SESSION 2C

GAMES TECHNOLOGY AND VIRTUAL ENVIRONMENTS
Secure Learning in 3 Dimensional Multi User Virtual Environments – Challenges to Overcome

Indika Perera, Colin Allison, Alan Miller

School of Computer Science, University of St. Andrews,
North Haugh, St. Andrews, Fife, KY16 9SX, Scotland
{indika, ca, alan}@cs.st-andrews.ac.uk

Abstract— This paper presents research in progress about security management in learning processes situated in 3 Dimensional Multi User Virtual Environments (MUVE). While there is considerable interest throughout academia in using such MUVE for education, academics are experiencing many challenges in exploring various possible use-cases. Awareness of the security challenges associated with learning in 3D MUVE helps to formulate sustainable and reliable learning activities, as well as in designing security solutions for those challenges. Moreover, the effective approaches to manage user identities, access rights, and learning content ownership helps to address most of the security, privacy, and trust issues within a virtual learning environment. Therefore, with the emphasis on those key issues, this paper initiates a discussion of the security issues in 3D MUVE based learning from our experiences.

I. INTRODUCTION

3D Virtual worlds are getting into various segments of our society day by day. Virtual worlds where thousands of people can interact simultaneously in a shared 3D space, show frontier and critical implications for business, education, social and technological sciences, and society at large[1]. The world’s leading universities have been researching on how to use this novel technological medium for their learning processes. In fact, virtual worlds are likely to become a mainstream feature of UK education [2]. They are particularly appropriate for educational use due to their alignment with the Kolb’s[3] concept of experiential learning [3], and learning through experimentation as a particular form of exploration [4]. Dalgarno and others have described how researchers have argued that interactive 3D virtual environments demonstrate a great educational potential due to their ability to engage learners in the exploration, construction and manipulation of virtual objects, structures and metaphorical representations of ideas[5].

Accordingly, many higher education courses when looking for novel and engaging approaches to conducting their practical coursework are interested in the potential of virtual worlds in academia. With the interest for extensive use of 3D MUVE for learning, we believe that the security aspects are essential for its success. As a result this research primarily focuses on developing effective security models for learning in 3D MUVE. For our research and learning activities, we choose Second Life[6] and Open Simulator[7] MUVE; more details about the work we have done with these environments will be discussed later.

II. BACKGROUND AND RELATED WORK

Managing computer security is a broad area of interest with many sub disciplines. It covers technical aspects and managerial aspects in the context of different domains. From the technical point of view, Confidentiality, Integrity, and Availability are the fundamental aspects of information security[8][9]. Many security researchers have based their work on these three aspects. Some have tried to extend the model further with other aspects as well. Despite the different forms of attacks and threats, system security goals are based on Confidentiality, Integrity, Availability and Identity[10]; identity can be also be considered under confidentiality, however. Organizational information based soft assets endure threats to their integrity, availability and competitiveness, i.e. the confidentiality [11]. Moreover, any successful security management policy or practice should comparatively address these three basic security goals.

Another dimension of security management is to enhance security, privacy, and trust within the security management framework. Not like the basic security goals, this perspective is somewhat ambiguous, as some prefer to consider all three aspects under the umbrella term security, where as others prefer to consider separately. Hurlburt and others have clearly indicated how security and privacy could be defined in different viewpoints [12]. Understanding how the different human, organizational and technological elements interplay could explain how different factors lead to security vulnerabilities within organizations[13]. Importantly, these different viewpoints could lead to heterogeneous security management practices. Also, security and privacy requirements vary with different learning parameters such as technological differences, learning organization objectives and practices, different learner focus, and content management approaches[14]. However, the security management models based on commercial aspects have limitations in the context of learning and education. Furthermore, Rezgui and Marks observe that universities are amongst the least IS secured
environments with a poor level of security awareness programs, citing a survey by EDUCAUSE [15]. These emphasise the importance of security management practices in virtual learning.

Weippl in his book on e-Learning security, has also considered an extensive set of factors and use cases for e-Learning security management [16]. Therefore, to formulate sustainable security management models for 3D MUVE learning environments, we have to consider these different security management aspects while preserving our focus on possible security issues for 3D MUVE based learning. With that in mind, we have started to identify critical security issues for learning in 3D MUVE as the first phase of our research.

A. Previous Work

Various educational projects at the University of St Andrews have used virtual environments in their course delivery. These include LAVA [17] and WiFiSL [18]. MMS, the Module Management System, is an online learning management system which interoperates with Second Life in order to maintain an association of institutional and virtual world identities as one of its many features. The Lacroix Acropolis Virtual Archaeology (LAVA) project allows students to engage with a simulated archaeological excavation, and then explore a recreation of the site in Second Life. The WiFi Virtual Laboratory in Second Life project (WiFiSL) aids teaching and learning about wireless networking by using virtual world interfaces to collaboratively explore and visualise simulations of wireless traffic. Further, we have successfully used Second Life [19] and OpenSim for teaching Human Computer Interaction.

III. SECURITY ISSUES FOR LEARNING IN 3D MUVE

With consideration of various factors, we found the following three security concerns; Identity Management, Access Rights Management, and Content Ownership Management are essential for learning activities. Since most of the other identified security issues can be addressed through successful solutions to the above three concerns, we are interested to narrow our focus to these.

A. Identity Management

The proper management of user identities is one of the fundamental system security requirements for multi-user system environments. There have been various security practices developed to facilitate accurate identity management. 3D MUVE use the familiar username and password based authentication approach as their primary identity management method. If the academic institution uses a proprietary 3D MUVE such as Second Life and relies on its user registration, the institution has to develop its own approach to map user identities accurately. Using the general user registration interface of Second Life, users can easily provide fake registration details, hence, masquerade as the real identity. This creates a significant flaw with accurate mapping between student real world identities and their Avatar identities. Moreover, if the learning institution has an existing identity management method, which is the obvious situation, there is another dimension of mapping avatar identities with the existing method. Second Life allows a customizable option of user registration through its RegAPI service; however, it does not provide complete independence for selecting usernames as required. Further, it requires additional payments based on the services being used. On the other hand, the Open Simulator has more freedom on user management, which allows creating avatar names without any restriction. This provides the opportunity to use existing unique user identities in an academic institute.

Second Life and OpenSim use the same approach of identity management within an avatar session. It is based on the Capability concept which each entity (avatar, content object, texture, region, etc.) gets a unique URL for a given session of user interaction. These capabilities are dynamically generated UUID based URLs and act as session tokens for that time period. However, an actual capability URL is encapsulated from a fake URL, which is the visible form of the Capability URL to the users. Internally, the Capability server maps the in-world objects to their representing URL. This can be a single point of failure if the Capability service is compromised or fails. However, since it is an architectural approach, the impact on its use for learning has no specific influence. Importantly, the Identity mapping between existing learning systems is a significant challenge as we expect higher levels of interoperability with heterogeneous learning environments.

B. Access Rights Management

Role Based Access Control (RBAC)[20] or Access Right Management is the next requirement after authentication, for a secure system environment. Access Control on created content will be considered under content ownership. With respect to environment access, there are different roles and access categories we should consider. The highest privileged user category is known as ‘God’, which represents the administrator role of the environment. In Second Life, this role is used only by Linden Lab’s employees; OpenSim allows anyone to use this role within their own installations. The appropriate use of this role in a private OpenSim based learning environment is important, as this role can overrule other roles’ actions. The ‘Region Owner’ role has the next highest authorization to manage the owned virtual region. A region can be a single virtual island or a group of islands. There are number of functions including, land management, user management, media management, etc. with respect to a region, where the Region Owner can practice. In a situation where a region/has been divided into different sub areas (known as land parcels) by the Region Owner, these parcels can be again assigned to different users, known as ‘Parcel Owners’. Parcel Owners also get some administration capability within the control span of the respective Region Owner. For a given region, its owners or environment administrators can decide a number of services such as Flying, Content Creation, Teleporting, and Media Streaming, etc., to be offered to its residents or visitors. Clearly, these access control mechanisms are new to the existing learning
environments; hence require proper research on how these should be used.

A key form of 3D MUVE user collaboration is through the concept of groups, which can be used to manage learning activities as well. Group management also comes with a certain level of access right management. Mainly, the roles are ‘Group Owner’, ‘Officer’, and ‘Member’. However, the Group Owner can create customized user roles with different access right levels if required, from the predefined set of functions.

The important point we want to emphasize is, that with the exception of the Group management, the other virtual-land based controlling models are new to the existing virtual learning access control mechanisms. In order to relate these new RABC models with present learning systems, we need to research different use cases for learning in 3D MUVE and formulate appropriate RABC models for the requirements.

C. Content Ownership Management

3D virtual content creations and transactions are multibillion industries today. Content frauds and thefts have become a major concern for MUVE service providers and end users. Even though within a learning environment, we are not interested on commercial transactions on the created content, it is possible that the virtual content of users gets affected, if there has not been secure ownership practices implemented. Second Life and OpenSim Content Ownership Management models are identical. Second Life gives no flexibility to manage even though OpenSim allows ownership management at the configuration and the source code level. The fundamental difficulty with the existing models is that they do not extend readily to offer appropriate support for learning requirements. Awareness of the present content ownership model helps to customize the OpenSim environment for specific learning requirements. Even though Second Life and OpenSim do not reveal their content ownership model effectively, with a few experiments using multiple avatars, we identified that the basic norms of the model are as shown in Table 1.

<table>
<thead>
<tr>
<th>Role</th>
<th>Move (V)</th>
<th>Modify (M)</th>
<th>Copy (C)</th>
<th>Transfer (T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creator</td>
<td>True*</td>
<td>True*</td>
<td>True*</td>
<td>True*</td>
</tr>
<tr>
<td>Owner</td>
<td>True*</td>
<td>True*/False</td>
<td>True*/False</td>
<td>True*/False</td>
</tr>
<tr>
<td>Next Owner</td>
<td>True</td>
<td>True/False</td>
<td>True/False</td>
<td>True/False</td>
</tr>
<tr>
<td>Group</td>
<td>True/False</td>
<td>True/False</td>
<td>True/False</td>
<td>False</td>
</tr>
<tr>
<td>Everyone</td>
<td>True/False</td>
<td>False</td>
<td>True/False</td>
<td>False</td>
</tr>
<tr>
<td>God</td>
<td>Overrule</td>
<td>Overrule</td>
<td>Overrule</td>
<td>Overrule</td>
</tr>
</tbody>
</table>

The content ownership role model is based on two distinct roles; the Creator and the Owner. Furthermore, there is a provision to set permission on the roles Next Owner, Group Member, and any user known as ‘Everyone’. Permissions for different roles are based upon the actions Move, Modify, Copy, and Transfer. The * mark indicates the ability to delegate permissions to the Next Owner, Group and Everyone. The OpenSim model of permission augments the SL model by allowing God privileges on an island. With the God permission model, a privileged user can overrule content permissions which were set by other users.

There is another permission model, which is applicable to all the roles we discussed above: Base permission and Effective permission. These are sets of combinations of the permissions to Move, Modify, Copy and Transfer. According to the implementation, Base permission is the maximum possible permission set available for a given instance of ownership, where as the Effective permission indicates the actual effective permission set that a user gets considering different roles and rules relevant, therefore;

\[
\{\text{Effective Permission}\} \subseteq \{\text{Base Permission}\}
\]

Importantly, the Base permission cannot be elevated through transferring the ownership; it either degrades or remains unchanged. There is another rule for the content ownership, i.e. the owner of the content object always gets the permission to move and delete the object.

One problem with content permission model is that there is a permission loss in cyclic ownership transfer (Fig. 1); i.e., the user A creates an object, transfer it to the user B on default permission settings, and then, B transfers it back to A. In this situation, even though the user A is the creator and the present owner of the content, the new permission set has lost the Copy and Modify permissions irrevocably. This can create unexpected permission issues in learning activities.

