|  |
| --- |
|  |
| Princess with a Rocket Launcher |
| Technical Design Document |
|  |
| **Fall 2014 GAM430** |
|  |

|  |
| --- |
|  |

Table of Contents

[Architecture 4](#_Toc406493851)

[Primary programming environment 4](#_Toc406493852)

[Descriptions 4](#_Toc406493853)

[Flow Charts 4](#_Toc406493854)

[Coding Standards 5](#_Toc406493855)

[Naming standards 5](#_Toc406493856)

[Style standards for classes, functions, etc... 5](#_Toc406493857)

[Tools 6](#_Toc406493858)

[Standard Applications 6](#_Toc406493859)

[Content Editing Applications 6](#_Toc406493860)

[Game Engine 7](#_Toc406493861)

[Licensed Technology(s) 7](#_Toc406493862)

[Risks and Contingencies (What-Ifs) 8](#_Toc406493863)

[Risk Assessment plan 8](#_Toc406493864)

[Disaster Recovery plan 8](#_Toc406493865)

[Security (policies and procedures) 9](#_Toc406493866)

[Security Policy 9](#_Toc406493867)

[Revision Control 10](#_Toc406493868)

[Revision Control System (description) 10](#_Toc406493869)

[Revision Control procedures 11](#_Toc406493870)

[Artificial Intelligence 12](#_Toc406493871)

[Software 12](#_Toc406493872)

[Style 12](#_Toc406493873)

[Physics (describe where necessary) 13](#_Toc406493874)

[Rigid body dynamics 13](#_Toc406493875)

[Soft body dynamics 13](#_Toc406493876)

[Fluid dynamics 13](#_Toc406493877)

[Collision Detection 13](#_Toc406493878)

[Hair and cloth simulation 13](#_Toc406493879)

[Finite-element fracturing 13](#_Toc406493880)

[Input/Output (I/O) 14](#_Toc406493881)

[Media formats 14](#_Toc406493882)

[Streaming 15](#_Toc406493883)

[Hardware Considerations 17](#_Toc406493884)

[Minimal Specifications 17](#_Toc406493885)

[Multiprocessors 17](#_Toc406493886)

[Performance goals 17](#_Toc406493887)

[Multiplayer and Networking 18](#_Toc406493888)

[Client-Server vs Peer-to-peer 18](#_Toc406493889)

[Communication protocols 18](#_Toc406493890)

[Number of players supported 18](#_Toc406493891)

[Matchmaking process 18](#_Toc406493892)

["Join-in" and "Drop-out" procedures 18](#_Toc406493893)

[Latency and lost packets 18](#_Toc406493894)

[Managing cheating / hacking 18](#_Toc406493895)

[Clans, observers, tournaments 18](#_Toc406493896)

[Graphics 19](#_Toc406493897)

[3D Pipeline 19](#_Toc406493898)

[User Interface (UI) 19](#_Toc406493899)

[Art tools 20](#_Toc406493900)

[Sound 21](#_Toc406493901)

[Architecture 21](#_Toc406493902)

[Encoding 21](#_Toc406493903)

[Quality / Noise reduction 21](#_Toc406493904)

[Sample Rate 21](#_Toc406493905)

[Stereo vs. Mono 22](#_Toc406493906)

[Localization 23](#_Toc406493907)

[No Localization will be performed for this Game Project. 23](#_Toc406493908)

[Technical Design Review (TDR) 24](#_Toc406493909)

[Prototype 25](#_Toc406493910)

[Paper Prototype 25](#_Toc406493911)

[Digital Prototype 25](#_Toc406493912)

