificial Life is the enterprise of understanding biology by constructing biological phenomena out of artificial components, rather than breaking natural life forms down into their component parts. It is the synthetic rather than the reductionist approach.“

ALife is often described as an attempt to understand high-level behavior from simple low-level rules. Therefore, the goal of ALife would then be to detect these rules from observations and to reproduce the behavior in a machine such as a computer.
Again, C.G. Langton describes the goals of ALife:

“ALife views life as a property of the organization of matter, rather than a property of the matter which is so organized. Whereas biology has largely concerned itself with the material basis of life, ALife is concerned with the formal basis of life. (...) It starts at the bottom, viewing an organism as a large population of simple machines, and works upwards synthetically from there — constructing large aggregates of simple, rule-governed objects which interact with one another non-linearly in the support of life-like, global dynamics. The ‘key’ concept in ALife is emergent behavior.”

He goes on and states:

“Of course, the principle assumption made in ALife is that the ‘logical form’ of an organism can be separated from its material basis of construction, and that ‘aliveness’ will be found to be a property of the former, not of the latter.”

With that, one could imagine that if the basic principles of a living organism can be uncovered, then the material used to realize life is irrelevant.

2.2 ALife Fields

The theories of ALife can be used to develop applications in robotics, spacecraft, medicine, nano-technology, industrial fabrication and assembly, and many others. Generally, ALife can be split into two main areas: Autonomous agents and ALife simulators. Autonomous agents are software programs designed to pursue goals either individually or in teams, and can do so interactively or without user control. ALife simulators are generally smaller programs, and they are written in order to extend research in ALife.

J. Greenbank lists in [38] different applications of ALife in our everyday life:

- Virtual Reality (VR)
- Robotics
- TeleRobotics
- Traffic Control
- Education
- Military
- Information Agents
- Entertainment

In this survey, we will focus on entertainment7 applications such as games or movies. In games, ALife and AI are being incorporated increasingly and will probably become an important part of most games. Also, the special effects in movies rely more and more on ALife theories. For example, Craig Reynolds’ famous Boids [94] have been used in many movies to generate creature movement in groups.

2.3 ALife History

We will provide here a short overview on ALife techniques and on previous work.

A Cellular Automaton is an array of cells, each of which can have any one of a finite number of states. The state can be a variable, property or other information. By receiving input from the neighboring cells, a cell uses its own local set of rules to determine its next state and reaction.
The state of all cells are updated simultaneously, and the state of the entire array advances in discrete timesteps. John von Neumann’s cellular automata (around 1950) had the goal to achieve self-reproduction. The problem with machine reproduction is that the universal constructor is a mindless robot. It has to be told very explicitly what to do. If such a machine were given a description of an alleged self-reproducing machine, this constructor needs to understand that it is supposed to be reproducing the machine, including its description. Von Neumann used arrays of two hundred thousand cells to represent an organism. Each cell had 29 possible states, and different combinations of these states defined the behavior of the system. The effect of the local behaviors caused a global behavior to emerge: the self-reproducing structure interacted with neighboring cells and changed some of their states. Eventually, the organism made a duplicate of its main body, resulting in two identical creatures, both capable of self-reproduction.

*L-Systems*, named after a botanist Astrid Lindenmayer, were developed in the 1960s. They try to produce fractal patterns as they can be seen in the nature. To obtain the desired self-similarity, a set of simple rules are executed repetitively. A L-System mainly consists of variables, constants, rules, and a start sequence.

Conway used a similar approach as Neumann in 1970 for his famous *Game of Life*. He suspected that cellular automata with less cells and less states could also emerge computing capabilities, despite their simpler structure. The key to this simplicity would be the rules dictating survival, birth, and death. The number of possible states was reduced to two: alive or dead.

As mentioned before, Craig Reynolds proposed a individual-based approach to simulate the aggregate motion of a flock of birds, a herd of land animals, or a school of fish [94]. As in nature, every entity decides individually its own course depending on its perception of the dynamic environment and a set of behaviors programmed into it. Then, the aggregate motion is the result of the dense interaction of relatively simple behaviors. The results are very impressive and have been used many times in movies or games.

Karl Sims et al. [103][115] evolved their virtual creatures, which are able to move and behave in three-dimensional dynamical environments. The morphology and the neural system are both generated automatically using genetic algorithms (see section 3.2.1). With different fitness functions, they were able to direct the evolution towards specific behaviors such as swimming, walking, jumping, and following. Chris Leger [68] uses a similar approach in his PhD to produce robot configurations for specific tasks.
3 Artificial Intelligence

3.1 Overview

In Artificial Intelligence (AI) researchers try to construct intelligent machines whether or not these operate like humans do [30]. Among the areas in computer science, AI probably covers the largest number of other fields namely psychology, economics, physics, and many others. It is, thus, very difficult to define. Russel and Norvig classifies different definitions for AI by distinguishing the behavior of the system (acting vs. thinking) and the way it behaves (human vs. rational) [99]. This distinction leads to four different approaches to AI:

- **Acting humanly**
  This category is basically the area where the Turing Test can be applied to test the human characteristics of a program. This test consists of four main aspects: natural language processing, knowledge representation, automated reasoning, and machine learning. Note that this test does not test the physical abilities. This is done by the Total Turing Test which only works in connection with computer vision and robotics.

- **Thinking humanly**
  This category can also be covered by the field of cognitive science, which brings together computer models from AI and experimental techniques from psychology. Here, the goal is to imitate human thinking as closely as possible. It is not only the solution of a problem which is interesting, but how the program achieves this solution.

- **Thinking rationally**
  This field of AI is based on Aristotle who was one of the first to codify “right thinking”. His famous syllogisms lead later to the field of logic. There are two main obstacles pointed out here. First, it is not easy to handle uncertain things (which obviously do exist in our world) and, second, there is a big difference between being able to solve a problem “in principle” and doing so in practice.

- **Acting rationally**
  Acting rationally means acting so as to achieve one’s goals, given one’s beliefs. The last category leads therefore towards an rational agent\(^2\) approach. An agent is something that perceives and acts. Correct inference from the last category is only a part of a rational

---

1. They point out that they do not suggest that humans are irrational in the sense of “emotionally unstable”. They note that systematic errors in human reasoning are normal for human beings and, for example, not everyone can be a grandmaster in chess.
agent. A rational agent also needs possibilities to make decisions upon not provably cor-
rect things and on events which cannot be reasonably said to involve inference (e.g.
reflexes, etc.).

Jon Doyle et al. divide present-day AI research into the following primary (and overlapping) areas [30]:

- knowledge representation and articulation
- learning and adaptation
- deliberation, planning, and acting
- speech and language processing
- image understanding and synthesis
- manipulation and locomotion seeks
- autonomous agents and robots
- multiagent systems
- cognitive modeling
- mathematical foundations

Most of the research projects cover just one or some minor number of these areas and are therefore not totally suitable for an all-in-one solution (if this would be possible) which would be desired for a rational agent. We will concentrate on some aspects of these areas in the following sections. First, we’d like to gain insight into several learning techniques, then go on with decision making and problem solving. Further on we take a look at planning and acting which guides us to more complex AI systems and frameworks.

3.2 Adaptation and Learning

This chapter will give the reader an overview on self-adapting techniques for genetic evolution or learning.

3.2.1 Genetic and Evolutionary Algorithms

Genetic Algorithms (GA) [78][123][111] are used to solve a very complex problem with many degrees of freedom where normal computational power would not be sufficient to search the whole domain of possible states to get the very best solution. Instead this class of algorithms finds a good (but maybe not the best) solution. The nature of the algorithm stems from the evolution theory, where three basic rules are important for the development of a natural entity (be it a plant or a creature) over several generations:

- **Combination** is the process of combining the genetic information of one entity with another where some information of either parent is used to form a new one.
- **Mutation/Recombination** is the process where the genetic information is (partly) changed/recombined with or without considering it’s direct impact on the life-form.
- **Selection** is the process of the survival of the fittest, where only the life-forms survive which are better than the others. This is the main process in genetic algorithms where a function has to be defined to evaluate the fitness of the life-form.

2. The term agent is widely used within this chapter. We will try to define it in chapter 5. Until this def-
inition, the reader can think about it as a software robot which can act autonomously.
The main task while building a GA solution for a particular problem is often to find this fitness function which gives a value of fitness for every solution instance. A GA specific task is to code the variables of the problem as chromosomes to obtain the possibility of combination and mutation. Even more important is the inverse of this function which maps a certain set of chromosomes to a instance in the solution space, since only the chromosomes are mutated and only the instance is evaluated.

GA start with a (random) set of chromosomes which then are evaluated and recombined with respect to their fitness values in a way that chromosomes with a better value to the target problem have better chances to reproduce. This leads towards local extrema in the domain of all possible states but the mutation or recombination process will help to overcome barriers and find other regions with even better solutions than the actual.

See Tomassini for a good survey on GA [111], Whitley for a tutorial on GAs [123].

Mitchell and Forest [78] list the various forms of problems where GA have been successfully applied to: Optimization (numerical and combinatorical), automatic programming (evolving of computer programs, design of cellular automata and sorting networks), machine and robot learning (classification and prediction tasks, e.g. weather forecast, rules for classifier systems and symbolic production systems, control of robots), economic models (model of the process of innovation, development of bidding strategies, emergence of economic markets), immune system models (somatic mutation, discovery of multi-gene families), and ecological models (biological arms races, host-parasite co-evolution, symbiosis, resource flow).

Designing a GA can be very difficult. Beside the mentioned tasks of defining a fitness function and the chromosome coding, Harik et al. considers the size of the population as a further parameter [42]. They provide a model for predicting the convergence quality of GA depending on the size.

GAs have been used in various research projects to draw natural looking graphics [102], evolve virtual creatures [103], simulate an ant colony and their evolution [25], or generate robot configurations [68]. Balakrishnan et al. use GA to design neural networks [7] and Yamauchi et al. explore the use of GA to evolve neural networks capable of sequential behavior and learning [127].

### 3.2.2 Simulated Annealing

**Simulated annealing** can be considered a similar approach to find local extrema in a unknown multi-dimensional space. Like genetic algorithms, it is a very general optimization method which stochastically simulates the slow cooling of a physical system. The main idea is to use the temperature as a measure of flexibility for choosing the next step. While the temperature cools down, the steps get smaller and smaller and will finally be minimal when the solution is near the optima.

Martin and Otto combine simulated annealing with local search heuristics in order to find a solution faster [72]. Boese and Khang use simulated annealing on neural networks, which are discussed in the next section [11].

### 3.2.3 Neural Networks / Learning from Examples

Beside genetic algorithms and simulated annealing, Neural Networks (NN) are another group of algorithms derived from a natural phenomenon. NN try to simulate a part of a natural brain with neurons, synapses and dendrites. It consists of several layers of so called units, where one layer is the input layer and one is the output layer [99]. The layers in the middle are called hidden layers. From every layer to the next, every unit (i.e. neuron) is connected by a weighted
joint to every unit in the above layer. Every unit sums up the incoming signals and produces an output signal (continuous or discrete) dependent on the activation function which is assigned to the units. This activation function is usually chosen among the step, sign or sigmoid function.

Learning in NN is the process of updating the weights between the units. NNs can be trained to give a desired output given a specific input stimulus by applying training examples which must be available. Different algorithms use supervised or unsupervised training and are usually a variation of back-propagation learning.

Difficulties with NN arise when deciding about the structure of the NN, the number of input and output units and the number and size of the hidden layers. Ripley provides a good reference on pattern recognition issues with NN [96]. Minsky and Papert show the limits of this approach [77].

Applications of NN arise in many different areas, such as handwritten character recognition [66] or driving [85]. Controllers for different tasks such as back-parking of a truck with a trailer, the landing of a space shuttle on the moon surface, and an animated dolphin learning swimming [40] are also implemented using NN. The game Creatures by Inscape [36] uses heterogeneous NN to simulate the brain of each Norn. There, the initial model contains approximately 1’000 neurons, grouped into 9 lobes, and interconnected through roughly 5’000 synapses.

Research on NN has been done in various ways, on which we will provide some examples here. Sontag briefly surveys some results relevant to the suitability of NN as models for dynamical systems as well as controllers for nonlinear plants [107]. LeCun et al. improves the performance of a NN by removing unimportant weights [67]. Yamauchi et al. use genetic algorithms to evolve continuous-time recurrent NNs capable of sequential behavior and learning [127]. Mathias replaces the batch model of learning by an interactive teaching model to obtain better performance in learning [74]. Balakrishnan discusses evolutionary designs of neural architectures [7], and Boehse and Khang use simulated annealing on NN to show that the cooling strategy not necessarily decreases monotonically to zero [11]. At last, Miltrup and Schnitger deepen into the memory complexity of NN [76].

### 3.2.4 Reinforcement Learning

In contrast to neural networks, where the learning process is based on examples, the reinforcement learning (RL) uses success, failure, reward, and punishment as indicators of the result. Therefore, a trial-and-error interaction has to be performed. This process is described in [99] and Kaelbling and Littman provides a large survey on RL [52] where they summarize both the historical basis of the field and a broad selection of current work.

RL is a task of learning a successful function. The main difficulty originates from the fact that the desired function is completely unknown. Therefore the reinforcement only gives a fuzzy rating, since the exact reason of success or failure is also unknown. Consider a chess-playing agent who looses a game. It has no intuitiveness about the reason and has to determine by itself which moves were poorly chosen. Furthermore, [99] distinguishes between passive and active learning. A passive learner simply watches the world going on while a active learner has to act on the learned information.

Another problem is the task of exploration. This task has to be designed very carefully to obtain an equilibrium between exploration and exploitation. The problem bases on the fact that the agent has to choose an action. Should it take the most appropriate action or should it choose one of the non-optimal actions to receive a better reward at the end of the action sequence? This question leads to the so called bandit problem. Most algorithms overcoming this problem use
statistical analysis of the sequences, whereas Auer et al. use an advisory with full control over the payoffs [3].

Applications of RL arise in many fields. [52] lists game playing, robotics, and controlling as the major areas. Research has been done for example with a social reinforcement agent [48] capable of learning preferences of a number of users of an online-chat. [53] verifies game designs and playing strategies using RL. [129] evolved decision making agents working on large data sets able to communicate in order to optimize the performance.

