2nd Hellenic Conf. on AI, SETN-2002, 11-12 April 2002, Thessaloniki, Greece,
Proceedings, Companion Volume, pp. 177-188

Application of Autonomous Agents for Crowd Scene Generation

Adam Szarowicz[1], Peter Forte[2], Juan Amiguet-Vercher[1], Petros Gelepithis[2]

[1]WhiteSpace Studio, Kingston University, Penrhyn Road, Kingston-Upon-Thames, KT1 2EE, UK
{a.szarowicz, jamiguet}@kingston.ac.uk, tel. +48 (0) 20 8547 7984
[2]School of Computing and Information Systems, Kingston University, Penrhyn Road, Kingston-Upon-Thames, KT1 2EE, UK
{pforte, p.gelepithis}@kingston.ac.uk

Abstract. Generation of animated human figures especially in crowd scenes has many applications in such domains as the special effects industry, computer games or for the simulation of the evacuation from crowded areas. Currently such scenes have to be created by human animators using dedicated software packages. This is both expensive and time-consuming. Our "FreeWill" prototype proposes and implements a cognitive architecture designed for easy creation of animated scenes with many autonomous agents interacting in various ways. Agents maintain an internal model of the world and fulfil their goals. The design allows for easy co-operation of different software packages (geometry engine, AI engine, sensing/actuating modules, simulation managing unit and a visualization environment). The implementation language is Java and the graphics package is 3D Studio Max. The requirements capture process and design is being conducted and documented in the Unified Modeling Language (UML).

1 Project's origin

In its current state, computer assisted animation frees the individual from frame by frame animation of characters by allowing the animator to specify key frames and leaving the software to interpolate between them. This has prompted developers to look for further advances in automatic animation in the form of enhanced character autonomy. A good test case is the ability to generate automatic crowd scenes in which avatars inter-mingle without collision and interact in various ways, for example by shaking hands. Applications for this exist in computer games, digitized special effects for post-production of feature films, and simulations of the safe evacuation of crowded areas in the event of fire or other disasters.

Software packages such as 3D Studio Max or Maya (widely used by production studios) provide limited ability to do this but current results fall far short of the long term

potential. One reason is that any enhancements to the avatar behavior repertoire are usually very specific with limited scope for extendibility or generalization.

Most of the work is still done by the animators and any automatic autonomous behavior in the animation sequence tends to be limited and 'hard-wired'.

The aim of our project is to address the shortcomings of present systems by proposing and implementing an extendable cognitive architecture designed to accommodate goals, actions and knowledge and supported by a well engineered design (Forte at al. 2000, Vercher-Amiguet 2000). The design must allow for easy co-operation of different software packages – namely a geometry engine, AI engine, sensing/actuating models, simulation managing unit and a visualization environment. The implementation language being used is Java and the graphics package is 3D Studio Max. The design is being undertaken and documented using the recently standardized Unified Modeling Language (UML) (OMG, 1999).

2 Subsystems

An avatar is implemented as an agent, i.e. something: "that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors" (Russel and Norvig, 1995). Our model builds on the cognitive architecture proposed by John Funge (1998, 1999). Each avatar comprises two main components – its body and an AI engine. The body is the 3D geometric representation of the avatar (together with its position and speed), whereas the AI engine ('mind') supplies all functionality necessary for world representation, goal planning, sensing and acting, and emotions.

The software is divided into 6 principal subsystems:

Geometry engine – provides a full range of geometric functions within an object-oriented framework. This engine uses bounding boxes for easy detection of collisions and also for object hiding. Further extensions of this package may include incorporation of a physics engine.

AI engine – this subsystem is responsible for managing the processes of space sensing and perception (interpretation of information), goal planning, collision avoidance, taking appropriate actions and other cognitive tasks. For example this will allow an avatar to recognize 'friends', initiate actions such as handshakes and update plans to achieve desired goals. Ultimately it is proposed that the AI engine should also model emotions.

