BSc Final REPORT

Submitted for the
BSc Honours in Computer Science with Games Development

May 2009

Particle System for the PSP

by

Jamie Sandell

Acknowledgements

I would like to thank Dr Jon Purdy for his support and feedback on my Initial and Interim Reports, and, all of my third year project work in general.
I would also like to thank Dr Warren Viant for his feedback on my Interim Report.
I would also like to thank Jeremy Fitzhardinge (who wrote the PSPGL graphics library for the PSP), for his answers to various questions (via e-mail) pertaining to the PSP and PSPGL.
I would also like to thank Stuart James and Kenny Chung (fourth year students who are, as of writing this report, studying Msc Computer Science with Games Development), whose work on the psp (most notably the general framework they have setup and developed), has helped me greatly. Also I would like to thank them both for the answers they have given me to the PSP and third year project related questions I had.

Indexing Information

Title:
Particle System for the PSP
Author:
Jamie Sandell
Abstract:
This document covers the aspects of what a particle system actually is, how best to create them, what they are useful for representing, efficiency issues regarding particle systems in general, and, efficiency issues of the PSP (with regards to a particle system).

Contents

1 – Report Introduction and Project Introduction

1.1 – Report Introduction

The purpose of this report is to clearly outline what the entitled ‘Particle Engine for PSP’ project is about, and to report the progress so far, highlighting any problems and/or setbacks that have thus far occurred. The report highlights and discusses: the initial specification, any changes and modifications made up to this point, the aims, background context, development chapters, review and conclusion chapters, bibliography/references, and appendices of the project.

1.2 - Project Introduction

Particle effects are a staple of any modern computer game. Whilst particle effects themselves don’t ‘make or break a game’; what they can do is to add a great atmospheric resonance to a scene. A few examples highlight this perfectly; imagine a game cut-scene where a spaceship is hurtling through space, now, the spaceship could be hurtling through space without displaying the thrusters that hurtle it, or, the spaceship could be hurtling through space, and be displaying the thrusters (created through a particle system) that hurtle it. Now compare the two, and the difference is definitely most notable.
Another good example that demonstrates the impact that a particle effect can have is spell effects in a game. Take any decent role-playing game, such as Neverwinter Nights 2 (see the background reference entry for more information), now, in Neverwinter Nights 2 the casting of spells is abundant throughout the duration of the game. These spell effects are produced and drawn using particle systems and they bring the spell to life, and as such these particle effects impact the game significantly in a positive manner. Please see Appendix V to see a picture of such a spell effect in Neverwinters Nights 2.

1.3 – Original Specification

Below is the original specification that was presented to me when choosing my third year project, it has remained the same since.

Project Code: JHP1
Title: Particle Engine for PSP.

Specification:
This project is to produce a particle engine and tool for use on the

PSP development kit. This will require the student to use an industry standard PSP development kit and SDK. The project is suitable for a proficient programmer and is ideal for someone hoping to work in the games industry.

2 – Project Aims

The ultimate aim and goal of the project is to design and create a fully working particle engine and tool for the Sony PSP development kit. The first piece of analysis that must be done is to actually discover not only just what a particle system is and how a particle system works, but, also, why such a system is required. On-top of this, the question - *how can the project be incorporated onto the Sony PSP and how can it run efficiently on the hardware that the Sony PSP offers* must be addressed. A broad overview of how this can be achieved must be supplied. Finally any alternatives to achieving the intended project must be touched upon, and if necessary, explained in further detail.

3 – Analysis

This project's objective is to create a software solution to allow Sony PSP game developers to quickly create and edit particle effects for use in Sony PSP games. The third year project software I have developed and created will allow the Sony PSP game developer to quickly create and edit particle effects without the need to hard code any special effects, i.e. the various particle effects.
For this third year project (with the analysis in mind) I intend to create the project split into two separate software application parts. The first part will be the Particle System Editor, which will be the development suite for the Sony PSP game developer and will create and edit various particle effects. The second part will be the Sony PSP software application, basically a template (Sony PSP code file ready to be compiled into a Sony PSP console executable) that will load in the various particle effects created via the Particle System Editor software application.
The two pieces of software will be coded in C++; the PSP software template will use the PSPGU graphics library, and the Particle System Editor will use the PSPGL graphics library.

4 – Similar Software

I have found two software applications that are have similar qualities to that of my third year project software application. These software applications are, Delta3D Graphical Particle System Editor and Particle System Editor (please see the background references *Delta3D* and *Particle System Editor* for more information).
The Delta3D Graphical Particle System Editor software application installation file can be found in the following directory:
.\Files\SimilarSoftware
and is named:
dt_win32_REL-2.2.0_setup.exe.
The Particle System Editor software application can be found in the following directory:
.\Files\SimilarSoftware\game317925\Particle System Editor
and is called:
ParticleSystemEditor.exe

4.1 – Delta3D Graphical Particle System Editor

The Delta3D Graphical Particle System Editor is similar to the Particle System Editor I created, in the sense that it indeed allows the user to easily create and edit various particle effects. It too is GUI based like my project's software application. It also allows the user to customize the particle effect in the following ways:
texture, alignment, shape, life, radius, mass, size, colour, emitter, minimum particles, maximum particles, loop emitter forever, run for a set amount of time; which are all customizations that my editor provides.
The Delta3D Graphical Particle System Editor allows for customization that my particle system editor software does not, such customizations as:
Initial rotational velocity range, operators (such as new force, new acceleration, and new fluid friction).

4.2 – Particle System Editor

The Particle System Editor, like the Delta3D Graphical Particle System Editor, is similar to the Particle System Editor I have created as part of my third year project. The Particle System Editor is similar to mine because it also provides the functionality of allowing the user to easily create and edit various particle effects. Like my Particle System Editor and the Delta3D Graphical Particle System Editor it too is GUI based. The Particle System Editor has the following customizations in common with my own Particle System Editor:
textures, maximum particles, batch size (particles), emit rate, blending (though it has

more options than mine), min spawn position, max spawn position, gravity, minimum particle colour, maximum particle colour, particle life (minimum and maximum), and speed (minimum and maximum).
The Particle System Editor has some functionality and features that the project's Particle System Editor does not, these features are:
ground texture, draw ground plane, and wireframe.

4.3 – Similar Software Conclusion

Overall it is evident that the Particle System Editor software application is superior to the Delta3D Graphical Particle System Editor software application. This is because visually the Particle System Editor is more visually appeasing then the Delta3D Graphical Particle System Editor (see Appendix VII and Appendix VIII). In conjunction with this, the Particle System Editor not only looks better, in comparison, to the Delta3D Graphical Particle System Editor software application, but, it is also more efficient. This was proven by the Particle System Editor running at a higher framerate than the Delta3D Graphical Particle System Editor, whilst producing and displaying more particles (with a better image quality).

5 – Background Context

5.1 – What is a Particle System?

A particle system is a "method for modeling fuzzy objects such as fire, clouds, and water. Particle systems model an object as a cloud of primitive particles that define its volume." (Reeves, 1983, p 1).

5.2 – How a Particle System Works

There are many different approaches to a particle system, however, they all work on the same underlying basic principles, i.e.:
The particle is generated by an emitter (see Appendix II)
It's new position is calculated
Render the particle
If it's at the end of its lifespan it is removed
If appropriate it is reinitialised.

It is how the particle engine does this that makes the effects very different from a

different effect, for example, an explosion effect compared against a rain effect, these effects are completely different but are in fact created using the exact same principles. For instance the emitter can be changed to any emitter type wanted. The calculation of the particle's new position can involve physics based simulation or stochastic ("*The most common choice is to implement some kind of dynamics to the particles so they mimic real-world phenomena.*"(Dalmau)). Then the particle engine renders the particle, this can be anything that the graphics library supports, e.g. a point or a triangle strip.

5.3 – Why a Particle System?

Why would someone need to program a formal algorithm to create such visual effects, why can't conventional tools (e.g. Maya 3D) be used to create such an effect and animate it? Well this is answered and explained perfectly by Sabo in his article, he states, "*Particles making up the system are non-static unlike standard geometry objects. Particles are born; they change over time, and then die off. A key point regarding particle systems is that they are chaotic. Instead of having a completely predetermined path, each particle can have a random element, called a stochastic component, which modifies its behavior. This random element is, in fact, the main reason why particle systems are so good at reproducing realistic effects.*"(Sabo).

5.4 – Porting to the PSP

Porting the project to the PSP will not be too difficult; however, converting some graphical commands (OpenGL) will require converting the commands to their equivalent using a graphical library on the PSP.
The biggest challenge with porting the engine will be getting it to run on the PSP efficiently, it must be kept in mind that the PSP is a handheld device, and as such, when compared to a cutting edge PC, or even an average PC it is lacking in hardware capabilities (See Appendix I). This actually poses a big challenge because a particle system can be displaying literally thousands of particles on the screen at any given timeframe, and not only just drawing them but also calculating their positions.

5.5 – Efficiency

5.5.1 – Inlining

Efficiency is a big issue for the project deliverable (see 3.4 – Porting to the PSP). One way to keep the program efficient is to make use of inlining where appropriate, such as inlining singleton methods (methods which are only called from one place in the program), because that will not increase the program size but will increase the

program speed. Also small methods, such as accessors will be inlined, however excessive inlining can dramatically increase the execution speed of the program because of a resulting lower cache hit rate. For methods that aren't too obvious whether or not they should be inlined test samples of the code will be ran, timing inlined and non-inlined versions.

5.5.2 – Virtual Methods/Templates

Virtual methods will be avoided because as Giplin clearly highlights:
"Virtual functions negatively affect performance in 3 main ways:

The constructor of an object containing virtual functions must initialize the vptr table, which is the table of pointers to its member functions.

Virtual functions are called using pointer indirection, which results in a few extra instructions per method invocation as compared to a non-virtual method invocation.

Templates can be used to avoid the overhead of virtual functions by using a templated class in place of inheritance. A templated class does not use the vptr table because the type of class is known at compile-time instead of having to be determined at run-time. Also, the non-virtual methods in a templated class can be inlined.
The cost of using virtual functions is usually not a factor in calling methods that take a long time to execute since the call overhead is dominated by the method itself. In smaller methods, for example accessor methods, the cost of virtual functions are more important."

5.5.3 – Constructors/Destructors

Giplin also makes a very valid efficiency point on Constructors and Destructors:
"The performance of constructors and destructors is often poor due to the fact that an object's constructor (destructor) may call the constructors (destructors) of member objects and parent objects. This can result in constructors (destructors) that take a long time to execute, especially with objects in complex hierarchies or objects that contain several member objects. As long as all of the computations are necessary, then there isn't really a way around this. As a programmer, you should at least be aware of this "silent execution".

If all of the computations mentioned above are not necessary, then they should be avoided. This seems like an obvious statement, but you should be sure that the computations performed by the constructor that you are using are doing only what you need.
Objects should be only created when they are used. A good technique is to put off object creation to the scope in which it is used. This prevents unnecessary

constructors and destructors from being called.
Using the initializer list functionality that C++ offers is very important for efficiency. All member objects that are not in the initializer list are by default created by the compiler using their respective default constructors. By calling an object's constructor in the initializer list, you avoid having to call an object's default constructor and the overhead from an assignment operator inside the constructor. Also, using the initializer list may reduce the number of temporaries needed to construct the object."

