Introducing movement and animations to virtual victim in USARSim

Author: S. Katt (#6151248) **Supervisor: A. Visser**

June the 26st, 2012
University of Amsterdam
Faculty of Science
Institute of Informatics

UNIVERSITEIT VAN AMSTERDAM

FACULTY OF SCIENCE (FNWI)

POSTBUS 94216

1090 GE AMSTERDAM

SCIENCE PARK 904

Bachelor Thesis (18 EC)

Abstract

USARSim is a simulator used for rescuing tasks in the RoboCup Rescue based on the Unreal Engine. In order to fulfil the rising expectations of this cup, not only the robots but the environment has to keep up with the demands. Vital to the environment is a good representation of victims in need of rescue. Up to now victims in USARSim have not been capable of performing movement, controlled animations or sounds. This thesis provides a method for designing such behaviour for these victims using Unrealscript and Unreal Kismet. As a result a scenario has been made in which two victims escape a building, applying different behaviour patterns. Future research could be directed towards more sophisticated behaviour, introducing Artificial Intelligence, heat emission of victims and a deeper understanding of blending animations in the Unreal Engine.

Table of content

Acknowledgement

I would like to thank my supervisor, Arnoud Visser, for his advice and technical support throughout the project.

Introduction

After a disaster involving human lives, such as the attack on the World Trade Centre in New York, it is desirable to be able to immediately investigate the area in order to rescue survivors. The area is, however, often too hostile for humans to enter in order to find and help survivors. As a result, lives are lost during the period that the area is inaccessible for humans. Highly developed robots, however, could access such environments.

Robots could possess properties such as high resistance to heat, a small body capable of moving in tight spots, or other properties beneficial to rescuing. So far, robots are not used for rescue operations yet, however, they are used for recognition missions on various locations, such as on planet Mars (Matthies, Gat, Harrison, Wilcox, Volpe, & Litwin, 1995). Furthermore, the development of technology is fast; according to Bill Gates (2007) it is comparable to the period when Microsoft was founded and when he was at the cutting edge of computer technology, which was a dynamic period for technology. It is plausible that robots can and will be used in the future for operations such as exploration of post-disaster areas and localisation of survivors.

The RoboCup rescue project is a product in the field of computer-technology. The trigger for the project was the Great Hanshin-Awaji earthquake, causing more than 6500 casualties and resulting into various requirements for information systems (http://www.robocuprescue.org)
These are those requirements:

- Collection, accumulation, relay, selection, summarisation, and distribution of necessary information.
- Prompt support for planning disaster mitigation, search and rescue.
- Reliability and robustness of the system during routine and emergency operations

The RoboCup rescue project aims to fulfil these needs, their goal is to
"Promote research and development in this socially significant domain at various levels involving multi-agent team work coordination, physical robotic agents for search and rescue, information infrastructures, personal digital assistants, a standard simulator and decision support systems, evaluation benchmarks for rescue strategies and robotic systems that are all integrated into a comprehensive systems in future."
(http://www.robocuprescue.org)

RoboCup

In the Robocup, teams from all over the world compete with each other for the best robot. The Roborescue is divided into two strands: a project for real robots and a simulation project. In the Project for real robots teams build and program their own robot which is then tested on its capabilities of navigation through and exploration of an (unknown) area. The goal is to navigate through the environment, create a correct map of its surroundings and locate any possible survivors. The environment is a reconstruction of an imaginative post-disaster situation (figure 1). The purpose of this project is to investigate new technologies for navigating and recognition. Any successful method may be useful in the future; these robots have already proven to be capable of mapping and detecting victims in post-disaster environments and could be used for real rescue operations.

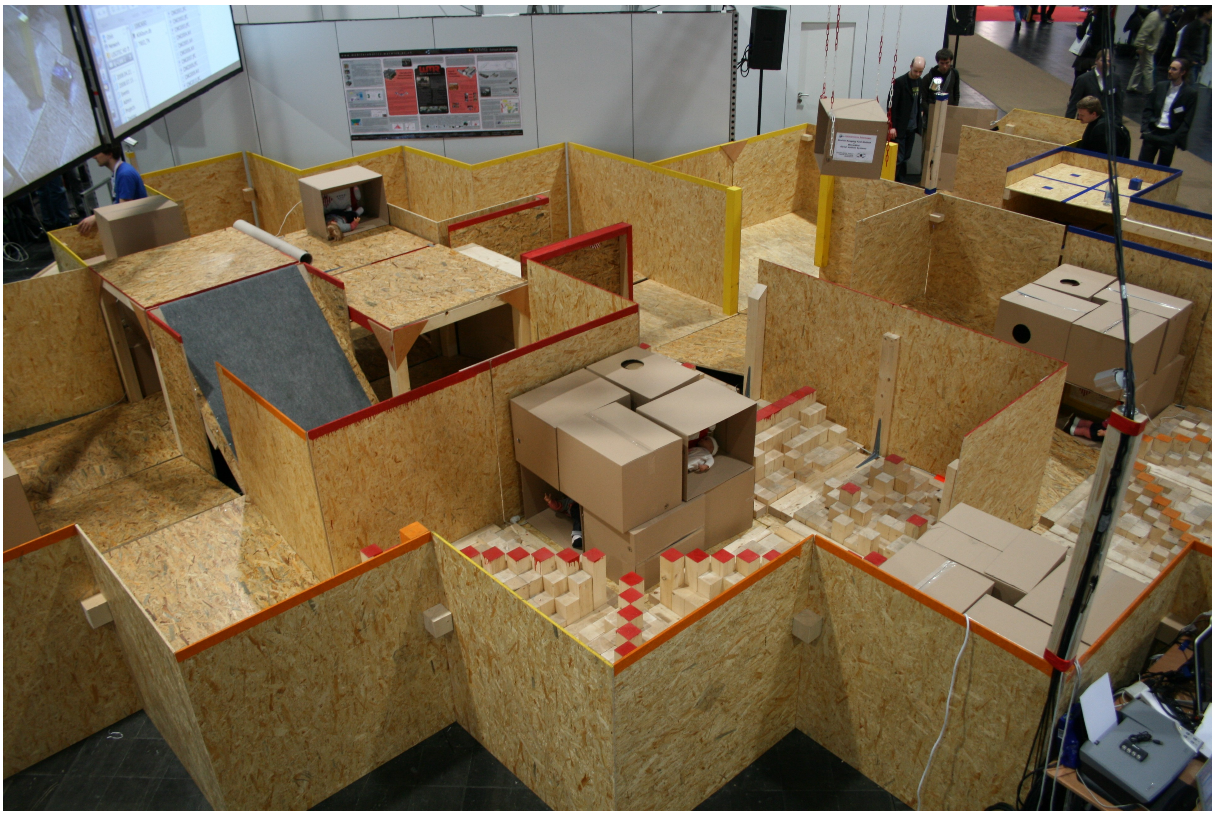


Figure 1: RoboCup_Rescue_2008_German_open_test_arena. Source: http://en.wikipedia.org/wiki/File:RoboCup_Rescue_2008_German_open_test_arena.JPG

The simulation project is quite similar to the real robots project: in both projects different teams make an attempt at creating the best robot, suitable for navigating through environments and localizing victims. The difference is that the simulation-project does not involve the real world, because the robots and their environments are simulated on a computer, much like a video-game. The teams compete with each other between the best simulated robots in a simulated environment (made by the creators of the RoboCup rescue, figure 2).

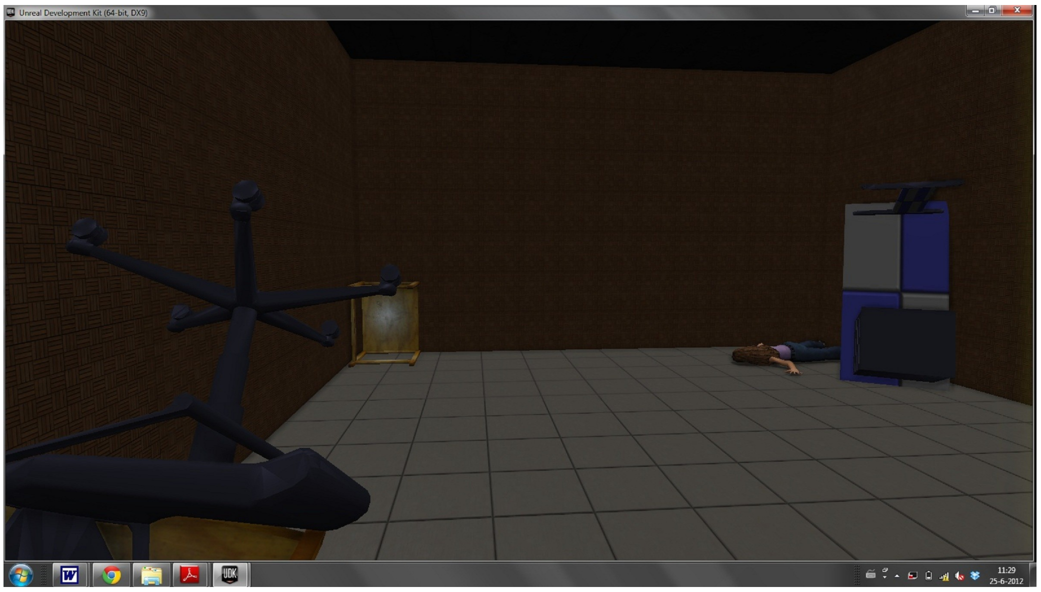


Figure 2. Simulated post-disaster environment

Simulation

The importance of the second cup is that simulating is less expensive and of shorter duration that creating a real robot. Additionally, real robots are easily damaged by incidentally bumping into objects, or by simply losing balance, due to a fault in program or design. These problems are overcome by simulations. Furthermore, one could improve a robot by testing its design and program in a simulated environment before ever building it. Lastly, it is possible to make a simulation of technology that is not yet available now, but could be in the near future. It is possible to create anything in a simulation, which means that an upcoming technology can be tested before it is (or can be) built for real. Simulating future technology may lead to a deeper understanding and implementation of the technology in robots, speeding up the research and development in general. A simulation, for example, has been used for testing and proposing a method for using an omni directional camera for distance estimation (Nguyen and Visser, 2009).