3.2.5 Support Vector Machines

Supporting Vector Machines (SVM) is a classification technique developed at AT&T Bell Labs [82][15][14]. It is an alternative training technique for polynomial, radial basis function and multi-layer perceptron classifiers. It solves a quadratic programming problem with linear inequality and equality constraints to obtain the weights of the network. Compared to neural networks, SVM do not solve a non-convex, unconstrained minimization problem and obtain therefore a better generalization performance [15]. On the other hand, SVM require a large amount of memory since the quadratic form is completely dense and the memory needed grows with the square of the number of data points.

Therefore, SVM can be used for several classification and recognition problems, such as face detection [82], text categorization [51] and many others. A list of online available applications is provided at http://www.clopinet.com/isabelle/Projects/SVM/applist.html.

Faloutsos et al. uses SVM to determine preconditions for controllers of skeleton movements [32]. He states the advantages of SVMs in the better performance and learning rate than competing methods like NN. As stated above, the most obvious disadvantage is the enormous memory consumption of the algorithm.

3.2.6 Other Learning Techniques

Ram presents an overview of goal-driven learning techniques in [91]. More and more researchers support the view that the learning process is strongly influenced by the learner’s goal:

“The fundamental tenet of goal-driven learning is that learning is largely an active and strategic process in which the learner, human or machine, attempts to identify and satisfy its information needs in the context of its tasks and goals, its prior knowledge, its capabilities, and environmental opportunities for learning.” [91]

This work addresses issues such as the role and utility of goals in learning, the justification of such models through cognitive results, goal-based processes for deciding what to learn and for guiding learning and the learning process, and pragmatic implications of goal-driven learning for design of instructional environments.

3.3 Problem Solving

This section shows how an agent can act by establishing goals and considering sequences of actions that might achieve these goals. A goal and a set of means for achieving the goal is called a problem, and the process of exploring what the means can do is called search.

After a goal has been formulated, it is necessary to consider which actions and states should be considered. If this has been specified, the search for a sequence of transitions or actions that
lead to the desired goal state can be started. This returns a solution in the form of an action sequence which can be carried out.

### 3.3.1 Problem Formulation

The task of problem formulation is the first issue in this process. [99] specifies four essentially different types of problems: single-state problems, multi-state problems, contingency problems, and exploration problems.

**Single State Problems.** If the world is accessible and the agent’s sensors give it enough information to tell exactly which state it is in, and each possible action determines exactly the next state then the agent is able to find a sequence of actions which lead to the goal state if there is one.

**Multiple-State Problems.** Considering a world, where the agent’s sensors do not provide enough information to ensure the agent about its current state. It has to act under uncertainty and has to start the search from multiple states and search for a common solution. Therefore, the agent has to reason about a set of states rather than a single state.

**Contingency Problems.** If it turns out, that there is no fixed action to reach a certain goal, the current problem cannot be solved in the current state. But maybe, the agent’s sensors could receive new information during the execution phase which could help to solve the problem. Notice that the agent must now calculate a whole tree of actions, rather than a single action sequence. In general, each branch of the tree deals with a possible contingency that might arise. Many problems in the real, physical world are contingency problems, because exact prediction is impossible.

**Exploration Problems.** Imagine being lost in a foreign country without a map and no information about the effects of one’s actions\(^1\). The agent must experiment, and gradually refine its knowledge about the effects of its actions and the possible states. The blind selection of actions can lead to dangerous or lethal situations. But if the agent survives, it learns a “map” of the environment, which can then be used to solve subsequent problems.

A problem is a collection of informations that the agent will use for its decisions. The basic elements therein are actions and states. Actions can be seen as operators on states or successor functions. Therefore, the initial state should be known. The state space consists of all states which can be reached from the initial state by applying any sequence of actions. The goal test can be applied to any state and returns true if it is a goal state. If there are multiple ways that reach a certain goal, a path cost function should be provided to select the one with the lowest costs. We now can define a datatype with which one can represent problems:

```ml
datatype problem {
    initial state;
    operators;
    goal-test;
    path-cost function
}
```

---

1. This is actually the task faced by newborn babies.
### 3.3.2 Acting

The afore mentioned problem formulations lead to the question about the particular point of time for selecting an action. While the single-state and multiple-state problems generate a sequence of actions that lead to the goal state, these can be applied after the search has been finished. Therefore, similar algorithms can be used to find such a solution. Some of these are discussed in the next section. But contingency problems do not lead directly to a solution sequence. Therefore, the agent has to act before it has found a guaranteed plan. This is useful because it is often better to actually start executing and see which contingencies do arise instead of considering in advance all possible contingencies. This is called *interleaving* of search and execution.

### 3.3.3 Searching

Searching for a solution is basically the generation of new sets of states by applying the operators. This process is called *expanding* the state. If different states can be reached from another state, the choice of which state to expand first is determined by the search strategy.

The choice of the right search strategy is very important. One can take four criteria into account to find the appropriate strategy:

- **Completeness**: Is it guaranteed to find a solution if there is one?
- **Time complexity**: How long does it take to find a solution?
- **Space complexity**: How much memory does it need to perform the search?
- **Optimality**: Does the strategy find the highest quality solution when there are several different solutions?

Furthermore, one distinguishes between *uninformed* and *informed* search. While in the first case, there is no information about the steps or path cost, the latter can use considerations about the search space (e.g. direction, maximal depth). The formalized considerations are called *heuristics*.

### 3.3.4 Comparison of Search Strategies

The following table presents an evaluation of different uninformed search strategies. Breadth first searches through all nodes expanded from the root node first, where depth first searches along a branch of the tree first. Uniform cost always expands the lowest-cost node on the fringe. Depth limited search avoids the pitfalls of depth first by setting a cutoff on the depth. Iterative deepening tries all possible depth limits in a ascending way. Last, the bidirectional attempt
searches simultaneously from both the initial state and the goal state and stops when both searches meet in the middle.

When using informed search methods, there is usually an evaluation function that returns a number purporting to describe the desirability of expanding a node. The best-first search uses this classification and expands the node with the best evaluation. One of the simplest best-first search strategies is to minimize the estimated cost to reach the goal. Therefore, we need a heuristic function $h(n)$. The resulting algorithm is called greedy search.

Another approach is to minimize the total path cost, which leads to the so called $A^*$ search. Unfortunately, this search is neither complete nor optimal. When $g(n)$ gives the path cost from the initial state to node $n$ and $h(n)$ is the estimated cost of the cheapest path from $n$ to the goal, $f(n) = g(n) + h(n)$ provides the estimated cost of the cheapest solution through $n$. A* is optimally efficient for any given heuristic function. That is, no other optimal algorithm is guaranteed to expand fewer nodes than A*. A* is also discussed in chapter 8.7 with respect to the implementation.

The choice of a heuristic function can be a difficult task. In many cases, different heuristics seem to be applicable, but only one can be the best. [99] discusses the appropriate questions concerning the choice of a heuristic function. Additionally, the size of the memory restricts in many cases the choice of an search strategy. Here, we will only state that there are memory conserving search strategies such as IDA* or SMA*. Here again, [99] deals with this aspect.

Last but not least, iterative improvement algorithms like gradient descent search or simulated annealing provide the most practical approach for problems, where the path to the goal state is irrelevant (e.g. 8-queens, VLSI layout).

### 3.4 Knowledge

The last section showed that an agent can have goals and can search for solutions to reach such a goal. This often leads to a more sophisticated behavior than a simple reactive agent has. We now focus on general logical reasoning.

The central component of a knowledge-based agent is it’s knowledge base (KB). This is a set of representations of facts about the world. Each individual representation is called a sentence. There must be a way to add new sentences or query what is known. These are commonly called

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth First</th>
<th>Uniform Cost</th>
<th>Depth First</th>
<th>Depth Limited</th>
<th>Iterative Deepening</th>
<th>Bidirec-tional$^a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>$b^d$</td>
<td>$b^d$</td>
<td>$b^n$</td>
<td>$b^l$</td>
<td>$b^d$</td>
<td>$b^{d/2}$</td>
</tr>
<tr>
<td>Space</td>
<td>$b^d$</td>
<td>$b^d$</td>
<td>$b^m$</td>
<td>$b^l$</td>
<td>$b^d$</td>
<td>$b^{d/2}$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if $l \geq d$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.1: Evaluation of different uninformed search strategies. [99]

$b$ is the branching factor;
$d$ is the depth of solution;
$m$ is the maximum depth;
$l$ is the depth limit

a. if applicable

When using informed search methods, there is usually an evaluation function that returns a number purporting to describe the desirability of expanding a node. The best-first search uses this classification and expands the node with the best evaluation. One of the simplest best-first search strategies is to minimize the estimated cost to reach the goal. Therefore, we need a heuristic function $h(n)$. The resulting algorithm is called greedy search.
3.4 KNOWLEDGE

TELL and ASK. Determine, what follows from what the KB has been TELLed is the job of the inference mechanism, the other main component of a knowledge based agent.

At any point, we can describe a knowledge-based agent at three levels:

- The knowledge level or epistemological level is the most abstract. Here we describe the agent by saying what it knows.
- The logical level is where the knowledge is encoded into sentences.
- Finally, the implementation level is the level that runs on the agent architecture. The choice of implementation is very important to the efficiency performance of the agent, but it is irrelevant to the logical and knowledge level.

The agent’s initial program, before it starts perceiving, is built by adding one by one the sentences that represent the designer’s knowledge of the environment. This is called the declarative approach to system building. Also, one can design learning mechanisms that output general knowledge about the environment given a series of percepts. By hooking up a learning mechanism to a knowledge-based agent, one can make the agent fully autonomous.

Michael van Lent and John Laird name knowledge as one of the most important parts of an AI engine for games [118]. They even call for an general KB with game-independent knowledge. Furthermore, they state that the cause of unrealistic behavior of game agents is often unrealistic or missing knowledge in the KB.

3.4.1 Knowledge Representation

Jon Doyle et al. assert knowledge representation as a key direction in AI:

“The problem of formalizing knowledge remains one of the principal challenges to AI research. Current successful knowledge-based systems rely on carefully limiting the scope and domain of the formalized knowledge, in order to make it tractable to collect, codify, and correct this knowledge. (...) The current formalizations (...) fail to support the integration aims of AI research in several ways, and overcoming these limitations forms a major task for AI research that forces consideration of many fundamental issues in knowledge representation.” [30]

The subject of knowledge representation is the task to represent knowledge in a way the computer can work with. There are two aspects of the representation: The syntax and the semantics. The syntax determines how to write down the knowledge (in memory) while the semantics describe what the meaning or content of this sentence is.

If both semantics and syntax are defined precisely, we call the language a logic. Additionally, one needs a proof theory, a set of rules for deducting the entailments of a set of sentences.

One further classifies logics by both the ontological and the epistemological commitments. The word ontology means a particular theory of the nature of being or existence. Therefore, the ontological commitments have to do with the nature of reality while the epistemological ones
have to do with the possible states of knowledge an agent can have. The following table classifies a subset of known logic languages.

<table>
<thead>
<tr>
<th>Language</th>
<th>Ontological Commitment (What exists in the world)</th>
<th>Epistemological Commitment (What an agent believes about facts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional Logic</td>
<td>facts</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>First-Order Logic</td>
<td>facts, objects, relations</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Temporal Logic</td>
<td>facts, objects, relations, times</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Probability Logic</td>
<td>facts</td>
<td>degree of belief 0..1</td>
</tr>
<tr>
<td>Fuzzy Logic</td>
<td>degree of truth</td>
<td>degree of belief 0..1</td>
</tr>
</tbody>
</table>

Table 3.2: Formal languages and their ontological and epistemological commitments.

3.4.2 Building a Knowledge Base

When building a KB, the task of knowledge engineering, [99] present five steps which have to be considered first in order to obtain a clear and correct KB:

- **Decide what to talk about.** Understand the domain well enough to know which objects and facts need to be talked about, and which can be ignored.
- **Decide on a vocabulary of predicates, functions, and constants.** That is, translate the important domain-level concepts into logic-level names.
- **Encode general knowledge about the domain.** By writing logical sentences or axioms about the terms in the ontology (see below), we accomplish two goals: first, we make the terms more precise so that humans will agree on their interpretation. Second, we make it possible to run inference procedures to automatically derive consequences from the KB.
- **Encode a description of the specific problem instance.** This will mostly involve writing atomic sentences about instances of concepts that are already part of the ontology.
- **Pose queries to the inference procedure and get answers.** This is the reward for the work so far. We can derive facts from the KB we are interested in knowing.

The first two steps are also known as **ontological engineering.** Both steps together determine what kind of things exist, but do not determine their specific properties and interrelationships. [99] deepen into this field and present ontological thoughts about measures, places, processes, times, intervals, actions, objects, substances, mental events, and mental objects.

3.5 Planning

A planning agent differs from a problem-solving agent in its representation of goals, states, and actions. The use of explicit, logical representations enables the planner to direct its deliberations much more sensibly [99]. We will introduce a simple planning agent and situation calculus. In addition, partial-order, hierarchical, and conditional planning will be discussed in the second part of this section.

Assuming that the world state is accessible\(^1\), i.e. that the agent can build a complete representation of the world. Given a goal, the agent can generate a plan using a suitable planning algorithm and execute it.

\(^1\) The classification of the environment is discussed in section 5.2.
[99] present the outline of a simple planning agent as in figure 3.2. The agent first perceives

the world, updates its KB, and gets the current state description back. Then, it generates a goal which is then transformed into a plan, a sequence of actions. Once a plan is present, the agent keeps executing it until it has finished. The agent notifies the KB about every executed step to keep the KB up-to-date.

We will not discuss the usefulness of this algorithm. However, we will reflect on the IdealPlanner which is used here. While a problem-solving agent can have difficulties to choose an action appropriate to a certain situation, the planning agent can connect states and actions using a first order logic. For example, consider the task to buy some items. When the agent has many actions to choose from, the branching factor of its search-tree would be in the thousands or millions because the problem-solving agent cannot eliminate actions from its considerations. In the supermarket, the agent would consider buying apples, milk, and cheese, or even going to sleep, reading a book and so on. Furthermore, the problem-solving agent cannot start the execution of its plan while the plan is not complete. However, the planning agent uses a first-order logic to represent its states, goals and actions. This enables the planner to make direct connections between states and actions. With the example above, the planning agent would only consider actions which include the goal state, for example Buy(x) => Have(x), when the goal is Have(Milk).

Situation calculus provides a good approach for a planning algorithm. Today, most planners use a restricted language known as STRIPS\(^1\), or extensions thereof. In the STRIPS language,
states and goals are described by conjunctions of function free ground literals. Actions or operators consist of three components: The action description, the precondition and the effect.