5.5.4 – Avoiding New and Delete

It is a well known fact that allocating and freeing memory is an extremely slow process. Therefore it is imperative that, the re-spawning and killing of particles algorithms are scrutinised in-depth and made as efficient as possible. Dalmau explains that "*freeing memory each time a particle dies (or allocating memory for newborn particles) can have a significant performance hit, especially in those systems with large particle numbers or short life spans."*
The solution is to use arrays, because arrays provide the most linear access to the available data. Accessing the data linearly helps with caching, and as such, speeds up the code; however, there is a problem with this solution, and that is, reallocating arrays can be costly. The way to handle this problem to the solution is, implement some kind of memory management. How this will work is, initially, a single large block of memory will be allocated. I will store a pointer to the beginning of the allocation, a pointer to the very end of the allocation, and finally, a pointer to the next free space in the array.
For example:
After the block of memory has been allocated, both the start and end pointers will point to the start of the block since it's empty. When a new element is added to the array, it is added where the last pointer is currently referencing, and then moves the last pointer to the next memory location.
As a result, if we want to remove an item from the array, then, all that has to be done is to simply copy the last element over the element to be removed, and move the last pointer back an item in the array.
To surmise, using this technique all particles are maintained tightly packed at the beginning of the array with no gaps. In addition adding and removing particles is possible without the overhead of memory allocations. One slight problem is that we must not go above the static array allocation, but that is a small price to pay for the huge efficiency gain.

5.5.5 – VertexArrays

In OpenGL there is a standard drawing technique which is:
glBegin(<PrimitiveType>);
glEnd();
However this is an inefficient method for drawing thousands of particles. A much more efficient drawing routine is to make use of glDrawArrays. Since the particles are all nicely packed into one place, vertex arrays can be used to render the data. To do this we need to give the address of the first data element in the array. We also need to specify a stride parameter, i.e. the number of bytes between each data element. In this case, each position is seperated by another particle. The same is true of the colours.

5.6 – Alternatives to Achieving the Goal

Classically there are two methods of achieving the goal, the first is to code the particle system as a class hierarchy with ParticleSystem as an abstract class and new particles effects (e.g. fire) deriving via inheritance from this parent.
The second alternative is to "*create a deep, complex particle system class, where individual systems are nothing but data-driven objects that parameterize the class in many ways.*" (Dalmau).
The plan of action for the project is to actually utilise both methods to achieve the project's goal. The second method will be utilised first to create the editor (which will be used on a PC); this method is chosen for the editor because once an effect has been edited it will be easy to update the display to show the new particle effect (e.g. a switch statement to select the kind of emitter to use). The editor will then write the effects to a new file which will be in the form of a specified class framework.
The first method will be the one actually used on the PSP to show the particle effects in action. This method has been chosen for this task because when in a game environment, or any big project environment, it is best to use an Object-Orientated approach as it becomes more manageable and easier to understand (easier to abstract and view high-level concepts).

5.7 – Ethical Issues

5.7.1 – Process

This project and all its operations are been conducted in an ethical manner. The IPR of others' ideas have been respected greatly, any influences have been correctly citated.
I have been honest in assessing and reporting my progress and in future planning of the project.
I have given full compliance with recognised procedures.

5.7.2 – Product

My product can potentially affect my professional standing, if it fails then it could, conceivably, be the failing point of my degree. On the flip-side if it succeeds then it can only complement my professional standing, e.g. achieving my degree.
There are no reliability/safety issues for users/clients/society at large.
Legitimacy of actions available through the proposed system/project is wholly legal and ethical.

6 – Proposed Solution

The proposed solution must include the following efficient algorithms:

- Initialise Particles
    - This algorithm must initialise the particle's position based on the selected emitter, and then the algorithm must initialise the colour based on what the Sony PSP Game Developer has selected.
- To String
    - This algorithm must convert a type (for example, int, double, float, byte) to a string type.
- Set Particle Type
    - This algorithm must set the type of the particle.
- Set Size
    - This algorithm allocates enough memory for the amount of possible particles.
- Seconds Spinner Changed
    - This algorithm must reinitialise the particles once the seconds spinner has changed by the Sony PSP Game Developer.
- Maximum Particles Spinner Changed
    - This algorithm must, (just like the seconds spinner changed algorithm) reinitialise the particles once the maximum particles spinner has been changed by the Sony PSP Game Developer.
- Write PSP File
    - This algorithm must write the current particle effect to a file that can be read by the PSP Software Code Template (that is included in the distribution of this third year project).
- Save Settings
    - This algorithm must save the current particle effect settings to a file.
- Load Settings
    - This algorithm must load the saved particle effect settings from a file.
- Is Empty

    - This algorithm determines whether or not there are currently any particles.
- Is Full
    - This algorithm determines whether or not the memory allocated for the particles is full (currently occupied by the particles).
- Draw Particles
    - This algorithm must determine what geometry to draw to represent the particles and whether or not textures are used and what blending mode to use.
- Get Number Of Particles
    - This algorithm must be able to determine the number of currently active particles.
- Add One
    - This algorithm must add a particle if, and only if, there is enough free memory (from the pool of memory that was originally allocated for the particles) to do so.
- Erase One
    - This algorithm simply ‘erases’ a particle, it must not use new and delete to achieve this, as this would be a significant detriment to the program’s speed and efficiency.
- Update Particles
    - This algorithm must update all of the properties of the particle accordingly. Such particle properties as position, colour and life.
- Add And Initialise Particles
    - This algorithm must be used before any update or draw algorithms are executed to add the first batch of particles and initialise them.

The proposed solution must also utilise the PSPGU graphics library and the OpenGL graphics library, and the proposed solution must also be coded in C/C++. The first part of the proposed solution, the Particle System Editor, must be written as one class and be heavily parameterised (easier for changing the various options to create new effects quicker). The second part of the proposed solution, the Sony PSP Software Code Template file, must be written as a class and when particle effects are read in they are created as child classes. This must be done to ensure that the various particle effects that are created and wanted to be used can be done so in an organised and collectively object orientated manner.

7 – Project Design

7.1 – Basic Design

The software section of this third year project is split into two (explicit) components; those two components are the Particle System Editor, and the actual PSP Particle System. The Particle System Editor creates and edits various particle systems; the PSP Particle System is essentially a single file whose code is converted from OpenGL to PSP code. The PSP Particle System is the component that runs the desired particle system on the Sony PSP console.

7.2 – The Components Abstract

7.2.1 – Particle System Editor Abstract

The Particle System Editor is essentially the crux of the project. The Particle System Editor is a GUI program coded in OpenGL, and utilises the GLUI library; which is an OpenGL library that enables easily deployed visual controls, such as, editable text boxes and numeric vertical spinners.

7.2.2 – PSP Particle System

The PSP Particle System is a PSP application that loads a file (created via the Particle System Editor) and converts the saved information into a particle effect that is then displayed on the Sony PSP's display window.

7.3 – Choice of Language, Additional Libraries, and No Longer Used Libraries

7.3.1 – Choice of Language

It had been stated, outright, in the Project's description that the project must be coded in C++. This makes perfect sense because C++ is not only object-orientated (essential for high level concepts), but is incredibly fast. The reason why this is so important is because the saved particle effect needs to run on a PSP, thus any speed and efficiency benefits are incredibly welcome. On-top of this, the PSP code compilation tools compiles C/C++ code; therefore it makes perfect sense that the choice of language is C++.

7.3.2 – Additional Libraries

OpenGL

OpenGL is used for the Particle System Editor because I have previous experience with creating graphic effects and various other projects using OpenGL as the graphics library.

OpenGL is also very similar to PSPGU (the PSP graphics library I have used on the PSP). The two codes samples (of drawing a vertex array) below show the similarities between the two graphics libraries.

OpenGL

```
glVertexPointer(3,GL_FLOAT,sizeof(ParticlePointsVertex),startPointsVertex>position
);
glColorPointer(4,GL_FLOAT,sizeof(ParticlePointsVertex),startPointsVertex->colour);
glDrawArrays(GL_POINTS,0,lastPointsVertex-startPointsVertex);
```

PSPGU

```
sceGumDrawArray( SCEGU_PRIM_POINTS,
    VER_FORM,
    sizeof(ParticlePointsVertex),
    NULL,
    startPointsVertex->position);
```

PSPGU

PSPGU was chosen as the main graphics library on the PSP (opposed to PSPGL) because the commands, as stated above, are very similar to OpenGL, which I already have experience with through various past projects and lecture materials from various past modules that have been undertaken during my time at the university.

GLUI

GLUI is an OpenGL library that allows a developer to quickly develop and produce a windows form application, development isn't as fast as creating a windows form in Visual Studio C# .Net for example, which is highlighted by MSDN, *Visual Basic and C# both have RAD (rapid-application development) support in Visual Studio .NET, with project templates, designers, and other features of the development environment. Both languages use the .NET Framework base classes.*
However, even with that in mind it was decided to use GLUI as it was written in C++, and therefore, would be more efficient (processing time wise) when compared against the use of OpenGL in C# .Net.

7.3.3 – No Longer Used Libraries

PSPGL

PSPGL was initially chosen as the graphics library to use when writing for the PSP. This is because it essentially uses OpenGL commands, and I have experience with

OpenGL through various projects and modules undertaken. PSPGL is essentially a basic PSPGU (PSPGU is the official graphics library for the PSP) interpreter, meaning that all PSPGL commands are sent/converted to PSPGU so there is a slight efficiency issue, but it isn't great.
The use of the PSPGL graphics library was dropped for two reasons. The first was due to the fact native PSPGU was more efficient then PSPGL. The second is that the PSPGU commands are similar to the PSPGL (OpenGL) commands, in some cases "*gl-something"* could be replaced with "*sceGu-something"*. Therefore converting the PSPGL code to PSPGU was not too difficult and allowed for an efficiency gain.

7.4 – User Interface

The solution will be made in two parts (as previously stated) and thus will have two user interfaces, one user interface for the Particle System Editor software application, and, one user interface for the Sony PSP Code Template file.

7.4.1 – User Interface for the Particle System Editor Software Application

The user interface for the Particle System Editor Software Application will be a graphical user interface that the Sony PSP Game Developer can interact with via the mouse and keyboard. On the left will be a form control, and on the right will be a preview window that shows the current particle effect. The table below highlights the components of the graphical user interface for the Particle System Editor Software Application:

Form Control Name	*Form Control Type*
Load Settings	Button
Save Settings	Button
Write PSP File	Button
Preset Properties	Panel
Presets	Drop down menu
Emitter Properties	Panel
Shape	Drop down menu
Height	Spinner
Width	Spinner
Depth	Spinner

Position X	Spinner
Position Y	Spinner
Position Z	Spinner
Position X Min	Spinner
Position X Max	Spinner
Position Y Min	Spinner
Position Y Max	Spinner
Position Z Min	Spinner
Position Z Max	Spinner
Maximum Particles	Spinner
Duration in Seconds	Spinner
Repeat	Check box
Particle Properties	Panel
Shape	Drop down menu
Height	Spinner
Width	Spinner
Depth Spinner	Spinner
Blending Mode	Drop down menu
Start Colour Red	Spinner
Start Colour Green	Spinner
Start Colour Blue	Spinner
End Colour Red	Spinner
End Colour Green	Spinner
End Colour Blue	Spinner
Random Colour	Check box
Gravity X	Spinner

Gravity Y	Spinner
Gravity Z	Spinner
Speed X Min	Spinner
Speed X Max	Spinner
Speed Y Min	Spinner
Speed Y Max	Spinner
Speed Z Min	Spinner
Speed Z Max	Spinner
Fade Minimum	Spinner
Fade Maximum	Spinner
Quit	Button
Particles	Display window

7.4.2 – Sony PSP Code Template File

The Sony PSP Code Template File (that is distributed with this third year project), will not have a Graphical User Interface, but will have a User Interface instead. The User Interface will allow the user to quit from the software application via the ‘Start’ button on the Sony PSP Console Device.

