Grabin **Design Process Report**

v1.5

26 April, 2004

Purpose

This document exists primarily to facilitate an understanding of the development process to be adhered to during the prototyping and later development stages of *Grabin*, a mod for the computer game *Unreal Tournament 2004*. It is based on sound knowledge of the gaming industry and critical examination and adaptation of standard software engineering process models into such a development environment.

This document can be considered live and will be updated to reflect changes in design, development and scheduling, though outdated copies will be kept online for historical reasons. It references the original *Grabin* project proposal document, which can be found at http://student.ci.qut.edu.au/~n4405714/projects/grabin/images/proposal.pdf.
It is a large document, and so includes extensive cross-referencing for ease of reading and deconstruction.

Note that at the time of writing, some development work has already been undertaken, and so some process dates will precede this document's creation. Previous versions of this document which were in existence during this period were very rough and are not available for review. This was the fourth version of the process report at the time of initial writing, now the fifth.

Project Goals

1. Produce a working subsystem comprising the core features and functionality of the game. This subsystem can be thought of as the first project prototype, indeed the term 'subsystem' is used only in reference to terminology discussed later in this document. This prototype will encompass phases 1 and 2 of the original development plan as given in the project proposal, with possibility of the inclusion of aspects from phase 3 if time permits.

2. Completion of the remainder of the *Half-Grabin* museum discussed in the project proposal, and distribution to facilitate knowledge of the game as well as obtain player feedback. This and the following goals are long term only and are outside the scope of this unit.

3. Produce a multiplayer-only version of the game, with full functional and non-functional requirements developed, integrated and tested, to be distributed freely to the gaming community.

4. Complete an expansive singleplayer version of the game and distribute it.

Process Considerations

In devising a process model for the development of a game, one must first understand the differences between game development and traditional software development. This is particularly important as there are few stock process models yet designed to target the specifics of the game development process, or more accurately, few which the model's designers have made publicly available (Randy Angle, William Dwyer, 2001). Due to this, a custom development process must be designed which incorporates aspects of typical software engineering process models pertinent to games design.

Game development is different from regular application development for three major reasons, outlined below:

- Games are primarily response-driven unlike the function-driven nature of applications.

By this we mean that commercial software applications, those that traditional process models have been created for, simply provide a suite of functions that help users do their work (Pugh, S, 1991). Games, on the other hand, require an emotional and somewhat immersive interaction from the user- their primary purpose is to be enjoyable rather than functional. This is the first obstacle to successful development of a game, as maintaining objectivity whilst designing and developing a game can be quite difficult.

For applications, the developer need only observe how users interact with similar products and determine what features need to be added to create a better experience. Based on the project goals and unbiased research, detailed functional requirements can be defined (Frederic Brooks, 1995). Game development can never be so simple, as it is impossible to measure the experience and entertainment of a game objectively. Game designers have no unbiased criteria to determine which requirements should be added, and which discarded to make the game work. They can of course guess, and may sometimes create successful results, sometimes not. It is as a result of this major difference that a game design process should have a more careful design phase, and also that brings us to the second difference.

- Games require a different and much more expansive test phase than other applications.

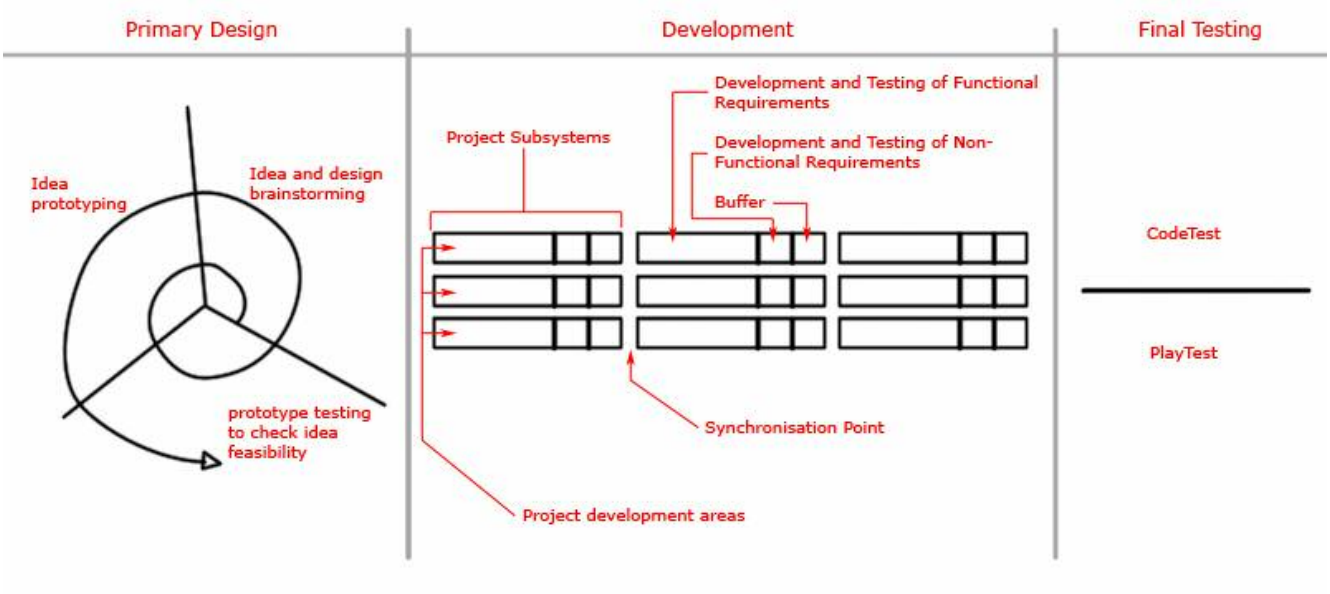
Traditionally, testing for software applications involves testing and debugging code. For games, it is much more complicated – both code and gameplay must be adequately tested and refined to achieve success. In effect, the game development process involves two discreet testing phases – 'codetest' and 'playtest', often making the test phase for games twice the length expected. There is no reason, however, that both these test phases cannot proceed at the same time.

- Games are a true multimedia content development effort.

The other important difference between game development and application development is the sheer scope of work needed to develop a game. A typical *Unreal Tournament 2004* mod generates somewhere between 5 and 20 packages comprising upwards of 50 files each. The design of a full game from scratch can generate anywhere upwards of half a million files of various types: textures, animations, movies, 3D models, scripting, dialogues, sound, music etc (NxN software, 2001) .This multi-content explosion can require many different team members with different specialties all working together, trying to reach release status in time to save delaying other aspects of the project which depend on their work. In the case of *Grabin* this is not such a big issue as it involves only a one-man development team, however precautions must still be taken so that some aspects of the project's development do not inhibit the development of others.

Process Model

*Grabin'*s process model is built on the belief that gameplay is the most important aspect of a game. To achieve the best gameplay it provisions for constant revisions and additions to the design of the game as well as code and play testing throughout the development phases (Bob Bates, 2001 and Zhan Ye, 2001). It also generates working subsystems of the final game at the addition of each new feature, which provides a playable prototype at every stage. This is particularly useful for preview distribution and as a failsafe, as previous subsystems can be returned to if necessary and can even be distributed instead of later builds if time constraints come up (Jacob Marner, 2002).



Grabin's process model

Primary Design

The process model begins with a 3-stage variant on the common iterative spiral model. A spiral model is generally perfect for short-term projects (Roger Pressman, 1997), which is why it is used here - It is common practice in games development to get a working kernel of the game up as quickly as possible so that the feasibility of new features can be examined before refining the design (Bob Bates, 2001). Because of this, preliminary design should be kept at a low level of complexity.

The first phase in the design stage is labeled 'Idea and Design Brainstorming', and corresponds loosely to the design stage in the standard spiral model. This phase comprises generating and examining major ideas for the game.

The second phase is a prototyping phase, wherein the ideas generated in brainstorming are developed on paper.

In the third testing/analysis phase, the prototypes are tested to make sure they will be developable. If any seem too difficult or simply need further work, we return to the brainstorming phase and think about the ideas some more, hence starting the process over again.

It is important to design the project this way so that focus can be achieved on both game design and the technologies available to support the design ideas (Randy Angle, William Dwyer, 2001). After enough iterations through this cycle that the project's ideas and goals are sufficiently defined, The development stage begins.

Development

The development stage exists as an incremental model with sequential models embedded within it (Roger Pressman, 1997). Each sequence represents a subsystem of the project, for example in *Grabin's* case the first sequence may be the development of an effect that shows the user is drawing something. These sequences are then broken down into development areas, shown above by three identical blocks sitting atop each other. Possible development areas could be coding, modeling and texturing for example.