A plan can be found using various ways. One could imagine to search in the situation space beginning at the initial situation (progressive planner) or at the goal state (regressive planner). An alternative might be to search in the plan space for a suitable plan. There, we start with a simple, incomplete plan, which we call a partial plan.

Representations for plans are also manifold. Consider the example of putting on a pair of shoes. But which shoe should come first? Here, the principle of least commitment is often used, which says that one should only make choices when it’s really necessary. This reduces the amount of backtracking. But we also want to make sure that putting on the right sock comes before putting on the right shoe. But this is uncorrelated to the left sock and shoe. A partial order planner can represent plans where some actions are ordered with respect to each other and other steps are not. A totally ordered plan that is derived from such a plan is called a linearization. [99] provide a large chapter about partial order planning.

Hierarchical planning is an approach to decomposing the planning into multiple levels of abstraction. At a higher level, a planner may find a sequence of places to go, while at a lower level, concrete movement instructions for the effectors of the agent must be found. Resolving the plan directly at the lower level would produce a much longer plan. At the lowest level, primitive operators complete the plan which can be executed directly by the agent.

Conditional Planning deals with incomplete information by constructing a conditional plan that accounts for each possible situation. The agents decides on its sensory information which conditions evaluate to hold. Another approach with incomplete or incorrect information is execution monitoring. Here, the agent monitors what is happening and can therefore decide on what is going on and when things went wrong. In this case, a replanning has to be done to find a new plan from the unexpected situation. Conditional planning introduces sensing actions that cause relevant conditions to become known by the agent.

3.6 Uncertain Knowledge

While first-order logic seems to be able to handle the above mentioned problems and tasks, one problem arises when using first-order logic. Because agents almost never have access to the whole truth about their environment, they have to act under uncertainty. Consider for example an agent planning to drive to the airport. It could find a plan that says that the airport can be reached within 90 minutes, but only if there is no accident, no traffic jam, and so on. It could also find a plan stating to leave the home 120 minutes before the airplane starts. This might increase the agents belief to get there on time but it will also increase the probability of a longer wait. The rational decision what to do therefore depends on both the relative importance of various goals and the likelihood that, and degree of which, they will be achieved.

The agent has to provide a degree of belief for the relevant sentences with which the probability theory can deal. The degree of belief is strongly coupled to the evidence, which stands for the acquisition of sensory information. One distinguishes between prior or unconditional probability before the evidence is obtained and posterior or conditional probability afterwards. To distinguish between the various possible plans, the agent needs an utility theory to represent and reason with preferences. These preferences expressed by utilities are then combined with

1. STRIPS has been named after a pioneering planning algorithm known as STanford Research Institute Problem Solver. Despite the term problem solver the program is what we call here a planner.
1. Utility is used here in the sense of “the quality of being useful"
probabilities in the decision theory. Here the fundamental idea is that an agent is rational if and only if it chooses the action that yields the highest expected utility, averaged over all the possible outcomes of the action [99]. This is called the principle of Maximum Expected Utility (MEU).

The use of probabilities leads to subjects like conditional probability, joint probability and the Bayes’ rule which we will not discuss here. The reader is once again referred to [99] and others.

Inference under uncertainty is strongly coupled to belief networks, a data structure to represent the dependence between variables and to give a concise specification of the joint probability distribution. A belief network is a graph in which the following holds [99]:

1. A set of random variables makes up the nodes of the network
2. A set of directed links or arrows connects pairs of nodes. The intuitive meaning of an arrow from node X to node Y is that X has a direct influence on Y.
3. Each node has a conditional probability table that quantifies the effects that the parents have on the node. The parents of a node are all those nodes that have arrows pointing to it.
4. The graph has no directed cycles (hence is a DAG)

These rules generate a compact representation for beliefs and their relationships. The order in which nodes are inserted affects the number of links which influences the complexity of inference. Belief networks can then be used to inference in four different ways:

- **Diagnostic inference**, which concludes from effects to causes.
- **Causal inference**, in the opposite direction.
- **Intercausal inference**, which infers between causes of a common effect.
- **Mixed inference**, which combines two or more of the above.

Furthermore, a sensitivity analysis can be performed to understand which aspects of the modeled situation have the greatest impact on the probabilities.

The above mentioned utility theory has its roots in economics, where money is the utility. This theory provides six axioms to determine the preference of different states. To classify these states one needs a utility function which maps states to real numbers. On which specific attributes this function depends is not specified and can be for example the agent’s energy level, money, or a combination of these.

In research, Raja et al. focus on constructing a framework for robust agent control in open environments, using a soft real-time scheduling approach which satisfies all aspects of problem solving [90].

### 3.7 Communication

When multiple agents act in the same environment, they might communicate with each other in order to gain an advantage as a collective or as an individual. Here we will only provide a list of different possible communication acts [99]:

- **Inform** each other about the part of the world each has explored, so that each agent has less exploring to do. This is done by making statements.

1. This is the most common name. Others are Bayesian network, probabilistic network, causal network, and knowledge map.
• **Query** other agents about particular aspects of the world. This is typically done by asking questions.
• **Answer** questions. This is a kind of informing.
• **Request** or **command** other agents to perform actions. This can be seen as impolite and therefore often an indirect speech act is performed.
• **Promise** to do things or **offer** deals.
• **Acknowledge** requests and offers.
• **Share** feelings and experiences with each other.

Beside the intention of transferring some information to the hearer or make him taking some action, communication has the further purpose of establishing trust and social ties. Free communication can only be obtained with the definition of a suitable lexicon and grammar.

Jon Doyle writes in “Strategic Directions in AI”:

“Efficient and natural communication holds the key to many of the promises of computers, given that relying on command languages, menus, textual display, and other traditional media stymies many potential applications. (...) AI has long addressed these issues, and has contributed to great progress in realizing linguistic and visual communication mechanisms involving multiple modalities, including natural language, gestures, and graphics. The most general form of these abilities, however, lies far beyond current scientific understanding and computer technology.” [30]

Research with communicating agents has been done with simulated colonies of ants by Parunak and Brueckner [83], where a decentralized method for motion planning inspired by synthetic pheromones\(^1\) of insect populations. This approach can be used to solve the classical Missionaries and Cannibals problem\(^2\) without central control. Demazeau proposes an interaction language associated with different interaction protocols [29]. He illustrates the use of them for social and individual control issues.

---

1. A pheromone is a scent produced by glands which is not sensible by the nose but only by special receptors.
2. Three missionaries and three cannibals find themselves on a bank of a river to be crossed with a canoe capable of carrying one or two people. The number of cannibals must not outnumber the missionaries on either bank of the river. The problem is to plan a sequence of moves that gets all six people safely across the river.
This chapter will give the reader a short introduction to commercial computer games and entertainment with respect to ALife and artificial intelligence. First, we will define the term game and classify the most common genres of games. After some words on game characters and game development, we will finish this chapter by some remarks on some recent computer-based cinematography issues.

4.1 Games

Angelides et al. provide a good definition for a game:

“The term game refers to those simulations which work wholly or partly on the basis of a player’s decisions.” [1]

This includes a wide area of different styles of games which are classified first by the number of players or opponents. Games with only one player (e.g. Solitaire, etc.) will not be discussed in this survey due to the non-existence of other than the players decisions. Two-player games seem to be more interesting since the decisions of a player must consider the opponents possible reactions and can create therefore a computational problem when trying to predict more than a minor number of draws.

The multi-player games which are considered in this report can be classified into two main categories:

- **Classical Games** such as boardgames (Chess, Checkers, etc.), and
- **Pure Computer Games** such as first-person shooters (FPS), simulations, sport simulations, etc.

Boardgames have been very deeply investigated by researchers, because they mostly have simple and clear rules, a finite board and state, and are therefore computational feasible. With a large number of possible moves it is impossible to create a database containing for every position the perfect move towards a winning situation.

Computer games differ from boardgames. In these games, the player is interactively steering one or more avatars in a virtual world and must overcome some obstacles and/or opponents.
4.1.1 Classical Games

Since the famous Chess game between IBM’s Deep Blue and Gary Kasparov, board gaming has received a huge amount of publicity. Games like Chess, Checkers, Go and others require the computer to think ahead of the current state of the game. The goal is to predict what move the opponent might pick. Therefore, the computer needs a rating on all of the possible moves. To obtain that, the computer searches the tree of all possible moves and assigns each node a value to indicate the gain of the sequence of moves until this node. Then, the move which minimizes\(^1\) the opponents maximal move is chosen as the one to perform. Therefore, the tree is called minimax tree. An extension to this algorithm is the alpha-beta pruning, which prunes these parts of the tree which must not be considered for further search, because the value of the node is too small. Chapter 5 of \[99\] provides an overview on these algorithms and also on several board-games.

A refutation to this algorithm might be the assumption that the computer and its opponent use the same set of heuristics to determine which move to take. This is a rather tenuous assumption when playing against humans. Another weakness is the limited depth of the search tree where only five to ten moves ahead can be considered. This might not be enough to see the advantages of a positional game\(^2\) where the gain of a move is not clear until the very end of the game.

Some boardgames have been completely solved in the sense that every possible state of the game is marked as a winning, loosing or draw game. For example, Ralf Gasser, a former member of Prof. Nievergelt’s group at ETH Zurich, showed that Merrils (Mühle) is a draw game, when played perfectly by both players.

4.1.2 Pure Computer Games

In computer games the player controls usually one (sometimes more than one) virtual avatar which moves and behaves inside a virtual world. In such games, the time available to calculate the behavior is usually very short and the task must not be computationally expensive. Therefore, such games often use Finite State Machines (FSM) to model the behavior of the players avatar and its opponents. These FSM’s usually have less than ten states to keep it compact. In every state, the behavior of the avatar is predefined and also deterministic. This leads to very few computational needs and is therefore very fast. But after playing the game several times, this model is obviously to weak to produce any surprise. The fun factor is therefore not the avatar but the huge number of different enemies, puzzles, and levels.

The simulation of the non-player characters (NPC) requires an intelligent modeling of their behavior. In first person shooter games (Quake, Doom, etc.), role playing games (Final Fantasy, The Myst, etc.), and sport games the opponents should behave like humans. Also, this has often been done with FSM’s to have a compact, fast and deterministic code. Especially here, determinism is often regarded as a necessity, since prediction of the game behavior makes the development and debugging process easier. Also, some puzzles rely on the fact, that the opponents always appear in the same situation.

More recently, games such as Half Life, Descent 3, Quake III, and Unreal Tournament have incorporated path-planning and many tactics that make these enemies more human-like. Laird

\(^1\) With respect to the well-defined rating

\(^2\) Kasparov’s tactics in the above mentioned game was to get into such positional advantages. Nevertheless, he lost the match 3.5 to 2.5
et al. have implemented enemies for Quake II that have the same strengths and weaknesses as human players [118][61].

Laird and van Lent list seven categories of computer games [59]:

- **Action games**, which involve the human player controlling a character in a virtual environment.
- **Role-playing games**, where a human can play different types of characters.
- **Adventure games** rather do emphasize on puzzle solving than on fighting.
- **Strategy games**, where the player controls many units (often with military origin) against one or more opponents.
- **God games**, where the player has god-like control over a simulated world.
- **Team sports**, in which the player is part of a team and plays a combination of coach and player.
- **Individual sports**, where the player competes directly with other players.

In action games, AI is used to control the enemies. During the last years, realism in graphics has been the competitive aspect, but this seems to have run its course, with better AI becoming the point of comparison. Recent action games are no longer single-player games and have extended to team-player games over a network. These teams can be built with human or AI players.

As in action games, role-playing games use AI to control the enemies of the player’s avatar. Similar, but not in the same way, are adventure games. Here AI can be used to create realistic goal-driven characters that the player must interact with appropriately to further their progress in the game. Laird and van Lent state the Holy Grail of interactive fiction to have a computer director who can dynamically adjust the story and plot based on the actions of humans [59]. Today’s games often have fixed scripts with little variability to force the player through essentially linear stories.

Strategy games, on the other hand, work similar to the cellular automata introduced in the first chapter. A large number of entities are controlled by very simple rules that emerge a complex behavior as a collective. AI is used here in two roles: To control the behavior of the controlled units¹, and as a strategic opponent that must play the same type of game. God games do not differ very much of strategy games with respect to the incorporated AI. Here, AI controls again the behavior of the individual characters in the world directed by the player.

There are various possible scenarios to model. Examples are towns (the various versions of SimCity [101]), people or families (The Sims [104]), both together (the upcoming SimsVille [105]), or sophisticated animals (Creatures [26][36]). Whereas the first simulates the economical, social and growth development of a town with a large number of citizens, the second simulates a small number of humans with different skills in a virtual home which can be furnished individually. The last example provides virtual pets with an neural network for sensory-motor coordination and behavior selection, an artificial biochemistry that models a simple metabolism, and a hormonal system which influences the neural network and ontogenetic development. The information about the network and the detail of biochemistry are stored in a genetic code allowing a evolutionary development.

In team sports games, AI is used in a similar way as in strategy games. First, to control the individual players, second, to take the part of the opponents. Usually, the human controls one

---

¹. Laird and van Lent point out that this behavior is not meant to be autonomous but these units are meant to act as good soldiers who “follow orders” [59].
key player (the quarterback, or the ball-guiding player in soccer), while the computer controls all the other members of the team. Here, team tactics as a collective behavior can influence the behavior of an individual player. An additional AI role is the commentator, who gives the play by play, and color commentary of the game. Examples for team sports are soccer (FIFA2002), football (NFL2002), ice-hockey (NHL2002), etc.

Last, individual sports games need similar AI techniques as action games: the opponents are more like enemies than strategic opponents. Examples here are skate-boarding (Tony Hawk Pro Skater), snowboarding, biking and others.

4.2 Game Characters

The different kinds of computer games have been provided in the previous section. Here we will focus on the roles of different computer-driven game characters.

Laird and van Lent claim that human-level AI systems are the ones that one dreamt about when first heard of AI: HAL from “2001 - A Space Odyssey”, Data from “Star Trek”, and CP30 and R2D2 from “Star Wars”. Their capabilities integrate seamlessly into human skills: real-time response, robustness, autonomous intelligent interaction with the environment, planning, communication with natural language, common sense reasoning, creativity, and learning. They further declare computer games as the human-level AI killer application because of its constraints and the required realism [59].

They also provide a list of different basic roles:

- **Tactical enemies**, where many general AI problems have to be solved: Navigation, path planning, spatial reasoning, and temporal reasoning. They also need a perception system with the same capabilities as humans.