7.5 – In-Depth Design

7.5.1 – Problem Statement

Particle Systems are a part of every modern game, and are a great way of cheaply (little processor computational costs in comparison to other components of modern games) depicting fluid, non-rigid, natural phenomena such as fire, water, snow, and rain. The problem is that whilst particle systems can be created with a minor CPU usage hit on modern PCs, they could potentially struggle tremendously on a PSP (see Appendix I for the system specification of the PSP). Thus the problem is two-fold, the first is creating a flexible particle system that is efficient on the PC, and two converting the code to something the PSP can run, and that can be ran efficiently at a smooth animation rate.

7.5.2 – Business Goals

Below are the business goals of the project, which are arranged according to priority (either a high priority, or a low priority):

High

- Create a Particle System Editor on the PC that can modify current settings to create a new Particle Effect.
- Create a PSP template class that loads in a particle effect created by the Particle System Editor and displays the particle effect on the PSP's display screen.

Low

- Make sure the Particle System Editor on the PC is intuitive and non-cumbersome to operate.

7.5.3 – Project Risks and Severity of Risk Table

Project Risks

Here are the potential risks that could happen to the project, or could happen because of the project, and their corresponding risk avoidances:

Hardware Failure
Hardware failure is an incredibly severe risk that could affect the project. Potentially, if hardware failure occurred, I could lose all of the work done on the project. To avoid this risk I systematically created backups of my work in various places, e.g. at home on to a usb drive, an external drive, at the university campus onto my g drive.

Missing The Final Deadline
Missing the final deadline is also a very severe risk that could affect the marks of my project, this is because if the final deadline was missed then I would very quickly lose potential marks. I have avoided this risk by attempting to keep on top of my work by sticking to my timeplan (see Appendix III and Appendix IV), and by seeking any help and assistance that I needed.

Cognitive Shortfall
Cognitive shortfall, in respect to my project work, is the failure to understand something that is required for this project, e.g. how to code a particle system. To avoid this problem initial background research has been done, and seeking the advice and help of others when needed has been done.

Personal Injury

If something were to happen to me, injury wise, the project would suffer, if the injury was bad enough then the project could be effected to such a degree that it meant a sub-par project or one that failed entirely all together. However, with that in mind, I feel the actuality of such an injury to be highly unlikely, and, therefore it is a risk of low significance.

Incompatibility Issues

If the program has compatibility issues then the project will have a smaller potential user base, which will reduce its usefulness. This is an unlikely situation but not one that is impossible to occur. It isn't impossible to occur, this is because the Sony PSP console already has different firmware versions and will continue to have more. With each firmware version the project's software applications will have to be (potentially) updated and modified to work with each different Sony PSP Console's firmware.

Loss of Data

This is a risk of medium severity, because losing data will certainly add further setbacks to the projects. To avoid this risk of surfacing and affecting my third year project I have ensured backups are regular and in sync with my main work folder.

Severity of Risk Table

Table Legend:

- Low = 1
- Medium = 2
- High = 3+
- Significance = Likelihood x Severity

Risk	Severity	Likelihood	Significance
Hardware Failure	High	Low	High
Missing The Final Deadline	High	Low	High
Cognitive Shortfall	Medium	Low	Medium
Personal Injury	Medium	Low	Medium
Incompatibility Issues	Low	Medium	Medium
Loss of Data	Medium	Low	Medium

7.6 – System Scope

The proposed system will be split into two distinct parts, the PC side which will be the Particle System Editor, and the Sony PSP side which will be the Sony PSP software template code file.
The Particle System Editor will allow the Sony PSP Game developer to perform the following particle customizations:
shape, height, width, depth, blending mode, start colour red, start colour green, start colour blue, end colour red, end colour green, end colour blue, random colour, gravity x, gravity y, gravity z, speed x min, speed x max, speed y min, speed y max, speed z min, speed z max, fade minimum, fade maximum.
The Particle System Editor will also allow the Sony PSP Game developer to perform the following emitter customizations:
shape, height, width, depth, position x, position y, position z, position x min, position x max, position y min, position y max, position z min, position z max, maximum particles, repeat.
The Particle System Editor will also allow the Sony PSP Game developer to perform the following functions:
Load Settings, Save Settings, Write PSP File, and select from a set of Preset Properties such as ‘Snow’ and ‘Rain’ for example.

7.7 - System Vision

The proposed system will provide a simple and easy to use way for a Sony PSP Game Developer to design new particle effects, to edit particle effects, and save particle effects. The proposed system will also allow the Sony PSP Game Developer to display these particle effects on the Sony PSP console.

8 – Requirements

8.1 – Functional Requirements

The user should be able to create, edit, and save particle effects via the Particle System Editor for the PSP.
The user should be able to take a saved particle effect and insert it into their PSP application, they won’t necessarily have to fully understand how the particle effect works because a PSP application template will be included in the project deliverable.
The PSP application template that is included with the project deliverable will load a saved particle effect (that was also included in the project deliverable) automatically.

8.2 – Task List

#1 Initial Report
The initial report is a report outlining the ultimate aims and goals of the project and also contains a project plan, timescale, appendices and bibliography/references.
Duration – 9 days.

#2 Research and Design
Continue with background research and design the system at a high level.
Duration – 3 days.

#3 Prototyping
Produce as many prototypes of the project software as necessary, and assess the prototypes internally and externally.
Duration – 1 day.

#4 Design
Go back to the initial design document/(s) (which were present in the initial report documents), and alter the design document/(s) according to feedback and current project development status.
Duration – 1 day.

#5 Prototyping
Alter the prototype/(s) accordingly.
Duration – 1day.

#6 Unit Implementation
Code all of the necessary units.
Duration – 42 days.

#7 Unit Testing
Test all units extensively (white box and black box testing).
Duration – 42 days.

#8 Unit Integration
Integrate all of the necessary units.
Duration – 5 days.

#9 Integration Testing
Test the unit integration extensively, testing should entail white box and black box testing.
Duration – 5 days.

#10 Interim Report
The interim report is essentially a progress review of the project as a whole, including any and all revisions, updates, and modifications that need addressing at

this stage of the project.
Duration – 14 days.

#11 Final Report
The final report is an all encompassing document of everything that I have done and achieved during my final year project. It also provides a personal review and critical appraisals of the project and begins to discuss possible future works of the project. The final report also includes appendices, and references/bibliography.
Duration – 60 days.

#12 Project Appraisal
The project appraisal is an honest review of how successfully I have managed my project and my time.
Duration – 14 days.

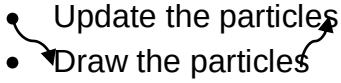
#13 Presentation
Preparing and executing the presentation of my final year project.
Duration – 28 days.

9 – Algorithms

9.1 - Software Application Algorithm Brief

This is the Particle System Editor and Sony PSP Template Code File algorithm in brief:

- Initialise the particles
- Update the particles
- Draw the particles



How the Particle System Editor works is a single large allocation of memory is created (big enough to hold the maximum amount of particles), a pointer is created to point to the beginning of the memory allocation, let's call this pointer startPoints. A pointer is created to point to the end of the memory allocation, let's call this pointer endPoints. Finally a pointer is created to point to the next free space in the array, let's call this pointer lastPoints. So, for example, initially after the block of memory for the particles has been allocated both the pointers startPoints, and, lastPoints point to the start of the block since it's empty. Then when a particle is created, we add it where lastPoints is currently referencing and then move lastPoints to the next free memory location. After a while the array of particles fills up and we want to remove some particle from the array. To do this all that is needed to do is to simply copy the last element of the array over the element to be removed and move lastPoints back an item in the array. Using this technique described all particles remain tightly packed at the beginning of the array with no gaps. Also particles can be removed without the overhead of memory allocations such as new and delete.

9.2 – In-depth Algorithm Analysis

9.2.1 - Initialise Particles

Here is the prototype of the method:

```
template<typename T>
void InitParticles(T *&inStartVertex, T *&inLastVertex, int startingPoint)
```

The algorithm doesn't return anything. The method is a template method, this is to enable different types (structs) of particles to be able to utilise this algorithm. The inputs of the method are: inStartVertex, inLastVertex, startingPoint. inStartVertex points to the start in the memory allocation of the particles' vertex data. inLastVertex points to the last active particle in the memory allocation of the particles' vertex data. startingPoint tells the algorithm how many places to offset from the point in the particle memory allocation supplied by inStartVertex.

```
T* p = inStartVertex+startingPoint;
```

The above line of code creates a pointer (of the type that called the method) that points to the first particle that needs initialising.

```
switch (emitterShape)
```

Then, based on the shape of emitter selected by the Sony PSP Game Developer, all of the particles that need initialising, have their positions initialised, e.g.

```
p->position[0] += rand() % (emitterXMax - emitterXMin + 1) + emitterXMin;
```

Once that has completed the particles' colour is initialised, if the Sony PSP Game Developer ticked the 'Random Colour' checkbox then a random colour is applied:

```
p->colour[0] = rand() % (255 - 0 + 0) + 0;
p->colour[1] = rand()% (255 - 0 + 0) + 0;
p->colour[2] = rand()% (255 - 0 + 0) + 0;
```

Otherwise the colour is set according to what the Sony PSP Game Developer specified (via the Particle System Editor GUI):

```
p->colour[0] = particleStartColour[0];
p->colour[1] = particleStartColour[1];
p->colour[2] = particleStartColour[2];
```

9.2.2 – ParticlePointsState/ParticleLinesState/ParticleTrianglesState

The algorithms 'ParticlePointsState', 'ParticleLinesState', and 'ParticleTrianglesState' are very similar to eachother. Each is a constructor for the corresponding structures, and holds and initialises the state information for the particle, i.e. direction, fade, life and speed.

The algorithm works like this:

The initial direction of the particle is set to a random value, this is done ensure that there is a randomness to the effect so it looks less-programmed and more natural.

```
this->direction[0] = (10000 - rand()%20000)/10000.0f;
this->direction[1] = (10000 - rand()%20000)/10000.0f;
this->direction[2] = (10000 - rand()%20000)/10000.0f;
```

Then the other state properties of the particle are set with a randomness too:

```
this->fade = rand () % (int(particleMaximumFade) - int(particleMinimumFade)
+ 1) +int(particleMinimumFade);
this->life = rand()%int(this->fade)/9500.0f;
this->speed[0] = rand () % (int(particleSpeedMax[0]) -
int(particleSpeedMin[0]) + 1) +
            int(particleSpeedMin[0]);
```

```
this->speed[1] = rand () % (int(particleSpeedMax[1]) -
int(particleSpeedMin[1]) + 1) +
            int(particleSpeedMin[1]);
this->speed[2] = rand () % (int(particleSpeedMax[2]) -
int(particleSpeedMin[2]) + 1) +
            int(particleSpeedMin[2]);
```

9.2.3 – Set Particle Type

Here is the method prototype:

```
template<typename T, typename Y>
void SetParticleType(T *&inStart, T *&inLast, T *&inEnd, Y *&inStartVertex,
Y *&inLastVertex,
                                    Y *&inEndVertex)
```

The method is a template method, this is so particles of different types can call it and make use of its algorithm. The parameters are: inStart – a pointer to the start of the memory allocation of the particles' state information, inLast - a pointer to the last active particle in the memory allocation of the particles' state information, inEnd - a pointer to the end of the memory allocation of the particles' state information, inStartVertex – a pointer to the start of the memory allocation of the particles' vertex information, inLastVertex - a pointer to the last active particle in the memory allocation of the particles' vertex information, inEndVertex – a pointer to the end of the memory allocation of the particles' vertex information.
The algorithm sets the type of particle, e.g. ParticlePoints, ParticleLines, and ParticleTriangles. It does this by creating new pointers (of the type specified through the method parameters) and then setting the pointers passed in through the method to point to these:

```
T* start = 0;
T* last = 0;
T* end = 0;
inStart = start;
inLast = last;
inEnd = end;

Y* startV = 0;
Y* lastV = 0;
Y* endV = 0;
inStartVertex = startV;
inLastVertex = lastV;
inEndVertex = endV;
```

