Project Title: 3D Gaming Technologies

Game Title: Akari

Student Name: Russell Kentish

Student Number: K0933605

Degree: BSc (Hons) Games Technology

Kingston University

Credits

Russell Kentish

Game Designer

Programmer

Animation Director

Level Designer

Writer

Ivy Tsai

Concept Artist

Character Artist

Environment Artist

Character Rigger

Charles-Jean Boucher

Motion Capture Actor

Calle Bennett

Composer

Sound Effects

Charles Evanson

Composer

Sound Effects

Darrel Greenhill

Producer

Project Supervisor

Andreas Hoppe

Second Marker

This report documents the development of a PC game called Akari that was submitted as Russell Kentish's Final Year Project for his BSc Games Technology course at Kingston University. This paper also analyses the use of 3D graphics video games, the various methods of video game development, and the rise of independent video games.

Table of Contents

Introduction

Project Summary

This project was set as a final year project for the BSc Games Technology students at Kingston University. The aim for this project was to create a 3D game for the PC using a games engine that is based on OpenGL, as well as to improve our game development and C++ programming skills. The game uses Irrlicht[i] as its graphics engine, as well as other libraries and APIs that include, Bullet[ii], XInput[iii] and IrrKlang[iv].

The game is a top down shooter/adventure game called Akari, which is set in a fictional universe where mankind has unlocked the mysteries of quantum mechanics. The story follows the self-taught scientist Akari, who wants to continue her late parents research into developing this new technology for the good of mankind.

This project proves that it is possible to create a highly polished, deep video game, at a relatively low budget and without the need for funding from a third party, the very definition of an independent video game.

Background

Over the past ten years, independent games, or indie games for short, have become more prevalent in the video game industry. One could argue that most developers during the late 1980s could be considered indie, and judging from their development methods back then one would be correct. Currently, an indie game is primarily defined by the fact that it was developed by a smaller development team who fund themselves, in contrast to bigger triple A studios and their reliance on their publishers for funding. So the term “indie game” was coined only recently to help differentiate between the big and small studios[v].

As our technology improved during the late 1990s, so did the quality of our games, thus raising the consumer’s expectations. Due to the increase of costs that came with the raised tier of technology, developers became more reliant on publishers for funding rather than funding their projects out of their own pockets. During this period, indie games became less prevalent, gaming systems were just too expensive to develop for and many indie developers just could not afford to develop their games[vi]. This trend continued until the early-mid 2000s where a growing new phenomenon would change our software distribution methods forever, Digital Distribution.

Digital Distribution is the future of software distribution. It reduces the risk of piracy, requires no packaging and physical distribution costs, and it available on demand for the consumer, increasing the likelihood of a sale. Developers are now able to create a game, and upload it onto platforms like Steam, The App Store, Xbox Live Arcade and the PlayStation Store, to name but a few, for the convenience of the consumer and without having to pay thousands in packaging and distribution costs. Some distribution platforms contributed to the increase of indie games even more. The XNA Game Studio, released alongside the Xbox 360, is a great way for new developers to start out as they provide all the vital libraries needed to develop a commercial game and it is very accessible.

The increase of mobile technology also played a vital role and the iPhone is a great example to show where mobile gaming is going[vii]. Games like Angry Birds prove that games do not need to be big budget to be a critical success[viii], and Rovio Mobile have set an example to many new graduate game designers looking to become as successful as they have.

Using the game studio, Colossal Games[ix] as an example of the accessibility of games design. All they required was a Unity3D License and registration with Apple to become an iPhone developer, all, which has cost them under £300. Knowledge of the implementation process can be gained from game development courses at colleges and universities as well as other sources, Code Hero[x] for example.

Many games development courses have been appearing recently, the course[xi] at South Thames College has only been around for about five years, and the quality of these courses has been getting better over time. It will not be too long until graduate games designers will be flooding recruitment agencies. Modding is another great way to break into the industry, ten years ago, roughly 40% of all games designers did not have a degree, and instead, they were "modders" who had proven themselves through a successful mod/map. An example would be Dave Johnston, creator of the exceptional Counter Strike level de_dust, who now works as Lead Level Designer at Splash Damage[xii].

The rise of indie games had a number of contributing factors:

- Technological advancement;
- Ease of software distribution and;
- Accessibility of games development.

Indie games require innovation to make up for their small budgets thus giving them the potential to keep creativity in the gaming industry afloat.

Solution

Following the introduction, this report is comprised of five chapters:

Analysis – Planning is necessary before undergoing any sort of important or complicated task. This chapter documents the research that went into this game. A good understanding of popular game genres, platforms, engines and gameplay will help a designer decide the best course of action before starting the development process.

Design – Creating a game that is well designed is quite a challenging feat, but only the most enjoyable games will become successful. This chapter documents the games design process, it details the decisions that were made when designing with gameplay, aesthetics, music and narrative.

Implementation – Envisioning a well-designed video game is only half the battle. Arguably the most challenging part of video game development is the implementation process. This chapter documents how the game was programmed and how the different elements were woven together to produce an interactive application.

Testing and Functionality – One of the most important stages of games development is the testing phase. No game is perfect, and testing a product is the only way to ensure that it is performing at optimal capacity. This chapter documents the testing phase of development.

Conclusion – The final chapter to conclude this project, states any final thoughts and what was learned during the development process.

Analysis

Video Game Design

Video Game design is an art form where Game Designers are tasked with creating enjoyable interactive experiences. For the best results, a designer is not only required to have a deep knowledge of video games and video games development, but they must also be quite well informed in other types of media as well as have a lot of life experience. This is to ensure that their creations have a certain depth and elegance that cannot be matched against a narrow-minded individual's design.

While there are many ways to break down the individual elements of an interactive experience, Jesse Schell, who wrote 'The Art of Game Design: A Book of Lenses'[xiii] describes four of arguably the most important ones:

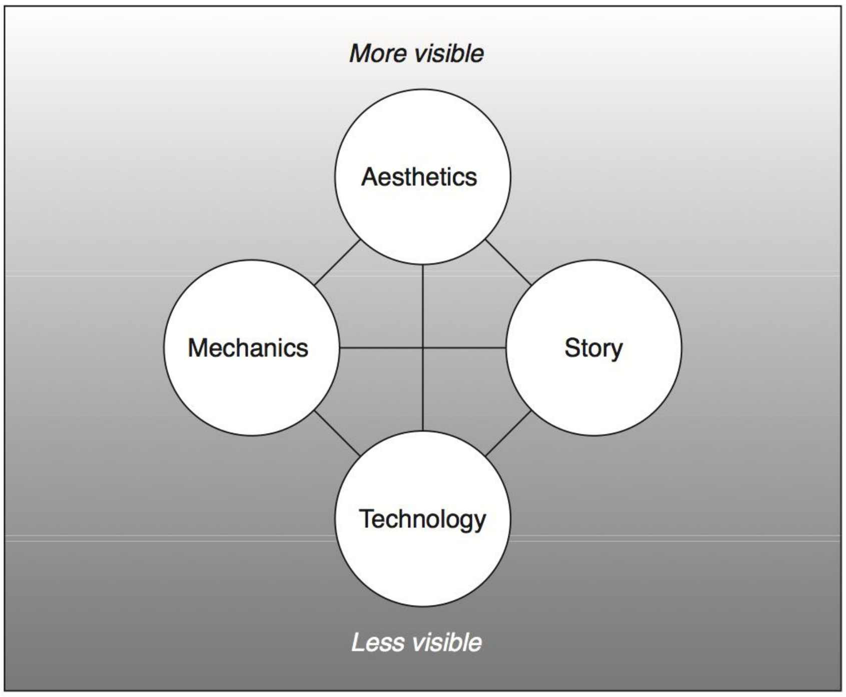
1. Mechanics: "*These are the procedures and rules of your game. Mechanics describe the goal of your game, how players can and cannot try to achieve it, and what happens when they try. If you compare games to more linear entertainment experiences (books, movies, etc.), you will note that while linear experiences involve technology, story, and aesthetics, they do not involve mechanics, for it is mechanics that make a game a game. When you choose a set of mechanics as crucial to your gameplay, you will need to choose technology that can support them, aesthetics that emphasize them clearly to players, and a story that allows your (sometimes strange) game mechanics to make sense to the players.*"[xiv]

2. Story: "*This is the sequence of events that unfolds in your game. It may be linear and pre-scripted, or it may be branching and emergent. When you have a story you*

want to tell through your game, you have to choose mechanics that will both strengthen that story and let that story emerge. Like any storyteller, you will want to choose aesthetics that help reinforce the ideas of your story, and technology that is best suited to the particular story that will come out of your game." [xv]

3. Aesthetics: *"This is how your game looks, sounds, smells, tastes, and feels. Aesthetics are an incredibly important aspect of game design since they have the most direct relationship to a player's experience. When you have a certain look, or tone, that you want players to experience and become immersed in, you will need to choose a technology that will not only allow the aesthetics to come through, but amplify and reinforce them. You will want to choose mechanics that make players feel like they are in the world that the aesthetics have defined, and you will want a story with a set of events that let your aesthetics emerge at the right pace and have the most impact."* [xvi]

4. Technology: *"We are not exclusively referring to "high technology" here, but to any materials and interactions that make your game possible such as paper and pencil, plastic chits, or high-powered lasers. The technology you choose for your game enables it to do certain things and prohibits it from doing other things. The technology is essentially the medium in which the aesthetics take place, in which the mechanics will occur, and through which the story will be told."* [xvii]



[xviii]

The best designers are able to build a relationship with their players. Understanding the audience's thoughts and feelings and allowing them to immerse themselves into the game world will get them emotionally invested. Whether it is by having them relate to a main character, impressing them with the way the game looks and sounds, or engrossing them with engaging gameplay, the player will experience the imagination of the designer and hopefully arouse some aesthetic or moral feelings within them.

But there is more to game design than coming up with enjoyable game concepts, the designer is also tasked with turning a concept into reality[xix]. A good understanding of the development process will give the designer an idea of the tasks that will need to be completed and what kind people that will needed to complete them. A typical modern 3D video game requires programmers, artists and sound experts. Usually in bigger development teams, each of these disciplines will be split into specialized paradigms where the specialist will focus only on one particular element, so there may be job titles such as Gameplay Programmer or Texture Artist.

Once a development team has been established and the game concept has been envisioned, the next stage is to analyze available game engines or graphics libraries to determine which one will be appropriate, and then to establish a familiarity with that engine or graphics library. An analysis into supporting engines and APIs, such as a physics engine, is also recommended. Information, as well as tutorials, on recommended engines and API can be found online, via Google, or on various development forums.

Communicating well with associates is extremely important because people will have radically different skill sets and backgrounds. Ideas will need to be communicated to them as well as possible, as they will be the ones who will be translating that idea into an element of the game.

The first hurdle of games development is its cost. Fortunately, there are a number of useful tools available to developers for free to assist in the development of video games. Examples would include using Blender[xx] to create 3D geometry, and Gimp[xxi] to create textures and concept art, Ogre3D[xxii] and Irrlicht[xxiii] are two examples of graphics engines and Bullet[xxiv] is an open source physics library, all are available at no extra charge.

Game Engines

Video game engines assist in the development of video games by providing the tools necessary to create various game elements and allow them to run in real-time.

Irrlicht

The Irrlicht[xxv] Engine is an open source 3D graphics engine, which is written in C++. It is based on OpenGL so it is compatible with both OSX and Windows operating systems. A 3D graphics engine is used to draw 3D graphics onto the screen in real time. Irrlicht is a relatively high level engine, meaning that it has a lot of advanced features available to the user rather than the user having to program these features themselves, for example, this engine has real time lighting, it is capable of importing multiple 3d geometry formats and has particle emitter nodes. While it is possible to render 3D graphics using this engine, it lacks other features that come standard with other more sophisticated games engines, such as a built-in physics and sound engine. [xxvi]

CopperCube

CopperCube[xxvii] is an application that allows a developer to edit 3D scenes in real time to be exported and implemented into an Irrlicht project. It features real-time lighting effects, scripting support, character animation and more. It can be downloaded for free and supports all Irrlicht related exports, but a license is required for any non-Irrlicht related development.

Ogre3D

Ogre[xxviii] is an open source 3D graphics engine, just like Irrlicht, and has a lot of the same features. It does not include a real-time scene editor like CopperCube and it is a lot more limiting in terms of mesh file formats.

Gameplay Analysis

The gameplay of a video game is what defines its genre. When coming up with how a game will play, analyzing the market and doing research into the most popular games out there, by using the VGChartz[xxix] website for example, should be considered.

The key great gameplay is; accessibility, which allows the game to easily played by a broad range of people of different skill levels, and; depth, which means that there could be multiple ways in which this single, easy-to-learn, gameplay concept can be exploited. In short, the gameplay should be easy to learn but hard to master.

Geometry Wars: Retro Evolved[xxx] is an Xbox Live Arcade[xxxi] indie game and is one of the games that Akari is based on. It is a 2D shooter game where the player has to score as high as possible while trying to stay alive. The enemies will kill the player if they touch them and more enemies will spawn the longer the player is alive. It was a very enjoyable and addictive game to play because its gameplay was very accessible as well as in-depth.

One of the key differences between Akari and Geometry Wars is that Akari has a narrative to it. An arcade style game like Geometry Wars is not suitable for this kind of storytelling. A third-person gameplay perspective encourages exploration and platforming and is a far better way t portray Akari's story. The third-person perspective is one of the best ways to navigate through an environment as gives the player a sense of the position of the main character, allowing them to be more precise with their movement and the moveable camera position the player to easily judge distances between objects, making platforming much easier. The Legend of Zelda: Ocarina of Time[xxxii] was one of the first games to perfect the third-person gameplay perspective and it is widely regarded to be one of the best games ever made.

The main issue of having multiple gameplay modes is that the player may get overwhelmed having to learn two different button at once, though Sonic Colors and Sonic Generations[xxxiii] manages to achieve a 2D-3D perspective gameplay shift very well.

3D Graphics

The current technological generation has played host to some of the best graphics gaming has ever seen. Hardware stills plays a vital role in the performance of our games, and due to the popularity of console titles over PC titles[xxxiv], most developers have decided to develop their games for consoles rather than PC. This is a shame because the current generation of consoles is long due for a hardware upgrade. It is common knowledge that the graphics in games like The Elder Scrolls: Skyrim, could have looked a lot better if they were developed specifically for high end gaming PC, rather than for our current-gen consoles.

Computer graphics has been long defined as the quest to achieve photorealism[xxxv], and thanks to Moore's Law, photorealism will soon be possible, but in the meantime, game developers have equipped themselves with various techniques that workaround hardware constraints to give the illusion of photorealism in games. While photorealism is great for immersing a player into the game world, it should be made clear that there is more to images than realism alone, particularly for games development [xxxvi]. Abstract aesthetics engages the player's senses and imagination, which is something many triple-A developers today overlook.

A good way for developers to manage their resources is to focus on important objects in a scene, the objects that the player will take the most notice of. So the main character, for example, may have three times the amount of polygons, than the typical enemy in the game, because the main playable character will be on screen for most of the game so the player will be more likely to notice any flaws that will break their immersion, where as enemies will be seen from further away, and designing them to use less polygons will allow the developer to render more of them onto the screen.

2.5D is the method of using 2D elements and manipulating them in a in a way that makes those 2D elements, appear to be 3D [xxxvii]. This method of rendering was used heavily in the early days of video games, due to the fact that the hardware was not powerful enough to calculate 3D environments.

Thankfully, our current hardware generation is capable of rendering full 3D environments with very little effort. Akari will benefit from this greatly because a third-person camera perspective will not be possible with a fixed isometric camera.

Maya

Maya[xxxviii] is 3D computer graphics software and can be used to animate, UV map, texture and render 3D geometry that is used in many creative industries. It was used as the primary 3D graphics software for this project. Primitive objects and polygons can be created and edited using the tools provided to create assets for movies and video games. Irrlicht is compatible with several static mesh file formats as well as a few animated mesh file types [xxxix]. To import animated models into an Irrlicht project, the recommended format is DirectX (.x) [xl]. Markis Bergqvist, from the Irrlicht development forums, wrote a very useful script [xli] for Maya that exports models as an .x to be used specifically for Irrlicht. There are other exporters [xlii] that export animated models into a compatible file formats but the exportation process of animated meshes may have some unexpected results.

