A Survey of ‘Game’ Portability

Ahmed BinSubaih, Steve Maddock, and Daniela Romano

Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello Street, Sheffield, U.K.
+44(0) 114 2221800
{A.BinSubaih, S.Maddock, D.Romano}@dcs.shef.ac.uk

Abstract. Many games today are developed using game engines. This development approach supports various aspects of portability. For example games can be ported from one platform to another and assets can be imported into different engines. The portability aspect that requires further examination is the complexity involved in porting a 'game' between game engines. The game elements that need to be made portable are the game logic, the object model, and the game state, which together represent the game's brain. We collectively refer to these as the game factor, or G-factor. This work presents the findings of a survey of 40 game engines to show the techniques they provide for creating the G-factor elements and discusses how these techniques affect G-factor portability. We also present a survey of 30 projects that have used game engines to show how they set the G-factor.

Keywords: game development, portability, game engines.

1 Introduction

The shift in game development from developing games from scratch to using game engines was first introduced by Quake and marked the advent of the game-independent game engine development approach (Lewis & Jacobson, 2002). In this approach the game engine became “the collection of modules of simulation code that do not directly specify the game’s behaviour (game logic) or game’s environment (level data)” (Wang et al, 2003). This makes the game engine reusable for (or portable to) different game projects. However this shift produces a game which is notoriously dependent on the game engine. For example why can’t a player take his favourite game (say Unreal) and play it on Quake engine or vice versa?

Hardware and software abstractions have facilitated the ability to play a game on different hardware and on different operating systems (in some cases with some modifications). These abstractions have also facilitated the ability to use level data assets such as 3D models, sound, music, and texture across different game engines. This ability should also be extended to allow for the game itself to be portable. The goal of our work is to make the game engine’s brain portable, where the brain holds the game state and the object model and uses the game logic to control the game. We collectively refer to these three things as the G-factor. We see the portability of the G-factor as the next logical step in the evolution of game development. Following Lewis and Jacobson's terminology (Lewis & Jacobson, 2002), we call it the game engines independent game development approach (see Figure 1). Figure 1 illustrates the evolution of game development and highlights the issues facing each approach.

A benefit of making the G-factor portable would be to encourage more developers to make use of game engines, since a particular game engine's future capability (or potential discontinuation, as was the fate of Adobe Atmosphere which was used for Adolescent Therapy – Personal Investigator (Coyle & Matthews, 2004)) would not be a worry as a different game engine could easily be substituted. This problem has

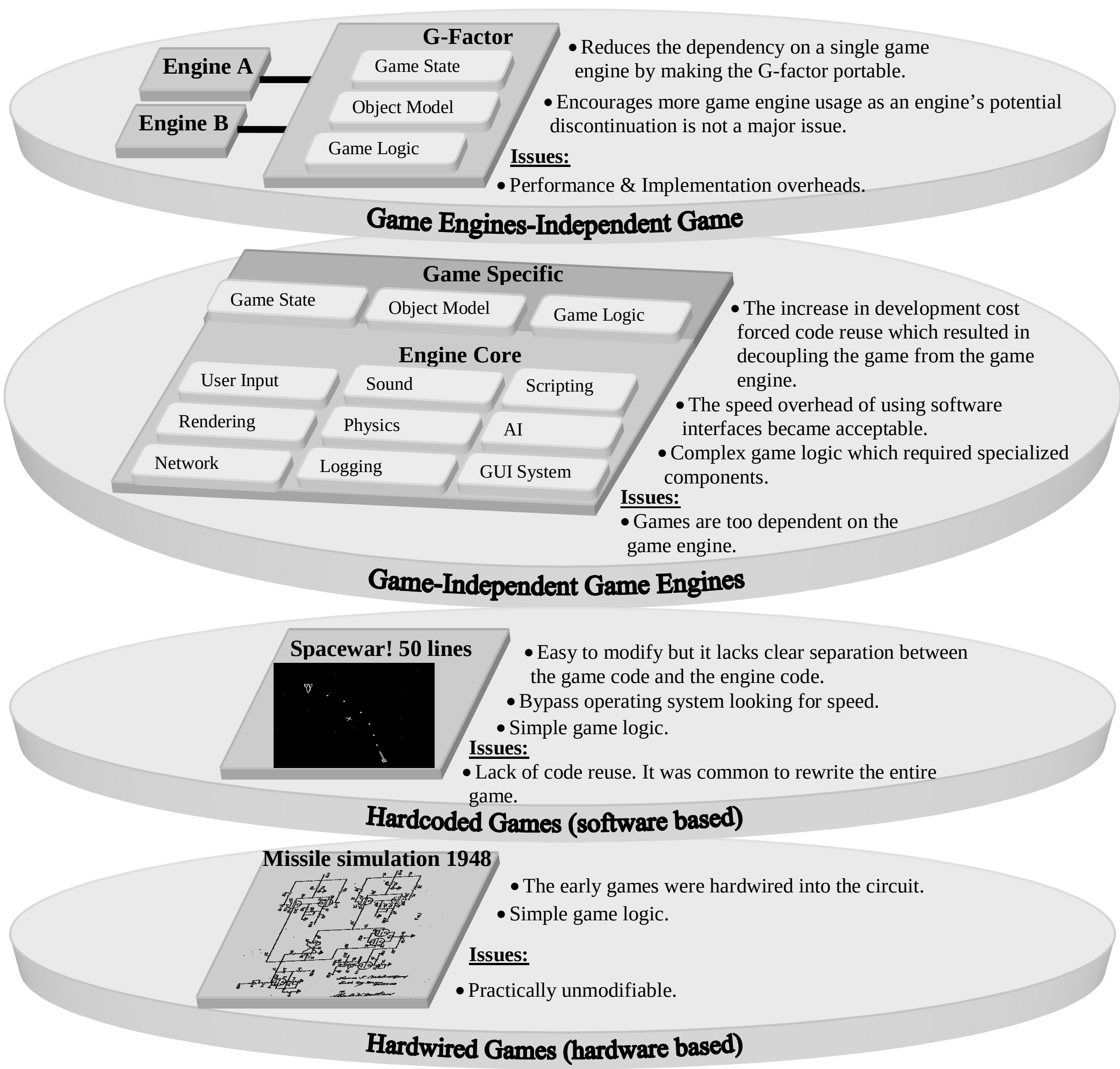

Figure 1: Game development evolution.

recently been referred to as "the RenderWare Problem" (Carless, 2007) after the acquisition of RenderWare engine by Electronic Arts (EA) and its removal from the market. We see the issue of rewriting the G-factor from scratch every time we migrate from one engine to another as similar to the undesired practice of developing games from scratch which was deemed unfeasible and resulted in the advent of game engines.

To identify the extent of the portability problem in game development in general and game engines in particular we decided to conduct two surveys. The first survey was a survey of game engines. The objective of this survey was to illustrate the effects the development practices encouraged by game engines have on the G-factor elements. The second survey was on projects that have used game engines. The objective of this survey was to examine how portable the G-factor for projects that use game engines is.

Section 2 describes the aspects of portability in relation to game engines and the techniques that have been tried to aid G-factor portability. Section 3 describes the governing variables and how they affect the G-factor implementation and how we used them to create a categorization for game engines based on how they promote portability. Section 3 also presents the findings of a survey conducted to identify the methods game engines provide for creating the G-factor. Section 4 presents the second survey which examines the common practices followed by projects using game engines. Finally section 5 presents the conclusions.

2 Portability and G-factor

Figure 2 illustrates the current aspects of portability addressed in game engines. First, with hardware and software portability the game can be played across different platforms and operating systems by employing hardware and software abstractions. Second, portability of assets means that 3D models, textures and sounds can be used across different game engines. Third, middleware portability allows for components to be used across game engines such as AI and physics.

The aspect of portability that requires further investigation is the G-factor portability. Examining what has been done to aid this portability we found initiatives and projects which can be grouped into four areas: artificial intelligence (AI) architectures, interfaces, standards and file formats, and frameworks or protocols.

