A Goal Processing Architecture for Game Agents

Elizabeth Gordon[*]
School of Computer Science and IT
University of Nottingham
Nottingham NG8 1BB, UK
esg@cs.nott.ac.uk

Brian Logan
School of Computer Science and IT
University of Nottingham
Nottingham NG8 1BB, UK
bsl@cs.nott.ac.uk

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Design

Keywords

Agent architectures, computer game agents, teleo-reactive programs

1. INTRODUCTION

The domain of computer games is becoming increasingly popular as a research platform for artificial intelligence (e.g., [6, 3, 4]). Games may be simplified compared to the real world, but they provide complex, dynamic environments which even human players find challenging. Current game AIs often exhibit inflexible, predictable behaviour. In this poster, we present an architecture for game agents capable of dynamically creating and adjusting their own goals.

2. GRUE: A NEW ARCHITECTURE

We use teleo-reactive programs (TRPs) as described in [2], which consist of a series of rules, each of which contains some number of conditions and actions. A TRP is run by evaluating all the rules and executing the actions of the first rule whose conditions evaluate to true when matched against a world model stored in the agent's memory. The actions can be durative, in which case the action continues as long as its condition is true. Our agents use TRPs as prewritten plans for achieving goals. In contrast to [2], which executes multiple programs in pseudo-parallel, we allow multiple actions to be executed during each cycle.

The teleo-reactive architecture described in [2] and [1] is not completely autonomous. Goals are proposed by a human user who is also responsible for determining the reward for achieving a goal. This is reasonable for robots, but not for game agents. Running programs in pseudo-parallel requires knowing when it is safe to switch programs. This is difficult or impossible in fast-paced games or when programs contain disjunct conditions. Game agents also encounter situations where several items might be adequate for the same task, or where objects come in quantities (money, ammunition).

[*]Elizabeth Gordon is a PhD student supported by Sony Computer Entertainment Europe.

We have developed a new teleo-reactive architecture, GRUE (Goal and Resource Using architecturE), designed specifically for situations encountered in games. GRUE is built around the key concept of resources, which overcomes some of the limitations of [2]. GRUE includes the following features:

- it allows an agent to generate new top-level goals in response to the current game situation and assign priorities to goals based on the current situation;
- it includes an arbitration algorithm designed to handle situations which are common in computer games;
- it enables multiple tasks to run actions in parallel during each cycle when possible

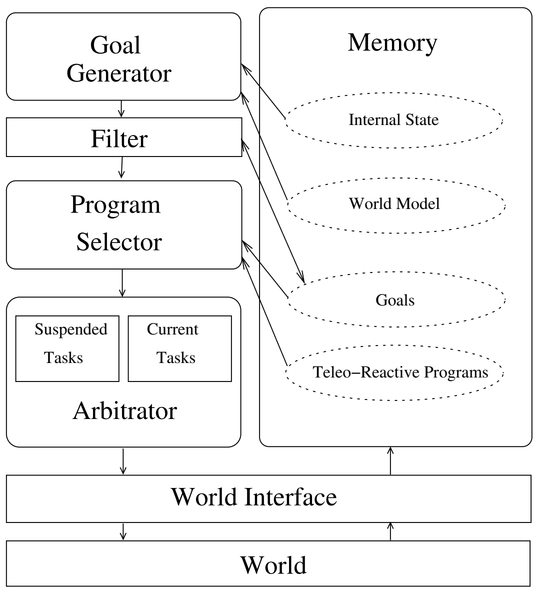

Figure 1: GRUE

Our architecture contains four main components: memory, a set of goal generators, a program selector and the arbitrator (see Figure 1). The system runs in cycles. During each cycle, information from the environment is processed by the world interface and then placed in memory. Goal generators are triggered by the information in memory, then associated with programs by the program selector, and then executed by the arbitrator.

The world interface takes information from the environment, in this case a game engine, does any necessary pre-processing, e.g., computing the distance between game objects and the agent, and stores the results as resources in the memory module. Goal generators are triggered by the presence of particular information in memory. They create appropriate goals, computing the priority values as necessary. The program selector is a simple look-up function, which matches the condition specified by a goal to the success condition of a program. The program replaces the condition in the goal structure, creating a task. The arbitrator manages a list of current tasks, and decides which task(s) to run at the current cycle.

A resource is anything necessary for a rule in a program to run successfully. Resources are stored in the agent's world model and represented as lists of lists, where each sublist contains a label and one or more values associated with that label. Each of these sublists represents a property. The ID and TYPE properties are required for all resources. In addition, resources may list additional properties as required by the application. For resources representing physical objects, these might include location, colour, or shape.