Physics Engines

Real time dynamics in video games give them a sense of realism that is important for keeping players immersed. The best video games incorporate physics into the actual gameplay, giving the game a layer of unpredictability and variety. Most game engines have physics included in them, such as the Unity3D engine, which has PhysX physics engine built in [xliii]. Akari is a 3D video game and the inclusion of realistic behaving physics adds a layer of immersion for the player. The importance of realistic looking physics is imperative as the human mind is susceptible to picking up on ‘fakeness’ [xliv], that can leave the player feeling uneasy, thus breaking their immersion.

Bullet
Bullet Physics[xlv] is a professional open source collision detection, rigid body and soft body dynamics library. It can easily be integrated into Irrlicht, especially with the IrrBullet Wrapper[xlvi]. The physics in Akari were based off the guide written by Punong Bisyonaryo[xlvii]. A guide[xlviii] on including Bullet into projects can be found on the Bullet Physics wiki[xlix].

DMM
Digital Molecular Matter (DMM) [l] is a physics engine that creates objects with realistic material properties, such as shattering wood, bending sheet metal and jelly. While this is a very impressive engine, each DMM object requires a lot of processing power and it is quite expensive to obtain a license. If the gameplay directly requires such a sophisticated physics engine, it would have been strongly considered, but it does not.

Music and Sound Effects

Sound effects and music are just another way to add a more layers of depth onto video games, so choosing a respectable sound engine is important. Charles Evanson and Calle Bennet created the main theme for Akari.

IrrKlang
irrKlang[li] is an open source, high level, 2D and 3D, cross platform sound engine. It is extremely simple to use and can be incorporated into projects the same way as irrlicht. It can be used to change the volume of sounds and it can also give sounds a particular sound effect, which include echoes and a Doppler effect.

It is easy to use and can be implemented pretty flexibly into computer applications.

Legal, Social and Ethical Issues

Video games have become apart of young adult life for the past 30 years. Like all life experiences, they can affect the way we think and behave. The Video Games: Issues and Problems journal states, "A meta-analytic review of the video game research literature reveals that violent video games increase aggressive behavior, physiological arousal and aggression-related thoughts and feelings in children and young adults. Playing video games also decreases pro-social behavior." [lii] In contrast, video games can also be an effective and entertaining way of educating players. Problem solving, strategy assessment and media and tools organization are all skills that can be developed by playing mentally stimulating video games.

Violent Video Games

Like music and film, video games have been the focus of many media outlets when an individual shows signs of aggression. This subject hits quite close to home for me, as one of my high school classmates was recently found guilty for grievous bodily harm in a video game related incident [liii]. A recent content analysis of video games shows that as many as 89% of games contain some violent content [liv].

Solutions

Like drugs and alcohol, video games can be used as a way of escapism. People who suffer from depression or who find themselves lacking something in their lives may become addicted to video games. Video games are a form of entertainment, and should be treated as such, over-playing can become detrimental to a persons lifestyle and is not a way to live an enjoyable life in the long run. This is a problem with variables degrees of severity, and the people who can control these problems should be held accountable, these people include:

The government – They can set Age Ratings for video games so that violent, sexual or disturbing content does find itself in the hands of minors and they should prevent the distribution of video games with illegal content.

The parents – As responsible parents, they should ensure that inappropriate video games are not being played by their children. Moderation is key and parents should ensure that their children are not indulging themselves in their games for too long.

The video game developer – By all means, video game designers should have a right to artist expression, but they should monitor the content of their game and be mindful of these ethical and legal issues.

Functional Requirements

Following are the main functional requirements of the game organised by their tangible object classes. Priority is tagged as HIGH, MEDIUM or LOW as appropriate.

Main Menu

Priority: HIGH

Description: The player will interact with a main menu to choose if to start a new game, to continue from a saved checkpoint or change in-game preferences.

Stimulus/Response Sequences: Accepts player input at the start of the game.

Functional Requirements:

1. It must be easy to understand and navigate.
2. It must allow the player to start a new game or to continue from a saved state.
3. It must allow the player to edit system preferences on how the game will look aesthetically.

Main Character

Priority: HIGH

Description: The player will be interacting with the main character for the majority of the game. The game's genre is defined by how the player controls Akari.

Stimulus/Response Sequences: Majority of user input, all in-game elements.

Functional Requirements:

1. Can be translated using an Xbox 360 controller
2. Can be rotated with an Xbox 360 controller
3. Has the ability to shoot orbs
4. Has the ability to jump
5. Is animated

6. Can interact with other in-game elements.
7. Its geometry must not clip with other in-game objects.

Isometric Camera Perspective and Gameplay

Priority: HIGH

Description: This gameplay mode is used for the game's combat sequences, it will keep the main character in view while in combat.

Stimulus/Response Sequences: Majority of user input, all in-game combat sequences.

Functional Requirements:

1. It must keep the player in view at all times during combat.
2. It must show where the enemies are on screen clearly.
3. Objects must not obstruct its view.

Third-person Camera Perspective and Gameplay

Priority: HIGH

Description: This gameplay mode is used for the exploration and platforming in-game sequences.

Stimulus/Response Sequences: Majority of user input, all in-game exploration sequences.

Functional Requirements:

1. It must rotate smoothly and intuitively.
2. It must not be a burden to the player.
3. It must follow the player around.
4. It must keep the player in view at all times.

Grunt Enemy Type

Priority: MEDIUM

Description: As the main melee adversary to our main character, they provide support gameplay and create challenge for the game.

Stimulus/Response Sequences: They provide a challenge the player must overcome to progress, present in combat sequences.

Functional Requirements:

1. They must be aware of the players position.
2. They must be well balanced and not cause cheap deaths.
3. They must be animated.
4. The must emulate separation behaviour to avoid geometry clipping.
5. They must be able to be defeated by the player.
6. They must respawn once all other enemies have been defeated.
7. They must follow and attack the player.

Runner Enemy Type

Priority: HIGH

Description: As the main melee adversary to our main character, they provide support gameplay and create challenge for the game. They are also faster than the Grunt class.

Stimulus/Response Sequences: They provide a challenge the player must overcome to progress, present in combat sequences.

Functional Requirements:

1. They must be aware of the players position.
2. They must be well balanced and not cause cheap deaths.
3. They must be animated.
4. The must emulate separation behaviour to avoid geometry clipping.
5. They must be able to be defeated by the player.
6. They must respawn once all other enemies have been defeated.
7. They must follow and attack the player.

Caster Enemy Type

Priority: MEDIUM

Description: As the only ranged adversary to our main character, they provide support gameplay and create challenge for the game.

Stimulus/Response Sequences: They provide a challenge the player must overcome to progress, present in combat sequences.

Functional Requirements:

1. They must be aware of the players position.
2. They must be well balanced and not cause cheap deaths.
3. They must be animated.
4. The must emulate separation behaviour to avoid geometry clipping.
5. They must be able to be defeated by the player.
6. They must respawn once all other enemies have been defeated.
7. They must follow the player.
8. They must be able to fire orbs at the player.

Physics

Priority: LOW

Description: Physics objects will populate the scene, providing a sense of realism.

Stimulus/Response Sequences: They will physically react when the player shoots them.

Functional Requirements:

1. They must react when hit.
2. They must not fall through the floor of the stage.

Environment

Priority: HIGH

Description: An environment will provide some context for the player; it will keep the main character within specified boundaries.

Stimulus/Response Sequences: The player will stop moving when they wonder too far from the scene.

Functional Requirements:

1. It must keep the player bound to the environment.
2. Enemies must not venture out of bounds.
3. Objects must not obstruct the games camera.

HUD

Priority: LOW

Description: A Heads-up-display, may be used to keep the player up to date with in-game statistics.

Stimulus/Response Sequences: The game shows the player how far they have progressed through a level.

Functional Requirements:

1. The User interface elements must not obstruct the player's view of the game.
2. It should be well organized and aesthetically pleasing.

Non-Functional Requirements

This section identifies any non-functional requirements of the game required for the game to operate as proposed and to be successful in the form of a game.

1. The game is aesthetically pleasing
2. The user interface is consistent throughout its design
3. The game is executable on any PC using Windows XP or any newer version of Windows
4. The game is executable on any Mac using OSX.
5. Runs smoothly on any 1.5GHz+ processor and 2GB+ of RAM PC
6. Frame rate is consistently above 30 FPS on a system of this specification
7. Loads and is playable from an executable in less than 10 seconds
8. The game does not crash at any point and is bug free in that respect
9. The gameplay is responsive and intuitive.
10. The user controls are accessible
11. The user controls are logically laid out

Design

This segment will describe the preproduction phase we went through, as well as the design process for each game element.

Visual Studio 2010

The compilation software used to write the code for Akari is Visual Studio 2010[lv]. With this software, a developer is able to link their chosen libraries together and manipulate their classes and methods in the appropriate programming language to develop a meaningful video game. (Appendix 1)

Gameplay Design

The game has two modes of gameplay: the first plays like a top down shooter (similar to Geometry Wars[lvi]). This gameplay mode is used primarily for the combat portions of the game; the camera is isometric to the main character; and the player can move the main character freely in the X and Z-axis. Enemies will spawn randomly around the stage and will seek and pursue the protagonist. It is the player's role to steer Akari away from the enemies, and fire her weapon to destroy then before they destroy her, it only takes a few hits to be defeated.

The second gameplay mode plays like a third person action/adventure game. One of the unique selling points of this game will be the switching between these two gameplay modes. This will keep the game from becoming repetitive.

XInput[lvii] is an API that reads the input commands of an Xbox 360 controller. The gameplay of Akari is the kind that would benefit greatly from the use of an input device with duel analogue sticks, and as the Xbox 360 controller is the most popular gamepad for Windows; the decision to include it was a simple one. The API comes preinstalled with the latest Windows Operating systems and is easy to link with projects. A Windows Microsoft Xbox 360™ Wireless Gaming Receiver[lviii] is necessary to play the game with a wireless Xbox 360 Controller, or a wired controller can be plugged in via USB.

Story

The game takes place in the 23rd century, on Earth, in a fictional universe where mankind has unlocked the mysteries of quantum mechanics, the story follows the self-taught scientist Akari, who wants to continue her late parents research into developing this technology to help society.

Four years prior to the start of the game, Akari's parents were scientists who helped in the development of technology that allows humans to manipulate matter and the very fabric of space and time. While they sought to use this technology for good of humanity, other departments of the same organization gave into greed and used this technology to make weapons. After various political disputes concerning these new weapons, war breaks out. As her parents are involved in the research of this technology, they are assassinated while trying to protect Akari and her younger sister. Akari is the only survivor and is enrolled into an orphanage. At the age of 16, she signs up for the military, to get her revenge on <insert country name here> and is trained with the technology her parents helped develop, she is proud of this fact and is an exceptional cadet. Guns are now obsolete and soldiers now fight with extra dimensional matter generators. After her initial training, she uses the money she has earned to buy back her parents house. She is then shipped abroad to occupy the enemy country. Days of killing soldiers take place and after a particularly long battle, she is given an order to execute some civilians. She is about to go through with it, but she witnesses a mother protecting her child and is reminded of how her parents died trying to protect her and her sister. She refuses, and the civilians are executed anyway. She is then disciplined and is discharged from the military. She returns to her parent's house, she is now ashamed with her parents for creating such destructive technology, after witnessing the slaughter of innocent people. In her frustration with her powerlessness, she starts to trash her parent's home. In the process, she stumbles across a hidden room in the house. She discovers her parents advanced research into matter manipulators and that her parents were researching how to use it for medical purposes. She now realizes that her parents were good people and vowed to continue their research in secret for the good of humanity while the war continues.

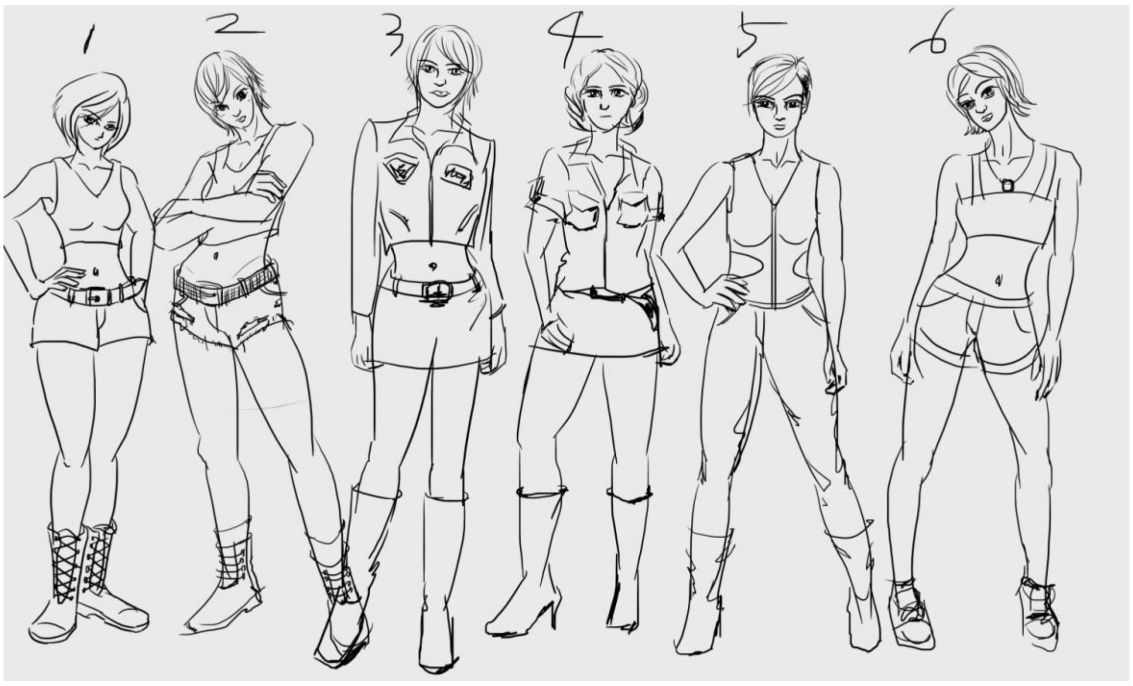
We went through a few game concepts. The first one we came up with was set in a fantasy universe, that followed two spell casting brothers who sought to find the truth behind their father's murder, but we decided to change it to something that is more unique. Cut-scenes will be played in-between acts of the gameplay and voice-overs will be played, in-game, to help narrate the games story to the player.

Main Character

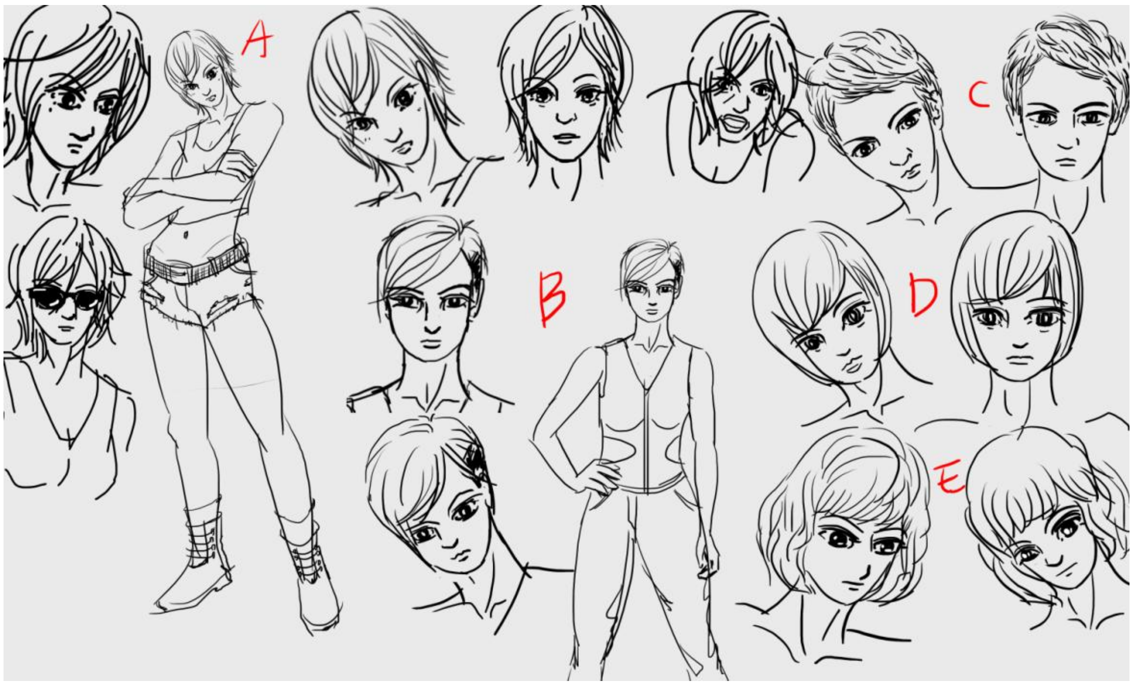
The protagonist of any story should be relatable to the audience in one way or another. Likeable main characters allow the audience to have an emotional connection with a video game, thus providing a much deeper experience.