9.2.4 – Set Size

This is the method prototype for SetSize:

```
template<typename T, typename Y>
void SetSize(unsigned int size, T *&inStart, T *&inLast, T *&inEnd, Y
*&inStartVertex, Y *&inLastVertex,
                  Y *&inEndVertex)
```

The SetSize method is a template method, once again this is so that more than one type of particle can call it and make use of the algorithm. The method does not return anything. The parameters are: size – the size of memory to allocate, inStart – a pointer to the start of the memory allocation of the particles' state information, inLast – a pointer to the last active particle in the memory allocation of the particles' state

information, inEnd – a pointer to the end of the memory allocation of the particles' state information, inStartVertex – a pointer to the start of the memory allocation of the particles' vertex information, inLastVertex - a pointer to the last active particle in the memory allocation of the particles' vertex information, inEndVertex – a pointer to the end of the memory allocation of the particles' vertex information.
The 'Set Size' algorithm deletes the previous particle (both state information and vertex information structs) memory allocations:

```
delete [] inStart;
delete [] inStartVertex;
```

the algorithm then allocates memory based on the 'size' parameter passed in to the method and sets the pointers accordingly:

```
inLast = inStart = new T[size];
inLastVertex = inStartVertex = new Y[size];
inEnd = inStart+size;
inEndVertex = inStartVertex+size;
```

9.2.5 – Is Empty

The following is the method prototype for the 'Is Empty' algorithm:

```
template<typename T>
inline bool IsEmpty(T *&inStart, T *&inLast)
```

Once again this method is a template method due to the fact that more than one type of particle needs to call it and make use of the algorithm contained within. The parameters are: inStart – a pointer to the start of the memory allocation of the particles' state information, inLast – a pointer to the last active particle in the memory allocation of the particles' state information.
N.B. there is parameter to pass in the various particles' vertex information, this is because there is no need to do so because if inStart and inLast are equal then the particles' state information is empty, and as such the particles' vertex information will also be empty too.

```
return inStart==inLast; // if start equals last then it must be empty
```

9.2.6 – Is Full

The following is the method prototype for the 'Is Full' algorithm:

```
inline bool IsFull()
```

It doesn't need to be passed in an parameters because it uses a switch statement to determine what geometric primitive is been used to draw the particles, and from that it can then determine if the memory allocation for the particles is full or not. It returns true if the memory allocation for the particles is full, else it returns false.

```
switch(particleShape)
	{
		case shapePoint: // Points
			return lastPoints==endPoints;
			break;
		case shapeLine: // Lines
			return lastLines==endLines-2; // because a line needs two
lots of vertex info
			break;
		case shapeTriangle: // Triangles
			return lastTriangles==endTriangles-3; // because a
triangle needs three lots of vertex info
			break;
```

```
            default:
                  break;
      }
```

9.2.7 – Draw Particles

The prototype for the method that contains the algorithms for drawing the particles is this:

```
inline void DrawParticles()
```

It is an inline function because it is only called from one place in the program's code, therefore it is more efficient to make it an inline function. The method does not return anything; the method does not have any parameters because it uses a series of if statements inside the method to determine which type of particle are calling it.
The algorithm for drawing the particles works in the following way:
If the Sony PSP Game Developer has not ticked the 'Repeat' checkbox then the algorithm checks to see if there is any remaining time left to draw the particles, if there isn't then no particles are drawn

```
if (!emitterRepeat) // if the repeat option isn't ticked then
                                    // check that there is still time
remaining
      {
            if (timeLeft.getTime() > emitterDurationSeconds)
            {
                  return; // No time remaining so don't draw any particles
            }
      }
```

The algorithm then determines which geometric primitive to use for drawing the particles:

```
if (particleShape==shapePoint)
      {
…draw code
}
if (particleShape==shapeLine)
      {
…draw code
}
if (particleShape==shapeTriangle)
      {
…draw code
}
```

The actual draw code for shapePoint is shown below (it is almost identical for the other shapes too):

```
glPointSize((float)particleHeight);

      glVertexPointer(3,GL_FLOAT,sizeof(ParticlePointsVertex),startPointsVe
rtex->position);

      glColorPointer(4,GL_FLOAT,sizeof(ParticlePointsVertex),startPointsVer
tex->colour);
glTexEnvf(GL_POINT_SPRITE, GL_COORD_REPLACE, GL_TRUE);
            glEnable(GL_POINT_SPRITE_ARB);
glDrawArrays(GL_POINTS,0,lastPointsVertex-startPointsVertex);

            glDisable(GL_POINT_SPRITE_ARB);
```

9.2.8 – Get Number of Particles

The algorithm 'Get Number of Particles' is contained in the method of the same name and has the following method prototype:

```
template<typename T>
int GetNumberOfParticles(T *&inLast, T *&inStart)
```

It returns any integer type (which contains the number of active particles). The method is a template method, once again as previously iterated, this is done to ensure that any type of particle can call the method and access the algorithm within. The algorithm simply uses pointer to the last active particle and the pointer to the first particle and computes the difference, the result is the currently active number of particles.

```
int numberParticles = (int)(inLast-inStart);
return numberParticles;
```

9.2.9 – Add One

The 'Add One' method which contains the algorithm of the same name has the following method prototype:

```
template<typename T, typename Y>
void AddOne(T *&inLast, Y *&inLastVertex)
```

The method returns nothing. The method is a template method this is so that other types of particles can call the method successfully and access the algorithm contained within. It takes to parameters: inLast – a pointer to the last active particle in the memory allocation of the particles' state information, inLastVertex – a pointer to the last active particle in the memory allocation of the particles' vertex information. The algorithm uses a series of if statements to determine the type of particle that is calling it. The algorithm then creates a new particle by creating its state information and vertex information and moving the two pointers passed in as parameters (inLast and inLastVertex) along. This is demonstrated with a code snippet below for the shapePoint type of particle:

```
if (particleShapeListboxItemId==shapePoint)
	{
		if (!IsFull()) {
			*inLast = T(); // create the particles state
			++inLast;
			*inLastVertex = Y(); // create the particles vertex
information
			++inLastVertex;
		}
	}
```

9.2.10 – Erase One

The 'Erase One' method which contains the algorithm of the same name has the following method prototype:

```
template<typename T, typename Y>
inline void EraseOne(T* p, T *&inStart, T *&inLast, Y* pVertex, Y
*&inStartVertex, Y *&inLastVertex)
```

The method returns nothing. The method is an inline function, this is because it is only called from one location in the software application's code and therefore inlining the function will add an efficiency gain. The method is a template method, this is so

that other types of particles can call the method successfully and access the algorithm contained within. It takes 6 parameters, the 6 parameters are: p – a pointer to the particle's state information that needs to be deleted, inStart – a pointer to the first particle in the memory allocation of the particles' state information, inLast – a pointer to the last active particle in the memory allocation of the particles' state information, pVertex – a pointer to the particle's vertex data that needs to be deleted, inStartVertex – a pointer to the first particle in the memory allocation of the particles' vertex information, inLastVertex – a pointer to the last active active particle in the memory allocation of the particles' vertex information. The algorithm works in the following manner:
Given a pointer to the particle that needs to be deleted, this function simply swaps the last particle with the one to be deleted. This means that all living particles are always tightly packed at the front of the array, and more importantly there is no need to perform any memory allocations.

```
if (particleShape==shapePoint)
      {
            if (!IsEmpty(inStartVertex, inLastVertex)) {
                  *pVertex = *(--inLastVertex);
            }
      }
```

9.2.11 – Update Particles

The 'Update Particles' method which contains the algorithm of the same name has the following method prototype:

```
template<typename T, typename Y>
void UpdateParticles(float dt, T *&inStart, T *&inLast, Y *&inStartVertex,
Y *&inLastVertex)
```

The method returns nothing. The method is an inline function, this is because it is only called from one location in the software application's code and therefore inlining the function will add an efficiency gain. The method is a template method, this is so that other types of particles can call the method successfully and access the algorithm contained within. It takes 5 parameters, the 5 parameters are: dt – dt is an acronym for Delta Time and is used to ensure smooth animation at each scene update, inStart – a pointer to the first particle in the memory allocation of the particles' state information, inLast – a pointer to the last active particle in the memory allocation of the particles' state information, inStartVertex – a pointer to the first particle in the memory allocation of the particles' vertex information, inLastVertex – a pointer to the last active active particle in the memory allocation of the particles' vertex information. The 'Update Particles' algorithm works in the following manner:
All particles (both pointers that reference the particles' state information and particles' vertex information) are traversed:

```
T* p = inStart;
Y* pVertex = inStartVertex;
while(p!=inLast)
```

the lifespan is decreased:

```
p->life -= dt;
```

If the particle has any life remaining then update the particle's colour, speed, direction and position.

```
if (p->life>0.0f) {
//{($end_r - $start_r) / $life}
```

```
// if the start colour is greater then the end colour then minus them, else
add
for (int i = 0; i < 3; i++)
// the alpha value
{
	if (particleStartColour[i]>particleEndColour[i]) // Then we want to
decrease the colour
	{
		if (particleEndColour[i]>0) // no point taking away if taking
away 0
		{
			pVertex->colour[i] -=
(particleColourRateOfChange*dt);//(particleStartColour[i] -
particleEndColour[i]);

	//*particleColourRateOfChange;
				if (pVertex->colour[i]<particleEndColour[i])
				{
				// Make sure the new colour is not less then the end
colour
					pVertex->colour[i] = particleEndColour[i];
				}
		}
	}
	else // Then we want to increase the colour
	{
		if (particleEndColour[i]>0) // no point adding if taking away 0
		{
			pVertex->colour[i] += (particleColourRateOfChange*dt);
			if (pVertex->colour[i]>particleEndColour[i])
			{
				// Make sure the new colour is not greater then the
end colour
				pVertex->colour[i] = particleEndColour[i];
			}
		}
	}
}
```

9.2.12 – Add and Initialise Particles

The algorithm Add and Initialise Particles is contained within the method called AddAndInitParticles. The AddAndInitParticles method has the following prototype:

```
void AddAndInitParticles()
```

The AddAndInitParticles method returns nothing because it doesn't need to return anything. The AddAndInitParticles method does not take any parameters because it doesn't need any to function properly.