Fig. 1 The Cyclic Permission Loss in content transfer with default permission

Many MUVE users are vulnerable to possible threat/fraud approaches due to their poor awareness on the environment and existing ownership models. Moreover, it is possible that a user develops fraudulent content to mislead the tutors. In fact, these could result to plagiarism and misconduct within the learning environment. Misconduct such as plagiarism is a specific issue which has significant importance to academic environments. There have been various studies and tool developments to prevent academic misconduct within traditional and e-Learning environments. However, 3D MUVE allow students a unique set of features, which can be easily exploited to commit plagiarism. Moreover, the present
level of knowledge and tools appear to be less effective in detecting and preventing such misconduct in a 3D MUVE. With the flexibility and open nature of OpenSim, we can alter its current ownership model to suit learning requirements, which will help to overcome most of the content related misconduct and frauds.

D. Other Security Issues

Following are other possible security challenges, which can degrade learning activities. However, with appropriate environment settings, these security concerns can be eliminated. In particular, once the user identities, access rights, and content ownerships are properly managed, the following security issues can be addressed through institutional academic policies and disciplinary actions. Therefore, our main focus is with the security concerns discussed above, for this research.

Automation Attacks - Automation attacks can take two forms: attacks outside the virtual world, or attacks within the virtual world. The former type of attacks is common to any information system, which indeed is a widely explored security concern nowadays. However, the latter type of attack, i.e. attacks within the MUVE is quite novel, yet significant for the learning activities. As discussed above, 3D MUVE provide a dynamic, programmable 3D environment for the users. This flexibility can be exploited through automated avatars also known as Non Player Characters (NPC) or bots. Though the LSL programming is not powerful enough to easily create advanced intelligent bots, a simple script associated to a NPC or content prim could destroy learning activities quite effectively. Moreover, if there is a provision for generic programming languages (C#, etc.) then threats could be more severe than the LSL based attacks, given that no sandbox protection is implemented in the runtime environment.

Spamming - Spam is a common issue with regular email and other message services. Virtual worlds have their own internal message services (instant messages, note-cards, and streaming media) as well as interfacing to external systems through XML-RPC and HTTP. An attacker could try to exploit these communication channels and generate destructive messaging to learning activities. It is possible for a script to fill students’ inventories with genuine like note-cards, resulting in a disturbance to the learning. However, with appropriate security measures, we can prevent spamming in 3D MUVE learning environments without much difficulty.

Denial of Service (DoS) attack – This is a concern for the security of server environments. In fact, there have been numerous researches done to mitigate/prevent DoS attacks and its derivative forms. We also believe that 3D MUVE servers are equally vulnerable as any other server environment. Furthermore, it is observed that even within the 3D MUVE, there can be various forms of DoS type attacks to interfere with the learning processes. For an example, a user could simply create an obstructive content barrier, preventing other students accessing learning materials or completing an assessed task on time. Another possible attack form is that a student consumes an extra amount of a limited resource/service, creating a scarcity, obstructing other students completing their learning tasks. Though it appears a simple concern, the result could be devastating for the whole learning activity.

IV. RESEARCH ENVIRONMENT

The School of Computer Science deployed an Open Simulator installation as a test environment for 3D MUVE learning activities. The main idea was to evaluate present OpenSim capabilities and compare with the existing SL environment. OpenSim is still a developing project, which hasn’t yet reached alpha distribution (the latest stable version as at present is 0.6.8). Therefore, serious functional and non-functional requirement issues are inevitable. This OpenSim environment was initiated with an objective to use it for HCI course work. We are also in the process of transferring existing projects (LAVA, WiFiSL) from Second Life to OpenSim for experimentation. With the flexibility to have multiple regions, we created a dedicated region for the next cohort of students, named ‘HCI’.

Students were assigned a parcel of land within the dedicated ‘HCI’ region giving them a space within which they could build and script with minimal restrictions imposed on them. Each parcel was 2500 sq. m (50m x 50m) size piece of land, with exclusive ownership for the assigned student. (This
was approximately 10 times larger than could be afforded to them by the relatively heavily used Second Life region). There were 18 students registered for the course module; hence an area of 45,000 sq. m. was allocated for student parcels, and the remaining land (20,536 sq. m.), kept as a sandbox, to support student collaboration through a shared place.

Students were asked to individually implement an interactive door system for an enclosure of their own creation. The enclosure system was expected to involve at least two doors - one from simple types and another from moderate/hard categories, given at the beginning of the project. Students were given the flexibility to incorporate their own creative design ideas, while meeting the course project requirements. Among the many successful projects, two creative designs are shown in the Figs 2a & b.

A. Security Issues Experienced

There have been a few issues with the OpenSim test environment. Among those, we found the scripting vulnerability and parcel access issues are specifically important for a learning activity.

Scripting Language Vulnerability - Linden Scripting Language (LSL) is a platform specific scripting language mainly used to allow objects to react to events occurring to/near them. It is the only allowed scripting language in Second Life. Even though the OpenSim sets LSL as the default scripting language, some other languages are also allowed with appropriate server configuration. These languages are C#, J#, and VBScript. By selecting a generic programming language, programmers can call .NET APIs directly within the script. Furthermore, it opens the option to use rich data structures. This is a vital opportunity to advance scripting beyond LSL, which lacks features such as exception handling and advanced data types.

Initially, we decided to use C# for the scripting within the OpenSim environment. Interestingly as expected, we could manage the file system within the server environment, through direct C# File handling.

System.IO.File.Delete("./XYZ.abc");

Facilitation of static method calls such as the above code line creates a severe vulnerability of C# scripting that the OpenSim server environment could be affected. We identified that either through appropriate object channels, or through static methods, any user can attack the server environment by file management API. The other significant disadvantage is that through the client application, a user can perform all these attacks without additional tool support. Therefore, we decided to use LSL for the project work.

Parcel Access Issue - Having dedicated land parcels for student projects is important if the projects are individual or grouped. Also, if there is an assessment, access restriction to non-allowed parcels can be required according to the course. There is a provision to achieve this in OpenSim, through prevention of public access to that parcel. However, even if the parcel shows a virtual fence with wording ‘No Entry’ (see Fig 3); any user can still enter without difficulty.

Having experienced this unexpected system behaviour, we tried an alternative approach known as ‘banned residents’. We made the respective parcel owner as the allowed user for the parcel while putting the rest of the users to the banned list of that particular parcel. This worked as expected during the testing sessions; once the laboratory session started with multiple user sessions, the system failed to manage access controls as per the given banned lists, however.

V. FUTURE WORK

The research overview is shown in the Fig. 4. As the next phase of this research, we would be focusing on identifying and formulating generic security models for different learning use cases. Learning use case identification is particularly important as they can be used for effective security model design. Necessarily, different use cases have various security aspects; hence the required solution models also need to consider these key use cases. Alongside, we will consider the open source e-learning environment MOODLE for existing use cases and its role-permission model.

This helps us to formulate accurate and standard learning use cases and required security enforcement models based on widely used and more reliable knowledge. Moreover, by doing so, we can achieve the interoperability of security models between the OpenSim and MOODLE which support for a secure open source virtual learning system suite with 3D environment support. After that, domain customization of the identified use cases for 3D MUVEs will be done to ensure appropriate learning facilitation.
After that we will be able to develop generic permission and access control models for previously identified 3D MUVE learning use cases. For this phase, our knowledge on existing permission and access models of SL and OpenSim will be helpful to decide possible security enhancements. However, our main consideration will be on learning activities, than generic security model development on 3D MUVE.

The final phase of this research is to implement developed secure learning models as a way of a secure 3D virtual island for learning activities, and evaluate its use for required enhancement. This would eventually help to customize and reuse through different archiving mechanisms such as OAR, IAR, XML, etc., provided by the OpenSim platform.

VI. CONCLUSIONS

Virtual worlds have great potential for engaging students in innovative, immersive learning environments. The use of the world’s most popular virtual world - Second Life - for education however, leaves much to be desired in the area of secure learning with assignments and assessments. With some configuration skill, the validity of online identity and the privacy of personal work from others can be supported. However, the accessibility of work to markers, and reliable submission and assessment are challenges that can be hard to address.

OpenSim is a public domain, open source, virtual world system that can be used in conjunction with the widely available SL viewer. OpenSim, partly by virtue of being capable of local installation, allows staff to gain “root” like access to students’ learning activities, when necessary. However, OpenSim is still under development and the privacy of personal work is not well supported. In addition the extra utility of using a more powerful language than LSL, such as C#, cannot yet be exploited due to security loopholes.

With this research we are looking forward to provide a comprehensive content and access management policies for generic learning requirements in 3D MUVE. The proposed security policy models will be implemented at the application level, independent of the underlying platform constraints to ensure seamless customization and reuse, as required.

ACKNOWLEDGMENT

Second Life region rental was supported in part by the University of St Andrews Fund for Innovations in Learning, Teaching and Assessment (FILTA). The Higher Education Academy for Information and Computer Sciences (HEA ICS) supported part of the work on OpenSim.

REFERENCES


Abstract— Biometric mechanisms of authentication are becoming a popular approach to securing computer systems, networks and other critical resources. This paper proposes a behavioural authentication system based on user interaction with computer games. Literature about current physical and behavioural biometric methods is reviewed and game based authentication design and implementation issues are discussed. Some common biometric data analysis measures are introduced and reviewed.

Keywords- Behavioural Authentication; Biometric Authentication; Identification; Access Control.

I. INTRODUCTION

A reliable authentication system is a critical component for many applications that require access control. Biometric mechanisms of authentication are becoming a popular approach to securing computer systems, networks and other critical resources. While there has been a significant surge in the use of biometric systems for user authentication in recent years, they have not been a perfect solution and there are a large number of known attacks against these systems. A few security attacks have been reported in [1-5]. Reference [5] identifies a number of different types of spoofing attacks to biometric systems:

- Coercive Impersonation- in this type of attack the attacker physically forces a genuine user to identify himself to the authentication system or removes the biometric trait (for example finger) to gain access to the resources.
- Replay Attack- is based on re-presentation of a previously recorded biometric trait, such as recording someone voice or taking a picture of a person and presenting it to a face recognition biometric system.
- Impersonation Attack- is based on a change in attacker’s appearance to match a genuine user, for example using makeup to copy somebody’s face.

Although there are a number of counterattacks against spoofing by using liveness detection methods as described in [6] or using a multi-model biometric system [7], these methods add to the complexity and cost of the biometric system and they are not always successful. For example a multi-model voice and face recognition system can be attacked by concurrent presentation of recorded voice together with a picture of user’s face.

Studying possible attacks against current biometric systems reveals a common characteristic of such systems that makes the spoofing attack possible. This common characteristic is the trait physical accessibility. This is where behavioural biometrics takes advantage over physiological biometrics. The behavioural traits of a user are not physically accessible. Examples of behavioural biometrics are keystroke dynamics [8,9], pointing device interaction [10] and game based authentication [5,11,12].

In this paper we propose a behavioural authentication system based on computer games similar to the one that has been created in [11]. Reference [11] performed a feasibility study for an authentication system that authenticates based on user interaction with a three dimensional maze. The results of this study showed that the system had an average accuracy of 88.33% in identifying different users from each other. This paper is an extension to the work that is done in [11] and describes design and implementation of three different games that can be used for authentication purposes. A number of users have been asked to play each game for a specified amount of time. During this period the system collects and audits statistics about the users’ behaviour. This process is to be repeated a few times in a one month period. At the end of the data collection, statistical methods will be used to distinguish between different user behaviours.

II. BACKGROUND

A. Recent Advances in Biometric Authentication

Biometrics refers to the recognition of a legitimate user by using physical or behavioural traits associated with that individual. Any physical or behavioural trait can be used for biometric recognition as long as it has the following characteristics [13]:

- Universality: any human should possess the trait.
- Distinctiveness: all humans should be sufficiently distinctive in terms of the measured trait.
- Permanence: the trait should remain constant throughout a period of time.
- Collectability: the trait should be measurable quantitatively.

