Controlling Unreal Tournament 2004 Bots with the Logic-based Action Language GOLOG

Stefan Jacobs and **Alexander Ferrein** and **Gerhard Lakemeyer**

Knowledge-based System Groups
Computer Science Department
RWTH Aachen
52056 Aachen, Germany
{sjacobs,ferrein,gerhard}@cs.rwth-aachen.de

Abstract

Computer games and the accompanying entertainment industry branch has become a major market factor. AI techniques are successfully applied to tasks like path planning or intelligent swarm behavior. On the decision-making level the state of the art are state machines with a fixed set of behaviors. The perception of the computer player is perfect. They exactly know where the other players are located, even if they cannot "see" them. This approach seems to be limited for intelligent decision making. Therefore, we propose another approach. We use a variant of the logic-based action language GOLOG for implementing so-called game bots in the UNREAL TOURNAMENT 2004 environment. First results show that we can compete with the omniscience game bots in the Unreal domain.

Introduction

Computer games and the accompanying entertainment industry branch has become a major market factor. With increasing processor speed and powerful graphic engines computer games are becoming more and more realistic.

An important aspect for game-play is that the computer opponents act in an "intelligent" and realistic way. AI techniques are applied to so-called "low-level" tasks like path planning or intelligent swarm behavior. On this level computer animated creatures seem to act human-like.

A very successful branch of computer games are the so-called ego-shooter games, where the player is controlling one character perceiving the environment from the character's point of view. In those games, the characters act mainly in an adversarial environment where the goal is to eliminate opponent players. Examples for such games are Quake (Id Software Inc. 2002) or Unreal Tournament (Epic Games Inc. 2004b). Computer controlled characters are called game bots or bots, for short.

While ego-shooter games may not be everybody's cup of tea, they are nevertheless very challenging from the AI perspective. The environment is highly-dynamic, one has to deal with uncertainty, and decisions have to be made in real-time. Programming those bots in a way that the human player regards them as "intelligent" and intriguing opponents is a challenging task for game AI.

The state of the art techniques for implementing game bots are mainly state automata with fixed transitions. To let them appear more realistic, the programmer equips them with a nearly perfect world model. The human player can see through this omniscience very easily. To let the behavior of a game bot appear more realistic, it should be equipped with a human-like world model. Then, the decision making algorithms become more complex, and, for AI research, more interesting.

Therefore, our approach to decision making of a game bot is based on the logic-programming language GOLOG which combines explicit agent programming with the possibility to reason about actions and change. In particular, we use our variant READYLOG which, moreover, integrates decision-theoretic planning techniques.

Our application domain is the game "Unreal Tournament 2004" which will be outlined in the next section. Afterwards, we give an overview of the language features of READYLOG which we use to model Unreal game bots, before we describe the contents of the demonstration.

Unreal Tournament 2004

UNREAL TOURNAMENT 2004 (Epic Games Inc. 2004b) was developed by EPIC GAMES (Epic Games Inc. 2004a). It is a multi-player game where human players compete in performing different tasks. The opponents are either human players or computer-controlled game bots. There exist different game modes like *death match* or *capture the flag*. The goal of the former is to disable as many opponents as possible, while in the latter two teams of players have to defend their own home base. Here, strategic planning and coordination is needed.

While the game engine of UNREAL TOURNAMENT 2004 is not publicly available, the decision making routines of the game bots are open source. The bots are implemented in the object-oriented language USCRIPT, a script language which comes with the game. The algorithms for path planning or collision detection are part of the game engine and therefore cannot be modified.

The bot framework in USCRIPT offers high-level actions like *move to*, *attack*, or *retreat*. The original bots were implemented using a state machine consisting of nine different states. The state classification is controlled by the game engine.

As the decision making code of the bots is available, it is possible to change it in such a way that agent technologies become applicable. We therefore implemented a new interface to the game engine which allows to transmit world model information like the topology of a game level, visual information of opponents, team-mates, or items like weapons or health packs. This means that these items are only transmitted if the game bot can perceive them. Using this information we build a world model which is used for specifying the bot's behavior in the READYLOG framework. In contrast to omniscient agent programs, the information provided by the world model is incomplete in general.