The 'Add and Initialise Particles' algorithm works in the following manner:

```
void AddAndInitParticles()
{
	//InitParticle(startPoints, lastPoints);
	// add particles before initialising them
	// before adding any particles, get the current amount of particles
	// that it will be known which particles need initialising.
	int numberOfParticlesBefore = 0;
	int numberOfParticlesAfter = 0;
	int val = rand()%100000;
	switch (particleShape)
```

```
	{
	case shapePoint:
		numberOfParticlesBefore = GetNumberOfParticles(lastPoints,
startPoints);
		for(int i=0;i!=val;++i)
		{
			AddOne(lastPoints, lastPointsVertex);
		}
		InitParticles(startPointsVertex, lastPointsVertex,
numberOfParticlesBefore+1);
		break;
	case shapeLine:
		numberOfParticlesBefore = GetNumberOfParticles(lastLines,
startLines);
		for(int i=0;i!=val;++i)
		{
			AddOne(lastLines, lastLinesVertex);
		}
		InitParticles(startLinesVertex, lastLinesVertex,
numberOfParticlesBefore+1);
		break;
	case shapeTriangle:
		numberOfParticlesBefore = GetNumberOfParticles(lastTriangles,
startTriangles);
		for(int i=0;i!=val;++i)
		{
			AddOne(lastTriangles, lastTrianglesVertex);
		}
		InitParticles(startTrianglesVertex, lastTrianglesVertex,
numberOfParticlesBefore+1);
	default :
		break;
	}
}
```

10 – Testing

10.1 – Particle System Editor Test Report

Test	*Expected Result*	*Actual Result*	*Pass or Fail*	*Comments*
Starting the Particle System Editor	Starts without any errors	Starts without any errors	Pass	None
Clicking the ‘Load Settings’ button whilst the settings.txt	Load the settings from the settings.txt file and apply them to the	Loaded the settings from the settings.txt files and applied them	Pass	None

file is present.	particles appropriately.	to the particles appropriately		
Clicking the ‘Load Settings’ button whilst the settings.txt file is not present.	Show a message to the Sony PSP Game Developer, explaining that the settings.txt was not found and therefore not loaded. The Particle System Editor should continue to run unhindered.			
Clicking the ‘Save Settings’ button.	Saving the current settings (as present on the gui of the Particle System Editor) to the settings.txt file.	Wrote the current settings (as present on the gui of the Particle System Editor) to the settings.txt file.	Pass	None
Clicking the ‘Write PSP File’ button	Writing a file that can be loaded by the Sony PSP Code Template File that will be a part of this project’s distribution.	Nothing happens.	Fail	Not implemented
Clicking the ‘Preset Properties’ panel’s minimise	Minimise the ‘Preset Properties’ panel.	Minimised the ‘Preset Properties’ panel.	Pass	None

button whilst the ‘Preset Properties’ panel is maximised.				
Clicking the ‘Preset Properties’ panel’s maximise button whilst the ‘Preset Properties’ panel is minimised.	Maximise the ‘Preset Properties’ panel.	Maximises the ‘Preset Properties’ panel.	Pass	None
Selecting the ‘Snow’ option from the drop-down menu on the ‘Preset Properties’ panel.	Change the particle effect to that of a snow particle effect.	Changed the particle effect to that of a snow particle effect.	Pass	None
Selecting the ‘Rain’ option from the drop-down menu on the ‘Preset Properties’ panel.	Change the particle effect to that of a rain particle effect.	Changed the particle effect to that of a rain particle effect.	Pass	None
Clicking the ‘Emitter Properties’ panel’s minimise button whilst the ‘Emitter Properties’ panel is maximised.	Minimise the ‘Emitter Properties’ panel.	Minimised the ‘Emitter Properties’ panel.	Pass	None
Clicking the	Maximise the	Maximises the	Pass	None

‘Emitter Properties’ panel’s maximise button whilst the ‘Emitter Properties’ panel is minimised.	‘Emitter Properties’ panel.	‘Emitter Properties’ panel.		
Selecting the ‘Box’ option from the ‘Shape’ drop-down menu on the ‘Emitter Properties’ panel.	Change the geometric shape of the emitter to a box.	Changed the geometric shape of the emitter to a box.	Pass	None
Selecting the ‘Circle’ option from the ‘Shape’ drop-down menu on the Emitter Properties’ panel.	Change the geometric shape of the emitter to a triangle.			
Change the value in the ‘Position X’ box on the ‘Emitter Properties’ panel to a numerical value	Change the particles’ x position to match the value contained within the ‘Position X’ box on the ‘Emitter Properties’ panel.	The particles’ x position was altered to match the value contained within the ‘Position X’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Position Y’ box on the	Change the particles’ y position to match the	The particles’ y position was altered to match the	Pass	None

‘Emitter Properties’ panel to a numerical value	value contained within the ‘Position Y’ box on the ‘Emitter Properties’ panel.	value contained within the ‘Position Y’ box on the ‘Emitter Properties’ panel.		
Change the value in the ‘Position Z’ box on the ‘Emitter Properties’ panel to a numerical value	Change the particles’ z position to match the value contained within the ‘Position Z’ box on the ‘Emitter Properties’ panel.	The particles’ z position was altered to match the value contained within the ‘Position Z’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Position X Min’ box on the ‘Emitter Properties’ panel to a numerical value greater than that of the current value of the ‘Position X Max’ box on the ‘Emitter Properties’ panel.	Change the value in the ‘Position X Min’ box on the ‘Emitter Properties’ panel to the same numerical value to that of the ‘Position X Max’ box on the ‘Emitter Properties’ panel.	Altered the value in the ‘Position X Min’ box on the ‘Emitter Properties’ panel to match the numerical value to that of the ‘Position X Max’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Position Y Min’ box on the ‘Emitter Properties’ panel to a	Change the value in the ‘Position Y Min’ box on the ‘Emitter Properties’ panel to the	Altered the value in the ‘Position Y Min’ box on the ‘Emitter Properties’ panel to match	Pass	None

numerical value greater than that of the current value of the ‘Position Y Max’ box on the ‘Emitter Properties’ panel.	same numerical value to that of the ‘Position Y Max’ box on the ‘Emitter Properties’ panel.	the numerical value to that of the ‘Position Y Max’ box on the ‘Emitter Properties’ panel.		
Change the value in the ‘Position Z Min’ box on the ‘Emitter Properties’ panel to a numerical value greater than that of the current value of the ‘Position Z Max’ box on the ‘Emitter Properties’ panel.	Change the value in the ‘Position Z Min’ box on the ‘Emitter Properties’ panel to the same numerical value to that of the ‘Position Z Max’ box on the ‘Emitter Properties’ panel.	Altered the value in the ‘Position Z Min’ box on the ‘Emitter Properties’ panel to match the numerical value to that of the ‘Position Z Max’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Position X Max’ box on the ‘Emitter Properties’ panel to a numerical value less than that of the current value of the ‘Position X Min’ box on the ‘Emitter Properties’ panel.	Change the value in the ‘Position X Max’ box on the ‘Emitter Properties’ panel to the same numerical value to that of the ‘Position X Min’ box on the ‘Emitter Properties’ panel.	Altered the value in the ‘Position X Max’ box on the ‘Emitter Properties’ panel to match the numerical value to that of the ‘Position X Min’ box on the ‘Emitter Properties’ panel.	Pass	None

Change the value in the ‘Position Y Max’ box on the ‘Emitter Properties’ panel to a numerical value less than that of the current value of the ‘Position Y Min’ box on the ‘Emitter Properties’ panel.	Change the value in the ‘Position Y Max’ box on the ‘Emitter Properties’ panel to the same numerical value to that of the ‘Position Y Min’ box on the ‘Emitter Properties’ panel.	Altered the value in the ‘Position Y Max’ box on the ‘Emitter Properties’ panel to match the numerical value to that of the ‘Position Y Min’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Position Z Max’ box on the ‘Emitter Properties’ panel to a numerical value less than that of the current value of the ‘Position Z Min’ box on the ‘Emitter Properties’ panel.	Change the value in the ‘Position Z Max’ box on the ‘Emitter Properties’ panel to the same numerical value to that of the ‘Position Z Min’ box on the ‘Emitter Properties’ panel.	Altered the value in the ‘Position Z Max’ box on the ‘Emitter Properties’ panel to match the numerical value to that of the ‘Position Z Min’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Position X Min’ box on the ‘Emitter Properties’ panel to a numerical value equal to or less than	Update the particles’ position to reflect the change in value of the ‘Position X Min’ box on the ‘Emitter Properties’	Updated the particles’ position to reflect the change in value of the ‘Position X Min’ box on the ‘Emitter Properties’	Pass	None

that of the current value of the ‘Position X Max’ box on the ‘Emitter Properties’ panel.	panel.	panel.		
Change the value in the ‘Position Y Min’ box on the ‘Emitter Properties’ panel to a numerical value equal to or less than that of the current value of the ‘Position Y Max’ box on the ‘Emitter Properties’ panel.	Update the particles’ position to reflect the change in value of the ‘Position Y Min’ box on the ‘Emitter Properties’ panel.	Updated the particles’ position to reflect the change in value of the ‘Position Y Min’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Position Z Min’ box on the ‘Emitter Properties’ panel to a numerical value equal to or less than that of the current value of the ‘Position Z Max’ box on the ‘Emitter Properties’ panel.	Update the particles’ position to reflect the change in value of the ‘Position Z Min’ box on the ‘Emitter Properties’ panel.	Updated the particles’ position to reflect the change in value of the ‘Position Z Min’ box on the ‘Emitter Properties’ panel.	Pass	None

Change the value in the ‘Position X Max’ box on the ‘Emitter Properties’ panel to a numerical value equal to or greater than that of the current value of the ‘Position X Min’ box on the ‘Emitter Properties’ panel.	Update the particles’ position to reflect the change in value of the ‘Position X Max’ box on the ‘Emitter Properties’ panel.	Updated the particles’ position to reflect the change in value of the ‘Position X Max’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Position Y Max’ box on the ‘Emitter Properties’ panel to a numerical value equal to or greater than that of the current value of the ‘Position Y Min’ box on the ‘Emitter Properties’ panel.	Update the particles’ position to reflect the change in value of the ‘Position Y Max’ box on the ‘Emitter Properties’ panel.	Updated the particles’ position to reflect the change in value of the ‘Position Y Max’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Position Z Max’ box on the ‘Emitter Properties’ panel to a numerical	Update the particles’ position to reflect the change in value of the ‘Position Z Max’ box on	Updated the particles’ position to reflect the change in value of the ‘Position Z Max’ box on	Pass	None

value equal to or greater than that of the current value of the ‘Position Z Min’ box on the ‘Emitter Properties’ panel.	the ‘Emitter Properties’ panel.	the ‘Emitter Properties’ panel.		
Change the value in the ‘Maximum Particles’ box on the ‘Emitter Properties’ panel.	Update the number of active particles to reflect the change in value of the ‘Maximum Particles’ box on the ‘Emitter Properties’ panel.	Updated the number of active particles to reflect the change in value of the ‘Maximum Particles’ box on the ‘Emitter Properties’ panel.	Pass	None
Change the value in the ‘Duration in Seconds’ box on the ‘Emitter Properties’ panel to a numerical value of less than 1.	Set the value in the ‘Duration in Seconds’ box on the ‘Emitter Properties’ panel to a numeric value of 1.	Changed the value of the ‘Duration in Seconds’ box on the ‘Emitter Properties’ panel to a numeric value of 1.	Pass	None
Change the value in the ‘Duration in Seconds’ box on the ‘Emitter Properties’ panel.	Change how long the particle effect lasts for to reflect the new value in the ‘Duration in Seconds’ box on the ‘Emitter Properties’ panel.	Changed how long the particle effect lasts for to reflect the new value in the ‘Duration in Seconds’ box on the ‘Emitter Properties’ panel.	Pass	None

Tick the ‘Repeat’ checkbox on the ‘Emitter Properties’ panel.	Repeat the particle effect indefinitely.	The particle effect was repeated indefinitely.	Pass	None
Untick the ‘Repeat’ checkbox on the ‘Emitter Properties’ panel.	Stop repeating the particle effect indefinitely.	The particle effect was stopped repeating itself indefinitely.	Pass	None
Clicking the ‘Particle Properties’ panel’s minimise button whilst the ‘Particle Properties’ panel is maximised.	Minimise the ‘Particle Properties’ panel.	Minimised the ‘Particle Properties’ panel.	Pass	None
Clicking the ‘Particle Properties’ panel’s maximise button whilst the ‘Particle Properties’ panel is minimised.	Maximise the ‘Particle Properties’ panel.	Maximises the ‘Particle Properties’ panel.	Pass	None
Selecting the ‘Point’ option from the ‘Shape’ drop-down box on the ‘Particle Properties’ panel.	Change the geometric shape of the particles to a point shape.	Changed the geometric shape of the particles to a point shape.	Pass	None

Selecting the ‘Line’ option from the ‘Shape’ drop-down box on the ‘Particle Properties’ panel.	Change the geometric shape of the particles to a line shape.	Screen goes black	Fail	Not implemented
Selecting the ‘Triangle’ option from the ‘Shape’ drop-down box on the ‘Particle Properties’ panel.	Change the geometric shape of the particles to a triangle shape.	Screen goes black	Fail	Not implemented
Change the value in the ‘Height’ box on the ‘Particle Properties’ panel to a value of 0 or lower.	Set the value in the ‘Height’ box on the ‘Particle Properties’ panel to a value of 1.	The value in the ‘Height’ box on the ‘Particle Properties’ panel was set to a value of 1.	Pass	None
Change the value in the ‘Height’ box on the ‘Particle Properties’ panel to a value of 1 or greater.	Change the height of the particles to match the value contained within the ‘Height’ box on the ‘Particle Properties’ panel.	The value in the ‘Height’ box on the ‘Particle Properties’ panel was set to a value contained within the ‘Height’ box on the ‘Particle Properties’ panel.	Pass	None
Change the value in the ‘Width’ box on the ‘Particle	Set the value in the ‘Width’ box on the ‘Particle	The value in the ‘Width’ box on the ‘Particle Properties’	Pass	None.