A practical biometric authentication system should also have a satisfactory accuracy and speed, good acceptability among users and high security against attacks. Reference [13] reported that biometric systems based on fingerprints [14], face [15] and iris [16] have received the most attention in recent years. Reference [13] also reported the state of the art error rates of four popular biometric traits (TABLE I). Due to
TABLE I. FALSE REJECT AND ACCEPT RATES WITH STATE-OF-THE-ART FINGERPRINT, FACE, VOICE AND IRRIS VERIFICATION SYSTEMS [13]

<table>
<thead>
<tr>
<th>Biometric Trait</th>
<th>False Rejection Rate</th>
<th>False Acceptance Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fingerprint</td>
<td>0.1%-2.2%</td>
<td>1%-2.2%</td>
</tr>
<tr>
<td>Face</td>
<td>0.8%-1.6%</td>
<td>0.1%</td>
</tr>
<tr>
<td>Voice</td>
<td>5%-10%</td>
<td>2%-5%</td>
</tr>
<tr>
<td>Iris</td>
<td>1.1%-1.4%</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

intrinsic variability of biometric traits, it is a challenging task to maintain the performance of the biometric system while ensuring the security demands of an authentication system. Whilst the development of a comprehensive biometric mechanism is crucial as biometric systems grow rapidly into access control systems, developing such a mechanism with guaranteed security demands and recognition performance has remained an unreached objective.

B. Behavioural Authentication, an Overview

Behavioural authentication is a subset of biometric authentication which uses measurable properties of a person’s actions to identify the person. All behavioural biometric systems work in a similar manner: by analysing the current user actions create a model of individual user behaviour and then use this model to predict the future user behaviour.

Although behavioural biometrics generally introduces less accuracy levels (error rates) than the physical biometrics, they have some advantages over physical methods:

- As discussed in the introduction, behavioural methods are more resistive against spoofing attacks.
- Data gathering process is often unnoticed by the user.
- They are often less expensive and require no specialized hardware.

Most of the current research in behavioural biometrics concentrates on very low level user behaviour such as keystroke dynamics and mouse movements. Although these traits are accurate, they reveal user behaviour directly related to physical abilities of the human and ignore higher level intentional behaviours, which may describe identity of the person more successfully [17].

C. Related Work

1) Pointer device authentication

Reference [10] proposed a behavioural biometric technique based on human computer interaction. Their developed system captures data via a pointing device, and uses this data to authenticate an individual. The acquisition module of the system is a web based memory game. An equal error rate of 0.02 was reported.

2) User verification via web interaction

Reference [18] proposed the use of a biometric trait based on behaviour extracted from interaction with a web page. It is proposed to integrate this trait into a conventional login web page to enhance the security of the system. The proposed system is a multi-model authentication system that uses both biometrics and memorable traits. This work suggested that the users are more cooperative in data collection if it is done in game environment than a non game environment. The results of testing the system on a population of 50 users were equal error rates of 6.2%-12.5%.

3) Strategy-based behavioral biometrics

Reference [12] showed that a behavioural biometric signature can be generated based on the strategy used by an individual to play a game. They implemented a software to extract behavioural profiles for each player in the game of poker. The generated behavioural signature is continuously compared with players’ current actions and significant changes in behaviour are reported as security breaches. Yampolskiy reported equal error rates of as low as 7% for the behavioural profiles enhanced with temporal and spatial information [19].

D. Three Dimentional Authentication

Reference [11] proposed using a three dimensional behavioural authentication system. The project involved implementing a three dimensional maze game that records user interactions as users navigate through the game. The project included testing of the proposed system on a small group of users and analyzing the collected signatures by means of statistical analysis methods. The results of conducting the tests showed an average true rejection rate of 88.33% with an average false acceptance rate of 11.67%. Reference [11] suggested that the developed system has the potential to be introduced as a powerful authentication system for very high security demands. However the authentication system was only tested on 5 users enabling multiple analysis techniques to be assessed. For this reason only a small amount of data were collected to test the method proposed. Another drawback of the work was the unattractiveness of the game and that it occasionally caused headache and dizziness for some users.

In the current paper it is proposed to extend the game authentication method developed by [11]. Three different games has been designed and implemented. Then an average of 50 users have been asked to play the games for four periods of 4 minutes during a one month period. This paper focuses on the design of the games and presents some user feedback about the games.

III. METHODOLOGY

A. Games

Two parameters are considered in games design. First was that the games intended to be easy to play and graphically interesting for users. Failing to meet this requirement may decrease the acceptability of the system among users. To meet this requirement Blender 3D game engine is used to implement the games. Some open games are chosen as a base for two of the games aiming to provide a state of the art game experience to the users.

Second parameter was that the games should provide users with easy decisions in the course of the game. This is against a
strategy based game which needs complicated decisions to be made inside the game. The difference is between two approaches: pointless game and target based game. In a target based game the users should follow a logical method to finish the game and win or lose based on the quality of their decisions. Examples are chess, poker and other strategy based games. In this type of games since the decisions are logical, behaviour of the user can be predicted based on the current situation in the game. However the degree of predictability differs among different strategy based games. For example a chess game behaviour is more predictable than a poker game. Reference [17] stated this difference and created a behavioural biometric using a poker game [12]. However he reported that a spoofing attack is possible against his developed system by secretly and automatically monitor the target user during the play in an online casino.

In this paper the idea is brought further and the games designed to be pointless. The authors believe that the pointless nature of the game can reveal more accurately the behavioural difference of a human and increase the resistance of the system against spoofing attacks.

1) Maze game

The maze game consists of a large series of corridors that connected together and created repeated random shapes. The user will not have any idea during the game of the start and the end point of the maze. The user controls the movement of the game character; a squirrel (Fig. 1(a)). The character has the abilities to walk forward and backward, turn right and left, jump, double jump, rotate its tale and run. The user has complete control over the mentioned activates.

2) Car game

The car game uses the same map as the maze game, but this time instead of corridors the map is used to create roads. Again the game environment is large enough for the user to navigate through. A third person camera view is used from the back of the car. The user can drive the car through the provided controls (accelerate, brake, steer left, steer right) (Fig. 1(b)).

3) Subracer game

The subracer game is a submarine game that can be played in an underwater environment. The submarine can be driven in different directions as well as up and down in a large circular path. This game provides more controlling options to the users than the first two games (Fig. 1(c)). Subracer is slightly breaking the rule of pointless game as there are some collectable boxes around the game environment. However the users are not aware of existence of the boxes until they play the game. These objects are included in to the game to compare with two first games which are completely aimless. The environment also includes some rocks, mines, plants and other different objects.

Fig. 1. Snapshot of the games: (a) maze game. (b) car game. (c) subracer game.
B. Data Acquisition

A unique data acquisition module is written for all three games. This module collects user behaviour throughout the game. This behaviour includes the keys pressed, exact time of key press and the location of game character (animal, car or submarine) at the time of key press.

50 users (engineering students) have been asked to play all the games one time a week for a period of one month. Each time the games allow the user to play for a period of 4 minutes. At the end of data collection period it is expected to have a minimum of 200 samples of data for each game.

C. Users’ Feedback

Some users stated that their emotions in different situations may change their behaviour. This could affect the false rejection rates adversely. However this could also make the system more secure against Coercive Impersonation attack. When an attacker try to force a genuine user to authenticate, the change in user’s emotions may affect his behaviour significantly and he may not be able to log in to the system. For other situations when the user is naturally in a different emotional mood, a longer registration period can help to improve false rejection rates.

Many users complained about the time consuming process of the authentication. They argued that this could be a very tedious and boring process. Some suggested that defining goals in the games could improve them. This was the case for a few who found subracer game more interesting for not being totally pointless.

Some users liked the non-requirement for memorable or physical tokens, and other users found the environments user friendly. A few users noted the privacy concerns of such systems. They argued that these systems would not be acceptable until the attitude of the society as a whole towards privacy changes.

IV. DATA ANALYSIS

When a new biometric data sample is presented to an authentication system, the system should be able to measure how similar the new sample is to template data, gathered at registration stage [19]. A similarity measure should consider the statistical characteristics of the data distribution assuming that enough data is available to determine these properties [20]. A number of similarity measures have been used in behavioural biometrics. Some are introduced in this section.

A. Maximum and Minimum Similarity Measure

Reference [11] used a simple algorithm to measure the similarity of a given vector \( x = (x_1, x_2, x_3, \ldots, x_n) \) to a group of vectors with the maximum vector \( M = (M_1, M_2, M_3, \ldots, M_n) \) and the minimum vector \( m = (m_1, m_2, m_3, \ldots, m_n) \). The similarity score of vector \( x \) to the group of vectors is:

\[
S = \sum_{i=1}^{N} |x_i - M_i|, \quad m_i < x_i < M_i.
\]  

The similarity score \( S \), has a maximum value of \( N \). Hence the distance of the vector \( x \) to the group of vectors can be calculated as:

\[
D = N - S.
\]  

B. Distance of Two Samples Using Degree of Disorder of an Array Method

Reference [8] used the concept of degree of disorder of an array to calculate the distance between two samples. Given an array \( V \) of \( N \) elements, the degree of disorder of \( V \) with respect to a second array \( V’ \) can be calculated as the sum of the distances between the position of each element in \( V \), with the same element in \( V’ \) [8]. The arrays elements should be identical, and they should differ only in position of the elements. As an example, assume that array \( A = \{R, T, Y, U, I\} \) and \( B =\{I, R, T, U, Y\} \). The degree of disorder of \( A \) and \( B \) is: \((1 + 1 + 2 + 0 + 4) = 6\) (Fig. 2).

C. Euclidean Distance

Euclidean distance is one of the most popular distance functions. Euclidean distance can be calculated as the sum of the squared distances between the elements of the \( n \)-dimensional vectors \((x, y)\) [12]:

\[
d_E = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3}
\]

D. Mahalanobis Distance

The Mahalanobis distance of a multivariate vector \( x = (x_1, x_2, x_3, \ldots, x_n)^T \), from a group of values with mean \( \mu = (\mu_1, \mu_2, \mu_3, \ldots, \mu_n)^T \) and covariance matrix \( S \) is defined as [12]:

\[
D_M(x) = \sqrt{(x - \mu)^T S^{-1} (x - \mu)} \tag{4}
\]

Mahalanobis distance is widely used in classification techniques and is closely related to Hotelling’s T-square distribution used for multivariate statistical testing and other analysis methods [21].

E. Manhattan Distance

The Manhattan distance between two points is defined in a Euclidean space with fixed Cartesian coordinate system, and is the sum of lengths of the projections of the line segment between the points on to the coordinate axes. It can be considered as the absolute differences of the elements of two vectors [12].
V. CONCLUSION

In this paper current physical and behavioural authentication methods were reviewed. Advantages of behavioural methods in comparison to physical methods were argued. A game based behavioural authentication system was proposed. Pointless and target base types of games were discussed and it has been shown that pointless games have more capacity to collect behavioural traits of users and can be more secure than strategy based games. Finally some common similarity measures that can be used in a behavioural authentication system have been reviewed.

REFERENCES

Dynamic Interactive Storytelling for Computer Games Using AI Techniques

Simon Cooper 1, Dr Abdennour El Rhalibi 1, Prof. Madjid Merabti 1
1School of Computing & Mathematical Sciences, Liverpool John Moores University, Byrom Street, Liverpool, UK, L3 3AF
s.cooper@2003.ljmu.ac.uk
{a.elrhalibi, m.merabti}@ljmu.ac.uk

Abstract—In this paper we introduce DISE “The Digital Interactive Storytelling Engine”. Using software engineering methodology and framework development methods, we aim to create a full interactive storytelling system involving a story manager, a character engine, an action engine, a planner, a 3D game engine and a storyworld editor. The framework and the 3D game engine are described in detail and specified to meet the requirement of bringing a more dynamic interactive story experience to the medium of computer games. Its core concepts borrow from work done in the fields of computer science, software engineering, computer games technology and artificial intelligence.