During the development of each subsystem, redesign may take place. In such a case, new or modified features are added to the project's design. There have been many arguments over the years on whether it is possible to draw a clear line between design and development in games design, with no conclusions apparently reached. For *Grabin,* it is impossible to do so: the addition of one feature may mean the redesign of another to link them together properly. It may also mean discovering new things which make more features possible, and thus these new elements must be added to the design documentation. This is the ongoing nature of games design – game designers always want to improve on their initial ideas and gamers needs may change as time goes by. The functional requirements of a game are rarely fully known to begin with (Zhan Ye, 2001).

The development of each subsystem is also broken down into three smaller areas, much like the Microsoft model (Michael Cusumano, Richard Selby, 1998). Keep in mind that redesign may occur at any of them. Firstly the functional requirements for that subsystem are developed and tested. When these are in perfect working order development of non-functional requirements may commence. The final area of the subsystem's development acts as a time buffer which is extremely useful for any schedule slippage that may occur – if time runs out, the core system can always be submitted without the non-functional features. These features can be rescheduled in such a case and re-added in the development of a later project subsystem.

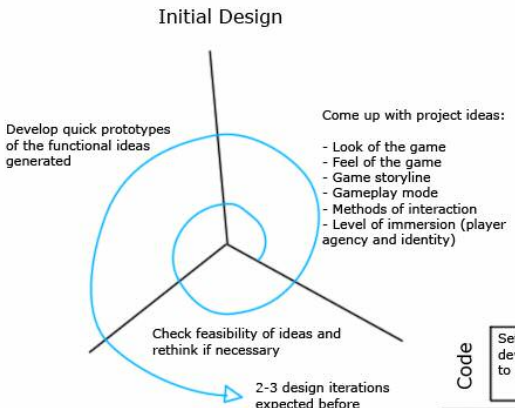
The final features of the development phase are the synchronization points which exist between the development of each project subsystem. These provide organisation between the different development areas of each subsystem so that, using an earlier example, coding, modeling and texturing all finish at the precise same time for them to be united and form a cohesive project subsystem.

This development model is particularly useful for games design in that at each synchronization point in the game's development, a working subsystem of the game is complete. Later subsystems will include all functionality (some revised) from previous ones, and a gradual incremental build of the game occurs.

Final Testing

This is the simplest stage in this process model, and merely comprises extensive code and play testing before the final game is released. Since testing recursively happens at each subsystem's development, final testing should not be overly arduous. It should consist of a limited beta distribution across many different operating environments and players so that all bugs and gameplay issues can be fixed before a full release. It will not be reached during the course of this unit and is merely explained for the purposes of ongoing development.

Development Schedule



SUBSYSTEM 1 — Feasability Test Build

Code	Set up base for mod to be developed and code the ability to draw shapes	Refine the drawing effect a little to make it aesthetically pleasing	BUFFER < 3rd April
Texturing			
3d			
Audio		Use a placeholder sound for gesturing at the moment	BUFFER < 3rd April

13th - 27th March — 27th - 31st March

SUBSYSTEM 2 — Project Website

Code	Create PHP and other code backend for a project website	Implement a fancy development journal	BUFFER < 6th April
Texturing	Create graphics for website as well as CSS and other visual features		BUFFER < 6th April
3d			
Audio			

3rd - 4th April — 4th - 5th April

SUBSYSTEM 3 — Melee Weapon

Not done yet

Code	Code a melee weapon to balance out the mod	~~Melee weapon animation~~	BUFFER < 13th April
Texturing		Design Textures for the melee weapon model	BUFFER < 13th April
3d		Model the melee weapon, reduce its polycount and import into the game	BUFFER < 13th April
Audio		~~Create swinging and hitting sounds for the weapon~~	BUFFER < 13th April

Not done yet

6th - 8th April — ~~9th - 11th April~~

11th

11th – 13th April

SUBSYSTEM 4 — Gesture Recognition

Code	Implement gesture recognition and create explanatory documentation	Optimise code for multiplayer and smooth gameplay	BUFFER < 5th May
Texturing			
3d			
Audio			

13th - 30th April — 1st - 3rd May

SUBSYSTEM 5 — Finalise Gesturing Effects

Code		Finalise gesturing effect and ensure it runs smoothly	BUFFER < 23rd May
Texturing	Texture new hand model	Create textures to use for the new gesturing effect and crosshair	BUFFER < 23rd May
3d	Model and animate a representation of the player's hand to use as a first person weapon mesh	Additional animation for visual purposes	BUFFER < 23rd May
Audio		Create proper custom sounds for the gesture drawing	BUFFER < 23rd May

5th - 15th May — 15th - 19th May

SUBSYSTEM 6 — Create Attacks and Other Results

Code	Create final fireball projectile attack and gravity manipulation gesture		BUFFER < 5th June
Texturing		Textures for fireball attack effect and perhaps levitation effect	BUFFER < 5th June
3d			
Audio		An ambient sound for the fireball, as well as sounds for when gestures happen	BUFFER < 5th June

23rd - 27th May — 27th - 31st May

SUBSYSTEM 7 — Gestures that Effect Other Objects

Code	Create interface for gestures between the player and world objects.	Create a door which opens when gestured at.	
Texturing		Texture the door	
3d		Model and animate the door	
Audio		Create sounds for the door's movement	

IF TIME PERMITS BEFORE PROTOTYPE DEADLINE

The live development schedule on the previous page is the application of the aforementioned process model to *Grabin*'s development. It is especially useful as it identifies the functional and non-functional requirements of the *Half-Grabin* museum prototype at each subsystem's development stage. It also shows the key development aspects of each subsystem, in the case of *Grabin:* coding, texturing, 3d modelling, 3d animation and sound.

Note that the development schedule has been updated to reflect changes to the project's development – the melee weapon's animation and sound effects were not finished on schedule and so are now left to be implemented at future, non-functional requirement development stages.

It is also left open to an extent. The development of subsystem 7 is included as a possible product, providing subsystem 6 development is completed well ahead of the buffered schedule time. It can be considered for the moment however that *Half-Grabin* prototype deliverables will be ready at the completion of subsystem 6 development, between the 31st May and 5th June.

The process model does not provide contingency plans and risk management procedure. It merely leaves a buffer time at the end of each subsystem's development for these procedures to be carried out. A contingency plan is outlined in Risk Management Plan

Project Requirements Tests

These are the more intangible requirements which should always be kept in mind during the project's development. Easily quantifiable requirements can be found on the Development Schedule timeline.
The tests identified for each of the following requirements will be used after the prototype is developed, in a player questionnaire to test the project's effectiveness.

1. Operational

1.1. Gesture recognition must be intuitive and simple to use for both experts and newcomers – that is to say, it must recognise gestures done at any speed and have enough error tolerance that drawing gestures is not too difficult.

TESTS:
Can testers all draw satisfactory gesture shapes?
Can gestures be successfully drawn really fast?
Can gestures be successfully drawn really slowly?
Do really inaccurate gestures fail?
Can players work out how to gesture easily?

1.2. Player movement and gesture execution must be familiar to those who play FPS games so that the learning curve is not too steep.

TESTS:
Can all FPS veterans move about the world with ease?
Can all FPS veterans instinctively work out how to gesture?

1.3. Gameplay must be balanced and enjoyable.

TESTS:
Are expert players and newcomers reasonably matched despite the skill difference?
Are more difficult gestures also more useful / powerful?
Can players who are very bad at drawing correct gestures still defend themselves?
Is there a lack of 'spammy' and annoying gestures that can be overused to irritate other players?

1.4. Readme files must be useful and informative.

TESTS:
Do players know exactly what to do ingame after they have read the documentation / help files?

1.5. The player should feel immersed in the game.

TESTS:
Does the framing and viewport of the player's character make them feel immersed in the game?
Do ingame objects interest and engage the player?
Do overly strange / surreal objects not confuse and alienate the player too much?
Is the player able to identify with the gesture system and bend it to their will after only a short time playing?

2. Visual

2.1. The gesture drawing must look visually pleasing and give information to other players as to what is being drawn by their opponents.

TESTS:
Do players find the gesture drawing effect nice to look at?
Do players who draw more slowly have less visible gestures? [relates to game balance, **1.3**]
Do player's gestures stay in the air long enough for other players to see what they are about to do and prepare defence against it? [relates to game balance, **1.3**]

2.2. The effects of attacks and other features executed by gestures must be equally or more spectacular as effects from the base game and other mods.

TESTS:
Are players impressed by the visual nature of these features?

2.3. There must be adequate visual effects in the game to provide cues on what state the player has put themself in.

TESTS:

Do visual cues such a countdown timers, crosshairs and cursors provide adequate direction to the player?

Do these visual cues appear visually impressive as well as useful?

2.4. The 3d modelling and animation of ingame objects must be of a high standard.

TESTS:

Do the 3d models and animation used in the game promote immersion and enhance the visual appeal / movement of the game?

Do other 3d modellers and animators find the modelling and animation work to be of a high standard?

3. Software

3.1. The unrealscript coding development environment must be easy to use through whichever code editor is chosen for development.