Developing a solid character story and personality will give the artist designing them a much clearer picture of what the character should look like. Ivy Tsai was lead concept artist, so she was responsible for designing the look of the main character. The lead designer was tasked with overseeing the design process, so the final concept comes out the way it was envisioned.

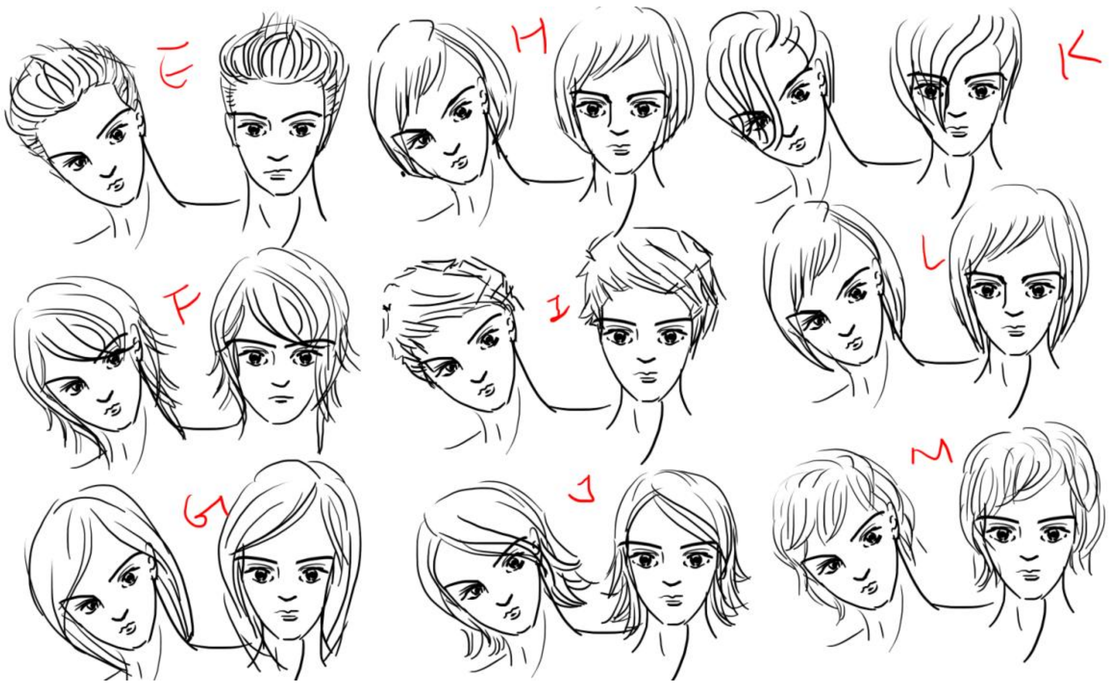
The lead characters for our first game concept.



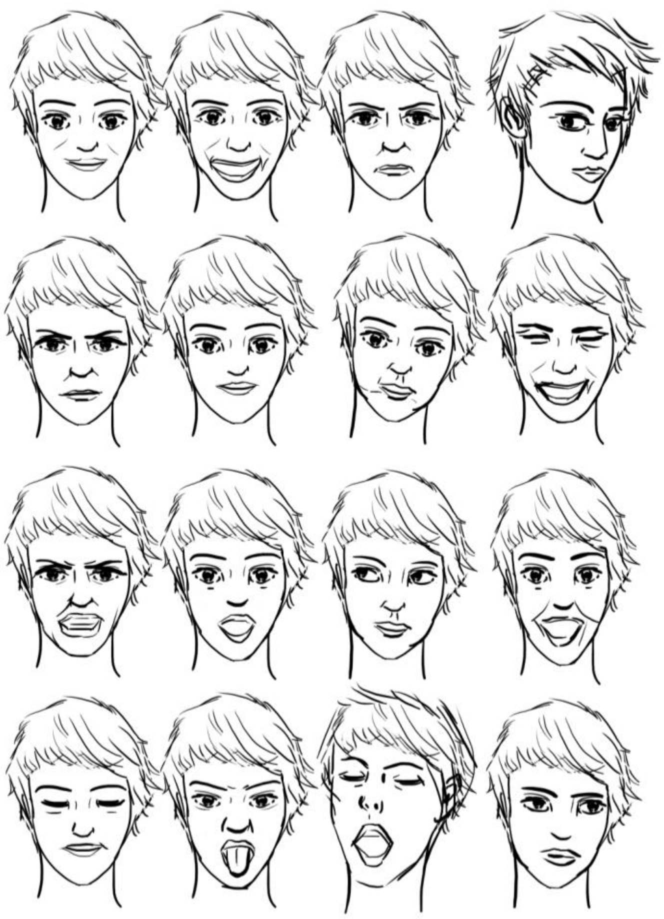
This was the first piece of concept art that Ivy produced.



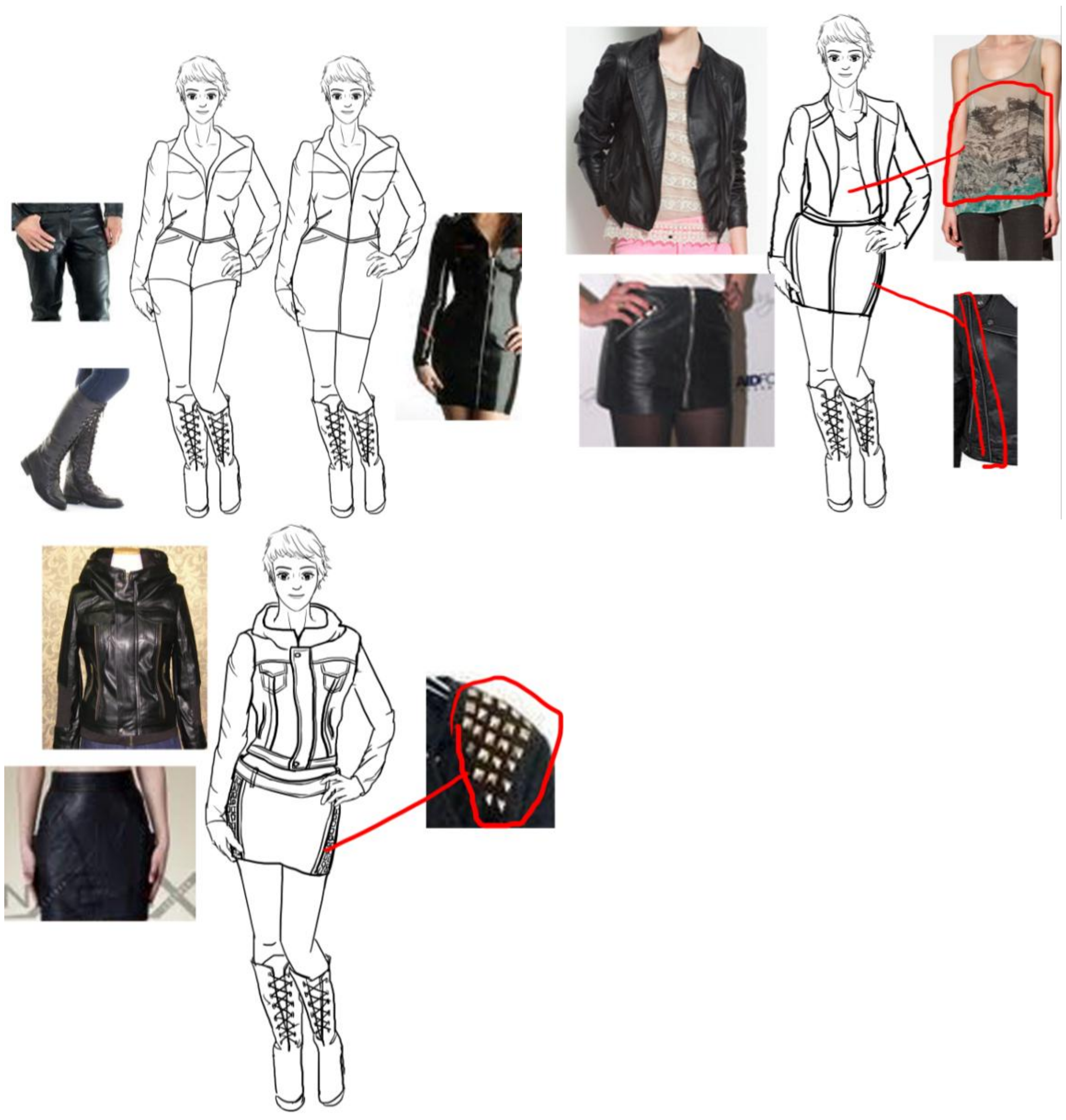
Once I had chosen the body type, Ivy drew a selection of close ups.



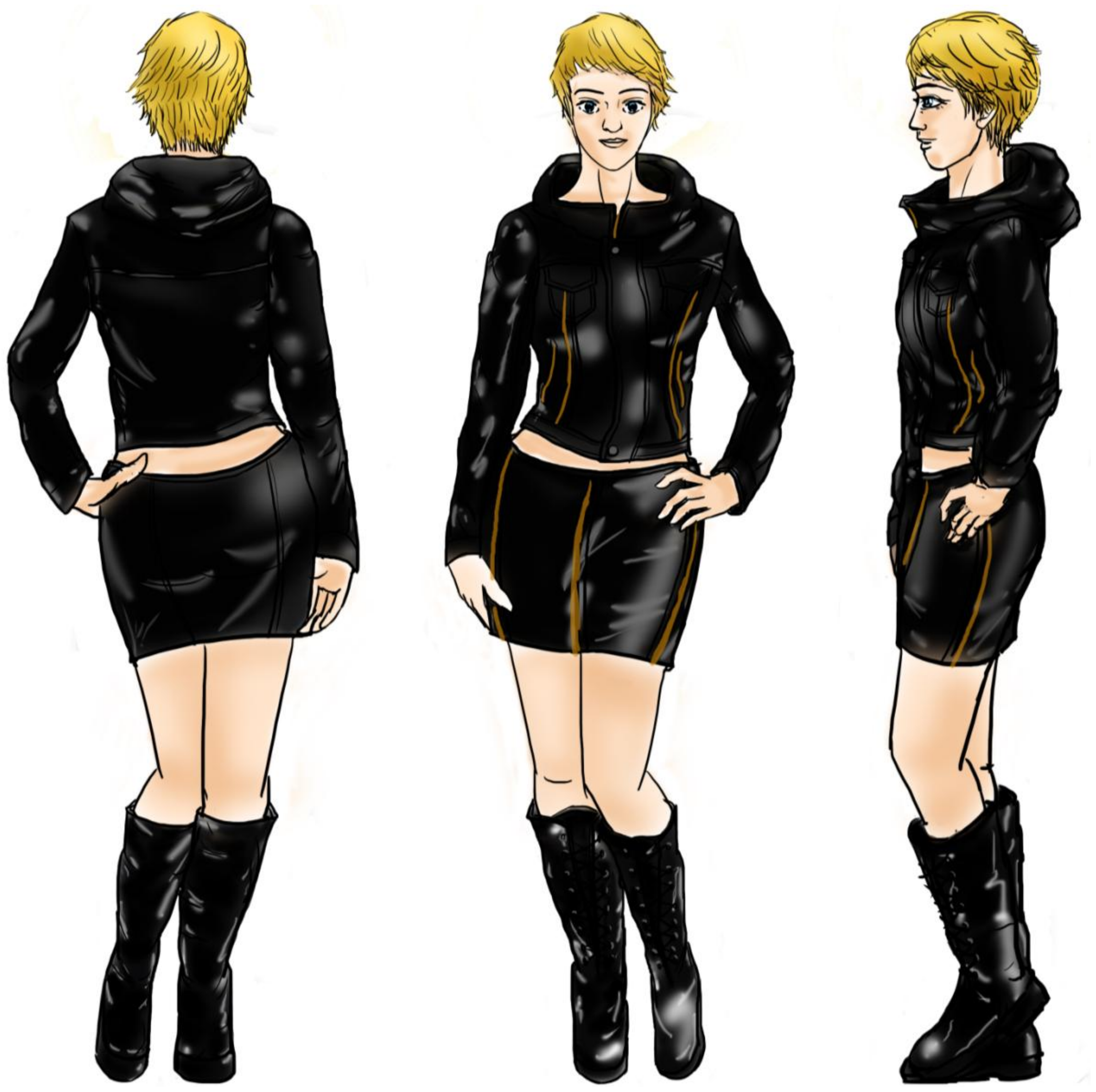
Multiple headshots were drawn so I could sample each look.

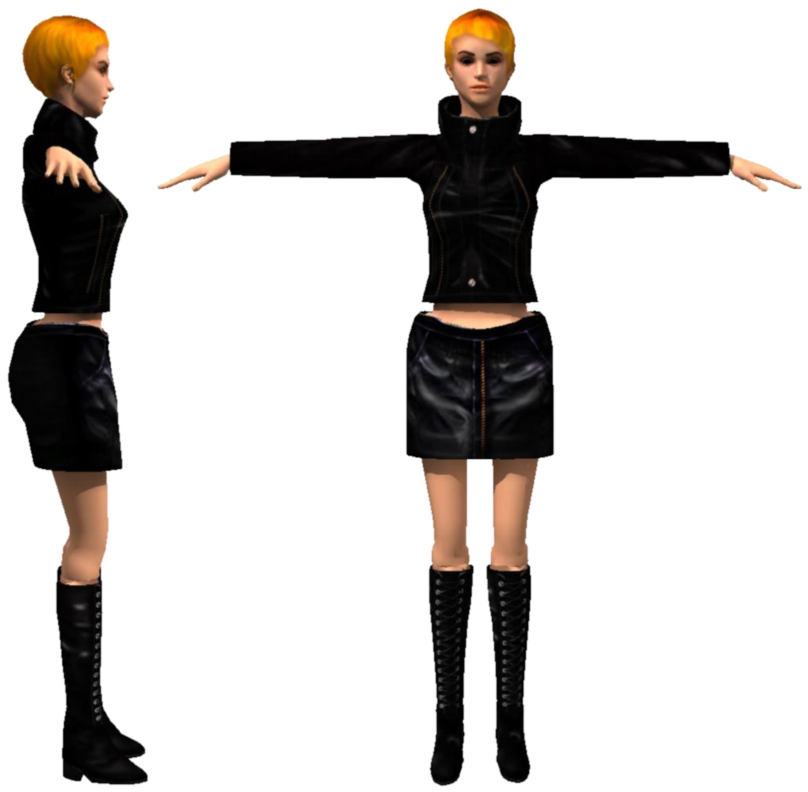


Sketching facial expressions gives the animator some reference material when animating the face.



After her face was designed. Ivy then came up with a selection of outfits for me to sample.


Once we had chosen her outfit, Ivy produced one final piece of concept art to refine our concept.



This is the finished design of our main character Akari. She was modeled using Maya 2012[lix]*, the sketches and textures were created using Photoshop CS5*[lx]*, by Ivy Tsai.*

Female main characters are a rarity in video games. The majority of gamers are male and it is more difficult to relate to a character when they are of the opposite gender. It was decided that the main character for the game would be female to make the games story more unique and appeal to a niche audience who may be turned away from the female stereotype represented in most modern video games. A rational, independent female main character that is realistic in terms of physical appearance is a break from the stereotype set by game developers and can act as a good role model.