Index Terms—Artificial intelligence, Games

I. INTRODUCTION

Similar to other entertainment media, stories in games play a big role in increasing immersion, building tension and maintaining a player’s interest. One main difference between games and other media is that games are interactive; they expect participation from the player and in turn, players expect to participate in the events that the game is presenting and their outcomes.

Stories can be presented in different ways: they can be linear, branching, parallel, or threaded. Games typically follow a linear storyline, where the events of the story are presented in a predefined sequence. It can be argued that making a player follow a defined story can diminish the interactivity level of a game; the player is, after all, following a preset path already laid out for him by the author. In order to still convey a story and allow the player to feel a high degree of interactivity, the concept of interactive or non linear storytelling has to be introduced. Simply put, interactive storytelling presents the opportunity for players to have an input on what is happening in the game world they are placed in, to be the ones who dictate how certain events may come to pass.

II. LITERATURE REVIEW

This section contains a literature review that critically analyses peer-review publications and other scholarly output related to interactive storytelling and artificial intelligence, mapping the research area and identifying the techniques and area of novelty that the project will address.

A. Games Master

One of the earliest games to use interactive storytelling was in fact a tabletop game Dungeons & Dragons, which was released in 1974 [1]. A player is appointed the role of games master (GM), whose job is to direct the players through the dungeon, whilst reacting to their actions, using the original story as a guide and filling in the blanks using their imagination and reasoning for a more entertaining experience. In computer games the computers rule systems, which have to decide how to respond to player actions, replace the GM. The rules aren’t capable of responding to any action in the same way a human can, so the freedom in the number of actions that can be performed at a given time is significantly less [1].

B. Directors

To create more interactive storytelling we need a larger entity controlling and adapting the story in accordance to the players’ actions. Champagnat, Estraillier and Prigent suggest a narrative framework to simulate this activity. In this framework, they suggest having a set of agents in charge of evaluating the current state of the game, and one director agent, which “chooses a set of relevant actions to be executed” [2]. An AI director is used in Valve’s survival horror game Left 4 Dead, which monitors the player’s behaviour and input to control the pacing of the zombie attack and judge which events to trigger next [3].

C. Goal Net & Fuzzy Cognitive Maps

Another technique generating interactive storytelling is described by Cai, et al. [4]. In this approach, a tool based on Goal Net is used to plan the story and Fuzzy Cognitive Maps are employed to analyse the user inputs and decide which path should be followed; they were selected due the fact that they act as “collection of the rules such that it not only concerns the relationships between the causes and effects, but also considers their relationships among the causes” [4]. Goal Net’s ability to break goals down into simpler states make it useful for interactive storytelling [4][5].
Three main advantages can be distinguished from employing this method to create interactive storytelling [4][5]: All events are simplified into less complex scenes; The engine can react not only to the user inputs but also to the actual state of the game, and look for the best path; Since all the events are loaded into the engine in real time, its performance is increased.

D. Goal-Inference & Planning

Ciarlini, M, et al. created a tool called LOGTELL to interactively generate and dramatize stories. [6] Their system uses goal-inference rules, planning and user intervention to drive a 3D drama visualisation that can be used for entertainment or story creation within a specified genre. Its main focus is on generating a large variety of coherent stories, rather than “creating an immersive experience in which the user takes part in the story as one of the characters”.

E. Hierarchical Task Network (HTN) Planning

Cavazza, Charles and Mead use a hierarchical task network (HTN) planning for interactive storytelling “as it is generally considered appropriate for knowledge-rich domains” [7]. The game characters roles in the story can be represented as tasks, with a single HTN corresponding to several possible decompositions for the main task and the story being composed of situations that are the “cross-product” of the actors’ roles [7]. The HTN is traversed executing primitive actions along the way and allowing backtracking if they fail. The fitting aspect of HTN planning is that it is based on forward searching “while being goal directed at the same time” [7], with the top task being a characters main goal/objective (e.g. a cartoon villain would want to take over the world, but their sub goals would be to get a doomsday device and secret base in order to achieve this main objective).

F. Dilemma Based

Barber and Kudenko developed an interactive narrative system called Generator of Adaptive Dilemma-based Interactive Narratives (GADIN) [8]. Using GADIN they created a finite children’ story about a dinosaur’s adventure designed to end when a dynamically created goal is met, thus providing a variable satisfying (clear) ending regardless of choices made. The story designer has minimum input and just needs to provide character information and other world related knowledge. “The system then instantiates all generic knowledge on story actions and dilemmas and thus creates the narrative in collaboration with the user’s actions and dilemma decisions.” [8]

G. Case Based Reasoning

Similar Fairclough, R. & Cunningham, P [9] investigated a multiplayer story engine that uses case based reasoning/planning to create a story adventure MMO game. “The system handles multiple users in a game world and directs the non player characters therein to perform for the users parallel storylines, interweaving character roles in each story. The story is told through a ‘narrative of actions’ and automatically generated dialogue.” Similar to the examples in: 2.2 Directors (above), an omnipotent director agent is used to control the story. By including a multiplayer aspect their director must be more capable and “assign story goals that are relevant to each player”.

H. Genetic Algorithms

Ong, TeongJoo and Leggett, J. use this approach in their work to generate interactive narrative [10]. Their engine, the Hybrid Evolutionary-Fuzzy Time-based Interactive (HEFTI) storytelling engine uses genetic algorithms to recombine and evaluate story components generated from a set of story templates.

I. Storytron

The Storytron platform by Crawford & Mixon [11] is one of the most complete examples of interactive storytelling. It is character based and uses a verb based action system. “It is centred on artistic works called storyworlds. Each storyworld is a universe of possible narratives. The technology is comprised of four parts: SWAT, the Storyteller, Deikto, Sappho, and the Storyengine”. SWAT is used to create new storywords, the Storyteller is the software used to play storyworlds, Deikto is the English-like language used by players to create the sentences that drive their actions, Sappho is the scripting language the author uses to create a storyworld and the Storyengine drives the computer characters AI. Storytron’s main novelties are the use of verbs and objects in the construction of action sentences. Storytron represents characters emotions with cartoon face emotion icons called ‘Emoticubes’. These are driven by the characters mood taken from an array of data in their personality model.

J. A Behaviour Language (ABL)

Mateas and Stern created an experiment in interactive drama called Façade [12], where the player has to interact with two NPC’s that are having marriage problems. Façade’s novel features include: The player’s actions have a significant influence on what events occur, which are left out, and how the drama ends; Drama manager monitors the simulation and adds and retracts procedures (behaviours) and discourse contexts by which Grace and Trip (the NPC’s) operate; Plot is made up of dynamically sequenced story beats; The multi-agent co-ordination is sequenced by these beats; Supports joint behaviours.

III. DIGITAL INTERACTIVE STORYTELLING ENGINE FRAMEWORK

Our goal is to specify, design, implement and evaluate an interactive storytelling engine, with an accessible story model structure and editor for easier creation of new story content. Our engine called DISE (Digital Interactive Storytelling Engine) is made up from the following components as illustrated in Fig. J. The following chapters will expand on the five main areas of the DISE framework and toolset.
A. Action Engine

The action engine will provide the interface for the player to interact with the story’s world. It provides a traditional first-person view and allows the player to move with the computer keyboard and look around with the mouse. To carry out an action the player must first left-click on something of interest. After considering a text based approach to choosing actions, we decided an object based selection method would be more familiar to users and provide a faster and freer flowing experience. When the mouse reticule is moved over an object that is interactive it will highlight and spin giving instant feedback of the items interactive nature. This interface is tried and tested and was very widely used in popular classic point and click adventure games such as LucasArts’ The Secret of Monkey Island [13]. Our interface also differs from this slightly; when the player clicks the left mouse button the object is selected and a circular menu will be displayed showing which actions are possible for that object (Fig. 2).

B. Character Engine

The character engine’s role will be to maintain and update the goals, actions and behaviour of each computer controlled non-player-character (NPC). For a character driven story to emerge each character has to behave differently to the others and make certain choices that progress the story in an interesting way as an individual. The character engine we propose is loosely based on the guidelines Crawford explains in his book [14], combined with the Games Master concept from classic tabletop RPGs[1][15], Ciarlini, et al.’s use of partial order planning, which allows simultaneous actions [6] and the work done on Character-based Storytelling by Fairclough & Cunningham [9] and Pizzi, et al. [16].

Characters will be controlled by the following states:

- Wandering Mode – No Goals or Actions Pending, so set to default wondering pattern as set by story author, e.g. random, patrol your own house, follow player 1, etc.
- Goal List – queue up goals to carry out and plan which actions are needed to reach them and their order & priority. When an action is chosen it is sent to the action list.
- Action List – queue up actions to carry out, if existential preconditions are met execute actions in order of priority, otherwise send to execution list.

in a fairytale story for children the actions shoot/kill may not be appropriate and therefore can be omitted entirely from every scene. All verbs are stored in the verb dictionary; these will be narrowed down by cross referencing them with the verbs available in the current scene. The scenes verbs will then be loaded into a planning system, which will check the defined preconditions for each verb. Finally the verbs that have their preconditions returned as true will be displayed in the action menu and are available for selection by the player (Fig. 3).

This should stop the occurrences where players select an action and an object and are shown a message that their chosen combination is not valid.

To interact with the story a player needs to be able to describe the actions that they wish to carry out. Players are given a choice of words that link together to form coherent sentences. There are a limited number of words, which are made available depending on the particular scene and context of the story at a given time.

The story developer can choose these verbs/actions by listing them in the scene verb xml file, which allows them to control to some extent the direction of the story. For example

![Fig. 1. DISE Architecture](image1.png)

![Fig. 2. Prototype of the in-game Action Menu](image2.png)

![Fig. 3. Basic verb/action context filter](image3.png)
• Execution List – store actions that can’t be carried out yet here. If highest priority action then carryout action plan and wait until all conditions are met.

The actions a character should execute are chosen by the engine in two different ways:

• Story Manager Goal – the story manager processes a scene that feeds a predetermined goal to a character which gets passed to the action list.

• Reaction Goal – the player or another character interacts with this character and drives a reaction goal which is chosen depending on this character’s role during the event and the outcome produced when its personality model variables are substituted into the choice formulae for that actions role. When either of the above generates a goal it is added to the goal list in a prioritised sequence.

C. Story Manager

The story manager sits on top of the game engine and dynamically updates the state of the game world see Fig. 4.

The story manager completes three main goals:

• Monitor – the story’s overall state needs to be monitored to make sure it always stays interesting and relevant. Some summary variables can give an impression of what has been happening and the general mood.

• Process – after monitoring the story, the data gathered must be processed to decide what needs to happen next.

• Change – the game world then has to be changed according to the processing results. This change will progress the story in the desired way.

The story manager will process data differently depending on the story module plug-in created by the developer, which includes the specific criteria that has to be met to change the story’s path to follow their desired outline.

D. Planning Algorithms for Story Actions

To correctly sequence our story actions we will use a branch of artificial intelligence called automated planning and scheduling. Planning is used widely in computer science for problem solving, timetabling and controlling intelligent agents. The player knows to open a locked door they need to pick up a key first, but a more complex system is needed to get a non-player character to carry out this sequence of instructions thus reaching a goal state. Planning languages have recently been standardised in 1998 to a single description language called PDDL (Planning Domain Definition Language) [17]. PDDL is made up of smaller optional modules, but not all planners implement the whole set. The main subset language of PDDL is called STRIPS (Stanford Research Institute Problem Solver) after the automated planner that used the language as its input.

Barros & Musse [18] consider the following five extra language-constructs valuable to interactive storytelling applications:

• Type hierarchies – each object and parameter is given a type classification to avoid errors.

• Equality – uses the ‘=’ sign to mean equals, which is useful for conditional effects.

• Negative preconditions – allows the ‘not’ command to be used with conditionals and equality, for example to describe actions for when a character does not know a fact.