Actuating/sensing modules – these components are responsible for physical interaction with the environment (actuators) and gathering raw information from the environment (sensors) – although higher-level information processing (perception) is left to the AI package. At the moment sensors capture only visual data but the model also

accommodates other types of input such as auditory and haptic channels. In the current model, visual data is "sensed" merely by having access to the pre-defined objects of the virtual world via a VisionCone object, which captures the world state and allows for update of the avatar internal world model. In future this unit may incorporate image recognition subsystems which are being developed by other research groups at Kingston University.

World objects – this subsystem models all physical entities within the virtual world.

Management module - this is based on discrete event simulation and a queue handler (a scheduler for Decker at al. 1995, Lux at al. 1995) enabling the autonomous behavior to unfold within the virtual world by passing control to appropriate world objects (including avatars) according to the event which is currently being processed. An event is an object which includes:

- name of the object it should be sent to
- time of execution
- type of action which should be performed
- additional parameters.

Visualization engine – this part of the system is responsible for displaying the world model and the interacting avatars. At the moment this is performed by the package 3D Studio Max (with which the system interacts through Max Script or step files, a sample of which is depicted in Fig. 1). However the system can also interface other products e.g. those using motion capture files. The visualization engine must also allow for rendering the scenes and for saving the final animation.

```
biped.AddNewKey LarmCont3 0
biped.AddNewKey RarmCont3 0
      sliderTime = 10
rotate RForearm3 30 [-1,0,0]
biped.AddNewKey LarmCont3 10
biped.AddNewKey RarmCont3 10
      sliderTime = 20
rotate RForearm3 80 [0,0,-1]
biped.AddNewKey LarmCont3 20
biped.AddNewKey RarmCont3 20
```

Fig. 1. Sample script for generating avatar behavior

The design is being carried out in UML, as depicted in figures 2 and 3. Fig. 2 shows a simplified class model of the system. Fig. 3 presents the overview of the system design – divided into main components.