The AI architectures use custom made or off-the-shelf components such as the AI Middleware (e.g. SOAR (Laird et al, 2002), AI.Implant[1], etc). The need for using a component to handle the AI emerged because of the increase in AI complexity and the increase in the processing time allocated for it. This made reinventing the AI wheel every time a game is developed a redundant process. From a software engineering perspective the use of AI architectures is encouraged as it promotes above all reusability. The practice of specifying the game using the AI middleware format is not what we eventually want since this merely moves it from one proprietary format (game engines) to another (AI middleware). Nevertheless it is a

[1] http://www.biographictech.com (accessed 5/5/2007).

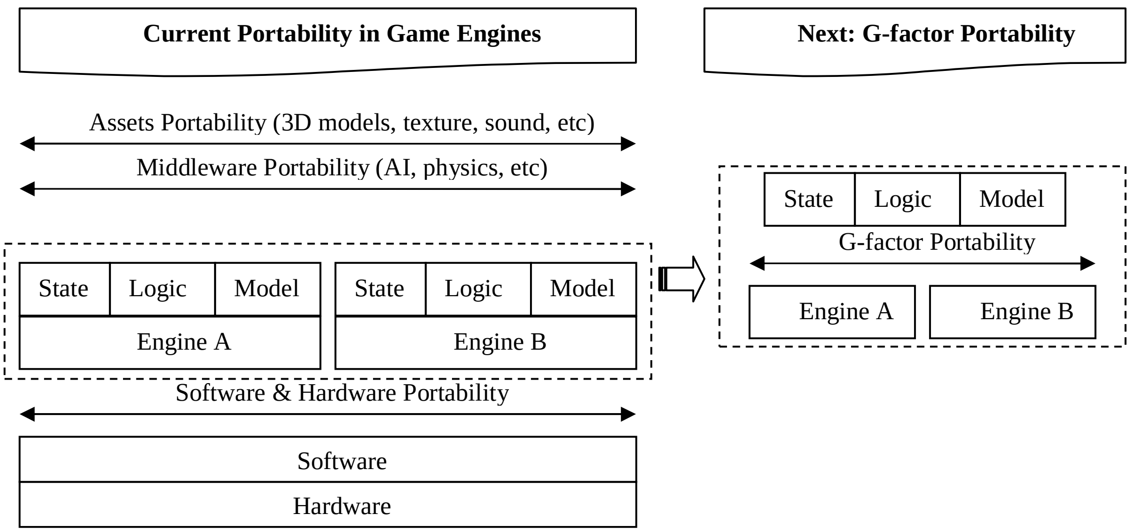

Figure 2: Portability in game engines.

step in the right direction of moving the game away from the game engine's format. The architectures that promote portability more than others are those that allow complete removal of the game from the game engine such as TIELT (Aha & Molineaux, 2004). Others that only partially remove the game are obviously less portable such as Mimesis (Young et al, 2004) and MissionEngine (Vilhjalmsson & Samtani, 2005). The AI architectures promote the use of their own proprietary format which is similar to what game engines do. Furthermore suggesting a monolithic architecture as a complete entity is not what is needed. Instead initiatives must examine the causes of the G-factor portability problem and provide practical solutions that can be employed even if their architecture or middleware is not chosen.

The interfaces aim to provide access to external programs and in game engines we found two types of interfaces: specific and common. These provide access to the G-factor elements to overcome the difficulty raised by the lack of interoperability. A number of interfaces have been developed to provide access to specific game engines. For example the interfaces that have been used to access Unreal are Gamebots (Adobbati et al, 2001) and GOLOG Bots (Jacobs et al, 2005). To access Quake one can use Quakebot (Laird, 2001). FlexBot (Khoo et al, 2002) is used to access Half-Life and Shadow Door (Hussain & Vidaver, 2006) is used for Neverwinter Nights. These provide interfaces for specific game engines. Other projects are attempting to provide common interfaces to game engines such as the initiative by International Game Developers Association (IGDA) for world interfacing (Nareyek et al, 2005) and OASIS (Berndt et al, 2005). Interfaces may have more success in the serious games community rather than in a fast evolving games industry.

The third area is the standards and file-based formats such as VRML/X3D[2]. These still lack the maturity needed for game development. For instance VRML lacks the rendering capability required. It also suffers from speed and security issues (Jankovic, 2000).

The fourth area is the frameworks or protocols that aid interoperability between different simulations like the High Level Architecture (HLA) (Smith, 1998) and Java Adaptive Dynamic Environment (JADE) (Oliveira et al, 2003). Despite the fact that this category focuses more on the interoperability between simulations and less on how the game is linked to the simulation it is mentioned here to illustrate that portability exists at different levels. HLA for instance promotes it at the simulation and object level and JADE promotes interoperability at the functionality level. HLA identified the simulation functionality that are generally required across all systems and thus should not only be part of a single simulation system but available for others. To achieve this it moved the general simulation functionality from the simulation system to the HLA infrastructure and thus made the simulation functionality accessible to other simulation systems (Smith, 2000). An example of the functionality provided is object management which is used to share object instances between different simulations. JADE was designed to address the monolithic nature of current Virtual Environment (VE) systems. Oliveira et al argue that in current VE systems it is not possible to replace or increment the necessary functionality. JADE proposes to host Modules without the concern for their functionality which is the responsibility of the VE developer. A Module can encapsulate an entire system or a block of code and thus can be reused by others. These frameworks and protocols require the projects to comply with their infrastructure to be able to interoperate with other systems. The other challenge facing them is to create a generalizable infrastructure to support any kind of environment (Kapolka, 2003).

This section has presented the different aspects of portability that are supported by game engines and has analyzed what has been done so far to address G-factor portability. The next two sections present two surveys to help better understand G-factor portability and to highlight what is required from a development approach that aims to promote G-factor portability.

3 Survey of G-factor in Game Engines

The objective of this survey is to discover the development practices encouraged by game engines through examining the tools they provide to specify the G-factor (e.g. scripting language, object model, API access, world and interface builders, etc). We also aim to create a categorization that groups engines by the way they promote G-factor portability. Current game engines' categorizations listed by researchers include ones based on the player's point of view (Stang 2003) or based on the game genre (e.g. action, strategy, sports, simulation, etc). Another categorization is one proposed by Young et al (Young et al, 2004) that is based on the integration between intelligent reasoning capabilities and game engines. Young et al's categorization is divided into

[2] http://www.web3d.org/x3d/ (accessed 5/5/2007).

three groups: mutually specific, AI specific, and game specific. In the mutually specific category the essence is on creating new functionalities using specific intelligent reasoning tools or techniques (such as planning algorithms) for a specific game engine. This can be described as having a one-to-one relationship between the new AI functionality and the game engine. In the second category, AI specific, a set of AI functionalities can be used across a range of game engines – a one-to-many relationship between the AI functionalities and game engines. The game specific category allows more than one AI element to be used on a specific game engine. This is many-to-one relationship between the AI elements and the game engine.

In our categorization the focus is on the relationship between the G-factor and the game engine. We identified three variables to govern this relationship: location (hard-coded or data-driven), object model (static or dynamic), and scripting constraint (precompiled or compiled and interpreted on-the-fly). These variables describe how the three G-factor elements are set. Location describes how the game state, object model, and game logic are specified (i.e. whether hard-coded in the game engine or specified using data-driven techniques). The object model and scripting constraint variables refer to how the G-factor's object model and logic are set respectively.

The following sections describe how these variables affect G-factor portability and provide examples from industry, wherever possible, to show how these variables are being implemented and more importantly what lessons have been learnt in doing so.

3.1 Game Location

Location refers to whether the engine promotes hard-coded or data-driven approaches to create the G-factor. The hard-coded approach is inflexible and does not meet the current dynamic game design requirements since embedding the game too deeply in the code is very restrictive as it shields it from the designers and artists (Keith, 2003; Schertenleib, 2006). Keith also reports another problem with this approach which is the over dependency on the object hierarchies for behaviour which makes the code fragile and very difficult to maintain. This problem was also mentioned by Bilas (Bilas, 2002) who noted that the line between the content and the engine keeps moving as the requirements get fuzzier and advised a change to a data-driven development approach, warning that resistance would only cause regular refactoring.

The data-driven approach allows the data to be defined by configuration files and/or scripts (Schertenleib, 2006) and these are then fed into the program to dictate its flow. The need for the game engines to be extremely flexible is the reason why it is crucial to have a data-driven design focus where the game is controlled by data which resides outside the engine (Tong, 2003). The advantages alongside the aforementioned flexibility are: extensibility and improved process (Fermier, 2002). The disadvantages are performance, its too powerful (Tapper, 2003), there is more work up-front (Leonard, 1999), over-engineering and lack of ownership (Fermier, 2002), and difficulty in debugging (Wilson, 2003).