TESTS:

Is there proper syntax highlighting?

Can code be navigated easily?

Is there support for working cTag definitions? (code cross-referencing)

3.2. Future expansion of the mod must be adequately documented, and code must be sufficiently object-oriented that the addition of new gesture functions can be performed easily.

TESTS:

Are gesture functions modular in the way they are coded?

Can other developers understand the documentation enough to add their own gestures?

3.3. External programs must provide conversion into the game's file formats.

TESTS:

Can 3d models be converted easily from lightwave format and 3ds MAX format?

Can textures be imported easily from photoshop?

Can sounds be converted easily from wav and mp3 format?

Relevant Works / Practise

Gesture Recognition:

- Gesture recognition is important in the development of languages – characters and symbols which represent certain objects or ideas. Jean-Jaques Rousseau says in *The Origin of Languages* that "Pantomime without discourse will leave you nearly tranquil, discourse without gestures will wring tears from you".

- The concept of gesture recognition in computer hardware is often thought of as a more natural way of interfacing with a computer. Numerous commercial products have been developed to interpret hand gestures for controlling computer operating systems and games. These products often claim the keyboard is a clunky interface and that gestures are a clear evolutionary step to the next method of human-computer interaction.
  (Immersion Corporation, 2004, Ireality.com, 2003, Ireality.com, 2003, Toshiba America, 1998, Joel Bartlett, 2000, IdeoGramic, 2002 and Vivid Group, 2003)

- Gesture recognition through software using existing hardware has also been in development for some time. This approach, although less expandable in the long run, is much more affordable and widely distributed. It can also be applied to software in a variety of interesting and engaging ways. It is most commonly used for navigational functions at present, particularly in internet browsers.
  (Lars Bretzner, Tony Lindeberg, 1998, Optimoz, 2004, Opera Software, 2004, Bite Size, Inc, 2003, myIE2 Team, 2003 and Farlex, Inc, 2004)

- Gesture recognition in computer games has been rather infrequent in its use, but when utilised has always been considered a much more natural, intuitive method of interacting with a game. It also promotes more involvement in the game as it can be more rewarding than a simpler interface, and requires more skill to use.
  (Vivid Group, 2003, Farlex, Inc, 2004, 'Irish Player', 2004, Lionhead Studios, 2003 and Game Developer's Conference, 2004)

Game Development Methodologies

- The game development process, particularly the importance of gameplay and the ongoing design of games during their development, is embraced many famous game designers. Sid Mier, a veteran in the industry and designer of games such as *Civilisation* and *Railroad Tycoon*, says he always gets the core of a game up and running so he can constantly play and refine it during development. Peter Molyneux, the developer of the popular *Black & White*, says in a recent interview that he always invites gamers to his company to test gameplay and suggest improvements.
  (Bob Bates, 2001 and Zhan Ye, 2001)

UnrealScript Coding Language

- Various communities exist to further the understanding of the unrealscript coding language. There is a particular bevy of code enthusiasts when it comes to the unreal engine, as the game is released as open-source. These coders are often happy to help resolve each other's code difficulties.
  (BeyondUnreal, 2004 and Jelsoft Enterprises, 2004)

- There are many tutorial sites to begin an understanding of unrealscript and how it can be used to modify *Unreal Tournament 2004*'s gameplay. These are very useful for initial project development.
  (Tim Sweeny, 1998, Ray Davis, 2001 and Various, 2004)

Modelling and Animation for the Unreal Engine

- There are many tutorial sites and forums that explain the use of lightwave and 3DS MAX to create custom models and animations for the *Unreal* engine.
  (Polycount, 2004, Neomagination, 2003 and Pancho Eekels, 2004)

- Countless tutorial resources also exist on using *Unreal*'s proprietary editor, *UnrealEd*, to manage game models, textures, animations, levels and sounds.
  (Various, 2004 and Google, 2004)

Issues in First-Person Gameplay and Playability

- Thousands of articles litter the internet concerning game playability. Most crucial to multiplayer gaming are issues of enjoyable gameplay and balance.
  (Dept of Cognitive Science, 2003, Andrew Rollings, Dave Morris, 2003 and 'GreatWhite', 2003)

- Immersion within the game space is also a crucial role in FPS game development, due to the first-person viewpoint. Since the game places the player so directly and powerfully into the game, the game world must appear interesting and detailed enough that players feel immersed in the environment.
  (Eldon Alameda, 2002, TechTV, 1999 and Marc Saltzman, 2003)

Project Documentation

Treatment

The prototype implementation of *Grabin* to be delivered at the completion of this unit can be thought of as an experimental interactive piece. It is designed to provide a strongly immersive reality using the technology of the *Unreal* engine.

The gameplay of this prototype has a strong metaphorical base in 'creating art' – the player has to draw onscreen to interact with the environment, thus creating interesting visual media through action of their own. This method of interaction is also designed to provide a more natural way of interfacing with a computer, more like drawing on paper than crunching a keyboard.

The interactions players can execute in the game world are also designed to promote a higher level of player agency. Rather than the usual 9 or 10 standard weapons usually found in FPS games, *Grabin* has potential for the addition of limitless attacks and other features which could change agency in a variety of interesting ways. Of course, players will attempt to broaden their agency through lessening of other's, and an interesting agency juggling game could develop.

Gesturing Help Guide

This is the (work in progress) official documentation to be included with the multiplayer release of *Grabin*. It is written to explain the gesturing system and suggest methods for players to become better at using it, and so has some instructional value to the prototype release. It will most likely not be released with the *Half-Grabin* museum however, as it is hoped that the museum will teach players the interactions by itself.

> Welcome to the world of *Grabin* multiplayer. *Grabin* is a deathmatch – oriented game that combines fast paced action with careful thought. Instead of the same old reskinned and remade weapons, *Grabin* features an intuitive 'gesture system', whereby you must draw symbols in the air to defeat your foes.
>
> Mastering the gesture system is the key to succeeding in *Grabin*. You'll have to become proficient at drawing the shapes accurately and remembering which to use in which situation. Some attack your enemies, some counter attacks and some behave quite strangely. It's up to you to work out which are most effective.
>
> The slower you draw gestures, the less visible they will be. So if you're an expert at the game you'll have to decide between drawing really fast and letting everybody see what you're about to do, or drawing slowly to be a bit stealthy. Of course, the faster you draw the more likely you are to make a mistake…
>
> There is no ammo in *Grabin*. You are always free to do whatever you wish with a full inventory of gestures. The balance lies in their difficulty – more powerful attacks require more intricate gestures which require more time to draw, sometimes spanning multiple shapes.
>
> *How to gesture effectively*
>
> To attack someone or do something else by gesturing, you must follow these steps:
> - Always use a mouse, since there is no keyboard support.
> - Hold your secondary fire button to relinquish view control and bring up a cursor to draw with (this is known as entering gesture mode).
> - Use primary fire to draw gesture shapes. You can release the primary fire button and click it again to begin drawing another shape.
> - To cancel an unwanted gesture, you must let go of both buttons.
> - When you have finished drawing a shape or series of shapes, let go of both buttons. If you were successful, something will happen. If not, you'll have to try again.
> - For most thrown attacks, a crosshair will appear after you have gestured. It counts down from five seconds, giving you this time to hold the attack before you throw it at someone.
> - Left click while holding an attack to execute it, right click to cancel it. If the timer runs out, the attack is automatically cancelled.

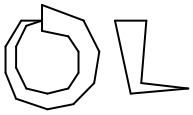
Gesture shapes are displayed at the bottom of your HUD, and can be toggled on or off by hitting your translocator key. As as example, the fireball shape looks like this:

This gives all the necessary information about the fireball gesture:
It shows that it has only one shape to the complete gesture, and
It shows the shape of the gesture (half circle counter clockwise, ie starting at the wide end of the circle).

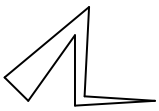
If you are unable to get any gesture shapes to recognise, try the following:

- Draw slowly and try to be as accurate as possible.
- Make sure you start and finish in the proper direction and at the proper point (for a gesture like the fireball above, you must start drawing to the right and finish drawing to the left).
- If the gesture has sharp corners in it, make sure you go around them sharply.
- For curved gestures, do not draw overly slowly as sometimes this can cause unwanted corners to be detected.

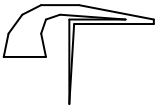
Here are some imaginary, more complicated gestures and their meaning:



Full circle clockwise, starting at 12 o'clock, followed by an 'L' shape (down then right).



A cornered shape - Up-Right, Down, Right.



A quarter circle clockwise starting at 9 o'clock, followed by Right, Left, Down (the reverse direction can sometimes be a little confusing).

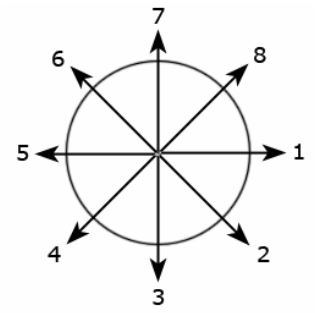
That's about all you need to know! Now get in there and start shaping up your gesture skills. Only the best will survive.