# Architecture

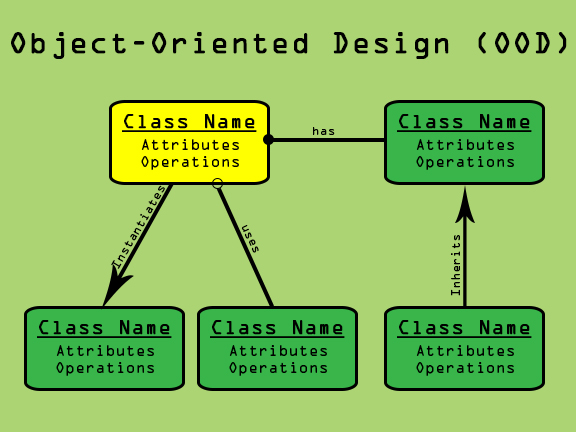
## Primary programming environment

The primary programming environment will be within the Unreal Engine using BluePrints and Unreal Scripting. As per Unreal’s own guidelines, each UnrealScript class should be in its own .uc extension text file. However, for ease of development programmers will be using Visio for creation and debugging of the programming code.

## Descriptions

For Princess with a Rocket Launcher the team will be using object-oriented design within Unreal’s BluePrints – a visual scripting method and UnrealScript – traditional OOD programming systems. The systems will be a collection of objects that act on each other. All scripting, behaviors, physics, and triggers will be handled within this architecture.

## Flow Charts



# Coding Standards

There are numerous reasons it is important for us to maintain coding standards.

1. It is important that programmers be able to easily read and understand each other's code.
2. People make fewer mistakes in consistent environments.
3. Standardized code is more easily maintainable by different programmers.
4. Standards mean that new people can get up to speed quickly.

If a programmer must violate a standard, the reason for doing so must be strong and clearly documented via code comments.

## Naming standards

1. Clear and informative names are one of the best ways to create easily understandable code.
2. The name of an identifier should succinctly describe the purpose of that identifier.
3. Avoid abbreviations except where the abbreviation is an unambiguous industry standard.
4. All names should be written in English.
5. Identifier names should not begin with an underscore.

## Style standards for classes, functions, etc...

# Tools

## Standard Applications

Microsoft Word will be used to create all documentation, communication memorandums, and script/dialog. Microsoft Excel will be used to support documentation with statistic sheets and organizational charts. Microsoft PowerPoint will be used for presentation purposes. In addition, Microsoft Visio will be used for level design paper maps when applicable.

## Content Editing Applications

3D Studio Max® will be used, in addition to Autodesk’s Maya 3D modeling software. Adobe Creative Suite, including Photoshop and Illustrator, will be used for graphic design, texturing, and paper prototyping. Unreal Engine 4 will be used as the primary game engine and level editing software. Audacity will be used for audio software recording and editing purposes.

# Game Engine

Epic’s Unreal Development Kit will be the primary game editor and game engine. Primarily a First Person Shooter developer, UDK will work well with the overall development of Princess with a Rocket Launcher. The game engine also allows for compatibility between 3ds Max, Maya, ZBrush, Photoshop, SpeedTree, Bink, and FaceFX.

## Licensed Technology(s)

Unreal Development Kit: Educational institutions can use UDK under the "FREE for educational and non-commercial use" policy. The rights to develop and release a game for free are contained in the end-user license agreement (EULA).

\*Unreal Development Kit (UDK) © 2009, Epic Games, Inc. Epic, Epic Games, Gears of War, Gears of War 2, Unreal, AnimSet Viewer, AnimTree Editor, Unreal Cascade, Unreal Content Browser, Unreal Development Kit, Unreal Editor, Unreal Engine, Unreal Kismet, Unreal Lightmass, Unreal Matinee, Unreal PhAT, UnrealScript and Unreal Tournament are trademarks or registered trademarks of Epic Games, Inc. in the United States of America and elsewhere. All other trademarks are the property of their respective owners.

# Risks and Contingencies (What-Ifs)

This section will outline the possible risks of what could happen while working on the project, in addition to remedies to the problems that may occur.

## Risk Assessment plan

In the event of a loss of data

Section deals with the problem of data loss

Saving and backups should be regular when working on the project. In the event of lost data, refer to a previous build date. Artists, designers, and programming leads should be responsible for their ends, as well as the project lead to have the master builds. These builds should be updated and backed up a few times a day to ensure that in the event of loss, the amount to work will not be delayed. Backed up files should be saved to flash drives or external hard drives.

