An Event Tree for Assisting the Director Agent in Automatic Camera Positioning

#60340 #60340 #60340 #60340

#60340

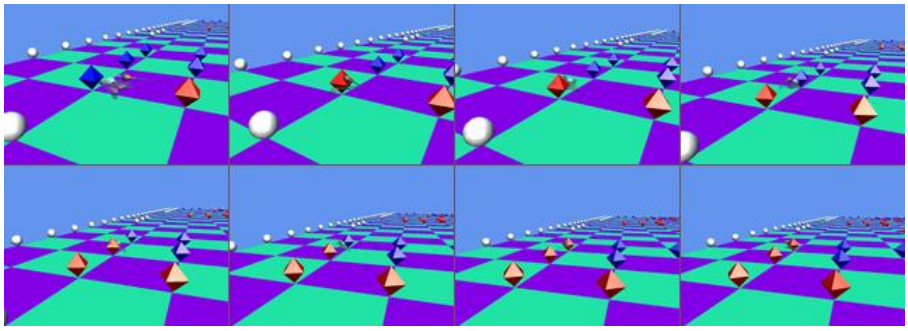

Figure 1: *Non-interactive replay generated through the technique of recognition and prediction of physical actions.*

Abstract

The camera freedom in 3D virtual environments is enabling applications to use techniques similar to those used in cinematographical presentations. However, with the presence of interactivity in these environments, there is no predetermined sequence of events that enables optimal dynamic positioning. Hence, it is necessary to use techniques for the recognition and prediction of actions in order to position the camera in a satisfactory manner. This paper presents an Event Tree to help an agent director in the automatic placement of cameras that uses techniques for the recognition and prediction of actions in an interactive virtual environment based on the physical state of the simulation.

Keywords: virtual cameras, actions prediction, cinematography, tree structures

Author's contact:
#60340

1. Introduction

Nowadays it is very common to see non-interactive sequences in 3D games which enrich the story and draws more attention and interest from the player. However, these sequences are strongly linked with a story script with actions of the characters and default camera settings. In applications where we want to real-time broadcast a game match for an audience, where there is no knowledge of what may happen, or even in games where the creation of an instant replay is reasonable, it is practically impossible to get good results by using preset camera takes.

In cinematography, the camera is one of the main components used to highlight what we want to show, and to introduce subjectivity to the scene, depending on the viewing angle, on what is receiving focus and on the time of taking of the scene [Martin 1985] 3D game developers have noticed very soon that the movies productions used very interesting effects which could be adapted for the production of games. By incorporating similar techniques, developers could add more effects and excitement to their productions [Hawkins 2005]. Terms such as zoom and pan were added to the internal handling of the virtual camera. In fact, the programming of a virtual camera is becoming increasingly similar to the commands that a camera operator receives in the real world, with instructions at a very high level of abstraction. The experience of the film industry explains and justifies the use of each type of camera-taking, constituting a proper language, which today is also used in the production of 3D presentations and games.

In interactive 3D applications, the proper use of a virtual camera is very difficult due to the unpredictability of the events. In applications with pre-defined sequences, the director already knows exactly what will happen with the elements of the scene, and this makes possible the optimal placement and orientation of the camera, according to the desired results. But when the changes that will occur with the objects of the 3D environment are not known in advance, it is necessary to use alternative techniques for camera handling to achieve good results. We must, above all, recognize the actions and identify the relevant objects for the scene to be transmitted in an appropriate way, still considering the probability of occurrence of another event. After the recognition of actions in the interactive environment, the system still needs to choose which is the most relevant at each moment, displaying it in a convenient way to get the

expected emotion. To do this we must choose a good position for the camera, define which objects will get the focus and determine the time of taking. At this moment the decision tree enters into activity.