The advantages of the data-driven approach outweigh the disadvantages as reported by the developers of a number of commercial games. The developers of 'Gabriel Knight 3' (budget over \$4.5 million, development time almost 3 years) reported that the initial hard-coding of the story sequence of the game in C++ meant that engineers were creating content instead of working on the engine and also that the tiniest changes to the game required recompilation which "made the development process unbelievably inefficient." (Bilas, 2000). Similar problems were reported by the developers of 'Thief: The Dark Project' (budget approximately $3 million, development time 2.5 years) who also moved to adopt the data-driven approach (Leonard, 1999). The developers of 'Jurassic Park: Operation Genesis' (development time 22 months) said that the data-driven approach they used required initial investment but the time spent was saved many times over and it opened up the possibility of creating expansion packs (Chan et al, 2003). They also reported that "the data-driven approach worked so well that through much of our development, Thief and System Shock 2 (two very different games) used the same executable and simply chose a different object hierarchy and data set at run time".

From the portability point of view the separation encouraged by a data-driven approach allows for clearer specification of the boundaries between the data and the system – thus making it more modular. A game that is represented by data is much easier to manipulate and understand than one which is intertwined in the application code. Therefore, any technique that moves the game away from the engine is beneficial to the G-factor portability cause. Moreover, it also allows for the creation of intuitive tools (Shumaker, 2002) for manipulating the data thus increasing modifiability.

3.2 Object/Class Model

The object model describes the classes for the objects in a game. These objects can be divided into two types: game objects and decorative objects. Game objects represent all non-terrain and interactive logic content (Bilas, 2003) and they are the ones that are of interest to the G-factor. The decorative objects are merely used to enhance the look of the environment such as terrain, sky, etc. The object model used can either be static or dynamic.

A static object model has hard-coded representation and cannot be modified at run-time. For instance a new object type (or class) cannot be added without having to modify the hard-coded representation and recompiling and loading the application (e.g. Java is an example of static object model). The problem with this is highlighted by the development of 'Ultima Underworld 1' (Duran, 2003). Initially the development started under the impression that the non-player characters (NPCs) and doors do not share many components. Later on, the designer wanted to allow the player to have a conversation with a door just as he can have a conversation with NPCs but since the initial design only allowed NPCs to have the conversation component, they found that pushing the component up the hierarchy was very difficult and resolved to use a hack around the problem. Similar lessons were learnt by the developers of 'Dark Engine' (Leonard, 1999). The success of that was demonstrated by the ability to have no code-based game object hierarchy of any kind in Thief. This was handled through a general database where an object can possess properties and hold relations with other objects.

A dynamic object model allows the creation and modification of classes along with their properties and hierarchies dynamically. The advantages and disadvantages of using a dynamic object model pattern are clearly described by Riehle et al (Riehle et al, 2005). The primary advantages that aid portability are: end-user configuration, language independent, run-time object type creation, and explicit model. The end-user configuration ability means that the game developer or designer is able to define concepts from his domain (c.f. ontologies (Chandrasekaran et al, 1999)) and does not have to hard-code them. This means the object model can exist outside the game engine and more importantly is modifiable independently of the engine. This promotes flexibility and extensibility. The second advantage is being specified in a language that is independent from the implementation language since object model can be stored outside the application in a file or a database which makes it easier to port between engines of different implementation languages. It also simplifies sharing the object model between games. The run-time object type creation is important for games with persistent worlds like the massively multiplayer online games (MMOG) (e.g. 'Toontown' (Goslin, 2004)). The final advantage is the explicit model provided by the dynamic object model enables querying the object model to find the classes and their properties, property type, inheritance, etc.

The potential disadvantages of using the dynamic object model pattern are the performance and memory usage penalties associated with it. The use of it in industry by games such as Thief shows that it is not undermining the game to the point of making it unplayable. Another disadvantage is that it requires extra work initially to create the framework that is going to hold the dynamic object model. For systems that do not provide a dynamic object model there is a workaround which involves constructing classes dynamically by using on-the-fly scripting languages (described in the next section). These languages can be grouped into two categories: class-based (e.g. Python) and prototype-based or instance-based (e.g. JavaScript). The difference is that in the prototype-based approach there are no distinct entities for classes and instances. The prototype-based approach makes sharing the classes more cumbersome and counterintuitive to developers familiar with object-oriented programming since the class description is embedded in the instance which blurs the separation object-oriented developers are accustomed to.

3.3 Game Logic Scripting Constraint

The third variable to govern the relationship between the G-factor and the game engine is the language processing constraint. As game development moves away from code-driven approaches to a data-driven approach it makes the data more complex to represent and manipulate. What is needed is a simpler approach than the code-driven approach but one that still retains some, if not all, of its flexibility and power. Scripting is an answer to this. Scripting is a programming language that is similar to coding but generally simpler and also requires shorter edit-compile-link-run process[3]. Examples of scripting languages are: Python, Ruby, Lua, etc. They share a number of characteristics (Garces, 2006) such as: they are high-level languages, provide flexible flow control, and they are interpreted languages (not compiled into machine code). Although scripting uses code as the basis for its representation it is considered to fall

[3] http://en.wikipedia.org/wiki/Scripting_language (accessed 5/5/2007).

into the data-driven category (Schertenleib, 2006). Many game development teams found in scripting an ideal solution to the programmer bottleneck problem as was stated by the developers of ‘Treyarch’s Draconus’ (Fristrom, 2000). Despite the known performance issue with scripting, the developers of ‘Centipede 3D’ (Rouse, 1999) and ‘Shiny’s Wild 9’ (Malenfant, 2000) found that the tradeoff for scripting flexibility and ease of use over performance was a positive move. LaMothe (LaMothe, 2002) estimated that about 99% of all commercial games use scripting. Our survey in section 3.4 puts this figure to 74.4%.

Scripting languages can either be precompiled or compiled and interpreted on-the-fly. Precompiled means the code is compiled before the game starts whereas on-the-fly means compiling happens at run-time. This makes the on-the-fly feature very useful for programs that cannot afford to make the application offline such as Massively Multiplayer Online Games (MMOG). However these languages run slower than the precompiled ones. Despite this many developers think the tradeoff is worthwhile. The developers of ‘Pirates of the Caribbean – Battle for the Buccaneer Gold’ (Schell and Shochet, 2001) found on-the-fly scripting very valuable to conduct guest testing. They used the Scheme scripting language to be able to reprogram the game while the guests were live in the game. The MissionEngine (Vilhjalmsson & Samtani, 2005) architecture found in on-the-fly scripting an ideal solution to avoid making the architecture too rigid and too slow to respond to design changes. The dynamic nature of the language used by the architecture (Python) meant that the class definitions in the architecture did not have to be changed every time the data format changed when new features were requested. However that was not the case with the second scripting language they used because they chose Unreal engine. Unreal provides UnrealScript which requires precompiling. They found it to be less flexible than Python as for every change to the page type in the skill builder a new class in UnrealScript had to be created.

For portability, on-the-fly scripting plays a vital role. The first role is to facilitate the dynamic object model workaround described in the previous section. The second role of the scripting is to enable translation through the use of the script mapping technique described in BinSubaih and Maddock (BinSubaih & Maddock, 2006). The third role is to avoid undermining the current flexibility associated with programming directly on the game engine (i.e. avoid introducing a restrictive layer). For instance, Gamebots uses predefined text-based protocol messages to interact with the game engine to receive sensory information (synchronous and asynchronous) and send actions (e.g. CHANGEWEAPON, RUNTO, JUMP, STOP, etc). A project for teaching Bayesian behaviors to game characters (Le Hy et al, 2004) made use of Gamebots and found it to be restricting the interaction they could have with the game engine. TIELT requires adding the actions and sensors that have to be exchanged between the game engine and the decision system to the knowledge bases residing inside TIELT. In a project (Ponsen et al, 2005) that used TIELT for integration with Stratagus, which provides on-the-fly language (Lua), it was found that every time a new action was needed the knowledge base had to be updated to allow that. This shields the on-the-fly language from the decision system undermining the power of the language. Another problem with TIELT, also shared by the protocol messages of Gamebots, is that they introduced their own scripting languages which is not ideal as we now explain.