Code Expansion Guide

This guide is written mainly as a developer resource / help guide. It explains the addition of new gestures to the game.

Gesture Format

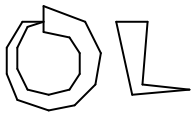
Gestures in *Grabin* take on the format of a simple string variable, for ease of expansion. Each gesture consists of a string of any length, containing numbers from 0-8. These numbers are used to determine the direction of the lines that make up a gesture, according to the following diagram:



A '0' indicates the start of a new shape – the space between 2 gestures.

This 45° / sector model is extremely accurate for recognising all forms of shape. An area 22.5° each side of a number comprises its 'tolerance angle' – the angle in which any line inside it counts for that number. Curved shapes are able to be recognised as easily as cornered ones – a circle will comprise sequentially every angle in the circle.

To provide an example, the gesture:



gives, by application of the above model, the string "0123456781031". Note that the circle shape has a "1" at both the start and end of its ID, since the shape's final angle returns to the original.

Adding a Gesture

Adding a new gesture to the game at this stage involves adding some functions to the 'grabinPRI' class. This may likely change in future builds for modularity reasons. These simple steps should guide you through it:

1. Work out what string the shape you wish to recognise should generate. Make sure it is a new shape, since gesture string ID's must be unique.
2. Add a string variable for the new gesture at the top of grabinPRI. Simply add a new name to the line 37:
   ```
   var() string fireballStr ... ;
   ```
3. Go to the base of the file and give the new string a value in defaultproperties near the others.
   ```
   fireballStr="018765"
   ...
   ```
4. Add an 'if' statement into the interp() function under the rest, using the string you declared in step 3. Make up a function name to execute for this gesture. This function will execute everything the gesture does.
   ```
   if ( strID == fireballStr )
         fireBall();
   else if ( strID == ...
   ```
5. Implement this function near the rest under the heading 'Gesture Recognition Driven Functions'. It is up to you what effect this function has. For thrown attacks, it is very easy- just tell it what projectile to use, and tell the player's pawn to set its gesture inventory's state to 'holdProjAttack'.
6. Add a replication definition for this function in the replication block at the top of the file. This is extremely important as failure to do so will mean that the gesture only works in offline games.

And that's it! Now you have a new gesture fully integrated into the game's gesture recognition system and ready to use.

Core Functionality Pseudocode

Drawing the gesturing effect in front of the player
- relates to Drawing the gesturing effect in front of the player

```
Find the position of the mouse cursor.
Convert it to screen coordinates.
Convert the screen coordinates to world cooridinates.
If it's the initial stage in a gesture
                    Create a 'pen tip'
Else
                    Move the pen tip to the new coordinates, drawing a line between
```

Gesture recognition algorithm
- relates to Gesture Recognition Algorithm

```
If a new shape starts to be drawn
      Create a gestureID array
      If it's the initial stage in a gesture shape
                  Work out a time tolerance for keeping the pen still
                  Initialise the mouse position
                  Add a '0' to the start of gesttureID
      Repeat
            Work out the current mouse position
            If the difference between current mouse pos and old mouse pos > a tolerance
                        Reset the time tolerance
                        Work out the angle between the current pos and old pos
                        Update the old pos to reflect the current pos
                        Work out what number to add to gestureID
                        Add the number to the end of gestureID
            Else if the time tolerance is exceeded
                        Reset the old mouse pos
      Until finished drawing a shape
Else If at least one shape is drawn in the gesture
                    Send the gesture array to be interpreted
```

Performing actions based on gestures
- relates to Performing Actions Through Gestures

```
If the generated string matches a predefined string
      If the predefined string is a thrown attack
                  Wait until the player activates it
                  Execute the projectile throwing function
      Else
                  Execute a particular function
```

Core Functionality Implementation

Drawing the gesturing effect in front of the player, Gesture Recognition Algorithm and Performing Actions Through Gestures

Code for the main functional areas of the mod is outlined below. Hopefully this, coupled with the pseudocode above, will give some idea of the unrealscript syntax.

Drawing the gesturing effect in front of the player

The drawing() function resides in the gestureInventory class. As a general rule, the implementation declares that code in gestureInventory is executed client side in LAN games, and code in grabinPRI is executed server side. Since the important calculations are done client side, bandwidth consumption is reduced. The function calls on the getOffset() function to determine where in front of the player the effect should spawn.

```
simulated function drawing()
{
                local vector offset;

                playerCamLoc = P.Location + P.EyePosition();
                  //set location of player cam before function execution
                offset = getOffset();
                spawnTemp.X = PC.Player.WindowsMouseX * 2.05;
                spawnTemp.Y = PC.Player.WindowsMouseY * 2.05;
                //these depend on the placement of the effect i suppose, //but i
                //hardcoded them
                  //cos vector maths are hard. Yeah, i know, i suck.
                spawnTemp.X -= offset.X;
                spawnTemp.Y -= offset.Y;
                  //vector to spawn effect to in relation to camera
                spawnTo = ir.ScreenToWorld(spawnTemp,playerCamLoc,) + (10 *
                vector(P.GetViewRotation()));
}
```

The getOffset() function is responsible for calculating where to move the effect, based on what resolution the player runs the game in. Since the 'spawnTemp' vector is calculated in pixels, higher resolutions must move the effect more to line it up with the players viewport before passing to the inbuilt function ScreenToWorld() which converts a screen vector to one in the world.

```
simulated function vector getOffset()
{
                local string CurrentRes;
                local vector halfRes;
                //the screen resolution check for per-pixel tolerance calc

                CurrentRes = PC.ConsoleCommand("GETCURRENTRES");
                switch (CurrentRes)
                {
                  case "320x240":
                        halfRes.X = 320/2;
                        halfRes.Y = 240/2;
                        break;
                  case "512x384":
                        halfRes.X = 512/2;
                        halfRes.Y = 384/2;
                        break;
                  case "640x480":
                        halfRes.X = 640/2;
                        halfRes.Y = 480/2;
                        break;
                  case "800x600":
                        halfRes.X = 800/2;
                        halfRes.Y = 600/2;
                        break;
```

```
                case "1024x768":
                        halfRes.X = 1024/2;
                        halfRes.Y = 768/2;
                        break;
                case "1152x864":
                        halfRes.X = 1152/2;
                        halfRes.Y = 864/2;
                        break;
                case "1280x960":
                        halfRes.X = 1280/2;
                        halfRes.Y = 960/2;
                        break;
                case "1280x1024":
                        halfRes.X = 1280/2;
                        halfRes.Y = 1024/2;
                        break;
                case "1600x1024":
                        halfRes.X = 1600/2;
                        halfRes.Y = 1024/2;
                        break;
                case "1600x1200":
                        halfRes.X = 1600/2;
                        halfRes.Y = 1200/2;
                        break;
                case "1920x1200":
                        halfRes.X = 1920/2;
                        halfRes.Y = 1200/2;
                        break;
                //larger resolutions can be the same tolerance
                default:
                        halfRes.X = 1024/2;
                        halfRes.Y = 768/2;
                        P.ClientMessage("Uhoh. Gestures may look dumb now cos you run
the game in a stupid resolution.");
                        break;
        }
        return halfRes;
}
```

DrawEffect() is called by the gesture recognition functions in gestureInventory to actually display the effect. It, however, resides in grabinPRI as the server must do it so that all players can see the effect. The position of the effect and the player's position are passed to it from gestureInventory, as well as the Boolean firstTime. FirstTime is needed due to the nature of the effect, defined in the class 'gesturePainter'. In a nutshell, the effect need only be spawned the first time, after which it is moved to draw a trail.

```
simulated function drawEffect(vector spawnTo, vector playerCamLoc, bool firstTime)
{
        if (firstTime)
        { //set up the drawing actor
          if (gEffect != none)
                gEffect.Destroy();
          gEffect = spawn(Class'gesturePainter',self,,,);
          gEffect.Move(playerCamLoc);
        }
        gEffect.Move(spawnTo - gEffect.Location + playerCamLoc);
          //move it each time after set up
}
```