Crucial to testing simulated robots in a simulation is the quality of the simulated environment: A simulation test is useless if the best robot in the simulation does not perform among the best in a real situation, and as the performance of the simulated robots of the teams in the cup increases every year, it demands the simulated environment to keep up. Realistic simulation of the real world, including possible events or actions caused by the robot or survivors, is necessarily in order to test the robots. Victims are an important aspect of this environment, not only is placing them in the environment a goal on its own, it is important to simulate their behaviour correctly.

There are various ways to make a representation or simulation of a human and both research on human behaviour in post-disaster situations (victims) as simulating this behaviour has been flourishing. New methods for simulating crowds (Brenner, Wijermans, Nussle & De Boer, 2005) (Helbing, Farkas &Vicsek, 2000) (Shao & Terzopoulos, 2007) and individual victims (Chaim, 2010) are presented in order to keep up with the knowledge we gain from research on humans. Different views on the representation of humans have been introduced, varying from cue balls (BrownBridge, 2008) and electrons (avoidance of obstacles could be simulated like electrons “bounce” of each other due to magnetic forces) to as realistic as possible. In our case, involving rescue operations, it is of importance that humans are as realistic as possible, as explained below.

Human recognition is critical to localisation of human, because in a real post-disaster situation a robot has to be able to localise, and thus recognize, a human at all costs. Secondly is that the robot has to be able to interact with a human correctly, it has to be able to react on actions such as being approached by a human (a possible reaction could be not moving, in order to not harm a human), this area of study is called Human-Robot interaction (HRI). Furthermore, if a victim is capable of making sounds and movements, having the capability of interpreting these sounds and detecting the movements would be a significant advantage. This means that a realistic representation of a victim must also lead too a more suitable robot-design.

Problem definition

So far, victims in Roborescue cup have had limited behaviour. Victims do no interact with or react on the environment and neither do they move from their place. Their behaviour is restricted to playing a certain animation; in the previous cups, victims would lie, sit or stand, waiting to be discovered the entire simulation. Chaim (2010) has provided a design for victim behaviour, presenting a method for animating a victim. Using Chaim’s findings as a basis, what kind of human behaviour is possible to simulate in USARSim? This thesis attempts to give an introduction towards providing the victim classes various functionalities, in order to expand the possibilities of simulating behaviour. This thesis has been divided into a few sections.

The next section describes typical human post-disaster behaviour, then the second section covers the software used in the Roborescue cup. In the third section a possible scenario is described and a method is presented for creating such scenarios, followed by the result and discussion section. Lastly steps in how to create behaviour and packages in UDK are presented in the appendix.

Related Work

Human behaviour in post-disaster situations

Human behaviour has been widely studied and there is a significant amount of insight on post-disaster behaviour. Articles about post-disaster behaviour have been produced since 1950, in which it was made clear that humans do not enter a 'panic-state' or lose all common sense (Quarantelli, 1960). Quarantelli claims that, instead of the common misconceptions that human will flee the area as soon as possible, we tend to react quite "cool" and show different rational behaviour from rescuing others to estimating the dangers of the situation.

Other studies as well claimed that humans do not simply enter a panic-state unless certain conditions are met (Fritz & Williams, 1957). One of those conditions is that the human(s) in question must believe that there are only a limited amount of escape routes combined with the believe that these escape routes are closing quickly. People try to protect their immediate associates even during hectic and violent impacts.

Because it is evident that even in post-disaster situations, humans show rational behaviour, it can be concluded that a simulation of human behaviour in such situation must include reacting/interacting with the environment and rescue robot, searching for other survivors and, of course, moving to an exit.

Human detection

Several characteristics of humans are used for human detection and it depends on the achievements made in the future which will prove to be most successful. Chaim (2010) claims that promising subjects of human recognition include skin (Visser, Slamet, Schmits, Jaime, & Ethembabaoglu, 2007) and object recognition (Flynn, 2009). These approaches are based on colour, shape and pattern as features to distinguish humans from their surroundings. Other characteristics could be heat emission, because living beings distinguish themselves from animate objects by emitting heat and this could be used for human recognition. In order to be able to test human detection in a simulation, realistic texture and materials of both humans as the environment, as well as realistic sensors are required.

A thorough understanding of UDK is necessarily, in order to implement a realistic representation of victims in the USARSim by applying knowledge about human behaviour as described in related work. Therefore, the next section is dedicated to the software used in this thesis and in the Robocup rescue.

Software

UDK and USARSim

The software used for simulations in the Roborescue cup is called USARSim (Urban Search and Rescue Simulator). This software is an extension of (based on) the Unreal Engine.
The Unreal Engine is a programmed environment made for the purpose of creating games; various are created using this engine, such as Unreal Tournament 3 and Borderlands 2. The Engine is flexible, capable of supporting numerous desired environments and, because of its

commercial origin, the graphics are realistic. It is easy to create an environment using code of the Unreal Engine because the programming language is object orientated. Therefore creating a simulation can be done by extending and modifying classes from the Unreal Engine, in stead of starting from scratch. Because of these reasons, the Unreal Engine has become the basis for the Roborescue simulation software.

The Unreal Development Kit (UDK) is a release of the Unreal Engine in which it is, as the name suggests, possible to develop your own game or, in our case, simulation. It comes with native language code and development code. The native language code is the core language of the Unreal Engine, and is not accessible for regular developers. The development code in the language unrealscript, on the other hand, is accessible and can be used for your own convenience, be it extending or modifying it. The kit also contains an editor (figure 3), in which it is possible to create environments, add objects and even game play (events, actions) without touching the unrealscript. This is ideal for developers unknown to programming, it is fairly easy to create a landscape with tree's and mountains, add a few lights, and "play".

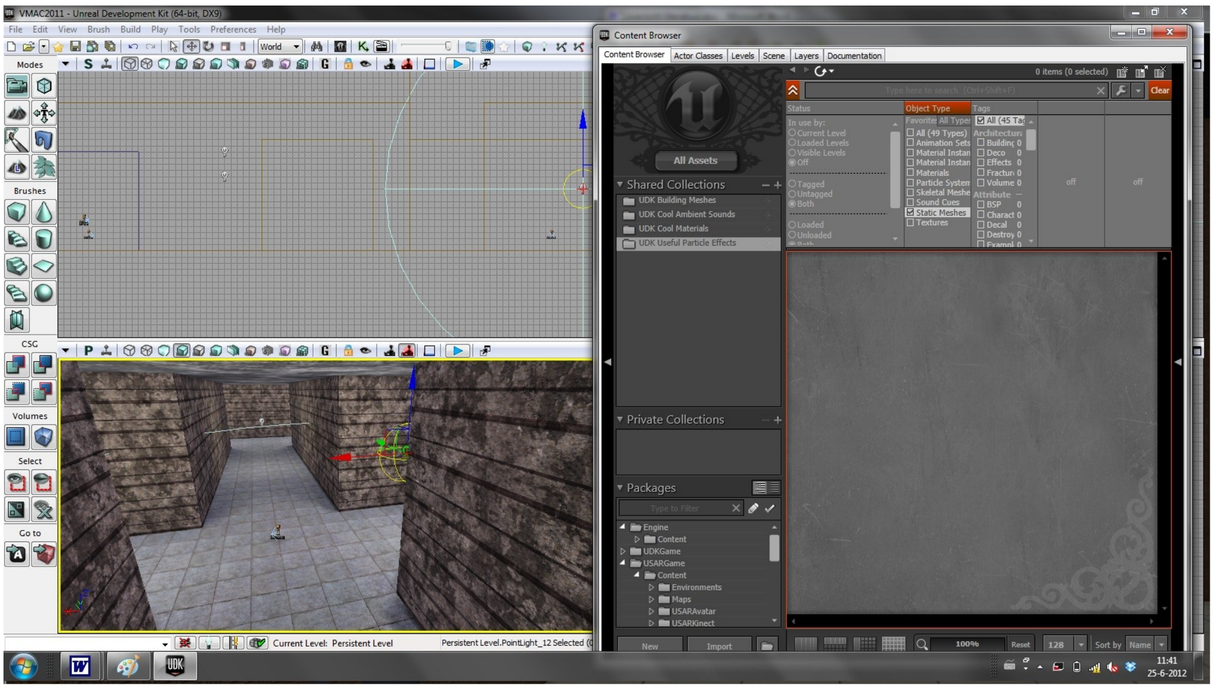


Figure 3. UDK editor

USARSim is a software extension of the Unreal Engine which creates the environment and provides the instances, objects and behaviour necessary for simulating a rescue operation. USARSim provides code, additionally to the already available UDK, in the form of its own classes, such as rescue robots, but also more abstract instances, smoke for example. Lastly USARSim provides various tools, of which one is controlling a robot using *IridiumSE*. USARSim has been proven to be able to realistically simulate real-time events. Although most often the results of methods or algorithms are less effective as implementation in the real world, USARSim simulations have been proven valid (Noort, 2012) (Balaguer, Balakirsky, Carpin, Lewis & Scrapper, 2008).