- **Partners** involve similar AI techniques as tactical enemies. However, while enemies emphasize autonomy, partners emphasize effortless cooperation and coordination between the both player and the partner. The partner AI must coordinate its behavior, understand teamwork, model the goals of the human, and adapt to his style.

- **Support characters** are usually some of the least sophisticated AI characters, but they have the most promise to improve games. Since these characters need to exist in a virtual world and generally play a human role in this world, they provide a useful first step towards human-level AI. They must interact with and adapt to human players and provide human-like repossessions.

- **Strategic opponents** often have advantages because most game developers resort to cheating to obtain a challenging opponent. Even with these advantages, most strategic opponents are predictable and easily beaten once their weaknesses are found. In team sports games, these opponents style of play must match a real world team about which the human player is likely to be very knowledgeable. The tasks of strategic opponents can be divided into two categories: allocating resources and issuing unit control commands.

- **AI-driven units** are used in strategy games, god games, and team sports games. Generally, a high-level command of the player has to be carried out. Because of the large number of simulated units, their computational needs must be kept very low [2]. Therefore, they are often controlled by FSMs and augmented by some path-planning and path-following.
The role of commentators is to observe the actions and to generate natural language comments suitable to describe the action [33]. The obvious challenge for a commentator is to create a natural language description of the on-going action in the game. The description may include both the moment to moment action as well as key tactical and strategy events that can require complex plan recognition and a deep understanding of the game.

After Michael van Lent and John Laird, an effective artificial intelligence engine should support agents that are reactive, context specific, flexible, realistic, and easy to develop [118]. They state that current computer games usually excel in some of these requirements while falling short on others. They further conclude that game agents as described above should display these properties.

In research, School advocates a knowledge acquisition approach to building characters and uses model-based classification techniques as a learning method [84].

### 4.3 Game Development

Developing a game is a very challenging task. For modern computer games, the development company has first to decide on which kind of machine the game should run: A console or an ordinary PC. These two possibilities are very different in developing. While console games can always expect the same hardware and a fixed speed, there are no such conditions on an ordinary PC. Every PC differs in CPU speed, graphics capabilities, memory size, input devices, and so on. Therefore, the main stage of a PC game development process is testing on compatibility. In return, the PC can offer more recent hardware than a console, such as a state-of-the-art graphics accelerator, and therefore it can make the game more impressive by using novel effects.

Because of the determined hardware of a console, the development process for a pure console game differs from the above described. The development team has a single goal, which is to find a setup where the interactive gameplay uses exactly 100% of the resources available and not a single instruction above (or below). The main constraint in this setup is the framerate which is fixed at a certain value. While the processing of geometric detail has first priority, simulation calculations have to deal with the remaining CPU time. Therefore, the simulation of ‘intelligent’ behavior for the game opponents often remains a simple state-driven process which is mainly deterministic and not computationally intensive.

For boardgames, Kalles and Kanellopoulos examine the use of reinforcement learning to the design of a new strategy game [53]. They use intelligently generated self-playing sequences to determine the playability of various initial board configurations. The machine’s a priori knowledge is restricted to the rules only, so that this design verification strategy can be adapted to other simple games, too.

### 4.4 Cinematography

During the last years, computer animated movies have become more and more popular. Toy Story 1&2, A Bug’s Life, Antz, Final Fantasy, Monsters Inc., and Ice Age are only some of the most popular movies. But not only this kind of movie uses computer animations. Almost every recent action movie uses computer generated animations to create realistic looking scenes which would be impossible to be filmed in the real world (e.g. Jurassic Park, The Matrix, Titanic, etc.).

Presently, the most difficult task in this genre is the animation of human movements which is the reason why completely computer animated movies look rather like an animated cartoon
than a realworld movie. Final Fantasy was the first fully computer generated movie presenting (more or less realistic) humans as main actors instead of animals.

While the main requirement in games is computational resources per time-step, animations for movies can be calculated off-line and their needs for computation exceeds the one for games by far. But the demand for realistic looking graphics and motions is much higher in movies. Therefore, the models used for cinematography are always much more precise and detailed than game characters.

In chapter 7 we will provide an overview on motion synthesis for both game- and movie-characters.
The concept of agents is relatively new in computer science. We will try here, to give the reader a short overview on what agents are and how they work.

Doyle’s work on strategic directions in AI [30] contains a section covering agents:

“Building integrated agents that perceive and act in extant complex and dynamic environments requires integrating a wide range of subfields of AI and computing research. (...) Many of the major areas of AI and computing research play essential roles in work on robots¹, from planning, sensing, and learning to high-performance numerical computing and interacting with multiple databases across networks. Robots working in informational environments require little investment in additional expensive or unreliable robotic hardware, since existing computer systems and networks provide their sensors and effectors. (...) Scaling the operation of autonomous robots to more complicated tasks (...) requires further integration of perception, action, and reasoning. (...) The marriage of these abilities aims to produce robots that combine the high-level programmability of traditional AI systems with the fault tolerance of current autonomous robots.”

In order to find a definition for an agent, we consider Wooldridge’s remarking [126]:

“(...) the question what is an agent? is embarrassing for the agent-based computing community in just the same way that the question what is intelligence? is embarrassing for the mainstream AI community. The problem is that although the term is widely used, (...) it defies attempts to produce a single universally accepted definition.”

Nevertheless, we’ll try to find some definition or notion for agents. Then, we classify the environments in which agents are used and we’ll introduce some concepts for autonomous agents. After that, we discuss game agents as a special type of agents.

5.1 Definition

Despite the above statement, Wooldridge still considers the question and presents two notions of agency, a weak one and a strong one.

1. Doyle refers to both hardware-based and software-based systems as robots.
A Weak Notion of Agency

According to Wooldridge, the perhaps most general way in which the term agent is used is to denote a system, be it hardware- or software-based, with the following properties [126]:

- **Autonomy**
  Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal states.

- **Social Ability**
  Agents interact with other agents (or possibly humans) via some kind of agent-communication language.

- **Reactivity**
  Agents perceive their environment (which may be the physical world, a user via a graphical UI, a collection of other agents, the internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it.

- **Pro-Activeness**
  Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the *initiative*.

Based on that, Wooldridge also presents a definition of softbots (software agents) by Etzioni [31]:

“A *softbot* is an agent that interacts with a software environment by sensing commands and interpreting the environment’s feedback. A softbot’s effectors are commands meant to change the external environment’s state. A softbot’s sensors are commands meant to provide (...) information.”

A Stronger Notion of Agency

Wooldridge also presents a stronger notion of agency:

“(...) mean an agent to be a computer system that, in addition to having the properties identified above, is either conceptualized or implemented using concepts that are more usually applied to humans. (...) Some AI researchers have gone further, and considered *emotional* agents.”

5.2 Environments

An important issue when constructing an agent is the environment in which the agent will act. [99] lists some principal distinctions that can be made upon environments:

- **Accessible vs. inaccessible**
  If the agent can access the complete environment it is said to be accessible. With that, an agent does not have to maintain an internal state to keep track of the world.

- **Deterministic vs. nondeterministic**
  If the next state of the environment is completely determined by the current state and the chosen action, the environment is deterministic. In such an environment, an agent does not have to worry about uncertainty if it is accessible, too. Inaccessible environments are not implicitly nondeterministic.

- **Episodic vs. nonepisodic**
  In an episodic environment, the agent’s experience is divided into “episodes”. Each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself. Episodic environments are much simpler because the agent does not need to think ahead.
• **Static vs. dynamic**
  If the environment can change while an agent is deliberating, then we say the environment is dynamic for the agent; otherwise it is static. Static environments are easy to deal with because the agent does not need to keep looking at the world while it is deciding on an action, nor need it worry about the passage of time.

• **Discrete vs. continuous**
  If there are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete. Chess is discrete while taxi driving is continuous.

Based on this classification, an inaccessible, nondeterministic, nonepisodic, dynamic, and continuous environment\(^1\) seems to be the hardest case. And even though some actions in the real world seem deterministic, for practical purposes it has to be seen as nondeterministic because of eventual, unpredictable events.

### 5.3 Autonomous Agents

As stated in the overview of this chapter, an agent is something that perceives and acts in its environment based upon the perceived information. We split an agent into an architecture and an agent program, which maps the percepts to some actions. An **ideal agent** is one that always takes the action that is expected to maximize its performance measure, given the percept sequence it has seen so far. An agent is **autonomous** to the extent that its action choices depend on its own experience, rather than on knowledge built in by the designer [99].

We will now shortly introduce three different types of agents which make use of the presented techniques. **Reflex agents** respond immediately to percepts, while **goal-based agents** act so that they will achieve their goal(s). Finally, **utility-based agents** try to maximize their own “happiness” which is expressed with a so called utility function.

In principle, an agent program mainly consists of the following tasks which are executed repeatedly:

1. **Perception**: The agent gets actual information from its environment to sense its new situation after the last action.
2. **Inference**: The agent infers with respect to its percepts about the world and what has to be done.
3. **Selection**: The agent selects one or more actions considering the possible outcomings of step 2.
4. **Acting**: The selected action is performed.

Lent refers to the inference mechanism as the central component of a AI engine because it sets forth constraints that the other components must meet. Further, the job of the knowledge machine is to apply knowledge from the knowledge base to the current situation to decide on internal and external actions. The most characteristic details of an inference machine are how it implements the think step of the decision cycle and any internal actions of the act step. [Lent99]

The inference mechanism can be very simple for a reactive agent. It is just the selection of a “if ... then ...” rule according to the sensory input. A more sophisticated approach uses internal memory to represent the current state or a goal state which has to be reached. Additionally, the knowledge base can contain scripted actions which denote a sequence of basic actions which are executed after the script-action has been selected.

---

1. For example, the real world fulfils all these conditions.
Usually, the inference mechanism is divided into hierarchical ordered levels. For example, Lent’s AI engine for game agents uses three levels: A the top level, there are goals or modes of behavior, the second level represents the high-level tactics to achieve the top-level goals, and the lower level contains the steps and sub-steps, called behaviors, used to implement the tactics [118].

Nareyek presents a classification of autonomous agents with respect to computer games. First, there is the simple reactive agent, then triggering agents with several states, and deliberative agents with goals and plans. Furthermore, hybrid agents and anytime agents are classified [80].

Here, we will discuss first the different architectures for agents and then have a deeper look into the requirements for game agents.

### 5.3.1 Simple Reactive Agents

A reactive agent\(^1\) (Fig. 5.1) responds immediately to its percepts. The decision is based on so called *condition-action rules*, which are simple if-then relations. Humans have many such reflexes, for example closing the eyes when something is approaching them. The whole knowledge of the agent is then encoded into these rules.

Reynold’s flocks, herds, and schools [94] provide a stunning approach for group behavior and path planning in flocks based on reactive behavior. The entities do not hold an internal state but only react to their environment according to four simple rules: Separation, alignment, cohesion, and avoidance. This algorithm has been used in many games and movies and emerges a natural movement behavior in a group of animals.

Although such agents can be implemented very efficiently, their range of applicability is very narrow. Even for very simple environments, the need for an *internal state* arises to keep track of specific information, when the complete access to the environment is not guaranteed (Fig. 5.2). Even Sims’ evolving virtual creatures [103] with a very simple action selection mechanism need internal states to have the possibility to distinguish several possible states.

---

1. Sometimes also referred to as reflex agent.
5.3.2 Goal-based Agents

Knowing about the state of the environment is often not enough to decide what to do. If the agent is extended by some sort of goal information, the agent can combine this with the possible outcome of its actions in order to choose actions which lead towards the goal state. The resulting agent architecture of a goal-based agent is shown in figure 5.3. As we have seen, the agent now needs a search or planning unit to find the appropriate sequence of actions. Although the goal-based agent can be less efficient, it is far more flexible. We can specify a certain goal and the agent adapts its plan based on that goal. In dynamic environments, replanning can sometimes be necessary because of changes in the environment.

Nareyek describes goal-based (or deliberative, as he calls them) agents and their requirements with respect to game agents as follows:

“The goals and a world model containing information about the application requirements and consequences of actions are represented explicitly. An internal refinement-based planning system uses the world model’s information to build a
plan that achieves the agent’s goals. (...) A great deal of research has been done on planning, and a wide range of planning systems have been developed, e.g. STRIPS, UCPOP, Graphplan and SATPLAN. The basic planning problem is given by an initial world description, a partial description of the goal world, and a set of actions/operators that map a partial world description to another partial world description. A solution is a sequence of actions leading from the initial world description to the goal world description and is called a plan. The problem can be enriched by including further aspects, like temporal or uncertainty issues, or by requiring the optimization of certain properties. (...) The problem with deliberative agents is their lack of speed. Every time the situation is different from the one anticipated by the agent’s planning process, the plan must be recomputed. Computing plans can be very time-consuming, and considering real-time requirements in a complex environment is mostly out of the question.\[80\]

In research, for example, Kuffner presents goal-directed navigation using path-planning in real-time [57].

### 5.3.3 Utility-based Agents

Only dealing with goals will not generate high-quality behavior. If a goal state can be reached by multiple action sequences, there is no specification about which way to take. The need for a more general performance measure arises. As we have seen, the utility theory provides a good solution to this problem. The utility function has to provide an appropriate trade-off between concurring goals and has to provide a way in which the likelihood of success can be weighed up against the importance of the goals. The agent architecture concerning this approach is known as the *utility-based agent* (Fig. 5.4).

![Utility-based agent](image)

**Figure 5.4:** An utility-based agent

[99]

### 5.4 Game Agents

As we have seen in the previous chapter, games are interactive and user-driven. Placing agents into a classical game environment has several implications. While the first restricts the
5.4 GAME AGENTS

ETH Zurich, CS Technical Report #377, Institute of Scientific Computing

processing time for an agent, the second requires some sort of communication if the agent is another character or a controlling mechanism if the agent is the game-character itself.

The environment of games is accessible, nondeterministic, and nonepisodic. For single-player games, it can be static, while for distributed multi-player games, it can be highly dynamic depending on the implementation. Also, games can be either discrete or continuous; while a jump&run game is discrete, a flight-simulator is continuous.

Because of the accessible environment, the perception part of the decision cycle is not very tough. In opposition to real-world perception of robots, game agents can access the whole description of the world. Therefore, game agents often lack a certain plausibility when, for example, they can see through walls or hear noises from another room with closed doors. Therefore, the perception system has to implement a filter function which should separate the perceivable objects from the others. Often, “cheating” has been used to obtain somehow more intelligent agents because their inference mechanism had not enough resources (knowledge or CPU time). Lent states that the cost for having realistic agents is that these will require more knowledge and better tactics and behaviors to challenge human opponents [118]. Perception will be discussed further in chapter 8.5.