Developers wanting to add scripting support to their architecture are faced with two options: either to build their own scripting language or use one from the off-the-shelf languages available. Tong (Tong, 2003) noted that as people stop wanting to spend resources on developing their own specific scripting languages a more common option is to leverage the use of existing languages. The advantages to be gained from doing so are: having a rich feature set with plenty of documentation, utilizing a wealth of existing tools, simplifying the interface with the engine code, and utilizing fast and efficient code. The disadvantages are: performance, interface between C/C++ and the scripting language can be constraining, lacks good debugging and development tools, and lack of easily available libraries and extensions. Examples from the industry also echo Tong's call. The developers of 'Gabriel Knight 3' recommend using an existing language to avoid spending time creating documentation of the syntax and training scripters. A more forceful example was cited by 'Toontown' developers who had to change the scripting language after more than six months into the project. The issue with their own proprietary language was to do with performance and code management which forced them to switch to an existing language (Python).

3.4 Categorization

Table 1 describes the categorization we have created for the game engines using the three governing variables (location, object model, and scripting constraint) described in the previous sections. For simplicity and clarity purposes we do not create any category for game engines that might support two properties of the three governing variables. For example, if a game engine locates the game inside it (hard-coded) and can read it from outside (data-driven) we categorize the engine with the most superior property – outside is superior to inside, dynamic object model is superior to static object model, and on-the-fly language is superior to precompiled. The superiority-deciding factor is based on how it promotes G-factor portability. Based on that we have created six categories for game engines: serviced-dynamic, serviced-static, loaded-dynamic, loaded-static, hard-coded-dynamic, and hard-coded-static. The portability column in table 1 indicates the direction of increased portability support. Table A.1 shows the engines surveyed and the category they belong to. The table also includes two columns for the tools provided by the engine (world builders and scripting languages used) and a column for the game engine cost. We added these to the survey to help explain the popularity reasons of a particular engine.

Table 1: Engines' categories.

Category	Location	Object Model	Scripting Constraint	Portability
Serviced-dynamic	Data-driven	Dynamic	On-the-fly	
Serviced-static	Data-driven	Static	On-the-fly	↑
Loaded-dynamic	Data-driven	Dynamic	Precompiled	
Loaded-static	Data-driven	Static	Precompiled	
Hard-coded-dynamic	Hard-coded	Dynamic	-	
Hard-coded-static	Hard-coded	Static		

The findings of the survey are summarised by the four pie charts in figure 3. The categorization chart (figure 3a) shows that 43% of the game engines fall into the serviced-dynamic category. However none of the engines implemented the dynamic object model directly and the ones that do have done so either through the workaround using on-the-fly scripting (section 3.2) or through different techniques. The findings also show that scripting is very popular with 74.4% of the engines supporting it (figure 3c). Figure 3d show that “on-the-fly” scripting (48.8%) to be more popular than precompiled scripting (25.6%). Finally, figure 3b shows that most (69%) of the game engines surveyed cost $100 or less.

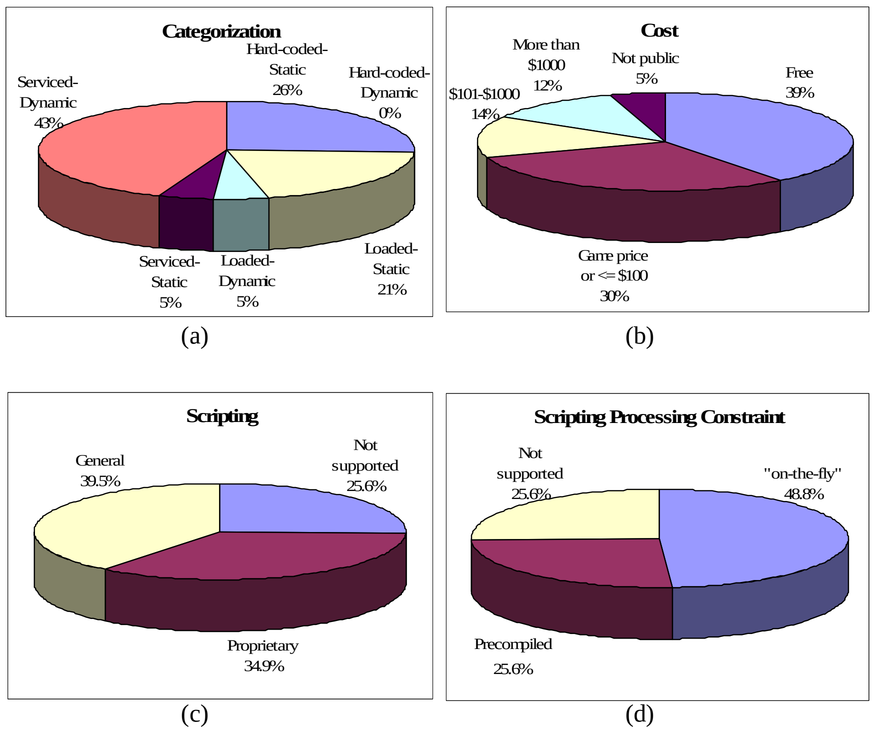

Figure 3: Game engines’ survey showing the G-factor portability, scripting used, and cost.

4 Survey of Projects Using Game Engines

The objective of our survey of projects using game engines is threefold. First it aims to examine how portable the G-factor for projects that use game engines is, by checking how they choose location, object model, and language. The second objective is to find out the reasons cited by the projects for using a specific game engine. This should help identify the attributes that increase the game engine’s popularity and examine how they affect portability. These attributes should help form the base list of the attributes that should be addressed by any game development approach. Finally, the survey gauges the acceptance of using any of the approaches described in section 2 to aid portability and the reasons for doing so. This should provide us with an indicator of how acceptable a development approach that promotes G-factor portability would be.

Table A.2 gives a list of the projects surveyed by listing six items for each project. The first item (column three) specifies the game engine used. The second item (column four) specifies whether the project uses a hard-coded or a data-driven approach or a combination of both. To find out if the concept of having the game state (or part of it) outside the engine is acceptable, item three (column five) shows where the game state is at run-time (i.e. inside or outside or uses a combination of both). The game state holds the game objects. If these objects are living inside the engine only then are they labelled inside. If they are living outside the engine and have corresponding objects inside the engine then they are labelled outside. Finally if part of them is inside and the other part is outside then they are a combination of both.

The fourth item (column six) describes whether the object model is specific or independent or uses a combination of both. If the object model uses the engine specific model or extends it then it is considered specific. If however it uses its own model independently from the engine's model then it is considered independent. If it mixes both then it is considered to be a mixture of both. The fifth item (column seven) specifies the language used to set the game logic. This can either be specific/custom made (e.g. UnrealScript) or independent/general (e.g. Python) or a combination of both. The last column details the approach used to aid portability.

Figure 4 shows five pie charts for the G-factor location, object model, game language, where the game state held at run-time and engine usage. We were concerned that the results are swayed by Unreal as it was used in the majority of projects surveyed (51%). To alleviate this concern we balanced the table to one project per engine which reduced table A.2 to 10 rows of unique game engines. As the listing of the projects in the table was not organized in any way we selected the first occurrence of the engine and disregarded the rest of the projects that use the same engine. The result of the balanced table is shown in figure 5. These results assured us of the trend that was exhibited by the previous results (i.e. unbalanced table) which indicated that the majority of the projects surveyed share the same characteristics of: a high tendency to use data-driven approaches, a high tendency to use the engine's specific object model, a high tendency to use the engine's proprietary language, and a high tendency to specify the game state inside the engine.

