Virtual victims in USARSim

Author: Chaim BASTIAAN (#5742889)

UNIVERSITEIT VAN AMSTERDAM
FACULTY OF SCIENCE (FNWI)

POSTBUS 94216
1090 GE AMSTERDAM
SCIENCE PARK 904

BSC THESIS, 9EC

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BSC

Virtual victims in USARSim

Author:
Chaim BASTIAAN
Student #5742889

Supervisor:
Dr. Arnoud VISSER

Abstract

USARSim is used to simulate search and rescue tasks in the RoboCupRescue. A necessary asset when it comes to most research related to rescuing victims are the victims themselves. As general expectations rise and (RoboCupRescue) leagues become more demanding, the environment has to keep up with the possibilities and accuracy of the simulated (robot) observers. This thesis presents a design on victim behaviour, as well as an analysis of the resources and demonstrations. In addition to this, a description is given of steps such as importing models and textures into USARSim (compatible with UDK).

June 25, 2010

Contents

Acknowledgement

I'd like to thank my parents, my supervisor Arnoud Visser and Christine, for their continuous support throughout the whole project. Last but not least, a big thanks and full credits to Tom Lin, for the UDN models and a figure related to their tongue structure.

1 Introduction

The *Unified System for Automation and Robot Simulation* or *USARSim* is a high-fidelity simulation of robots and environments [5]. Due to a recent switch of USARSim from the Unreal Tournament 2004 based game engine (Unreal Engine 2.5) to Unreal Tournament 3 (Unreal Engine 3, or short UE3), the USARSim data has to be ported. The newer game engine not only brings performance improvements, it also adds new functionality such as dynamic shadows. It also leads to a setback in functionality for USARSim, where victims up to now are inable to move or perform gestures.

The RoboCupRescue project is an international joint effort in promoting research and development in rescue environments. It uses USARSim to simulate post-disaster environments in which robots are deployed to assist in rescuing victims. A necessary asset when it comes to most research related to rescuing victims are the victims themselves. Therefore, to allow for future developments such as new detection algorithms and sensors, additional realism or functionality in terms of victim behaviour and appearance inside USARSim is desired.

There is surprisingly little accessible information on (simulation of) post-disaster victim behaviour. This is in part due to the fact that the panic situation[11] (disaster scenario) itself or the mental state of a victim is generally of greater general interest. Examples of this type of research range from larger scale crowd simulation[3] to more individual behaviour of civilians or communities[10][14]. Another reason is that at the time robots are deployed, most victims are well hidden and barely relocate. Nevertheless there is a great desire for virtual victims that correspond to real ones in terms of appearance and behaviour. A decent simulation will allow future research in multiple areas, as well as serve as a base for more complex behaviour and more dynamic (changes of) appearance.

It is useful not just to look at past research on behaviour, but also how existing detection algorithms work. This should help determine what features make a victim recognizable as one by such an algorithm. Flynn [8] has recently shown that algorithms within USARSim continue to work in a real world application, though the results are less impressive than in the simulated environment. The used models were unable to provide hand gestures (even though the structure of the model is capable of showing such animations), obstructing research.

Even with such a simple model however, USARSim has been able to provide a validated tool for simulating robot environments [1]. As the general expectations rise and (RoboCupRescue) leagues become more demanding [2], the environment has to keep up with the possibilities and accuracy of the simulated observers. As the engine has recently been exchanged for another, victims are again needed more then ever. A more sophisticated, dynamic behaviour pattern should suffice and serve as foundation for future research.

This thesis presents a design of victim behaviour in USARSim. It can be roughly divided into three parts. Section 2 describes earlier research and simulations of disaster or urban search and rescue scenarios and related work. Section 3 presents the main design (process), as well as an analysis of the resources used, in order to investigate the boundaries of the design. Finally, in section 4 possible improvements and (logical) extensions to the design are discussed.

Since UE3 is also used as a commercial game engine it can be difficult to find specific information on certain aspects. In appendix A a thorough description of specific steps in the design process is given. Most, if not all steps in the appendix hold for the more recent, UE3 engine based Unreal Development Kit[1] (UDK). UDK is free for non-commercial and educational use, but at the time of writing this thesis incompatible with USARSim.

2 Theoretical context

What makes the topic of robot search and rescue so difficult is in part the fact that it is stretched across so many research areas. In order to create a decent design of the behaviour and appearance of victims in the simulation it is therefore useful to look at what has been done in for example psychology and human-robot interaction (HRI). Where psychology only in very specific branches touches upon the subject of victims in a post-disaster situation where robots could be deployed, interaction with the victim and detection (and therefore appearance) is of great interest to HRI. The first is usually concerned with the disaster as it strikes (a panic situation) and the post-disaster (traumatic) impact, where the latter is by definition in a situation where robots are deployed in order to rescue victims or assist to do so.

As psychology continues to evolve and propose new, more refined descriptions of human behaviour, we might be tempted to merely show interest in the most current or in our eyes appropriate description. From an AI perspective however, if a simpler model of behaviour is sufficient within the domain being used, it is generally less interesting whether it is an accurate model of the actual world or not. This is especially true when one of the main goals is to allow future research, as is the case with this thesis. It is of course still of great interest and importance to make the behaviour as realistic as resources allow.