Some things to consider:

- Save often

- Make sure nothing gets lost and back up all work no matter how small.

- Supervisors should be watching after everyone's work ot make sure everything is  backed up and saved.

In the event of hacking

(Section reserved for information dealing with malicious hacking. Possibly refer to security section for further details regarding)

In the event of hardware malfunction

Section dealing with the event of a hardware malfunction or destruction.

Small scale: (Minor hardware malfunctions such as computer crashing or computer getting unplugged)

Large Scale (flooding, overheating, terrorist attack, etc.)

In the event of a schedule slip: (Section intended offer suggestions in the event of a schedule slip and the project falls behind.)

## Disaster Recovery plan

Backups of all development data shall be backed up on-site and off-site every week. Master files shall also be backed up off-site with versions kept for every major milestone in addition to the regular back up procedures and timeline.

# Security (policies and procedures)

This section outlines security protocols and procedures to safely conduct work within the workplace to secure the project’s assets. Section also outlines the roles and responsibilities for the security team. Anyone who signs the agreement is legally bound to adhere to the security policies for the project.

## Security Policy

Asset Classification

Assets need to classed into who needs what and include license and copyright restrictions.

Access Control

To prevent theft of assets or any other data, prevention protocols need to be made. Certain people will have access to certain assets and information.

Asset Protection

Outline of how the assets and other information will be protected during the project’s lifespan.

Compliance

Outline of how the security protocols will be enforced and managed.

Examples of security policies (Really good ones mind you:

<http://www.dmoz.org/Computers/Security/Policy/Sample_Policies/>

http://www.sans.org/security-resources/policies

# Revision Control

## Revision Control System (description)

The Git feature that really makes it stand apart from nearly every other SCM out there is its branching model.

Git allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.

This means that you can do things like:

* **Frictionless Context Switching**. Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.
* **Role-Based Codelines**. Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work.
* **Feature Based Workflow**. Create new branches for each new feature you're working on so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.
* **Disposable Experimentation**. Create a branch to experiment in, realize it's not going to work, and just delete it - abandoning the work—with nobody else ever seeing it (even if you've pushed other branches in the meantime).



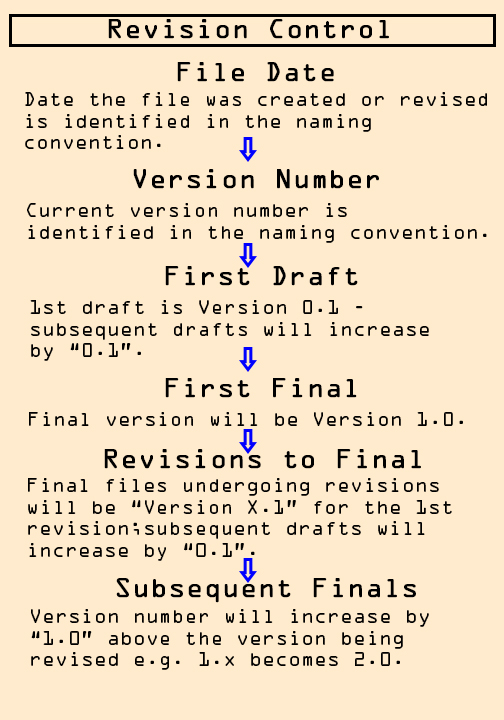
Notably, when you push to a remote repository, you do not have to push all of your branches. You can choose to share just one of your branches, a few of them, or all of them. This tends to free people to try new ideas without worrying about having to plan how and when they are going to merge it in or share it with others.

There are ways to accomplish some of this with other systems, but the work involved is much more difficult and error-prone. Git makes this process incredibly easy and it changes the way most developers work when they learn it.

(<http://git-scm.com/about>)

## Revision Control procedures

The development team will follow the check-out/check-in system enforced by the architecture of the GIT revision system to ensure work does not get overridden. Team leads are responsible for merging all workflow back into a master file at the end of the work day/week. The lead programmer is responsible for ensuring a weekly build to show team progress.