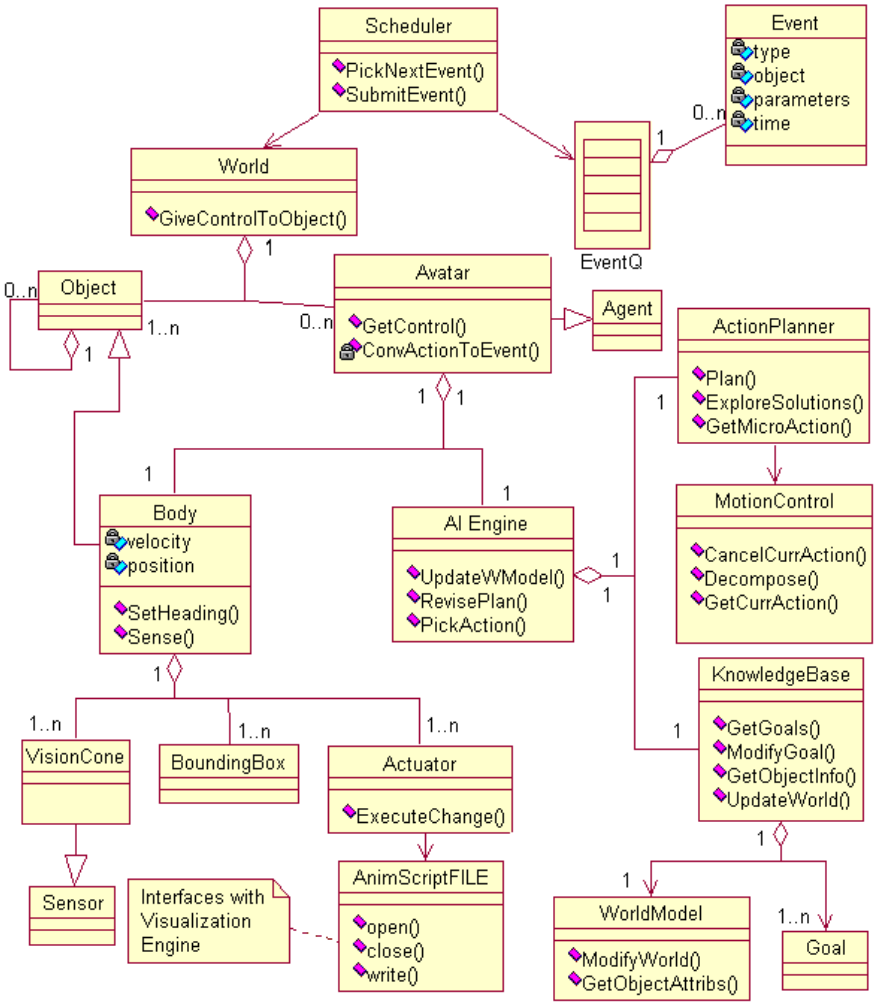

Fig. 2. The system class diagram modeled in UML

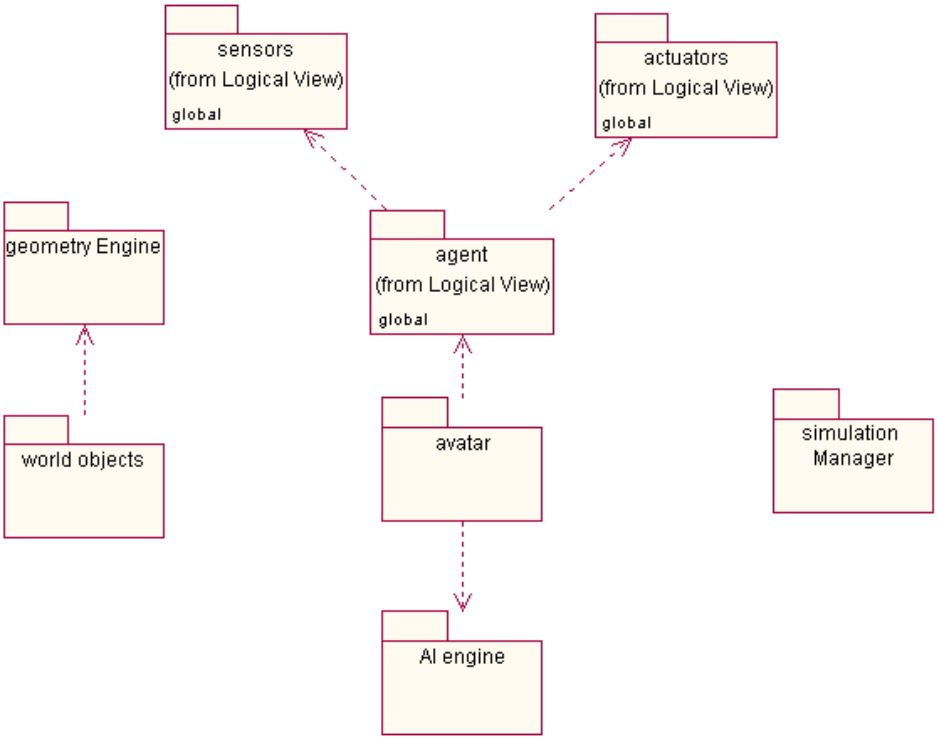

Fig. 3. Agent and Avatar packages modeled in UML

3 Operation of the AI Engine

3.1 Algorithm for Acting and Sensing

The main simulation loop (Fig. 4a) is located within the Scheduler class which consecutively picks events from the Event Queue. The chosen message is then decoded and passed to the World. Control is then passed to the appropriate World Object (which in most cases is an avatar) and necessary actions are taken. These can be

- an 'act' action – such as move a hand or make step. The action is rolled out (avatar's state variables are updated) and a new line is added to the MaxScript file. This action

returns a new sensing event, which should be inserted to the Event Queue (see Fig. 4b).

- 'sense' action – which means that the avatar should compare the perceived fragment of the world with its own internal model. Then the avatar has a chance to 'rethink' its plan and possibly update goals and the planned set of future actions. This action returns a new acting event (see Fig. 4c).

The returned actions are inserted to the Event Queue and the time is advanced so that the next event can be selected.

If thinking events were only generated by acting events then there would be a danger that temporarily passive avatars (e.g. an avatar waiting for a bus) could get stuck. To prevent this a PeriodicEventGenerator class is introduced which generates cyclic sensing events for each avatar.