When an event has been chosen to be filmed, a low computational cost strategy to position the camera is to predict what will happen in the next few moments inside the virtual environment, based on the physical simulation [Pires et al. 2008]. To illustrate the use of the technique, a game was developed, where the user has his view at gameplay, and after that, when level is complete or player dies, his moves are shown again as a replay of all past actions. This also represents the view that the audience should have when watching.

Trees are one of the most important data structures with innumerous applications in Computer Science. They are composed of a finite number of elements, allowing a hierarchical arrangement of its elements, enabling processes such as indexing and search to be performed very efficiently. Different problems and applications have led to the development of several variants of tree data structures, such as binary trees, AVL trees and octrees [Cormen et al. 2001].

This work aims to show the use of an Event Tree, whose goal is to assist the director agent in the placement of an automatic camera in a virtual 3D environment. The Event Tree takes into account the main objects in a given virtual environment, and yields an acceptable configuration that can be used by the application to position the camera in the environment and to frame the target by the camera, including some film effects such as tilts and trajectories.

2. Related Work

Since the emergence of cinema in the nineteenth century, several authors have been studying new techniques to make their products more attractive to the spectators. Cinematography schools have also multiplied and grown. Today one can find very good articles and books on film techniques, such as the work of Martin [1985], and specialized websites such as Mnemocine [2008]. More recently, these same techniques have been widely adopted by games, to increase the player's immersion in the 3D environment. Moreover, they began to think about the possibility to show a 3D game match just as in sports broadcasts. Drucker's work [1994] explores the feasibility of this use of film techniques for computer games, showing a very positive outlook.

There are several works aiming to facilitate the problem of positioning the camera in the virtual environment, such as the ones of He et al. [1996] and Amerson & Kime [2001]. Some of them use Domain-Specific Languages (DSLs), which are extremely more useful for positioning the camera through higher level commands than the conventional programming languages, but have the disadvantage of presetting the placement of cameras and activating one camera or another depending on the events happening in the environment.

In other works, such as Drucker's [1994], Hermann & Celes' [2005] and Pires et al. [2008], the camera is positioned in a more automatic way, without user intervention. But these methods provide a poor result if compared to the use of DSLs. The system of Hermann and Celes [2005] also distributes the work of positioning the camera into 3 modules: screenwriter, director and videographer. The experiments of Pires et al. [2008] used Microsoft's XNA framework to develop a complete game that shows the player a non-interactive replay of his moves, using action recognition and physical prediction to assemble a satisfactory way to film the scene. However, with this framework, the proposed module videographer is merged with the actual mechanism for rendering images. With this, the director module can output this data directly to the graphics pipeline of the application.

Pinhanez [1999] is one of the main works dealing with action recognition in virtual environments. His work proposes a formal representation for actions in the context of interactive narrative systems involving real and virtual actors. This representation is based on the decomposition of actions in elementary sub-actions, being context-sensitive, with the possible temporal relations between sub-actions and the main action, represented by a network of constraints with three possible values: past, present and future (PNF networks). The recognition of actions within the virtual environments is done through the propagation of the temporal state of these PNF networks, taking into account the state of elementary sensors. However, in his work, there is no experiment with systems including physical simulation engines, which could provide more information about the state of the objects, or even predict events before they occur. In such environments with physics engines, a collision may be predicted depending on the speed and direction of the movement of an object, its distance to an obstacle and its maximum coefficient of friction or diversion.

The XNA framework, launched by Microsoft in 2006, became very popular for academic and commercial use. Its focus is to produce games for Windows and X-box 360. Within the target audience of XNA there are students and independent developers. Its documentation is full of examples and with complete explanations about the mathematics involved in graphical processing and programming techniques. With its growth, several websites specialized in this tool have been created; they offer from tutorials to ready-to-use classes and libraries. There are now many studies demonstrating different techniques for camera implementation in XNA. Among them there are Riemers' [2008] and Fegelein's [2008] websites. They explain all mathematical fundaments in the manipulation of virtual cameras in 3D environments.

There are also many books on the XNA framework, such as the Nitschke's [2007] and Carter's [2007].