Properties' panel to a value of 0 or lower.	Properties' panel to a value of 1.	panel was set to a value of 1.		
Change the value in the 'Width' box on the 'Particle Properties' panel to a value of 1 or greater.	Change the width of the particles to match the value contained within the 'Width' box on the 'Particle Properties' panel.	The value in the 'Width' box on the 'Particle Properties' panel was set to a value contained within the 'Width' box on the 'Particle Properties' panel.	Pass	None
Selecting the 'Filtered' option from the 'Blending Mode' drop-down box on the 'Particle Properties' panel.	Change the blending mode to a filtered blending mode to reflect the selection made.	Changed the blending mode to a filtered blending mode to reflect the selection made.	Pass	None
Selecting the 'Additive' option from the 'Blending Mode' drop-down box on the 'Particle Properties' panel.	Change the blending mode to an additive blending mode to reflect the selection made.	Changed the blending mode to an additive blending mode to reflect the selection made.	Pass	None
Changing the numerical value in the 'Start Colour Red' box on the 'Particle Properties'	Change the numerical value in the 'Start Colour Red' box on the 'Particle Properties'	Changed the numerical value in the 'Start Colour Red' box on the 'Particle Properties'	Pass	None

panel to a numerical value greater than 1.0.	panel to a numerical value of 1.0.	panel to a numerical value of 1.0.		
Changing the numerical value in the ‘Start Colour Green’ box on the ‘Particle Properties’ panel to a numerical value greater than 1.0.	Change the numerical value in the ‘Start Colour Green’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Changed the numerical value in the ‘Start Colour Green’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Pass	None
Changing the numerical value in the ‘Start Colour Blue’ box on the ‘Particle Properties’ panel to a numerical value greater than 1.0.	Change the numerical value in the ‘Start Colour Blue’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Changed the numerical value in the ‘Start Colour Blue’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Pass	None
Changing the numerical value in the ‘Start Colour Red’ box on the ‘Particle Properties’ panel to a numerical value less than 0.0.	Change the numerical value in the ‘Start Colour Red’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.	Changed the numerical value in the ‘Start Colour Red’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.	Pass	None
Changing the numerical value in the ‘Start Colour	Change the numerical value in the ‘Start Colour	Changed the numerical value in the ‘Start Colour	Pass	None

Green’ box on the ‘Particle Properties’ panel to a numerical value less than 0.0.	Green’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.	Green’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.		
Changing the numerical value in the ‘Start Colour Blue’ box on the ‘Particle Properties’ panel to a numerical value less than 0.0.	Change the numerical value in the ‘Start Colour Blue’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.	Changed the numerical value in the ‘Start Colour Blue’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.	Pass	None
Changing the numerical value in the ‘End Colour Red’ box on the ‘Particle Properties’ panel to a numerical value greater than 1.0.	Change the numerical value in the ‘End Colour Red’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Changed the numerical value in the ‘End Colour Red’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Pass	None
Changing the numerical value in the ‘End Colour Green’ box on the ‘Particle Properties’ panel to a numerical value greater than 1.0.	Change the numerical value in the ‘End Colour Green’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Changed the numerical value in the ‘End Colour Green’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Pass	None

Changing the numerical value in the ‘End Colour Blue’ box on the ‘Particle Properties’ panel to a numerical value greater than 1.0.	Change the numerical value in the ‘End Colour Blue’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Changed the numerical value in the ‘End Colour Blue’ box on the ‘Particle Properties’ panel to a numerical value of 1.0.	Pass	None
Changing the numerical value in the ‘End Colour Red’ box on the ‘Particle Properties’ panel to a numerical value less than 0.0.	Change the numerical value in the ‘End Colour Red’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.	Changed the numerical value in the ‘End Colour Red’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.	Pass	None
Changing the numerical value in the ‘End Colour Green’ box on the ‘Particle Properties’ panel to a numerical value less than 0.0.	Change the numerical value in the ‘End Colour Green’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.	Changed the numerical value in the ‘End Colour Green’ box on the ‘Particle Properties’ panel to a numerical value of 0.0.	Pass	None
Changing the numerical value in the ‘End Colour Blue’ box on the ‘Particle Properties’ panel to a numerical	Change the numerical value in the ‘End Colour Blue’ box on the ‘Particle Properties’ panel to a numerical	Changed the numerical value in the ‘End Colour Blue’ box on the ‘Particle Properties’ panel to a numerical	Pass	None

value less than 0.0.	value of 0.0.	value of 0.0.		
Ticking the ‘Random Colour’ tickbox on the ‘Particle Properties’ panel.	Change the particles’ colour to a random colour.	Changed the particles’ colour to a random colour.	Pass	None
Changing the numerical value in the ‘Gravity X’ box on the ‘Particle Properties’ panel.	Change the effect of gravity on the particles to reflect the new gravitational x value.	Changed the effect of gravity on the particles’ to reflect the new gravitational x value.	Pass	None
Changing the numerical value in the ‘Gravity Y’ box on the ‘Particle Properties’ panel.	Change the effect of gravity on the particles to reflect the new gravitational y value.	Changed the effect of gravity on the particles’ to reflect the new gravitational y value.	Pass	None
Changing the numerical value in the ‘Gravity Z’ box on the ‘Particle Properties’ panel.	Change the effect of gravity on the particles to reflect the new gravitational z value.	Changed the effect of gravity on the particles’ to reflect the new gravitational z value.	Pass	None
Changing the numerical value in the ‘Speed X Min’ box on the ‘Particle Properties’ panel.	Change the effect of speed on the particles to reflect the new speed x min value.	Changed the effect of speed on the particles to reflect the new speed x min value.	Pass	None
Changing the numerical value in the	Change the effect of speed on the particles	Changed the effect of speed on the particles	Pass	None

‘Speed Y Min’ box on the ‘Particle Properties’ panel.	to reflect the new speed y min value.	to reflect the new speed y min value.		
Changing the numerical value in the ‘Speed Z Min’ box on the ‘Particle Properties’ panel.	Change the effect of speed on the particles to reflect the new speed z min value.	Changed the effect of speed on the particles to reflect the new speed z min value.	Pass	None
Changing the numerical value in the ‘Speed X Max’ box on the ‘Particle Properties’ panel.	Change the effect of speed on the particles to reflect the new speed x max value.	Changed the effect of speed on the particles to reflect the new speed x max value.	Pass	None
Changing the numerical value in the ‘Speed Y Max’ box on the ‘Particle Properties’ panel.	Change the effect of speed on the particles to reflect the new speed y max value.	Changed the effect of speed on the particles to reflect the new speed y max value.	Pass	None
Changing the numerical value in the ‘Speed Z Max’ box on the ‘Particle Properties’ panel.	Change the effect of speed on the particles to reflect the new speed z max value.	Changed the effect of speed on the particles to reflect the new speed z max value.	Pass	None

11 – Project Appraisal

11.1 – Software Achievements

On inspection of the initial brief of this project it appears that my project has indeed started along the path that was intended for it and making some progress along the way. However the project hasn't fully reached the goal that was set by the project's initial brief, which was to get a particle system working on the Sony PSP.
An achievement of the software is that a user (game developer) can, in fact, create various particles effects using the 'PC Particle System Editor'; these effects can be saved, and loaded, thus they are editable and maintainable.

11.2 – Software Limitations

Whilst various particles effects can be created using the 'PC Particle System Editor' the must be points, i.e. they cannot be lines or triangles like planned. Since the particle effects must be points they cannot be textured.
Another limitation of the software is that whilst it is indeed true that the 'PC Particle System Editor' can save and load particle effects, it cannot write them to a file that the PSP can load. That is to say that there is currently no framework developed and in place to load custom particle effects created by the user (game developer) and have them running on the Sony PSP.

11.3 – Critical Appraisal

Overall I am not impressed, nor am I pleased with progress of the project. The aim of the project was to have a working particle system on the Sony PSP, and I have not achieved this. However, with that in mind, I have managed to produce a fairly robust Particle System Editor on the PC, a particle system editor which can save and load the particle effects it creates. The software I have produced is efficient and efficiency was a cause for concern since the intended target platform was a handheld console with limited hardware capabilities. Lastly (and oft the most overlooked part of software development) the graphical user interface is operational and fairly intuitive, however I do not like the look of it and I feel it is too big and cumbersome to use.

11.4 – Future Work and Improvements

First and foremost I would focus on getting the particle system to work on the Sony PSP. I do not foresee this being too much trouble, it would just be time consuming, and require having more patience on my part, with hardware that I have had little experience and technical exposure to.
The second thing that I would work on is allowing the particles to be drawn using lines and triangles, this way, when using particles to represent the particles, the particles can be textured, thus looking more aesthetically pleasing.
The third and last thing that I would work on is the graphical user interface. I would rewrite it completely using C# .Net as it is much easier (developer time wise) to

developer (decent and intuitive) graphical user interfaces using the .Net framework. Plus I have lots of experience working with C# and the .Net framework.

11.5 – Experience Gained

I have gained a tremendous amount of experience whilst working on this project, and in my opinion there is no such thing as bad experience. I have had exposure to new hardware, namely the Sony PSP. I have also gained experience with working to a schedule and working under extreme pressure with multiple project (other ACWs) deadlines to meet. And of course I have gained a great amount of experience and knowledge about Particle Systems.
In conclusion, whilst I feel the project has failed, I do not deem it a complete failure (no where a complete failure, in fact).

12 - Background References

Lander J, 1998, *The Ocean Spray in Your Face* [online], Available: http://www.darwin3d.com/gamedev/articles/col0798.pdf [Accessed 16th October 2008].

Dalmau DSC, 2003, *Core Techniques and Algorithms in Game Programming*, New Riders Publishing.

Reeves TR, 1983, Particle Systems – *A Technique for Modeling a Class of Fuzzy Objects*, SIGGRAPH [online], 17(3), p359. Available: http://www.lri.fr/~mbl/ENS/IG2/devoir2/files/docs/fuzzyParticles.pdf [Accessed 16th October 2008].

Zhang J, Angel E, Alsing P, Munich D, *An Object-Orientated Particle System for Simulation and Visualization* [online], Available: http://www.cs.unm.edu/~treport/tr/01-06/particle.pdf [Accessed 17th October 2008].