2.1 Victim analysis

2.1.1 Behaviour in post-disaster (rescue) situation

There are plenty of field studies from a psychological perspective like those carried out by Fritz and Marks[9], where an extensive description is given of human behaviour during disaster. There are even more studies like those of Fritz and Williams[10], where a more refined research is performed on the results of field

[1]Unreal Development Kit by *Epic Games*, URL: http://www.udk.com/

studies, before, during or after (such as post-traumatic research) a disaster situation. The disaster situations of robot rescue and this literature tend to differ, for example victims are far less likely to relocate in robot rescue situations, rendering most research less applicable to the robot rescue topic.

Perhaps the most promising field studies, those of Human-Robot Interaction (HRI), supply us with detailed information on the behaviour and appearance (especially as viewed on video footage) of victims in a situation where robots are deployed. A big question in search and rescue is how to recognize victims based on cues, which are features that distinguish them from their environment. Examples of victim cues would be those presented by Murphy[12]:

> "[...] heat of a victim (or ignition source), color (that a live victim has moved, thereby knocking off some of the gray dust typically covering a void), motion, sound, and non-random areas and textures (gestalt or perceptual organization)." [12, p. 9]

Heat of a victim is the only thing left unmentioned throughout this thesis, but is briefly mentioned in section 4. Instead of cues, the term features is often used to refer to these characteristic victim cues. The above article also reminds us that the environment is generally filled with debris, instead of a clean office.

Human emotions have been investigated for a long time[6] and are of continuous interest to many areas of research. Directions on a possible way to implement emotion are presented in section 4, as well as suggestions for future areas of research related to this topic in section.

2.1.2 Victim detection

There are several approaches to the detection of victims and it is unclear which will be more error prone in the future, if any. Examples of approaches would be based on skin [15] or object detection [8]. These use respectively the colour and shape or pattern as features (cues) to distinguish victims from their surroundings. The two mentioned approaches both classify as victim and non-victim, but detect different things. To enable classifiers to work reliably on sensor data, the shapes, textures, movement and lightning (or lack of it) has to be sufficiently realistic.

Section 3 describes the design of adding multiple of these features, where section 4 gives an overview of what this means for topics like victim detection.

2.1.3 Victim simulation

As with research in the specific situation where robots are deployed, simulations of victims are hard to come by. Simulations are generally on a larger scale than USARSim generally uses, using a simplified model of the environment as proposed in literature to simulate for example (victim) crowds[3] and escape panic[11]. A larger scale, because USARSim is generally used for a few sites focusing on details, whereas these studies consider whole cities focusing on more general behaviour patterns and dynamical features. Some specific research may instead use a simpler model of victims, so as not to scare participants (ie. by replacing them with pool balls[4]). Another common reason for simple victims is that it is usually sufficient, such as with static victims or limbs (simply partial models) in USARSim research[16].

2.1.4 Related work

Since the research areas related to a virtual world such as USARSim are quite diverse, this section will primarily focus on my area of expertise, Arificial Intelligence (AI).

An important feature of USARSim is that it allows sensors, robots and algorithms to be created and tested virtually. This allows for cheap development and allows creating for example algorithms and sensors that do not exist yet in the real world. A good recent example of this is an acoustic sensor [13], entirely created and tested virtually (sounds emitters instead of victims). Developing algorithms and robots (rapid prototyping without driving robots off a cliff for real) is a hot topic. Current research in AI and Robotics is mainly focused on the problem of localization and identification of victims. Another active area of research is the simultaneous mapping and localization problem for agents, where an agent has to localize itself (with a map) while at the same time building a map (with respect to its current location).

3 Design and resource details

3.1 Challenges

The recent port to the newer Unreal Engine has created the need for victims in USARSim. The previous implementation contained an RFID tag based on which victims could be identified. A more realistic way of identification would be based appearance and behaviour. Both the identity and the status of a victim should be able to be determined based on cues[12, p. 9] emitted by appearance and behaviour.

Necessary steps in adding victims into the virtual world include:

- Gathering knowledge about
  - The features of real victims in a real urban search and rescue task
  - The used 3D models, including their structure and animation sets
  - The combining (blending and sequencing) of animations
- Build behaviour sequences based on available animations in the animation sets

- Adding the victims to the world and triggering their behaviour sequences

The presented design roughly consists of two parts, the behaviour and the appearance of the victims. Since the two are independent enough (for example the behaviour will not depend on the outfit a victim is wearing, nor the other way around), the sections below will address each seperately. If anything, behaviour would influence appearance, as appearance would not influence behaviour easily (unless one gets startled by his or her own appearance). This can be visualized by thinking of a sweating response of the body to the mind of the victim. While the appearance has yet to be experimented with, certain concepts of the appearance will be adressed, such as the importing of textures into USARSim in appendix A.1 and future research in section 4.

The claim here is not that this is a very accurate model of the actual world, but rather that it captures the concepts of a more abstract causal description of the world. One starts sweating as a result of the current state of mind, whereas state of mind does not generally depend on one's own appearance. Even though the model may not be the true behaviour in the real world, it can be reliably similar to some extent [8].

