Cognitive Architectures and Virtual Worlds: Integrating ACT-R with the XNA Framework

Paul Smart*†, Katia Sycara‡ and Christian Lebiere‡
*Electronics & Computer Science, University of Southampton, Southampton, SO17 1BJ, UK
‡Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
†Corresponding author – ps02v@ecs.soton.ac.uk

Within the sciences of the mind, issues of material embodiment and environmental embedding have emerged as important areas of research attention over recent decades. Embodiment and embedding are deemed to be important, it is argued, because extra-neural resources may shape the profile of brain-based processes, and, at least occasionally, they may feature in the realization of what are referred to as 'environmentally extended cognitive systems'. This interest in situated, embodied and extended cognition motivates the development of cognitive computational models that can engage in complex forms of perceptuo-motor processing within highly dynamic and perceptually-rich environments. While the ACT-R cognitive architecture is capable of supporting certain forms of environmental interaction via a set of perceptuo-motor modules, in the majority of cases these modules are used to emulate the interaction seen with relatively simple devices, such as computer keyboards and display screens. In this paper, we show how ACT-R can be integrated with Microsoft's XNA Framework to support sophisticated forms of interaction with 3D virtual environments. The XNA Framework forms part of Microsoft's XNA Game Studio, which provides a managed runtime environment that supports the process of video game development. By demonstrating how ACT-R can be integrated with the XNA Framework, we hope to show how ACT-R agents could be embedded in a range of virtual 3D multiplayer game environments. This capability could be used to support future research efforts associated with the development of computational models of embodied, situated and extended cognitive processes. This work builds on previous efforts to integrate ACT-R agents within virtual environments, most notably the work of Best and Lebiere [1] using the Unreal Tournament game engine.

The approach we have adopted to support the integration of ACT-R models with the XNA Framework relies on the use of TCP sockets to support inter-process communication, as well as the use of a custom .NET API (the ACT-R/XNA API) to support the processing of requests made by ACT-R models to retrieve game state information or engage in certain behaviors. The ACT-R/XNA API provides a `Connect()` method that is called during the initialization of the game. This method opens up a TCP connection on a specified port and prepares the XNA game for receiving instructions (or requests) from ACT-R models. Once this connection is open, an ACT-R model, which may be hosted on a separate machine, can post requests to the game environment using Lisp-based TCP sockets (in this configuration, the XNA game is playing the role of server, and the individual ACT-R model is playing the role of a client).

```
(p move-left
    =goal>
        isa explore-environment
        current-task explore
        task-status move-left
    ==>
    +xna>
        isa action
        type "MOVELEFT")
```

Fig. 1. An example of a rule that uses the custom 'xna' module to post a request to an XNA game. In this case, the rule is requesting that the camera be moved to the left. This corresponds to a 'strafe left' action in video game parlance.

The requests made by ACT-R models will typically relate to the execution of actions that can be made by characters in the game, such as directional movements. ACT-R models can implement these requests by issuing simple commands, such as 'MOVEFORWARD', 'LOOKLEFT', 'MOVERIGHT', 'SHOOT', and so on. Typically, these requests will be generated by the interaction of production rules with specific ACT-R modules, such as an ACT-R motor module. For the purposes of this paper, we created a custom ACT-R module called 'xna' that ACT-R models can exploit in order to post requests to the XNA game. An example of a rule that targets this module is shown in Fig. 1.

When a request is received by the XNA game, it is processed by the ACT-R/XNA API in order to either effect changes in game state or retrieve information from the game environment. Many of these requests involve the implementation of specific motor commands that result in changes to the position or orientation of the game camera. These inevitably result in changes to the visual scene: every movement will entail a change in the relative size, position or visibility of objects in the environment as they appear from the perspective of the ACT-R model that is controlling the camera. In order to coordinate behavior with respect to objects in the virtual environment, the XNA game needs to process the current visual scene and build representations that can be processed by ACT-R's vision module (i.e., the game needs to build representations that can be interpreted by ACT-R as a set of visual location and visual object chunks, each of which encodes the features of game objects that are visible within the camera's current field of view). This amounts to providing ACT-R models with the ability to perceive objects contained within the current visual scene. In order to realize

this capability, we first need to determine which objects are contained within the camera's viewing frustrum. We do this by creating an instance of the XNA `BoundingFrustrum` class and then determine whether the vectors encoding an object's position in 3D space are contained within the borders of the bounding frustrum. The features (e.g., the apparent size and position) of these 'visible' objects are then extracted with the help of the XNA `ViewPort.Project()` method. This enables us to obtain the 2D screen coordinates of 3D vectors that encode, for example, the position of objects in the virtual environment. Once the features of objects have been extracted, they are returned to the ACT-R model in the form of a comma-delimited string. The ACT-R model processes this string in order to create a set of visual location (and corresponding visual object) chunks. These are used to populate the model's visicon, and are thus made available for subsequent processing via the ACT-R vision module. It should be noted that, in the case of the current integration solution, ACT-R models are required to make explicit ('GETVISICON') requests to the game environment in order to update their visual representation of the environment. An alternative (and more realistic) strategy would be for the game environment to automatically update the visicon following any change in the visual scene, for example, after every change in camera position or orientation.