Unrealscript and Kismet

Objects (including victims) in USARSim are made out of various assets and classes (code). One asset is of importance to this thesis, the animation tree, the others will only be briefly described in the appendix A.

When the victim is walking, a combination of walking animations should be played, a diagonal walk is the combination of a forward walking animation and a walk side-ways animation. The animation tree is the asset responsible for determining what animation to play, accordingly to the situation. In addition to the assets, a victim class, *victimpawn* (extending pawn class of UDK), tells UDK (and USARSim) that all these different assets together form one object, the victim. (Chaim, 2010). The only thing left for the victim to show behaviour is Artificial Intelligence, which is a controller that determines the victim's behaviour. Whereas the animation tree is used for displaying animations, the controller determines when the conditions are met for the victim to start showing a type of behaviour. In other words, the controller tells the engine when and where to walk to or when to show a certain animation, while the animation tree actually shows the requested animations. Additionally, they UDK controllers work with states, which means that behaviour can be defined in particular states, such as "chasing" or "idle". Events could put a victim in a state, activating a particular desired behaviour, providing multiple possibilities for Artificial Intelligence.

The victim classes consist of three classes, the 1) *MaleVictim* and *FemaleVictim* classes, the 2) *VictimPawn* class and the 3) *VictimController* class. The first classes extend the *Victimpawn* class, initializing the sex-specific assets. *Victimpawn* extends the regular *UTpawn* classes of UDK, receiving a large amount of code necessarily for most pawn-like behaviour such as moving and rotating. The *Victimcontroller* class extends the UDK *AIcontroller* class, again receiving many pre-built functionalities for intelligent behaviour in pawns.

The actor class is the base class of all game play objects in the UDK, and both the *Victimpawn* as the *Victimcontroller* are indirect children of the Actor. The Pawn classes can be seen as the physical representation of the victim, while the Controller classes are the brains, the internal functions.

Whereas the victim contains of assets and unrealscript, the simulated environment is created through the UDK editor. In this editor the environment is made just like an image can be created with editors such as Photoshop. One can easily add a terrain, trees, static objects or even actors such as victims. This environment, however, is mostly static, since one does not desire any behaviour from terrain, buildings or static objects. In order to determine the behaviour of dynamic objects, such as the animation of trees or the behaviour of objects, Kismet is used. Kismet (figure 4) uses *Events*, *Actions*, a few flow-controller statements such as if, switch and variables. Events are triggers, such as the start of a simulation, touching an object or the event of an object getting destroyed. Actions are that what are triggered, this can be anything from showing an animation, and moving an actor, to inserting an object in the environment. Additionally, actions can call upon other actions, which means that actions can optionally act as triggers for other actions. The editor is created through native language, it cannot be altered. The objects that make up the environment, however, can altered using the editor (assigning material and texture to a mesh for example) or through unrealscript (adding a new object, a victim, to the editor). Unrealscript is also capable of adding objects to Kismet, such as Events or Actions.

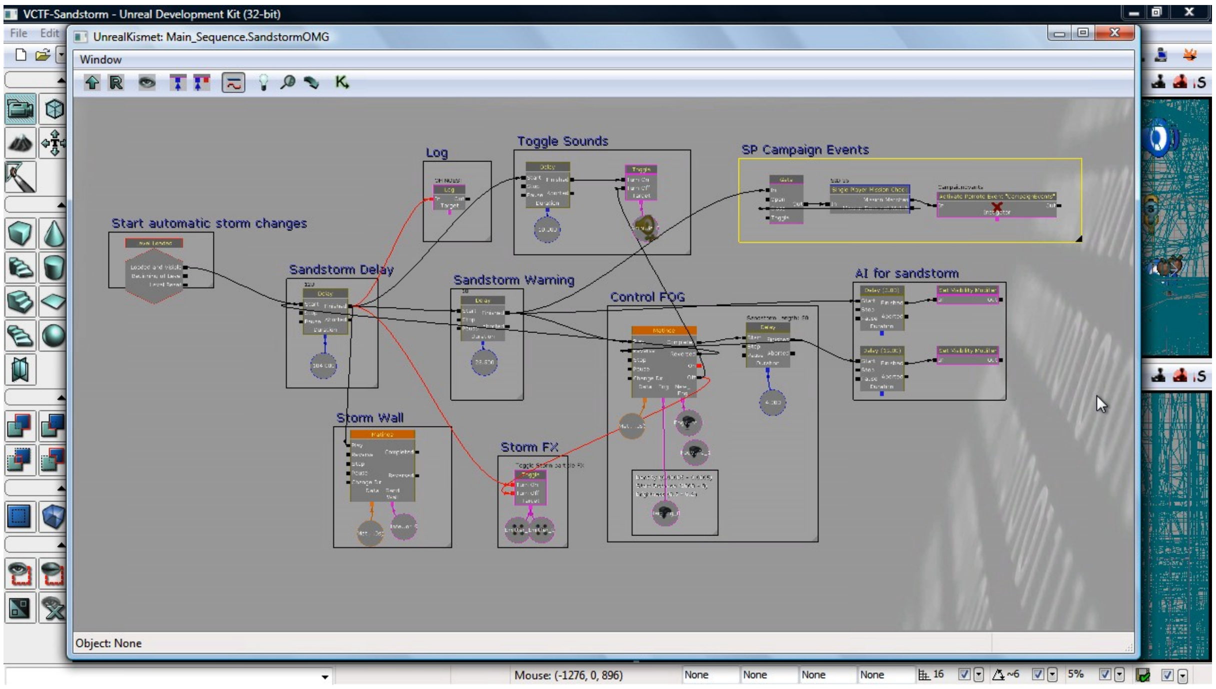
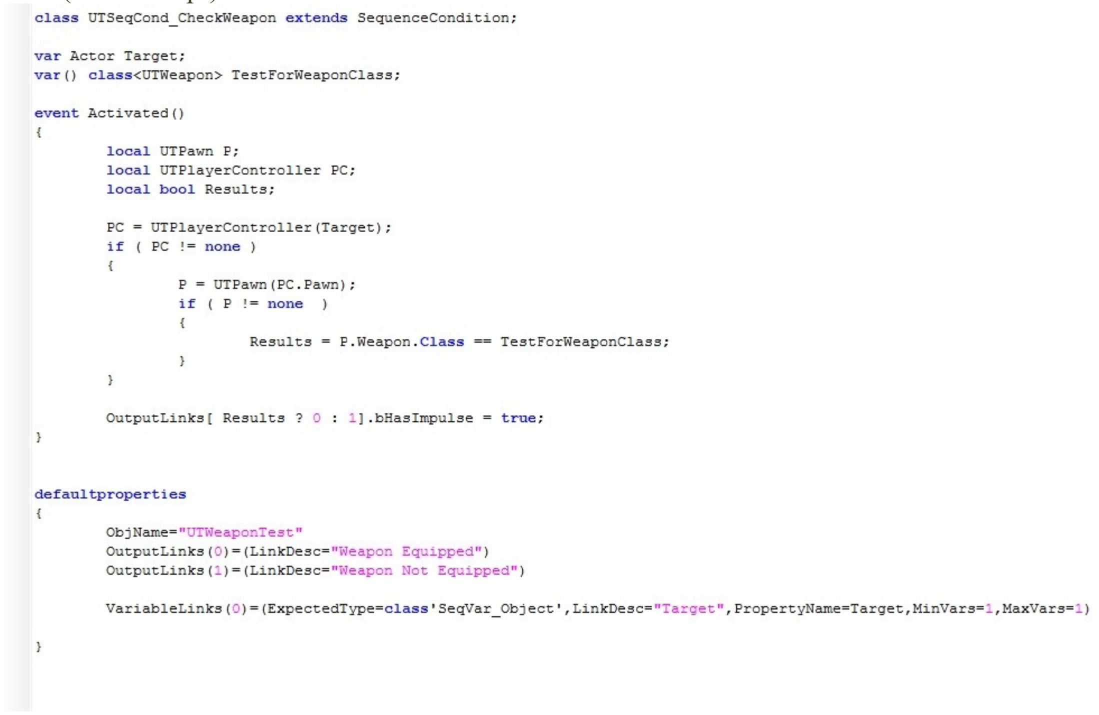


