Intelligent Agents for Computer Games

Alexander Nareyek

GMD FIRST, Kekuléstr. 7, D - 12489 Berlin, Germany
alex@ai-center.com
http://www.ai-center.com/home/alex/

Abstract. In modern computer games — like action, adventure, role-playing, strategy, simulation and sports games — artificial intelligence (AI) techniques play an important role. However, the requirements of such games are very different from those of the games normally studied in AI.
This article discusses which approaches and fields of research are relevant to achieve a sophisticated goal-directed behavior for modern computer games' non-player characters. It also presents a classification of approaches for autonomous agents and gives an overview of a solution developed in the EXCALIBUR project.

Keywords: commercial games, action planning, real time, dynamics, incomplete knowledge, resources.

1 Modern Computer Games

The use of game applications has a long tradition in artificial intelligence. Games provide high variability and scalability for problem definitions, are processed in a restricted domain and the results are generally easy to evaluate. But there is also a great deal of interest on the commercial side, the "AI inside" feature being a high-priority task [6, 7, 26] in the fast-growing, multi-billion-dollar electronic gaming market (the revenue from PC games software alone is already as big as that of box office movies [15]).

Many "traditional" games, such as card/board/puzzle games like Go-Moku [1] and the Nine Men's Morris [12], have recently been solved by AI techniques. Deep Blue's victory over Kasparov was another milestone event here. However, it is highly questionable whether and to what extend the techniques used in this field of research can be applied to today's "modern" computer games. Of the techniques used in this field, A* [14] (an improved version of Dijkstra's shortest-path algorithm [8]) and its variants/extensions are practically the only ones employed in modern computer games (see also [28]).

Such games pose problems for AI that are infinitely more complex than those of traditional games. Modern computer games feature:

- Real Time: There is only very limited time for reasoning.
- Dynamics: Computer games provide a highly dynamic environment.
- Incomplete Knowledge: A game character has only incomplete knowledge of the world.
- Resources: The game character's/world's resources may be restricted.

Techniques from the AI fields of autonomous agents, planning, scheduling, robotics and learning would appear to be much more important than those from traditional games.

AI techniques can be applied to a variety of tasks in modern computer games. A game that uses probabilistic networks to predict the player's next move in order to speed up graphics may be on a high AI level. But although AI must not always be personified, the notion of artificial intelligence in computer games is primarily related to characters. These characters can be seen as *agents*, their properties perfectly fitting the AI agent concept.

But how does the player of a computer game perceive the intelligence of a game agent/character? This question is dealt with neatly in [16]. Important dimensions include physical characteristics, language cues, behaviors and social skills. Physical characteristics like attractiveness are more a matter for psychologists and visual artists (e.g., see [11]). Language skills are not normally needed by game agents and are ignored here too.

The most important question when judging an agent's intelligence is the goal-directed component, which we look at in the rest of this paper. The standard procedure followed in modern computer games to implement a goal-directed behavior is to use predetermined behavior patterns. This is normally done using simple if-then rules. In more sophisticated approaches using neural networks, behavior becomes adaptive, but the purely reactive property has still not been overcome.

Many computer games circumvent the problem of applying sophisticated AI techniques by allowing computer-guided agents to cheat. But the credibility of an environment featuring cheating agents is very hard to ensure, given the constant growth of the complexity and variability in computer-game environments. Consider a situation in which a player destroys a communication vehicle in an enemy convoy in order to stop the enemy communicating with its headquarters. If the game cheats in order to avoid a realistic simulation of the characters' behavior, directly accessing the game's internal map information, the enemy's headquarters may nonetheless be aware of the player's subsequent attack on the convoy.

