MOTIVATED AGENTS

William Bricken
August 2004

Contents

Caveat: This technical description is not informed by game design, characters, or strategy. All technologies mentioned below are currently implemented efficiently and without error.

Purposive Behavior

Imagine a virtual character (in any context) interacting with a player's avatar. What virtual behavior would enhance the gaming experience? I suggest: purposive behavior. Humans, by nature, perceive or project purpose upon activity they see in the world. Should a rock move toward a hole, people will project an intention by the rock to fall into the hole. An efficient software architecture that imbues purpose into virtual characters is outlined below.

Motivated Agents

Artificial intelligence research has developed agent architectures, and has given agents a wide variety of characteristics and capabilities. The *motivated agents* approach is sufficiently advanced to be applied in commercial settings. Motivated agents are an amalgam of four known agent technologies:

reactive planning -- the agent has a set of goal-oriented skills that apply directly to environmental circumstances

goal/reward functions -- the agent seeks to achieve goals based on a value structure (an objective function)

reinforcement learning -- the agent can prioritize useful and frequent actions

meta-actions -- the agent can introduce new actions and goals

Advantages

The motivated agent architecture solves several problems that have have been obstacles in other approaches:

-- purposive agent behavior that both makes sense and provides challenge
-- unique dynamic reactions to player activity
-- changing and unexpected behavior that is consistent with game play
-- highly efficient implementation that meets frame-rate and memory constraints
-- low programming overhead.

Overview of a Motivated Agent Architecture

A computational cycle for an agent consists of sensing, triggering a plan, and acting; a cycle is usually constrained to fit within the display frame-rate of the game.

Reactive plans are incremental steps that can be taken toward a goal, given specific circumstances. Each plan is valid for one cycle, so that activity is opportunistic and not disrupted by changing circumstances. Which actions are triggered depends upon the state of the environment and whether or not the plan helps an agent reach its goal (as determined by the goal function). Plans that have been successful in the past are given a higher priority. Some plans construct new plans and goals rather than trigger observable actions. A skill is a plan that requires a sequence of steps; it may or may not play out depending upon dynamic changes in the environment.

Motivated Agent Methodology

The development methodology for motivated agents is:

-- develop a library of actions and skills for the particular game
-- identify how actions and skills can achieve goals (may incorporate learning)
-- define goal functions for each agent type
-- run agents in game context (may include statistical refinement of reactive plans)

Actions and skills need only be designed and programmed once. All agents begin with the same set of available actions; environment and learning differentiate them. Agent types are defined by having different goal functions. Behavior is determined dynamically as the result of experience.

Individual differences are the result of different goal functions applied to the same set of skills. Actions and skills are themselves differentiated by reinforcement learning, those that succeed in moving closer to the goal are given priority.

This methodology avoids the brittleness characteristic of other planning and behavior guidance systems. It also avoids the extreme computational and memory overhead of search, neural network, simulated annealing, and other exponential learning technologies. The programming overhead is orders of magnitude less than conventional planning and reactive agent architectures.

Motivated Agent Architecture

The four agent technologies combine in the following manner:

An agent can perceive/sense/input specific characteristics of the virtual environment. Relevant aspects of the current environment are posted to a bulletin board. Each agent uses a pattern-match to identify triggered actions.

More than one action may be triggered in a single cycle. Prioritization based on partial goal achievement determines which is taken.

As well, some actions may require more than one cycle to compute and complete.

Action priorities are categorized into immediate/delayed and conflicting/non-conflicting. Immediate actions are effected during the same cycle; all non-conflicting immediate actions are enacted together. Actions that are required to meet graphic update rates are immediate. Whenever more than one conflicting action is activated during the same cycle, Prioritization is dynamic, determined by the goal function and by learning. Delayed actions are those that require more than one cycle. Enacted actions are posted to the boundary partition.

Motivated agents avoid local minima and degenerate behavior by having plans that modify a goal when it is reached.

Behavior modification through learning is also substantively different in that the learning capability is tightly integrated with the reactive action algorithms. This is necessary so that learning does not need a separate world model and separate algorithms.

Meta-actions

The final component of the architecture is meta-actions. These are actions that occur with the internal partition of the agent, and that modify or add new actions, goals, and goal functions. Meta-actions provide a motivated agent with a vocabulary of new and unexpected behavior. Although adding/changing actions, goal functions, and goals are substantively different, they are all implemented with the same programming technique, that of dynamic macros.

A macro is a function that generates an internal component (ie an action, goal function, or goal), that can later be enacted as a normal component. The essential characteristic of a macro that makes it different than other learning and planning techniques is that its inputs are bound dynamically, during program execution. When environmental information triggers a macro, the macro builds a new internal component and then evaluates that component in the context that it is built in. This context may be different than the context that the macro was originally triggered in, so that the result of a macro construction can be used at a later time. Macro constructions can be permanent or temporary.

Thus, macros respond to current circumstances rather than to pre-programmed circumstances. The benefit is that an environmental model that incorporates all possible circumstances does not need to be constructed.