Enemies

The enemies are humanoids that were created by harvesting the souls from living humans and forcing them into bodies that were created from extra-dimensional matter. They are forced to obey their creator and are perfect cannon fodder. They are Akari's opponents are they will stop at nothing to put her down. Reverse-engineered extra-dimensional matter is quite affective against them; contact with this element separates the human soul from the matter, destroying the body instantly.

These enemies attack in swarms, so when they move towards Akari, they must separate from each other to avoid stacking.

Designing enemy characters is on the itinerary for our artist, so placeholder models have been used in the meantime.

Model Reference

Psionic's 3D Models

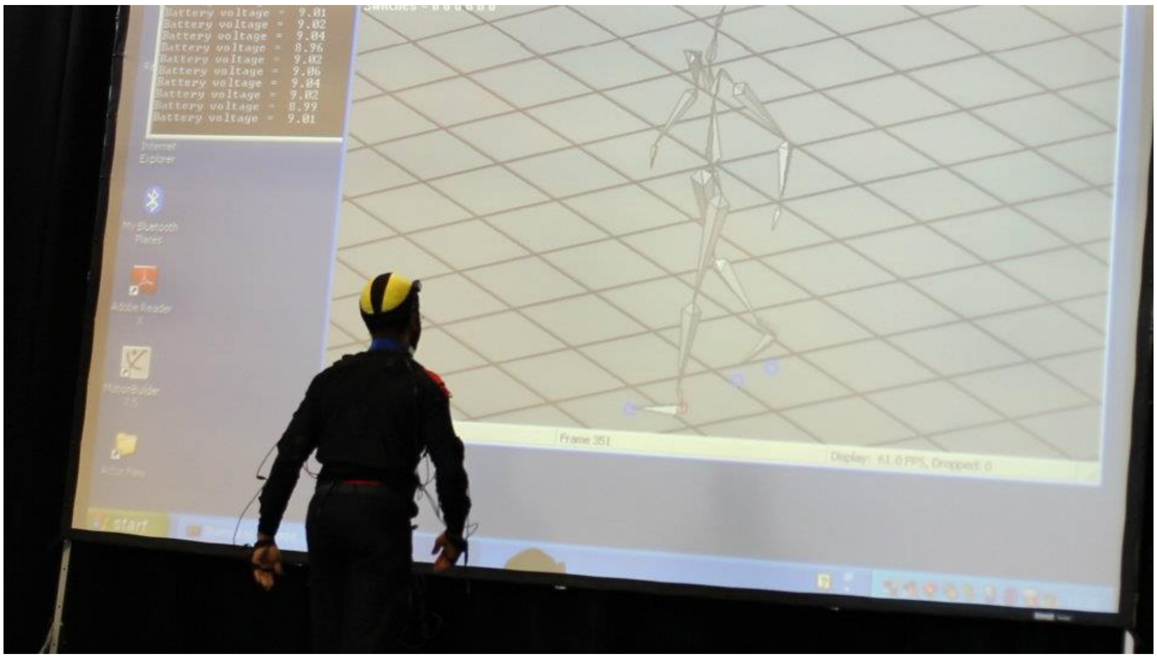
http://www.psionic3d.co.uk/?page_id=25

Models used: Zombie, Ninja, Dwarf

Motion Capture

The university has access to a motion capture suit so Akari's animations are made solely from motion-captured data. Charles-Jean Boucher was Akari's actor and he recorded all of the animations. The data recorded would render instantly in Animaview[lxi], where the animation were recorded as a .bvh file. Ivy used MotionBuilder[lxii] to skin these animations to the character mesh. Because there were multiple .bvh files. Ivy had to apply them to an actor node[lxiii] in MotionBuilder, which could then be applied to the character mesh.

Akari – Jog Cycle
Link



```
Battery voltage =  9.01
Battery voltage =  9.02
Battery voltage =  9.04
Battery voltage =  8.96
Battery voltage =  9.02
Battery voltage =  9.06
Battery voltage =  9.04
Battery voltage =  9.02
Battery voltage =  8.99
Battery voltage =  9.01
```

Akari's animations were performed by Charles-Jean Boucher with an Animazoo Motion Capture Suit[lxiv].

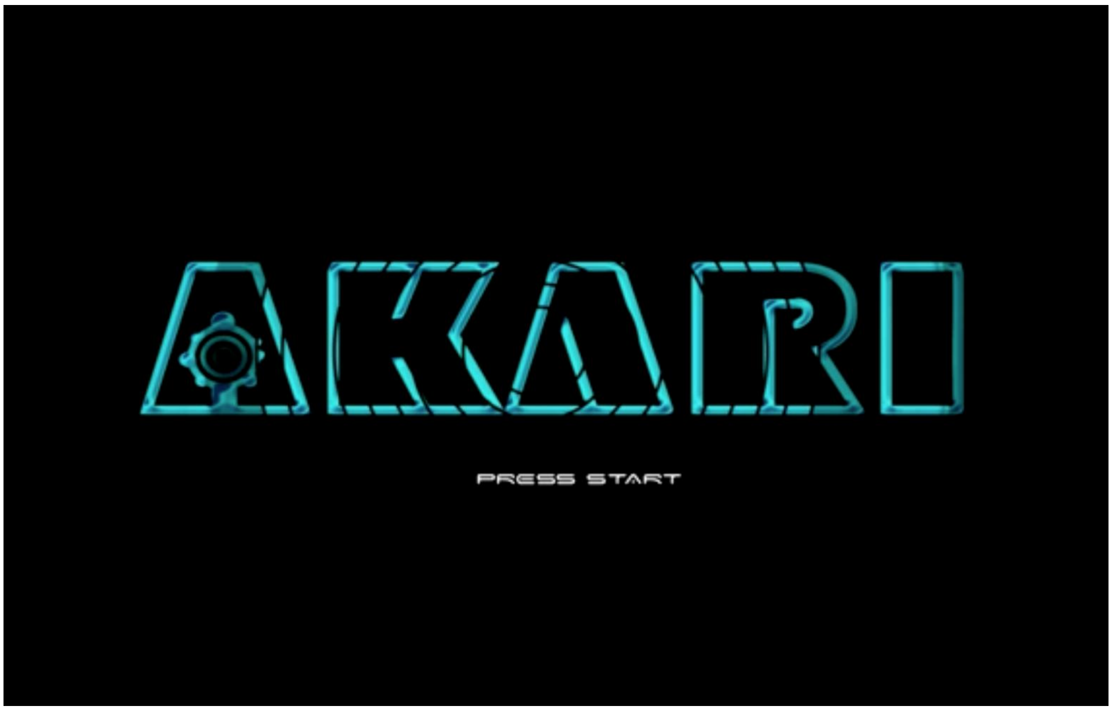
A projector allowed us to view Animaview while performing for instant feedback on the captured data.

Title Sequence and Main Menu

An opening sequence is used to introduce the game to the player, allows them to navigate through the games preferences before starting the gameplay and sets the tone of the game so the player knows what to expect.

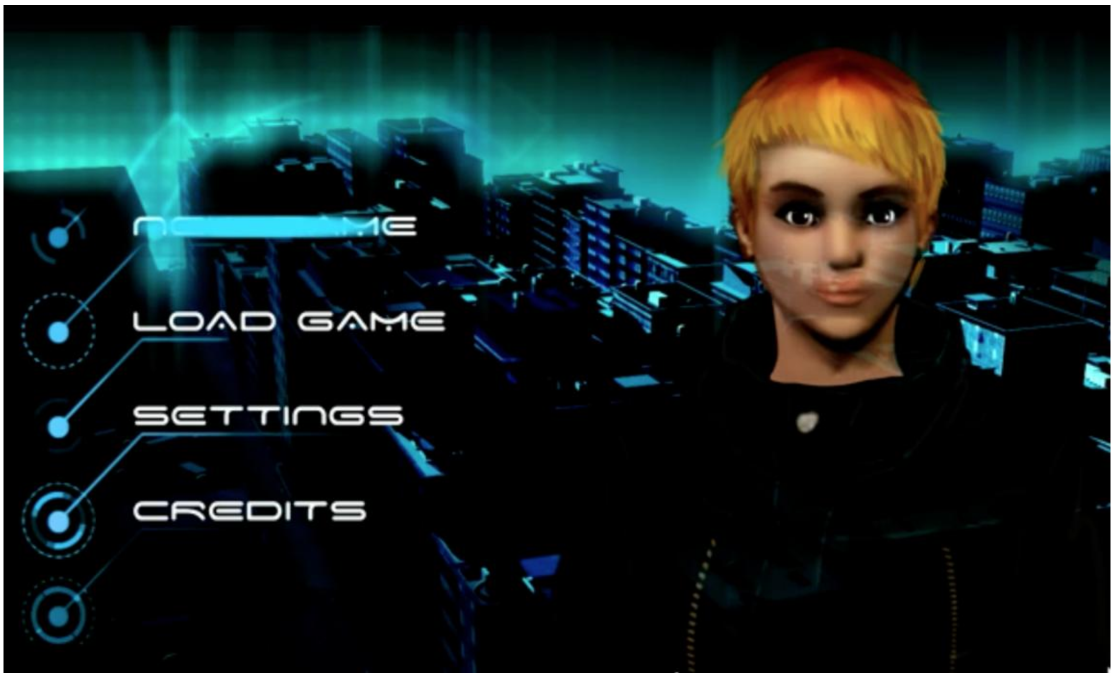
Title Sequence

The title of Akari fades in from black, while some ambient music plays, setting the dark and mysterious tone of the game. The title sequence is comprised of multiple .png images, that are played in a sequence to form an
animation.

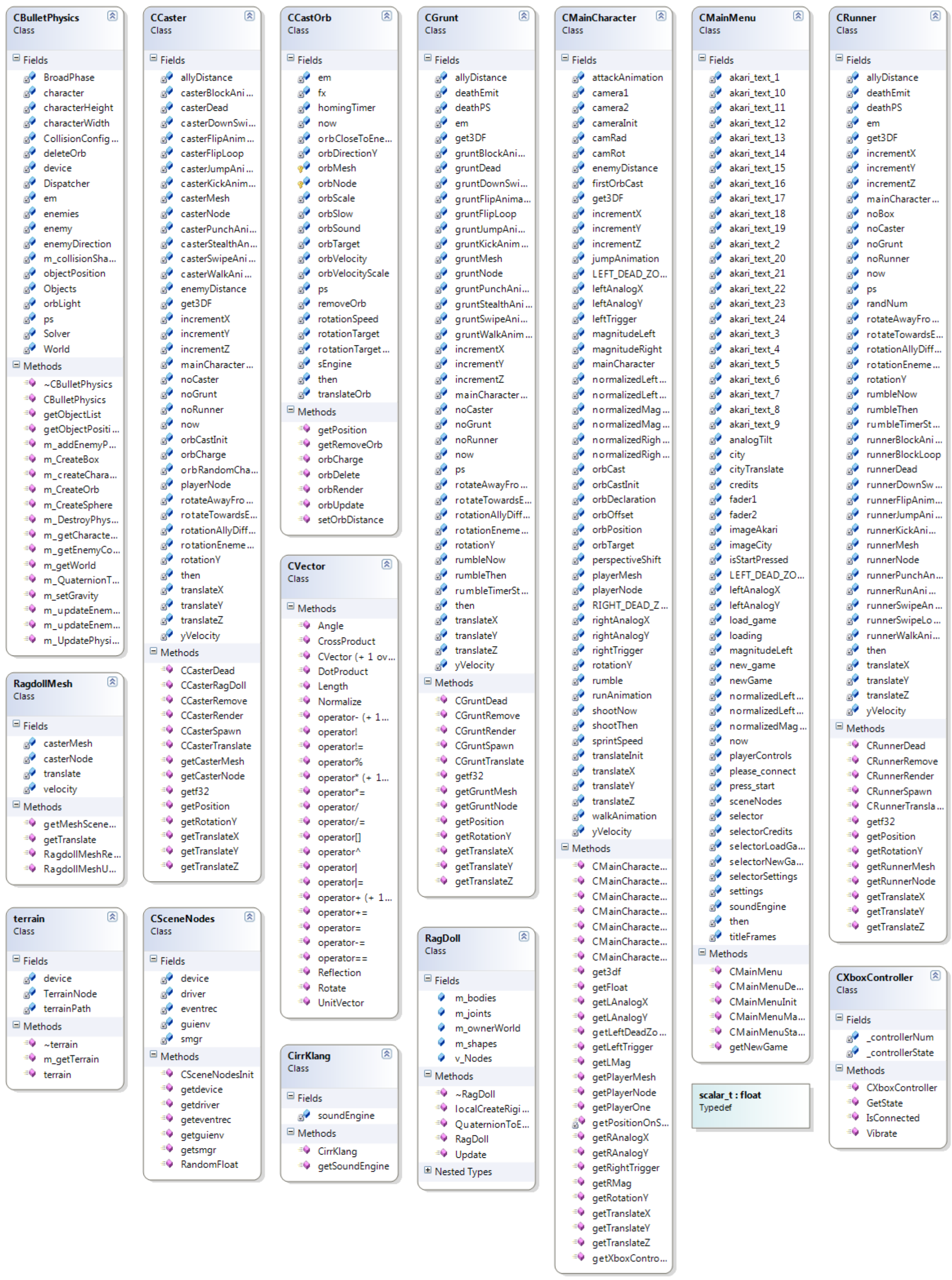


Main Menu

The main menu is comprised of multiple .png images that can be layed out in an aesthetically pleasing fashion. The picture of Akari is a prerendered .png image from Maya. The city background was created using the ghostTown [lxv] plugin for 3ds Max.



Class Diagram



};

Implementation

CSceneNodes Class

The declaration of the following classes are necessary to render a scene using Irrlicht:

- IVideoDriver is needed to perform 2D and 3D graphics functions;
- IGUIEnvironment is used to display variable GUI elements;
- IEventReceiver is used to read keyboard, mouse and joypad input;
- ISceneManager manages scene nodes, mesh resources camera and more;
- IrrlichtDevice is used to create a window to render the game in.

The methods of these classes are vital to implementing the various elements of any game made with Irrlicht. The object created from the CSceneNodes class must be present in any class that require these methods, so this object must be passed into the parameters of the methods of the other classes in this game.

```
void CSceneNodes::CSceneNodesInit()
{
        device = createDevice(video::EDT_OPENGL, dimension2d<u32>(1360,
        768), 16, false, true, false, 0);
        //if(!device){return 1;}
        driver = device->getVideoDriver();
        smgr = device->getSceneManager();
        guienv = device->getGUIEnvironment();
        srand(time(NULL));
};
```

This class has an initialization method, where the Irrlicht class objects are declared, that is called once at execution. Each of these objects has their own getter method that can be used to call these objects at any time.

As this class is present in most of the other classes, so the RandomFloat method was included, for convenience. This method generates a random floating-point number.

CXboxController Class

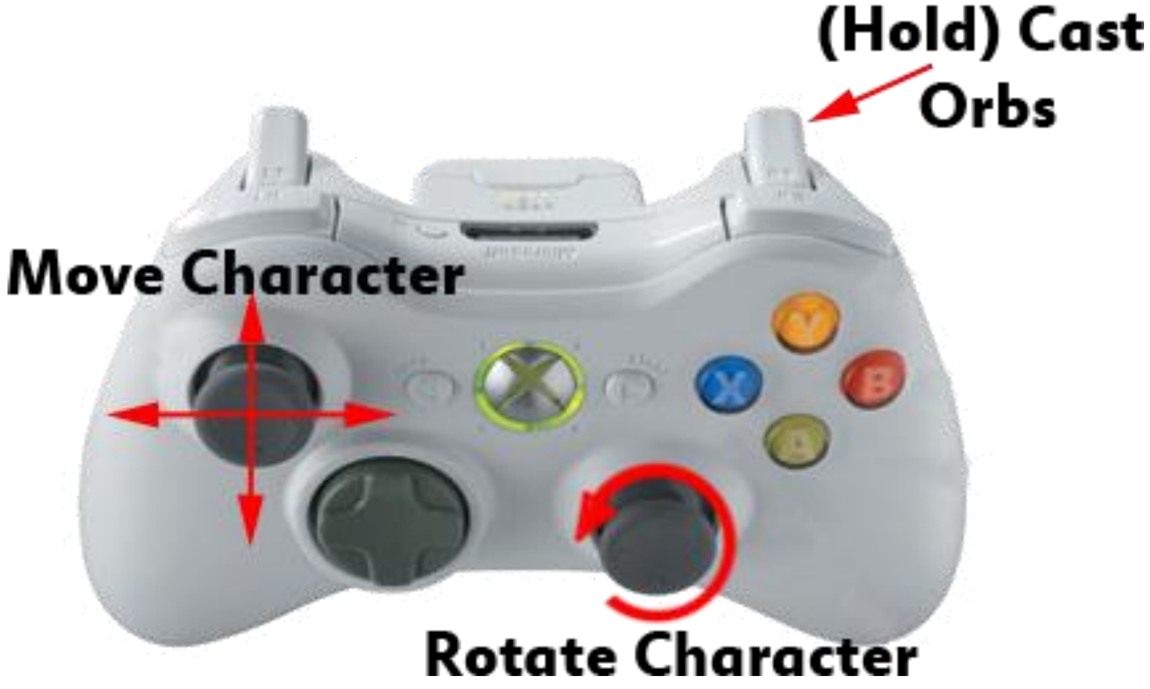
This class assists in the implementation of XInput. When a class is declared, a player number must be stated in the parentheses, as XInput can recognize up to four Xbox 360 controllers.