Figure 4. Kismet determining the behaviour of a storm

Although the procedures of Kismet is determined through native language, the objects within Kismet can be freely modified by unrealscript, Actions such as picking up items and announcements can be created through unrealscript (figure 5). These objects, however, are only capable of storing variables and trigger an unrealscript function. Because these unrealscript functions can then be created in classes (such as victim), Kismet operates as a communication link between the editor and the development code (unrealscript).


```
class UTSeqCond_CheckWeapon extends SequenceCondition;

var Actor Target;
var() class<UTWeapon> TestForWeaponClass;

event Activated()
{
        local UTPawn P;
        local UTPlayerController PC;
        local bool Results;

        PC = UTPlayerController(Target);
        if ( PC != none )
        {
                P = UTPawn(PC.Pawn);
                if ( P != none  )
                {
                        Results = P.Weapon.Class == TestForWeaponClass;
                }
        }

        OutputLinks[ Results ? 0 : 1].bHasImpulse = true;
}


defaultproperties
{
        ObjName="UTWeaponTest"
        OutputLinks(0)=(LinkDesc="Weapon Equipped")
        OutputLinks(1)=(LinkDesc="Weapon Not Equipped")

        VariableLinks(0)=(ExpectedType=class'SeqVar_Object',LinkDesc="Target",PropertyName=Target,MinVars=1,MaxVars=1)

}
```
Figure 5. Unrealscript example

For a more thorough explanation on how to program in unrealscript, in order to create Kismet functionalities, see appendix B. In the next a method is created in order to show to show the possibilities of behaviour in USARSim. This is done by creating a scenario in which victims show different behaviours.

Method

A scenario has been created in which two victims are in a building, or which one is hurt and the other is unhurt. The first victim will be unable to move without being treated by the other, because he is obstructed by a pillar.

A possible scenario would be that the unhurt victim starts looking for an exit, and after he has found one, he rushed back to his friend in order to help him reach the exit. Another could be that he first helps his friend and they search together.

UDK provides various methods for creating these scenario's, it is even possible to randomize actions sequences, leading to an unpredictable outcome. In the appendix an explanation for coding various victim behaviours in UDK is given.

Description scenario

The environment consists of a big building divided in two rooms by a wall with an opening. One room will represent an escape route, and the other room the post-disaster environment. The first room, for escaping to, is completely empty with the exception of *pathnodes* (See appendix for *pathnodes)*. The second room consists of a few chairs, tables, fallen pillars and rocks, it is a representation of a post-disaster environment. Additionally two victims reside in this room, of which one is laying face down with a pillar on to op, and the other is standing. These two victims represent a hurt person (with a pillar on top) (figure 6) and a 'fortunate' unscratched person. In stead of the regular situation, in which both victims would only show an animation and no other behaviour, movement or interaction with the environment, this time the victims are supposed to escape the room.

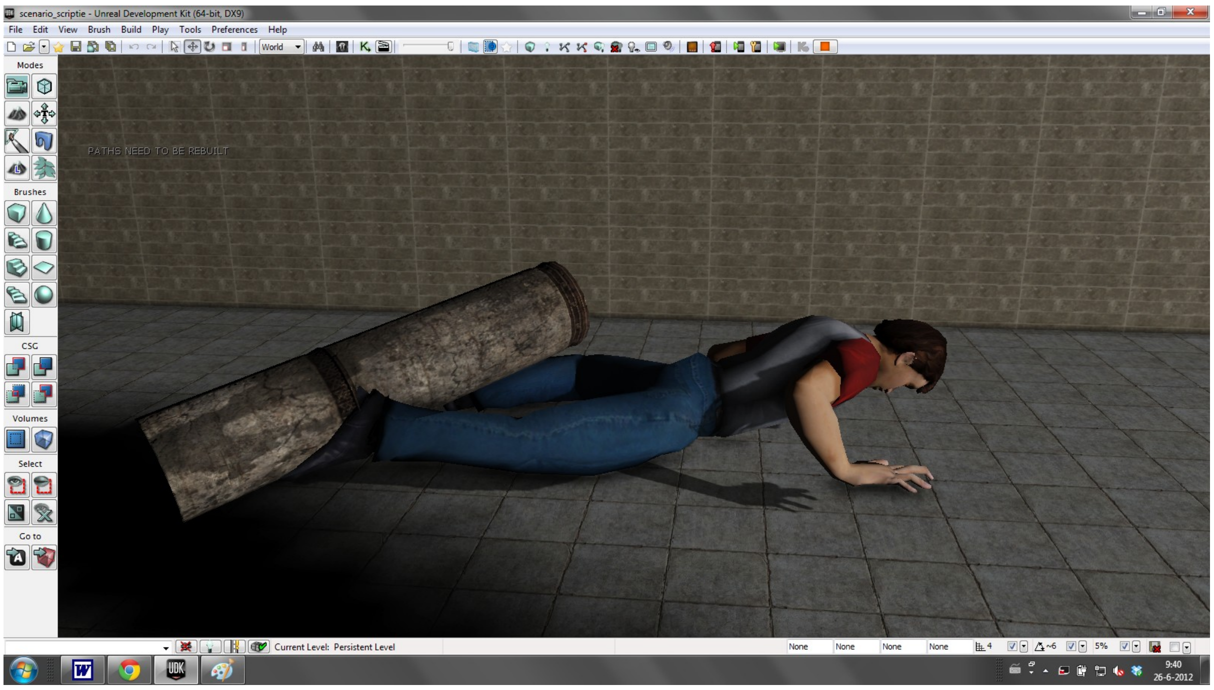

scenario_scriptie - Unreal Development Kit (64-bit, DX9)
File Edit View Brush Build Play Tools Preferences Help
Modes
Brushes
CSG
Volumes
Select
Go to
PATHS NEED TO BE REBUILT
Current Level: Persistent Level
9:40
26-6-2012

Figure 6. Pillar on victim

When the simulation has started, the standing victim will walk towards the lying victim and will start a crouching animation. While the victim is crouching, an animation in *matinee* will be played in which the pillar on top of the lying victim will be removed. After the pillar is remove, both victims stand up from their position and will walk towards the exit of the room.

Validation

The scenario just described has been chosen because it is a representation of the 1) most basic desirable human behaviour and because it is a good 2) example of a post-disaster situation: It is a representation of the most 1) basic human behaviour, because there are only two victims, which leads to the possibility of interaction between victims that is not too complex to understand. This basic behaviour are the fundamentals for more complex designs, more appropriate for simulating post-disaster situations.

This scenario is a good 2) example of post-disaster situation, because it has victims of which at least one is hurt and it has objects lying around due to, for example, an earthquake. Secondly the environment has been kept fairly simple, since it does not consist of any complex structures or paths as it consists of a single square room and some objects in it. The objects are placed in the room to display obstacle avoidance behaviour using paths, and in order to maintain a realistic simulation. Again, the scenario has been kept simple because its purpose was not to create a scenario as complex as possible, but to demonstrate the possibilities of creating more complex scenarios.

Introduction functionalities

As introduced in the software section, the UDK comes with an editor for creating environments, unrealscript for programming classes, and kismet in order to make interaction between the environment and the classes possible.

Moving around is a fundamental characteristic of humans, or any living being, and is necessarily for even the most basic behaviours. That is why adding movement to a victim can be seen as the first step towards providing possibilities for behaviour. If moving is possible, one could use the Events in kismet in order to create significantly different behaviour compared to what victims have been capable of up till now. The victim could be programmed to either follow or run from a rescue robot, whenever such robot comes in sight.

Animations are a second important aspect of behaviour in the UDK. Animations show what the victims are doing, and can vary from simply standing and breathing to making gestures. Therefore is the capability of controlling the animation sequence highly important, making the difference between endlessly lying on the ground and interaction with the world.
The third functionality described in this thesis will be making sound. Whereas in a real disaster information gained from sound is imminent (yelling for help for example), the victims in USARSim have not shown any capabilities of producing sounds. Producing sounds actually have nothing to do with the victim classes and it can be achieved without the use of any unrealscript, it is already available through Kismet in the editor. For a step by step explanation of implementing movement, animations and sound see appendix B.

Results

The result shows that sequences of animations are controllable in combination with movement. In the simulation the victims were capable of showing various behaviours: At the start of the simulation the standing victim walks towards the hurt victim, and crouches for 5 seconds. In that period, an animation is played in which the pillar is moved from the victim (figure 7). Afterwards, both of the victims stand up and walk toward the exit.