Another problem arises with the very hard real-time constraints of games. Because the game should always be responsive to user input and provide a decent frame-rate, the inference mechanism can not use very much CPU-time. As Nareyek states:

“What we need is a continuous transition from reaction to planning. No matter how much the agent has already computed, there must always be a plan available. This can be achieved by improving the plan iteratively. When an agent is called to execute its next action, it improves its current plan until its computation time limit is reached and then executes the action. (...) For short-term computation horizons, only very primitive plans (reactions) are available, longer computation times being used to improve and optimize the agent’s plan. The more time is available for the agent’s computations, the more intelligent the behavior will become. Furthermore, the iterative improvement enables the planning process to easily adapt the plan to changed or unexpected situations. This class of agents is very important for computer games applications.” [80]

In fact, what Nareyek is talking about are anytime agents based on anytime algorithms (chapter 8.6).

Chapter 8 will present a more detailed discussion of various aspects for game agents and their environment. Here, we will present shortly some recent research with respect to game agents and related issues:

Lent discusses some issues about an artificial intelligence engine for game agents. Nareyek also discusses some requirements for intelligent game agents [80]. Blumberg et al. [16][50] present a brain architecture with which they can simulate smart creatures such as animals in a game-like environment. Furthermore, Isla discusses new challenges for character-based AI in games [49]. John E. Laird discusses interactive computer games as a whole with respect to human-level AI [59].

Also, robotic soccer research has recently provided some noteworthy results which could be adapted for game agents not only in sports games. For example, Coelho presents coordination strategies in simulated robot soccer [23] and Stone presents an architecture for action selection in robotic soccer [109].
5. AGENTS
The modeling of an agent’s behavior and internal representation is very difficult and there is no single solution on how to do it because of the diversity of natural behavior within animals and humans. First, we’ll try to define the requirements and aspects of behavior modeling with respect to game agents which can be animals or humans. Then we’ll present an overview of different research topics and how other researchers have tried to model some specific behavior which has been introduced in the first part. The last part covers cognitive modeling, a novel method for high-level behavior modeling.

6.1 Behavior

During the last ten years, the simulation of animal and human behavior became more and more a research topic. Starting with Reynolds [94] in 1987, where the simulation of flocks, herds, and schools were presented, many other researchers began to investigate on behavior.

Individual behavior has been researched in many different areas, mainly animals. The motion of simple animals such as snakes and worms [75] were followed by kangaroos [89], moving synthetic evolved animals made from simple connected boxes [103] and fishes, snakes and dolphins [39]. An evolving ant colony [25] and the ant’s pheromone communication [83] has also been explored. Fishes were investigated about their physics, locomotion, and perception [115], perception and learning [110], their cognition [35], and collective behavior [130]. [35] also presented the predator and prey behavior of dinosaurs, which was also investigated by [115] for fishes. The social behavior, emotions, and learning of wolves in a social group [112] is one of the recent research results.

As one of the first games with artificial creatures, the commercial game Creatures [36] presents Norns, the inhabitants of a virtual world, who evolve with a biochemistry for hormone-and energy-modelling and behavior. The last years presented new directions, such as motivations and emotions [20][37], behavior with personalities [10], and universal characters [70]. Various brain architectures were discussed, e.g. [50] presents a layered brain architecture, [118] an artificial intelligence engine and more. We will discuss this topic in chapter 8.

Shehory has evaluated modeling techniques for agent-based systems [100] and states that a modeling technique should adhere to the following:

1. Preciseness. The model should be unambiguous so that the users cannot misinterpret it.
2. Accessibility. Both experts and novices should be able to understand the model.
3. **Expressiveness.** The modeling technique should present the structure, the knowledge, the
data flow, the control flow, and the interaction with external systems.

4. **Modularity.** Existing parts should not be modified when new specification requirements
arise.

5. **Complexity Management.** The model should support the easy integration of both high-
level and low-level requirements.

6. **Executability (and testability).** The model should be implemented in either a prototyping
system or a simulation capacity in order to demonstrate the intended behavior.

7. **Refinability.** The refining of the model should be provided and possible within a dedi-
cated path.

8. **Analyzability.** The model should support tools to check the internal consistency or impli-
cations of the model.

9. **Openness.** A modeling technique should provide a good basis without coupling it to a
specific architecture, infrastructure or programming language.

We will keep these thoughts in mind while concentrating on game agents and their behavior
modeling methods and requirements.

John E. Laird has written a short paper on design goals for autonomous synthetic characters
with regard to computer games [60]. Therein, a large part is devoted to the behavior of the
agents wherefrom we will quote here some parts:

“The variety and complexity of an agent’s behavioral 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 spe-
cific behaviors that the character will use and how they combine.”

“The most obvious and vague goal is to provide great gameplay. (...) This leads
to the conclusion that the Holy Grail for character designers should be to produce
characters whose behavior is indistinguishable from human 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 inexpen-
sive approximations, or as I like to say, ‘Cheat without getting caught’.”

Burke et al. seek for creatures with a robust, reactive, adaptable, honest, expressive, sensible,
and scalable behavior [16]. Furthermore, Laird subsumes capabilities which define a dimension
of variability where research would be appreciated [60]. The following points are taken from
this paper:

1. **Human sensing.** The character should not have superhuman abilities, such as seeing
through walls and should not have subhuman abilities, such as hot hearing someone run-
ning through a quiet room. Unfortunately, realistic models of sensing can be very diffi-
cult and computationally expensive to implement.

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.

3. **Human-level reaction times.** Humans do not respond instantaneously to changes in their
environment, nor do they take arbitrarily long to respond.

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.
5. **Memory.** Most current game characters can be described as, “out of sight, out of mind”, which leads to very inhuman-like behavior. Thus, complex characters need to maintain memories of the world and have a model of how the world changes over time.

6. **Common sense reasoning.** This is the most dreaded of all classes of reasoning in AI because it is completely ill-defined. The only viable approach seems to be to determine what knowledge is necessary for the tactics you wish to encode in your character and the encode the necessary common-sense knowledge to support those tactics. But this is insufficient if you wish to develop characters that develop their own tactics.

7. **Goals.** Characters must have a purpose, some goal they are trying to achieve. Often, there will be multiple goals, and the character has to decide which one to pursue or which actions 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.

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.

11. **Learning.** For most games, learning can be avoided. It is only an issue for characters that have prolonged interactions with the human players. Learning is sometimes difficult to implement and can lead to unexpected and undesirable behavior unless carefully controlled.

12. **Unpredictable behavior.** As with all of the capabilities covered here, 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, if a character has a sufficiently broad and rich set of fine-grained responses, its behavior may be very difficult to predict.

13. **Personality.** Personality can be thought of 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. One theory (Barrick and Mount, 1991) is that there are five factors influencing the style: openness, conscientiousness, extroversion, agreeableness, and neuroticism.

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 impacts 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 except when it goes down to zero (and the character dies). Other stressors include fatigue, heat, chemicals, radiation, ...

What kind of research has been done regarding some of these capabilities will be part of the following section. We will focus on the ones that do not seem to be easy to implement and where no appropriate or well-founded model exists and therefore research is still in progress.
6.2 Character Modeling

This chapter will discuss the implementation specific details of several earlier presented workings. The general agent architectures described in the previous chapters is widely used as a base for further extensions and enhancements which allow a more sophisticated behavior, peculiar characteristics, or accelerate the computation. This is mostly redundant to previously presented work in order to obtain a subsumptional list of the modeling approaches.

Sensing

Sensing and perception can be thought to be two different things. Sensing is the action of getting a stimulus while only the perception decides what it is and what the further meaning of this stimulus can be. Perception in games is usually no problem because of the completely accessible environment. For robots and other agents in open environments this task can become one of the most difficult ones. Just consider visual perception or speech recognition as examples.

Burke et al. present a biologically inspired brain architecture which clearly distinguishes between these two skills resulting in what they call sensory honesty [16]. The sensor system processes each sensed data so that it appears as it would originate from the creature’s point of view. This can include removing, culling or transforming it.

Once the stimulus has been sensed, it can be perceived which is principally a classification problem. Another important part of this architecture is proprioception. This type of self-perception is important to simulate the awareness of emotional state and of self-action.

Another form of sensing is anticipation. Laird presents an agent for Quake II [61] which uses the Soar architecture [106] for cognitive calculations. The agent anticipates the opponents actions by internally predicting them based on its own tactics.

Zhang et al. present a template-based and pattern-driven approach to situation awareness and assessment in virtual humans [131]. They represent situations as tree-structured organizational and spatial structures. Each situation has also a list of possible actions which reduces the search costs to a fraction of the costs for an unstructured search. With that, they can reduce the perceptual load of virtual pilots significantly, reducing the focused attention based pilot’s response time by more than 50 percent.

Closely related to emotions are expectations and prediction, hence emerging confusion and surprise. Expectations originate from discontinuous perception, such as a disappearing object. Expectations are covered by Kline and Blumberg who describe how to improve reactive agents with respect to believability and robustness by enabling the generation of short-term, observation-based expectations and react appropriately to violations of those expectations [54]. Burke et al. from the same group states that this kind of behavior is uniquely adopted by human brains [16]. Therefore, they introduce object persistence. When a percept’s data is not observed, its value is predicted based on previous observations as if it had been observed. This prediction depends on the kind of percept and therefore on the type of data it carries. Then, a function approximation technique or a extrapolated position are determined and applied to the object. Also, predictions imply expectations, and a creature that acts in anticipation of a future event can be said to have intentions.

Actions

Based on its perception, the agent has to choose an action which it is going to execute. In the simplest case, this is just an update of a state, be it in the agent’s knowledge base or the world model. We will not consider this case but have a look on locomotion actions. This will be covered deeper in chapters 8 and 7. Here we just give a short overview on natural locomotion in research projects. First, the generation of artificial locomotion has been developed for artificial
fishes by Tu and Terzopoulos [115] which uses three different parametrized motor controllers to steer a virtual fish in a dynamic environment. A different approach uses neural networks to control the effectors of fishes, parking cars or a lunar lander [40].

Rose et al. present a technique to generate smooth transitions between segments of human body motion based on a combination of spacetime constraints and inverse kinematic constraints [98]. They applied this technique to a human body model with 44 degrees of freedom and created basis motions, cyclic data, and seamless motion transitions. Characterizing motions by emotional expressiveness or control behavior has been the next extension to this system [97]. This has been done with parametrized motions called verbs and their controlling parameters called adverbs. Then, verbs can be combined with other verbs to a verb graph, with smooth transitions between them.

Another approach is the parametrized action representation (PAR) by Badler et al. to bridge the gap between natural language instructions and the virtual agent [6]. A PAR gives a complete description of an action and will be described in detail in chapter 8. This approach is also used by Bindiganavale et al. for simulating dynamically altering behavior [9].

**Spatial Reasoning**

The task of path planning will be discussed in detail in chapter 8.7. The prediction of moving objects has been discussed in the sensing section.

**Memory**

Burke et al. state that psychologists distinguish between different kinds of memory [16]. Procedural Memory is the label given to skill learning - for example, learning to play piano, or throw a baseball. Another type of memory is Declarative Memory, which allows us to remember facts, such as “my name is Chris”, or “the pen is on the table”. Declarative Memory is further broken down into Long-term Memory, which stores important facts and events, and Working Memory, which tracks the state of the environment that is relevant to the immediate task.

**Common Sense Reasoning**

Leslie G. Valiant describes an architecture for designing systems that acquire and manipulate large amounts of unsystematized, or so-called commonsense, knowledge [117]. This rather theoretical approach makes explicit requirements on the basic computation tasks that are to be performed. The architecture is designed to make these computationally tractable even for very large databases. Attribute-efficient learning algorithms, which allow learning from few examples in large dimensional systems, are fundamental to the approach.

**Goals**

Goal-oriented behavior has been discussed in chapter 5.3.2 with respect to agents. Additionally, several researchers have investigated on motivations.

Canamero [20] and Burt [17] model motivational behavior in intelligent agents. Tomlinson et al. use motivations for his expressive autonomous cinematography to model the desired shot elements [113]. Yoon et al. present motivation driven learning where the motivation system composes of two parts: a drive system and an affect system [128]. The drive system includes drives depending on internal states and sensory induced drives. The highest level of the affect system composes of three parts, which represent valence, stance, and arousal.

**Tactics**
Van Lent and Laird present a human-like Quakebot\(^1\) based on the Soar\([106]\) architecture, which uses goals, tactics, and behaviors in a hierarchical order with tree levels\([118]\). The top operators represent the agent’s goals or modes of behavior. The operators at the second level represent the high-level tactics the agent uses to achieve the top level goals. The lower level operators are the steps and sub-steps, called behaviors, used by the agent to implement the tactics. In any cycle, the system can select an operator to be active at each level of the hierarchy. Laird extended the above system and added anticipation-based tactics to the Quakebot[61]. These tactics include collecting, attacking, retreating, chasing, ambushing, and hunting. Selecting one tactic entails the selection of appropriate actions in a hierarchical manner. Provided such a goal, the agent tries to anticipate the opponent behavior in order to get an advantage over them.

**Planning**

The task of planning has been introduced in chapter 3.5.

**Communication and Collaboration**

Communication has been covered in section 3.7. We consider social behavior and collaboration modeling as a sub-problem of communication and will present some of the research that has dealt with these issues.

Doyle et al. list sociology, economics, politics, linguistics, and philosophy as academic sources for collaborative behavior[30]. Four primary contexts are mentioned to be important for AI: understanding dialogue, constructing intelligent assistants, supporting collaborative and group work, and designing “artificial societies.” We will concentrate on the last aspect, which “introduces a design perspective into economics by seeking to tailor the preferences of agents, the protocols of interaction, and the environmental constraints so as to automatically yield collaboration, noninterference and other desirable properties of group behavior”.