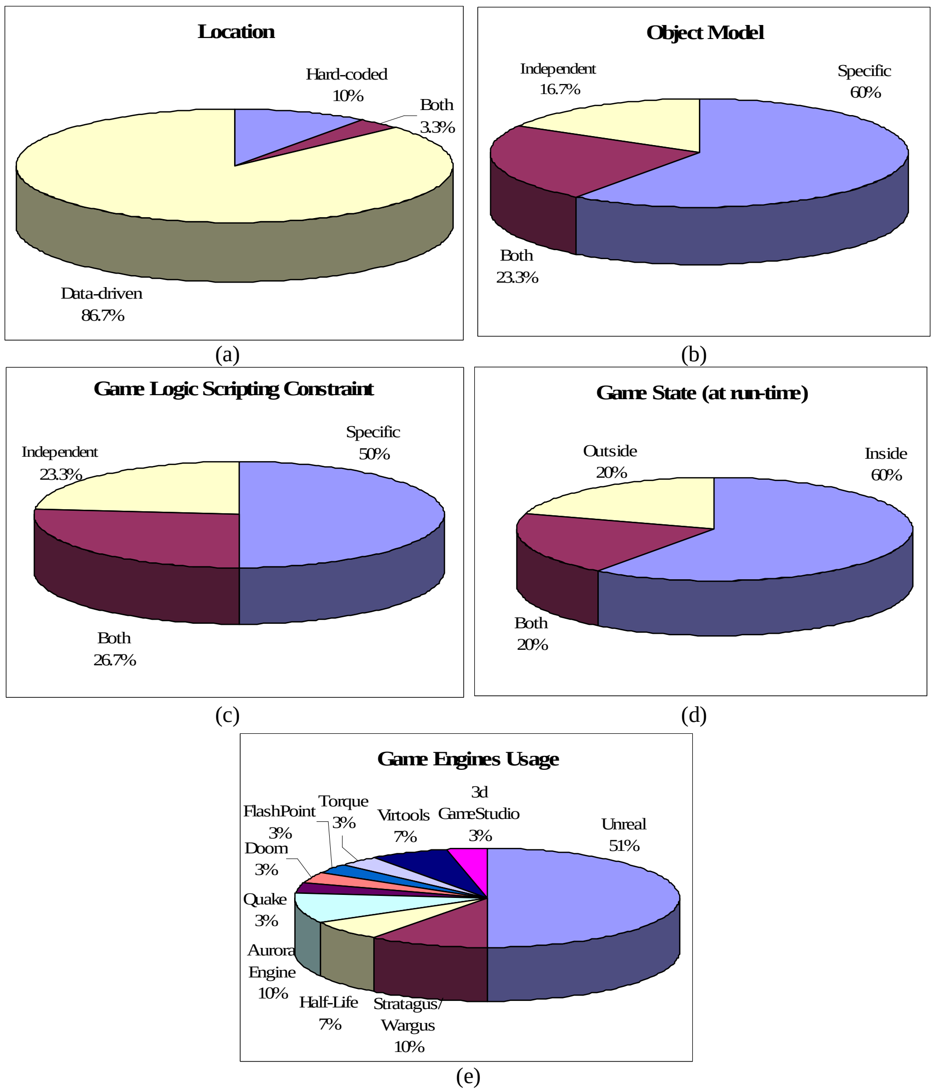

Figure 4: A survey of projects using game engines to show how they tend to set up the G-factor elements and also show the game engines used.

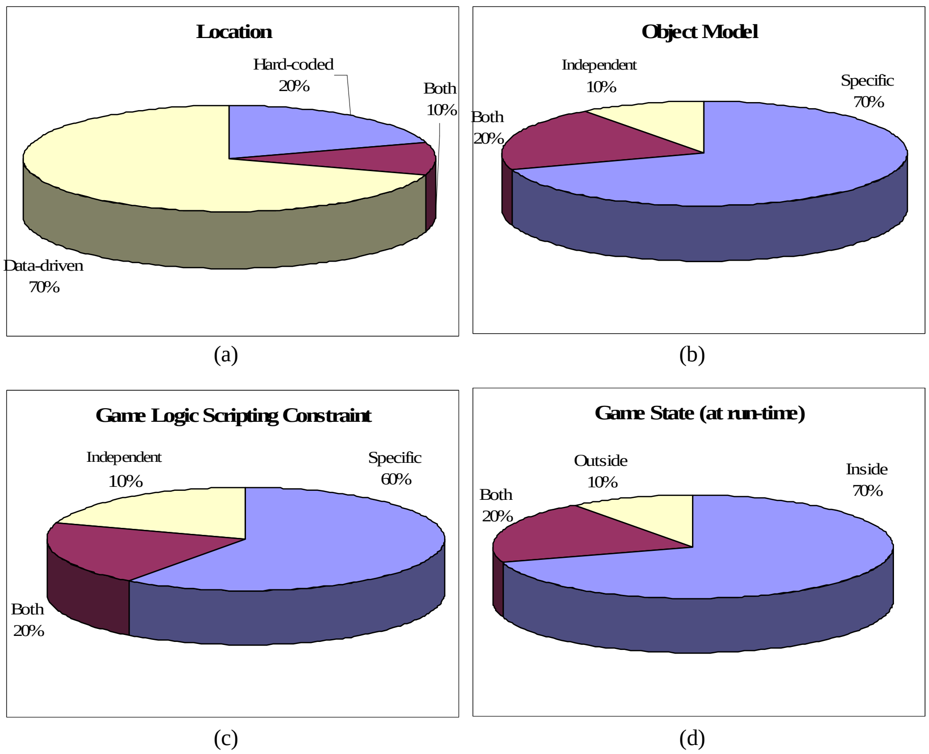

Figure 5: The results of the balanced table show similar tendencies to ones reported by figure 4.

In an attempt to understand the characteristics that make game engines attractive or unattractive we counted the comments made by projects described in table A.2 about each engine. Table 2 organizes the comments by the number of mentions they received (unique per project). As far as portability is concerned, figure 6 shows the six comments that are of importance to any game development approach that aims to promote G-factor portability. We believe these are the elements that should be guarded as much as possible by any new approach. The pie chart shows the level of importance each holds which should help trading off one over the other when a decision may affect more than one element. For instance scripting received 22% while performance was not highly mentioned. This makes scripting a high priority attribute. It is also reflected by the examples we cited earlier from the industry where trading scripting over performance was found to be a positive move (see section 3.3). The chart also shows that a small learning curve is also highly regarded. This backs our earlier argument that introducing something completely new (e.g. new scripting language or new standards) might not be the best option and instead any new approach should aim to make use of well-known practices wherever possible.

This should also reduce the time it takes to make a decision about a particular approach or engine since knowing that the basic building blocks have been tried and tested would increase the confidence in that approach or engine and correspondingly reduce the time to investigate it.

One of the concerns raised about game engines was with regards to the lack of integration ability with external modules. The need for that was raised because of either the lack of needed features (i.e. need for complex AI behaviour (Fielding et al, 2004)) or the need to avoid reinventing the wheel (e.g. building biomedical simulation (Ryan, 2005)). The other issue mentioned was with regards to the use of scripting languages. Interestingly both scripting issues raised were with regards to scripting languages that were custom made. This backs the earlier argument of the need to avoid creating custom languages.

Table 2: Comments order by the number of mentions received.

Comment	Number of mentions
Graphics	10
Scripting	9
Small Learning Curve	9
Features (Physics, AI, Statistics, Recordable)	9
Modifiability (configurable, extensible, flexible, integration, abstraction)	8
Popular/well-tested	8
Multiplayer	7
Low cost/open source	7
Authoring Tools	6
Outsourcing	4
Rapid prototyping	3

The third objective of the survey was to find out the reasons behind using approaches that aid portability. The findings show that 30% of the projects described in table A.2 made use of these approaches. The approaches used fall into the AI and interfaces groups. The primary reasons mentioned for adopting these were the integration with external modules something game engines not supporting very well as described in the previous section. The issues raised were with regards to the restriction introduced over the game engine access.

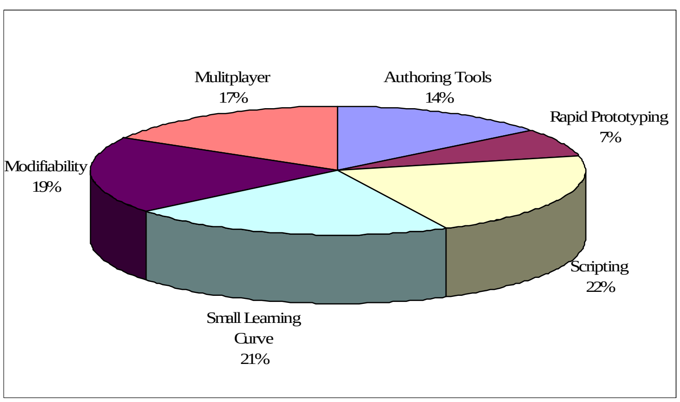

Figure 6: Comments made about the features that are important to projects using game engines which any game development approach should aim to preserve.