• Conditional effects – allows the use of conditional ‘if’ statements.

• Existential preconditions – “Using an existential quantifier in the precondition, we are able to state that the character can only perform this action if he is alone in the examined place”.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>TH</th>
<th>DO</th>
<th>NF</th>
<th>CE</th>
<th>LT</th>
<th>OPA</th>
<th>POP</th>
<th>AC</th>
<th>NE</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0.03 s</td>
</tr>
<tr>
<td>Graphplan</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0.118 s</td>
</tr>
<tr>
<td>DP</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0.001 s</td>
</tr>
<tr>
<td>Heur</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>3.601 - 7.75 s</td>
</tr>
<tr>
<td>RSVP</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>(2)</td>
<td>x</td>
<td>x</td>
<td>0.10 - 0.302 s</td>
</tr>
<tr>
<td>LPG-2D</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0.059 - 0.156 s</td>
</tr>
<tr>
<td>Mover</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0.07 s</td>
</tr>
<tr>
<td>Monte-IT</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0.079 - 0.053 s</td>
</tr>
<tr>
<td>SatPlan_A</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0.237 - 0.322 s</td>
</tr>
<tr>
<td>STAG_4</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0.00 s</td>
</tr>
<tr>
<td>TLog</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>(6)</td>
<td>x</td>
<td>x</td>
<td>0.00 s</td>
</tr>
</tbody>
</table>

Fig. 5. Benchmarking Planning Algorithms [18]

Also useful to interactive stories with multiple NPCs is partial order planning, which allows parallel actions or the extraction of different total order plans from one partial order plan.

A planner requires a domain definition, which lists the facts about the initial state of the world, its actions, requirements and constraints. Actions are usually made up of three parts: parameters, preconditions and effects. Another factor in planning for computer games is search optimisation. If new plans are to be generated in real-time alongside processor intensive game logic and rendering, they have to be fast to execute and highly optimised. There are now many different search techniques including: forward chaining; backward chaining; state-space search, with conditional relationships or heuristics (e.g. graphplan) and propositional satisfiability (satplan).

\[147\]
If the assumption of determinism is dropped and a probabilistic model of uncertainty is adopted, then this leads to the problem of policy generation for a Markov decision process (MDP) or (in the general case) partially observable Markov decision process (POMDP).

Barros & Musse benchmarked different planning algorithms testing their speed and feature sets with interactive storytelling applications in mind. Fig. 5 shows the main results.

E. DISEdit Story Editor

The DISE editor (DISEdit) is designed to be an easy way to create new stories and load 3D worlds into DISE. The story creator must go through several steps to create their story or to edit an existing one. The editor will be focused on enabling easy creation of the basic elements needed to build a new story world: verbs, objects/characters and scenes.

When an existing object is selected with a mouse left-click a menu will appear with the options to move or delete the chosen object. If the move command is selected the cursor will change to a hand and the object should now float in front of the camera (Fig. 6). Using the mouse to look around and the arrow keys to move, the object can now be positioned anywhere in the 3D world. After moving the object it can also be rotated to the desired direction and placed with another left-click. As some objects are affected by physics properties, they may not be able to pass through each other, when an object is dropped if it is dynamic, it may bounce around and find a natural resting place. The editor will also support new object creation, either from a list of predefined objects or allowing the user to load their own model files and editing some object properties to describe how it will behave in the story world.

Fig. 6. DISEdit Dynamic Object Manipulation.

IV. HOMURA GAMES FRAMEWORK

The story engine will need a 3D game engine underneath to provide all the base rendering, update, and control functions found in modern games. We have developed the Homura game engine which provides these essential functionalities. Homura architecture is depicted in Fig. 7. Homura is made up of three main components: Homura games framework, Homura IDE and Net Homura.

The Homura games framework provided an extensible, easy to use set of APIs and tools for the development of Java and OpenGL based 3D hardware accelerated games, which support cross-platform, cross-browser deployment using the Java Web Start (JWS). Homura is built on top of the open-source Java Monkey Engine (JME) scene graph engine which provides an efficient, flexible API for the development of 3D Java based games.

Homura IDE is a game development IDE written as an Eclipse Platform plug-in application which aims to simplify game development by providing an integrated toolkit which gives a graphical user interface to many tasks which would normally be code-based, for example positional and spatial editors, a curve editor to control animation paths, and a game template generator.

We decided on 3D Java based engine to maximise compatibility across multiple platforms including Windows, Linux & OSX. With this engine and some tools made for related projects the interactive story game can be distributed online using Net Homura and played online in most web browsers, or in an external Java Web Start window at full-screen, using the user computer’s full hardware capabilities as described in “Development of a Networking Middleware and Online-Deployment Mechanism for Java-Based Games”.

V. EVALUATING DISE

The DISE project’s main contribution is the creation of an alternative framework aimed to generate interactive storytelling, which draws inspiration and employs certain techniques and methods from existing approaches and combines them to form a system made up of a story engine, story editor, game engine and story planner among other components. This framework has the added benefit of clearly dividing all functionality and processes into independent components, which can be easily changed or modified in order to expand or improve the architecture. Finally, while the created framework accomplishes the desired goals for this projects requirements, it also has areas that can benefit from the inclusion and usage of alternative techniques and methodologies, and presents good opportunities to improve all aspects of the architecture; the inclusion of different techniques to generate plans, refinement of the 3D rendering module, optimization of the story handlers and incorporation of more complex AI to simulate emotional NPCs are some of
the areas that represent an interesting opportunity for our work and expansion in the future.

VI. CONCLUSION AND FUTURE WORK

The In this paper we have introduced DISE “The Digital Interactive Storytelling Engine”. Using framework development methods, we aim to create a full interactive storytelling system involving a story manager, a character engine, an action engine, a planner, a 3D game engine and a storyworld editor. The framework and the 3D game engine are described in detail and specified to meet the requirement of bringing a more dynamic interactive story experience to the medium of computer games. In future work we will continue the development of our DISE engine and its functionality concentrating on refining the necessary features to make a novel contribution to interactive storytelling in computer games including:

- A dynamic planning system that will run in two stages: Firstly to look ahead and filter a vast amount of verbs into a more manageable list of actions that can be carried out and fit with the player’s current context and scene; Secondly to determine the plans of future actions for each computer controlled character, with an optimised search technique to allow the game to run smoothly.
- Real-time first-person interaction with objects and characters in the story.
- A detailed 3D world featuring modern graphics techniques such as Shaders and Parallax Displacement mapping.
- The ability to distribute the game online and run in a web browser inline or in full-screen mode via a Web Start across multiple platforms.
- Physics based drag and drop editor for easier story content generation, including dropping in existing objects and new object creation including importing new 3D models in the .3ds format.
- Creation of new actions, their effects and possible reactions by computer controlled characters and also character personality and goal editing.
- Creating general story plot outlines via constraining a scenes possible actions and injecting new events & goal triggers using the scene editor.
- A set framework to build on for future interactive story development.

REFERENCES

Abstract—Network routing algorithms are naturally dynamic things. They are most easily understood if they can be seen in action. Of course in the real world it is not possible to see a routing protocol in action, so we often rely on diagrams and descriptions to understand what is going on. In this paper a simulation of Dijkstra’s shortest path algorithm has been developed in the Virtual World OpenSim [1]. Learners may define a network infrastructure and topology. They may then apply a routing protocol to nodes in the topology and to send traffic across it. Learners are able to observe the different routers building their routing tables, exchanging information between each other and the passage of data packets. Events can then be applied to the infrastructure. For example if a link is taken down, it is possible to observe how routers learn about this and the effect on traffic flows.

I. INTRODUCTION

Virtual worlds provide an intuitive 3D environment where users are represented by avatars [1][2][3][4]. This presence is engaging, users are able to, and like interacting with each other. This in turn provides natural support for group work and collaboration. The environment is programmable. Thus it is able to support a rich set of interactions, which allows it to be used in a variety of different ways. It is a multimedia platform; Video, sound, pictures animations can all be supported [5]. Thus a rich set of educational resources can be brought together and accessed through a 3D virtual world browser [6]. They can be navigated by avatars moving around a 3D space, much as a group of students might explore an art gallery or a museum that they had exclusive access to.

Content within 3D worlds like Second Life is created by residents. This extends beyond the ability to create simple objects [5], to being able to shape terrain, construct buildings and design clothes. Through the programming language LSL [7] users are able to program their world making it a truly interactive experience. Buildings may speak to you, vehicles may transport you around and you can buy things in automated shopping arcades.

The goals of this work are to situate an interactive simulation of routing algorithms within a 3D world. This learning environment would provide support for a collaborative exploratory approach to learning about routing. The virtual world acts as a 3D portal which provides access to video of lectures, sound clips, animations, presentations, lecture notes, research documents and standards. At the centre of this learning experience is a 3D interactive routing simulation. This allows learners to construct network topologies, and observe how network routing algorithms work in determining the shortest paths and routing tables. The flow of packets across the network can be observed. Further the effect of topological changes can be experimented with.

In achieving this two challenges had to be met

1) The programming environment provided by SecondLife, protects the system from the programmer, limiting the resources that a program can use. For example introducing delays after creating an object or communicating off world.

2) The software development support is very limited. Making it time consuming and difficult to develop programs that are either distributed or more than few dozen lines of code. Thus the development of a network simulator in world would not be practicable.

These challenges can be resolved by using the 3D virtual world OpenSim. As opensim is open source it is possible for the programmer to gain access to its inner workings. This can be achieved by utilising Mini Region Modules MRMs. An Api which allows the development of complex external software, allows the software to be connected to OpenSim so that a user may be provided an in world controller interface and may be given a 3D view of the system model.

II. MOTIVATION AND GOALS

Routing algorithms are complex things to understand. They involve layered communications between multiple entities and can present a challenge to visualise. Virtual worlds provide a platform in which multiple users can simultaneously interact with ‘physical’ items in a manner that is intuitive and responsive. Users in a virtual world have a presence and can
communicate with each other; changes made by one user will be instantly visible to all other users. This project aims to use the unique strengths of virtual worlds to create a learning environment that allows the teaching of routing algorithms. Whilst other network visualisation and simulation tools exist this project is unique in its leveraging of virtual worlds as a platform. By keeping the complexity of the simulation to a minimum and focusing on one specific aspect of network theory, routing, the system provides a simple and intuitive platform for students to experiment with network routing theory and see how changes to network topology affect routing algorithms. A simulation in action is shown in Fig. 2.

III. SECOND LIFE AND OPENSIM

One of the things that makes virtual worlds different from networked games is their programmability. Not only can users change the shape of the world around them and add new geometry and structures into the world they can also attach code to the objects in world. The fact that scripting behaviour of objects in world is an integral concept to virtual worlds is what makes this project different.

This innate programmability of the platform separates this project from, on the one hand, programmed simulators that use graphics libraries to visualise what they are simulating and on the other hand animated diagrams generated using imaging software which walk the viewer through an algorithm or concept. By approaching the teaching of an algorithm such as Dijkstra’s algorithm from a virtual world perspective rather than an animation or pure simulation perspective the fundamental concept of users having direct control over their environment is harnessed to give users control over the simulation.

One of the major challenges that this project faced was the level of programmability that specific virtual worlds provide for the user. Enabling users to write and execute code in a distributed environment is a dangerous proposition. As soon as the user is given the power to execute their own code within the larger framework of the distributed application they are given the ability to break the application.

When deciding how to face this challenge developers of virtual worlds are left with a choice. On the one hand they can give the user the power to create complex and involved systems within the environment they have created. This makes for great possibilities but also leaves the system vulnerable to exploitation. Alternatively the developers can limit users to develop within very controlled parameters which limit what the user can do but also limit their power to harm other user’s experience.

The virtual world Second Life is an example of the latter approach. As a commercially run venture where users may be paying for content the developers are obliged to limit the damage any single user can do by exploiting the system. As such the tools they give the user must work within very confined parameters which can never go as far as to break the system. This is a system that works, tens if thousands of users are actively creating content within second life. However for a project with the scope and ambition of this one such a world is too limiting.