3.2 Used models and animations

As victims were available in USARSim before the switch to UE3, the logical first attempt would be a direct port. Unfortunately the victim models were not placed as controllable agents, but rather as static statues looping a single animation. Another set of models was available on the Unreal Development Network (UDN) [2], and it featured virtually the same models. Virtually, because the number of bones in the skeleton was lower in the imported USARSim models. Since this was likely to be an optimization for the older engine and since there might be animations in the available set of animations that would depend on the extra bones, the (original) UDN models were imported.

The set of animations included with the UDN models lacks any victim-like behaviour such as lying on the ground, or otherwise behaving injured or exhausted. Extra animations were included with the previous USARSim version and because the models are so alike, the set of animations could be and is used to extend the possible behaviour of the victims.

There are a lot of animations that come with the mentioned models (94 for the male and 81 for the female model), but not all animations are of course appropriate for post-disaster victims. Table 1 shows a table of the available animations in the male and female animation sets. Not just the amount differs, but most animations do as well. The reasons behind the choice of having a different animation set

[2]*Example Models for the Unreal Engine* by Tom Lin, URL: http://udn.epicgames.com/Two/UnrealDemoModels.html

Table 1: Available actions for the available male and female victim models, as pulled from USARSim (final UT2004 based release)

Victim	Face or head***	Lying	Kneeling	Dead***	Standing	Movement*	Others
	Expressions:	**On back:**	Boot	**On back:**	Idle_Rifle**	CrouchX	Pickup
	Anger	CheckHead	Crouch	DeadOnBackMale	Idle_Biggun**	WalkX	Throw
	Disgust	CheckHeadBreathe			IdleGloves	RunX	
	Fear		LArmAmp_AHHH	**On chest:**	IdleKnuckles	SprintF	ImpactX***
	Happy	Supine	LArmAmp_Breathing	FaceDownDeath	IdleNeck	Falling	
	Sadness				IdleNose	FlyX	Push
	Surprise	**On chest:**			IdleScratch	FlyIdle	
Male		facedown			IdleStretch		HoldGun
	Visemes:	FacedownBreathing				Jump	
	Vis01-Vis13					Jump_Takeoff	
	Others:	**Upright:**				Jump_Mid	
	bLink	BrokenLeg				Jump_Land	
						JumpUp	
		HugKnees					
		HugKneesBreathe				JumpFTakeoff	
						JumpF	
		LookAround				JumpF_ok	
		LookAroundBreath				JumpFMid	
						JumpFLand	
		LostMyEye					
		LostMyEyeBreathing				Swim	
						SwimB	
		Writhing				SwimL	
		Writhing2				SwimR	
						SwimTread	
		SittingUp					
	Expressions:	**On back:**	Boot	**On back:**	Idle_Rifle**	CrouchX	IdleStretch
	Anger	justbreathe	CrouchIdle	DeadOnBack	IdleBreathe	WalkX	
	Disgust	LayedOutBreatheFast			IdleHip	FlyX	ImpactX***
	Fear	LayedOutBreatheSlow		**On chest:**	IdleTwist	FlyIdle	
	Happy	OnBackBreathing		pronedead		RunX	Pickup
	Sad					Falling	
	Surprise	PassedOut					Push
						JumpTakeoff	
	Visemes:	PostBonkedForehead				JumpSwitch	Shove
	Vis01-Vis14					JumpLand	
Female		Supine				JumpUp	Snowball
	Others:						
	bLink	**On chest:**				JumpFTakeoff	
	HeadTurnL	ChestDownBreathing				JumpF	
	HeadTurnR	ChestDownBreathing2				JumpFMid	
						JumpFLand	
		FaceDownDislocation***					
						Swim	
						SwimB	
						SwimL	
						SwimR	
						SwimThread	

```
*   Where X is one of the following letters, denoting a direction: B(ackwards), F(orwards), L(eft) ,R(ight)
**  Not a rifle animation, but an idle standing one (Rifle animation is with mouth closed, Biggun with mouth open)
*** These animations contain 2 frames (they are static, but may be blended into similar animations, if any)
```

between the two (similar) models, are best explained on the UDN website where they can be found[2]:

> "First of all, the male and female models are substantially different sizes. Secondly, the models don't share the exact bone hierarchy in the head. Finally, because we were doing very delicate and precise animation frames (adjusting the faces for lip synch capacity), it was deemed unwise to try to adapt them to each other."

The choice of specific animations included with the two different sets supplied with USARSim and the reasons behind the inconsistent names remains unknown for now.

The animations in Table 1 are spelled as they were supplied. The grouping is based on characteristics the animations have in common, which is not always clear from the animation name alone. Animations can be sequenced or blended together. An example of blending can be found on the UDN website [3]:

> "A new animation could be created, *talking*, which can be selectively played on certain bones during either the *happy* or *sad* animations. This saves animators from having to create separate *talking while happy* and *talking while sad* animations."

As selectively activating bones is applicable to all of the animations, instead here the animations themselves are described and whether or not they can be sequenced (without selectively playing bones). Detailed implementation details and information on blending in the Unreal Engine can be found in the appendix.