2 A Classification of Autonomous Agents

Wooldridge and Jennings [29] provide a useful starting point by defining autonomy, social ability, re-activity and proactiveness as essential properties of an agent. Agent research is a wide area covering a variety of topics. These include:

- *Distributed Problem Solving (DPS)*
  The agent concept can be used to simplify the solution of large problems by distributing them to a number of collaborating problem-solving units. DPS is not considered here because computer games' agents should normally act fully autonomously: Each agent has individual goals.
- *Multi-Agent System (MAS)*
  MAS research deals with appropriate ways of organizing agents. These include general organizational concepts, the distribution of management tasks, dynamic organizational changes like team formation and underlying communication mechanisms.

- *Autonomous Agents*
  Research on autonomous agents is primarily concerned with the realization of a single agent. This includes topics like sensing, models of emotion, motivation, personality, and action selection and planning. This field is our main focus here.

An agent has goals (stay alive, catch player's avatar, ...), can sense certain properties of its environment (see objects, hear noises, ...), and can execute specific actions (walk northward, eat apple, ...). There are some special senses and actions dedicated to communicating with other agents.

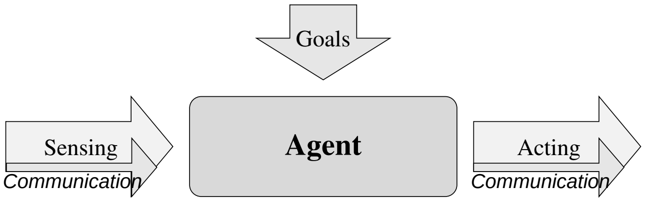

Fig. 1. Interaction with the Environment

The following sections classify different agent architectures according to their trade-off between computation time and the realization of sophisticated goal-directed behavior.

2.1 Reactive Agents

Reactive agents work in a hard-wired stimulus-response manner. Systems like Joseph Weizenbaum's Eliza [27] and Agre and Chapman's Pengi [2] are examples of this kind of approach. For certain sensor information, a specific action is executed. This can be implemented by simple if-then rules.

The agent's goals are only implicitly represented by the rules, and it is hard to ensure the desired behavior. Each and every situation must be considered in advance. For example, a situation in which a helicopter is to follow another helicopter can be realized by corresponding rules. One of the rules might look like this:

```
IF (leading_helicopter == left) THEN
  turn_left
ENDIF
```

But if the programmer fails to foresee all possible events, he may forget an additional rule designed to stop the pursuit if the leading helicopter crashes. Reactive systems in more complex environments often contain hundreds of rules, which makes it very costly to encode these systems and keep track of their behavior.

The nice thing about reactive agents is their ability to react very fast. But their reactive nature deprives them of the possibility of longer-term reasoning. The agent is doomed if a mere sequence of actions can cause a desired effect and one of the actions is different from what would normally be executed in the corresponding situation.

2.2 Triggering Agents

Triggering agents introduce internal states. Past information can thus be utilized by the rules, and sequences of actions can be executed to attain longer-term goals. A possible rule might look like this:

```
IF (distribution_mode) AND (leading_helicopter == left)
THEN
  turn_right
  trigger_acceleration_mode
ENDIF
```

Popular Alife agent systems like CyberLife's Creatures [13], P.F. Magic's Virtual Petz [25] and Brooks' subsumption architecture [5] are examples of this category. Indeed, nearly all of today's computer games apply this approach, using finite state machines to implement it.

These agents can react as fast as reactive agents and also have the ability to attain longer-term goals. But they are still based on hard-wired rules and cannot react appropriately to situations that were not foreseen by the programmers or have not been previously learned by the agents (e.g., by neural networks).

2.3 Deliberative Agents

Deliberative agents constitute a fundamentally different approach. 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. Planning systems are often identified with the agents themselves. A great deal of research has been done on planning, and a wide range of planning systems have been developed, e.g., STRIPS [9], UCPOP [24], Graphplan [3] and SATPLAN [17, 18].

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[1].