This paper is an extension of Pires et al. [2008] work, using the same experimental game to illustrate the technique. It presents an application capable of recognizing the actions occurring in the environment and positioning the camera in a way that the events of the virtual environment and future predicted events can be considered, through physical calculations and use of information passed by the game engine.

3. Simulation Prediction System

Below are described the main techniques that support the functionality of dynamically placing cameras in the virtual environment. These are action recognition, the prediction of physical simulation and prediction of state machines. Figure 2 shows a fluxogram of an application's game-loop.

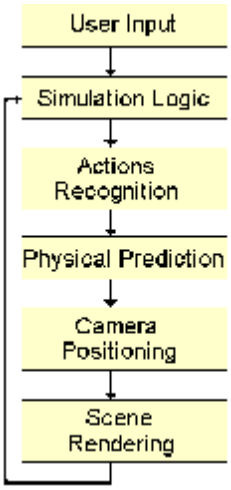

Figure 2: *A game-loop cycle.*

3.1 Recognition of actions

A virtual environment is usually composed of objects that are able to perform certain actions within its domain. These actions can be simple or have the ability to affect other objects, or even be able to change drastically the entire environment. For the cinematography, even static objects have semantics within the environment and will influence the scene in some way. Therefore it is necessary an appropriate mechanism to recognize what each object is doing and what kind of influence a given set of objects have on their neighborhood or on the virtual environment as a whole. Shortly speaking, to recognize the actions of a virtual environment means to correctly identify what is happening at any given moment within the virtual environment.

More objective or specific actions make up the game engine itself, or are part of a state machine of some object, or are represented as properties of some class. In these cases, to realize that an action just occurred becomes a trivial task, since there may be a marker or a variable indicating that it occurred or is occurring. Below are some examples of objective actions at different types of games:

- Level start;
- Character begins to move;
- Car collides with a wall;
- Enemy dies;
- The tick of a timer.

However, modeling all the actions in the game engine or those included in the objects's behavior can be, sometimes, a very difficult task due to the subjectivity of some actions and events. These are known as subjective actions [Pires 2009]. For example, in a soccer game match, is not easy to recognize that a user prefers and tends to make his moves on the right side of the field. In a racing game, the player can for some reason choose to never overtake a certain car. Actions that depend on the player's behavior are much more difficult to recognize because it is necessary an interpretation of events over time.

In order to facilitate the recognition of such type of events it is possible to use an agent that stays monitoring the actions of objects and the state of the environment in time, and that informs the application whether it finds a pattern of behavior that is similar to the expected events or relevant to the construction of the film scene. This ends up with the camera positioning and their next moves. The term "agent" denotes a software engineering concept that describes an entity capable of acting with a degree of autonomy in order to complete tasks [Barella et al. 2007], in this case, in regular intervals of time, observing what happens in the environment.

3.2 Actions Recognition Agent

The actions recognition agent keeps track of the actions of the objects and the state of the environment in time, and warns the application when it finds a pattern of behavior that is similar to the expected events or relevant to the construction of the film scene. Figures 3, 4 and 5 show the main data structures used by this agent.