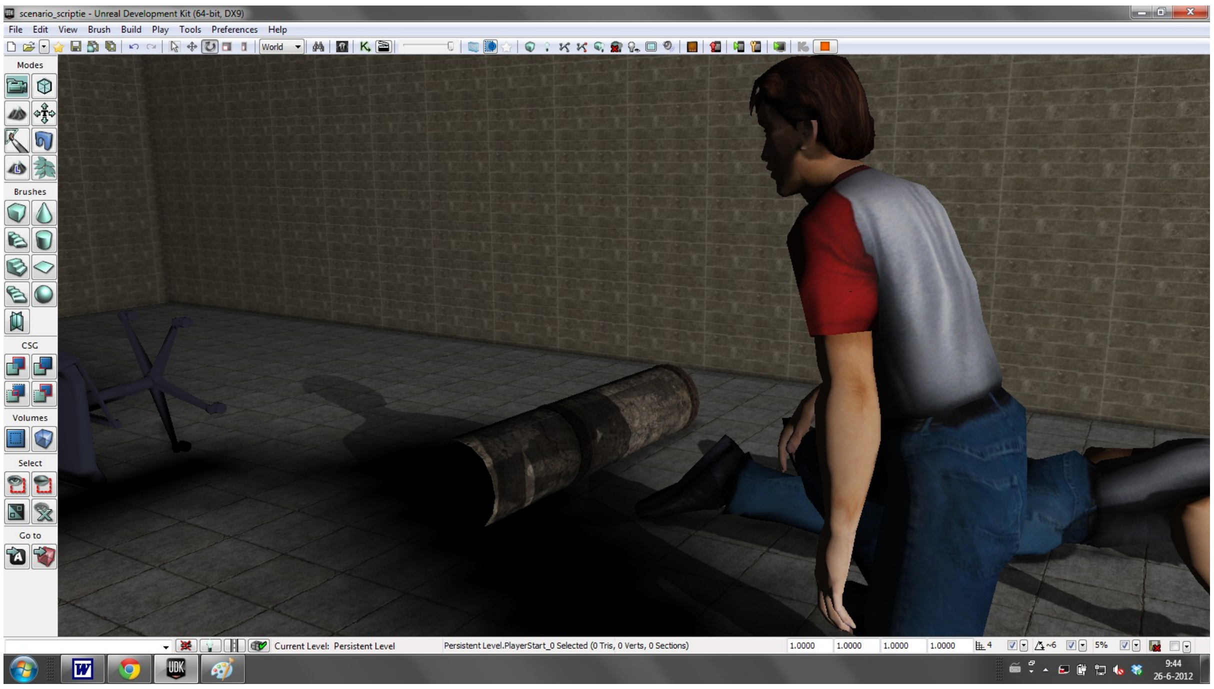

scenario_scriptie - Unreal Development Kit (64-bit, DX9)
File Edit View Brush Build Play Tools Preferences Help
Modes
Brushes
CSG
Volumes
Select
Go to
Current Level: Persistent Level
Persistent Level.PlayerStart_0 Selected (0 Tris, 0 Verts, 0 Sections)
9:44
26-6-2012

Figure 7. Victim helping other by removing pillar

Conclusion & Discussion

Conclusion

This thesis attempted to extend the possibilities of victim in USARSim used in the Roborescue cup. Compared to the previous behaviour of the victims, improvement has been made by adding movement and animation control capabilities. It has been shown through a scenario that basic human behaviour can be created using UDK and especially Kismet.

Problems

However, a few problems have been encountered in the thesis. Firstly, whenever the environment is created in such a way that a path is obstructed by two blocking objects, the victim tends to get stuck and quit halfway. This shows that the victim does not actually calculate a path before moving. Providing a different route by (re)placing *pathnodes* strategically or control its movement through Kismet can solve this.

Secondly, although blending of animations is done fairly realistic by the engine itself, some animation sequences may display surprising blending positions. Blending from a lying position into a standing position, for example, creates a flying victim halfway through the transition.

Additionally, it should be marked that even though this thesis aimed to provide victims behaviour, it does not claim to have created any Artificial Intelligence. However, in contrast to victim simulations so far, it attempts to show the capabilities simulating behaviour using USARSim.

Future research

Future research is possible in multiple directions:

1) Firstly extending the behaviour towards creating intelligent behaviour. This should be done in the controller in stead of the methods used in this thesis (Kismet mostly). Kismet is used for pre-programmed scenarios and events. On the other hand, the *VictimController* is more suitable for general (intelligent) behaviour. As briefly mentioned in the software section, states in the controller could be used for more sophisticated behaviour, such as programming curiosity whenever the victim is in an idle state: It would try and explore its surroundings, or be attracted to various objects around him for example.

The first steps toward achieving this would be creating a path-planning structure in the *VictimController*. As previously states, victims do not plan their paths when moving towards a particular goal, they simply start moving towards it and as a result end up stuck whenever the path is blocked (figure 8).

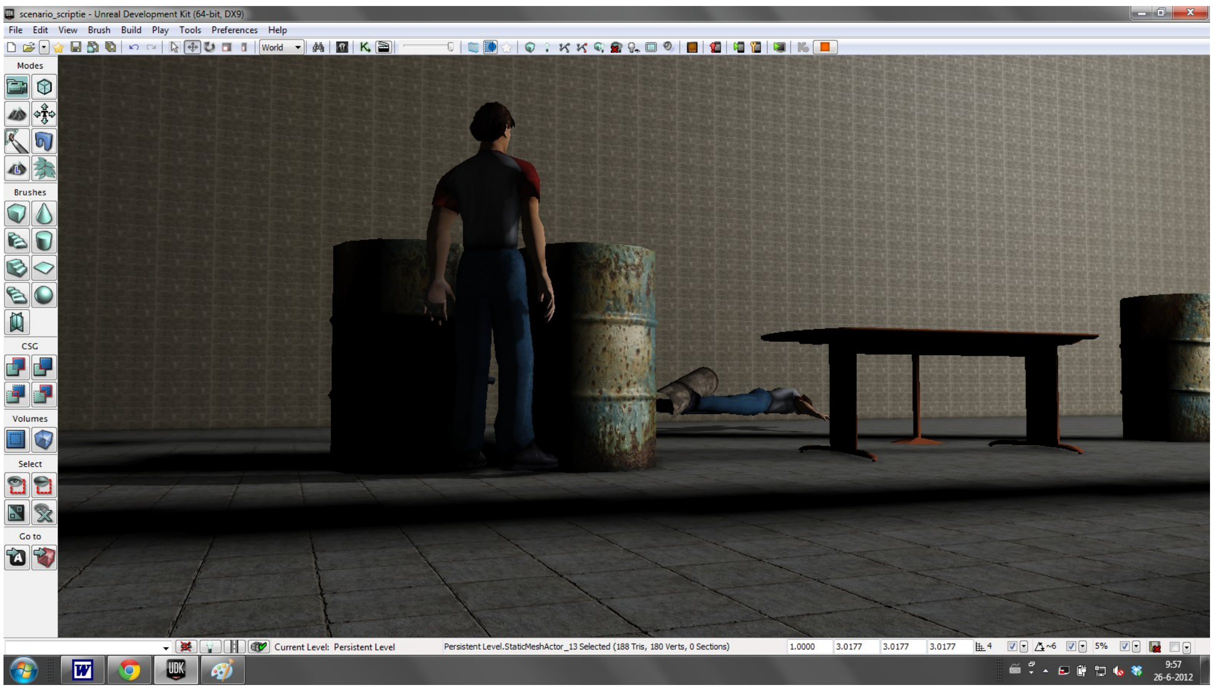

scenario_scriptie - Unreal Development Kit (64-bit, DX9)
File Edit View Brush Build Play Tools Preferences Help
Modes
Brushes
CSG
Volumes
Select
Go to
Current Level: Persistent Level
Persistent Level.StaticMeshActor_13 Selected (188 Tris, 180 Verts, 0 Sections)
9:57
26-6-2012

Figure 8. Two barrels successfully block a victim, walking towards the lying victim, even though plenty of space is available for actually reaching it.

2) Secondly, future research should be conducted towards heat emission of victims. Heat detection could prove to be an important factor of human recognition, and should be included in the victims as such. When heat emission is added to the victims, the teams in Roborescue cup are motivated to create heat-emission-detectors, which may lead to significant progress towards human detection.

3) Thirdly, a deeper understanding of animation blending in the UDK is required for a more realistic representation of humans. The UDK offers numerous methods for blending animations, from instructing a single bone to adding face animations. In this paper, an introduction is made towards

controlling the animations of a victim, investigating these possibilities will lead to a more realistic victim.
4) Lastly natural language processing is a possible direction towards creating communication between the victims and rescue robots. This paper is an introduction towards reaction between environment and victim, future research should determine whether the Unreal Engine is capable of creating and processing natural language, overseeing the possibilities of providing and processing information of victims.

References

B. Balaguer, S. Balakirsky, S. Carpin, M. Lewis, and C. Scrapper. USARSim: a validated simulator for research in robotics and automation. In Workshop on Robot Simulators: Available Software, Scientific Applications, and Future Trends at IEEE/RSJ, 2008.