Gesture Recognition Algorithm

Core of the mod, the gesture recognition algorithm exists as a state in gestureInventory. It is responsible for handling the player input and converting it to a string to be passed to grabinPRI for further calculation.

```
state gesturing

{
                    simulated function BeginState()
                    {
                      //log("   A NEW SHAPE A-HAPPENS ");
                      firstTime = true;
                      //get tolerance of mouse movement
                      tolerance = calcTolerance();
                      setTimer(0.02,true);
                    }

                    simulated function Timer()
                    {
                            //vector calculation variables
                      local vector newVec, diffVec;
                      local rotator angleBetween;
                      local float diffSize;
                      local int newDir; //computed direction ID

                      if (firstTime)
                      {
                              //return values to defaults
                             prevVec.X = PC.Player.WindowsMouseX;
                             prevVec.Y = PC.Player.WindowsMouseY;
                             prevVec.Z = 0;
                             diffVec = vect(0,0,0);

                              //add a blank to the end of the array for pen down
                             gestureID.insert(gestureID.Length, 1);
                             gestureID[gestureID.Length - 1] = 0;
                             ticks = 0;

                             grabinPRI(PC.PlayerReplicationInfo).drawEffect(spawnTo,
                             playerCamLoc, firstTime);
                              //call to create drawing projectile
                             firstTime = false;
                      }

                      grabinPRI(PC.PlayerReplicationInfo).drawEffect(spawnTo, playerCamLoc,
                      firstTime);
                        //call to move drawing projectile

                      newVec.X = PC.Player.WindowsMouseX;
                      newVec.Y = PC.Player.WindowsMouseY;
                      newVec.Z = 0;
                            //set up the vector holding mouse location
                      diffSize = VSize(newVec - prevVec);
                      if (diffSize > tolerance)
                      {
                        //if distance is above tolerance change diff vector and reset prevVec

                             //log("   ticks are am at " $ ticks);

                             ticks = 0;  //reset ticks of non-mouse movement
                             diffVec = newVec - prevVec;
                             prevVec = newVec;
                        //find angle between 2 vectors and compute
                             angleBetween = rotator(normal(diffVec));
                             newDir = computeOutput(angleBetween);
                             if ( gestureID[gestureID.Length-1] != newDir )
                             {  //if direction has changed
```

```
            gestureID.insert(gestureID.Length, 1);     //add an element
            gestureID[gestureID.Length - 1] = newDir; //fill the new
        }
  }
  else if (ticks > mouseSitTolerance)
  {
        prevVec.X = PC.Player.WindowsMouseX;
        prevVec.Y = PC.Player.WindowsMouseY;
        prevVec.Z = 0;
        ticks = 0;
         //this stops us getting rounded corners on straight shapes
         //a small pause in gesturing is now necessary on sharp corners.
  }
  ticks++;
}

simulated function EndState()
{
  if (grabinPRI(PC.PlayerReplicationInfo).gEffect != none)
        grabinPRI(PC.PlayerReplicationInfo).gEffect.Destroy();
    //kill the projectile which spawns the gesturing effect
}

function int computeOutput(rotator angle)
{
  // rotator notes:
  // 8192 == 45 degrees
  // TOLERANCES:
  // 1   61441 to 65535
  //      && 0 to 4095
  // 2   4096 to 12288
  // 3   12289 to 20480
  // 4   20481 to 28672                6 7 8
  // 5   28673 to 32768                 \|/
  //      && -32678 to -28673        5 --+-- 1
  // 6   -20481 to -28672             /|\
  // 7   -12289 to -20480           4 3 2
  // 8   -4096 to -12288
  local int Y; //angle's Y val, makes things easier

  Y = angle.Yaw;

  if ( 12288 >= Y && Y >= 4096 )
        return 2;
  else if ( 20480 >= Y && Y >= 12289 )
        return 3;
  else if ( 28672 >= Y && Y >= 20481 )
        return 4;
  else if ( (32768 >= Y && Y >= 28673) || (-32768 <= Y && Y <= -28673) )
        return 5;
  else if ( -28672 <= Y && Y <= -20481 )
        return 6;
  else if ( -20480 <= Y && Y <= -12289 )
        return 7;
  else if ( -12288 <= Y && Y <= -4096 )
        return 8;
  else
        return 1;           //find out which direction we draw in
}

function float calcTolerance()
{
  local string CurrentRes;
        //the screen resolution check for per-pixel tolerance calc
  CurrentRes = PC.ConsoleCommand("GETCURRENTRES");
  switch (CurrentRes)
  {
```

```
                        case "320x240":
                                tolerance = 24;
                                break;
                        case "512x384":
                                tolerance = 38;
                                break;


                        case "640x480":
                                tolerance = 48;
                                break;
                        //larger resolutions can be the same tolerance
                        default:
                                tolerance = 50;
                                break;
                  }
                  return tolerance;
                }
}
```

Performing Actions Through Gestures

The interp() function gets the gesture's ID array from gestureInventory and works out what to do. Note that gravity manipulation is different from the other if statements as it must modify gravity a different way for each direction indicated in the second gesture shape.

```
simulated function bool interp(array<int> gestureID)
{
                local int i;
                local string strID;
                  //this is kind of dumb, but better accessible. It stores
                  // the ID array as an easy, non-associative datatype.
                  //Stops us needing many iterators.

                for ( i=0; i < gestureID.Length; i++ )
                {  //converts to string
                  strID = strID $ gestureID[i];
                }

                debugStr = strID;     //FOR DEBUGGING

                setDefProj();
                if ( strID == fireballStr )
                  fireBall();

                 //gravity manipulation recognition needs special attention

                else if ( (strID == (gravStr $ "1")) || (strID == (gravStr $ "2")) ||
                    (strID == (gravStr $ "3")) || (strID == (gravStr $ "4")) ||
                    (strID == (gravStr $ "5")) || (strID == (gravStr $ "6")) ||
                    (strID == (gravStr $ "7")) || (strID == (gravStr $ "8")) )
                  changeGrav(gestureID[gestureID.Length-1]);

                return true;
}
```

SetDefProj() just resets all the variables for thrown attacks so that it can be done more easily.

```
function setDefProj()
{  //makes setting the values for firing 1 projectile easier
                ProjSpawnOffset.X = 35;
                ProjPerFire = 1;
                DamageAtten = 1.0;
                SpreadStyle=SS_None;
                Spread = 0.0;
}
```

The fireball's execution function is really easy now.

```
function fireBall()
{
                ProjectileClass = Class'skaarjPack.GasBagBelch';
                grabinPawn(Controller(Owner).Pawn).setGestureState('holdProjAttack');
}

function changeGrav(int dirID)
{
                //Not done yet
}
```

This state actually exists under gestureInventory. It facilitates the countdown timer for the player and handles everything to do with thrown attacks. The DoProjectile() function it calls in grabinPRI is a complicated function which generates the attack and throws it from the player. It is left out of this documentation as it is almost exactly a copy of the game standard weapon fire code.

```
state holdProjAttack
{  //hold the projectile attack until the user clicks again or 5 seconds
                simulated function beginState()
                {
                  countDown = holdTime;
                  setTimer(1.0,true);
                }

                simulated function timer()
                {
                  countDown--;
                  if (countDown <= 0)
                         gotoState('');
                }

                simulated function endState()
                {
                  if ( cRelease )
                         grabinPRI(PC.PlayerReplicationInfo).DoProjectile();
                  cRelease = false;
                  countDown = holdTime+1;
                }
}
```