The other end of the spectrum is OpenSim. OpenSim is an open source virtual worlds platform. OpenSim is essentially a virtual world server. The server it runs is highly configurable and multiple instances can be connected together to create a virtual world that scales. Users are free to create their own server, configure it however they wish and have users connect to it. OpenSim does not provide a client, it works by supporting the network protocols used by second life so that users can use the second life viewer to connect to OpenSim servers. The protocol it uses is available so other teams have developed alternative viewing clients.

OpenSim allows a local user to have administration privileges consequently choices over how the server is configured for security can be made. If the administrator wants to empower a programmer to access system components they may do so. OpenSim supports the scripting language LSL which is built into Second Life but it also supports other programming mechanisms. Being open source direct access to the source code may be allowed by the administrator. This could be directly changing the source code that exists already or it could be by adding in features in a modular manner.

OpenSim provides four modes for programming content.
1) Linden Scripting Language: allows users to write scripts in world. On one level this is powerful, in that it allows in world objects to be programmed to react to events in a rich and varied way. The creation of complex systems is difficult and time consuming. Access to system resources is controlled making the creation of moderately time sensitive applications impractical.

2) Mini Region Modules (MRM) Scripting: allows users with sufficient privileges to write scripts in world. This is more powerful than LSL as a structured API is provided which allows access to all system resources. It is however difficult to program a complex system as in world software development tools are limited. MRMs are programmed in C#.

3) Mini Region Modules Loader: These are similar to MRM scripting. However complex system support is provided, in that modules may be developed out of world and loaded into the system.

4) Region Modules: a more heavy weight approach to system development. The programmer has open access to OpenSim internals. The problem here is that the absence of a structured API means that changes in OpenSim are likely to cause Region Module to break.

The approach taken in this project was first to explore the limits of programming in LSL. When this was found to be wanting development using a mixture of MRM scripting and MRM Loader was used.

IV. VISUALISING DIJKSTRA’S ALGORITHM

The algorithm visualised by the system is Dijkstra’s Shortest Path Algorithm [8] which is used widely in Internet routing protocols such as OSPF. This is a standard graph theory algorithm for finding the shortest path between nodes on a weighted, undirected, graph. In routing it is used to build routing tables. Every time the network changes the nodes directly affected by the change send out packets informing all their neighbours of the change. These packets are passed on to the neighbour’s neighbours etc. etc. until the entire network has been notified of the change. Each node stores an internal representation of the network topology. Whenever a node is notified of a change it updates this inner model and runs Dijkstra’s algorithm over the updated model. The results of Dijkstra’s algorithm are used to build a routing table mapping every node in the system to one of the links directly connected to the node running the algorithm.

The steps for Dijkstra’s algorithm are as follows:

- Initialise a list of confirmed nodes
- Initialise a list of tentative nodes
- Add the root node to this list
- Initialise node except root to infinity
- Initialise the root node with a dist of 0
- Define a current node
- While there are nodes in the tentative list
  - Remove smallest dist node from list
  - set it as the current node
  - Add the current node to the confirmed list
  - For each neighbour of current NOT in confirmed list
    - if neighbour distance > current dist + link weight
      - store current as prev node for neighbour
      - neighbour’s dist = current dist + link weight
    - if neighbour is not in tentative list
      - add neighbour to tentative list

To represent the update process by which all nodes are notified of a new link being created the packets which are flooded containing this information are visualised moving through the system. When the user requests to visualise OSPF uses a close derivative.
Dijkstra’s algorithm a mini version of the section of the network connected to the root node is created above that node. On this mini model the current node in the algorithm is highlighted white and all nodes in the tentative list glow their normal colours. As neighbours are probed the link between the current node and the neighbour being probed is highlighted blue. As nodes are added to the confirmed list the connection between them and the previous node on the pathway back to root is highlighted red. When the algorithm finishes the red links represent the shortest paths. When the algorithm finishes the routing table for the router that ran the algorithm is printed to chat text and the links out of the root node are highlighted the colour of the node they route to one after another.

V. USER INTERACTIONS

The interface designed for this system aims to make use of the strengths of virtual worlds. The easiest interactions within a virtual world is to touch an object and to chat. These are the foundations of the user interface. The system is made up of 5 types of entities listed below.

End Points. Represented as cylinders - Routers. Represented as nodes - Packets. Represented as thin lines between nodes - Links. Visually travel along links, represented as small, cylinders - Inputs. The user clicks these to set state for how interactions with the other objects will be interpreted.

The interface is essentially a state machine. Clicking an object sets system state. System state determines the result of an interaction. The system has four basic states: idle, link selected, EP selected, router selected. The user enters each of these states by clicking an EP, link or router. Once one of these basic states has been entered the user can either click another object, click on an input or chat. What each input does is listed below. What they look like is in screenshot above.

1) Set Colour
2) Toggle visualisation on the main network or on a mini graph representing a node’s inner state
3) Toggle between sending packets 1 by 1 or sending 10 at a time
4) Run Dijkstra’s Algorithm (if an EP is selected choose another node to run the algorithm for shortest path EP to Node)
5) Pause / Play
6) Create Router
7) Create End Point
8) Delete

How a chat event is interpreted depends on current state. If a link is selected set link weight. Otherwise if a Node is selected set node name.

VI. ARCHITECTURE

The architecture of the system runs on the “model, view controller” pattern. Each layer in the pattern is separated from each other via a framework abstraction. The framework provides a series of interfaces which define the layers in the system and the components of the system (nodes, links and packets). The controller layer listens for touch or chat events, interprets them and then relays them into the system via the initialization layer interface. The initialisation layer turns events (add node, run algorithm) into objects and calls within the framework. e.g. For add node it would create a node object and pass it to the view and model layers to update their models. An abstraction to simplify the work of the controller layer. The model layer is where the actual algorithms work. This layer stores information relevant to routing and contains the logic of the algorithm. The view layer updates what the user sees visually. The Model and View layers are split into two separate interfaces. One set models the network stack. The network layer interface on the model layer receives a request to route a packet between two nodes and breaks it down into a series of hops. The view layer provides a physical layer interface which allows the model layer to request a packet be sent across a specific ‘physical’ link. Config handles adding/removing nodes. The internal implementations of layers are non-blocking. If a call is made to the simulation layer to run an algorithm the layer does not block further input until the algorithm completed. Similarly the view layer is able to receive multiple calls requesting that packets be sent in the time it takes to transport one packet. Thus many packets can be moving across the visualisation simultaneously. A further enhancement to the architecture is to de-couple the way the system starts so that assembles containing implementations of the framework interfaces are found and loaded at run-time using reflection. This would allow users to write new code and drop it into a directory to run it.

VII. DESIGN

A. The Controller Layer

There are multiple controllers, each of which function differently. The core behaviour of the controller layer is to track which in world objects exist, what type they are and to trigger calls to the command methods of the initialisation layer. There is an abstract controller layer which is extended by all different implementations which provides this functionality. Some of it is overridden in alternate versions. The controller layer tracks all objects that have been created, storing them in lists. How they are created is decided by the various implementations. These lists are used in a thread which wakes periodically to see if anything has changed in any of the objects. If an object has been moved or deleted by user input in world this thread will catch that and notify the system of the change via the initialisation layer. This controller class also maintains a thread which is a consumer for an event queue. The event queue specifies what the event is (create node, create link, remove node, send packet, calculate routing etc.) and whether the system should wait before triggering the next event.
B. The Initialisation Layer

This layer is essentially a book-keeping layer. It performs no purpose that the user is directly aware of but is necessary to make the parts fit together. This layer knows how to initialise the interfaces which define the logical system and as such keeps this complexity out of the controller layer. The controller layer is designed to understand what it wants to do and what the user is asking it to do but not how that is done. This separation is a layer of abstraction to help programmers wishing to develop alternative control mechanisms and to generally minimise the amount of boilerplate code that has to be written in the controller layer. The initialisation layer assumes that the controller layer will have an object representing each of the nodes and links active in the system. The initialisation layer is not concerned with what class this object is or how it functions in its role as part of view or controller, it merely needs to know that such an object exists. The initialisation layer maps these unspecified object instances to instantiations of the INode or ILink interfaces. This means the controller layer just has to pass in the object which it uses to represent a component (be it an MRM object, an OpenGL Scenegraph object or a windows form control) and the initialisation layer will translate that into an instance of an interface which the rest of the system can understand. This translation process will also allow the initialisation layer to associate a unique key with the object as described above. The controller layer has no knowledge of the key system.

The initialisation layer has its own implementations of the INode and ILink interfaces. These implementations have little information specific to the layer, they are mainly just implementations of the core features which are making a series of properties available. The only piece of extra information stored at this layer is the object which the controller uses to identify the component.

C. The Simulation Layer

The implementation of simulation layer interface provided consists of more than one assembly. The basic simulation assembly handles the generic functionality common to all algorithms that might be run over it. This basic functionality is then extended by other libraries which can directly extend the basic simulation classes or create their own implementations. The simplest way being to extend what is already there. The basic simulation layer will load in these assemblies at runtime in order to be able to run different algorithms.

The basic simulation assembly: The system is designed to model distributed algorithms running over network graphs. As such the work of the algorithms is done in the nodes themselves, each node does not necessarily know the structure of the graph as a whole. The first thing the simulation layer assembly provides is the class implementing the simulation layer interface which acts as the go between the system as a whole and the individual nodes which are running the algorithms. This provides the general key to object mapping found in all layers. This layer also dynamically loads in algorithm implementations and uses them to create instances of the nodes themselves. These nodes all conform to a common interface which provides a receive method to trigger the algorithm to run. The second thing the library provides is abstract implementations of node and packet interfaces specific to the simulation layer. These provide the basic functionality that is independent of the routing algorithm. For nodes this means maintaining two mappings. First a link’s mapping which maps the nodes any given node is connected to the link which connects it with that node. The second is a routing table map which maps every node in the system that the node knows about to the link along which packets for that node should be routed. Mappings are present in the routing table even if the node is not connected to the node, there is no guarantee that every node in the system will be in the routing table for every node. A node has to have every link which connects it to the links mapping, algorithms make this assumption when running and therefore may perform unexpectedly if it is not held to. As well as maintaining a table of nodes and a table of links the nodes have some default behaviour. This is behaviour that may be used by any algorithm and also which may be used independently of the routing algorithm. The default behaviours are specified by packet types. Just as the simulation layer provides an abstract implementation of a node so too does it provide an abstract implementation of a packet. This packet stores generic information such as source and destination and also information about what type of packet it is. What type of packet arrives at a node defines what the node does with it. If a packet which is part of a routing algorithm arrives it is sent to the receive method of the node implementation for that routing algorithm. If the packet is a packet for one of the default behaviours then the packet is dealt with by the default implementation of the node. The default behaviours supported by the layer are resetting, highlighting a path between two nodes, highlighting all routes back to a specific node, highlighting the routing table for the node and resetting all nodes in the network to their default states.

Algorithm assemblies: Any assembly which wants to implement an algorithm to be visualised using the system must contain implementations of the simulation assemblies node and packet interfaces and a class implementing the IAlgorithm interface. The IAlgorithm interface is essentially a factory interface. The only method that IAlgorithm specifies is the create node method. This method is used by the basic simulation assembly to create nodes which implement the algorithm itself. As well as providing an algorithm class of instantiating nodes an algorithm assembly must also define a node class to define the behaviour of the nodes as they run the algorithm and a packet class defining the packets which are sent between nodes as part of the algorithm. The implementation of the node’s receive method is what defines the behaviour of the algorithm. What a node does when it receives a packet and
what data that packet contains is essentially what constitutes an algorithm.

D. Visualisation Assembly

This is where the actual visualisation happens. This layer must place nodes and links in the world. Calculate where to place and how to shape links so they go directly between the nodes they are linking, visualise packets moving and provide access for other layers to set the colours of the nodes and the links.