The GetState method, returns the Xbox 360 controller input value. It returns a XINPUT_STATE, XINPUT_GAMEPAD_X for example, or a floating point number to represent the analogue sticks and triggers. The floating-point number would increase depending on how much you tilt the analogue sticks or how far you pull the triggers.

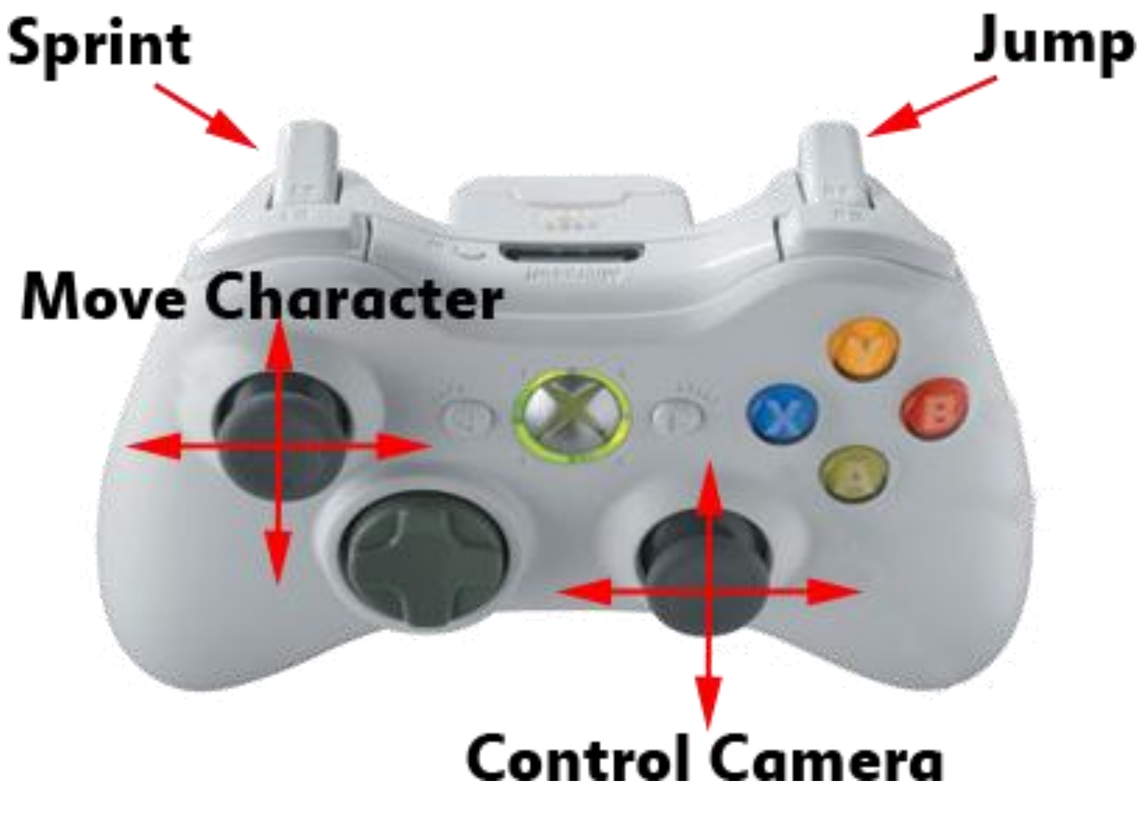
The isConnected method returns a Boolean that states whether the specified controller is connected or not.

The Vibrate method, when called, will vibrate the controller at the floating-point values, specified in the parentheses. The first parameter determines the vibration of the left motor in the controller, and the second determines the vibration in the right motor of the controller.

Isometric Controls



Third-Person Controls



CBulletPhysicsClass

This class was based on a tutorial[lxvi], by Punong Bisyonaryo, that can be found on the Irrlicht development forums[lxvii]. It has methods that create Irrlicht graphics objects, applies bullet collision to them, and adds them to a list.

When the class is declared, the sceneNodes object is passed into it as an argument, which is then assigned to the device object. As the class will be creating Irrlicht graphics objects, the sceneNodes object is necessary as an argument.

```
void CBulletPhysics::m_CreateBox(const btVector3 &TPosition, const
irr::core::vector3df &TScale, btScalar TMass)
```

The m_CreateBox method creates a cube and its start position, scale and mass are defined by the values entered in the parameters.

```
scene::ISceneNode *Node = device->getsmgr()->addCubeSceneNode(1.0f);
```

Using the sceneNode object, a cube is added to the scene, textures and material nodes are added to it.

```
btTransform Transform;
Transform.setIdentity();
Transform.setOrigin(TPosition);
```

The Transform class is used to manipulate the position of objects.

```
btDefaultMotionState  *MotionState = new btDefaultMotionState(Transform);
```

The MotionState class is declared, and the Transform object is passed into it as an argument. A MotionState updates the position of the render object from the physics body.

```
btVector3 HalfExtents(TScale.X * 0.5f, TScale.Y * 0.5f, TScale.Z * 0.5f);
btCollisionShape *Shape = new btBoxShape(HalfExtents);
```

The shape of the rigid body is declared as a btBoxShape and its scale is set to the same as the Irrlicht box.

```
btVector3 LocalInertia;
Shape->calculateLocalInertia(TMass, LocalInertia);
```

The local inertia is declared and is calculated from the mass of the object.

```
btRigidBody *RigidBody = new btRigidBody(TMass, MotionState, Shape,
LocalInertia);
RigidBody->applyDamping(1000);
```

Finally, the rigid body is declared with the arguments; object mass, its MotionState, the shape of the object and its local inertia. To allow the object to have diminishing returns on impact, damping is applied to let it slow to a halt sooner.

```
RigidBody->setUserPointer((void *)(Node));
```

Because Irrlicht and Bullet cannot directly communicate with one another, a user pointer can be set to hold the Irrlicht node data.

```
World->addRigidBody(RigidBody);
Objects.push_back(RigidBody);
```

The finished rigid object is then added into the physics world and pushed into the physical objects vector.

```
void CBulletPhysics::m_CreateOrb(const btVector3 &TPosition,btScalar TRadius, f32
rotation)
```

The m_CreateOrb method creates an orb whenever the player holds down the right trigger. Each orb has a rigid body node applied to it, so they interact with every other physics object in the scene. The rotation of Akari is passed into this method so that the orb can be fired in the right direction.

```
RigidBody->setLinearVelocity(btVector3(cos(degToRad((-
rotation)+90))*600,5.0f,sin(degToRad((-rotation)+90))*600));
```

The object is created in the same way as the box, except that an initial velocity is set. The velocity is calculated from the current rotation of Akari using trigonometry.

```
void CBulletPhysics::m_UpdatePhysics(CSceneNodes* sceneNodes, f32
TDeltaTime, vector<CGrunt*>& grunts, vector<CRunner*>& runners,
vector<CCaster*>& casters)
```

An update method ensures that the physics on each of the objects on this list are updated accordingly.

```
for(core::list<btRigidBody *>::Iterator it = Objects.begin(); it != Objects.end(); ++it)
{
        scene::ISceneNode *Node = static_cast<scene::ISceneNode *>((*it)-
      >getUserPointer());
        objectPosition = Node->getPosition();
```

A loop iterates through the physics objects vector to update their position and rotation.

```
btVector3 Point = TObject->getCenterOfMassPosition();
Node->setPosition(core::vector3df((f32)Point[0], (f32)Point[1], (f32)Point[2]));
```

The position of the Irrlicht cube is set to the same position as the Bullet rigid body.

```
btVector3 EulerRotation;
m_QuaternionToEuler(TObject->getOrientation(), EulerRotation);
Node->setRotation(core::vector3df(EulerRotation[0], EulerRotation[1],
EulerRotation[2]));
```

The Irrlicht cubes rotation is also set to the same rotation of the Bullet rigid body using the m_QuaternionToEuler method.

```
for(vector<CGrunt*>::iterator currentGrunt = grunts.begin(); currentGrunt !=
grunts.end(); currentGrunt++)
```

```
{
          if(Node->getTransformedBoundingBox().intersectsWithBox((*currentGrunt)-
        >getGruntNode()->getTransformedBoundingBox()))
        {
                    (*currentGrunt)->CGruntRemove(sceneNodes);
```

The vectors for each of the enemy classes are passed into this method as well. They contain the updated positions of the enemy characters, and a bounding box collision detection system is used to detect if the enemies collide with the orbs. If they do collide, then that enemy is removed from the scene and placed in the graveyard.

CirrKlang Class

This class declares and contains the irrKlang class.

```
soundEngine->getSoundEngine()-  >play2D("Sound/Shoot1.wav",false);
```

It has a getter method that returns the irrKlang object, so the sound engine can easily be passed into the parameters of other classes.

CMainCharacter Class

This class contains the declaration of the main character and is where a lot of her actions are implemented. It has three methods; the first is called CMainCharacterRender. It is where her model and texture are implemented and where various variables are given their initial values. The correct place to call this method is at the beginning of the code, outside of the main loop.

```
void CMainCharacter::CMainCharacterRender(CSceneNodes*& sceneNodes,
vector3df &initialPosition)
```

The arguments that are passed into this method are the sceneNodes class and coordinates for the main characters starting position.

```
playerMesh = sceneNodes->getsmgr()->getMesh("Models/sydney.md2");
//playerMesh = sceneNodes->getsmgr()-
>getMesh("Models/Akari/Akari_Animation.X");
```

The Sydney.md2 model is the placeholder animated mesh that was used while Akari was being designed.

```
playerNode = sceneNodes->getsmgr()->addAnimatedMeshSceneNode(playerMesh);
```

The playerNode object is an animated mesh scene node, and is where all of the attributes of the character scene node are stored.

```
playerNode->setMaterialFlag(EMF_LIGHTING, true);
```

The setMaterialFlag method sets whether the material of the mesh is affected by the lighting included in the scene. So shadows may be cast onto the mesh itself if a light is present in the scene.

```
playerNode->setScale(vector3df(0.5,0.5,0.5));
```

The setScale method sets the scale of the mesh. The scale may vary depending on how the model was scaled when it was created.

```
playerNode->setMaterialTexture(0,sceneNodes->getdriver()-
>getTexture("Models/sydney.bmp"));
```

The setMaterialTexture method, applies a texture to the mesh stored in the scene node. As long as the model is properly UV mapped in the appropriate computer graphics software, then the texture should map correctly.

```
playerNode->addShadowVolumeSceneNode();
```

The addShadowVolumeSceneNode method applies a depth map shadow to the mesh, as long as a light source is implemented into the scene.

The second method is called CMainCharacterTranslate, it is essentially the update method of this class and is supposed to be called inside the main loop of the game. It contains all of the gameplay of the main character so it is a very complicated and important method.

```
mainCharacter = new CXboxController(1);
```

mainCharacter is the XInput object that reads the Xbox 360 controllers input. Its methods are accessed regularly in this class.

```
//------ Value of the triggers. (0-255) ------//
leftTrigger = mainCharacter->GetState().Gamepad.bLeftTrigger;
rightTrigger = mainCharacter->GetState().Gamepad.bRightTrigger;

// Right Trigger
if(rightTrigger > 0)
{
	if(perspectiveShift == false)
	{
		if(orbCastInit == true)
		{
```

The right trigger allows Akari to fire orbs, this is done by checking whether the input value of the right trigger is above zero. The rightTrigger variable will change to a value between 0 and 255, depending on how far the trigger is pulled. The perspectiveShift variable regulates which gameplay mode is active, if this

Boolean equals false, then the isometric view is displayed, and if it is true, the third-person view is displayed.

```
shootThen = sceneNodes->getdevice()->getTimer()->getTime();
orbCastInit = false;
}
        shootNow = sceneNodes->getdevice()->getTimer()->getTime();
        if(shootNow >= shootThen +100)
        {
                physicsEngine-
>m_CreateOrb(btVector3(translateX,translateY+20,translateZ), 2, rotationY);
                soundEngine->getSoundEngine()-
>play2D("Sound/Shoot1.wav",false);
                orbCastInit = true;
          }
```

shootThen and shootNow are timers. They ensure that there are regular intervals in-between each shot. shootThen is given a time value when the right trigger is pressed, and shootNow will be updated with the current time every frame. Once shootNow is worth 100 more than shootThen, a new orb is created and a sound effect is played. The arguments passed into m_CreateOrb are the current position of the main character, the radius of the orb and the current rotation of the player. The orb will travel in the direction the main character is facing.

```
if(attackAnimation == false)
{
        playerNode->setMD2Animation(scene::EMAT_ATTACK);
        playerNode->setAnimationSpeed(25);
        attackAnimation = true;
}
```

The setMD2Animation method sets the animation cycle of the model and setAnimationSpeed sets its frameloop speed.

```
//----- Left Thumb Stick -----//
// Implements the Dead zone for the left thumb stick
leftAnalogX = mainCharacter->GetState().Gamepad.sThumbLX;
leftAnalogY = mainCharacter->GetState().Gamepad.sThumbLY;

magnitudeLeft = sqrt(leftAnalogX*leftAnalogX + leftAnalogY*leftAnalogY);

// determines what direction the thumbstick is pushed
normalizedLeftAnalogX = leftAnalogX / magnitudeLeft;
normalizedLeftAnalogY = leftAnalogY / magnitudeLeft;

normalizedMagnitudeLeft = 0;
```

leftAnalogX and leftAnalogY represent the direction the left analog stick is pushed with floating point numbers. It is necessary to normalize these values to determine how far the thumb stick is pushed to calculate how big its dead zone should be. A small radius around the center point represents a dead zone for the thumb stick, a dead zone is needed because the thumb sticks are not the most precise input devices and an irregular value will occur if the thumb sticks are in its rest position.

```
LEFT_DEAD_ZONE = 8000;
if (magnitudeLeft > LEFT_DEAD_ZONE)
{
	if(perspectiveShift == false)
	{
			incrementX = leftAnalogX/440*sprintSpeed;
			incrementZ = leftAnalogY/440*sprintSpeed;
			translateX += incrementX*dt;
			translateZ += incrementZ*dt;
```

The values of the left analog stick are translated into the incremental values of the main characters movement. These values are incremented onto the main characters current position, giving the appearance that the main character is moving around the environment. The characters speed will change depending how far the player tilts the analog stick. This character control method is only valid for the isometric gameplay perspective.

```
if(magnitudeLeft > 32767){magnitudeLeft = 32767;
```

An if statement keeps the left analogs magnitude below a specified amount.

```
rotationY = atan2(leftAnalogX,leftAnalogY)*180.0 / 3.14159265;
```

rotationY sets the rotation of the main character along the Y-axis. Akari will rotate towards the direction she is travelling.

```
else if (perspectiveShift == true)
{
        if(leftAnalogY >= 20000)
        {
                translateX += cos(degToRad(rotationY-90))*80*sprintSpeed*dt;
                translateZ -= sin(degToRad(rotationY-90))*80*sprintSpeed*dt;
        }
        else if(leftAnalogY <= -20000)
        {
                translateX -= cos(degToRad(rotationY-90))*80*sprintSpeed*dt;
                translateZ += sin(degToRad(rotationY-90))*80*sprintSpeed*dt;
        }
        if(leftAnalogX >= 10000)
        {
                translateX += cos(degToRad(rotationY))*80*sprintSpeed*dt;
                translateZ -= sin(degToRad(rotationY))*80*sprintSpeed*dt;
        }
        else if(leftAnalogX <= -10000)
        {
                translateX += cos(degToRad(rotationY+180))*80*sprintSpeed*dt;
                translateZ -= sin(degToRad(rotationY+180))*80*sprintSpeed*dt;
        }
}
```