Beginning with Reynold’s flocks, herds, and schools[94], research has begun to study the collective behavior of agents. Kube and Zhang try to control multiple behavior-based autonomous robots[56]. Based on observations made from the studies of social insects, he proposes five simple mechanisms used to invoke group behavior: a common task and non-interference, a follow behavior to keep the group together, environmental cues to invoke the group behavior, allowing group behavior only in groups, and autostimulation to permit a single agent to invoke a behavior within the group. The work by Demazeau presents an interaction language and several interaction protocols[29]. Zaera et al. tried to simulate a schooling behavior using neural network-based controllers but never succeeded in finding a suitable evaluation function for schooling[130]. Martinoli et al. modeled and implemented a biologically inspired calcites behavior[73]. This means, that there is no central organizational unit which controls the individual agents or robots. Weiss describes a novel algorithm for activity coordination that combines joint planning and joint learning[122]. Daunenhahn puts research on social intelligent agents in the broader context of how humans perceive and interact with the social world[27]. Phylogenetic (evolutionary) and ontogenetic (developmental) issues are discussed with respect to the social origin of primate and human intelligence and culture. A theory of empathy is presented that is based on research on the primate social brain. As mentioned earlier, Parunak et al. presented a pheromone-inspired approach to invoke collective behavior in a decentralized fashion[83]. Most recent research covered coordination strategies in simulated robot soccer[23], social role awareness in animated agents[88], social reinforcement learning[48], and social behavior in a group of wolves[112]. The last one is a continued work based on Burke et al.\[16\]

---

1. A Quakebot is an artificial agent in the famous game Quake.
which extends the therein presented system with the ability to have and express emotional states, to learn to use behaviors in novel social contexts, and to form context-specific emotional memories.

**Learning**

As we have seen in the AI chapter, learning has been studied for a long time now and it is possible to learn specific reactions in a clearly defined environment. Several researchers have tried to simulate the learning of locomotion, such as Grzeszczuk et al. for a dolphin, a parking car, and a lunar lander [40], Terzopoulos et al. for fishes [110], and Sims et al. for virtual creatures made of simple boxes [103].

More general approaches have studied motivation driven learning [128], collaborative learning [122], social reinforcement learning [48], and hierarchical multi-agent reinforcement learning [71]. One conspicuous paper deals with the opposite of learning, that is to say forgetting [69].

**Personality**

Modeling personality is a very difficult problem due to the existence of only very few appropriate models from psychology or social psychology which even do not suit into a computational background. Only during the last years, research has started dealing with personality.

Blando et al. start with modeling behavior with personalities with respect to object oriented programming [10]. Their goal is to encapsulate behavior on its own and they propose the concept of personalities as a design and programming artifice to model stand alone behavior which enables its reuse in different places in an inheritance hierarchy. Here, personality is rather a programming paradigm than the model of a psychological effect.

Wilson presents the Artificial Emotion Engine™, which is designed to generate a variety of rich emotional behaviors for autonomous, believable, interactive characters [124]. He divides emotions into three layers of behavior. At the top level are reactions or momentary emotions. The next level are moods, more precisely the cumulative effect of the above mentioned momentary emotions. Underlying both of these layers and always present is the personality, the general behavior when no other layer is active. The personality is defined as “the genetic and environmental differences in the brain structures of individuals.” The model defines a characters personality as an area in a Cartesian space with the axes defined by Extroversion, Fear, and Aggression (EFA-Space).

Broersen et al. introduce the Beliefs-Obligations-Intentions-Desires (BOID) architecture [13]. This architecture resolves conflicts between these components and the way in which these conflicts are resolved determines the agent’s type or personality: “In a single-minded or stable agent intentions override desires and obligations; in an open-minded or unstable agent desires and obligations override intentions; in selfish agents desires override obligations and in social agents obligations override desires.” The paper lists all possible conflicts and how these can be resolved with respect to an agents type.

**Emotions**

As stated by Wilson, emotions are a product of the personality [124]. Many researchers have tried to find suitable models for emotions and their impact on actions and behavior. As personality, this is a very young topic in computer related research.

Reilly’s PhD thesis presents a large and diversified architecture for believable emotional and social agents [93]. Many different emotions have been modeled to achieve a believable environment for interactive plots.
Canamero modeled motivations and emotions for two-dimensional agents where the emotions are triggered by certain events, produce a specified amount of hormones, and their physiological effects [20]. Six different types of emotions are used: Fear, anger, happiness, sadness, boredom, and interest. Velasquez from the same research group describes an agent architecture that integrates emotions, drives, and behaviors, and that focuses on modeling some of the aspects of emotions as fundamental components within the process of decision-making [120]. Emotional memories are used to obtain a possibility to choose actions also depending on the emotional state and therefore driving the inference process at an emotion-dependent direction.

Ventura proposes a model for an agent whose functioning and reasoning is based on emotions [121]. A complex, unfiltered, structured representation termed cognitive image, and a simple, basic, built-in one termed perceptual image from the used model.

Bazzan et al study the Iterated Prisoner’s Dilemma (IPD)\(^1\) which has been used as a paradigm for studying the emergence of cooperation among individual agents [8]. They extend the problem by adding and modeling emotions which are divided into three categories. First, attraction is bounded to objects and includes the emotions love and hate. Second, consequences of events are classified with respect to others as happy-for, resentment, gloating, or pity, and with respect to the agent itself as hope (which can implicate satisfaction or disappointment), fear (implicating confirmed fear or relief), joy, and distress. Furthermore, pride, admiration, shame and reproach are also included in their model.

The Artificial Emotion Engine by Wilson models joy, sadness, fear, anger, surprise, and disgust as possible emotional reactions [124]. The above mentioned personality modulates these reactions. The reactions are triggered by signals of punishment and reward and the levels of the motivational needs.

Gratch and Marsella try to model and simulate engaging and believable characters to populate virtual worlds [37]. Isla and Blumberg also emphasize the influence of emotions on decision-making, futural visions, planning, and even perceiving, since emotions are not simply coloring the physiological effects such as motion and stance. Secondary emotions, such as curiosity, are considered to go beyond the traditional happiness/sadness models to express more subtle aspects of the character’s mental state [49].

The pack of wolves by Tomlinson also shows emotional behavior [112]. For this project, a dimensional and a categorical approach of emotions have been used. The first maps a range of emotional phenomena onto explicitly dimensioned space based on the Pleasure-Arousal-Dominance model. The latter separates emotional phenomena into a set of basic emotions – for example fear, anger, sadness, happiness, disgust, and surprise. Emotions also influence the style of taken actions based on the ‘Verbs and Adverbs’ model [97]. Also, emotions are essential for the creatures’ social interactions and therefore they introduce context specific emotional memories (CSEMs), which cause the wolve to return to a emotional state similar to the one it experienced on previous encounters with specific environmental stimuli (e.g. a dominant wolf). This CSEM stores also a value of confidence to enable appropriate behavior.

As we have seen when discussing perception issues, expectation and prediction are closely related to emotions and emerge confusion and surprise. Blumberg et al. have introduced a system which predicts (i.e. extrapolates) its perceptions and compares this prediction with the

---

1. IPD is a non-zero-sum game to analyze cooperation. Two players can choose between either “cooperate” or “defect”. If both cooperate, each gains. If one cooperates and the other defects, the cooperative player will lose while the other wins more than when both are cooperating. If both are defecting, nobody gets anything.
perceived data. Creatures with expectations can then be surprised, relieved, disappointed, confused, frightened, tricked, teased, misled, taunted, and so on.

6.3 Cognitive Modeling

John Funge’s famous Siggraph paper [35] introduces cognitive models which “go beyond behavioral models in that they govern what a character knows, how that knowledge is acquired, and how it can be used to plan actions.” To help building cognitive models, the cognitive modeling language CML has been introduced, which enables the definition of domain knowledge specified in terms of actions, their preconditions and their effects. Additionally, the characters behavior is directed in terms of defined goals. Therefore, the animator just has to specify a behavior outline and the character will, through reasoning, automatically work out a detailed sequence of actions satisfying the specification.

CML is built on situation calculus which describes changing worlds using sorted first-order logic and maps precisely onto it. It consists of situations, changing properties known as fluents, primitive actions, precondition axioms, and effect axioms. CML assumes a closed world where effect axioms enumerate all the possible ways that the world can change. Therefore, it is not necessary to define effects that do not happen. Using complex actions, which are a set of recursively defined operators, the search tree can be pruned, and therefore the exhaustive search can be speed up.

The paper also introduces interval-valued epistemic fluents (IVE fluents), which describe the uncertainty about a quantity which can change over time. Therefore, sensing corresponds to making the intervals narrower and has only to be performed when the current interval is too large, i.e. the sensor value is too imprecise.

The mightiness of this approach is demonstrated with automated cinematography, where the camera itself is a cognitive character and chooses the appropriate shot direction and clipping for a dialogue. Also, a prehistoric world populated by dinosaurs is presented, where multiple raptors are herded by one T-Rex, which behaves like a sheep-dog. The last example shows a physics-based undersea world with a merman and a predator shark. The merman can use its knowledge to successfully escape from the shark although he is slower than its adversary.

In the subsequent paper, Funge discusses (not only) the above approach for its usability in computer games and other interactive entertainment [34]. A language is postulated, which

- allows to define the behavior of the characters in terms of their mental state, e.g. knowledge, goals, intentions, etc.
- contains primitives for communication which automatically update characters’ mental states appropriately when communication occurs.
- ensures that the characters can introspect their mental states, e.g. if an agent has a goal to do something then the agents should know it has that goal.
- gives the characters explicit knowledge of the physical environment, so they can construct and execute plans to achieve their goals in the environment when necessary.

Beside that, the requirements for a cognitive, interactive, and real-time multi-character system are encountered:

1. The characters are constrained to search for plans of less than a certain length. This necessitates the formulation of a goal that expresses partial success towards the ultimate goal, and periodic re-planning when the current plan becomes outdated. Providing problem-specific heuristics, the above mentioned complex actions help to increase the length of the plan.
2. There is an underlying reactive behavior substrate that can act as a backup system if a plan cannot be computed in time. Although the backup system lacks the sophistication to move the character closer to fulfilling its goals, it does prevent the character from doing anything stupid.

3. Planning is not necessary at every frame. The reactive system is responsible for executing the plan and monitoring the execution to check whether the plan’s assumptions have significantly diverged from what is transpiring.

L. Chen presents a logical approach to high-level agent control [21] similar to Funge’s. Both try to define and implement a cognitive model for animated agents, but Chen’s work is based on event calculus (EC), while Funge’s one is based on situation calculus. Although the two formalisms share the basic ontology of atomic actions and fluents, EC has the advantage of representing actual actions, in particular, actions with duration. Another feature of the EC is its ability to assimilate a narrative, i.e. the description of a course of events, and adjust the effects of actions and the time-line of the narrative as it becomes more and more precise in an additive only fashion.
The task of animal or human motion synthesis is a very challenging one because the result should look natural, smooth, authentic and believable. Since we are very sensitive to detecting anomalies in everyday physics and motions are usually rather complex, most of the introduced techniques are unsatisfying.

For real-time environments, the generation of motions can be classified into two basic classes: Prerecorded motion playback and procedural generation of motions in a simulated physical environment. The first approach uses a technique called motion capturing to obtain the desired position sequence of the body for a certain motion or motion cycle. The other uses spline curves or similar techniques to model a smooth motion trajectory in spacetime between previously defined postures.

Emotion-based motion generation is also discussed at the end of this chapter to risk an outlook on what might be the next competition. Since our daily motions not only consist of a task but also of a shape, the impression of a movement can tell much more than the movement itself. Consider a walk movement which in one extrema could be a sad, slow, and sluggish style where the other could be happy, frisky, and faster one. Both can be considered as walking, but there is a huge difference in perception.

7.1 Motion Capturing

Motion capturing is one of the simplest methods to obtain a realistic, smooth and natural looking motion. Motion sequences are recorded using actors wearing suits with marks on the most important joints (head, shoulders, elbow, hip, knee, feet, etc.). These actors are then captured while acting in the desired manner. Further, the traces of the marks in space are reconstructed from the recorded images and can be applied to a synthetic model of a human being. This technique can also be applied to artificial animals (e.g. a dinosaur) where people don’t know the exact way of their motions. Then, the actor has to imitate the animals motions.

Recent movies have extended this method to several concurrent actors to facilitate the composition of fights and other scenes where multiple interacting actors are placed into an artificial environment.

Motion capturing can produce the most natural looking movements because they are natural. On the other hand, captured motion can only be applied to scenes which do not differ too much from the original setup. Imagine a person going upstairs. If the motion is captured on a steeper
than in the target scene, it would be very difficult to adapt the original motion onto the new requirements. Humans can easily determine if a motion of another human lacks naturalness because of the habituation to human agitation. Therefore, the recorded motion has to fit exactly into the desired scene with its environment and obstacles.

### 7.2 Procedural Generation of Movements

Different scenes require different movements. Therefore, one could think about procedurally generating the required positions of the main body parts to reach the impression of a proper movement. In addition, an interactive game should not make use of prerecorded motion because the sequence of movements is not predefined. Anyway, almost every game uses predefined motions to keep the computational expenses to a minimum, because this goal is very hard to achieve. As we have stated, movements differ every time due to the emotional state of the actor.

#### 7.2.1 Kinematics

Without any physical constraints, procedural generation of movements is almost impossible. Consider an arbitrary arm movement and try to model the movement of every part from the shoulder to the front-most thumb part as functions for positions and angles. When combining these approximative movements the parts usually won’t fit exactly to each other and holes will emerge. Therefore, the first approach towards better movements is to use kinematical constraints such as joints and limbs.

Kinematical models use a hierarchical skeleton to represent the structure of the animated object. Joints are represented as edges in a tree, whereas limbs are represented as nodes. Therefore, a limb can have multiple joints which connect other limbs to it. When re-positioning a limb node, all children and subchildren are updated according to the hierarchical structure.

Like that, the forward generation of a simple movement becomes very easy. But the generation of goal-oriented movements becomes non-trivial, and this simple approach fails because the endposition of the outmost limbs (e.g. the fingers) cannot be predicted directly.

#### 7.2.2 Inverse Kinematics / Spacetime Constraints

To solve the problem of control mentioned above, inverse kinematics is used. Here, the start and endposition are fixed and the task is to find a trajectory in spacetime which matches these constraints. Usually the constraints are termed *spacetime constraints*. Of course, the trajectory should be collision-free which adds some additional constraints to the problem.

Cohen and Ngo discuss the inverse kinematic problem and provide useful solutions [24][81]. Koga et al. present a system which manages multiple cooperating arms to move an object on a collision-free trajectory to a goal configuration [55]. Additionally, they present an approach for realistic looking human arm movement based on their planner. Rose et al. [98] present efficient generation of motion transitions using spacetime constraints which is applicable to models with many degrees of freedom such as human models. The system uses a combination of inverse kinematics for the supporting limbs and dynamics (see next section) for every other limb. Finally, Gleicher presents a system to edit pre-recorded motions with spacetime constraints [125].
7.2.3 Physics-based Movements

The kinematics based approach keeps the whole model together without breaking. But the movement is constructed only by repositioning the limbs without any physical reason. This can lead to unnatural movements and incorrect contact with the environment.