Figure 3: *A list H_i of the few past events.*

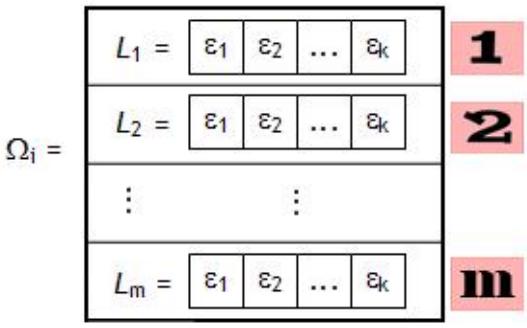

Figure 4: *A set W_i of expected events L_x.*

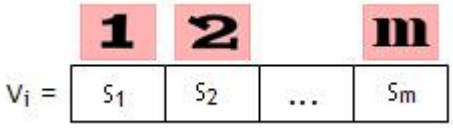

Figure 5: *A list V_i of cinematographical meanings.*

Firstly, at design time, we must define which set of objects O = [o_1, o_2, ..., o_n] will be monitored and how many past events will be considered. So, every game-loop, the state of each object $o_i \in O$ will be stored in a list called H_i. This list is responsible for storing the object's states history. There must be a list H_i for each object instance o_i we want to monitor. Alternatively, the current state of an object $o_i \in O$ can be stored in H_i as new events occur, or at regular intervals of game-loops. Moreover, instead of storing states of objects, the list H_i can be completed with sets of sentences involving boolean variables relevant to the assembly of the scene. This must be decided, modeled and set at design time.

The list H_i is then compared with a set Ω_i = [L_1, L_2, ..., L_m] of behaviors or actions L. Each element L represents an expected action or behavior for that object for the entire game and must be defined at design time. Each entry of this set Ω_i is a list L = [ε_1, ε_2, ..., ε_k], containing a sequence of states for the object or boolean expressions involving any element of the application. A list L_x is very similar to the list H_i, but it represents an action or condition with relevant cinematographical meaning. At the defined rate, the history H_i has to be compared with all the m entries in L_x of the set Ω_i, until the agent concludes that the past events form a relevant event for filming, or not. The comparison between the list L_x and the history H_i is done by evaluating the similarity between their elements. If both lists are sufficiently similar, then the action represented in L_x is given as recognized, and a new reference to Lx will be added to the list R_i of recognized actions. R_i is an ordinary list that is always empty at the start of the process.

If the few past events, represented by H_i, are similar to an expected action, represented by L_x, then this action is understood as an important action to film, thus having a cinematographical meaning, such as distress, calm, caution, expectations etc. Every action L_x has a corresponding index x in a vector V_i, which stores the cinematographical meanings for each $L_x \in \Omega_i$. Each position $x \in V_i$ keeps the cinematographical meaning of the action L_x. This is easily done just associating the indices of Ω_i and ones of the vector V_i, according to Figures 4 and 5. This will influence the decisions of the director agent later when it starts operating.

Alternatively, the algorithm for the recognition of actions can be performed at regular intervals of game-loops or when a new event happens. The choice of which approach to use depends on the mechanics of the game and how much work the system can bear. If, in a testing phase, the system becomes overloaded, a solution may be to decrease the frequency of work of this agent.

3.3 Physical Simulation Prediction

The prediction of the physical simulation is a technique that helps to find out what will happen in the next few moments. It almost always refers to movements and the update of positions in any way, since these are the main object of physical simulations [Pires 2009]. Since an object in a virtual environment may be under various influences, the prediction helps to determine the likelihood that a given event is going to occur or a given event can be concluded. The prediction returns a value depending on the probability of occurrence of an event, and it will be evaluated by the director agent.

At design time the designer has to define what he wants to predict so that he can implement the required calculations. Later in the scene assembly process, the event tree will consider every predicted event and so the director agent can choose a good camera angle to film the scene.

The prediction of events in the virtual environment can be used to assist the recognition of subjective actions, and is especially useful for automatic placement of cameras. For example, in a virtual environment where a motorcycle is riding onto a ramp, it is possible, through kinematics calculations, to find out where and how it will fall, thus taking the appropriate action accordingly to it, to show the fact in the best way possible, even before the motorcycle jumps. Of course, by using such prediction techniques, the application can perform other processes, such as making the pilot leave the motorcycle if the jump is going to be disastrous. Physical prediction is also useful for improving artificial intelligence. For example, the behavior of a smart fighter who calculates his next attack, according to time remaining to the enemy land after a jump [Pires et al. 2008].