5 Conclusions

Certain kinds of portability are supported as discussed in section 2 such as asset portability however G-factor portability has not received similar attention. The consequences of not supporting G-factor portability means that moving a game between game engines is cumbersome and makes the decision to choose a game engine a critical one. We believe there is a need to reduce the immediate and future risks associated with this decision. We believe increasing G-factor portability would make this decision less crucial.

Based on the findings of the this survey we have created an approach to aid G-factor portability (BinSubaih & Maddock, 2006) and have successful used this approach in the development of a serious game for traffic accident investigators in the Dubai police force (BinSubaih et al, 2006a).

References

1. Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S., Kaminka, G., Schaffer, S. and Sollitto, C. 2001. Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research. Proceedings of the International Conference for Autonomous Agents, Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada.
2. Aha, D.W. and Molineaux, M. 2004. Integrating learning in interactive gaming simulators. Challenges of Game AI: Proceedings of the AAAI'04 Workshop (Technical Report WS-04-04). San Jose, CA: AAAI Press.
3. Berndt, C., Watson, I., and Guesgen, H. 2005. OASIS: An Open AI Standard Interface Specification to Support Reasoning, Representation and Learning in Computer Games. IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games. 31-July 2005, Edinburgh, 19-24.
4. Bilas, S. 2000. Postmortem: Sierra Studios' Gabriel Knight 3: Blood of the Sacred, Blood of the Damned, Game Developer Magazine, June 2000.
5. Bilas, S. 2002. A data-driven game object system. Game Developers Conference Proceedings 2002. http://www.drizzle.com/~scottb/gdc/game-objects.ppt (accessed 5/5/2007).
6. Bilas, S. 2003. The Continuous World of Dungeon Siege, Gas Powered Games, www.drizzle.com/~scottb/gdc/continuous-world.htm (accessed 5/5/2007).
7. BinSubaih, A. and Maddock, S. 2006. Using ATAM to Evaluate a Game-based Architecture. Workshop on Architecture-Centric Evolution (ACE 2006). Hosted at the 20th European Conference on Object-Oriented Programming ECOOP 2006 July 3-7, 2006, Nantes, France.
8. BinSubaih, A., Maddock, S., and Romano D.M. 2006a. A Serious Game for Traffic Accident Investigators. Special Issue of International Journal of Interactive Technology and Smart Education on Computer Game-based Learning.
9. BinSubaih, A., Maddock, S., and Romano, D.M. 2006b. An Architecture for Portable Serious Games. Doctoral Symposium, hosted at the 20th European Conference on Object-Oriented Programming ECOOP 2006 July 3-7, 2006, Nantes, France.
10. Blackman, S. 2005. Serious games…and less! SIGGRAPH Computer Graphics 39, 1 (Feb. 2005), 12-16.

11.Carless, S. 2007. Rise of the Game Engine. Game Developer, April, 2007, pp.2.
12.Cavazza, M., Charles, F., and Mead, S. J. 2002. Interacting with virtual characters in interactive storytelling. In Proceedings of the First international Joint Conference on Autonomous Agents and Multiagent Systems: Part 1 (Bologna, Italy, July 15 - 19, 2002). AAMAS '02. ACM Press, New York, NY, 318-325.
13.Cavazza, M., Hartley, S., Lugrin, J., and Le Bras, M. 2004. Qualitative physics in virtual environments. In Proceedings of the 9th international Conference on intelligent User interfaces (Funchal, Madeira, Portugal, January 13 - 16, 2004). IUI '04. ACM Press, New York, NY, 54-61.
14.Chan, K., Spagnolo, S., Stevens, S., Hagger, N., Chau, D., and Carlton, G. 2003. Postmortem: Blue Tongue Software's Jurassic Park: Operation Genesis, Gamasutra March 17, 2003, http://www.gamasutra.com/features/20030317/chan_01.shtml.
15.Chandrasekaran, B., Josephson, J.R., and Benjamins, V.R. 1999. What are ontologies and why do we need them? IEEE Intelligent Systems, Jan/Feb 1999, 14(1), pp. 20-26.
16.Chao, D. 2001. Doom as an interface for process management, Proceedings of the SIGCGI conference on Human factors in computing systems, Seattle, Washington, 2001, 152-157.
17.Coyle, D. and Matthews, M. 2004. Personal Investigator: a Therapeutic 3D Game for Teenagers. CHI2004 Vienna 25-29 April 2004. Presented at the Social Learning Through Gaming Workshop.
18.Creel, J., Maslov, A., Mikeal, A., and Speight, C. 2006. Information and Decision-making in Immersive Digital Environments. http://loam.evans.tamu.edu/courses/cnm/files/Project%20Final%20Report.doc.
19.Darken, C., Morgan, J., and Paull, G. 2004. Efficient and Dynamic Response to Fire. AAAI 04 Challenges in Game AI workshop, July 2004
20.Davies, N.P., Mehdi, Q.H., and Gough, N. 2005. Creating and Visualising an Intelligent NPC using Game Engines and AI Tools. 19TH European Conference on Modelling and Simulation, June 1st - 4th, 2005, Riga, Latvia.
21.Davis, M., Shilling, R., Mayberry, A., Bossant, P., McCree, J., Dossett, S., Buhl, C., Chang, C., Champlin, E., Wiglesworth, T. and Zyda, M. 2004. Making America’s Army The Wizardry Behind the U.S. Army’s Hit PC Game. Jan, 2004.
22.Diller, D., Roberts, B., Willmuth, T. 2005. DARWARS Ambush! A Case Study in the Adoption and Evolution of a Game-based Convoy Trainer by the U.S. Army. Simulation Interoperability Standards Organization, 18-23 September (2005).
23.Duran, A. 2003. Building Object Systems. Game Developers Conference Proceedings. 2003. http://www.ionstorm.com/gdc2003/AlexDuran/ (accessed 5/5/2007).
24.Eliens, A., Bhikharie, S.V. 2006. Game VU developing a masterclass for high-school students using the Half-life 2 SDK.GAME'ON-NA'2006, September 19-20, 2006.
25.Fermier, R. 2002. Creating a Data Driven Engine: Case Study: The Age Of Mythology, GDC 2002, http://www.gamasutra.com/features/slides/fermier/index.htm
26.Fielding, D., Fraser, M., Logan, B., and S.Benford. 2004. Extending game participation with embodied reporting agents. Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology. ACE '04. 100 –108
27.Fristrom, J. 2000. Postmortem: Treyarch's Draconus, Gamasutra, August 14, 2000, http://www.gamasutra.com/20000814/fristrom_01.htm (accessed 5/5/2007).
28.Garces, D. 2006. Scripting Language Survey, Game Programming Gems 6, 2006, Charles River Media, pp. 323-340, ISBN: 1584504501
29.Goslin, M. 2004. Postmortem: Disney Online's Toontown, Gamasutra January 28, 2004, http://www.gamasutra.com/features/20040128/goslin_01.shtml (accessed 5/5/2007).
30.Heckenberg, S. G., Herbert, R. D., and Webber, R. 2004. Visualisation of the minority game using a mod. In Proceedings of the 2004 Australasian Symposium on information Visualisation - Volume 35 (Christchurch, New Zealand). N. Churcher and C. Churcher, Eds.