C. Bastiaan, Virtual victims in USARSim, Bachelor Thesis, Universiteit van Amsterdam, June 2010.

M. Brenner, N. Wijermans, T. Nussle, and B. De Boer. Simulating and controlling civilian crowds in robocup rescue. Proceedings of RoboCup, 2005.

J. Brownbridge. Teleoperation of Rescue Robots in Urban Search and Rescue Tasks. 2008.

H. Flynn. Machine learning applied to object recognition in robot search and rescue systems. Master's thesis, University of Oxford, 2009.

C.E. Fritz and H.B. Williams. The human being in disasters: A research perspective. The Annals of the American Academy of Political and Social Science, 309(1):42, 1957.

B. Gates. A robot in every home. Scientific American, 296(1):58–65, 2007.

D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. Nature, 407(6803):487–490, 2000.

L. Matthies, E. Gat, R. Harrison, B. Wilcox, R. Volpe and T. Litwin. Mars microrover navigation: Performance evaluation and enchanement. Autonomous Robots, 1995-12-01. Volume 2 Issue 4, p. 291-311.

Q. Nguyen and A. Visser. "A Color Based Rangefinder for an Omnidirectional Camera", Workshop on Robots, Games, and Research: Success stories in USARSim (IROS 2009), p. 41-48, St. Louis, USA, October 2009.

S. van Noort, "Validation of the dynamics of a humanoid robot in USARSim", Master's thesis, Universiteit van Amsterdam (May 2012)

E.L. Quarantelli. Images of withdrawal behavior in disasters: Some basic misconceptions. Social Problems, 8(1):68–79, 1960.

W. Shao, D. Terzopoulos. Autonomous pedestrians, Graphical Models, Volume 69, Issues 5–6, September–November 2007, Pages 246-274, ISSN 1524-0703, 10.1016/j.gmod.2007.09.001.

A. Visser, B. Slamet, T. Schmits, L.A.G. Jaime, and A. Ethembabaoglu. Design decisions of the UvA Rescue 2007 Team. In Fourth International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster (SRMED 2007), pages 20–26.

Appendix

A - Victim package

A victim consists of a 1) skeletal mesh, 2) texture and material, 3) physics, and an 4) a) animation set and b) tree. The 1) skeletal mesh is the silhouette of the victim; it is in fact the whole physical form of a victim, without any colour or texture on the victim. The 2) texture and material are the layers around the skeletal mesh, granting the victim colour (human skin, cloths) and surface (hair has got a different look and feel than clothing). The 3) physics asset is the skeletal as we know it from biology; this asset provides the UDK knowledge of the joints and bones of the skeletal mesh. Lastly the 4) a) animation set is a set of possible animations, pre-programmed motions of the skeletal mesh in accordance with the physics asset, and the b) animation tree decides what kind of animation should be used, depending on the situation. For more information on how to create a package, see Chaim's Virtual victims in USARSim.

B - Programming with UDK

Animations

There are two kinds of methods available to insert animations into an environment. The first method is *matinee*, which is a function in Kismet. It works very much like a video creator, by creating key frames one can place objects on different positions on each frame, and the UDK creates the frames in between. This however, is only useful with static meshes, like an elevator going up and down, or the animation of grass. For pawns (victim is a specific pawn), the second method is used: animation tree. An animation tree (figure 9) is a tree diagram consisting of mainly animation sequences and blending nodes, together with a few nodes with special functions. An animation sequence node is a specific animation out of the animation set provided in the package, such as crouching or walking forward. Special nodes each have their own purpose and should be investigated separately. There are various types of blending nodes, but their main purpose is either combining the tree's branches into one animation or forming a threshold for deciding what branch in the tree to take.

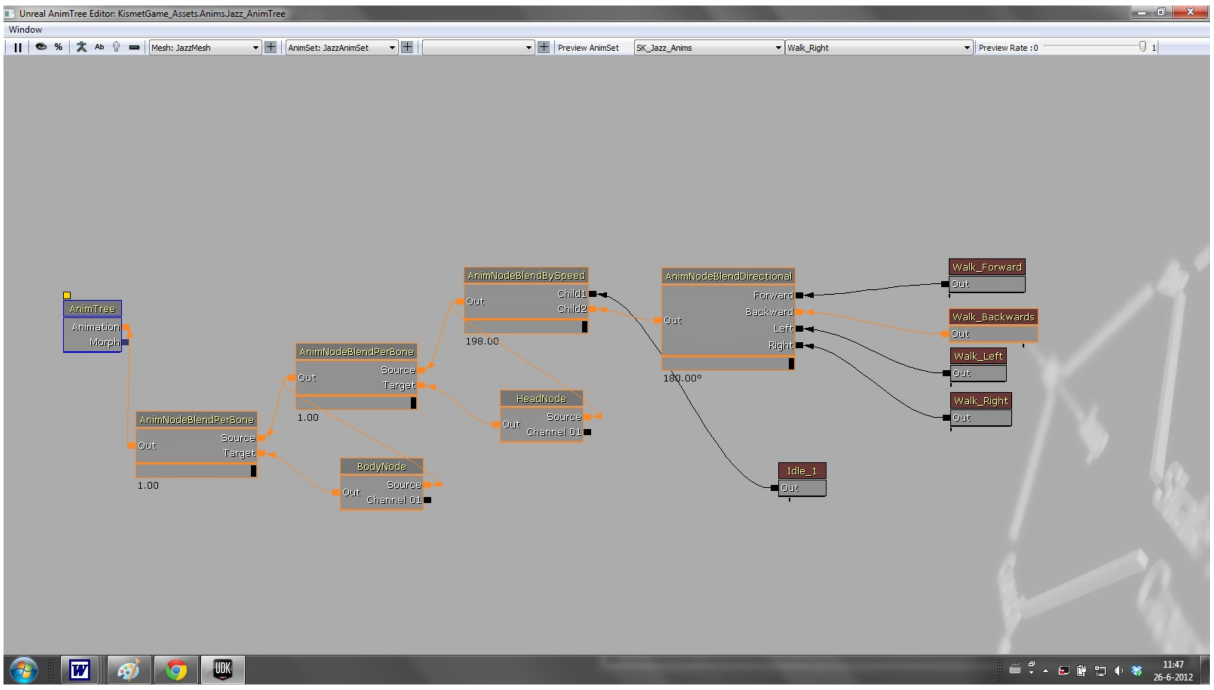


Figure 9. Animation tree example

Blending of animations consists of creating a combination of multiple animations and mixing them together to form one movement. Blending is applied for various purposes, such as blending face animations with body movement; Instead of creating numerous walking animations with different facial expressions, each facial expression is animated and blend with the movement desired. Additionally, one could create an animation of throwing a snowball and blend it with a walking animation, in order to create an animation used for throwing a snowball while moving.

The Unreal Engine calculates the angle of every joint and bone angle per active animation, and blends it together to form one movement, which is roughly the average of each animation. It is possible to tell the Engine at the ratio of the blending animations, granting the developer high precision tools for animating their desired movement. Two blending nodes will be described for a better understanding: The *AnimNodeBlendDirectional* and the *UDKAnimBlendByIdle*.

The *AnimNodeBlendDirectional* blends its branches into one animation, depending on the direction the pawn (victim in our case) is moving, it has 1 branch for each direction (forward, backward, left and right). When the pawn is moving 0 degree's, straight forward, it will prioritize the branch forward, when the pawn is moving diagonal, however, the forward and left or right branch are combined with different rates depending on the actual direction: a direction 10 degrees would result in a different blend ratio than that of 45 degrees direction. In the last case the two branches are equally strongly blended.

Whereas some blending nodes combine their branches into a single animation, other nodes differentiate between their branches by forming a threshold: One branch is active, the others are not. *UDKAnimBlendByIdle*, for instance, either actives the Idle branch (the pawn is not moving), or actives the Moving branch (the pawn is moving). It does not combine these branches depending on how fast a pawn is moving or not.

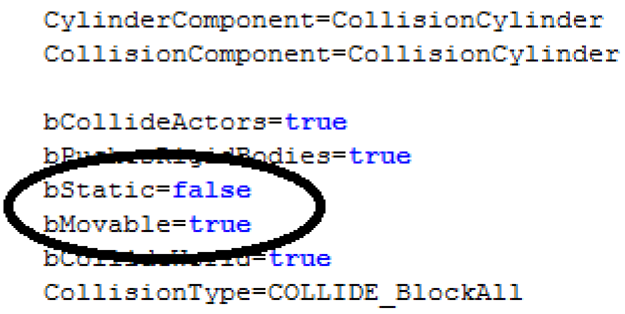
Whereas *UDKAnimBlendByIdle* is used to determine whether a character is moving, *AnimNodeBlendDirectional* is mostly used for walking, running, flying or any other kind of moving animations. An animation tree fit for a character that either moves or stand still, would contain the *UDKAnimBlendByIdle* at the start of the train, with its branches an idle animation (at the Idle branch) and the *AnimNodeBlendDirectional* (at the Moving branch). With an animation tree as described above, it is possible to create a moving victim in USARSim by adding just a few settings.