The node positioning is fairly basic. An interface is defined called IPosition which defines an X, Y and Z coordinate. These IPosition objects are used by other layers to denote a position without having to know about Opensim's method of specifying position. The Opensim implementation of a coordinate is a vector object. The positions taken from the IPosition instantiation defining where a node is to be placed is translated into a vector so that the IPosition is taken as relative to the object in which the script is running. This is then placed in world using the MRM IObject interface which has a method for specifying global position.

Calculating where a link is to be placed is also fairly trivial. A primitive shape in world has a position which is defined as the centre of the object. Primitives can be one of three types, sphere, cube or cylinder. Links are cylinders positioned directly between the two nodes they are to link. They are then rotated so that the flat ends of the cylinder face towards the nodes they are linking. Once positioned the links are scaled along their z axis, stretching them so they appear as a pipe, stretched between the two nodes they link in a straight line. The positions are rotations can be calculated through basic trigonometry.

Visualising packets is where the complexity is. A packet is visualised as cylinder with a bigger radius than that of the link. The packet cylinder is positioned so that the link cylinder passes directly through it. The packet is created in the centre of the node it starts from, rotated to the same rotation as the link it is to pass down and then its position is updated, step by step, so that it moves down the link. Each new position is calculated by trigonometry to put it one step down the link. How many steps it takes to travel down the link and how long the packet waits at each step determines how fast the packet travels along the link. Both these parameters are configurable.

The incrementation of each packet's position is done by a thread. This thread tracks all packets which have not yet arrived. This thread wakes up every few milliseconds, the precise number can be altered and affects how fast packets travel, and calls the 'updatePosition' method on each packet in transit. Each packet then updates its position. If a packet arrives it uses a delegate method it was instantiated with to notify the visualisation layer class that it has arrived. This method removes the packet from the list of packets to be moved and calls the receive method on the simulation layer.

Not every packet sent across the network gets visualised. Some packets such as those used to highlight routes between packets arrive instantaneously. Because of the requirement that the send method for the visualisation layer be non-blocking this functionality is implemented using a second thread. This thread is a consumer thread for a queue of packets. Every time a non-visualised packet arrives at the visualisation layer it is added to this queue of packets. Once the packet is added to the queue the thread is woken up and proceeds to consume all packets in the queue before returning to sleep. This awakening and sleeping is done using C#'s monitor construct uses the standard threading monitor pattern.

VIII. Conclusion

This paper has described the motivation, design and implementation of a system for visualising routing algorithms within virtual worlds. In the process we have demonstrated that it is possible to develop relatively complex software systems which leverage the power of virtual worlds. Thus learners will obtain in world presence through Avatars and be able to communicate with each other whilst interacting with the simulation. Educational context is provided by populating the virtual world with relevant educational material, such as videos of lectures, powerpoint presentations, standards documents and research papers.

REFERENCES


Investigating Simulation and Distributed Computation in Network State Propagation

Chris Dennett¹, Prof. Madjid Merabti², Prof. Abdennour El Rhalibi³
School of Computing and Mathematical Sciences, Liverpool John Moores University, Byrom Street, L3 3AF, UK.
C.R.Dennett@2006.ljmu.ac.uk¹, {M.Merabti², A.Elrhalibi³}@ljmu.ac.uk

Abstract—In this paper, we focus on the simulation element of a hybrid client-server and authority-driven peer-to-peer MMOG framework currently in development, created in Java and utilising the ns-3 network simulator through a native interface, with some discussion of the overall framework. The framework is intended eventually for use in the development of networked multiplayer games and will aid in respecting bandwidth, fairness and security requirements while reducing the complexity of programming, while facilitating hierarchical state propagation in large-scale game systems.

Our simulation component is implemented to facilitate the development of this system by removing the need for network nodes to be represented by real hardware, allowing the developer to actively visualise the system performance in real-time network application, or collect information for applications running in discrete time. For real-time applications we mimic the Java networking API and use existing network frameworks, requiring few modifications to existing Java applications to port them to our system.

Index Terms—Internet, Simulation, Wide area networks, Computer languages.

I. INTRODUCTION

Massively multiplayer online games have become popular in recent times due their social nature that immerses the player in a virtual world, fuelled by the recent advances in the communication infrastructure alongside software and hardware advances. Additionally, games can offer the potential for delivering education and training in the public sector, military, public safety, and health care industries, as well as for corporate interactions and commerce. However, as players continue to flock into these large virtual worlds, building, deploying and maintaining them has become increasingly difficult. A decade ago, all online games were hosted in monolithic game servers, with each game supporting only a few thousand concurrent players, and a single point of failure. However, with millions of players wanting to connect to these game worlds, this structure was no longer practical.

In order to solve the problem of handling large numbers of users, games such as World of Warcraft introduced the idea of ‘sharding’ where instances of the game world were hosted by several servers with each server supporting many thousands of users. While improving over the traditional monolithic box approach, this mechanism also suffers from critical points of failure for users. Additionally, as each server is independent of each other (except for sharing user login data, score, credits etc), users in a server cannot interact with those in another.

Second Life took a more complex approach for solving the scalability issue by partitioning the game world into several geographically defined cells, with each cell assigned to a separate server. This method allows the players to interact with all the players who are connected to the game by walking around the world and as a result moving between servers. While being elegant, this approach introduces several challenges that need to be solved to assure a satisfactory user experience. Issues arise with respect to supporting:

- Partition interaction as entities approach or cross edges;
- Load balancing / partitioning based on population distribution;
- Permitting non-spatially bounded interactions.

Due to the above problems with geographical partitioning, several new strategies were proposed. One of the most notable pieces of middleware implementing this concept has been Project Darkstar, initiated by Sun Microsystems. Darkstar depends on a fully transactional database backend to store game objects for synchronisation and consistency handling, while the multi-node aspect uses a distributed object approach implementing a publisher-subscriber approach [1] to relay messages to players. By doing this, Darkstar allows the processing workload to be broken down into small tasks which can be scheduled on any node.

While a single server implementation of Darkstar can handle several thousand users, the initial implementation of this ambitious system was found to scale negatively with multiple servers [2]. The developers are currently researching the possibility of introducing caching and clustering objects based on access patterns to solve this problem.

We are developing our tier-based framework along these lines, sidestepping existing issues by introducing new concepts while utilising client-server and peer-to-peer approaches to propagate data with simulator-linked debugging facilities. In particular, the aspect of state filtering with a state prediction engine is at the core of our framework, reducing data traffic by limiting unnecessary communication through a visibility system which defines the amount of entity state variable prediction and the data propagation frequency between tiers.

We use a hybrid client-server and peer-to-peer approach in order to reduce client-server latency and provide server mirroring in the case of server failure, with Area of Interest Management optimisations through state visibility and distributed computation. In addition, we wish to show how the usage of a simulator can feed visualisations back into the application to aid the developer in creating the game
networking logic, and also to help locate bottlenecks in their current configuration.

We believe that the novelty of our work lies in the creation of a Java interface for the ns-3 simulator which was previously not available, and interaction between the simulator and our framework in order to provide quality-of-service information, and allow adjustments to bandwidth, latency, and throughput on individual connections as the game progresses. Using these metrics, we can monitor interactivity on individual nodes with the aim of increasing it while reducing bandwidth requirements and network congestion.

In this paper we focus on the simulation framework. In chapter 1, we have introduced the aims and objectives of the work. In chapter 2, we discuss and contrast our work with related work from literature. In chapter 3, we give an overview of the simulation. In Chapter 4, we introduce the simulation architecture. In chapter 5, we discuss the framework. In Chapter 6, the prediction and integration systems. In Chapter 7, our implementation and testing. In chapter 8, we conclude the paper.

II. EXISTING WORK

There is much work in the field of network simulation, and the various simulator applications come under various guises for different applications. Network simulation has also helped to design some of the most prolific network optimisation mechanisms, sometimes utilising existing game engines such as the Quake 3 engine. Ferretti wrote a thesis on the subject of state synchronisation, titled ‘Interactivity Maintenance for Event Synchronisation in Massive Multiplayer Online Games’, which he has referenced in one of his papers [3] where he discusses this problem of effective state consistency while avoiding interactivity loss from dealing with old events.

He says that game state servers are too computationally involved with dealing with old events to concentrate on fresher events. To these ends, he introduces a new interactivity restoring approach known as ILA-RED (Interactivity Loss Avoidance – Random Early Detection). Here, the level of interactivity between game state servers is monitored, and when this drops below a set limit, events are dropped to remove some of the computational burden on the game state servers, allowing the execution of newer events more quickly [4]. This avoids the interactivity loss from happening in the first place from congestion and is therefore a proactive approach to the problem rather than a retroactive one. The dropping probability is based on a dropping probability which is inversely proportional to the current interactivity degree. In essence, some obsolete events are discarded, even in presence of sufficient interactivity [5].

One of the first instances of state synchronisation in computer gaming can be traced back to iMaze, a centralised multiplayer computer game that works well in low-latency and low-loss environments, but has problems when used on the Internet [6][7][8]. The iMaze server is tasked with managing the sessions of the clients, sending the game maze configuration, and propagating client input via multicast UDP. Following on from this, MiMaze was an experiment to produce a distributed version of this game [8], and used bucket-synchronisation mechanisms with RTP [6][9] to ensure that each client had their own compatible and up-to-date view on a networked game state [8][8]. Also, there was no static shared state; each client computed new states based on their current state and the state data received from other clients. Although it is true that this method is not directly suitable for fast-paced FPS games due to delayed event processing, MiMaze compensated for this latency by dead-reckoning the next state using the last valid message [6][9].

An interesting application of state propagation exists within the Unreal Engine, the first version of this being produced in 1998 for use in the popular Unreal and Unreal Tournament games [10], but is now up to Unreal Engine 3. The Unreal Engine itself provides dynamic state visibility and accuracy with synchronised game objects and remote function invocation. It uses the concept of ‘Actors’ for interactive game objects [11] with sub-classes providing extra functionality. The more important an actor is to the game session, the higher the network prioritisation it has [11], which is inferred in the actor sub-class (i.e., bots, movers, projectiles, pawns and decorations). Here, the state of a Pawn object is propagated more frequently than a decoration object which is less important to the game. Additionally, a lock-step predictor / corrector algorithm is used in earlier versions of the Unreal Engine to smooth out player movement, providing move storage, cancellation and correction. Later versions of the engine improve this, and tie the local prioritisation for an Actor instance into area of interest concepts based on their relative spatiality [12]. We are re-using some concepts from this framework in novel ways, reinterpreting how they function and adapting them to Java language features.

There have been several studies to analyse traffic patterns in a typical MMOG [13][14][15][16], which may help to design world partitioning mechanisms in order to balance players across multiple servers more easily. One of the most important results here is about how players in a particular region of the game tend to become active simultaneously [14], perhaps waiting for a particular quest to become available. This information may allow development of better spatial partitioning algorithms which can predict this behaviour.

It is widely known that different synchronisation mechanisms are more appropriate for different genres of game to reduce code complexity, maintain ideal levels of interactivity, and reduce data transfer [17][18][19]. For instance, turn-based strategy games are often strongly synchronised in lockstep fashion using an assured protocol such as TCP, while first-person shooter games are loosely synchronised using TCP and UDP. In some cases, a bespoke TCP-like protocol over UDP is used instead of TCP. An advantage of using this style of application-level protocol is the ability to take reception of packets before sending acknowledgement, and also removing blocking behaviour on sending data which can increase transmission frequency.

Games requiring quick user reactions concentrate on maintaining interactivity without forcing remote states to be equal, whilst handling discrepancies in a controlled manner.
Here, non-assured protocols such as UDP are used to transmit short-term variables such as entity positions or rotations [20]. Slower-paced turn-based games which may operate in lockstep use assured events containing large quantities of data to convey discrete state modifications, rotating input into the game state to each player as the game dictates. We can permit both of these approaches in our framework using states with optional real-time concepts operating within these states.