Physics-based movements rely on forces and torques which are applied to the model at any time. The resulting movement of the model is then the forward integration of the actual state in time. This process is commonly known as rigid body simulation \[41\][95]. A very important part of rigid body simulation is the detection of collisions since rigid objects cannot intersect or deform, a collision point becomes a temporary joint. The whole system then has to solve a system of equations, where the joints become equality constraints and the contact points become inequality constraints. Additionally, the coefficient of restitution and the coefficient of friction can be specified which adds some more inequality constraints to the system.

Moravanszky presents an overview of different methods to solve this system of equations and their runtimes \[79\]:

<table>
<thead>
<tr>
<th>Method</th>
<th>Runtime, n equality constraints</th>
<th>Runtime, n inequality constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. LCP</td>
<td>O(n), acyclic only</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>2. Penalty</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>3. Impulse</td>
<td>-</td>
<td>O(n)</td>
</tr>
<tr>
<td>4. Featherstone</td>
<td>O(n), acyclic only</td>
<td>-</td>
</tr>
<tr>
<td>5. Jakobsen</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>6. Barenburg</td>
<td>O(n^2)</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 7.1: Summary of Dynamics Simulation Methods by Moravansky \[79\]

As one can imagine, realistic looking natural movements are hard to generate. Even though the movements obey physical laws, the computation of the exact amount of forces and torques to apply still remains the key point. Additionally, the same problem as stated above occurs, where a specified goal configuration has to be reached which leads to an inverse dynamics problem \[47\] which we will not further discuss within this report.

Zoran Popovic presented a remarkable work at SIGGRAPH’00 related to that topic: Interactive manipulation of rigid body simulation \[86\]. This work presents a system which enables the user to interactively manipulate a physically based motion at any point in spacetime.

Many other researchers have used dynamics in their approaches for animation or simulation of agents. Auslander presents motion synthesis \[4\], Hodgins animates human athletics \[45\], while Laszlo animates balancing and walking \[65\]. Also, Sims virtual creatures evolve in a dynamics-based environment \[103\].

1. The state consists of position, velocity, acceleration along every axis and also for rotations around these axes.
7.3 Motion Controlling

For motion controlling, the divide-and-conquer approach is widely used. With that, the desired movement is decomposed into many submovements which are then somehow stuck together to form the desired overall movement. The composition of single motion pieces into a more complex movement and blending two movements together into a novel movement are both non-trivial tasks which will be discussed in this section.

Composition

The main problem for the composition is that the motion pieces often do not have the same configuration at their contact points, the motion inbetween has to be supplied by the animator or has to be computed somehow.

Raibert and Hodgins present a system for animating dynamic legged locomotion [89]. Three main controllers are used to simulate running of various characters, such as a biped and quadruped robot or a kangaroo. The first is responsible for the legs to step with exchanging support, called the hopping control. The second provides balance to regulate the running speed, called speed control, and the last maintains the body in an upright position, called posture control. Later, Hodgins presented visual impressive animated human athletics [45], where running, bicycling, and vaulting are controlled on a high level but not on a common basis.

Auslander et al. present controller-based automatic motion synthesis [4] which enables the user to couple several motion controllers for a two dimensional figure called Mr. Star-Man who consists of 5 sticks arranged like a star. Their system consists of enhanced controllers which can be started from different configurations in order to obtain a larger goal-configuration space, a composite-motion scripting language to define the fitness function for the synthesis, and a search heuristic which finds the combining trajectory in spacetime.

Tu et al. present artificial fishes [115], were the movement is generated as a superposition of three basic motion controllers. One for forward-swimming and two for turning left or right. The controllers only need one parameter, i.e. speed or angle.

Rose presents efficient generation of motion transitions using spacetime constraints [98]. This approach can be used for systems with many degrees of freedom, such as a human model. The motion transitions are generated using both kinematics and dynamics. The motion of the root of the body is determined kinematically, while the motion of non-supporting limbs is generated using spacetime constraints. The supporting limbs (e.g. the leg on the floor) are controlled using an optimization method to solve the inverse kinematics problem over the entire transition time instead of just one point in time. He later enhanced the system by adding verbs and adverbs [97] which are discussed in section 6.2.

Recently, Faloutsos et al. presented composable controllers for physics-based character animation [32]. The system is composed of a simple supervising controller which arbitrates between several controllers considered as black boxes. Each such controller provides his pre- and postconditions and a expected performance measure. With that, the main controller can select all possible controllers by their precondition when a new one is needed. The postcondition then defines a range of states for the final state of the character after execution. The performance measure is needed to detect failure at any point during execution. Using this approach, an impressive character capable of balancing, making protective steps or arm movements, and sitting up in various ways has been evolved.

Blending

Another problem is how to generate a movement with a style, e.g. walking slowly versus walking fast based on a simple walking movement. Two approaches are used to do this: First,
two distinct movements are interpolated. Here, both basis movements have to be synchronized in some way in order to prevent obliteration. Second, one extracts a single basis movement (e.g. walking) and the styles separately and applies the style onto the basis movement.

The emotion based figure animation by Unuma et al. [116] bases on Fourier principles. Fourier expansions of actual human behaviors are used to build a basis for novel movements. With this approach, the style or mood of a movement, such as walking, can be extracted and be applied to another movement, such as running. In addition, the speed, step-length, and hip-position are also modeled and can be controlled.

The style machines by Brand [12] approaches the problem with a large number of different motion captures from which common choreographic elements with a small number of stylistic degrees of freedom are learned. The learned model can synthesize novel motions as a process of inter- or extrapolation.

7.4 Other Approaches

The paper by Popovic et al. on physically based motion transformation [87] proposes a novel approach for this problem. They use motion captured data which is mapped on a very simplified human model which is not longer dynamically correct. To resolve the nonconformance, the algorithm searches for the spacetime optimization whose solution comes as close as possible to the mapped motion. With that, the model can be edited within spacetime and can be reused to generate novel motions. For example, the foot contact points can be varied in order to get a longer step-size.

7.5 Expressive Animation

While modeling and simulating artificial characters is one issue, the corresponding presentation of this behavior on the screen in an accurate, adaptable, and believable fashion is an additional task. We will have a look at some general approaches and especially at facial and body-movement expressions. Appropriate models for action representation are also presented within this chapter.

7.5.1 Principles of Animation

In 1987, John Lasseter (Pixar) presented a significant paper on principles of traditional animation applied to 3D computer animation [64]. He points out that understanding the fundamental principles of traditional animation is essential to producing good computer animation and lists eleven fundamental principles:

1. **Squash and Stretch.** Defining the rigidity and mass of an object by distorting its shape during an action.
2. **Timing.** Spacing actions to define the weight and size of objects and the personality of characters.
3. **Anticipation.** The preparation for an action.
4. **Staging.** Present an idea so that it is unmistakably clear.
5. **Follow Through and Overlapping Action.** The termination of an action and establishing its relationship to the next action.
6. **Straight Ahead Action and Pose-To-Pose Action.** The two contrasting approaches to the creation of movement.
7. **Slow In and Out.** The spacing of the inbetween frames to achieve subtlety of timing and movement.

8. **Arches.** The visual path of action for natural movement.

9. **Exaggeration.** Accentuating the essence of an idea via the design and the action.

10. **Secondary Action.** The action of an object resulting from another action.

11. **Appeal.** Creating a design or an action that the audience enjoys watching.

### 7.5.2 Research

Unuma et al. present emotion-based human figure-animation [116]. Fourier expansions of experimental data of actual human behaviors serve as a basis from which the method can interpolate or extrapolate the human locomotions. With that, transitions from walking to running and vice versa can be rendered smoothly and performed realistically. The superposition of these human behaviors is shown as an efficient technique for generating rich variations of human locomotions. In addition, step-length, speed, and hip position are also modeled, and then interactively controlled. But the achieved transitions are not as natural as expected, since human locomotion is not just a morphing effect between some atomic locomotions.

Vaughan’s paper on understanding movement [119] addresses the relation between emotions and movement in theatre and psychology in order to provide the designers of movements the basic characteristics of movements. He points out four different characteristics to be important in computer animation: Path, area, direction, and speed.

Almost at the same time, Norman Badler [6] and Charles Rose [97] tried to formulate a parameterized action model in order to bridge the gap between natural language instructions and the virtual agent carrying them out. While Badler found the Parametrized Action Representation (PAR), Rose’s model was called “Verbs and Adverbs”. The PAR includes slots for many types of information that can sometimes occur linguistically as adjuncts to the main verb phrase. These slots include spatio-temporal information such as paths (to the store, across the room), manner information that is often expressed as adverbs (quickly, carefully), and applicability and terminating conditions that can be inherent but can also be specified (until the door is open).

The “Verbs and Adverbs” model by Rose calls the motions *verbs* and the parameters that control them *adverbs*. Verbs can be combined with other verbs to form a *verb graph*, with smooth transitions between them. A combination of radial basis functions and low order polynomials is used to create the interpolation space between the example motions. Additionally, kinematic constraints are used to avoid unnatural motions. This allows the animated figure to exhibit an impressive variety of behaviors.

Chi et al. postulate that looking only at the psychological notion of gesture is insufficient to capture movement qualities needed by animated characters [22]. They advocate that the domain of movement observation science, especially *Laban Movement Analysis* (LMA) and its effort and shape components, provide valuable parameters for the form and execution of qualitative aspects of movements. They present a system called EMOTE (Expressive MOTion Engine) that applies these principles to independently defined underlying movements. LMA has five major components: Body, Space, Shape, Effort, and Relationship. Together, these components constitute a textual and symbolic language for describing movement. But only effort and shape are used for the computational model, where effort comprises four motion factors: space, weight, time, and flow. The shape component involves three distinct qualities of change in the form of movement: Shape flow, directional movement, and shaping.

Brand and Hertzmann approach the problem of synthetic motion generation by learning motion patterns from a highly varied set of motion captured sequences [12]. The learning pro-
cess identifies common choreographic elements across these sequences, the different style in which each element is performed, and a small number of stylistic degrees of freedom which span the many variations in the dataset. The learned model can synthesize novel motion data in any interpolation or extrapolation of styles.

Burke et al. specify the basic competencies of a motor system [16] for their system as simple gestures without noticeable discontinuities, locomotion, and eye/head/body-orientation in order to represent attendance. These are necessary to produce the variety of movements needed for performing the actions, but it is not enough. When looking at such movements over a longer time period, the illusion of life will break down as the character fails to interact with the environment and eventually repeats the same animation over and over. In order to get expressive characters, their list is expanded by parametrized motor actions, the support of new motor actions, and the ability to create new animations on-the-fly. But they undeline, that this rather vague specification should be considered as wishful elements rather than implemented effects.
This chapter presents approaches for agent design. After a short introduction, the requirements for an agent design are considered. Thereafter, some remarks about resource issues are followed by various basic approaches for an agent architecture. The main components of an agent like the perception system, inference machine and the operators used in it, path planning and the motor system are investigated in more detail.

Many of the previously mentioned examples and systems are presented here in a more detailed fashion. We try to concentrate on implementation-specific topics and show the variety of different approaches.

8.1 Introduction

Designing an agent architecture is a non-trivial task. Considerations have to be taken with respect to a large number of requirements and constraints. According Wooldridge [126], Maes defines agent architectures as follows:

“[A] particular methodology for building [agents]. It specifies how (...) the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions (...) and the future internal state of the agent. An architecture encompasses techniques and algorithms that support this methodology.“

Kaebling follows:

“[A] specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the data and control flow among the modules. A more abstract view of an architecture is as a general methodology for designating particular modular decompositions for particular tasks.“

Shehory and Sturm summarize the characteristics of an agent-based system with the following six points [100]:

1. Autonomy: Agents are responsible for their own activities. The self-controllability should be supported by the system.
2. **Complexity.** As stated above, agent-based systems are basically a collection of more or less complex components. This requires strong expressive power and many layers of details. The model should therefore be modular, support complexity management and describe the complex nature of an agent.

3. **Adaptability.** In a dynamic environment, the agent should adapt its behavior accordingly.

4. **Concurrency.** As agents may have to perform several actions at the same time, the system should be designed as a parallel processing system. Therefore, parallelism and concurrency is required to be included in the model.

5. **Distribution.** Multi-agent systems can be distributed over several independent hosts and should therefore support networking aspects.

6. **Communication richness.** The communication between an agent and its environment (e.g. other agents or information sources) is characterized by its type, content, and architecture. This should be integrated into the agent model.

With these requirements kept in mind we continue to specify the requirements for a game agent architecture.

### 8.2 Requirements

Burke seeks for agents that are robust, reactive, adaptable, honest, expressive, sensible, and scalable [16]. In order to achieve these goals, the agent has to be able to decide on partial knowledge, to react within an justifiable time, to learn from its experiences, to show surprise when something unexpected happens, to have personality, to display some common sense, and to work in a group of several other agents.

Travis first lists four criteria for architecture-supporting human-like agents in games [114]:

1. Since many games emphasize similar aspects of human performance, human characteristics underlying performance should be represented in a highly reusable architecture.

2. The main consideration in building a computer player is enabling it to play the game effectively. The agent architecture should facilitate constructing capable agents by incorporating sophisticated AI mechanisms for selecting and controlling action.

3. The architecture should emphasize limitations and temporal characteristics of human performance that tend to be game-relevant.

4. Games will differ in which aspects of human performance are worth representing. When developing any particular game agent, it should be easy to “turn-off” or ignore human attributes not currently relevant.

He therefore divides his architecture into a human resource architecture, which provides human-like sensing methods and memory, and a agent architecture, which mainly consists of a reactive planner. We will find similar distinctive modules in many of the approaches presented later.

Lent [118] asks for reactive, context specific, flexible, realistic, and easy to develop agents in computer games. Context specific means that the agent’s selected actions should correspond to prior sensor input and actions. Flexibility requires more than one possibility on how to achieve a goal in means of different tactics and different low-level behaviors within these tactics. Realism means that the agent has the same strengths as well as the same weaknesses as humans have. Reusability of the knowledge is considered the main aspect for easy developing.
8.3 Resources

Computational resources in computer games are limited mainly because of the real-time constraint. Laird [60] discusses this point with respect to CPU and memory.