# Artificial Intelligence

## Software

The Artificial Intelligence will be handled using Unreal’s BluePrints, in-engine

pathfinding, and UnrealScripts (.uc/Visio files).

## Style

The AI will be balanced between believable AI and performance in order to optimize the game. Shortcuts may be made using the in-engine pathnode system in order to give the AI a realistic looking movement and destination within the game.

# Physics (describe where necessary)

## Rigid body dynamics

This type of physic dynamic is used to create objects that will retain their shapes and composure even after physical interaction. This type of software is found as a standard in most game creation software and does not need to be created. Rigid body dynamics will be found in the assets of the games such as the rocks found in the environment or the building structures found throughout the game.

## Soft body dynamics

This type of physic dynamic is used primarily for creating objects that are solid but can be manipulated under certain circumstances. Soft Body dynamics may be found in character design, primarily the skin and character to character/environment physical interaction. These physics may also be found in the jumping platforms that may shift under Princess Margaret’s weight. Instances of the usage of soft body physics may be found in the pieces of destructible environment, hair, trees, and cloth.

## Fluid dynamics

This type of dynamic will be found in the moving portions of the environments, mainly water and shifting trees as a means to simulate a moving air current.

## Collision Detection

Standard in development kits, this type of physics is mainly used to show the actual tangible portions of the game. Characters and assets may use colliders for movement purposes.

## Hair and cloth simulation

Using APEX Clothing (PhysX Clothing) physical mesh is constrained to the animated mesh and is fully controlled by the artist to create realistic acting hair and cloth.

## Finite-element fracturing

<http://udn.epicgames.com/Three/DevelopmentKitFAQ.html>

<http://en.wikipedia.org/wiki/Multiprocessing>

<http://en.wikipedia.org/wiki/Unreal_Engine#Unreal_Development_Kit>

# Input/Output (I/O)

## Media formats

As Princess with a Rocket Launcher is being developed for the PC platform, the media format will be made available for both dual-layer DVD-ROM and Digital Download media formats. For the DVD-ROM media format, the following considerations need to be taken into account: storage capacity of 9.4 GB and data transfer rate (1x) of 11.08 Mbps.

The primary systems of the game are allowed to allocate up to the following amounts of memory

for internal use:

**Entity                                                    Reserved Memory**

UDK Functions/Engine 96KB

Key Bindings 128KB

Audio 16MB

Game Logic 4KB

Physics Manager 8KB

Level Manager 4KB

Widget Manager 4KB

Level Instance 1MB

AI Manager 4KB

Text Manager 8KB

Total: 17,256KB

Assuming an executable file of 3MB, this brings total system memory usage to ~20MB.

(Reserved Memory figures based off of/come from the TDD for [*Narbacular Drop*](http://jeepbarnett.com/nuclearmonkeysoftware/documents/narbacular_drop_technical_design_document.pdf)'s "System Memory" section - Pg. 14)

## Streaming

Due to the data transfer rate of 11.08 Mbps on DVD-ROM, data will be stored in interleaved format and retrieved continuously so that players can traverse the 3D space and move from one level to the next seamlessly while data (objects) load in the background as the player progresses.

**Video Memory Restrictions for Level Designers:**

Back Buffer (640x480 32bit game resolution)                                       1,228,800B

                       (1024x768 32bit)                                                                     3,145,728B

Letter textures for the text overlay                                                         1,048,576B

**Level Textures:**

Level Texture 1: 6 - 512x512 32bit (3 color, 3 bump)          6,291,456B

Level Texture 2: 4 – 512x512 32bit (2 color, 2 bump)         4,194,304B

Level Texture 3: 2 – 512x512 32bit (1 color, 1 bump)         2,097,152B

Level Texture 4: 2 – 512x512 32bit (1 color, 1 bump)         2,097,152B

Level Texture 5: 2 – 256x256 32bit (1 color, 1 bump)         524,288B

Level Texture 6: 4 – 256x256 32bit (2 color, 2 bump)         1,048,756B

Level Texture 7: 2 – 256x256 32bit (1 color, 1 bump)         524,288B

Total:     16,777,396B

**Particle Textures:**

Particle Texture 1: 1 – 256x256 32bit                                        262,144B