UML State and Class Diagrams

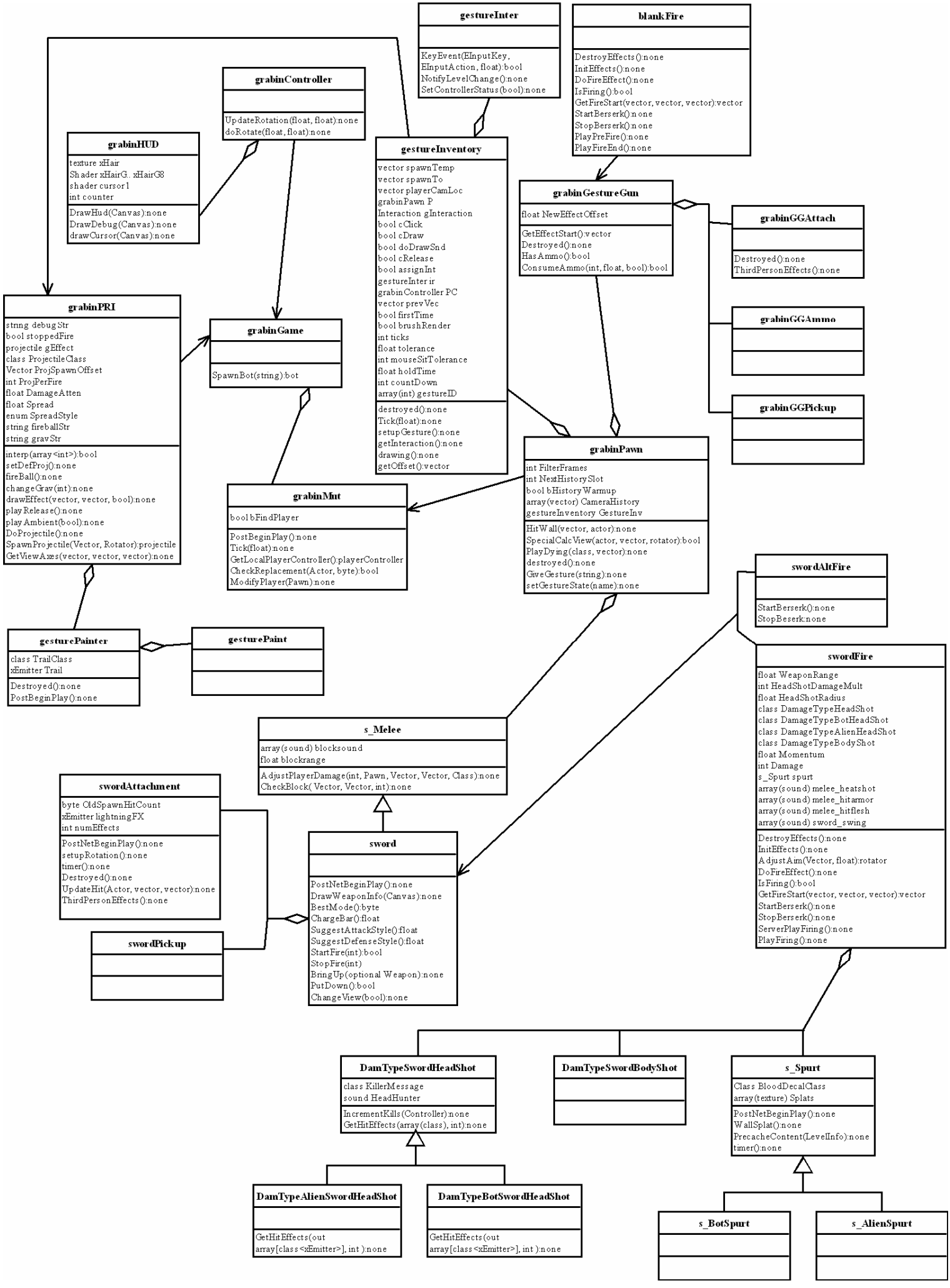
For the purposes of this project, use-case diagrams, interaction diagrams, activity diagrams and physical diagrams are unnecessary for modelling the important program aspects.

Of particular importance are state diagrams, for examining object state flow, and class diagrams, for keeping track of the complex class hierarchy of the project.

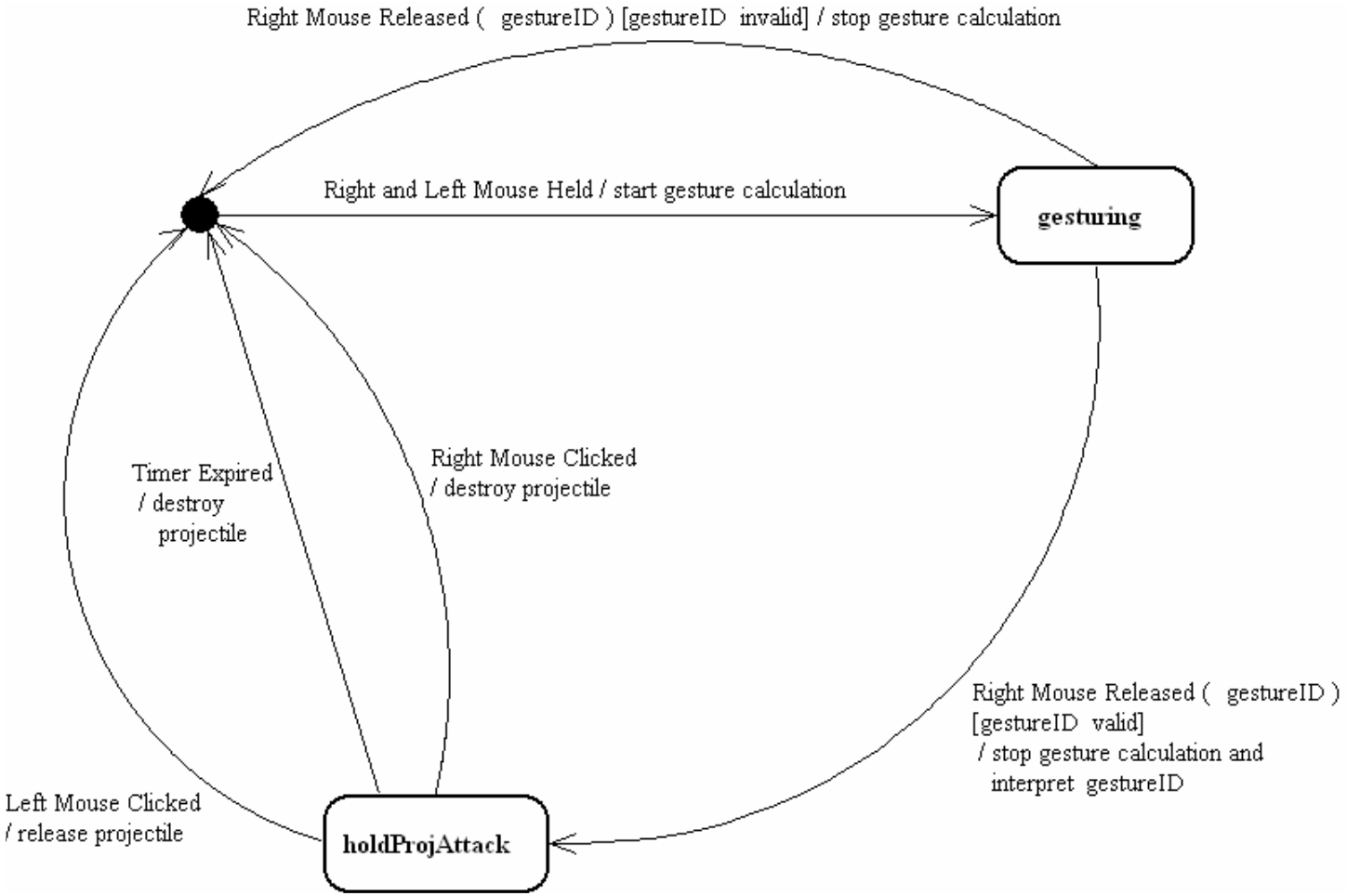
The Class Diagram is representative of UML, although there are certain changes to the syntax to make it useful for our purposes. It shows only aggregation, association and inheritance, as the addition of other relationships would make the diagram much too confusing. Also, for the purposes of programming in the *Unreal* engine, most classes depend on one another in at least a minimal way. It shows class variables by their type and name only, as initial values for variables are declared in a different way to most programming languages. It also hides any inherited variable or method names, so there are some classes - designed solely to change default properties - which appear empty. At the time of writing, all major classes are developed and so the addition of new classes to the diagram is left to the author's discretion.

A State Diagram exists only for gestureInventory. This is the class which provides core gesturing functionality, and also implements the state sequence that the player must interface with. All other classes have only limited or inbuilt state changes, which are hidden from the player by design.

Class Diagram



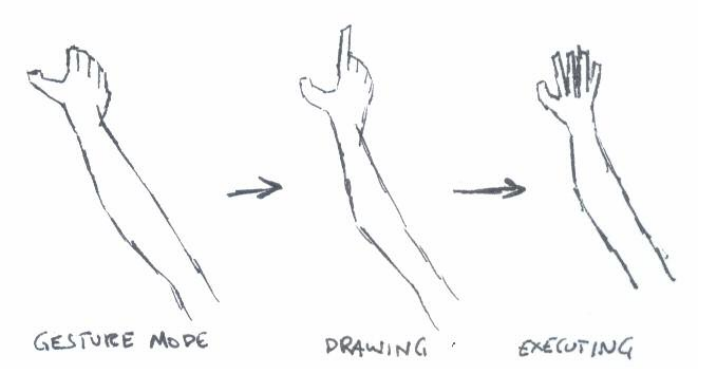
State Diagram for gestureInventory



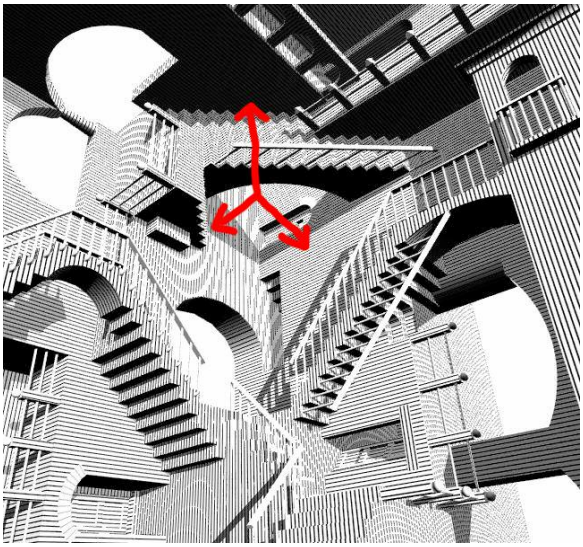
Design Sketches

This is something like how the gesture drawing effect should look. In any case, it should really embody an 'ink on paper' aesthetic.