The ordering of the (horizontal) groups in Table 1 is based on degree importance for victim behaviour, from important on the left, to the probably inapplicable group "Others" on the right. The ordering should not be taken strictly, but as an indication of the type of animations that belong to that group. Animations are separated by a newline if begin and end poses differ and therefore simple combination by sequencing of animations would not be sufficient. Movement would make blending less noticeable, but the post-disaster situation we are considering does not generally contain victims that move, or they move at slow pace.

3.2.1 Face and head animations

The group "Face or head" has similar animations in both the female and the male animation set and contains facial expressions (for simulating the status, mood and emotions of victims) and visemes (for lip syncing). Pictures of the individual expressions and blinking animation can be found in Table 3.2.1. The dashes in the viseme names in Table 1 indicate a range of numbered viseme animations. The female set has an extra viseme, but Vis14 is virtually a duplicate of Vis13 (the difference is unnoticeable for normal applications, Vis14 can be ignored). There are two extra female animations in this group for looking left and right. Each of the animations is static, in other words consist of a single pose that is supposed to be blended in another animation. The emotions as depicted in the figure correspond to a complete set of basic emotions[7]. Noteworthy is also that blending together "Fear" (70%) and "Disgust" and "Sad(ness)" seemed to give a reasonable victim expression, but this can of course be set dynamically through UnrealScript.

[3] *Preparing a Character for use with the Unreal Skeletal Animation System* by Erik de Neve and James Green, URL: http://udn.epicgames.com/Two/SkeletalSetup.html

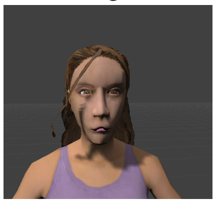
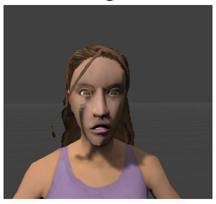
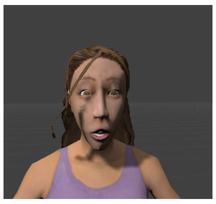
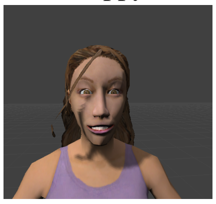
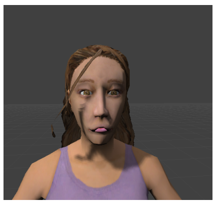
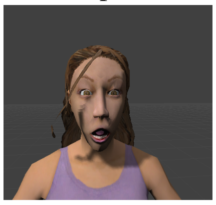
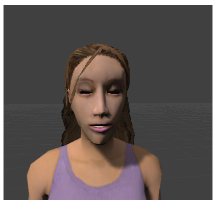
Anger	Disgust	Fear	Happy
			
Sad (Sadness)	Surprise	bLink (Blink)	
			

Table 2: Included facial expressions (excluding visemes) for the female UDN model.

3.2.2 Lying animations

The group ”Lying” contains a big list of animations specifically implemented for USARSim usage. Both models can lie on their back and chest through animation, the male model can also sit upright a bit. The most important animations are described below. **Male model:**
CheckHead: reach for the head while lying on the back flat. Appending *Breathing* uses the same starting pose, but does not grab for the head anymore.
Supine: similar to reference pose (arms and legs spread), but lying on back and breathing.
facedown: lie face down and try to get up. *FacedownBreathing* does the same, but does not try to get up.
BrokenLeg: sit upright with legs stretched and reach for left leg
HugKnees: sit upright with knees bent, bend forward and backward while hugging knees as if scared. Appending *Breathe* does the same but does not bend forward and backward.
LookAround: sit upright with knees slightly bent, look around and bend left knee a little more. Appending *Breath* does the same but does not bend knee more and does not look around.
LostMyEye: lie on right side, grab eye with left hand, seemingly waving with the other. Appending *Breathing* does the same except for waving.
Writhing: writhe, lying on back. *Writhing2* does something similar but reaches up with right arm.
SittingUp: (simple) animation of trying to sit up from lying on the back.
Female model:
justbreathe (or *LayedOutBreatheFast*): lie on back and breathe. *LayedOutBreatheS*-

low is the same animation but is one second longer. Finally *OnBackBreathing* can be sequenced with these 3 animations if blended a little (the position of the skeleton is almost the same, merely the arms are moved) *PassedOut*: lie on back with knees bent, left arm stretched out to the left *PostBonkedForehead*: reach for head with left hand, move both legs individually. *Supine*: see *Supine* description of the male model *ChestDownBreathing*: lie face down, breathing (*ChestDownBreathing2* seems to be another near duplicate)

3.2.3 Kneeling animations

The animation *Boot* exists for both models and consists of an animation where one leg is on the ground and the victim is resting on the other. The animations *Crouch* for the male model and *CrouchIdle* are part of the crouching movement animations and look less appropriate for victims (actively crouching). The male animation set has two extra kneeling animations. The first is *LArmAmp_AHHH*, where both knees are on the ground, the victim uses his right arm to grab his left arm and he leans backwards as if in severe pain. The second, *LArmAmp_Breathing* starts from the same pose as the first, but does not do any leaning.