Moving victims

UDK already has got features for moving pawns around the environment: The node called *Move to Actor* (Action => AI => Move to Actor) is especially made for these kinds of actions. In order to move a pawn from a certain point towards an actor, an event must be created and the event must be connected to a node. After that the pawn in the environment must be selected and assigned it Target in Kismet. The same must be done for destination and Look At. An option is to tell UDK what to do after the move is finished by connecting Finished with the next node.

The UDK is capable of finding its own paths through the environment. Sometimes however, the editor may require some help. Whenever a pawn is not capable of find a path on its own, it is required to add *pathnodes*. *Pathnode* is a class derived from Actor, and whenever the editor is told to build paths, it will look at all *pathnodes* and make paths between them. After setting this up any normal pawn in the UDK will be able to move from and to these nodes.

However, for victims in USARSim two settings must be altered before this will work. These two settings reside in the *Victimpawn* class: modify or add “bStatic=false” and “bMovable=true” in the default properties (figure 10), without these properties, the victim will not move.



```
CylinderComponent=CollisionCylinder
CollisionComponent=CollisionCylinder

bCollideActors=true
bPushesRigidBodies=true
bStatic=false
bMovable=true
bCollideWorld=true
CollisionType=COLLIDE_BlockAll
```

Figure 10. Settings in Victimpawn.uc necessarily for creating a moving victim

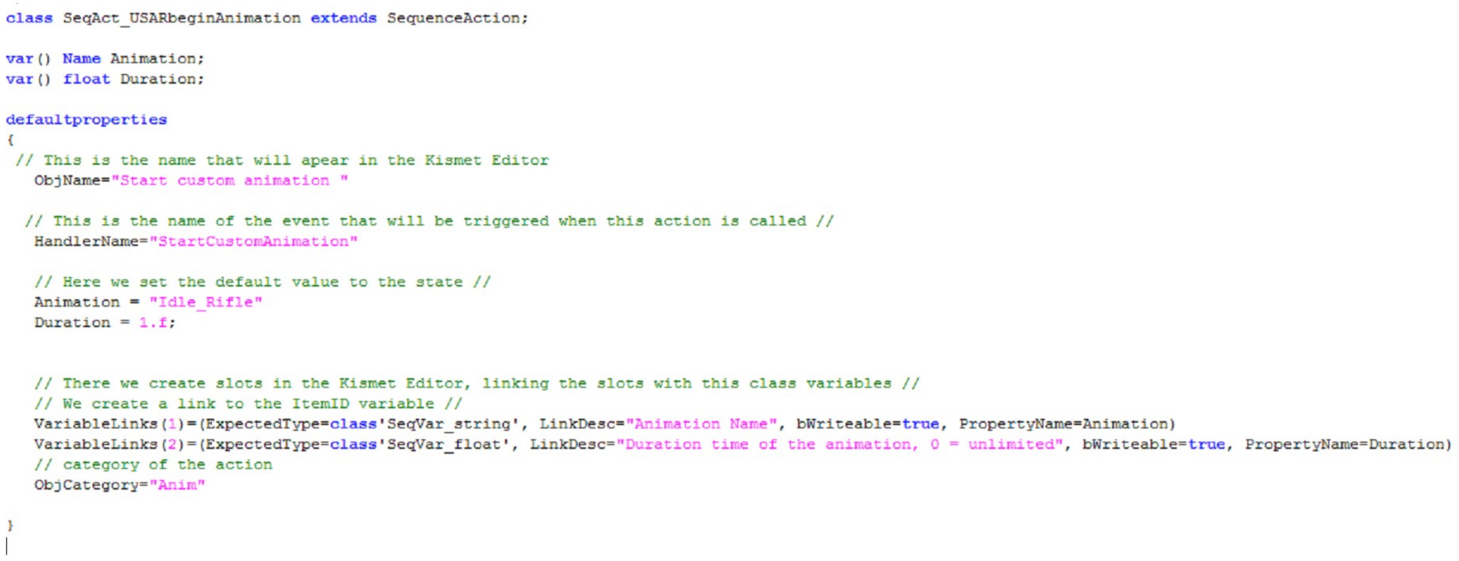
If the animation tree is configured correctly, the victim will now respond to the Kismet commands, which implies that victims are capable of moving around the environment. As mentioned before, it would be desirable to not only be able to move a victim, but also to be able to tell UDK what animation to perform. In order to achieve this, a deeper look in unrealscript is required.

In the next section the communication between kismet and unrealscript will be explained. If developers have the possibility of instructing victims explicit to show an animation, further development of victim-behaviour is possible.

Calling custom animations

The animation tree provides numerous ways of controlling your pawns animations, nevertheless sometimes a developer might want to simply control what animation a certain pawn must perform. In the case of victims, one might want several pawns to either move around or stand idle, while one pawn is lying on the ground. An animation tree is, unless different victim classes are made, it is impossible to have different animations for victims in the same state. Telling a particular victim to perform a custom animation is typically done using Kismet. In order to do is, one has to create a new Kismet Action, "*play animation*" for example.

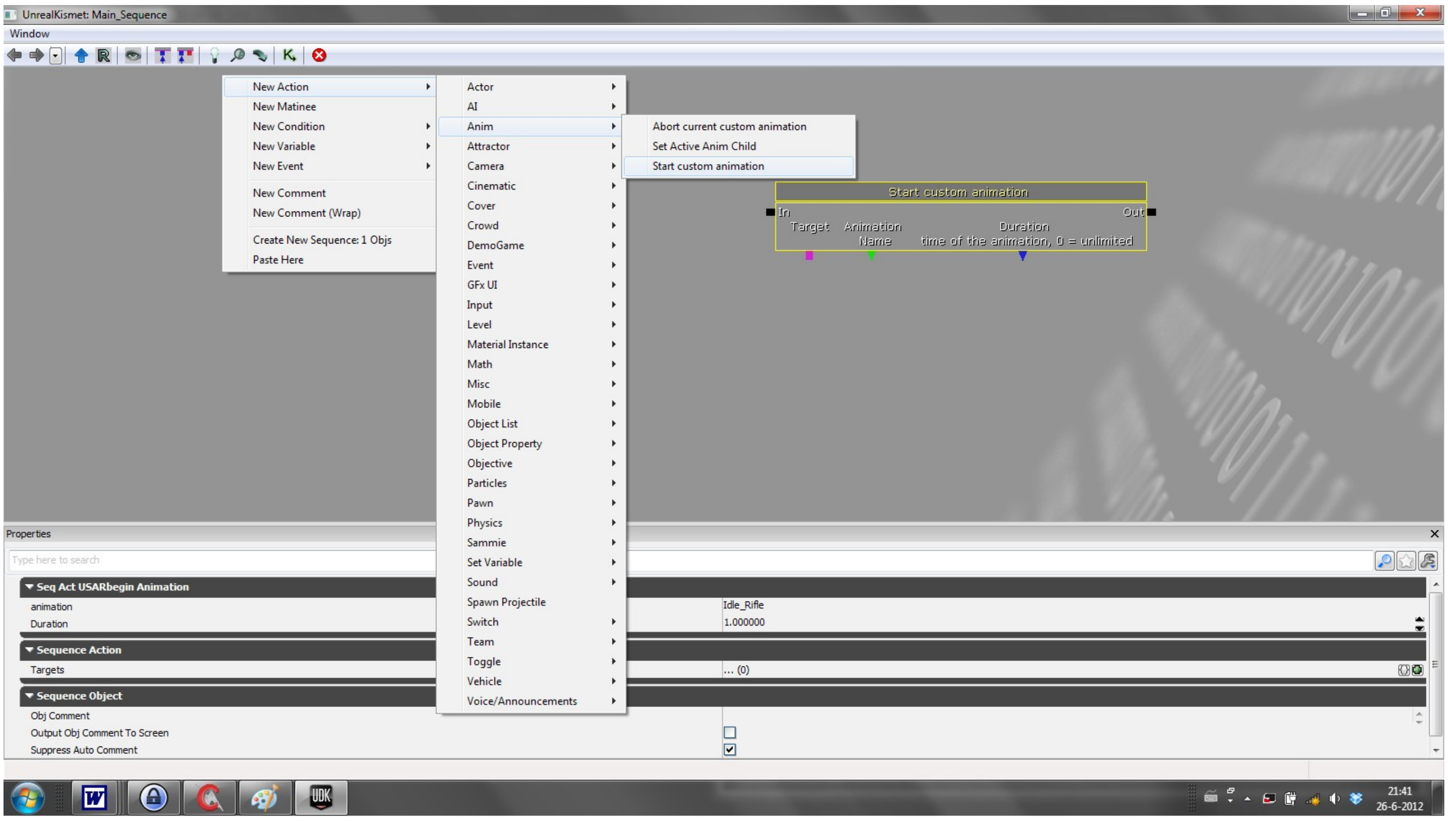
Making a Kismet function is done by creating a new file, make it extend the *SequenceAction* class and add your own properties and variables (figure 11). In order to create a function that will call an Animation, firstly two variables are necessarily: The name of the animation, and the duration you want the animation to be played, these variables should be initiated.



```
class SeqAct_USARbeginAnimation extends SequenceAction;

var() Name Animation;
var() float Duration;

defaultproperties
{
 // This is the name that will apear in the Kismet Editor
   ObjName="Start custom animation "

  // This is the name of the event that will be triggered when this action is called //
   HandlerName="StartCustomAnimation"

   // Here we set the default value to the state //
   Animation = "Idle_Rifle"
   Duration = 1.f;


   // There we create slots in the Kismet Editor, linking the slots with this class variables //
   // We create a link to the ItemID variable //
   VariableLinks(1)=(ExpectedType=class'SeqVar_string', LinkDesc="Animation Name", bWriteable=true, PropertyName=Animation)
   VariableLinks(2)=(ExpectedType=class'SeqVar_float', LinkDesc="Duration time of the animation, 0 = unlimited", bWriteable=true, PropertyName=Duration)
   // category of the action
   ObjCategory="Anim"

}
```