3.2 Knowledge Base and Goal Derivation

An avatar's behavior is goal directed. The first and main goal is provided by the user – this is called a primary goal and represents the aim of the simulation for that avatar. An example may be 'get to the end of the sidewalk'. However the fulfillment of this goal may be enacted with accomplishment of smaller secondary goals which are set and assessed by the avatar. An example may be 'shake hands with your friends'. Such goals are a part of the avatar's knowledge. In some cases they can be deduced from the current world state (e.g. in order to cross the street a secondary goal such as 'make sure there is no car speeding towards you' can be defined). The rules of avatars' behavior are stored in the knowledge base as sets of facts ('speeding cars are deadly') and rules ('a car going towards you should be avoided'). The knowledge base also provides logical information about static world objects (a sidewalk is 'walkable') and other avatars (a list of friends). (For further details on UML design of a knowledge base for multiagent communication see Raphael and DeLoach 2000).

3.3 Action Decomposition

The goal-planning algorithm constructs plans using the notion of an action as a generic planning unit. An action can be defined on various levels of specialization – from very general ones (e.g. 'get to the end of the sidewalk') to fairly detailed activities ('do the handshake'). The most detailed actions (microactions) are said to be at level 0. They correspond to action events in the Event Queue and also to MaxScript file entries. In general every action is specified by a pre and postcondition and is implemented by an avatar's member function, which will perform the action and update the state of objects affected by it. These objects can be World Objects or parts of the avatar's body. The planning unit (ActionPlanner) operates on actions from level N to 1 – creating general

plans and then refining them. The ActionPlanner maintains the chosen plan from which the last action is submitted to the MotionControl unit. It is then decomposed into a set of level 0 microactions (e.g. handshake consists of a set of arm and hand movements) which can be executed one by one. Any change in the plan may cause the list of microactions to be dropped and new ones to be generated.

4 Conclusion

Figures 5 and 6 show details from the animation video. To begin with a prototype has been implemented in which avatars interact as in Fig. 5 and 6. The further work will concentrate on the design of the AI engine to provide more flexible behavior including more elaborate construction and modification of goals and plans.

```
Scheduler.DoSimulation()
{
   while (simulation)
     PickNextEvent(&event)
     object = event.GetObject()
     newEvent = World.GiveControlToObject(object, event)
     {
          newEvent = objectArray[object].GetControl(event)
          {
                    if (event.type == ACT)
                    {
                              DoActing(event.detailedType)
                    }
                    if (event.type == SENSE)
                    {
                              DoSensing()
                    }
                    return newEvent
          }
     }
     SubmitEvent(newEvent)
}
```

Fig. 4a. Main simulation loop (Java pseudcode)*

```
DoActing(detailedType)
{
    switch    detailedType
    case      STEP
    {
              SetHeading()
              SetVelocity()
              SetPosition()
    }
    case MOVEHAND
    {
              SetHandPosition()
    }
    case MOVEARM
    {
              …
    }
    Actuator.ExecuteChange()
    {
              AnimationScriptFILE.write(line)
    }
    return new SensingEvent
}
```

Fig. 4b. Acting algorithm (Java pseudcode)*

```
DoSensing()
{
        image = Body.Sense()
        {
                return VisionCone.GetImage()
        }
        Mind.UpdateWorldModel(image)
        {
                KnowledgeBase.ModifyWorld(image)
                {
                        WorldModel.ModifyWorld(image)
                }
        }
        Mind.RevisePlan()
        {
                ActionPlanner.Plan()
                {
                        KnowledgeBase.GetGoals()
                        ExploreSolutions()
                        KnowledgeBase.GetObjectInfo()
                        {
                                WorldModel.GetObjectAttribs()
                        }
                        CreatePlan()
                        lastAction = SelectLastPlannedAction()
                        MotionControl.Decompose(lastAction)
```

```
                }
        }
        action = Mind.PickAction()
        {
                microA = ActionPlanner.GetMicroAction()
                {
                        return MotionControl.GetCurrentAction()
                }
                return microA
        }
        return ConvertActionToEvent(action)
}
```