This diagram shows the visual components of the fireball gesture. Several particle emitters will eventually make up the effect.



These three sketches show the animation of the player's hand in their view, based on what state the player is in. When the player is not gesturing, they will not see the hand at all.



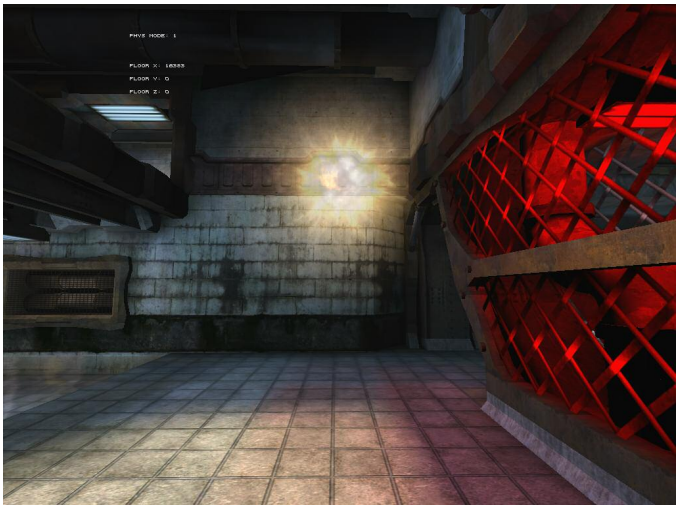
This is a conceptual diagram from an Escher sketch to show the possibilities of the gravity manipulation gesture. This will probably be the basis for the first level of *Grabin* and part of the *Half-Grabin* museum. Players could change their gravity direction and walk on any of these surfaces, providing some very interesting gameplay.

Early Implementation Screenshots

A test build of the gesture drawing effect, drawn fast (and hence easily visible). Note that the gesture effect still needs some work for it to be properly aesthetically pleasing, and these screenshots are taken from a test map that ships with *UT2004*.

The gesture drawing effect when done slowly for stealth.

Elements of the fireball spell.



The impact effect when a fireball collides with something (in this case a wall).

An earlier gesture effect shown working in a multiplayer game.

Risk Management Plan

Subsystem	Risk	Probability	F / NF	Impact description	Management action
Design	Cannot think of any plausible ideas for the mod.	Very Low	Functional	The project fails before it has even begun.	Ensure proper documentation to record all ideas.
	Early prototypes are too complex to be developed.	Medium	Functional	Another cycle of the design spiral must be completed to refine project ideas and goals.	Generate ideas which are plausible for development.
	Prototypes are not possible due to limitations in the technology.	Medium	Functional	Another cycle of the design spiral must be completed to refine project ideas and goals.	Generate ideas which are plausible for development.
1	Failure to set up a base for the mod to run on.	Very Low	Functional	The project cannot be persued.	View online tutorials and knowledge to ensure this runs smoothly.
	Shape drawing proves too complex to code or impossible due to technology limitations.	Low	Functional	The project cannot be persued.	View online tutorials on unrealscript coding and ensure the code works.
	Refining the drawing effect to make it more aesthetically pleasing is too difficult.	Very Low	Non-Functional	Gesturing will remain unnatractive until later build stages.	Ensure functional requirements for this subsystem are completed in time to develop this requirement.
	Unable to work out how to implement sounds.	Low	Non-Functional	The project will have few or sparse sounds and will be much less interesting.	View online documentation and tutorials on how to manipulate sound.
2	PHP and other coding for the website is too difficult.	Very Low	Functional	No website means generating interest in the project will be much more difficult.	Obtain outside help on website design if absolutely necessary.
	Graphics and visual elements prove too time consuming.	Very Low	Functional	No website means generating interest in the project will be much more difficult.	Obtain outside help on website design if absolutely necessary.
	Creating a development journal is too difficult.	Very Low	Non-Functional	Not having a development journal means keeping people updated on progress is much more difficult.	Obtain outside help on website design if absolutely necessary.
3	Coding melee attacks is impossible due to technology restrictions.	Very Low	Functional	The game becomes unbalanced without the ability to attack other players easily up-close.	Examine other melee weapon code in games and try to duplicate it.

Subsystem	Risk	Probability	F / NF	Impact description	Management action
3	Animating the melee weapn proves too time consuming.	High	Non-Functional	The melee weapon remains unanimated until a later subsystem's implementation.	Ensure functional requirements for this subsystem are completed in time to develop this requirement, or develop it in a later subsystem.
	Texturing the melee weapon proves too difficult.	Very Low	Non-Functional	The melee weapon remains untextured until a later subsystem's implementation.	Ensure there is adequate time available to develop this requirement, or develop it in a later subsystem.
	Modelling the melee weapon proves too difficult.	Low	Non-Functional	The melee weapon only hurts people without any visible effect until a later development phase.	Ensure there is adequate time available to develop this requirement, or develop it in a later subsystem.
	Creating melee sounds is too time consuming.	Medium	Non-Functional	The melee weapon is silent until a later development phase.	Ensure there is adequate time available to develop this requirement, or develop it in a later subsystem.
4	Gesture recognition is impossible to code due to technology restrictions.	Very Low	Functional	The project cannot be persued.	View online tutorials and knowledge to ensure this runs smoothly. Note that the successful completion of subsystem 1 almost guarantees the development of this requirement.
	Code optimisation and net replication issues are too confusing to make the mod multiplayer compliant.	Low	Non-Functional	The multiplayer aspects of the project cannot work, and ideas such as 'guided tours' of *Half-Grabin*are impossible.	View online documentation and tutorials on net replication, ask for help on forums if needed.
5	Modelling of the player's hand and importing into the game engine is too difficult.	Low	Functional	A portion of the player's immersion in the game is lost as they see only an empty screen before them.	Viewing of tutorials and documentation.
	Texturing of the player's hand is too difficult.	Very Low	Functional	A portion of the player's immersion in the game is lost as they see only an untextured hand before them.	Viewing of tutorials and documentation.
	Further optimisation and smoothing of the gesturing effect doesn't happen.	Medium	Non-Functional	Players may be less impressed by the visual appeal of the game.	Ensure there is adequate time available to develop this requirement, or develop it in a later subsystem.
	Creating new textures for the gesturing effect is too time consuming.	Low	Non-Functional	Players may be less impressed by the visual appeal of the game.	Use standard textures from within the game, or develop custom ones later on.

Subsystem	Risk	Probability	F / NF	Impact description	Management action
5	Additional animation of the player's hand is too time consuming.	High	Non-Functional	Players may be less impressed by the visual appeal of the game.	Ensure there is adequate time available to develop this requirement, or develop it in a later subsystem.
	Custom sound effects take too long to produce.	Medium	Non-Functional	Players may be less impressed by the aural appeal of the game.	Ensure there is adequate time available to develop this requirement, or develop it in a later subsystem.
6	Gravity manipulation proves impossible due to constraints in the technology.	Medium	Functional	The levitation gesture cannot be developed.	View online tutorials and knowledge to ensure this runs smoothly, and is in fact possible.
	Creation of a fireball attack takes too long.	Low	Functional	There will be no fireball attack.	View online tutorials and knowledge to ensure this runs smoothly.
	Custom textures for the fireball attack take too long to develop.	Medium	Non-Functional	Players may be less impressed by the visual appeal of the game.	Use standard textures from within the game, or develop custom ones later on.
	Custom textures for the levitation gesture take too long to develop.	Medium	Non-Functional	Players may be less impressed by the visual appeal of the game.	Use standard textures from within the game, or develop custom ones later on.
	Custom sounds for the fireball, levitation effect and gesturing in general take too long to produce.	Medium	Non-Functional	Players may be less impressed by the aural appeal of the game.	Ensure there is adequate time available to develop this requirement, or use stock game sounds until able to develop it in a later subsystem.
If Time Constraints Permit					
7	An interface between players and world objects is impossible to create.	Very Low	Functional	Gestures are only able to effect the player.	View online tutorials and knowledge to ensure this runs smoothly, and is in fact possible.
	Creation of a door which reacts to gestures is limited by the technology.	Very Low	Non-Functional	Doors cannot be used for gesture manipulation.	Possibly expand different world objects or leave this to a later development stage.
	Texturing the door takes too long.	Medium	Non-Functional	Players may be less impressed by the visual appeal of the game.	Use a stock door texture.
	Modelling and animating the door takes too long.	Medium	Non-Functional	Players may be less impressed by the visual appeal of the game.	Use a stock door model.
	Creating sounds for the door takes too long.	Medium	Non-Functional	Players may be less impressed by the aural appeal of the game.	Use a stock door's sound effects.