3.2.4 Death poses

Though appropriate, the animations in this group speak for themselves. Each of these are static poses and supposed to combined with another animation (or not, considering they are dead).

3.2.5 Less relevant animations

The animations (or groups) not mentioned above are considered less appropriate when it comes to victim behaviour. However, even these animations are valuable as they can be played selectively on certain bones, using for example merely the finger movements of the female *SnowBall* animation.

3.3 Behaviour and states

UnrealScript allows for state-machine like programming, leaving open plenty of possibilities for implementations of existing algorithms and concepts from AI. This provides an intuitive interface to implementing concepts such as the state of the environment and (mental) state of a victim. The actual animations that are used are defined by an animation tree inside the Unreal Editor. The animation tree (selectively) activated through UnrealScript. A useful sidenote is that certain animations are called by the engine natively, see the appendix for an example of a native falling animation.

UnrealScript is based on the Java programming language and therefore has many things in common, such as that classes extend one another in order to avoid code redundancy. The combination of state-machine like programming with a class hierarchy works well for a simulated environment.

The created victim code structure consists of three files, *VictimPawn*, *Victim-Controller*, *FemaleVictim* and *MaleVictim*. The last two files extend *VictimPawn*, initializing the male and female specific variables containing the assets in the victim package as mentioned in the appendix.

It all starts with an *Actor*, this is the base class for all gameplay objects. Both the Pawn and Controller class are (indirect) children of this class. The Pawn class can be seen as the physical representation of the victim, something external to what motivates it to behave. The Controller class on the other hand, while it may depend on events from the Pawn class or engine, is what actually controls the victim.

To trigger an animation on a victim, one would have to add the necessary functionality to the Pawn class and then make a call to execute this functionality from the Controller class.

3.4 Appearance

The imported UDN models were created with certain goals in mind. While the goals originated from the gaming industries, the design choices are important as they show what the models are capable of and what may be improved upon. The goals as presented on the UDN website[1]

(Inter)changeable skin/clothing[1]

The design part of this thesis is concluded with a few statements on the used victim models as given by the author.
Information about the hands[2]:

> "...each finger joint has two segments around each joint, which is important in keeping the deformation to a minimum when they are bent. If I was to make the models again, I might budget even more resources to the hands; the knuckles are not clearly defined when the hand is closed and the thumb could use more geometry where it joins with the hand."

Information about the feet[2]:

> "I experimented a little bit when making the feet. As explained in the UnrealModeling doc, perspective in games often makes 'normal' human proportions seem wrong. This is especially the case with feet, where the camera point of view is often looking down at your models from a position above and behind your character. To compensate for this effect, I made the feet on the male model much larger than is normal. I made the girl's feet normal sized, so that evaluating the appropriateness of both sizes is possible."

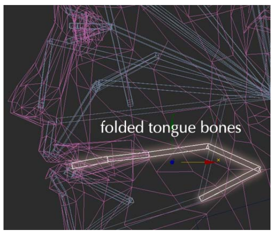


Figure 1: Folded tongue bones, Courtesy of Tom Lin[4]

Information about the tongue [4], see Figure 3.4

> "A tongue seems fairly self explanatory, and depending on how much movement you'll require, it could be. If you'd like to have major tongue extension outside of the mouth, however - sticking out the tongue at someone, or licking lips - you'll need a slightly unorthodox bone structure. Unreal CAN NOT deal with bones that stretch over time - it can only understand orientation and position of bones. I found that having the bones fold on themselves worked best for the tongue. Since it's not possible to stretch the existing bones, we unfold them, like an accordion."

4 Future directions

While the port of the models themselves has immediately been adopted by the USARSim community, most of this thesis should be seen as foundations for doing future research. The new engine brings new possibilities for simulate the dynamics and variance of a real world situation. This section will address the benefits of this thesis for future research and where future research with regard to virtual victims is likely to be beneficial. The section is concluded with resources that would be useful to have in USARSim, based on information acquired throughout the project.

Development and research in USARSim has gained virtual victims, including a dynamic design of behaviour. This puts research with a wide selection of sensor types back in business. Based on the more dynamic behaviour and blending of animations, sensors such as motion based sensors and object detection algorithms may perform better than before. A thorough description on the creation of a package in the Unreal Editor is given, allowing for additional victims to be easily added, boosting the variance of the virtual world.

Future research would be beneficial in a number of directions. One direction is the environment. Due to the newer engine models, materials and physics can be better simulated. Soft-body physics is also added in Unreal Engine 3, but only in

[4] *Unreal Modeling Guide* by Tom Lin, URL: http://udn.epicgames.com/Two/UnrealModeling.html

UDK and is therefore not yet available in USARSim at the time of writing. Ragdoll physics are mentioned in the appendix but have yet to be correctly combined with victim models.

Another direction would be the victims themselves (although they are also part of the environment mentioned above). Facial expressions can be modeled to great extent and should be further explored as detection of a face is of great importance in search and rescue. The iris of the eyes is connected to a bone, allowing iris movements to be part of facial expressions. Better models may be nice, but the current are already capable of sign language and lip synchronisation, which might be something worth investigating first.