Deliberative agents have no problem attaining longer-term goals. Also, the encoding of all the special rules can be dispensed with because the planning system can establish goal-directed action plans on its own. When an agent is called to execute its next action, it applies an internal planning system:

```
IF (current_plan_is_not_applicable_anymore) THEN
  recompute_plan
```

[1] Note that the use of the term *planning* in AI is different from that expected by people in the operations research (OR) community (e.g., *scheduling*).

```
ENDIF
execute_plan's_next_action
```

Even unforeseen situations can be handled in an appropriate manner, general reasoning methods being applied. The problem with deliberative agents is their lack of speed. Every time the situation is different from that 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.

2.4 Hybrid Agents

Hybrid agents such as the 3T robot architecture [4], the New Millennium Remote Agent [23] or the characters by Funge et al. [10] apply a traditional off-line deliberative planner for higher-level planning and leave decisions about minor refinement alternatives of single plan steps to a reactive component.

```
IF (current_plan-step_refinement_is_not_applicable_anymore)
THEN
   WHILE (no_plan-step_refinement_is_possible) DO
     recompute_high-level_plan
   ENDWHILE
   use_hard-wired_rules_for_plan-step_refinement
ENDIF
execute_plan-step_refinement's_next_action
```

There is a clear boundary between higher-level planning and hard-wired reaction, the latter being fast while the former is still computed off-line. For complex and fast-changing environments like computer games, this approach is not appropriate because the off-line planning is still too slow and would — given enough computation time — come up with plans for situations that have already changed.