Figure 11. An example of a kismet action code

Secondly Kismet requires the *VariableLinks* and the name and category of the action. The category means what kind of action it is; in this case the category is "Anim" for the purpose of structure within Kismet. The *VariableLinks* are the variables that are adjustable through kismet, the name and duration of the animation. Lastly it is a *handlername* should be set, any function with the same name as the *handlername* will be called in the target, the victim pawn. The Kismet Action is available through Kismet=>Action=>Category=>Name (figure 12), however, it will not perform any action. The victim will not react to this Action until it is programmed to do so.



Seq Act USARbegin Animation	
animation	Idle_Rifle
Duration	1.000000
Sequence Action	
Targets	... (0)
Sequence Object	
Obj Comment	
Output Obj Comment To Screen	☐
Suppress Auto Comment	☑

Figure 12. A custom kismet action

Before adding this functionality to the victim, creating the possibility to actually make the victim perform an animation other than the one defined in its animation tree is required. In order to do so, a node called *AnimNodeSlot* must be added at the base of tree (figure 13). With this node it is either possible to blend the animation in the three with another undefined source, or take over the animation with another source. Controlling whether it will overtake the animation tree or it will blend with the animation tree is decided in the internal settings.

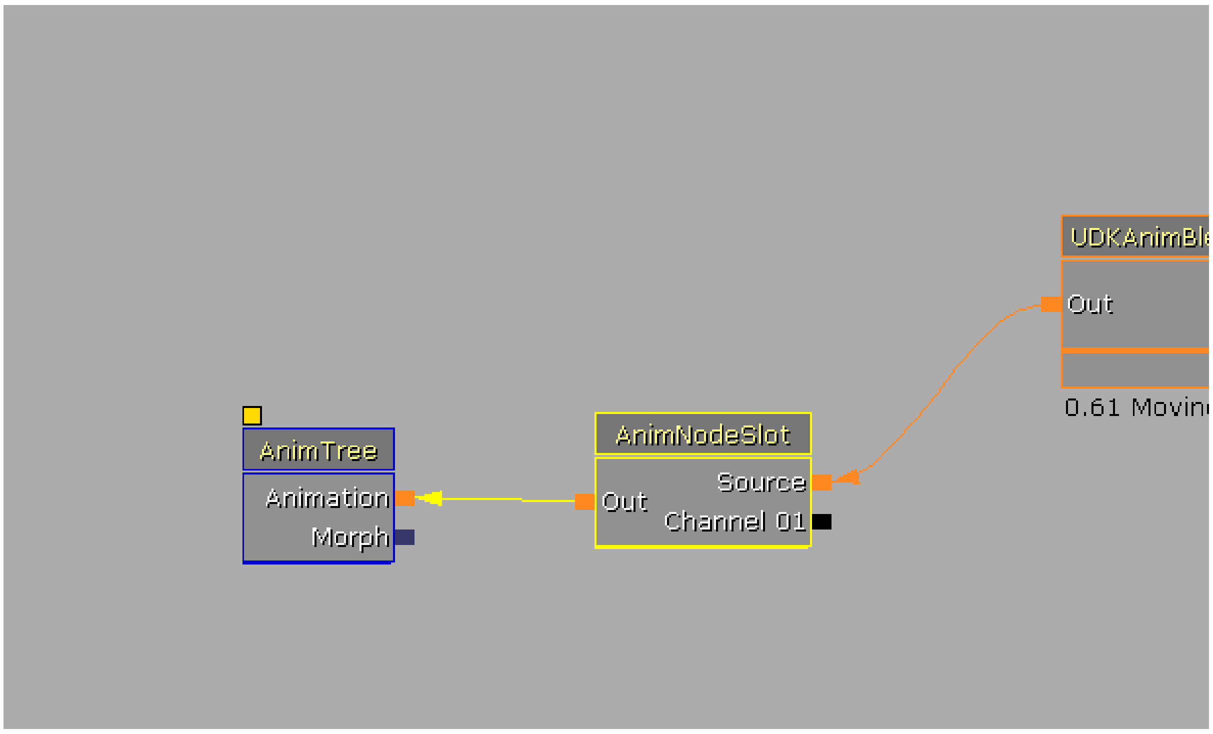
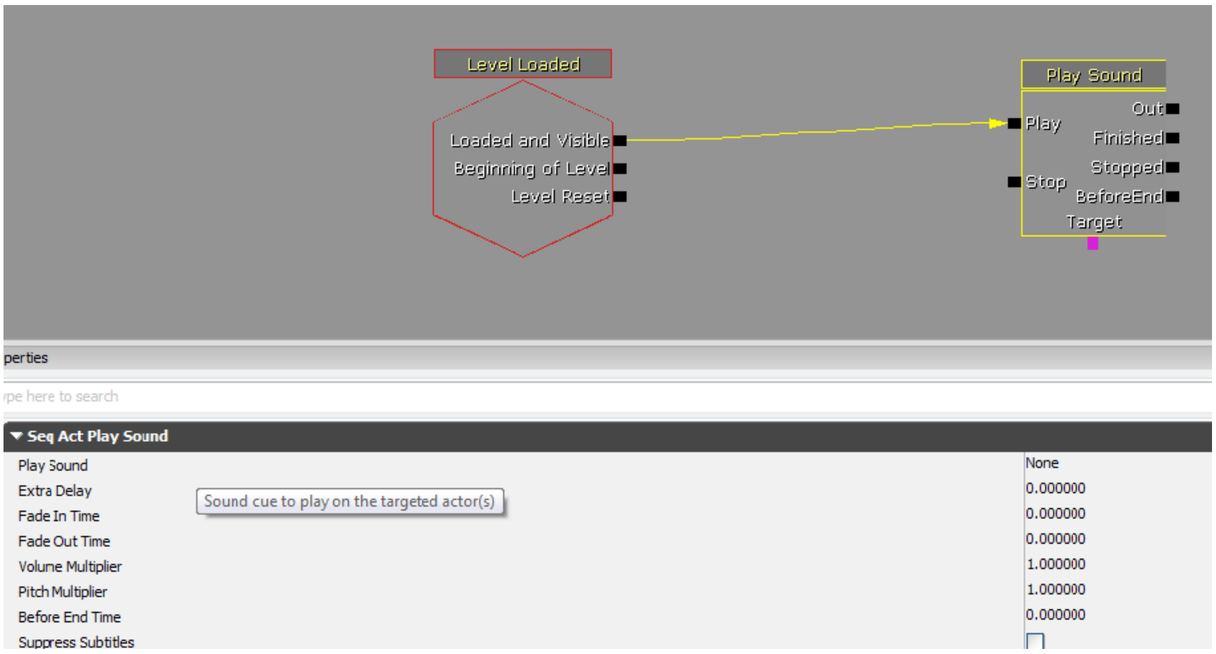


Figure 13. Animnodeslot node in animation tree.

Creating an animation start function to the victim is done by editing the *VictimPawn* class by adding a function with the same name as the *handlername* of the Kismet object. In order to play an animation, the *PlayCustomAnimByDuration* function of the class *AnimNodeSlot*, is used. The variables determined in the Kismet object can be called, and by passing the correct parameters to the pre-programmed function, an animation will be played, successfully overtaking the original animation tree.

Sound

Playing a sound can be done by connecting an event with the animation node called play sound in the category action =>sound. In order to do so, an event (for example: event => Level Loaded) and the play sound (action => play sound) node should be created, these two nodes should be connected from Loaded and Visible (Level Loaded node) to Play (Play sound node), then the only thing left is telling Kismet which sound to play (figure 14), by filling in the sound name given by the content browser. There are multiple options available to further enhance your action. Kismet can be instructed what to do while or after this sound is played.



perties

ype here to search

Seq Act Play Sound

Sound cue to play on the targeted actor(s)

Play Sound	None
Extra Delay	0.000000
Fade In Time	0.000000
Fade Out Time	0.000000
Volume Multiplier	1.000000
Pitch Multiplier	1.000000
Before End Time	0.000000
Suppress Subtitles	☐

Figure 14. How to start a sound