III. SIMULATION OVERVIEW

We decided to create the initial implementation of our state consistency and propagation middleware in Java. As we wanted to allow simulation of our system on a single machine to demonstrate how it can aid development, a simulation platform was required to emulate the lower layers of the OSI stack. This makes it possible to avoid needing multiple computers involved in the simulation, and also to inspect entirely what is happening at any given time using simulator tracing via trace files and active visualisation. We also require a simulation system which is capable of real-time synchronisation with the system wall clock. One of the difficulties we encountered was a lack of fully-featured Java-based wired-network simulation environments. Although we investigated native Java simulators such as Java Network Simulator (JNS), we found that they were limited with few protocols and limited trace facilities. Also, many of these had no real-time operation. To these ends, either of the mature ns-2 or ns-3 systems were identified as possible avenues for network simulation of the OSI layers. As these have been implemented in C and C++, a communication mechanism between the Java and native layers was required. In this regard, we looked at some systems which allow this functionality.

AgentJ [21] is an ns-2 agent middleware which enumerates multiple ns-2 agents across multiple nodes. It runs the Java Virtual Machine (JVM) within ns-2 and uses the ProtoLib library, a C++ networking abstraction toolkit. As the name suggests, AgentJ acts as a Java agent within the ns-2 simulator, providing the ability to run a specified Java class which conceptually extends AgentJ code. It avoids the need for multiple JVM instances for the separate nodes by using a custom class loading system, thus speeding up the simulation when many nodes are required, but encapsulates the whole Java Runtime Environment (JRE) initialisation. An important feature of AgentJ is that it instruments the Java networking classpath with bytecode rewriting [21], allowing for existing Java network applications to run unmodified with their normal behaviour (the Java UDP implementation testing tool referenced in ‘Automatic Simulation of Network Problems In UDP-Based Java Programs’ [22] by Farchi et al. also does this, simulating network disturbances by instrumenting the bytecode by replacing calls to the datagram socket API with its own).

With AgentJ we were unsure if we could obtain debug information which would be essential in visualising network activity – it hides implementation details behind Protolib. We were also concerned about the overhead of the Tcl scripting interface over which data is transferred, and the portability of the overall solution. In the end, we focused on ns-3 and our own Java interface, allowing use of the simulator tracing facilities and interaction with ns-3 objects without being hindered by abstraction, with a similar API to Java net classes.

IV. SIMULATION ARCHITECTURE

ns-3 is a discrete network simulator with both discrete real-time support. Discrete applications differ from real-time applications in that use threads which typically delay their execution periodically to yield execution to other threads, avoid resource overuse, and handle user input. The ns-3 build process creates a DLL and executables that use this as opposed to ns-2 which uses a scripting interface to set up simulations. In our project, we use a side-by-side Java and C++ build system using Ant and cppunit to enable the developer to adjust any part of either component and quickly recompile our intermediary DLL which acts as the arbiter between Java and ns-3.

The diagram in Fig 1 (above) demonstrates this architecture. Our interface sets up the initial simulation environment with input from the Java code (for instance, the network topology details) and forwards callbacks from ns-3 objects into Java objects. To mirror the Java classpath and avoid bytecode instrumentation, we utilised modified OpenJDK source code for the implementation of our network API, implementing concepts such as socket channels and selectors, and similar interfaces for dealing with sockets and network addresses. These functions consume real-time in threaded scenarios where required through a blocking API, which utilises an accessible underlying non-blocking API.

V. FRAMEWORK OVERVIEW

In our framework, a node maintenance layer handles details such as client connections, topology information and message passing, implemented using high-level networking libraries such as Apache MINA and Project Grizzly.

The logic layer advances the game state itself and also manages feedback from the player on client nodes, as well as controlling class specialisation via generics and annotations.

In Fig 2 (below), the specific client player is essentially provided by a subclass of ClientSelfPlayer, essentially a specialisation of the ClientPlayer class for the client controller only which can provide local temporary storage and accept immediate events which are later server-confirmed. It is important here that the view remain separate from the logic, as
references to client objects may become invalidated during state rollback. Within entities, the engine uses JavaBean technology to track when variables are modified, with high-level entity concepts notified if they are responsible for the affected variables, otherwise generic variable change events are propagated. Our high-level events can take other relevant fields into themselves in order to transfer and maintain correct event contexts (e.g., a gun shooting event on a player character takes position and rotation).

We can say that the storage for the player data actually happens at multiple layers. The topmost tiers hold authentication details about the player, which is restricted information and stored as variables in tier-specific classes. These tiers also store information about the player’s username, avatar details, equipped items, and so forth. These state variables can be strongly synchronised arbitrarily or at entity context transitions (such as an avatar transitioning from a ‘dead’ to ‘alive’ state) with real-time concepts such as ‘position’ or ‘weapon’ providing loose synchronisation that can facilitate quicker – but less accurate – state propagation between nodes. Entity concepts hold special temporal structures such as event streams or stored movement lists and can be used where performance is needed, providing extrapolation, interpolation, correction, and dead-reckoning of their managed variables. They can also push data into entity variables, with changes made to these variables propagating back into the concept.

Any propagation of variables between concepts in entities on different tiers is managed by the concepts themselves, with the ability to customise the propagation code via sub-classing for tier specialisations. For accessing the state contained in a concept at a particular point in time, the concept can emit a class satisfying elements of an interface for a given timestamp which the entity also implements. Concepts are organised in a tree, such that a ‘weapon’ concept may depend on the ‘position’ concept, whereupon if an event is emitted from this concept (such as ‘gun fire’), position information is automatically added in a chaining mechanism. Concepts are also linked to the current entity context and can be invalidated by this, preventing automatic integration into the entity state during this invalidation period. This mechanism also makes good use of space available within datagram packets.

For concept-managed variables in our system we define concept visibility ranges which affect how the variables are propagated between tiers, with cut-off points defined where visibility becomes zero. In Fig 4 (above), we can say that the cut-off point for character position is the zone level. However, the zone that a player occupies is of importance to the world, and the server managing this needs to be updated on this. Therefore, the cut-off point for this concept is the world tier. For entity variables which aren’t managed by a concept, this visibility is either 0% or 100% depending on whether the variable actually exists for that tier in the specialised entity object, or we can manually provide tier cut-offs in annotations.

**VI. PREDICTION AND INTEGRATION**

Although each client is capable of predicting the future values of state variables by state advancement, only state authority nodes are able to integrate these into the authoritative state (i.e., the state which commands the game as a whole). This authoritative state is distributed across the network and driven by local processes on nodes via distributed computation mechanisms. No single machine has an accurate copy of the entire state, but nodes which hold authoritative states may synchronise these periodically with a database.

Elements of the state, represented by state entities, have a locality in the server node hierarchy on a particular tier, which additionally has authority over that state. This information can be cancelled ahead of time based upon new input received from the client, and new information can be distributed to the server nodes which hold inaccurate information. This method of propagation promotes interactivity as nodes can seed information when the system is less stressed. For connected peers, we need to monitor when the margin for error of certain game variables become more than is tolerable, causing loss of interactivity. In order to do this, nodes need to store concept information from entities of other directly-connected nodes, both sent and predicted. In this system, we can also get peer-to-peer participation in the seeding of this information which
can be merged into a higher-level view of the data.

The spatial node indicated by the points would resend its information to interested peers when it leaves the position tolerances, indicated by the shaded areas. It is possible that $t_4$ is ahead of the event synchronisation delay (the game state on the remote nodes here could be at $t_3$), and so the resend process would be invisible to the user. Here, interactivity has been increased at the cost of introducing some delay to allow for bandwidth usage optimisation. If $t_4$ is behind the delay (with remote nodes at a future game state, i.e., $t_{4>\text{ref}}$) correction can take place by cancelling local and remote states and issuing new information, but this process is not invisible to the player – events relating to players with high latency may not connect and this may seem unfair to players with better connections. For instance, they saw their fire hit the other player on their local state, but the state was corrected on the authoritative server so the event didn’t ‘connect’.

Entity concept prediction engines feed values into the entity on updates in order to ensure that the level of interactivity with the game is kept as high as possible. Ideally, all players should have the same game state such that $S_i(t) = S_j(t)$ for any pair of players $(i,j)$ and at any time instant $t$, but this is not possible in networked environments due to network latency. With $D_{i,j}$ being the network delay from player $i$ to player $j$, an action performed by player $j$ at time $t_0 + D_{i,j}$ will become known to player $i$ at time $t_0 + D_{i,j} + D_{i,j}$ [23]. To compensate, we need to extrapolate future information from past information, correcting error in the state when new information is fed from other nodes. The diagram Fig 7 (below) shows how spatial predictions using position and velocity sampled at discrete time-steps can often not lead to the generation of correct position information at a future time-step.

However, there are problems with the processing required to undo events and the cost of implementation. An efficient mechanism to handle this is named trailing state synchronisation and discussed by Cronin et al. in ‘A Distributed Multiplayer Game Server System’ where he talks about the development of a peer-to-peer multiplayer game system [24] which is used in his mirrored game architecture. It is important to take into account that each client has events playing out at separate times and use event connection windows to compensate for possible error. The unfairness which arises from interacting with players that have excessive latency can be handled by further increasing the delay synchronisation time or penalising these players by increasing the aforementioned hit on their emitted events to catch movements seen by other players but not replicated on the authority.

VII. IMPLEMENTATION

To demonstrate the visualisation abilities and work out problems in our system, we have implemented the logic behind the Boids distributed flocking simulator [25], with ‘boids’ (simulated creatures) represented by nodes in a hub configuration. Fig 8 (below) demonstrates the aforementioned simulator interface running multiple nodes with our initial state propagation framework installed on them, with sections for viewing the game world, statistics, simulator information, and the event timeline for individual entities.

From our experiments, we already have some feedback...
which showed that we were running out of memory with many simulated nodes on the system, as MINA was using buffer allocation which didn’t scale for our requirements. To fix this, we re-routed buffer allocation to the slab allocator in the Grizzly framework, which appeared to stop the simulation from running out of memory and also reduced pauses caused by garbage collection.

The simulation is now able to handle over 100 nodes on Intel Core 2 Quad Q6600 processors with 50ms delay between client transmissions (based on the levels of skew in game logic loops and receive buffer fill rate), but we expect this amount to rise when we implement movement prediction to reduce data transfer between nodes. Although there was clock skew which was likely a result of the processing on client nodes, we were able to reduce this by adjusting thread priorities for logic and UI threads. In the future, we will increase the number of nodes in the network to simulate a real internet with full routing, and add error models to simulate packet loss on links. This should introduce out-of-order packets and variable packet latency. Additionally, we will add tiers into the simulation and divide up the world, initially with binary space partitioning but moving up to move complex mechanisms, showing the effects of entity transit between spatial boundaries in the framework.

VIII. CONCLUSION

In this paper, we have shown how our Java ns-3 interface and state synchronisation framework are advancing, and have given some initial stress-testing results from our simulator with Apache MINA functioning via the Java NIO classes. Although more work needs to be done on both aspects of the project such as looking into threading and performance issues, the simulation part of the project is mostly complete and will be of use to the network simulation community. We are also reducing the footprint of third-party networking frameworks which were originally designed only to run as a single instance on a machine, but start to exhibit performance issues when used in unexpected ways.

Future work includes completing simulations and the MMOG engine, and giving more flexibility when defining the network topology which will eventually be visible in pyviz, a simulation visualiser that uses the Python ns-3 bindings. Additionally, we plan on extending further tracing facilities from ns-3 into Java, parsing this streaming information and inserting it into Esper and using the facilities of flomwon, a network monitoring framework for ns-3 which provides high-level flow data collection. We hope to get these facilities integrated into visualisations into our MMOG engine to demonstrate weaknesses in networked game logic to developers which can be improved with iterative development.

REFERENCES