2.5 Anytime Agents

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:

```
WHILE (computation_time_available) DO
  improve_current_plan
ENDWHILE
execute_plan's_next_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. An architecture for anytime agents was realized within the EXCALIBUR project, which is described in Section 4.

3 Other Requirements

The previous sections pointed to the need for *anytime planning* capable of *easily adapting to a changing environment*. However, these are only two aspects relating to the planning of an agent's behavior. The expressiveness of planning is also very important:

- **Temporal planning:** The planning mechanisms must take a rich temporal annotation into account. Actions have a duration, the game world's agents can act simultaneously, and actions may have synergetic effects.
- **Resources:** Resources such as food, hit points or magical power are standard features of today's computer games. Computer-guided characters often have goals that are related to these resources and must take this into account in planning their behavior.
- **Incomplete knowledge:** An agent can only know about the things that it has sensed via its sensors. The type and number of the world's objects is not known in advance and can change without any action on the part of the agent.
- **Domain-specific guidance:** The ability to make use of domain-specific knowledge often makes the difference between a planning's quick success and miserable failure. It is critically important that this knowledge can be incorporated in a simple and appropriate manner.

There are many planning approaches that address one of these problems, but there are virtually none (except the EXCALIBUR agent's planning system) that try to address all of them.

4 The EXCALIBUR Project

To illustrate how computer games can implement anytime agents that also meet the requirements formulated in the previous section, we give a brief overview of the approaches applied in the EXCALIBUR project. In this project, a generic architecture was developed for autonomously operating agents, like computer-guided characters/mobiles/items, that can act in a complex computer-game environment.

The planning of an agent's behavior is realized in a constraint programming (CP) framework, which makes the approach highly declarative. Furthermore, CP methods have proved very successful in handling problems that involve resources and scheduling and are thus highly suitable as an underlying paradigm for the agents. Problems are formulated in a framework of variables, domains and constraints. The constraint satisfaction problem (CSP) consists of

- a set of variables $x = \{x_1, \ldots, x_n\}$

– where each variable is associated with a domain $d_1, ..., d_n$
– and a set of constraints $c = \{c_1, ..., c_m\}$ over these variables.

The domains can be symbols as well as numbers, continuous or discrete (e.g., “door”, “13”, “6.5”). Constraints are relations between variables (e.g., “x_a is a friend of x_b”, “$x_a < x_b \times x_c$”) that restrict the possible value assignments. Constraint satisfaction is the search for a variable assignment that satisfies the given constraints. Constraint optimization requires an additional function that assigns a quality value to a solution and tries to find a solution that maximizes this value.

However, if the CP framework is used to compute an agent’s behavior, several extensions are needed to achieve the required efficiency and expressiveness. These include structural constraint satisfaction [19, 20], which enhances conventional constraint satisfaction by allowing problems to be formulated in which the constraints and variables must not be given in advance. The search for the correct structure of the CSP is part of the satisfaction process here. This is necessary for the task of generating an agent’s behavior plan because multiple CSP structures are possible, the number/type of actions for the plan being unknown a priori.

Another important extension is the combination of constraint programming and local search [22]. This combination is a key feature, providing the agent with efficient anytime reasoning and allowing the uncomplicated handling of the environment’s dynamics. Local-search approaches perform a search by iteratively improving an initial state/plan. In the applied approach, domain-dependent knowledge can be encapsulated in the constraints, which choose advantageous changes to the current plan by themselves. Figure 2 shows how sensing, planning and execution are interleaved.

The resource focus is reflected in the architecture’s planning model [21] and allows us to use and optimize temporal, spatial and all other kinds of resources. Furthermore, the model is expressive enough to handle incomplete knowledge and information gathering. The system was tested on several sample problems and yielded good results for specialized domain-dependent solutions as well as for domain-independent tasks. Detailed information on the EXCALIBUR project is available at:
`http://www.ai-center.com/projects/excalibur/`

5 Conclusion