If perspectiveShift is true, then the third-person gameplay mode is enabled. This time, the direction Akari will travel is relative to her current rotation; so if the analog stick is pushed up, then she will travel in the direction she is facing, but if it pushed down, then she will back peddle.

```
if(runAnimation == false)
{
	playerNode->setMD2Animation(scene::EMAT_RUN);
	playerNode->setAnimationSpeed(25);
	runAnimation = true;
}
```

An if statement is present that makes sure that the run animation is always playing if the character is moving.

```
else if (magnitudeLeft <= LEFT_DEAD_ZONE)
{
	magnitudeLeft = 0.0;
	normalizedMagnitudeLeft = 0.0;
	leftAnalogX = 0.0;
	leftAnalogY = 0.0;
	if(runAnimation == true)
	{
		playerNode->setMD2Animation(scene::EMAT_STAND);
		playerNode->setAnimationSpeed(25);
		runAnimation = false;
	}
}
```

Once the value of the analog stick has dropped below the dead zones value, Akari will stand still. An if statement is present that makes sure her standing animation is playing.

```
//----- Right Thumb Stick -----//
// Implements the Dead zone for the right thumb stick
rightAnalogX = mainCharacter->GetState().Gamepad.sThumbRX;
rightAnalogY = mainCharacter->GetState().Gamepad.sThumbRY;

magnitudeRight = sqrt(rightAnalogX*rightAnalogX + rightAnalogY*rightAnalogY); //
determines how far the thumbstick is pushed
RIGHT_DEAD_ZONE = 20000;

// determines what direction the thumbstick is pushed
normalizedRightAnalogX = rightAnalogX / magnitudeRight;
normalizedRightAnalogY = rightAnalogY / magnitudeRight;

normalizedMagnitudeRight = 0;
```

The right analog stick is normalized in the same way as the left analog stick.

```
// check if the controller is outside a cercular dead zone
if (magnitudeRight > RIGHT_DEAD_ZONE)
{
        // clip the magnitude at its expected maximum value
        if(magnitudeRight > 32767){magnitudeRight = 32767;
        }
        magnitudeRight -= RIGHT_DEAD_ZONE;
        normalizedMagnitudeRight        =       magnitudeRight       /       (32767       -
RIGHT_DEAD_ZONE);
        if(perspectiveShift == false)
        {
                rotationY = atan2(rightAnalogX,rightAnalogY)*180.0 / 3.14159265;
        }
```

If perspectiveShift is equal to true, then the rotation of Akari is equivalent to the angle the right analog is pointing in. This is how the player aims where the orbs will be fired

```
else if (perspectiveShift == true)
{
if(rightAnalogX >= 10000 || rightAnalogX <= -10000)
{
          rotationY += (rightAnalogX/90)*dt;
}
if(rotationY >= 361)
{
          rotationY = 0;
}
else if(rotationY <= -1)
{
          rotationY = 360;
}
camRot.Y += (rightAnalogX/2000000)*dt;
camRad += (rightAnalogY/100)*dt;
if(camRot.Y>= 0.01){camRot.Y = -0.00001;}
else if(camRot.Y <= -0.000011){camRot.Y=0.011;}
}
        }else
        {
          magnitudeRight = 0.0;
          normalizedMagnitudeRight = 0.0;
          rightAnalogX = rightAnalogX;
          rightAnalogY = rightAnalogY;
        }
```

If perspectiveShift is equal to false, then Akai's rotation is incremented in the direction that the right analog is tilted in the X-axis. Some camera variables are also incremented, but the camera position and rotation are dealt with in another method.

```
// Gravity
incrementY = (-10.05 * yVelocity);
translateY += incrementY*dt;
if(translateY >= 1.0)
{
        yVelocity += 0.1;
        if (jumpAnimation == false)
        {
          playerNode->setMD2Animation(scene::EMAT_JUMP);
          playerNode->setAnimationSpeed(25);
          jumpAnimation = true;
        }
}
else if(translateY <= 0.0)
{
        translateY = 0.0;
        yVelocity = 0.0;
        jumpAnimation = false;
}
```

Akari also has the ability to jump, in third-person mode. Her velocity in the Y-axis is incremented when the player pushes the jump button. It also calculates gravity, the longer she is in the air, the stronger the force of gravity becomes, so that value is included in the increment equation.

The CMainCharacterCamera method is similar to the translate/update method, as it should be should be called every frame. It updates the positions of both cameras and sets the active camera in the scene.

```
if(cameraInit==false)
{
        camera1 = sceneNodes->getsmgr()-
>addCameraSceneNode(0,vector3df(0,0,0),vector3df(0,0,0));
        camera2 = sceneNodes->getsmgr()-
>addCameraSceneNode(0,vector3df(0,0,0),vector3df(0,0,0));
        camera2->setParent(playerNode);
        camRad = -100;
        cameraInit = true;
}
```

The first if statement initializes some variables that was relevant to the scenes cameras.

```
if(perspectiveShift == false)
{
        sceneNodes->getdevice()->getSceneManager()->setActiveCamera(camera1);
}
else if(perspectiveShift == true)
{
        sceneNodes->getdevice()->getSceneManager()->setActiveCamera(camera2);
}
```

The second controls which camera is the active camera.

```
camera1->setPosition(vector3df(translateX,translateY+100,translateZ-100));
camera1->setTarget(vector3df(translateX,translateY+15,translateZ));
camera2-
>setPosition(getPositionOnSphere(camRot.Y+1.55,camRot.X+50,camRad));
camera2->setTarget(vector3df(translateX,translateY+15,translateZ));
```

The methods that are called next set the positions of the cameras and sets the cameras target positions, which are set onto Akari.

```
vector3df CMainCharacter::getPositionOnSphere(f32 angleH, f32 angleV, f32 radius)
{
        f32 posZ = radius * cos(angleV);
        f32 posY = radius * sin(angleV);

        vector3df camPos(0,posY,posZ);

        matrix4 yawMat;
        yawMat.setRotationRadians(vector3df(0,angleH,0));
        yawMat.transformVect(camPos);
        return camPos;
}
```

The getPositionOnSphere method is based on a tutorial[lxviii] that can be found on the Irrlicht development forum. It appears to produce a vector from the rotation of the camera in the X-axis and Y-axis. It allows the camera to rotate around the main character, giving the game a third-person perspective.

CGrunt Class

This class is contains the grunt scene node information, as well as its artificial intelligence. Each in-game character is included into the scene in the same way, but with some key differences.

```
gruntNode->setMaterialType(video::EMT_REFLECTION_2_LAYER);
```

All of the enemy classes have a two-layer reflection material applied to them. This gives the illusion that their bodies reflect light.

```
// Particles
// create a particle system
ps = sceneNodes->getsmgr()->addParticleSystemSceneNode(false);
em = ps->createMeshEmitter(
gruntMesh, // Specify mesh
true, // Use normal Direction
vector3df(0.0f,0.7f,0.0f), // Direction
100.0f, // Normal direction modifier
-1, // mb number
false, // emit from every vertex
100,200, // min and max particles per second
video::SColor(0,255,255,255),       // darkest color
video::SColor(0,255,255,255),       // brightest color
100,200,0,                          // min and max age, angle
core::dimension2df(1.f,1.f),        // min size
core::dimension2df(3.f,3.f));       // max size
```

Each enemy emits black particles from their bodies. This can be achieved by created a particle system, and applying a mesh emitter to it. A mesh emitter uses a specified mesh to emit particles from, so by stating gruntMesh as the emitter, the engine will constantly emit the particles created from the particle system.

```
ps->setPosition(vector3df(translateX,translateY,translateZ));
```

The particle systems position is set is constantly set to the characters current position.

```
mainCharacterDistance = sqrt((mainCharacter.X - translateX) * (mainCharacter.X  - 
translateX) + (mainCharacter.Z - translateZ) * (mainCharacter.Z  - translateZ));
```

The distance between the character and Akari is calculated with the distance formula.

```
rotateTowardsEnemey =  atan2(mainCharacter.Z-translateZ,mainCharacter.X-
translateX)*180.0 / 3.14159265;
```

The enemy characters all follow and attack Akari, so their rotation is calculated by triangulating the current position of Akari.

```
if(mainCharacterDistance <= 12)
{
	this->CGruntRemove(sceneNodes);
	mainCharacterNode->getXboxController()->Vibrate(65535,65535);
	if(rumbleTimerStart == false)
	{
	  rumbleThen = sceneNodes->getdevice()->getTimer()->getTime();
	  rumbleTimerStart = true;
	}
}
```

When the distance between the character and Akari drops below zero, the player is hit, so the character explodes and the players controller starts to vibrate. The original design of the enemies was that when they reach a certain distance from Akari, they would start attacking her, which was demonstrated by their attack animation loop, and when Akari got to close to them, they would backflip away.

```
for(vector<CGrunt*>::iterator currentGrunt = grunts.begin(); currentGrunt !=
grunts.end(); currentGrunt++)
{
        if((*currentGrunt) == this)
          continue;

        allyDistance = sqrt(((*currentGrunt)->getTranslateX() - translateX) *
          ((*currentGrunt)->getTranslateX() - translateX) + ((*currentGrunt)-
>getTranslateZ() - translateZ) * ((*currentGrunt)->getTranslateZ() - translateZ));
        rotateAwayFromAlly  = atan2(translateZ-(*currentGrunt)-
>getTranslateZ(),translateX-(*currentGrunt)->getTranslateX())*180.0 / 3.14159265;
        if(gruntFlipLoop == false)
        {
          if(allyDistance <= 15)
          {
                rotationAllyDifference = rotateAwayFromAlly - rotationY;
                if(rotationAllyDifference < -180)
                        rotationAllyDifference += 360;
                if(rotationAllyDifference > 180)
                        rotationAllyDifference -= 360;
                rotationY += rotationAllyDifference * 4.0 * dt;
          }
          else
          {
                noGrunt = true;
          }
        }
}
```

The enemies avoid colliding with one another with the separation AI behaviour. Every enemy class is contained within its own vector, and a loop is used to allow the current enemy to access the information of every other enemy object in each vector. First the distance between the current character and the ally, present in the current iteration of the loop, is calculated. If the distance between the two allies drops below 15, then the character is steered away from the position of its ally. The direction to steer away from the ally is determined by subtracting the current rotation of the character from the angle between the character and its ally and by using an if statement to check whether this value is above or below 180 degrees. Within these if statements, a variable is incremented with multiples of 360 or -360 and this value is incremented onto the current rotation of the character.

```
noGrunt = false;
noRunner = false;
noCaster = false;
```

Three Booleans are included in this code segment to check whether the character is within steering distance of at least one of its allies.

```
if(noGrunt == true && noRunner == true && noCaster == true)
{
        rotationEnemeyDifference = rotateTowardsEnemey - rotationY;
        if(rotationEnemeyDifference < -180)
          rotationEnemeyDifference += 360;
        if(rotationEnemeyDifference > 180)
          rotationEnemeyDifference -= 360;
        rotationY += rotationEnemeyDifference * 4.0 * dt;
}
```

If the character is not within steering distance of one of its allies, then it will steer towards Akari by default.

```
if(gruntFlipLoop == false)
{
        if(mainCharacterDistance > 5)
        {
          translateX += cos(degToRad(-rotationY))*40*dt;
          translateZ -= sin(degToRad(-rotationY))*40*dt;
          // Walk
          if(gruntStealthAnimation == false)
        {
                gruntSwipeAnimation = false;
                gruntFlipAnimation = false;
                gruntNode->setAnimationSpeed(20);
                gruntNode->setLoopMode(true);
                gruntNode->setFrameLoop(14,30);
                gruntStealthAnimation = true;
        }
      }
}
```

When the character is apart from the main character, its position is updated to the rotation that was calculated earlier in the code. Its animation cycle is also set to walk.

```
void CGrunt::CGruntSpawn(const f32& spawnPositionX,const f32& spawnPositionZ)
{
        translateX = spawnPositionX;
        translateY = 0.0f;
        translateZ = spawnPositionZ;
        gruntDead = false;
}
void CGrunt::CGruntRemove(CSceneNodes*& sceneNodes)
{
        then = sceneNodes->getdevice()->getTimer()->getTime();
        deathPS->setPosition(vector3df(translateX,translateY+10,translateZ));
        translateX = -10000.0f;
        translateY = -10000.0f;
        translateZ = -10000.0f;
        gruntDead = true;
}
```

When the enemies are defeated, the CGruntRemove method moves them to a graveyard off-screen. When the enemies are ready to respawn on the stage, the CGruntSpawn method sends them back to a random position generated in the main loop. An explosion particle effect is also placed at the place of defeat to give the illusion that the enemy explodes into red matter.

CRunnerClass

The runner enemy class behaves identically to the grunt class but several key differences.

```
translateX += cos(degToRad(-rotationY))*100*dt;
translateZ -= sin(degToRad(-rotationY))*100*dt;
```

The movement iteration rate has been increased by 60, so that they move faster than the grunts.

```
if(runnerNode->getFrameNr() == 126||runnerNode->getFrameNr() == 142||
runnerNode->getFrameNr() == 160||runnerNode->getFrameNr() ==
180||runnerNode->getFrameNr() == 192)
        runnerSwipeLoop = false;
if(runnerNode->getFrameNr() == 210)
{
        runnerBlockLoop = false;
        runnerSwipeLoop = false;
}
```

The runner is a different animated mesh than the grunt, so it has different animations. In the first build of the game, when the player gets close to the runner, it would perform a block stance and get pushed back a little.

CCaster Class

The caster is a lot more unique in terms of behaviour. It will still separate from its allies and seek to defeat Akari, but it uses a ranged attack rather than melee.

```
// Cast orbs at target
else if(mainCharacterDistance <= 400 && mainCharacterDistance >= 80)
{
        if(casterSwipeAnimation == false)
        {
          casterStealthAnimation = false;
          casterFlipAnimation = false;
          casterNode->setAnimationSpeed(15);
```

```
            casterNode->setLoopMode(true);
            casterNode->setFrameLoop(117,128);
            casterSwipeAnimation = true;
            }
```

When the caster is within 400 units of the Akari, it will run through an attack animation cycle and begin to cast orbs in her direction.

```
// Charge Orbs
if(orbCastInit == false)
{
        then = sceneNodes->getdevice()->getTimer()->getTime();
        casterOrb = new CCastOrb();
        or1.push_back(casterOrb);
        or1.back()->orbRender(sceneNodes, soundEngine, 1);
        orbCharge = 0.01;
        orbRandomCharge = sceneNodes->RandomFloat(800, 1300);
        orbCastInit = true;
}
// Orb Charge Up
now = sceneNodes->getdevice()->getTimer()->getTime();
orbCharge += 6.0 *dt / 10;
or1.back()->orbCharge(dt,
vector3df(orbCharge,orbCharge,orbCharge),vector3df(translateX,translateY+20,trans
lateZ), -rotationY+90);
        if(now >= then+orbRandomCharge)
          orbCastInit = false;
          }
```

The orbs have their own class, so when the caster is within range of Akari, a new orb object is created. First they start to charge up their orb, the longer they charge, the larger the orb becomes. The charge-up time is random.

```
// Back peddle from target
if(mainCharacterDistance < 80)
{
        if(casterSwipeAnimation == false)
        {
          casterStealthAnimation = false;
          casterFlipAnimation = false;
          casterNode->setAnimationSpeed(15);
          casterNode->setLoopMode(true);
          casterNode->setFrameLoop(117,128);
          casterSwipeAnimation = true;
        }
        // Charge Orbs
        if(orbCastInit == false)
        {
          then = sceneNodes->getdevice()->getTimer()->getTime();
          casterOrb = new CCastOrb();
          or1.push_back(casterOrb);
          or1.back()->orbRender(sceneNodes, soundEngine, 1);
          orbCharge = 0.01;
          orbRandomCharge = sceneNodes->RandomFloat(500, 600);
          orbCastInit = true;
        }
        // Orb Charge Up
        now = sceneNodes->getdevice()->getTimer()->getTime();
        orbCharge += 6.0 *dt / 10;
        or1.back()->orbCharge(dt,
vector3df(orbCharge,orbCharge,orbCharge),vector3df(translateX,translateY+20,trans
lateZ), -rotationY);
        //or1.back()->orbCharge(dt,
vector3df(orbCharge,orbCharge,orbCharge),casterNode->getJointNode("LeftHand")-
>getAbsolutePosition(), -rotationY);

        if(now >= then+orbRandomCharge)
          orbCastInit = false;

        translateX -= cos(degToRad(-rotationY))*50*dt;
        translateZ += sin(degToRad(-rotationY))*50*dt;
}
```

If Akari gets to close to the caster, they will back pedal away from her and fire smaller orbs to try and deter her.

```
// Updates the orbs
std::vector<CCastOrb*>::iterator orbIt;
for(orbIt = or1.begin(); orbIt != or1.end(); ++orbIt)
{
        if (*orbIt && !(*orbIt)->getRemoveOrb())
                        (*orbIt)->orbUpdate(dt, sceneNodes, soundEngine, orbCastInit,
enemyDistance, mainCharacter, vector3df(translateX,translateY,translateZ));
}
```

The orbs update method is also called in this class. This method contains the orbs AI as well as updates its position.