Particle Texture 2: 1 – 256x256 32bit                                        262,144B

Particle Texture 3: 1 – 256x256 32bit                                        262,144B

Particle Texture 4: 1 – 256x256 32bit                                        262,144B

Total:     1,048,756B

**Character Textures:**

Character Texture 1: 1 – 512x512 32bit                                   1,048,756B

Character Texture 2: 1 – 256x256 32bit                                   262,144B

Character Texture 3: 1 – 512x512 32bit                                   1,048,756B

Character Texture 4: 1 – 256x256 32bit                                   262,144B

Character Texture 5: 1 – 512x512 32bit                                   1,048,756B

Total:     3,670,556B

**Miscellaneous Object Textures:**

Miscellaneous Object 1: 2 – 256x256 32bit (1 color, 1 bump)         524,288B

Miscellaneous Object 2: 2 – 256x256 32bit (1 color, 1 bump)         524,288B

Miscellaneous Object 3: 2 – 256x256 32bit (1 color, 1 bump)         524,288B

Miscellaneous Object 4: 2 – 128x128 32bit (1 color, 1 bump)         131,072B

Total:     1,703,936B

**Grand Totals:**

-- 640x480 32bit resolution:              27,935,620 Bytes

-- 1024x768 32bit resolution:            33,686,404 Bytes

(Memory Restriction figures based off of/come from the TDD for [*Narbacular Drop*](http://jeepbarnett.com/nuclearmonkeysoftware/documents/narbacular_drop_technical_design_document.pdf)'s "Video Memory" section - Pg. 15)

# Hardware Considerations

Certain hardware is required to run this game at minimum and in order to achieve peak performance. Using the Unreal Development Kit engine, a computer must be able to support the minimal specifications or else the game will not run. In order to achieve best results a computer should be able to run the Specifications listed within performance goals. Program processing is left to a multiprocessor which should also meet at least the minimum specification.

## Minimal Specifications

Windows XP SP3 (32-bit only), Windows Vista, or Windows 7

2.0+ GHz processor

2 GB system RAM

SM3-compatible video card

3 GB free hard drive space

## Multiprocessors

2.0+ GHz processor (Minimal)

2.0+ GHz multi-core processor (Recommended)

## Performance goals

60 Frames Per Second (FPS)

Peak Performance Specifications

Windows 7 64-bit

2.0+ GHz multi-core processor

8 GB System RAM

NVIDIA 8000 series or higher graphics card

Plenty of HDD space

# Multiplayer and Networking

Princess with a Rocket Launcher is a singleplayer FPS platforming game and as such, will not include any multiplayer and networking features.