Fig. 4c. Sensing algorithm*

**A word on the notation:* The notation applied in the algorithms is a Java pseudocode in the form of

```
Object.method()
{
 method body in a recursive format
}
```

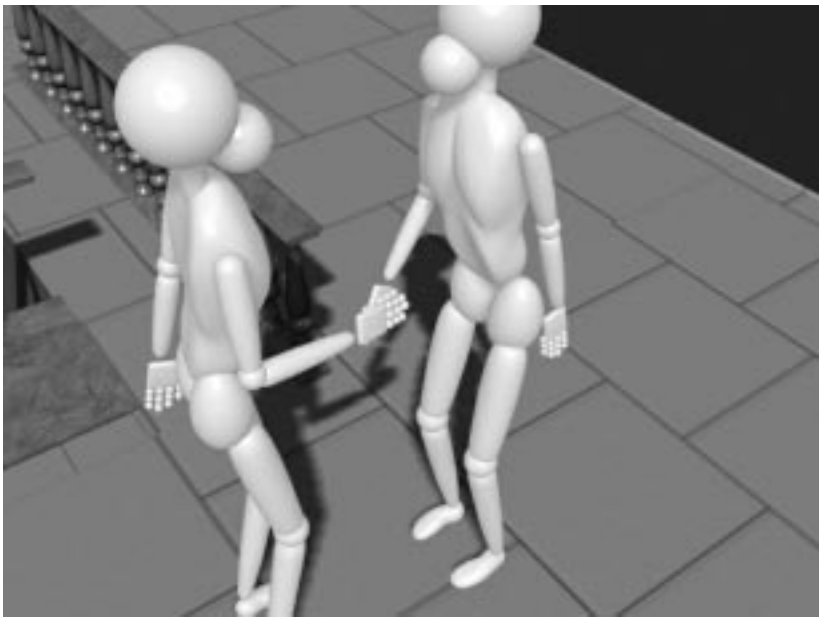

Fig. 5. Agent interaction: shaking hands

Fig. 6. Agent interaction: crowd scene

References

1. P. Forte, J. Hall, P. Remagnino, P. Honey (Kingston University / Whitespace Studio), VScape: Autonomous Intelligent Behavior in Virtual Worlds, Sketches & Applications Proceedings, SIGGRAPH 2000, Aug 2000.
2. J. Vercher-Amiguet. (Kingston University / Whitespace Studio), Automatic Crowd Scene Generation, Electronic Imaging, Vol 11 no 1, Dec 2000.
3. Object Management Group. OMG Unified Modeling Language Specification, June 1999. Version 1.3. See also http://www.omg.org
4. S. Russell, P. Norvig. Artificial Intelligence. A Modern Approach. Prentice Hall, 1995.
5. J. Funge. Making Them Behave: Cognitive Models for Computer Animation. PhD thesis, Department of Computer Science, University of Toronto, 1998.
6. J. Funge, X. Tu, D. Terzopoulos. Cognitive Modeling: Knowledge, reasoning and planning for intelligent characters. Computer Graphics Proceedings: SIGGRAPH 99, Aug 1999.

7. K. Decker, V. Lesser, Designing a Family of Coordination Algorithms, Proceedings of the International Conference on Multiagent Systems 73-80, American Association for Artificial Intelligence, 1995
8. A. Lux, D. Steiner, Understanding cooperation: An agent's perspective. Proceedings of the International Conference on Multiagent Systems 261-268, American Association for Artificial Intelligence, 1995
9. M. Raphael and S. DeLoach. A Knowledge Base for Knowledge-Based Multiagent System Construction, Proceedings of the National Aerospace and Electronics Conference (NAECON), Dayton, OH, October 10-12, 2000. http://en.afit.af.mil/ai/papers.htm
10. A. Szarowicz, J. Amiguet-Vercher, P. Forte, J Briggs, P Gelepithis, P Remagnino, The Application of AI to Automatically Generated Animation, Australian Joint Conference on Artificial Intelligence, AI'01, Adelaide, Dec 10-14, 2001