VDB John, 2000, *Building an Advanced Particle System* [online], Game Developer, Available: http://www.mysticgd.com/misc/AdvancedParticleSystems.pdf [Accessed 18th October 2008].

Molofee J, *Lesson: 19* [online], Available: http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=19 [Accessed 17th October 2008].

Allen M, *Particle Systems* [online], Available: http://web.cs.wpi.edu/~matt/courses/cs563/talks/psys.html [Accessed 17th October 2008].

Latta L, 2004, *Building a Million-Particle System* [online], Available: http://www.gamasutra.com/view/feature/2122/building_a_millionparticle_system.php [Accessed 22nd October 2008].

Sabo M, *Improving advanced particle system by adding property milestones to particle life cycle* [online], Available: http://www.cescg.org/CESCG-2004/web/Sabo-Miroslav/ [Accessed 22nd October 2008].

Giplin A, *C++ Performance Tips* [online], Available: http://www-2.cs.cmu.edu/~gilpin/c%2B%2B/performance.html [Accessed 12th January 2009].

Rademacher P, Stewart N, *GLUI User Interface Library* [online], Available: http://www.cs.unc.edu/~rademach/glui/ [Accessed 15th January 2009].

MSDN, *What's New in Visual Basic and Visual C#* [online], Available: http://msdn.microsoft.com/en-us/library/aa984213(VS.71).aspx [Accessed 1st April 2009].

Altizer R, *Sony PSP (Playstation Portable) Specifications and Details*, Available: http://playstation.about.com/od/psp/a/PSPSpecs.htm [Accessed 3rd April 2009].

Atari, *Neverwinter Nights 2*, Available: http://www.atari.com/nwn2/US/index.php [Accessed 3rd April 2009].

Delta3D, *Graphical Particle System Editor*, Available: http://www.delta3d.org/article.php?story=20041130085141252&topic=docs [Accessed 4th April 2009].

Particle System Editor, *Game Development Showcase*, Available: http://www.gamedev.net/community/gds/viewentry.asp?projectid=317925 [Accessed 4th April 2009].

13 – Appendix

13.1 - Appendix I

Altizer lists the PSP specifications as been:

PSP Product Specifications

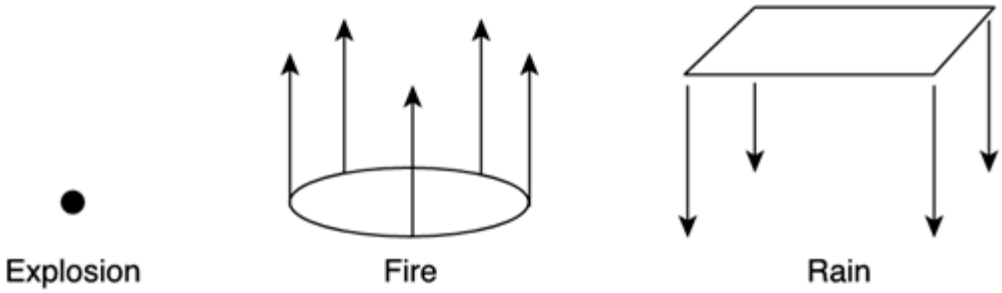
- Product Name: PlayStation Portable (PSP)
- Color: Black
- Dimensions: Approx. 170 mm (L) x 74 mm (W) x 23 mm (D)
- Weight: Approx. 260 g (including battery)
- CPU: PSP CPU (System clock frequency 1~333MHz)
- Main Memory: 32MB
- Embedded DRAM: 4MB
- Display: 4.3 inch, 16:9 widescreen TFT LCD, 480 x 272 pixel (16.77 million colors), Max. 200 cd/m2 (with brightness control)
- Speakers: Built-in stereo speakers
- Main Input/Output: IEEE 802.11b (Wi-Fi), USB 2.0 (Target), Memory Stick™ PRO Duo, IrDA, IR Remote (SIRCS)
- Disc Drive: UMD Drive (Playback only)
- Profile: PSP Game, UMD Audio, UMD Video
- Main Connectors: DC OUT 5V, Terminals for charging built-in battery, Headphone/Microphone/Control connector
- Keys/Switches: Directional buttons (Up/Down/Right/Left)Analog pad, Enter keys (Triangle, Circle, Cross, Square), Left, Right keys START, SELECT, HOME, POWER On/Hold/Off switch, Brightness control, Sound Mode, Volume +/-, Wireless LAN On/Off switch, UMD Eject
- Power: Built-in lithium-ion battery, AC adaptor
- Access Control: Region Code, Parental Control
- Accessories: Stand, Headphone with remote commander, Headphone with remote commander and microphone, External battery pack, Case, Strap
- E3 Prototype Exhibition: USB Camera for PSP, USB GPS for PSP, USB Keyboard for PSP

UMD Specifications

- Dimensions: Approx. 65 mm (W) x 64 mm (D) x 4.2 mm (H)
- Weight: Approx. 10g
- Disc Diameter: 60 mm
- Maximum Capacity: 1.8GB (Single-sided, dual layer)
- Laser wavelength: 660nm (Red laser)
- Encryption: AES 128bit
- Profile: PSP Game (full function), UMD Audio (codec ATRAC3plus™, PCM, (MPEG4 AVC)), UMD Video (codec MPEG4 AVC, ATRAC3plus™, Caption PNG)

-from Sony

13.2 - Appendix II

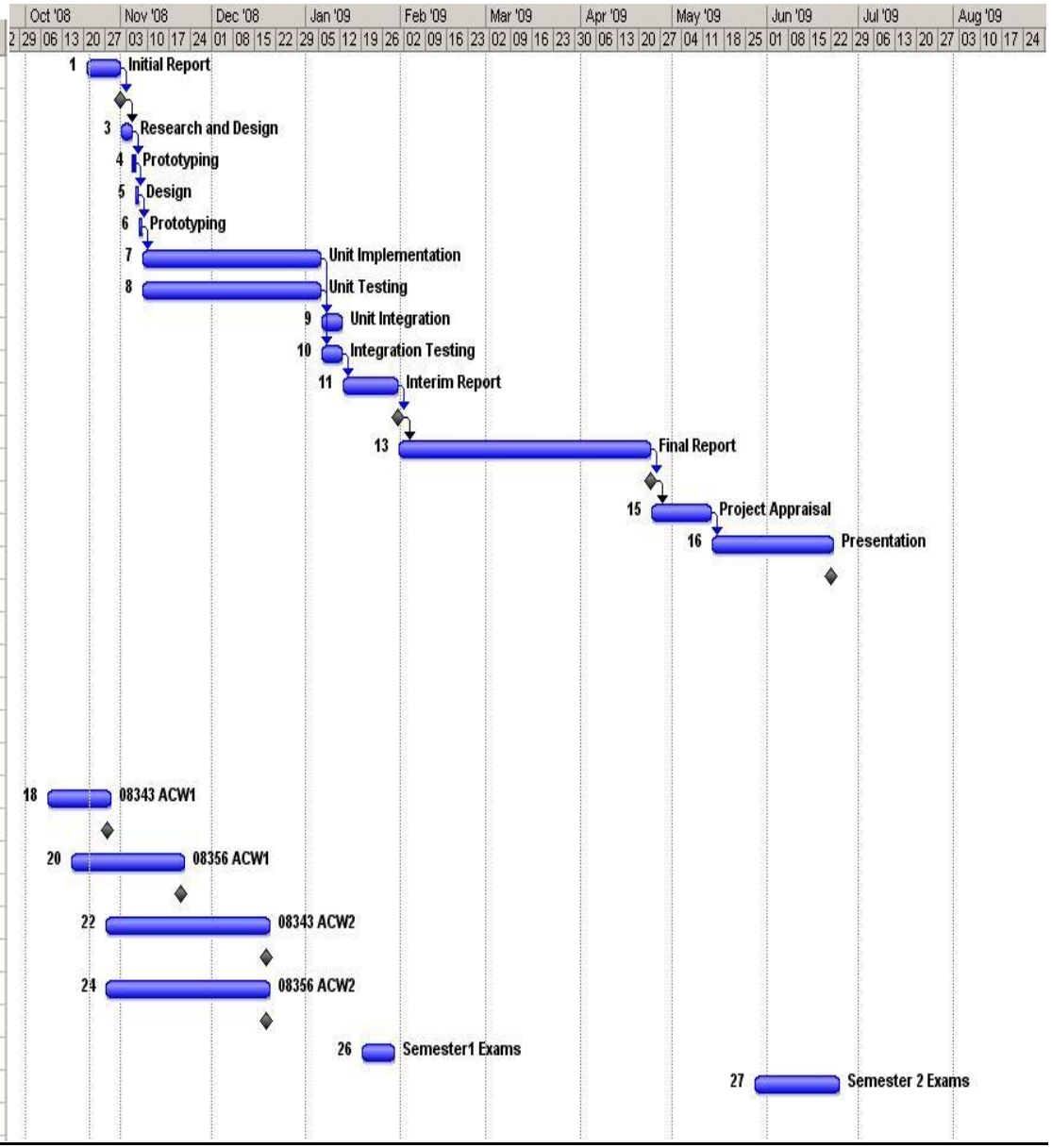


"*Particles are created at some kind of emitter, which initializes their parameters. If we want our system to behave in any interesting way,*

the key is to generate each particle with slightly different initial values, so the behavior rules (which are shared by all particles) make each one look like a unique element." (Dalmau)