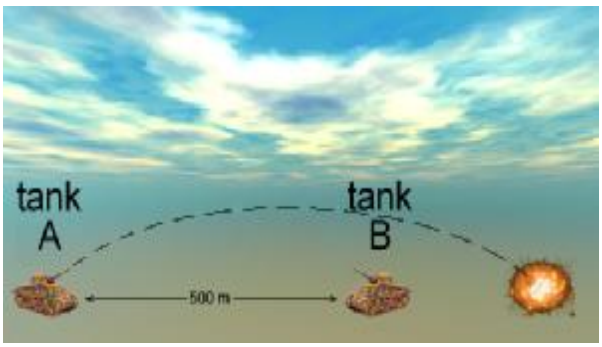

Figure 6: *A tank A launches a missile targetting tank B.*

For example, in a scene with a war tank A that launches a missile against another tank B, and one tank is 500 meters (or any measurement unit) away from the another, in a 3D environment with physics that ignores air resistance, as in Figure 6, the success of the tank A can be calculated at the moment of the shot. If that missile was launched with a speed v_0 of 100 m / loop and angle Φ equal to 30 degrees, and gravity g equal to 10 meters / loop, by applying kinematics formulas we can determine where the projectile will fall (range A) and the time (Δt) it takes to travel the path:

$$A = \frac{v_0^2 \sin(2\phi)}{g} \qquad \Delta t = \frac{2\, v_0 \sin\phi}{g}$$

Figure 7: *Formulas for parabolic movement [Taveira et al. 2009]*

Applying the formulas, we would know that the missile will fall to 809 meters from tank A, nine game loops after its shot, concluding that the tank B remains untouched. In this case, the system can now be prepared to film the action in an interesting way for the viewer [Pires 2009].

To achieve a reliable prediction, it is necessary to be careful with some problems. It is always necessary to consider which movements the object tends to do, like gravity force and other forces applied to the object. The other objects that are around should also be considered, especially to know the distance from the subject and how this distance will vary in the next few moments. The arrangement of objects in the area is also an important issue for the cinematography. Finally, it should be considered how the user or other components may intervene in the scene. If these are not taken into account, it may in some cases reduce the predictability of actions, leading the director agent to think that nothing important is going to happen.

Due to the computational complexity of some physical calculations or to obtain information about many objects, other data structures can be used, in order to speed up these operations. For example, it is very fast to get the positions of objects in a discretized space, where these objects are represented by a matrix [Pires et al. 2008]. In this case, we simply calculate the position of an object in the environment by the position which is stored in the matrix.

3.4 State Machine Prediction

The prediction through the state machine of an object is made in cases where an event with relevant meaning for filming can be identified by the sequence of states that the object undergoes. For example, a soccer player can very likely get tired after running hard for a given period. So, when this player runs too hard, the director agent can get ready to film him getting tired. If this really happens, the camera will film the fact precisely.

The implementation of prediction based on state machines structures may involve the study of graphs and their theoretical aspects, which are not in the scope of this work. However, it is also an important way to improve the viewer experience when watching a real-time game match.

4. Event Tree Operation

The camera positioning is the result of all the previous steps. The game engine is responsible for raising the events relating to the game logic or to the properties of a given object. Using this information, it is possible to recognize subjective events and also predict future events. Then, the director agent can position the camera according to the cinematographical language being used, because it has access to all the information it needs.

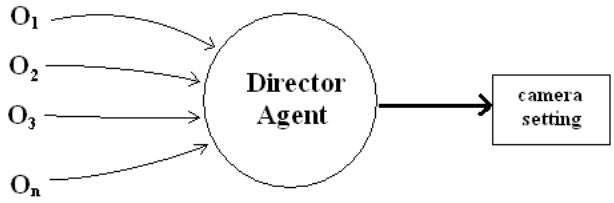

Figure 8: *Director Agent input and output.*

As in Figure 8, the director agent receives as input the state of the environment and a reference to the main objects involved, i.e., a list of pointers O = [o_1, o_2, ..., o_n] to access the properties of the main characters and their enemies or any other objects o_i that have some influence on the scene. In practice, it is easier to make the director agent to have direct access to the state of the virtual environment, the characters' classes and lists of enemies or other objects. Moreover, for each object, the director agent has access to the list of recognized subjective actions R and the results of physical prediction. With this, the desired result is a scene with satisfactory camera settings.