CCastOrb Class

These orbs are implemented in a very similar way to the melee enemies of the game, except that they are created in real time and not at the start of the code. A render method creates a new orb object, textures it and applies a particle effect to it.

```
orbSound = soundEngine->getSoundEngine()->play3D("Sound/Orb.mp3",
orbVelocity, false, true);
fx = orbSound->getSoundEffectControl();
```

The orbs have a 3D sound effect applied to them, so the closer the player is to the orb, the louder the sound becomes.

```
void CCastOrb::orbCharge(const f32& dt, const vector3df& scale, const vector3df&
translate, const f32& playerRotation)
{
        orbDirectionY = -playerRotation+90;
        orbVelocity = translate;
        orbVelocityScale = scale.X*100000*dt;
        orbScale = scale;
        orbNode->setScale(scale);
        orbNode->setPosition(translate);
        rotationSpeed = 8.0;
}
```

The orbCharge method keeps the created orb at the casters position, and it will increase the orbs scale the longer it is charged.

The orbs behave very differently depending of their scale.

```
if(orbVelocityScale <= 300)
{
        orbSlow+=0.001;
        if(orbSlow >= 1.0)
          orbSlow = 1.0;
```

```
	rotationTarget =  atan2(casterPosition.Z-orbVelocity.Z,casterPosition.X-orbVelocity.X)*180.0 / 3.14159265;
}
```

If the orb's scale is below 300, then the orbs gradually get faster, up to a point, and they follow the caster around.

```
else if(orbVelocityScale > 300)
{
	orbSlow = 1.0;
	homingTimer = 1000;
	if(now >= then+homingTimer)
	{
	  rotationTarget =  atan2(targetPosition.Z-orbVelocity.Z,targetPosition.X-orbVelocity.X)*180.0 / 3.14159265;
	  rotationSpeed -= 0.01;
	}
}
```

If the orbs are above 300, then their speed is based on how big they are and they home onto their target. Their rotation increment increases over time, so they will not follow their target for long before they fly off-screen.

```
void CCastOrb::orbDelete(IAnimatedMeshSceneNode* &playerNode)
{
	if(now >= then+10000 && removeOrb == false)
	{
	  removeOrb = true;
	  orbNode->remove();
	  ps->remove();
//	  orbLight->remove();
	  orbSound->setIsPaused(true);
	  orbSound->drop();
	  orbSound = 0;
	}
	if(orbNode->getTransformedBoundingBox().intersectsWithBox(playerNode->getTransformedBoundingBox()) && removeOrb == false)
	{
	  removeOrb = true;
	  orbNode->remove();
	  ps->remove();
//	  orbLight->remove();
	  orbSound->setIsPaused(true);
	  orbSound->drop();
```

```
            orbSound = 0;
        }
}
```

There is a collision test where if the bounding box of the orb intersects with Akari's, then the orb is removed from the scene and removed from the orb vector.

CMainMenu Class

The CMainMenu class is loaded before the start of the game, and it displays the title sequence as well as the main menu. It contains its own loop to accommodate player input.

```
now = sceneNodes->getdevice()->getTimer()->getTime();
if(!playerControls->IsConnected())
            please_connect = sceneNodes->getguienv()->addImage(sceneNodes-
>getdriver()->getTexture("GUI Elements/Please_Connect.png"),
vector2d<irr::s32>(600,580));
if(!isStartPressed)
{
        if(now >= then+42*2)
            akari_text_1 = sceneNodes->getguienv()->addImage(sceneNodes-
>getdriver()->getTexture("GUI Elements/Clip_Logo/Akari_1_00001.png"),
vector2d<irr::s32>(-20,-50));
        if(now >= then+84*2)
            akari_text_2 = sceneNodes->getguienv()->addImage(sceneNodes-
>getdriver()->getTexture("GUI Elements/Clip_Logo/Akari_1_00002.png"),
vector2d<irr::s32>(-20,-50));
        if(now >= then+126*2)
            akari_text_3 = sceneNodes->getguienv()->addImage(sceneNodes-
>getdriver()->getTexture("GUI Elements/Clip_Logo/Akari_1_00003.png"),
vector2d<irr::s32>(-20,-50));
            if(now >= then+168*2)
```

Displaying each frame of the animation at regular intervals using a timer animates the title sequence.

```
if(playerControls->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_START)
{
        isStartPressed = true;
}
```

If start is pressed, a Boolean is changed and the main menu loads.

```
imageCity = sceneNodes->getguienv()->addImage(sceneNodes->getdriver()-
>getTexture("GUI Elements/City.bmp"), vector2d<irr::s32>(0,0));
cityTranslate = vector2d<irr::s32>(0,0);
city = rect<irr::s32>(0,0,2200,1300);
imageAkari = sceneNodes->getguienv()->addImage(sceneNodes->getdriver()-
>getTexture("GUI Elements/Akari.png"), vector2d<irr::s32>(550,0));
```

The main menu consists of .png files that Ivy created using Photoshop; these images are rendered onto the screen using the addImage method of the GUIENV class.

```
if (magnitudeLeft > LEFT_DEAD_ZONE)
{
	if(leftAnalogY >= 20000 && selectorNewGame && analogTilt == false)
	  analogTilt = true, selectorCredits = true;
	else if(leftAnalogY <= -20000 && selectorNewGame && analogTilt == false)
	  analogTilt = true, selectorLoadGame = true;
	if(leftAnalogY >= 20000 && selectorLoadGame && analogTilt == false)
	  analogTilt = true, selectorNewGame = true;
	else if(leftAnalogY <= -20000 && selectorLoadGame && analogTilt == false)
	  analogTilt = true, selectorSettings = true;
	if(leftAnalogY >= 20000 && selectorSettings && analogTilt == false)
	  analogTilt = true, selectorLoadGame = true;
	else if(leftAnalogY <= -20000 && selectorSettings && analogTilt == false)
	  analogTilt = true, selectorCredits = true;
	if(leftAnalogY >= 20000 && selectorCredits && analogTilt == false)
	  analogTilt = true, selectorSettings = true;
	else if(leftAnalogY <= -20000 && selectorCredits && analogTilt == false)
	  analogTilt = true, selectorNewGame = true;
}
```

The player can navigate through the main menu using the Xbox 360 controller by tilting the left thumb stick up and down.

```
if(playerControls->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_A)
{
	newGame = true;
	sceneNodes->getguienv()->clear();
	loading = sceneNodes->getguienv()->addImage(sceneNodes->getdriver()-
>getTexture("GUI Elements/Loading....png"), vector2d<irr::s32>(900,600));
}
```

When the player pushes the A button, the GUI is cleared from the scene and a loading image appears at the bottom right of the screen, to tell the player that the scene is loading.

```
CMainMenuInit();
while (sceneNodes->getdevice()->run() && !newGame)
{
	if (sceneNodes->getdevice()->isWindowActive())
	{
	  CMainMenuStartScreen();
	  CMainMenuMainMenu();
	  sceneNodes->getdriver()->beginScene(true,true,video::SColor(0,0,0,0));
	  sceneNodes->getsmgr()->drawAll();
	  sceneNodes->getguienv()->drawAll();
	  sceneNodes->getdriver()->endScene();

	  if(playerControls->GetState().Gamepad.wButtons &
XINPUT_GAMEPAD_BACK)
			sceneNodes->getdevice()->closeDevice();
	}
}
if(newGame)
{
	CMainMenuDeconstruct();
}
```

The main menu and title sequence loop is present inside of the CMainMenu class and is where all of the methods of this class are called. The loop breaks when the newGame Boolean is true, which happens when the player selects New Game from the main menu.

Main

The main function is where all of the classes and methods are declared and structured. Code without a main function is like a painting without a canvas.

```
CSceneNodes* sceneNodes = new CSceneNodes();
sceneNodes->CSceneNodesInit();
```

First, the sceneNodes object is created and its initialization method is called to create those important objects that are held within the CSceneNodes class.

```
CMainMenu* mainMenu = new CMainMenu(sceneNodes, soundEngine);
mainMenu = 0;
```

The main menu object is created next, this is where the main menu loop is initiated. Once the player has broken the loop by selecting New Game, the mainMenu object is given a value of 0 and the code continues.

```
ITimer* timer = sceneNodes->getdevice()->getTimer();
f32 dt;
u32 then = sceneNodes->getdevice()->getTimer()->getTime();
```

The variable dt is where the Delta Time value is stored. The value of dt varies depending on how fast the game is able run. This ensures that the games calculations scale to the machine it is run on, this is achieved by multiplying dt to any calculation that may be affected by a change in frame rate, allowing to game to run in real time rather than on a frame-by-frame basis.

```
CBulletPhysics* physicsEngine = new CBulletPhysics(sceneNodes);
physicsEngine->m_setGravity(-100.0);
```

The Bullet Physics Engine object is then created and the gravity of its objects are set.

```
// Bullet Objects
```

```
u32 i, k;
f32 randomBoxPileX, randomBoxPileZ;
for(i=0;i<10;i++)
{
        randomBoxPileX = sceneNodes->RandomFloat(-200.0f,200.0f);
        randomBoxPileZ = sceneNodes->RandomFloat(-200.0f,200.0f);
        for(k=0;k<2;k++)
        {
          physicsEngine-
>m_CreateBox(btVector3(randomBoxPileX,8.0f*k,randomBoxPileZ),
core::vector3df(8.0f,8.0f,8.0f),50.0f);
          physicsEngine-
>m_CreateBox(btVector3(randomBoxPileX+8.0f,8.0f*k,randomBoxPileZ),
core::vector3df(8.0f,8.0f,8.0f),50.0f);
          physicsEngine-
>m_CreateBox(btVector3(randomBoxPileX,8.0f*k,randomBoxPileZ+8.0f),
core::vector3df(8.0f,8.0f,8.0f),50.0f);
          physicsEngine-
>m_CreateBox(btVector3(randomBoxPileX+8.0f,8.0f*k,randomBoxPileZ+8.0f),
core::vector3df(8.0f,8.0f,8.0f),50.0f);
        }
}
```

Stacks of Bullet Physics boxes are created randomly within the 200x200 square boundaries of the stage.

```
// Floor
physicsEngine->m_CreateBox(btVector3(0.0f,-0.5f,0.0f),

core::vector3df(800.0f,0.5f,800.0f),0.0f);
```

These boxes need a floor to hold them up, so another box, that is flattened and stretched, is used as the floor. The last argument represents the mass of an object; an object with no mass will become static.

```
CMainCharacter* mainCharacter1 = new CMainCharacter();
mainCharacter1->CMainCharacterRender(sceneNodes, vector3df(0,0,0));
```

The main character is initialized next.

```
sceneNodes->getsmgr()->loadScene("Scenes/scene1.irr");
```

The scene that was created in CopperCube is initiated next.

```
ILightSceneNode* light1 = sceneNodes->getsmgr()->addLightSceneNode(0,
vector3df(-500,500,0), SColorf(1.0f,1.0f,1.0f), 1000.0f, 1 );
light1->enableCastShadow(true);
sceneNodes->getsmgr()->setAmbientLight(video::SColorf(0.8,0.8,0.8,0.0));
sceneNodes->getsmgr()->setShadowColor(video::SColor(150,0,0,0));
```

An Irrlicht light scene node is required to render shadows for the characters, its position, ability to cast shadows and brightness are set once it had been initialized.

```
u32 grunt_num = 2;
u32 runner_num = 20;
u32 caster_num = 0;
u32 enemies_dead = 0;
u32 enemy_num = grunt_num + runner_num + caster_num;
```

The number of enemies from each class is specified from some integer variables. The enemies_dead variable stores the value of the number of enemies that have been defeated. The enemy_num variable stores the total number of enemies in the scene.

```
vector<CRunner*> runners;
CRunner* runner;
for(i=0;i<runner_num;i++)
{
        runner = new CRunner();
        runner->CRunnerRender(sceneNodes, vector3df(-1000,-1000,-1000), i+1);
        runners.push_back(runner);
}
```

Loops are used to render multiple enemies at once; the number of times the code in the loop is executed is specified by the value stored in class num variables. For each loop iteration, a new enemy object is created, that enemy's render/initialization method is called, and the enemy object is pushed into a vector.

```
soundEngine->getSoundEngine()->play2D("Sound/Twister.mp3",true);
soundEngine->getSoundEngine()->setSoundVolume(0.5f);
```

A method from the soundEngine class is called to play the sound file that can be found in the project directory, and another method from the same class is called to lower its volume.

```
while(sceneNodes->getdevice()->run())
{
```

The main game loop is where the update methods are called. This loop is repeated indefinitely, until the player chooses to end the game.

```
//--------------- Inital Spawns ------------//
spawnNow = sceneNodes->getdevice()->getTimer()->getTime();
if(initSpawn == false && spawnNow >= spawnThen+50000)
{
        // Grunts
          for(vector<CGrunt*>::iterator currentGrunt = grunts.begin(); currentGrunt !=
        grunts.end(); currentGrunt++)
          (*currentGrunt)->CGruntSpawn(spawnPositionX,spawnPositionZ);
```

The enemies do not spawn as soon as they are rendered. When they are first initialized, they are placed in the graveyard and their spawn methods are called using a timer.

```
now = sceneNodes->getdevice()->getTimer()->getTime();
frameDeltaTime = (f32)(now - then) / 1000.f; // Time in seconds
then = now;
```

The Delta Time is calculated by finding subtracting the past time from the current time, and then dividing by 1000.0.

```
physicsEngine->m_UpdatePhysics(sceneNodes, frameDeltaTime, grunts, runners,
casters);
```

The physics update method is called next.

```
// Main Character Update
mainCharacter1->CMainCharacterTranslate(sceneNodes, physicsEngine,
soundEngine, frameDeltaTime, grunts, runners, casters);// Runs the translation
Method
mainCharacter1->CMainCharacterCamera(sceneNodes);
```

The main characters translation and camera method are called next. The enemy vectors are passed as method arguments, these contain the updated positions of every enemy.

```
// Misc Objects //
enemies_dead = 0;
// Grunts
for(vector<CGrunt*>::iterator currentGrunt = grunts.begin(); currentGrunt !=
grunts.end(); currentGrunt++)
{
        (*currentGrunt)->CGruntTranslate(sceneNodes, physicsEngine,
mainCharacter1,
          vector3df(mainCharacter1->getTranslateX(),mainCharacter1-
>getTranslateY(),mainCharacter1->getTranslateZ()),
          grunts,
          runners,
          casters,
          frameDeltaTime); // main character location
        (*currentGrunt)->getGruntNode()->setPosition(vector3df((*currentGrunt)-
>getTranslateX(),(*currentGrunt)->getTranslateY(),(*currentGrunt)-
>getTranslateZ())); // Translates the grunts position
        (*currentGrunt)->getGruntNode()->setRotation(vector3df(0,(*currentGrunt)-
>getRotationY()*-1+90,0)); // Rotates the grunt
        if((*currentGrunt)->CGruntDead() == true)
        {
          enemies_dead++;
        }
}
```