Another promising direction would be adding heat emission to a victim (and detection to a sensor) or emotion. Emotion would likely best be implemented through the Controller class of the victim, as part of the "mental" state of the victim.

What is needed with respect to information gathered during this project would mainly be a more concise list of animations in both animation sets. Both the names as well as the animations themselves have to be cleaned up. As blending is possible per bone, specific animations (for example multiple arm movements, multiple laying poses, etc.) would be most beneficial. Hand(/finger) and head movement gestures are by themselves worthy additions, as these are (parts of) objects that are defining of humans (and thus often used for object detection).

5 References

References

1.) 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.

2.) S. Balakirsky, C. Scrapper, S. Carpin, and M. Lewis. UsarSim: providing a framework for multirobot performance evaluation. In *Proceedings of PerMIS*, volume 2006, 2006.

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

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

5.) S. Carpin, J. Wang, M. Lewis, A. Birk, and A. Jacoff. High fidelity tools for rescue robotics: Results and perspectives. *Robocup 2005: Robot Soccer World Cup IX*, pages 301–311, 2005.

6.) P. Ekman and W.V. Friesen. A new pan-cultural facial expression of emotion. *Motivation and Emotion*, 10(2):159–168, 1986.

7.) P. Ekman, W.V. Friesen, and P. Ellsworth. What emotion categories or dimensions can observers judge from facial behavior. *Emotion in the human face*, pages 39–55, 1982.

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

9.) C.E. Fritz and E.S. Marks. The NORC studies of human behavior in disaster. *Journal of Social Issues*, 10(3):26–41, 1954.

10.) 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.

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

12.) R. R. Murphy. Human-robot interaction in rescue robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(2):138–153, 2004.

13.) S. Nunnally and S. Balakirsky. Acoustic Sensing in UT3 Based USARSim. In *Robots, Games, and Research: Success stories in USARSim", Workshop Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2009)*, pages 68–73.

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

15.) 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.

16.) J. Wang, M. Lewis, and J. Gennari. USAR: A game based simulation for teleoperation. In *Human Factors and Ergonomics Society Annual Meeting Proceedings*, volume 47, pages 493–497. Human Factors and Ergonomics Society, 2003.

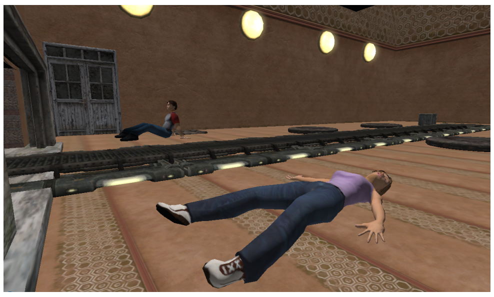


Figure 2: A single frame capture of victims as implemented in USARSim (based on UT3).

Appendices

A Constructing a victim package

A *package* refers to a set of assets, where common examples of assets are 3D models, 2D textures, sounds or animations. Both UDK and UT3 allow for packages to be imported easily into the Editor and generally accept the same import formats, but a UDK package cannot be directly imported into UT3 and vice versa. This appendix will guide the reader through the steps necessary to import all of the assets used for victim animation in the Unreal Editor (UT3 based USARSim), but the steps have been verified to work within UDK as well. The result of going through the whole process of importing and setting up the world can be seen in Figure 2.

Since different assets of a package require different methods to be constructed or imported, this section is divided into the subsections *Textures and Materials, Meshes, Animations, AnimationTrees, PhysicsAssets* (not used in described design) and the *UnrealScript* code is included in appendix B (elaborating on the class structure explained in section 3.3).

A.1 Textures and Materials

To import a texture: open the Contents Browser, click `File > Import...`. Imported images can only be of certain formats, the textures delivered with the models can be directly imported as they are in `.TGA` format and have the correct layout.

When importing a texture, a property window will appear. For the victim package the default settings were used, except for `Create Material?` being checked, and `Two Sided?` being checked for just the hair. The textures in this package were given the same package and group for convenience.

Once imported, for the hair texture to appear transparent, the hair material should be edited. Doubleclick it in the *Content Browser* and drag the top-most layer (the *RGB channel*) to `Diffuse` and the lower one (the *alpha channel*) to

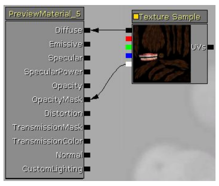


Figure 3: Unreal Material Editor, example setup for transparency.

`OpacityMask` in the editor. You can click the `Texture Sample` to select it and then hold control and drag it to move it if it is blocking the in- and output nodes. See Figure 3 for an example.

Now deselect the Texture Sample (click anywhere else in the editor) and put `BlendMode` to `BLEND_Masked`. If all is well, the material preview sphere will render transparency.

A.2 Meshes

Import the mesh the same way as textures, in this case the `.PSK` files of the models were imported. The default settings were used (no Maya coordinates).

Once imported, the mesh looks rather gray. To add the created materials, edit the mesh and add them to the `SkeletalMesh` properties window in the lower left. `Material [0]` should be the skin, `Material [1]` the cloth and finally `Material [2]` the hair.

Since the mesh's settings are set through UnrealScript, these can be found in the source in appendix B. If one would rather set the settings through the Editor, the variables and values have the same name in there.