At this point the director agent begins to assemble a way to film the scene. A decision tree A, called Event Tree, is built dynamically and through this the final result C will be reached. The root of A serves only as a starting point; A will have as many children as the

number of objects in the list O = [o_1, o_2, ..., o_n], The notation n_k indicates the number of nodes in the k-th level of the tree. For instance, n_2 represents the number of nodes in the second level of the tree. Afterwards , the director chooses one of the leaves at random and begins to pursue a path C in the tree; the other leaves are discarded to minimize memory consumption. This movement means that the object o_i was chosen to be the main subject of the scene. From this node on, new leaves are generated; the number of leaves is an n-tuple with the same number of recognized subjective actions r, plus the number of predicted actions p, plus the number of recent past actions u. That is, the number of leaves on the third level of the Event Tree is:

$$n_3 = r + p + u$$

Again the director agent chooses a leaf randomly; this time the choice represents the action to be filmed. This action in the third level of the tree is related to the object chosen in the second level. According to the elements in this third level, the director's choice may be an action that just happened, a recognized subjective action or a predicted action that can happen in a few moments. Each one of those has its corresponding cinematographical value, which was set at design time. For performance issues, after the director chooses a leaf, the others are ignored.

Finally, the director agent creates its last set of leaves in the Event Tree. This time the number of leaves will be equal to the number of objects involved in the action chosen by the previous step. The last random selection of the director determines which objects are framed and how the camera will be actually positioned, following the concepts of the film language set at design time. Figures 9, 10 and 11 show the entire process:

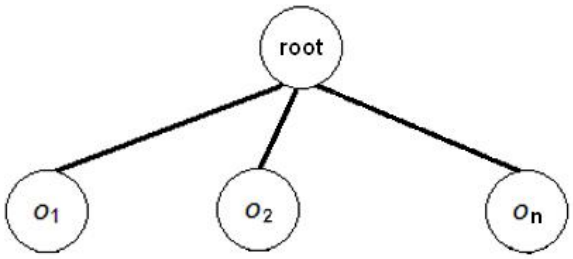

Figure 9: *Step 1: monitored objects are represented as tree nodes.*

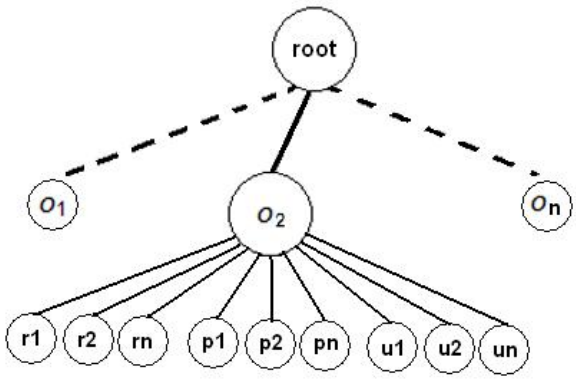

Figure 10: *Step 2: recognized subjective actions, predicted actions and past objective actions are represented as tree nodes.*

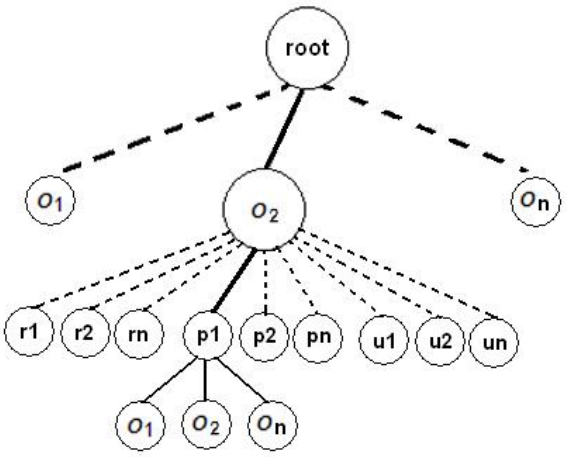

Figure 11: *Step 3: objects related to the action are represented as tree nodes.*