Given the fast-growing complexity of modern computer games, approaches used for traditional games can no longer be directly applied to the modern games. Techniques from fields like autonomous agents and planning would appear to be more appropriate. However, even these fields have to be extended by mechanisms designed to satisfy the sophisticated requirements of modern computer games — such as real time, dynamics, resources and incomplete knowledge. The EXCALIBUR agents illustrate how these requirements can be tackled.

To conclude, I wish to draw attention to the setting up of an Artificial Intelligence Special Interest Group within the International Game Developers Association. Technology transfer from academia is one of the main issues, and anyone interested is invited to join the SIG and present/discuss their work. More information is available at:
`http://www.igda.org/SIGs/game_ai.htm`

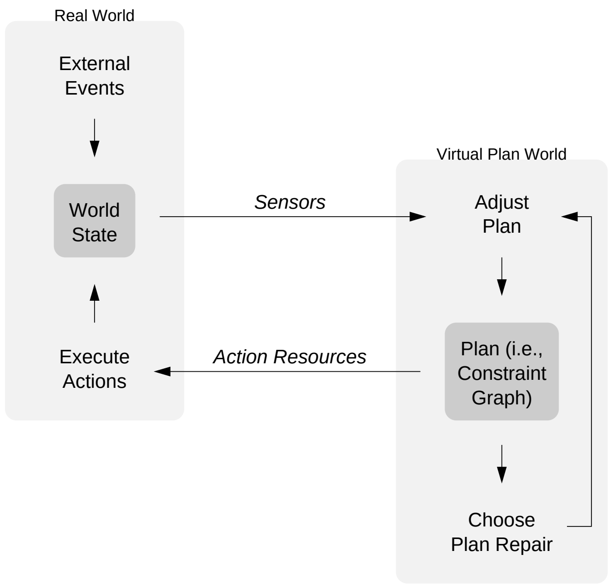

Fig. 2. Interleaving Sensing, Planning and Execution

6 Acknowledgments

The EXCALIBUR project is supported by the German Research Foundation (DFG), Conitec Datensysteme GmbH, Cross Platform Research Germany (CPR) and NICOSIO. Thanks to Reijer Grimbergen for his suggestions on how to improve this paper.

References

1. L. V. Allis, H. J. van den Herik, and M. P. H. Huntjens. Go-Moku Solved by New Search Techniques. In *Proceedings of the 1993 AAAI Fall Symposium on Games: Planning and Learning*, 1993.
2. P. Agre and D. Chapman. PENGI: An Implementation of a Theory of Activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (AAAI-87), 268–272, 1987.
3. A. L. Blum and M. L. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* 90: 281–300, 1997.
4. R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, M. G. Slack. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence* 9(1), 1997.
5. R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation RA-2* (1): 14–23, 1986.
6. C. Charla. Mind Games: the Rise and Rise of Artificial Intelligence. *Next Generation* November, 1995.
7. D. Coco. Creating Intelligent Creatures. *Computer Graphics World*, July, 1997.

8. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1: 269–271, 1959.
9. R. E. Fikes and N. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 5(2): 189–208, 1971.
10. J. Funge, X. Tu, and D. Terzopoulos. Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques* (SIGGRAPH'99), 29–38, 1999.
11. T. Gard. Building Character. *Gamasutra*, `http://www.gamasutra.com/features/20000720/gard_01.htm`, June, 2000.
12. R. Gasser. Solving Nine Men's Morris. *Computational Intelligence* 12(1): 24–41, 1996.
13. S. Grand, D. Cliff, and A. Malhotra. Creatures: Artificial Life Autonomous Software Agents for Home Entertainment. In Proceedings of the First International Conference on Autonomous Agents (Agents'97), 22–29, 1997.
14. P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2): 100–107, 1968.
15. K. Hause. What to Play Next: Gaming Forecast, 1999-2003. Report #W21056, International Data Corporation, Framingham, Massachusetts, 1999.
16. K. Isbister. Perceived Intelligence and the Design of Computer Characters. *Lifelike Computer Characters* (LCC'95), Snowbird, Utah, 1995.
17. H. Kautz and B. Selman. Planning as Satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence* (ECAI-92), 359–363, 1992.
18. H. Kautz and B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (AAAI-96), 1194–1201, 1996.
19. A. Nareyek. Structural Constraint Satisfaction. In *Papers from the 1999 AAAI Workshop on Configuration*, Technical Report, WS-99-05, 76–82. AAAI Press, Menlo Park, California, 1999.
20. A. Nareyek. Applying Local Search to Structural Constraint Satisfaction. In *Proceedings of the IJCAI-99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, 1999.
21. A. Nareyek. Open World Planning as SCSP. In *Papers from the AAAI-2000 Workshop on Constraints and AI Planning*, Technical Report, WS-00-02, 35–46. AAAI Press, Menlo Park, California, 2000.
22. A. Nareyek. Using Global Constraints for Local Search. In E. C. Freuder and R. J. Wallace (eds.), *Constraint Programming and Large Scale Discrete Optimization*, American Mathematical Society Publications, DIMACS Volume 57, 9–28, 2001
23. B. Pell, D. E. Bernard, S. A. Chien, E. Gat, N. Muscettola, P. P. Nayak, M. D. Wagner, and B. C. Williams. A Remote Agent Prototype for Spacecraft Autonomy. In *Proceedings of the SPIE Conference on Optical Science, Engineering, and Instrumentation*, 1996.
24. J. S. Penberthy and D. S. Weld. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (KR'92), 102–114, 1992.
25. A. Stern, A. Frank, and B. Resner. Virtual Petz: A Hybrid Approach to Creating Autonomous Lifelike Dogz and Catz. In *Proceedings of the Second International Conference on Autonomous Agents* (AGENTS98), 334–335, 1998.
26. A. Stern. AI Beyond Computer Games. AAAI Symposium on Computer Games and Artificial Intelligence, 1999.
27. J. Weizenbaum. ELIZA — A Computer Program for the Study of Natural Language Communication between Man and Machine. *Communications of the ACM* 9(1): 36–45, 1966.

28. S. Woodcock. Game AI: The State of the Industry. *Game Developer*, August, 2000.
29. M. Wooldridge, and N. R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review* 10(2): 115–152, 1995.