A.3 Animation set

In the same mesh screen as above, create a new *AnimSet* (`File > New AnimSet`) and then import the corresponding AnimSet. This can be done by `File > Import PSA`. Both the AnimSet and AnimTree assets were given the same group in this package for convenience.

A.4 AnimTree

The *AnimTree* or animation tree is what determines the animation that is actually played. Figure A.4 depicts an AnimTree created for the female victim. Noteworthy nodes are *AnimTree*, *AnimNodeBlendPerBone*, *AnimNodeCrossfader*, *AnimNodeRandom* and *AnimSequence Player*.

AnimTree is the root of the tree, where all the animations combined lead to.

AnimNodeBlendPerBone lets you pick specific bones are defined that the animation below plays on. In this example, there are two instances of this node. The

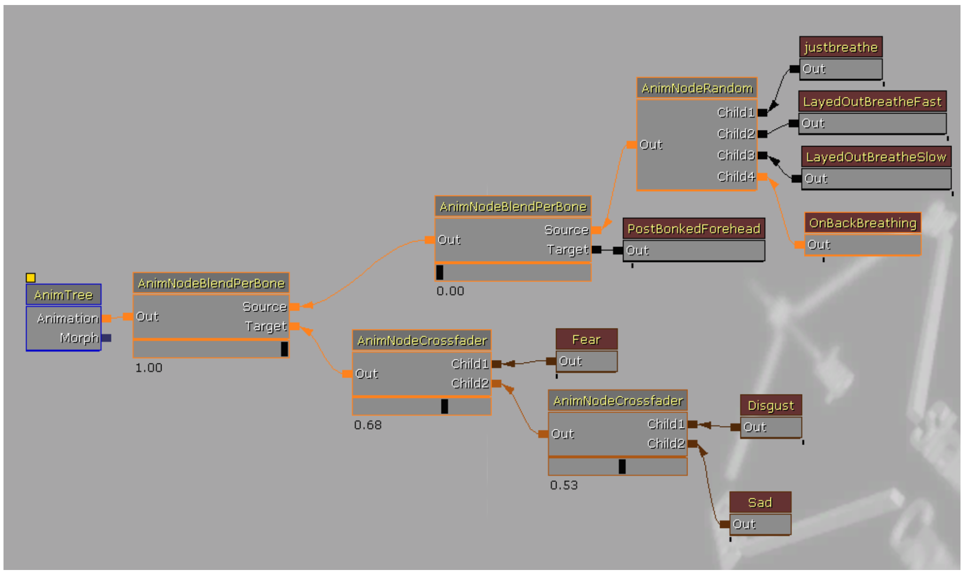


Figure 4: An example AnimTree in the Unreal Editor (*main AnimTree example*).

left-most specifies to play the facial animations on the bottom on the HeadBone. The top-most makes sure PostBonkedForehead (a full body animation) only triggers the left arm.

AnimNodeCrossfader lets you merge two animations - the slider underneath indicates the crossfading percentage.

AnimNodeRandom allows randomly picking an animation. It seems only one random node is supported within an AnimTree, more functionality would have to be implemented through UnrealScript.

`AnimSequence Player` denotes a certain animation. If for example just one animation was wanted, the output node of an AnimSequence Player can be directly connected to the Animation input node of the AnimTree.

Figure 5 depicts an AnimTree created for the male victim. All nodes are mentioned above except for *AnimNodeBlendByPhysics*. Blending by physics is natively done by the engine when it detects and passes on certain events. The physics flag can also be set through UnrealScript.

A.5 PhysicsAssets (not used in described design)

The *PhysicsAssets* defines how the joints and constraints in the skeletal mesh work. The created victim physics asset, not used due to scaling issues, was made by right clicking the Mesh in the Content Browser and selecting `Create New Physics Asset`. The minimum bonesize can be increased to skip bones such as fingers or inside the head, which make the (manual!) process of fixing the constraints and joints a lot of work. The other details can be left to default, though for *Collision Geometry* the

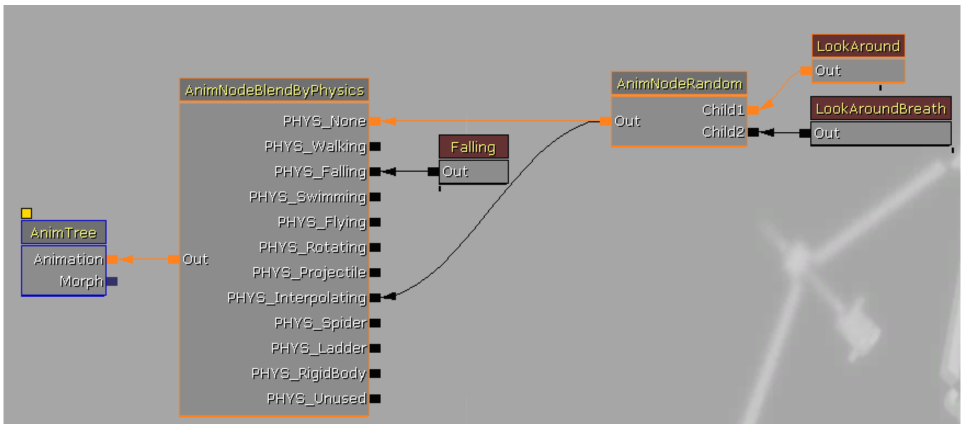


Figure 5: An example AnimTree in the Unreal Editor, featuring *AnimNodeBlend-ByPhysics*.

value `Sphyl/Sphere` is often used for characters. This setup should provide a very basic PhysicsAsset, that still has to be corrected for realistic joints, constraints, and bone weights.

