Model-Based Design of Computer-Controlled Game Character Behavior

Jörg Kienzle, Alexandre Denault, and Hans Vangheluwe

McGill University, Montreal, QC H3A 2A7, Canada

Abstract. Recently, the complexity of modern, real-time computer games has increased drastically. The need for sophisticated game AI, in particular for Non-Player Characters, grows with the demand for realistic games. Writing consistent, re-useable and efficient AI code has become hard. We demonstrate how modeling game AI at an appropriate abstraction level using an appropriate modeling language has many advantages. A variant of Rhapsody Statecharts is proposed as an appropriate formalism. The Tank Wars game by Electronic Arts (EA) is used to demonstrate our concrete approach. We show how the use of the Statecharts formalism leads quite naturally to layered modeling of game AI and allows modelers to abstract away from choices between, for example, time-slicing and discrete-event time management. Finally, our custom tools are used to synthesize efficient C++ code to insert into the Tank Wars main game loop.

1 Introduction

Recently, global sales of the world’s computer game industry have grown higher than those of the movie industry. Consequently, there is a growing demand for technology which supports rapid, re-usable game development accessible to non software experts. Computer games can be roughly classified into two categories: turn-based games (such as board games, adventures, and some role playing games) and real-time games (such as action or arcade games, and real-time strategy games). The kind of artificial intelligence found in computer games is different for turn-based and real-time games.

Board games are usually computerized versions of existing board games. Real board games typically require 2 or more players, but in a computerized version the computer can play the opponent. A good example of a board game that has seen many successful computerized implementations is Chess [6]. In turn-based games and particularly in board games, an artificial intelligence component that plans the moves of a player typically uses advanced search algorithms and heuristics to evaluate many possible future game situations. It then chooses as the current move the one that maximizes the likelihood of winning the game in the future. Timing is not that critical. Since the game is turn-based, the state of the game does not change until a player makes a move. Usually, waiting several seconds for an artificial intelligence component to make a move is acceptable.
Real-time games are very different in nature. The state of the game changes continuously (or in tiny increments), and the screen is continuously updated to present the new game state to the player. Modern computer games usually provide at least 30 frames-per-second updates. In real-time games (with the exception of real-time strategy games) the player usually controls one character (or a small number of characters), and plays within a game environment against a set of computer controlled characters (or in multiplayer games against characters controlled by other players).

In such games, the term artificial intelligence is used to designate the algorithms that specify the behavior of computer-controlled game characters, often also called non-player characters (NPC). The ultimate goal is to make the NPCs’ own actions and reactions to game events seem as intelligent and natural as possible. For example, a guard protecting a building might walk back and forth in front of the main door. If he ever hears shots nearby, he should not simply continue this behavior, but for instance seek cover and call for backup. In its simplest form, such AI can be specified with scripts or rules that specify the NPC’s behavior case by case. More realism can be achieved if the NPC has the ability to analyze a situation and evaluate different options, taking into account even the game history.

We believe that the specification of such advanced real-time AI should not be done within a programming language, but at a higher level of abstraction using visual modeling formalisms. Since the main focus of the models is to define reactions to game events, an event-based formalism seems to be the most natural choice. We decided to use our own variant of Rhapsody statecharts, a combination of state diagrams and class diagrams, for our experiments.

Our paper is structured as follows. Section 2 describes our approach to modeling game AI, and explains the details by designing a game AI that controls the behavior of a tank. Section 3 shows how we used our model to generate code that executes within the EA Tank Wars environment. Section 4 presents some related work and Section 5 discusses the benefits of our approach and concludes.

2 Modeling Game AI

In games or simulations, a character perceives the environment through his senses or sensors, and reacts to it through actions or actuators. For instance, a character might observe an obstacle with his eyes, and subsequently decide to turn left. Our AI modeling framework follows this control-inspired philosophy. The transformation from sensor input to actuator output is described by means of simple components. Each component’s structure is modeled by a class, and its behavior by a statechart. The main mechanism of communication between the components is the asynchronous sending/receiving of events. This lowers the coupling between components and hence makes reconfiguration and reuse easier. In some situations, a component may also synchronously invoke an operation of another component.

The architecture of our AI models is described in Fig. 1. The first level contains components that represent the sensors that allow the character to observe