## Client-Server vs Peer-to-peer

## Communication protocols

## Number of players supported

## Matchmaking process

## "Join-in" and "Drop-out" procedures

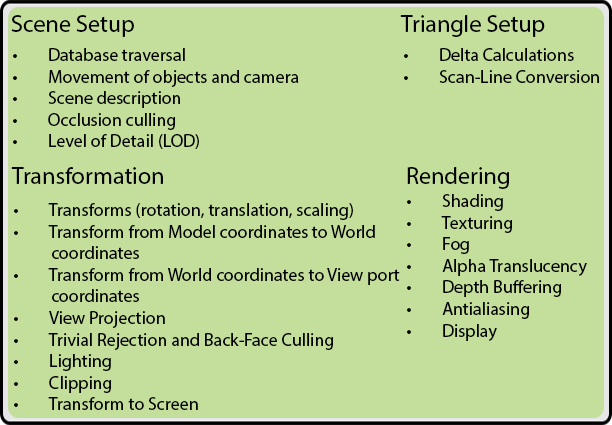
## Latency and lost packets

## Managing cheating / hacking

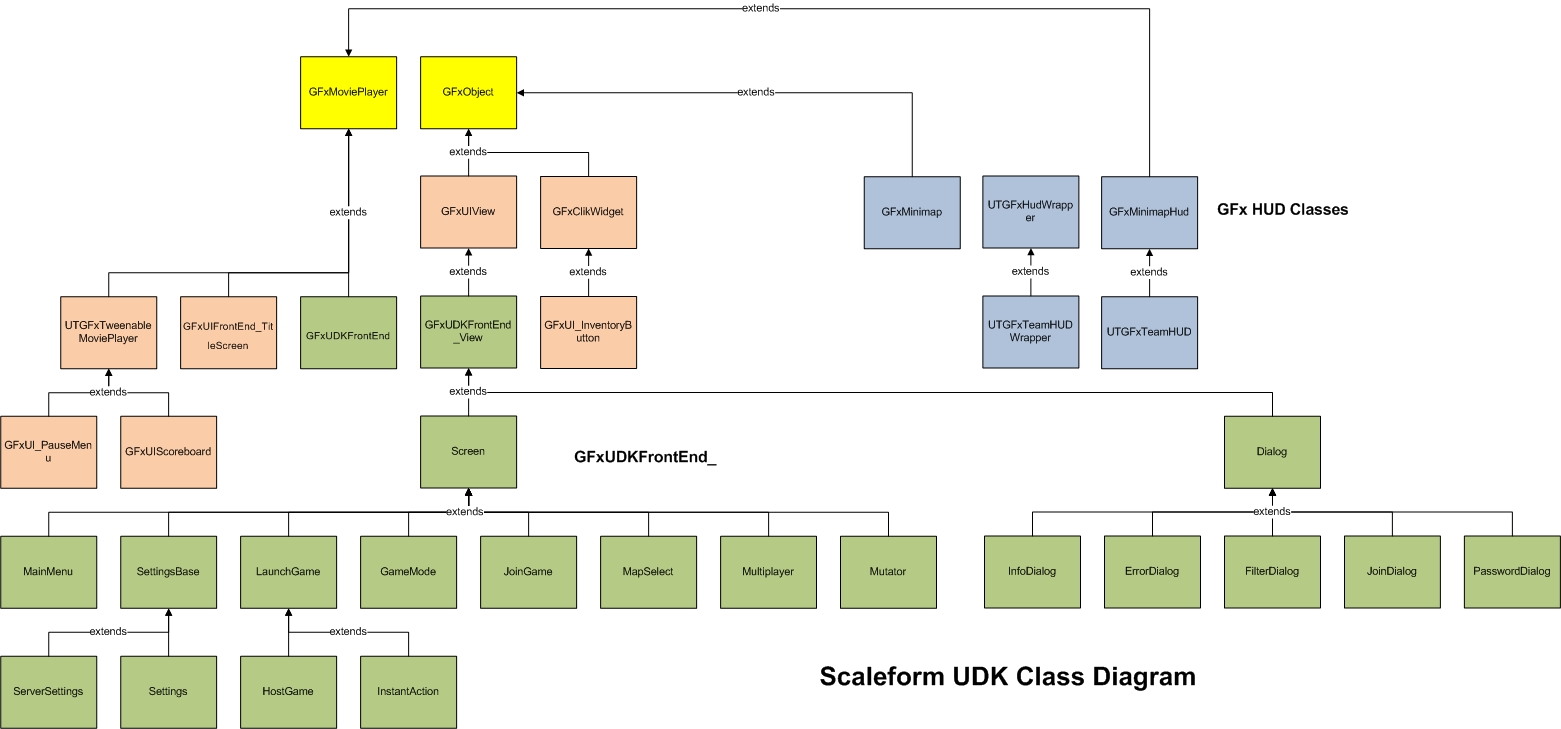
## Clans, observers, tournaments

# Graphics

## 3D Pipeline



## User Interface (UI)



(<https://udn.epicgames.com/Three/rsrc/Three/ScaleformBuildingUDKUIs/scaleformudkflowchart.jpg>)

## Art tools

Level Editor

* Unreal Engine 4 (UE4)

Particle Effects Editor

* ParticleSystems (included in UE4)
* Cascade (included in UE4)

Geometry and/or Scene Converter

* Handled by the modeling software and UE4 (see “Tools” for more info)

Animation and Object Viewer

* Handled by 3D Studio Max/Maya and UE4

# Sound

## Architecture

The game will be comprised of a set of different musical themes that are tied to certain characters or levels or battle scenes. Each of these will be called by the game when they are encountered in the game per the script. Sound effects are going to be implemented for the different sounds made by each character and recorded voice found in movie scenes.