The experimental game makes use of this structure. The director agent works all the time to assemble a good camera setting while the game logic executes [Pires et al. 2008]. Suppose that the ship is trying to get the blue crystals, which are needed, in an area completely surrounded by red crystals that are lethal, according to Figure 12. The scene will be built based on the Event Tree. At the second level of the tree (the first is the root) there will be only one node, because the ship (object o_1) is the only object being monitored. At the third level there are the following nodes:

- The ship has turned right (objective action ε_4);
- A blue crystal was collected (objective action ε_7);
- The ship has turned left (objective action ε_5);
- A blue crystal was collected (objective action ε_7);
- It was recognized that the player is trying to get blue crystals in a certain order to gain more points (subjective action L_1);
- There is a high collision probability with a red crystal (predicted action p_1);

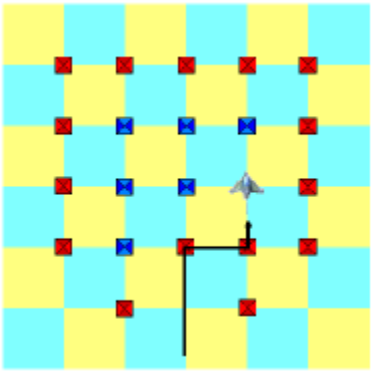

Figure 12: *Black arrow represents the path of the ship.*

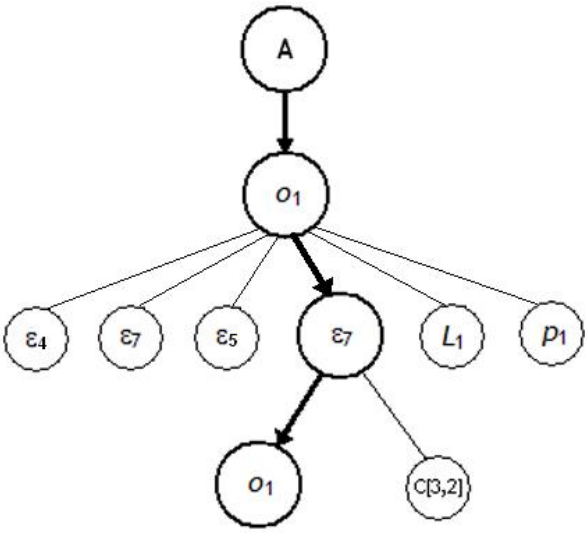

Figure 13: *An example of a full Event Tree.*

As shown in Figure 13, the Event Tree will have 6 nodes at the third level, each representing a situation. The director chooses randomly one node to continue. Choosing the path ε_7, the other nodes are ignored and two new leaves are created for this node, each representing an object involved in the action: the ship and the newly collected crystal (represented as C[3,2], because crystals are arranged in a matrix C). Each leaf in this fourth level has parameters such as position, size and speed of the objects o_i and will interfere in the way of positioning the camera, acting as a local modifier.

The path taken by the director agent on the tree is stored, because the actions previously selected have to be consulted again later to read their cinematographical meaning. Hence the importance of borrowing the film industry experience in the development of this type of application. Based on those cinematographical information, the minimum time for the take can be determined, certain camera movements can be applied, and the position of the camera related to its target can be set.

After that, the scene is rendered and the time of the take starts to be counted. When the time is over, the director agent looks up all data structures again in order to repeat the entire process. Depending on the type of environment or how the user controls or interferes in it, this can happen more quickly or more slowly. As this approach is susceptible to problems, mainly due to the difficulty to predict the user's commands, it is interesting to make the engine throw a control action at a fixed time rate, to force the director agent to reset its parameters [Pires 2009].