ACM International Conference Proceeding Series, vol. 99. Australian Computer Society, Darlinghurst, Australia, 157-163.
31. Hunicke, R. and Chapman, V. 2004. AI for Dynamic Difficulty Adjustment in Games. In Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence (AAAI '04) (San Jose, California) AAAI Press, 2004.
32. Hussain, T.S. and Vidaver, G. 2006. Flexible and purposeful NPC behaviors using real-time genetic control. Proceedings of the 2006 World Congress on Computational Intelligence (July 16-20, Vancouver, BC).
33. Jankovic, L. 2000. Games Development in VRML. Virtual Reality 2000, 5. pp. 195-203.
34. Kapolka, A. 2003. The Extensible Run-Time Infrastructure (XRTI): An Emerging Middleware Platform for Interoperable Networked Virtual Environments. Proceedings of the Lake Tahoe Workshop on Collaborative Virtual Reality and Visualization, October 2003.
35. Keith, C. 2003. From the Ground Up: Creating a Core Technology Group,Gamasutra August 1, 2003, http://www.gamasutra.com/features/20030801/keith_01.shtml (accessed 5/5/2007).
36. Khoo, A., Dunham, G., Trienens, N., and Sood, S. 2002. Efficient, Realistic NPC Control Systems using Behavior-Based Techniques. Proceedings of the AAAI 2002 Spring Symposium Series: Artificial Intelligence and Interactive Entertainment, Menlo Park, CA.
37. Laird, J. 2001. It Knows What You're Going To Do: Adding Anticipation to a Quakebot. Agents, 2001, pp. 385-392
38. Laird, J.E., Assanie, M., Bachelor, B., Benninghoff, N., Enam, S., Jones, B., Kerfoot, A., Lauver, C., Magerko, B., Sheiman, J., Stokes, D., and Wallace, S. 2002. A Testbed for Developing Intelligent Synthetic Characters. In Artificial Intelligence and Interactive Entertainment: Papers from the 2002 AAAI Spring Symposium, Menlo Park, CA (2002).
39. LaMothe, A. 2002. Letter from the Series Editor which appeared in Alex Varanese. Game Scripting Mastery, Premier Press (18 Dec 2002),ISBN: 1931841578.
40. Le Hy, R., Arrigoni, A., Bessi ere, P., and Lebeltel, O. 2004 Teaching Bayesian Behaviors to Video Game Characters. Robotics and Autonomous Systems, 47:177-- 185, 2004.
41. Lenoir, T. 2003. Programming Theatres of War: Gamemakers as Soldiers, Robert Lathma, ed, Bytes, Bandwidth, and Bullets, New York: The New Press, 2003.
42. Leonard, T. 1999. Postmortem: Looking Glass's Thief: The Dark Project, Game Developer Magazine, July 1999.
43. Lewis, M. and J.Jacobson. 2002. Game Engines in Scientific Research. Communications of the Association for Computing Machinery (CACM), NY: ACM 45(1), 2002.
44. Malenfant, D. 2000. Postmortem: Shiny's Wild 9, Gamasutra, January 07, 2000, http://www.gamasutra.com/features/20000107/wild9_01.htm (accessed 5/5/2007).
45. McGrath, D. and Hill, D. 2004. UnrealTriage: A Game-Based Simulation for Emergency Response. The Huntsville Simulation Conference, October 2004. Sponsored by The Society for Modeling and Simulation International.
46. Muñoz-Avila, H. and Aha, D. 2004. On the Role of Explanation for Hierarchical Case-Based Planning in Real-Time Strategy Games. Proceedings of ECCBR-04 Workshop on Explanations in CBR.
47. Nareyek, A., Combs, N., Karlsson, B., Mesdaghi, S., and Wilson, I. 2005. The 2005 Report of the IGDA's Artificial Intelligence Interface Standards Committee. http://www.igda.org/ai/report-2005/report-2005.html. (accessed 5/5/2007)
48. Oliveira, M., Crowcroft, J., and Slater, M. 2003. An innovative design approach to build virtual environment systems. Proceedings of the workshop on Virtual environments 2003, ACM International Conference Proceeding Series; Vol. 39, Zurich, Switzerland Pages: 143 – 151, (2003) ISBN:1-58113-686-2.
49. Ota, M. 2003. Extending the AI in a Commercial Game Engine. School of Information Technology and Electrical Engineering, The University of Queensland, 29th Oct, 2003, Bachelor Thesis.

50. Ponsen, M., Lee-Urban, S., Mu˜oz-Avila, H., Aha, D., and Molineaux, M. 2005. Stratagus: An Open-Source Game Engine for Research in Real-Time Strategy Games. Workshop for International Joint Conference on Artificial Intelligence (IJCAI-05).
51. Riehle, D., Tilman, M., and Johnson, R. 2005. Dynamic Object Model. In Pattern Languages of Program Design 5. Edited by Dragos Manolescu, Markus Völter, James Noble. Reading, MA: Addison-Wesley, 2005.
52. Robertson, J. and Good, J. 2003. Ghostwriter: a narrative virtual environment for children. Proceeding of the 2003 conference on Interaction design and children, July 2003, 85-91.
53. Rouse, R. 1999. Leaping Lizard’s Centipede 3D, Gamasutra, September 10, 1999, http://www.gamasutra.com/features/19990910/centipede_01.htm
54. Ryan, M., Hill, D., and McGrath, D. 2005. Simulation Interoperability with a Commercial Game Engine. European Simulation Interoperability Workshop 2005, 27-30 June 2005.
55. Schell, J. and Shochet, J. 2001. Designing Interactive Theme Park Rides: Lessons From Disney's Battle for the Buccaneer Gold, Gamasutra, July 6, 2001, URL: http://www.gamasutra.com/features/20010706/schell_01.htm
56. Schertenleib, S. 2006. Designing a Multilayer, Pluggable AI Engine, Game Programming Gems 6, 2006, Charles River Media, pp. 291-305, ISBN: 1584504501
57. Shumaker, S. 2002. Techniques and Strategies for Data-driven design in Game Development. http://ai.eecs.umich.edu/soar/Classes/494/talks/Schumaker.pdf
58. Smith, R. 1998. Essential techniques for military modeling and simulation. Proceedings of the 30th conference on winter simulation, 1998, 805 – 812, ISBN:0-7803-5134-7.
59. Smith, R. D. 2005. Strategic directions for distributed simulation. Simulation 2000 Series, 2, 1-9.
60. Spronck, P. 2005. Adaptive Game AI. Ph.D. thesis, Maastricht University Press, Maastricht, The Netherlands.
61. Stang, B. 2003. Game Engines Features and Possibilities. Institute of Informatics and Mathematical Modeling at the Technical University of Denmark, 2003.
62. Stanley,K., Bryant,B., and Miikkulainen,R. 2005. Real-time Neuroevolution in the NERO Video Game, IEEE Transactions on Evolutionary Computation, volume 9, number 6, pages 653-668, December 2005.
63. Tapper, P. 2003. Personality Parameters: Flexibly and Extensibly Providing a Variety of AI Opponents' Behaviors, Gamasutra December 3, 2003, http://www.gamasutra.com/features/20031203/tapper_01.shtml
64. Tong, T. 2003. Scripting in C using Co-Routines Fully Scriptable Game Logic, 8/4/2003, http://www.gamedev.net/reference/articles/article1974.asp
65. Vilhjalmsson, H. and P.Samtani. 2005. MissionEngine: Multi-system integration using Python in the Tactical Language Project, PyCon 2005, March 23-25, Washington, D.C.
66. Wang, J.; M.Lewis, and J.Gennari. 2003. Emerging areas: urban operations and UCAVs: a game engine based simulation of the NIST urban search and rescue arenas. In 35th Winter Simulation Conference. 1039-1045, 2003.
67. Wilson, K. 2003. The GDC 2003 Game Object Structure Roundtable. Tuesday, March 11, 2003, http://gamearchitect.net/Articles/GameObjectRoundtable.html
68. Wunsche, B., Kot, B., Gits, A, Amor, R., Hosking, J. and Grundy, J. 2005. A Framework for Game Engine Based Visualisations, Proceedings of IVCNZ '05, Dunedin, New Zealand, 28-29 November 2005, pp. 465-470.
69. Young, R.M., Riedl, M.O., Branly, M., Jhala, A., Martin, R.J., and Saretto, C.J. 2004. An architecture for integrating plan-based behavior generation with interactive game environments. Journal of Game Development , 1(1), 51-70.