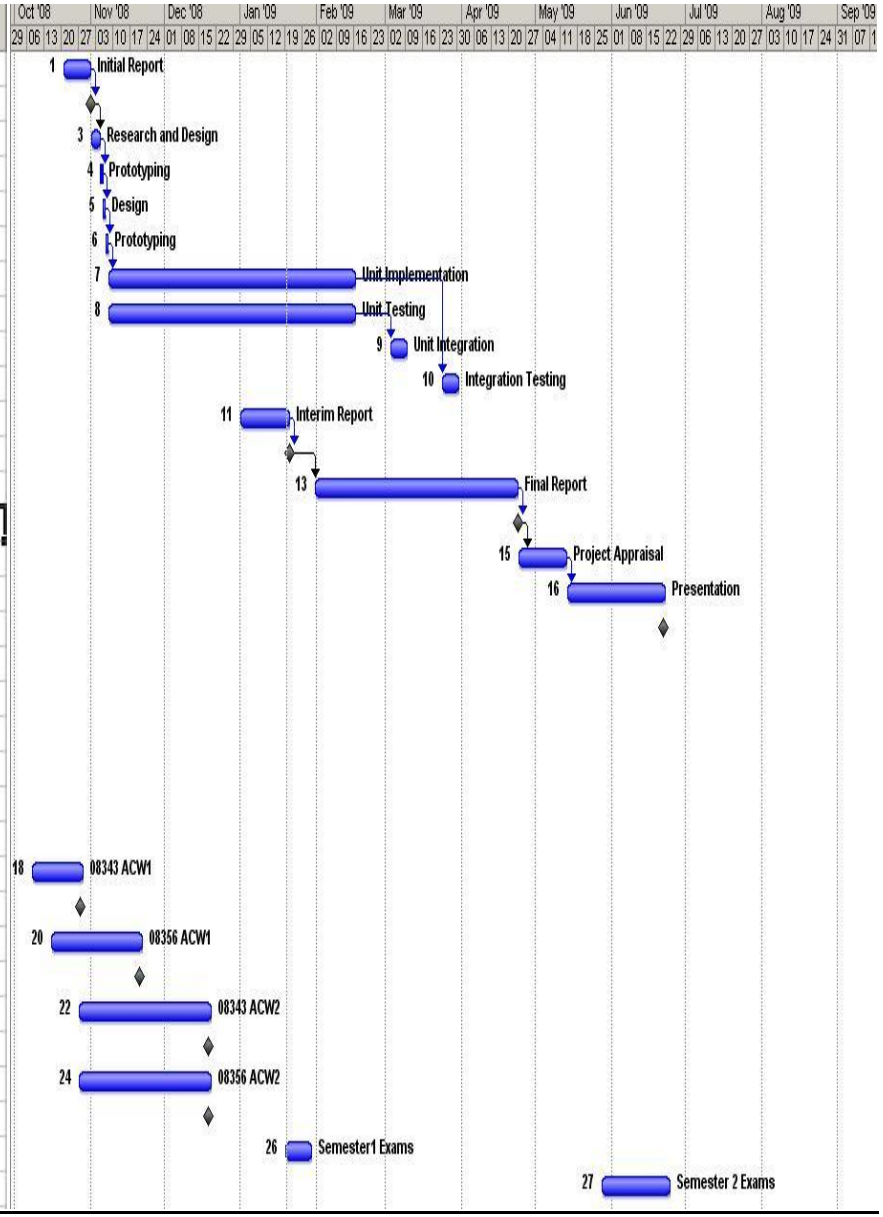
13.3 – Appendix III

	Task Name	Duration	Start	Finish	Predecessors
1	Initial Report	9 days	Tue 21/10/08	Fri 31/10/08	
2		0 days	Fri 31/10/08	Fri 31/10/08	1
3	Research and Design	3 days	Sat 01/11/08	Tue 04/11/08	2
4	Prototyping	1 day	Wed 05/11/08	Wed 05/11/08	3
5	Design	1 day	Thu 06/11/08	Thu 06/11/08	4
6	Prototyping	1 day	Fri 07/11/08	Fri 07/11/08	5
7	Unit Implementation	42 days	Sat 08/11/08	Mon 05/01/09	6
8	Unit Testing	42 days	Sat 08/11/08	Mon 05/01/09	
9	Unit Integration	5 days	Tue 06/01/09	Mon 12/01/09	8
10	Integration Testing	5 days	Tue 06/01/09	Mon 12/01/09	7
11	Interim Report	14 days	Tue 13/01/09	Fri 30/01/09	10
12		0 days	Fri 30/01/09	Fri 30/01/09	11
13	Final Report	60 days	Sat 31/01/09	Thu 23/04/09	12
14		0 days	Thu 23/04/09	Thu 23/04/09	13
15	Project Appraisal	14 days	Fri 24/04/09	Wed 13/05/09	14
16	Presentation	28 days	Thu 14/05/09	Mon 22/06/09	15
17		0 days	Mon 22/06/09	Mon 22/06/09	
18					
19					
20					
21					
22					
23					
24	08343 ACW1	15 days	Wed 08/10/08	Tue 28/10/08	
25		0 days	Tue 28/10/08	Tue 28/10/08	
26	08356 ACW1	29 days	Thu 16/10/08	Fri 21/11/08	
27		0 days	Fri 21/11/08	Fri 21/11/08	
28	08343 ACW2	42 days	Mon 27/10/08	Fri 19/12/08	
29		0 days	Fri 19/12/08	Fri 19/12/08	
30	08356 ACW2	42 days	Mon 27/10/08	Fri 19/12/08	
31		0 days	Fri 19/12/08	Fri 19/12/08	
32	Semester1 Exams	9 days	Mon 19/01/09	Thu 29/01/09	
33	Semester 2 Exams	20 days	Thu 28/05/09	Wed 24/06/09	



13.4 – Appendix IV

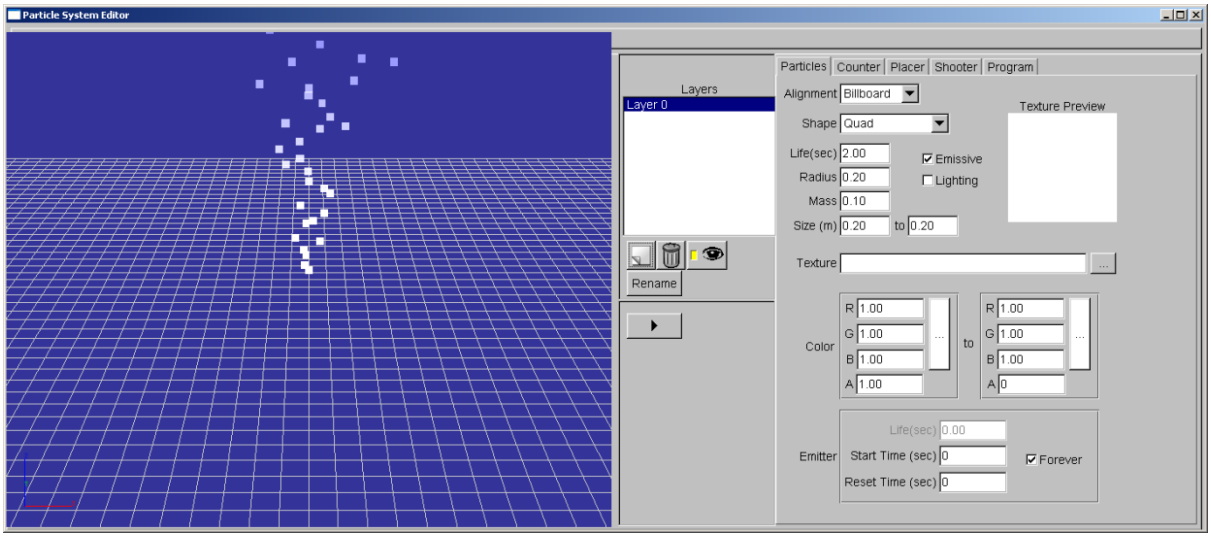
	Task Name	Duration	Start	Finish	Predecessors
1	Initial Report	9 days	Tue 21/10/08	Fri 31/10/08	
2		0 days	Fri 31/10/08	Fri 31/10/08	1
3	Research and Design	3 days	Sat 01/11/08	Tue 04/11/08	2
4	Prototyping	1 day	Wed 05/11/08	Wed 05/11/08	3
5	Design	1 day	Thu 06/11/08	Thu 06/11/08	4
6	Prototyping	1 day	Fri 07/11/08	Fri 07/11/08	5
7	Unit Implementation	73 days	Sat 08/11/08	Mon 16/02/09	6
8	Unit Testing	73 days	Sat 08/11/08	Mon 16/02/09	
9	Unit Integration	5 days	Tue 03/03/09	Mon 09/03/09	8
10	Integration Testing	5 days	Tue 24/03/09	Mon 30/03/09	7
11	Interim Report	14 days	Thu 01/01/09	Tue 20/01/09	
12		0 days	Tue 20/01/09	Tue 20/01/09	11
13	Final Report	60 days	Sat 31/01/09	Thu 23/04/09	12
14		0 days	Thu 23/04/09	Thu 23/04/09	13
15	Project Appraisal	14 days	Fri 24/04/09	Wed 13/05/09	14
16	Presentation	28 days	Thu 14/05/09	Mon 22/06/09	15
17		0 days	Mon 22/06/09	Mon 22/06/09	
18					
19					
20					
21					
22					
23					
24	08343 ACW1	15 days	Wed 08/10/08	Tue 28/10/08	
25		0 days	Tue 28/10/08	Tue 28/10/08	
26	08356 ACW1	29 days	Thu 16/10/08	Fri 21/11/08	
27		0 days	Fri 21/11/08	Fri 21/11/08	
28	08343 ACW2	42 days	Mon 27/10/08	Fri 19/12/08	
29		0 days	Fri 19/12/08	Fri 19/12/08	
30	08356 ACW2	42 days	Mon 27/10/08	Fri 19/12/08	
31		0 days	Fri 19/12/08	Fri 19/12/08	
32	Semester1 Exams	9 days	Mon 19/01/09	Thu 29/01/09	
33	Semester 2 Exams	20 days	Thu 28/05/09	Wed 24/06/09	



13.5 – Appendix V

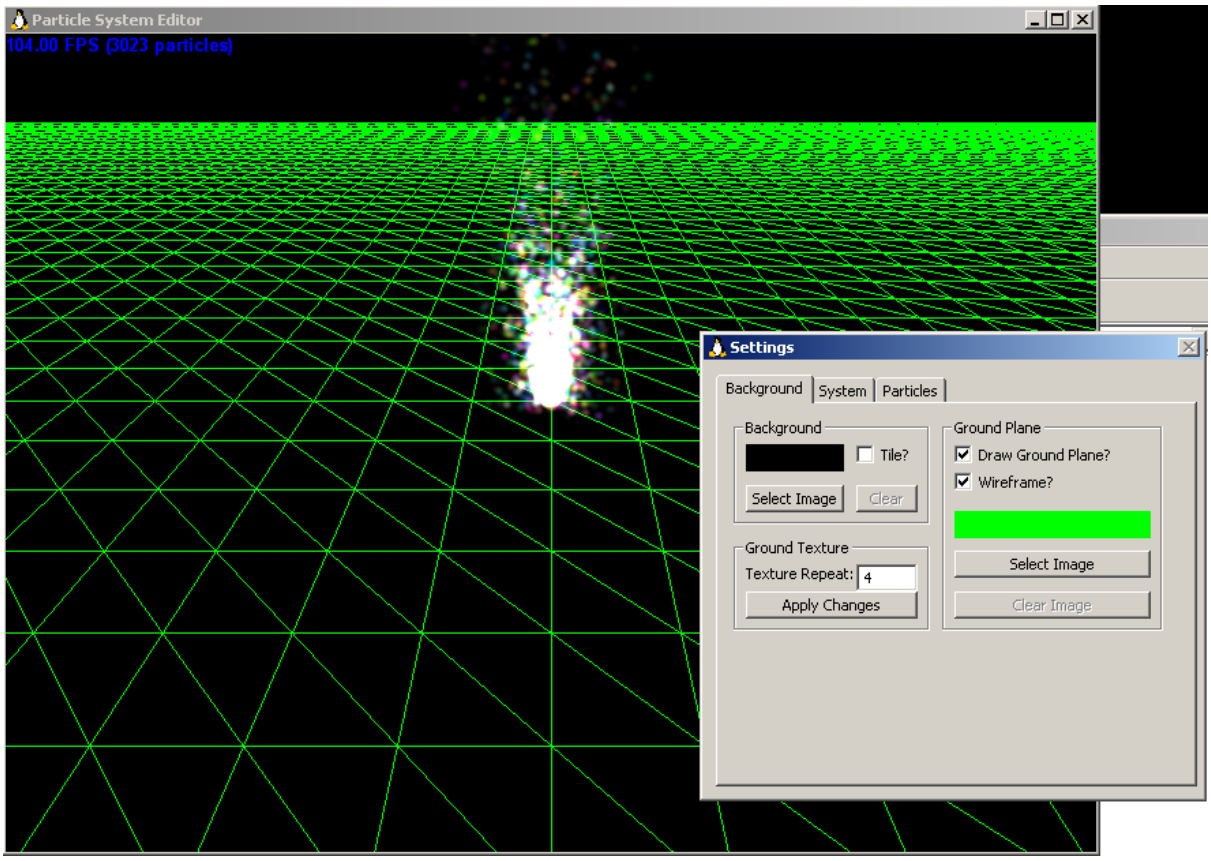
(Picture is from the game Neverwinter Nights 2, picture is taken from: http://uk.gamespot.com/pc/rpg/neverwinternights2/images/0/64/?tag=thumbs_below;thumb;64)

13.6 – Appendix VI



(Picture is taken from the Delta3D Graphical Particle System software application)

13.7 – Appendix VII



(Picture is taken from the Particle System Editor software application)