References

1. **NxN software,** 2001 - *AlienBrain 4.0 Case Studies,* "Alienbrain - Asset Management for Creative Teams" retrieved from http://www.nxn-software.com/ref_cast.php on 25 April, 2004

2. **Roger Pressman,** 1997 - *Software Engineering: A Practitioner's Approach*, McGraw-Hill, pp57-74

3. **Frederic Brooks,** 1995 - *The Mythical Man-Month*, Addison-Wesley, pp14-26

4. **Michael Cusumano, Richard Selby,** 1998 - *Microsoft Secret*, Touchstone, pp187-207

5. **Randy Angle, William Dwyer,** 2001 - *Effective Project Management*, "Proceedings of Game Developers Conference 2001"

6. **Bob Bates,** 2001 - *Game Design: The Art & Business of Creating Games,* Prima Publishing, pp256-265

7. **Bob Bates,** 2001 - *Game Design: The Art & Business of Creating Games,* Prima Publishing, pp229-230

8. **Zhan Ye,** 2001 - *Return to E3*, "Game Software Magazine" July Issue

9. **Pugh, S,** 1991 - *Total design: integrated methods for successful product engineering*, Addison-Wesley, Wokingham

10. **Jacob Marner,** 2002 **-** *Evaluating Java for Game Development,* "Rolemaker – A Computer Roleplaying Game Authoring System" retrieved from http://www.rolemaker.dk/articles/evaljava/Evaluating%20Java%20for%20Game%20Development.pdf on 27 April, 2004

11. **Immersion Corporation,** 2004 - *3D Interaction: Products,* "Immersion Corporation" retrieved from http://www.immersion.com/3d/products/cyber_glove.php on 28 April, 2004

12. **Ireality.com,** 2003 - *Motion Capture Solutions,* "General Reality Company" retrieved from http://www.genreality.com/motioncapture.html on 28 April, 2004

13. **Ireality.com,** 2003 - *5DT Data Glove (5 Sensor),* "General Reality Company" retrieved from http://www.genreality.com/p_glove5.html on 28 April, 2004

14. **Toshiba America,** 1998 - *Toshiba's Motion Processor Recognizes Gestures in Real Time--Basis for Future Generation of Natural Interfaces between People and Computers,* "Toshiba America, Inc" retrieved from http://www.toshiba.com/news/980715.htm on 28 April, 2004

15. **Joel Bartlett,** 2000 - *Rock 'n' Scroll Is Here to Stay*, "IEEE Computer Graphics and Applications" May/June Issue, pp40-45

16. **IdeoGramic,** 2002 - *Ideogramic: Products and Services,* "Ideogramic.com" retrieved from http://www.ideogramic.com/products/ on 28 April, 2004

17. **Vivid Group,** 2003 - *The Vivid Mandala GX System – Wireless Virtual Reality Games,* "Vivid Group.com" retrieved from http://www.vividgroup.com/ on 28 April, 2004

18. **Lars Bretzner, Tony Lindeberg,** 1998 - *Use Your Hand as a 3-D Mouse, or, Relative Orientation from Extended Sequences of Sparse Point and Line Correspondences Using the Affine Trifocal Tensor,* "Computational Vision and Active Perception Laboratory" retrieved from http://www.nada.kth.se/cvap/abstracts/brelin-eccv98.html on 28 April, 2004

19. **Optimoz,** 2004 - *Optimoz: Gestures/Installation,* "mozdev.org" retrieved from http://optimoz.mozdev.org/gestures/installation.html on 28 April, 2004

20. **Opera Software,** 2004 - *Mouse Gestures in Opera,* "Opera.com" retrieved from http://www.opera.com/features/mouse/ on 28 April, 2004

21. **Bite Size, Inc,** 2003 - *Mouse Gestures – Controlled by Motion*, "BiteSizeInc" retrieved from http://www.bitesizeinc.net/index.php/gesture.html on 28 April, 2004

22. **myIE2 Team,** 2003 - *Mouse Gestures*, "MyIE2 Online" retrieved from http://www.myie2.com/html_en/tour/02mousegesture.htm on 28 April, 2004

23. **Strokeit Team,** 2002 - *Mouse Gestures For Windows,* "StrokeIt.com" retrieved from http://www.tcbmi.com/strokeit/ on 28 April, 2004

24. **Farlex, Inc,** 2004 - *Mouse Gesture,* "The Free Dictionary" retrieved from http://encyclopedia.thefreedictionary.com/mouse%20gestures on 28 April, 2004

25. **'Irish Player',** 2004 - *Arx Fatalis,* "Campus.ie" retrieved from http://www.campus.ie/user?cmd=item-detail&itemid=4796 on 28 April, 2004

26. **Lionhead Studios,** 2003 - *Black and White*, "bwgame.com", retrieved from http://www2.bwgame.com/?url=/bw/BWABOUT on 20 March, 2004

27. **Game Developer's Conference,** 2004 - *GDC '04 Conference Guide* retrieved from www.gdconf.com/conference/GDC04_confguide.pdf on 28 April, 2004

28. **BeyondUnreal,** 2004 - *BeyondUnreal Forums: Unreal Development*, "BeyondUnreal.com" retrieved from http://forums.beyondunreal.com/forumdisplay.php?f=13 on 13 February, 2004

29. **Polycount,** 2004 - *Polycount Messageboard: 2d and 3d Discussion,* "Polycount" retrieved from http://dynamic.gamespy.com/~polycount/ubb/forumdisplay.cgi?action=topics&forum=2D+and+3D+Discussion&number=8&DaysPrune=10&LastLogin= on March 17, 2004

30. **Jelsoft Enterprises,** 2004 - *Gaming Forums: Unreal Tournament Series,* "GamingForums" retrieved from http://www.gamingforums.com/forumdisplay.php?f=148 on 28 April, 2004

31. **Tim Sweeny,** 1998 - *Unrealscript Language Reference,* "unreal.epicgames.com" retrieved from http://unreal.epicgames.com/ on 28 April, 2004

32. **Ray Davis,** 2001 - *CHIMERIC – The Unrealscript Coding Resource,* "BeyondUnreal" retrieved from http://chimeric.beyondunreal.com/tutorials.php on 28 April, 2004

33. **Various,** 2004 - *The Unreal Wiki,* "BeyondUnreal" retrieved from http://wiki.beyondunreal.com/wiki/ on 28 April, 2004

34. **Neomagination,** 2003 - *MAX to UT2003 Import,* "neomagination" retrieved from http://www.neomagination.com/unreal/tutorials/max2ut_01.html on 28 April, 2004

35. **Pancho Eekels,** 2004 - *Importing Level Models for UT2003,* "newtek" retrieved from http://www.newtek.com/products/lightwave/tutorials/modeling/UT2003/ on 28 April, 2004

36. **Google,** 2004 - *Numerous Sites Providing UnrealEd Tutorials,* "Google" retrieved from http://www.google.com.au/search?q=unrealed+tutorials&ie=UTF-8&oe=UTF-8&hl=en&meta= on 28 April, 2004

37. **Dept of Cognitive Science,** 2003 - *Introduction to Cognition and Gaming,* "" retrieved from http://www.cogsci.rpi.edu/courses/icg/lectures/lecture8.ppt on 28 April, 2004

38. **Andrew Rollings, Dave Morris,** 2003 - *Game Architecture and Design: A New Edition,* "Peach Pit" retrieved from http://www.peachpit.com/title/0735713634 on 28 April, 2004

39. **'GreatWhite',** 2003 - *Gameplay Balance Designers Diary,* "RoN Oracle Forums" retrieved from http://www.ronoracle.com/forums/index.php?s=98e9b4d84a03693b04779ef29cd7ceaf&showtopic=2606&st=0&#entry35104 on 28 April, 2004

40. **Eldon Alameda,** 2002 - *Breakdown Review,* "Console Gold" retrieved from http://www.consolegold.com/reviews/Reviews.php?ReviewID=136 on 28 April, 2004

41. **TechTV,** 1999 - *The First-Person Shooter Genre,* 'Tech TV" retrieved from http://www.techtv.com/extendedplay/videofeatures/story/0,24330,2388258,00.html on 28 April, 2004

42. **Marc Saltzman,** 2003 - *Game Playing Perspectives,* "Inform IT" retrieved from http://www.informit.com/articles/article.asp?p=98834 on 28 April, 2004

Bibliography

Roger Pressman (1997) *Software Engineering: A Practitioner's Approach*, McGraw-Hill

CMP Media, LLC (2004) *Gamasutra – The Art and Science of Making Games*, http://www.gamasutra.com/

CMP Media, LLC (2004) *Game Developer's Conference*, http://www.gdconf.com/

Bob Bates (2001) *Game Design: The Art & Business of Creating Games,* Prima Publishing

Tim Sweeny, Epic Megagames (2003) *Unreal Technology,* http://unreal.epicgames.com/

F&W Publications (2004) *The International Design Magazine*, F&W Publications

CMP Media, LLC (2004) *Game Developer Magazine,* CMP Media

Richard Rouse (2001) *Game Design: Theory & Practise*, Wordware Publishing