Appendix A

Table A.1 Game Engines Survey

Seq	Game Engine	Category	*World Editor*	*Scripting Language*	*Cost*
1	Panda3D 1.2.3§	Serviced-dynamic*	√	**Python**	Free
2	Torque Game Engine 1.4‡	Serviced-dynamic*	√	**TorqueScript**	$150 - $340
3	Nebula Device 2§	Serviced-dynamic*		**Lua, Python, Ruby, TCL, etc**	Free
4	Delta3D 1.3.0	Serviced-dynamic*	√	**Python**	Free
5	Luxinia	Serviced-dynamic*	√	**Lua**	Free - €100
6	C4 Engine‡	Serviced-dynamic*	√	**Graph-based**	$100
7	CryENGINE 2	Serviced-dynamic*	√	**Lua**	
8	Crystal Space 3D 1.0 §	Serviced-dynamic*		**Python, Java, Perl**	Free
9	Unigine v0.4	Serviced-dynamic*	√	**UnigineScript**	$1495 - $19985
10	Deep Creator‡	Serviced-dynamic*	√	**Lisp**	$1,995
11	Beyond Virtual‡	Serviced-dynamic*	√	**AngelScript**	$99-$155
12	Jet3D	Serviced-dynamic*	√	**Lua**	Free
13	Sylphis 3D	Serviced-dynamic*	√	**Python**	$122
14	Lawmaker Game Engine	Serviced-dynamic*	√	**Lua**	$149.99 - $7999.99
15	Soya 3D 0.11.2	Serviced-dynamic*		**Python**	Free
16	Shark 3D	Serviced-dynamic*	√	**Perch**	
17	Qube	Serviced-dynamic*	√	**QScript**	Free
18	Stratagus 2.1	Serviced-dynamic*	√	**Lua**	Free
19	Blender 2.43	Serviced-dynamic*	√	**Python**	Free
20	Operation Flashpoint	Serviced-static	√	√	$Game
21	3D GameStudio (A6 Game Engine 6.4)‡	Serviced-static	√	**C-Script**	$49-$899
22	Virtools 4	Loaded-dynamic	√	**VSL**	$9,500
23	Unity1.5‡	Loaded-dynamic*	√	**C#,JavaScript, Boo**	$250 - $1,499
24	DOOM 3	Loaded-static	√	**SCRIPT**	$Game
	DOOM	Hard-coded-static	√		Free
25	Quake III	Loaded-static	√	**QVM files**	$Game
	Quake II	Hard-coded-static	√		Free
	Quake	Loaded-static	√	**QuakeC**	Free
26	Unreal Engine 2.5	Loaded-static	√	**UnrealScript**	$Game -$350,000
27	Power Render 6	Loaded-static	√	**AngelScript**	$150 - $8500
28	Reality Factory	Loaded-static	√	**Simkin**	Free - $149.99
29	Serious Engine 2	Loaded-static	√	**Macro**	$20,000 - $100,000
30	Quest3D 3.5.2	Loaded-static	√	**Graph-based**	$999-$9,999
31	Aurora Neverwinter Nights 1	Loaded-static	√	**NWScript**	$Game
32	TV3D SDK 6‡	Hard-coded-static			Free - $500
33	Cipher‡	Hard-coded-static	√		$100
34	3Impact‡	Hard-coded-static			$99
35	DarkBASIC Pro‡	Hard-coded-static			$89.99
36	Irrlicht	Hard-coded-static	√		Free
37	OGRE	Hard-coded-static			Free
38	Half-Life 2 (Valve Source)	Hard-coded-static	√		$Game
39	Jupiter EX	Hard-coded-static	√		$10,000 - $50,000
40	Blitz3D	Hard-coded-static	√		$100

* Uses the work around suggested in section 3.2 or an alternative technique to create the dynamic object model.
‡ One of the top 10 commercial engines cited by http://www.devmaster.net/engines/ as of 23/Feb/2007.
§ One of the top 10 open source engines cited by http://www.devmaster.net/engines/ as of 23/Feb/2007.

Table A.2 Projects Survey

Seq	Project	Engine	Location		Game State (run-time)		Object Model		Game Logic Language		Approach
			Hard-coded	Data-driven	Inside	Outside	Specific	Independent	Specific	Independent	
1	Ambush! (Diller et al, 2005)	Operation Flashpoint		√	√		√		√		
2	Tactical Iraqi (TLTS) (Vilhjalmsson and Samtani, 2005)	Unreal Tournament 2003		√	√	√	√	√	UnrealScript	√ (C++, Python, database, and xml files)	Gamebots, MissionEngine
3	UnrealTriage (first version) (McGrath and Hill, 2004)	Unreal Tournament 2004		√	√		√		UnrealScript		
4	UnrealTriage (second version) (Ryan 2005)	Unreal Tournament 2004		√	√	√	√	√Anesoft simulator	UnrealScript	√Anesoft simulator	Extended version of Gamebots
5	Urban search and rescue (Wang et al, 2003)	Unreal Tournament 2003		√	√	√(RETSINA)	√	√(RETSINA)	UnrealScript	√(RETSINA)	Gamebots
6	VRND Notre Dame (Delon & Berry, 2000)	Unreal		√	√		√		UnrealScript		
7	Efficient and Dynamic Response to Fire (Darken et al. 2004)	Unreal		√	√		√		UnrealScript		
8	Sonocard[4]	Virtools		√	√		√		√ Graphical tools		
9	Le Redoutable[5] (Blackman, 2005)	Virtools		√	√		√		√ VSL		
10	3D Driving Academy (Traffic AI & Physics engine) (Blackman, 2005)	3D GameStudio (A6 engine)		√	√		√		C-Script		
11	Information and Decision-Making (Creel et al, 2006)	Neverwinter Nights Aurora Engine		√	√		√		NWScript		
12	Mimesis Virtual Aquarium (Young et al, 2004)	Unreal		√	√	√	√	√	UnrealScript	√	Mimesis
13	PSDoom (Chao, 2001)	Doom	√		√	√	√	√	√	√	
14	Visualisation Tools (software Visualization tool and a biomedical visualisation tool) (Wunsche	Quake 3	√	√	√		√		Shader script		

[4] http://www.virtools.com/applications/simulation-enteccs.asp (accessed 1/3/2007)
[5] http://www.virtools.com/applications/simulation-redoutable.asp (accessed 1/3/2007)

Seq	Project	Engine	Location		Game State (run-time)		Object Model		Game Logic Language		Approach
			Hard-coded	Data-driven	Inside	Outside	Specific	Independent	Specific	Independent	
	et all, 2005)										
15	Flying Mutator (Ota, 2003)	Unreal		√	√		√		UnrealScript		
16	VU-Life 2 (Eliens & Bhikharie, 2006)	Half-Life	√		√		√			√	
17	Creating and Visualising an Intelligent NPC using Game Engines and AI Tools (Davies et al, 2005)	Unreal		√		√		√		√	Gamebots
18	Stratagus: An Open-Source Game Engine for Research in Real-Time Strategy Games (Ponsen et al, 2005)	Stratagus		√		√		√		√	TIELT
19	Neverwinter Nights Game AI (Spronck, 2005)	Neverwinter Nights Aurora Engine		√	√		√		NWScript		
20	Wargus Game AI (Spronck, 2005)	Wargus		√	√		√			Lua	
21	Flexible and Purposeful NPC Behaviors using Real-Time Genetic Control (Hussain & Vidaver, 2006)	Neverwinter Nights Aurora Engine		√		√		√		√	Shadow Door + ACTB-NWN bridge
22	Interacting with Virtual Characters in Interactive Storytelling (Cavazza et al, 2002)	Unreal		√		√	√	√	UnrealScript	√C++ planner	
23	Qualitative Physics In Virtual Environments (Cavazza et al, 2004)	Unreal		√	√	√	√	√	UnrealScript	√QP Simulation module	
24	Extending Game Participation with Embodied Reporting Agents (Fielding et al, 2004)	Unreal		√		√		√	UnrealScript	√	Gamebots
25	Ghostwriter (Robertson and Good, 2003)	Unreal		√	√		√		UnrealScript		
26	America's Army third-person	Unreal		√	√		√		UnrealScript		

Seq	Project	Engine	Location		Game State (run-time)		Object Model		Game Logic Language		Approach
			Hard-coded	Data-driven	Inside	Outside	Specific	Independent	Specific	Independent	
	perspective helicopter physics (Davis et al, 2004)										
27	NERO project (Stanley et al, 2005)	Torque		√	√		√		TorqueScript		
28	The Minority Game (Heckenberg et al, 2004)	Unreal Tournament 2003		√	√		√		UnrealScript		
29	Explanation for Hierarchical case-based planning (Muñoz-Avila & Aha, 2004)	Stratagus		√		√		√		√	TIELT
30	Hamlet (Hunicke & Chapman, 2004)	Half-life	√		√		√			√	