A goal consists of an identification string, a priority, a type, and a condition to be made true. This condition is the same as the end condition in the program that will achieve the goal. Maintenance goals, where the agent is attempting to maintain a condition, are a special case, distinguished from achievement goals by the type field.

A resource variable is used by a program to specify a required resource. A resource variable is a 3-tuple containing an identifier for the variable, a set of required properties, a set of preferred properties. The required properties list specifies those properties that must be present for a resource to be bound to the variable. When several resources are available, the resource matching the largest number of preferred properties will be bound. To use a resource variable in the condition of a rule, the programmer uses a bind function which returns a binding if one can be made, and otherwise returns false. A binding is a pair containing a variable name and a resource.

Programs are pre-written teleo-reactive programs extended with resource variables. A TRP is a list, containing an identification string, a list of arguments, and 1 or more rules. The rules are evaluated in order, with the first rule whose condition is true proposing some number of actions to execute.

Tasks are created from goals by the program selector, and contain an identification string, a priority, a type, and a program. Runnable Tasks are those tasks that have had enough resource variables bound to make one or more rules runnable. Suspended Tasks are tasks which are not runnable due to lack of resources. Tasks achieving maintenance goals persist in the arbitrator, even when the goal condition (currently) achieved. As long as the condition is maintained, the rules in the TRP will not fire, but the task can still use resources. These maintenance goals should have a appropriate priority so they can be used to prevent the character from disposing of necessary items or "forgetting" to maintain a crucial condition.

3. GOAL ARBITRATION

It is the job of the arbitrator to execute as many tasks as possible. To do so, it must allocate resources to the tasks, resolving conflicts where possible. This may involve suspending currently executing tasks and/or resuming suspended tasks as new tasks are proposed by the goal generators and the set of available resources change, e.g., due to changes in the game environment or the passage of time.

The arbitration process first allocates resources to each task, then allows each program to propose actions, then checks the list of proposed actions for conflicts before actually executing them.

The main criteria used for binding resource variables is that a lower priority task may never take resources from a higher priority task. We allow the highest priority task to bind its resources first. Each resource variable is matched against the resources stored in memory. A resource variable can only be bound to a resource which has all of the properties listed in the required properties list. In the case of a tie, the resource variable will bind to the resource with the largest number of preferred properties. If two or more resources have all the required properties and the same number of preferred properties, then one resource will be chosen arbitrarily. When a resource variable is bound during one execution cycle and then used again during the next cycle, it remains bound to the same resource unless that resource is no longer available.

Once the variables have been bound and each task has proposed an action, there may still be some conflicts. For example, two tasks might propose moving in opposite directions. The easiest way to resolve such conflicts it to simply discard the action proposed by the lower priority task. GRUE chooses a set of non-conflicting actions, favouring actions proposed by higher priority tasks.

4. IMPLEMENTATION

We have implemented a complete GRUE agent for the Tileworld environment [7]. The Tileworld agent performs well, and we have demonstrated that using resources with preferred properties is advantageous.

We have also developed a basic agent for the game Unreal Tournament, using the Gamebots toolkit [5]. The basic agent uses a fixed set of goals, with predetermined priority values, and shows predictable behaviour. Our future work will focus on comparing GRUE agents to similar agents without the goal autonomy features of GRUE. In particular, we will implement a complete GRUE agent for Unreal Tournament, and compare it against our basic agent with fixed goals.

5. REFERENCES

[1] S. Benson. *Learning Action Models for Reactive Autonomous Agents*. PhD thesis, Stanford University, December 1996.

[2] S. Benson and N. Nilsson. Reacting, planning and learning in an autonomous agent. In K. Furukawa, D. Michie, and S. Muggleton, editors, *Machine Intelligence*, volume 14. The Clarendon Press, 1995.

[3] M. DePristo and R. Zubek. Being-in-the-World. In *Proceedings of the 2001 AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, 2001.

[4] N. Hawes. Real-time goal orientated behaviour for computer game agents. In *Proceedings of Game-ON 2000, 1st International Conference on Intelligent Games and Simulation*, pages 71–75, November 2000.

[5] G. Kaminka, M. M. Veloso, S. Schaffer, C. Sollito, R. Adobbati, A. N. Marshall, A. Scholer, and S. Tejada. Gamebots: A flexible test bed for multiagent team research. *Communications of the ACM*, 45(1), January 2002.

[6] J. Laird and J. Duchi. Creating human-like sythetic characters with multiple skill levels: A case study using the soar quakebot. In *AAAI Fall Symposium Seris: Simulating Human Agents*, November 2000.

[7] M. Pollack and M. Ringuette. Introducing the tileworld: experimentally evaluating agent architectures. In T. Dietterich and W. Swartout, editors, *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189, Menlo Park, CA, 1990. AAAI Press.