The random factor in the behavior of the director agent is due to two reasons: the first is to guarantee that the process does not become fully deterministic. The second is due to the limitation of working in real time. To make the director evaluate each action and the suitability of every object could bring a much greater computational effort than the game itself, and also involve theoretical issues that are outside the scope of this work. The random selection has also the advantage that a single replay, can have different camera settings, each time it is seen. For transmission in real time, each viewer can get a different picture filmed from a different angle.

5. Conclusion

This paper presented a decision tree structure for assisting a director agent in the assembly of a non-interactive scene. The proposed method combines techniques of action recognition, prediction of events through physics simulation within the virtual environment, and cinema concepts. The idea around the Event Tree for assisting the director agent in its choices was introduced, and its usage was explained. In association with film techniques, the Event Tree aids the director to reach good results.

The experimental game built in Pires et al. [2008] was used in order to test the technique. With some adjustments in the virtual environment, the technique could be applied correctly and in an optimized manner. Another essential factor for the success of the technique was the addition of a randomness factor, which avoids the determinism and is computationally faster than analysing all possibilities. As a final result, it produces more interesting scene viewing effects.

The experiments have shown that the technique does not introduce any major overhead, mainly because of the use of random choices by the director agent. Nevertheless, with this approach, the director is not aware of the relevance of the actions. The approach presented in this paper do not rely on the use of more intricate artificial inteligence techinques such as neural networks to evaluate which path would bring the best camera setting. By doing this it avoids large numbers of computational steps and thus being appropriate for real time applications such as games.

References

AMERSON, D. AND KIME, S., 2000. *Real-time Cinematic Camera Control for Interactive Narratives*.

CORMEN, T. ET AL, 2001. *Introduction to Algorithms*. MIT Press, 2001.

DRUCKER, S., 1994. *Intelligent Camera Control for Graphical Environments*. Doctoral Thesis, Massachusetts Institute of Technology.

ERLEBEN, K., 2002. *Module Based Design for Rigid Body Simulators*.

FEGELEIN, 2008. *Microsoft XNA Framework; Creating a Freelook Camera* [online]. Available from: http://www.fegelein.com/?p=18 [Accessed 8 August 2008].

HAWKINS, B., 2005. *Real-Time Cinematography for Games*. Charles River Media Publishing, 2005.

HE, L. ET AL., 1996. *The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing*.

HERMANN, R. AND CELES, W., 2005. *Posicionamento Automático de Câmeras em Ambientes Virtuais Dinâmicos*.

MARCHAND, É. AND COURTY, N., 2002. *Controlling a Camera in a Virtual Environment*.

MARTIN, M., 1985. *A Linguagem Cinematográfica*. Editora Brasiliense, 1985.

MNEMOCINE, 2008. *Linguagem e técnica cinematográfica* [online]. Available from: http://www.mnemocine.com.br [Accessed 8 August 2008].

NITSCHKE, B., 2007. *Professional XNA Game Programming for Xbox 360 and Windows*. Wrox Publishing, 2007.

PINHANEZ, C., 1999. *Representation and Recognition of Action in Interactive Spaces*. Doctoral Thesis, Massachusetts Institute of Technology.

PIRES, D. ET AL., 2008. *Posicionamento de Câmeras através de Previsão das Simulações Físicas*.

PIRES, D., 2009. *Posicionamento de Câmeras por meio da Simulação Física*. Master Thesis, Universidade Federal Fluminense.

RIEMERS, 2008. *Quaternion Camera* [online]. Available from: http://www.riemers.net/eng/Tutorials/XNA/Csharp/Series2/Quaternions.php [Accessed 8 August 2008].

SEUGLING, A. AND RÖLIN, M., 2006. *Evaluation of Physics Engines and Implementation of a Physics Module in a 3D Authoring Tool*. Master Thesis, Umea University.