**CPU.** At one extreme, actions can be selected using a precompiled lookup-table, while at the other extreme, every action might require inference and planning which can last longer than one cycle of the rendering subsystem. The number of active agents also influences the amount of CPU time available for one agent. As the number of entities increases, the behavior is forced towards more and more simple behavior. With hierarchical decomposition of the entities into groups and subgroup (such as platoons and companies, or a level of detail planning module), this problem can be attenuated. Usually, between 5 and 10 percent of the overall CPU time is available for AI in a real-time game, in strategy games usually significantly more.

**Memory.** Laird distinguishes three types of memory costs: First, the runtime-architecture may need memory to hold the code which is executed by the AI architecture, which does not include knowledge and runtime state. Second, the long-term behavioral knowledge and program, which is the storage for tactics, actions, etc. Finally, the runtime state denotes the last memory cost. This is the amount of storage required to hold the internal, dynamic data structures used by the behavior knowledge. This includes sensory information and internal state such as map information.

We consider the memory constraints not as important as the CPU constraints because recent game consoles and PCs are well equipped with memory.

8.4 Basic Approach

The game Creatures [36] implements a very open approach which aims at genetic inheritance and is therefore not really suitable for further investigation with respect to a modularization of the agent. Nevertheless, we will have a look at it and focus on the goodies of the approach. The whole brain consists of several lobes1 which consist of neurons interconnected by dendrites. Two main parts can be distinguished: The attention lobe and the decision making part. The former is used to direct the creatures attention towards one object. This limits the creature to “verb object” actions as opposed to “subject verb object” actions and reduces sensory and neural processing to acceptable levels. The decision making part is divided into a perception lobe, a concept lobe, and a decision lobe.

Isla [50] refines a modular approach with biology in mind (Figure 8.1) which consists of multiple interacting modules. First sensory information is fetched by the sensory system, which passes only the relevant sensory information as percepts through the perception system into the working memory. Therein, percept memory objects are stored. The action system selects an action based on the actual percept memory objects from one of two groups: First, a startle list is scanned for a deterministic action selection and if no startle action is active, default actions are probabilistically selected. Using the internal blackboard mechanism, the selected action is possibly propagated to a navigation system and decomposed into motor tasks. These are then finally applied to the agent. Burke et al. extend this approach by a proprioceptive channel which connects the blackboard back to the sensory system to emulate self-perception [16].

Chen’s behavior specification language BSL [21] which is similar to Funge’s CML approach is also integrated into an agent architecture (Figure 8.2). This model is layered into three control modules. The motor layer provides appearance and motion, the planning layer implements the

1. A lobe is a part of the biological brain.
brain and is responsible for motor, perception, and low-level action control. Finally, a reactive layer responds to unexpected events. Using a BSL interpreter, the planning layer can reason about how to achieve the agent’s goals. The reactive layer prevents the planning layer from stopping when it has not found a goal within reasonable time.

These different approaches represent the commonly used ways in which research has tried to build artificial brains to control autonomous agents. The first is a modular neural network approach which is really open for adaption and learning but often lacks controllability. The second tries to split the system into multiple control units, each responsible for a certain task. Here, the controllability is guaranteed, but it’s very difficult to find an open representation to easily add new behavior. The last is the decomposition into only a few modules, each responsible for a very general task. This is a mix of both the other approaches and tries to find a solution in between control and openness.
8.5 Perception

The first step in almost every agent cycle is sensing and/or perception. Burke [16] clearly distinguishes between sensing and perception: Sensing is the part where the system has to decide if a input is relevant to the agent (an object outside the field of view is irrelevant) while the perception mechanism transforms the sensed data into the agents space\(^1\) and inserts the sensed information into the knowledge base.

Computer vision provides many algorithms for visual perception and interpretation of input images. In games or other simulations, this problem can be omitted because the world is accessible and we do not have to care about incoming fuzzy information. Nevertheless, especially the vision and audio sensory system has to be designed very carefully in order to obtain a nearly humanlike agent.

Processing sensory information can be done in various styles. Tu and Terzopoulos [115][110] use neural inspired architectures for their artificial fishes which have the task to map the sensory information onto input neurons in a proper way. Tu implements a vision and temperature sensor and uses a focuser subsystem which rejects unimportant sensory information. The vision sensor is only simulated while in [110] it is based on computer vision algorithms and some ideas of the primate vision apparatus.

Kuffner’s PhD thesis focuses on motion generation and presents a hardware accelerated synthetic vision sensor which renders an low-resolution image from the agents viewpoint in order to find all objects within the line of sight [58]. To do this, each object is rendered in a different color and after all objects have been rendered, the different colors in the image result in a list of the visible objects. Additionally, Kuffner presents an algorithm for sensing in dynamic environ-

![Figure 8.3: The sense-plan-control loop for dynamic environments by Kuffner [58]](image)

ments as presented in figure 8.3. Here, the problem of hidden or moving objects arises immediately. The trick is to re-run the vision sensor with the objects \( M \) the agent expects to see with the result to get a set \( V_\mu \) containing the objects from \( M \) which are visible. The difference between these two sets depicts the set of objects which have disappeared.

Isla’s work on challenges in character based AI for games discusses some relevant issues for perception and sensing [49]. An important part of this discussion treats sensory honesty. This is

---

1. For example, the position of an object is transformed from world coordinates into the relative coordinate system with the origin at the agent’s point of view.
considered to be a fundamental principle for believable agents. Sensory honesty deals with both *what* can be perceived and *how* it is perceived. Occluded objects should not be visible and the coordinates of an object should be given in relative coordinates rather than global world coordinates. This produces a basic separation between the state of the world and the *view* of this state by the agent. This forces the agent to make intelligent assumptions about the further evolving of the world which includes anticipation and imagination.

Further, situation assessment has been investigated by Zhang [131]. He views situation assessment as a classification task where the appropriate situation templates are tested for a match to the current situation. First, they construct object patterns from the sensed entities. A first approach tries this with a hierarchical nearest-neighbor search which fails. The second approach uses situation templates. It first selects the possibly relevant templates, then builds a set of patterns, one for each template, and finally chooses the one with the highest relevance function value. This approach uses both a priori knowledge as guidance and the observed objects as variables. The overall process is shown in figure 8.4.

![Figure 8.4: Situation recognition by Zhang [131]](image)

### 8.6 Inference

The task of inference is very complex and manifold. Its goal is to predict future behavior which meets the requirements of the environment and the agent. Logical reasoning for deduction, planning, and learning builds the base of every inference machine. There are many different approaches to reasoning from first order logic or predicate calculus over situation calculus and spatial reasoning to reasoning under uncertainty.

A game agent should at least infer in situation calculus in order to set up a correct plan to achieve a certain goal. In order to do so, it faces two main problems: First, game environments are highly dynamic, complex, and therefore hard to predict because of the user input. Second, the amount of processing time for inference is strongly limited to about 10% of CPU time as mentioned before. In a dynamic world, a plan generated with situation calculus can fail during its execution due to external changes which were unpredictable during generation. Therefore, a replanning mechanism has to be provided which has to be started, when the agent realizes itself in an unexpected situation or it recognizes a change in any of its assumptions.

In real-time environments the inference mechanism cannot simply search the whole tree of possible states for the goal state and therefore suspend other important processes such as the rendering or simulation. The goal should be an inference mechanism which can provide at any time
an answer to the relevant question of what should be done. These algorithms are known as *anytime algorithms* [132]. An anytime algorithm is an algorithm that can be interrupted at any time, and the quality of the result that is returned after an interruption increases with processing time. The primary advantage of an anytime planner is that the agent using it is afforded complete control over the planning process. Recently, Hawes [43][44] has presented anytime planning for goal-oriented agent behavior. He points out that such algorithms hardly ever find an optimal or complete solution which he deems as an advantage rather than a disadvantage because humans seldomly behave with complete rationality: “For instance, interrupting a forward chaining total order planner will result in a plan fragment that would take the agent from its start state to some other point in the environment/search space. There may not be any guarantee that this point is any closer to the desired goal state than the start point was. If a total order regression planner was interrupted, the resulting plan would be a plan from an arbitrary point in the search space, to the goal state, and there would be no guarantee that the agent could get to that point. The outcome would be different again for a partial order planner. Because these results would be almost useless when generating behavior for an agent, an HTN planner has been used as the basis for an anytime planner.” Horsch et al. [46] presents an anytime algorithm for reasoning under uncertainty. He presents the algorithm with a maze-solving agent with either perfect or noisy sensors and actuators.

### 8.6.1 Operator Selection

Closely related to the inference mechanism is the task of operator selection. Burke et al. [16] specifies three essential issues to be addressed when designing an action selection mechanism: First, the fundamental representation of the actions used by the system. Second, how does the system choose the action(s) to perform, and third, how can the choice be modified to allow to learn from experience. The first issue is discussed in the next section. We will have a deeper look into the second one here.

Sims et al. with their fishes and creatures [103][115] do not use an abstraction of actions. Their system relies on a neural network which connects input sensors over internal functional neurons with the effectors directly on the artificial muscles. Depending on the fitness function the creature evolves either a swimming, walking, jumping, or following behavior. In contrast, Hodgins uses a state-machine for her animated human athletics [45]. The state-machine determines the sequence of constraints and motions for a running athlete. Velasquez’ emotional agents use a network of self-interested behaviors which are selected according to a value computed independently for each behavior [120]. The proposed model is able to select multiple behaviors at the same time and perform them simultaneously. Tomlinson’s expressive autonomous cinematography system uses a similar approach as Velasquez [113]. Because of the mutual exclusion of behaviors, the system calculates the value of each behavior depending on the currently active behavior in order to obviate an oscillation between two behaviors with closely matched values. They call this effect *behavioral aliasing*.

Burke [16] further states that the agent should be able to perform more than one action at a time. But not all combinations of actions are possible or permitted. The proposed selection mechanism uses ActionTuples (see next section) which can calculate their relevance themselves which depicts the importance of this action. The relevance depends on the state of the action. If the action is inactive, the TriggerContext is considered, if active, the DoUntilContext.

---

1. A HTN planner is a hierarchical task-network planner which takes a partially ordered set of tasks and actions, and uses schemata to reduce tasks by substituting them with networks of tasks and actions until only actions remain. When tasks and actions interfere with each other, critic functions perform arbitrary plan-transformation repairs.
ActionTuples are grouped into ActionGroups. When actions are permitted to run simultaneously, the group allows every action in it to run which has a relevance larger than zero. The groups also arbitrate among competing ActionTuples. ActionGroups are further divided into a startle list and a default list. The action returning the highest relevance wins and is executed. If no action in the startle list returns a positive relevance the actions in the default list are taken into account which compete probabilistically on the basis of a specified value.

Aylett [5] presents a motivation based planning algorithm as shown in figure 8.5.

Motivations are long-term aims or objectives, or as drives or emotional states, depending on the domain. [...] Motivations have an associated weight which often changes over time and provides a driving force directing the generation of goals to satisfy the motivation. [...] Motivations also allow an agent to evaluate the plans it generates to achieve its goals and to choose between alternative plans.” [5]

Motivations are modeled by a set of tuples \{name, weight\}. A hashtable links the name of each possible action and its parameters to the motivations that it supports or undermines. The way in which an action contributes to a motivation is represented by two sets: the pros and cons. Each set is made up of tuples \{name, strength\}, giving the name of a motivation and the degree to which the action supports or undermines it. At each step, either a goal or an action is selected to be executed. Goals or actions are selected according to their importance, effort, and deadline. With that, in every cycle, the agent has to decide whether to invest in planning or executing.

Finally, Stone presents an architecture for action selection in robotic soccer [109] which also relies on competing actions which evaluate a score and the action with the highest score is selected and executed. The problem here is, that the outcome of the action itself is not foreseeable and therefore, a set of heuristics are presented to allow the robot passing, shooting, dribbling, or moving in a tactical position without the ball. This won’t necessarily lead to a very sophisticated game, because high-level tactics or team-playing are not considered during the evaluation.
8.6.2 Operator Modeling

We consider operators of any hierarchical order as the major part of the knowledge base beside the knowledge about facts on which the operators work. This includes tactics, plans, complex actions, and atomic actions. We will examine the modeling of operators and actions in order to obtain a flair of the requirements for such entities.

Spector’s technical report on reactive planning within a multi-level architecture [108] defines a generic operator which consist of the components shown in figure 8.6. The concept of monitors is remarkable and can be used to trigger any effect such as success, failure, suspension or resetting. The steps denote the necessary sub-operators for this operator and the step-sequence indicates ordering constraints within these steps. The expect list indicates the necessary forms to successfully complete the operator. These expectations can depend on sub-operators.

Introduced in the last section, Burke’s actions [16] are similar but do contain only four parts, which mirror the four questions “when to do it?”, “what and how to do it?”, “what to do it to?”, and “for how long?”. Figure 8.7 shows an example. The trigger context corresponds to the

![Figure 8.6: The format of an operator in [108]](image)

![Figure 8.7: An example of an ActionTuple the fundamental action representation in [16]](image)

expect list in the above example, the action to the steps, the context to the consumers. The novel thing is the DoUntilContext, which determines the action. This can be the continuation of the action for some time period or until a state is achieved.

As we have seen in chapter 8, Badler’s parametrized action representation (PAR) [6] is more precise and mighty and is focused on human language as the instruction medium. As displayed in picture 8.8, almost every required component for an action is represented and can be specified.
8.7 Path Planning: A*

An important issue for game agents is path planning in the virtual world. Here, the task is to find a sequence of basic motor instructions such as forward, turn right, or turn left for the purpose of finding a path from the actual position to a goal position.

For a single agent, this task is usually solved using an A* path planning algorithm. While there are many different pathing algorithms, A* finds the shortest, if one exists, and is relatively fast. [19] provides a short explanation and introduction to A*. [18] also describes speed and aesthetic issues for A*. Davis discusses path planning for the game *Star Trek®: Armada* [28]. He also rates A* as the most promising algorithm but annexes that the choice of the decomposition of the environment is very important. His approach uses a quadtree to spatially decompose the playing field which greatly reduces the cells A* needs to search.

Kuffner presents in [57] a real-time path planning algorithm for goal-directed navigation. He first renders an off-line 2D top projection of the scene with the obstacles in the height range of the character and uses a circular disk in 2D representing the character to find a collision-free trajectory. This task should be computable within very few time, since moving obstacles can necessitate a replanning on-line. A path following module then generates the motion along this path.

---

**Figure 8.8:** Syntactic Representation of a PAR by [6]
References