Previously, (Kaminka *et al.* 2002) used the the Unreal framework as research subject. They proposed an interface to connect to the Unreal engine which is different from ours. Mainly their work focused on the "low-level" tasks like path planning.

READYLOG

For specifying our high-level control we use a variant of the logic-based high-level agent programming language GOLOG (Levesque *et al.* 1997). GOLOG is a language based on the situation calculus (Reiter 2001). Over the recent years many extensions like dealing with concurrency, exogenous and sensing action, a continuous changing world and probabilistic projections (simulation) (Giacomo, Lesperance, & Levesque 2000; Grosskreutz & Lakemeyer 2000; 2003) made GOLOG an expressive robot programming language. We further integrated a planning module into GOLOG which chooses the best action to perform by solving a Markov Decision Process (MDP) (we refer to (Puterman 1994) for reading on MDP and to (Boutilier *et al.* 2000) on integrating MDPs into GOLOG).

In the following we give an overview of some of the features of READYLOG, not going into details. We successfully apply READYLOG in the robotic soccer domain and refer to (Ferrein, Fritz, & Lakemeyer 2005; Dylla, Ferrein, & Lakemeyer 2003) for details.

- Sequence: a_1, a_2
- Nondeterministic Choice: $a_1; a_2$
- Solve an MDP: $solve(p, h)$, where p is a GOLOG program and h is the horizon up to which the MDP is solved
- Test: $?(c)$
- Event-Interrupt: $waitFor(c)$
- If-then-else: $if(c, a_1, a_2)$
- While-loops: $while(c, a_1)$
- Condition-bounded execution: $withCtrl(c, a_1)$
- Concurrent actions: $pconc(a_1, a_2)$
- Probabilistic actions: $prob(val_{prob}, a_1, a_2)$
- Probabilistic (offline) projection: $pproj(c, a_1)$
- Procedures: $proc(name(parameters), body)$

These are the ingredients for implementing an agent in the READYLOG framework. For space reasons we cannot give an example of a game bot program.

Demonstration

In the demonstration we show the current state of the game bot implementation. For the game variant *death match* we can show that the READYLOG-bot performs at an 80 % level compared to the original Unreal bots. We find this remarkable because the world model of the Unreal bot is nearly perfect compared to our world model. For the variants *capture the flag* more strategic planning is needed. We are currently working on that matter. Especially in this variant of the game we expect to perform much better in the future.

We want to present our framework as an agent programming framework for highly-dynamic and adversarial real-time domains. Other decision making algorithms could be easily integrated as our framework provides a well structured world model.

Several recorded demonstration sessions can be found at http://robocup.rwth-aachen.de/readybot/.

References

Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In *Workshop on Decision-Theoretic Planning, Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000).*

Dylla, F.; Ferrein, A.; and Lakemeyer, G. 2003. Specifying multirobot coordination in ICPGolog – from simulation towards real robots. In *Proc. of the Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating (IJCAI 03).*

Epic Games Inc. 2004a. http://www.epicgames.com/.

Epic Games Inc. 2004b. Unreal tournament 2004. http://www.unrealtournament.com/.

Ferrein, A.; Fritz, C.; and Lakemeyer, G. 2005. Using golog for deliberation and team coordination in robotic soccer. *KI* (1).

Giacomo, G. D.; Lesperance, Y.; and Levesque, H. J. 2000. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1-2):109–169.

Grosskreutz, H., and Lakemeyer, G. 2000. cc-Golog: Towards More Realistic Logic-Based Robot Controllers. In *AAAI-00.*

Grosskreutz, H., and Lakemeyer, G. 2003. Probabilistic complex actions in golog. *Fundamenta Informaticae* 57(1):167–192.

Id Software Inc. 2002. Quake 3 arena. http://www.idsoftware.com/games/quake/quake3-arena/.

Kaminka, G. A.; Veloso, M. M.; Schaffer, S.; Sollitto, C.; Adobbati, R.; Marshall, A. N.; Scholder, A.; and Tejada, S. 2002. Game Bots: A Flexible Test Bed for Multiagent Research. *Communications of the ACM* 45(2):43–45.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming* 31:59–84.

Puterman, M. 1994. *Markov Decision Processes: Discrete Dynamic Programming*. New York: Wiley.

Reiter, R. 2001. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press.