## Encoding

Sound files will be encoded for ones that are made in-house and for those accessed externally into MP4 format for use in the game program. Voice recordings may involve .wav files due to better quality and larger size. The sounds made in-house will have to be done and recorded in the studio.

## Quality / Noise reduction

In order to ensure quality of music and sound effects both must be recorded in a sound-proof room in a recording studio. Also, voice recordings will have to be done there as well to avoid cross-interference from outside sounds to the mic. Audio files will be recorded and then encoded into

## Sample Rate

[Calculating](http://en.wikipedia.org/wiki/Digital_signal_processing) the values of the new samples directly from the old samples. The latter approach is generally preferred since it introduces less noise and distortion;[[3]](http://en.wikipedia.org/wiki/Sample_rate_conversion#cite_note-antoniou-3) two possible implementation methods are as follows:

1. If the ratio of the two sample-rates is (or can be approximated by)[[nb 1]](http://en.wikipedia.org/wiki/Sample_rate_conversion#cite_note-4) a fixed, rational number L/M: generate an intermediate signal by inserting L–1 0s between each of the original samples. Low-pass filter this signal at half of the lower of the two rates. Select every Mth sample from the filtered output, to obtain the result.[[4]](http://en.wikipedia.org/wiki/Sample_rate_conversion#cite_note-raj-5)
2. Treat the samples as geometric points and create any needed new points by interpolation. Choosing an interpolation method is a trade-off between implementation complexity and conversion quality (according to application requirements). Commonly used are: [ZOH](http://en.wikipedia.org/wiki/ZOH) (for film/video frames), [cubic](http://en.wikipedia.org/wiki/Bicubic_interpolation) (for image processing) and [windowed](http://en.wikipedia.org/wiki/Window_function) [sinc function](http://en.wikipedia.org/wiki/Sinc_function) (for audio).

The two methods are mathematically identical: picking an interpolation function in the second scheme is equivalent to picking the impulse response of the filter in the first scheme. Linear interpolation is equivalent to a triangular impulse response; windowed sinc approximates a [brick-wall filter](http://en.wikipedia.org/wiki/Brick-wall_filter) (it approaches the desirable "brick wall" filter as the number of points increase). The length of the impulse response of the filter in method 1 corresponds to the number of points used in interpolation in method 2.

## Stereo vs. Mono

Stereo sound will be used for the game due to its nature as the most commonly used audio system in the industry. Stereo devices are used at home so the audio will be presented in stereo.

# Localization

## No Localization will be performed for this Game Project.

# Technical Design Review (TDR)

Upon completion of the TDD, the document will be reviewed by the development team. The goal is to make sure the TDD is completed to the upmost quality and ensures the project will be completed on time and on budget. As such, the team will be given two weeks to reach a consensus as to whether or not the TDD meets the team’s rigorous standards of quality.

# Prototype

### Paper Prototype

A paper prototype will be developed with a map or board for each level of the game. They will primarily be used to improve upon the layout of the levels, the enemies, and any hidden game elements. It will utilize a turn-based system for movements and actions. This will give a good idea of how well the path is laid out, whether or not hidden items are too out of the way, and whether or not the enemies are in good positions and quantities. The paper prototype will not however give a good sense of the shooting aspects of the game.

### Digital Prototype

While the paper prototype focuses on level design, the digital prototype will focus on the shooting mechanics. A simple whiteboxed level will be developed to play through with the mechanics. The level will contain several of the types of enemies that will be encountered as well as any powerups and new weapons the player will find.