B UnrealScript example source

Listing 1: FemaleVictim.uc

```
Class FemaleVictim extends VictimPawn config(USAR) placeable;

defaultproperties
{
  Begin Object Name=WPawnSkeletalMeshComponent
    SkeletalMesh=SkeletalMesh'VictimPackage.Mesh.GenericFemale'
    PhysicsAsset=PhysicsAsset'VictimPackage.Mesh.GenericFemale_Physics'
    AnimTreeTemplate=AnimTree'VictimPackage.animation.GenericFemale_AnimTree'
    AnimSets.Add(AnimSet'VictimPackage.animation.GenericFemale_AnimSet')
  End Object

  bIsFemale = TRUE
}
```

Listing 2: VictimPawn.uc

```
class VictimPawn extends UTPawn config(USAR);

var AnimTree defaultAnimTree;
var array<AnimSet> defaultAnimSet;
var AnimNodeSequence defaultAnimSeq;
var PhysicsAsset defaultPhysicsAsset;

var VictimController MyController;

var AnimNodeBlendList AnimationList;

defaultproperties
{
  Begin Object Name=MyLightEnvironment
    ModShadowFadeoutTime          = 0.25
    MinTimeBetweenFullUpdates     = 0.2
    AmbientGlow                   = (R=0.01,G=0.01,B=0.01,A=1)
    AmbientShadowColor            = (R=0.15,G=0.15,B=0.15)
    ShadowFilterQuality           = SFQ_High
  End Object

  Begin Object Name=WPawnSkeletalMeshComponent
    bOwnerNoSee                   = FALSE
    bUseAsOccluder                = TRUE
    CastShadow                    = TRUE
    bCastDynamicShadow            = TRUE
    LightEnvironment              = MyLightEnvironment
    LightingChannels              = (Dynamic=TRUE)

    BlockZeroExtent               = TRUE
    BlockNonZeroExtent            = TRUE
        CollideActors                 = TRUE
    Scale                         = 0.75

    RBChannel                     = RBCC_Pawn
    RBCollideWithChannels         = (  Default=TRUE, GameplayPhysics=TRUE, EffectPhysics=TRUE,
                  Vehicle=TRUE, Untitled3=TRUE, Pawn=TRUE  )
  End Object
  Mesh = WPawnSkeletalMeshComponent

  Components.Add(WPawnSkeletalMeshComponent)

  Begin Object Name=CollisionCylinder
    CollisionRadius=+0022.000000
    CollisionHeight=+0022.000000
    BlockZeroExtent=false
    Scale=0.75
  End Object
  CylinderComponent=CollisionCylinder
  CollisionComponent=CollisionCylinder

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

simulated function PostBeginPlay() {
        super.PostBeginPlay();
    SetPhysics(PHYS_Interpolating);

    if (MyController == none) {
        MyController = Spawn(class'VictimController', self);
        MyController.SetPawn(self);
    }
}

simulated event Tick(float DeltaTime) {
    Super(Pawn).Tick(DeltaTime);
}
```

Listing 3: VictimController.uc

```
class VictimController extends AIController config(USAR);

var VictimPawn MyVictimPawn;
var Pawn thePlayer;

var float distanceToPlayer;
var float distanceToTargetNodeNearPlayer;
var Name AnimSetName;

var Float IdleInterval;

var float perceptionDistance;
var bool PlayerInSight;

defaultproperties
{
    perceptionDistance = 1000

    AnimSetName = "IDLE"
    IdleInterval = 2.5f
    PlayerInSight = false;
}

function SetPawn(VictimPawn NewPawn) {
    MyVictimPawn = NewPawn;
    Possess(MyVictimPawn, false);
}

function Possess(Pawn aPawn, bool bVehicleTransition) {
    if (aPawn.bDeleteMe) {
        `Warn(self @ GetHumanReadableName() @ "attempted to possess destroyed Pawn" @ aPawn);
        ScriptTrace();
        GotoState('Dead');
    } else {
        Super.Possess(aPawn, bVehicleTransition);
        GotoState('Idle');
    }
}

state Idle
{
    event SeePlayer(Pawn SeenPlayer) {
        thePlayer = SeenPlayer;
        distanceToPlayer = VSize(thePlayer.Location - Pawn.Location);
        if (distanceToPlayer < perceptionDistance && !PlayerInSight) {
            Worldinfo.Game.Broadcast(self, "I can see you!");
            PlayerInSight = true;
        } else {
            PlayerInSight = false;
        }
    }

    Begin:
            Worldinfo.Game.Broadcast(self, "Idle ...");

        Sleep(IdleInterval);

        GotoState('Idle');
}
```