In order to test the integration solution, we created a virtual environment consisting of a simple rectangular room that was adorned with randomly generated picture objects displaying simple geometric shapes. A screenshot of the environment is shown in Fig. 2. The geometry for all objects within the environment (i.e., the walls, floor, pictures, etc.) was generated procedurally using the XNA `GraphicsDevice.DrawPrimitives()` method. In addition, the content of the picture objects within the environment was generated at runtime as part of the `LoadContent` stage of the XNA game loop. This was achieved by using the .NET managed code interface to GDI+ in order to dynamically create XNA `Texture2D` objects that were then assigned to the picture geometry. The environment contains a movable camera that is initially situated centrally within the room. This is the camera that is controlled by the ACT-R model. Using a simple cognitive model, we tested each of the requests that could be issued by the ACT-R agent to the game environment. The game environment responded successfully to each of the requests made by the ACT-R model, and the model was able to create and attend to objects within the dynamically changing visual scene. Given the focus of this initial study, we did not attempt to develop models in which behavioral output was coordinated with respect to visible objects (e.g., for the purposes of engaging in approach or avoidance responses). This is a potential target of future work.

In addition to closing the visuo-motor loop, there are a number of other areas that could serve as the focus of future work efforts. Firstly, it should be noted that the XNA Framework lacks many of the features seen in high-end game development environments, such as those used for the production of so-called AAA (triple-A) games. This potentially limits the applicability of the current solution and necessitates a consideration of alternative 3D modeling/game development environments. After evaluating a number of game development environments, we suggest that the Unity game engine [2] provides a compelling alternative to XNA in the context of

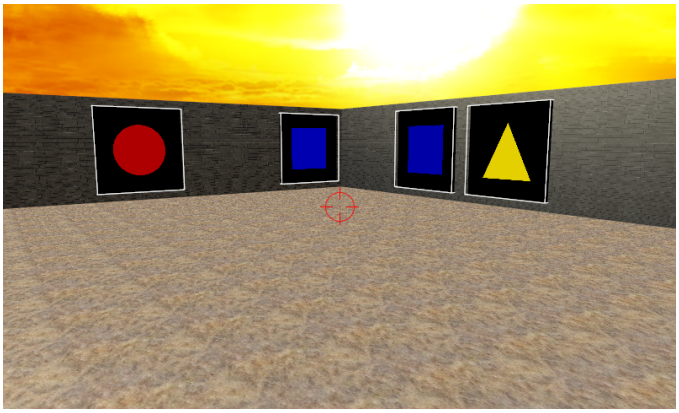

Fig. 2. Screenshot of the virtual environment used for testing the ACT-R/XNA integration solution. Picture objects are indicated by white borders.

the current work. The Unity game engine provides extensive support for the development of visually-rich 3D scenes, and its support for the C# language facilitates code migration efforts for the ACT-R/XNA API. A second focus of attention for future work concerns the communication protocol used to support inter-process communication between XNA games and ACT-R. The current solution relies on the use of a proprietary message format; however, we suggest that future work should aim to incorporate more generic solution strategies, such as the one proposed by Hope et al [3] using JSON. Finally, the current solution has been tested with a single ACT-R model. Future work should aim to extend this capability to multiple ACT-R models. This will provide the basis for future research efforts that seek to use ACT-R for the purposes of investigating issues of embodiment, social interaction and inter-agent communication within highly dynamic and perceptually-rich virtual worlds.

ACKNOWLEDGEMENT

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] B. J. Best and C. Lebiere, "Cognitive agents interacting in real and virtual worlds," in *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Interaction*, R. Sun, Ed. New York, New York, USA: Cambridge University Press, 2006.

[2] "Unity - Game engine, tools and multiplatform," URL: http://unity3d.com/unity [accessed: 2014-07-13].

[3] R. M. Hope, M. J. Schoelles, and W. D. Gray, "Simplifying the interaction between cognitive models and task environments with the JSON Network Interface," *Behavior Research Methods*, in press.