Each of the enemies update methods are called next. Their vectors call the methods of all the objects they contain by using for loops and iterators. The updated vectors of the other enemy classes, the updated player position and current delta time are called into this method as arguments to be used when calculating the enemies AI. If the CGruntDead Boolean method returns a true value, then the value of enemies_dead is increased by one. The value of enemies_dead returns to 0 before the number of dead enemies is counted.

```
if(enemies_dead >= enemy_num)
{
	enemies_dead = 0; // reset the dead enemy counter.
	spawnPositionX = sceneNodes->RandomFloat(-200.0f,200.0f);
	spawnPositionZ = sceneNodes->RandomFloat(-200.0f,200.0f);
	// Grunts
	for(vector<CGrunt*>::iterator currentGrunt = grunts.begin(); currentGrunt != 
grunts.end(); currentGrunt++)
	{
	  (*currentGrunt)->CGruntSpawn(spawnPositionX,spawnPositionZ);
	}
	// Runners
	for(vector<CRunner*>::iterator currentRunner = runners.begin(); currentRunner 
!= runners.end(); currentRunner++)
	{
	  (*currentRunner)->CRunnerSpawn(spawnPositionX,spawnPositionZ);
	}
	// Casters
	for(vector<CCaster*>::iterator currentCaster = casters.begin(); currentCaster != 
casters.end(); currentCaster++)
	{
	  (*currentCaster)->CCasterSpawn();
	}
```

If the number of dead enemies is equal to the number of enemies in the scene, then the spawn method is called for each of the enemy classes, returning them to a random position on the stage.

```
// Displays the FPS and Title
u32 fps = sceneNodes->getdriver()->getFPS();
if (lastFPS != fps)
{
	core::stringw str = L"Akari[";
	str += sceneNodes->getdriver()->getName();
	str += "] FPS: ";
	str += fps;
	sceneNodes->getdevice()->setWindowCaption(str.c_str());
	lastFPS = fps;
}
```

The updated FPS (frames-per-second) is then displayed as the game's window caption.

```
sceneNodes->getdriver()->beginScene(true, true, SColor(255,100,101,140));
sceneNodes->getsmgr()->drawAll();
sceneNodes->getguienv()->drawAll();
sceneNodes->getdriver()->endScene();
```

The scene node methods that draw the graphics onto the screen are called at the bottom of the code.

```
if(mainCharacter1->getPlayerOne()->GetState().Gamepad.wButtons &
XINPUT_GAMEPAD_BACK)
        sceneNodes->getdevice()->closeDevice();
```

Should the player want to close the game demo, the back button will do this for them.

```
sceneNodes->getdevice()->drop();
soundEngine->getSoundEngine()->drop();
physicsEngine->m_DestroyPhysicsWorld();
return 0;
```

If the loop is broken, then Irrlicht device, sound engine and bullet physics world are dropped.

Development Blog

A development blog (http://russkentish.wordpress.com/) was started to document the development progress of this game. It was updated once a week and explains the progress that was made and includes screenshots and videos of gameplay.

Alpha Footage

Akari Gameplay – v1.0
Link
Akari Gameplay – v2.1
Link
Akari Main Menu and Title Sequence
Link

Testing and Evaluation

Testing the game was a constant battle from the start of development. Every element that was implemented had to be tested in some way, and previously implemented elements had to be revised, even removed, when newer elements were introduced.

Functionality Testing

Game element functionality was tested and recorded below. It states the named of the functional requirement of an object and whether it passed or failed the test. If the test was failed, a brief explanation of why can be found below it.

Main Menu	
Requirement	**Passed/Failed**
Easy to understand/navigate	**Passed**
Start new game/continue	**Failed**
Info: *While the player has the ability to start a new game, the player cannot navigate down to continue from a saved game.*	

Main Character	
Requirement	**Passed/Failed**
Translated with Xbox controller	**Passed**

Rotated with Xbox Controller	**Passed**
Can shoot orbs	**Passed**
Has ability to jump	**Failed**
Info: *Gravity was implemented and the character attempts to jump when the left bumper is pressed, but the delta time for the calculation is bugged, so the character's jump height changes irratically.*	
Is animated	**Passed**
Can interact with in-game elements	**Passed**
Its geometry does not clip with other in-game objects	**Partially**
Info: The code does its best not to stack in-game objects on one another, but the characters animation cycles render it impossible to avoid all clipping.	

Isometric Camera Perspective and Gameplay	
Requirements	**Passed/Failed**
Keep the player in view	**Passed**
Show enemies on screen	**Passed**
Objects must not obstruct view	**Passed**

Third-person Camera Perspective and Gameplay	
Requirements	**Passed/Failed**
It must rotate smoothly	**Passed**
It must follow the player around	**Passed**
It must keep the player in view	**Passed**

Grunt Enemy Type	
Requirements	**Passed/Failed**
Aware of players position	**Passed**

Must be well balanced	**Passed**
Must be animated	**Passed**
Separation Behaviour	**Passed**
Can be defeated by player	**Passed**
They must respawn	**Passed**
They must follow and attack player	**Passed**

Runner Enemy Type	
Requirements	**Passed/Failed**
Must be ware of player position	**Passed**
Must be animated	**Passed**
Separation behaviour	**Passed**
An be defeated by player	**Passed**
Can respawn	**Passed**
Must follow and attack player	**Passed**

Caster Enemy Type	
Requirements	**Passed/Failed**
Must be aware of player position	**Passed**
Must be well balanced	**Passed**
Must be animated	**Passed**
Separation behaviour	**Passed**
Can be defeated by player	**Passed**
Must respawn	**Passed**
Must follow player	**Passed**

Can fire orbs at player	**Partially**
Info: *It is true that the caster can fire orbs at the player, but there was a problem with removing orbs from the scene and looping through the vectors and it causes the game to crash.*	

Physics	
Requirements	**Passed/Failed**
Must react when hit by player	**Passed**
Objects must not fall through floor	**Passed**

Environment	
Requirements	**Passed/Failed**
Must keep player within boundaries	**Passed**
Must keep enemies within boundaries	**Passed**
Objects must not obstruct camera	**Passed**

Performance Testing

Rendering in real-time is very hardware intensive so developers should monitor the elements and calculations that are executed in their games to keep the frame rate high. The game was tested on a number of machines to record the frames that were rendered per second. The optimal frame rate is roughly 30 frames per second.

Computer Specs	**Frames Per Second**
2.66 GHz Intel Dual-Core i7 4 GB DDR3	**30-50**
3.3 GHz Intel Quad-Core i5 4 GB DDR3	**80-100**
2.6 GHz Intel Celeron Dual-Core 2 GB DDR2	**20-40**

3.2 GHz Intel Core i3 2 GB DDR2	**30-40**

Conclusion

Many aspects of this project could have been implemented differently, if development of this title started from the beginning, the code would have taken advantage of inheritance a lot more, and many of the methods inside the main character class would become their own classes.

Development of this game will continue for the next few months as Ivy will continue to create assets for it for her final year project.

Regrettably, platforming and exploration scenarios could not be implemented, in time for the deadline, to demonstrate the practicality of the third-person gameplay perspective.

The failed exportation of animated meshes has been the most disappoint part of this project.

This has been a very successful project; it represents the encapsulated knowledge gained from studying Games Technology at Kingston University. The result is the creation of an innovative duel-genre video game prototype with simple and addicting gameplay. Many game development methods were researched while developing this project, expanding the horizons of all who were involved.

References

[i] Irrlicht3D, 2012. Irrlicht Engine [Online] Available at: http://irrlicht.sourceforge.net/ [20th April 2012]
[ii] Bullet Physics, 2012. Game Physics Simulation [Online] Available at: http://bulletphysics.org/wordpress/ [13th April 2012]
[iii] Microsoft, 2012. XInput Driver [Online] Available at: http://www.microsoft.com/download/en/details.aspx?id=4190#overview [13th April 2012]
[iv] Ambiera, 2012. irrKlang [Online] Available at: http://www.ambiera.com/irrklang/ [13th April 2012]
[v] Gamasutra, 2012. The State of Indie Gaming [Online] Available at: http://www.gamasutra.com/view/feature/3640/the_state_of_indie_gaming.ph [13th April 2012]
[vi] techradar, 2012. Is indie gaming the future? [Online] Available at: http://www.techradar.com/news/gaming/is-indie-gaming-the-future--716500 [13th April 2012]
[vii] Infinite Blade, 2012. Infinite Blade II [Online] Available at: http://infinitybladegame.com/ [13th April 2012]
[viii] Video Gamer, 2012. Angry Birds sales soar over 200 million [Online] Available at: http://www.videogamer.com/iphone/angry_birds/news/angry_birds_sales_soar_over_200_million.html [13th April 2012]
[ix] Colossal Games, 2012. Colossal Games [Online] Available at: http://www.colossalgames.eu/ [13th April 2012]
[x] Primerlabs, 2012. CODE HERO [Online] Available at: http://primerlabs.com/codehero [30th May 2012]
[xi] South Thames College, 2012. Games Development [Online] Available at: http://www.south-thames.ac.uk/CO1350-13a-13/Games_Development [30th May 2012]
[xii] Splash Damage, 2012, Dave Johnston [Online] Available at: http://www.splashdamage.com/node/67 [13th April 2012]
[xiii] Schell, J. (2008). The Art of Game Design: A Book of Lenses. Burlington: Elsevier Inc.
[xiv] Schell, J. (2008). The Art of Game Design: A Book of Lenses. Burlington: Elsevier Inc.
[xv] Schell, J. (2008). The Art of Game Design: A Book of Lenses. Burlington: Elsevier Inc.
[xvi] Schell, J. (2008). The Art of Game Design: A Book of Lenses. Burlington: Elsevier Inc.
[xvii] Schell, J. (2008). The Art of Game Design: A Book of Lenses. Burlington: Elsevier Inc.
[xviii] Schell, J. (2008). The Art of Game Design: A Book of Lenses. Burlington: Elsevier Inc.
[xix] YouTube, 2012, Extra Credits: So You Want To Be a Game Designer [Online] Available at: http://www.youtube.com/watch?v=zQvWMdWhFCc [30th May 2012]
[xx] Blender. Blender.org - Home [Online] Available at: http://www.blender.org/ [30th May 2012]
[xxi] Gimp, GIMP – The GNU Image Manipulation Program [Online] Available at: http://www.gimp.org/ [30th May 2012]
[xxii] Ogre3D, 2012, OGRE – Open Source 3D Graphics Engine [Online] Available at: http://www.ogre3d.org/
[xxiii] Irrlicht, 2012, Irrlicht Engine – A free open source 3D engine [Online] Available at: http://irrlicht.sourceforge.net/

xxiv Bullet Physics Library, 2012, Game Physics Simulation [Online] Available at: http://bulletphysics.org/wordpress/
xxv Irrlicht, 2012, Irrlicht Engine – A free open source 3D engine [Online] Available at: http://irrlicht.sourceforge.net/
xxvi Sithu-Kyaw, A (2011). Irrlicht 1.7 RealTime 3D Engine: Beginner's Guide. Birmingham: Packt Publishing Ltd.
xxvii Ambiera, 2012, CopperCube – a 3D editor for Flash Stage3D and WebGL, Mac OS X and Windows [Online] Available at: http://www.ambiera.com/coppercube/index.html
xxviii Ogre3D, 2012, OGRE – Open Source 3D Graphics Engine [Online] Available at: http://www.ogre3d.org/
xxix VGChartz, 2012, Video Game Charts, Game Sales, Top Sellers, Game Data [Online] Available at: http://www.vgchartz.com/
xxx Wikipedia, 2012, Geometry Wars [Online] Available at: http://en.wikipedia.org/wiki/Geometry_Wars
xxxi Microsoft, 2012, Xbox.com [Online] Available at: http://marketplace.xbox.com/en-US/Games/XboxArcadeGames
xxxii IGN, 2012, The Legend of Zelda: Ocarina of Time [Online] Available at: http://uk.ign64.ign.com/objects/000/000437.html
xxxiii SEGA, 2012, SEGA :: GAMES :: Sonic Generations [Online] Available at: http://www.sega.co.uk/games/sonic-generations?t=EnglishUK
xxxiv Gaming and graphics: the console and PC: separated at birth? Computer graphics [0097-8930] Rouse, Richard yr:2001 vol:35 iss:2 pg:5 -9
xxxv Haller, M (2004). Non-photorealistic rendering techniques for motion in computer games. *Computers in entertainment,* Volume: 2, Issue: 4, (Pages) 11-11
xxxvi Haller, M (2004). Non-photorealistic rendering techniques for motion in computer games. *Computers in entertainment,* Volume: 2, Issue: 4, (Pages) 11-11
xxxvii Foss, S (2004).2.5D modeling, inversion and angle migration in anisotropic elastic media. *Geophysical prospecting,* Volume: 52, Issue:1, (Pages) 65-84
xxxviii Autodesk, 2012, Maya – 3D Animation Software [Online] Available at: http://usa.autodesk.com/maya/
xxxix Irrlicht, 2012, Features – Irrlicht Engine – A free open source 3D engine [Online] Available at: http://irrlicht.sourceforge.net/features/
xl Gamedev.net, 2012, Working with the DirectX .X File Format and Animation [Online] Available at: http://www.gamedev.net/page/resources/_/technical/game-programming/working-with-the-directx-x-file-format-and-animation-in-directx-90-r2079
xli Irrlicht, 2012, .x file exporter for Maya [Online] Available at: http://irrlicht.sourceforge.net/forum/viewtopic.php?t=11666
xlii Irrlicht, 2012, Exporters [Online] Available at: http://www.irrlicht3d.org/wiki/index.php?n=Main.Exporters
xliii Unity, 2012, Physics [Online] Available at: http://unity3d.com/support/documentation/Manual/Physics
xliv Freud, S. (1919/2003). The uncanny [das unheimliche] (D. McLintock, Trans.). New York: Penguin.
xlv Bullet Physics, 2012, Game Physics Simulation[Online] Available at: http://bulletphysics.org/wordpress/
xlvi Sourceforge, 2012, irrBullet [Online] Available at: http://sourceforge.net/projects/irrbullet/
xlvii Irrlicht, 2012, Getting Started With Bullet [Online] Available at: http://www.irrlicht3d.org/wiki/index.php?n=Main.GettingStartedWithBullet
xlviii Bullet Physics, 2012, Creating a project from scratch [Online] Available at: http://bulletphysics.org/mediawiki-1.5.8/index.php/Creating_a_project_from_scratch
xlix Bullet Physics, 2012, Physics Simulation [Online] Available at: http://bulletphysics.org/mediawiki-1.5.8/index.php/Main_Page
l Pixelux, 2012, Pixelux Entertainment [Online] Available at: http://www.pixelux.com/DMMengine.html
li Ambiera, 2012, irrKlang [Online] Available at: http://www.ambiera.com/irrklang/
lii Rozali, W. Hamid, S. Sabri, M. (2007). Video Games: Issues and Problems.
liii The Sun, 2012, Man stabs wife for PlayStation nagging [Online] Available at: http://www.thesun.co.uk/sol/homepage/news/4276181/Man-stabs-wife-for-PlayStation-nagging.html
liv Gentile, D. Lynch, P. Ruh Linder,,J. Walsh, D. (2004). The Effects of Violent Video Game Habits on Adolescent Hostility, Aggressive Behaviors, and School Performance. *Journal of Adolescence*, Volume 27, Issue 1, (Pages): 5-22.
lv Microsoft, 2012, Visual Studio 2010 [Online] Available at: http://www.microsoft.com/visualstudio/en-us/products/2010-editions
lvi Wikipedia, 2012, Geometry Wars [Online] Available at: http://en.wikipedia.org/wiki/Geometry_Wars
lvii Microsoft, 2012, Getting Started with XInput [Online] Available at: http://msdn.microsoft.com/en-us/library/windows/desktop/ee417001(v=vs.85).aspx
lviii Microsoft, 2012, Wireless Gaming Receive for Windows [Online] Available at: http://www.microsoft.com/games/en-US/Hardware/Controllers/Pages/XboxWirelessGamingReceiverforWindows.aspx/
lix Autodesk, 2012, Maya [Online] Available at: http://usa.autodesk.com/maya/

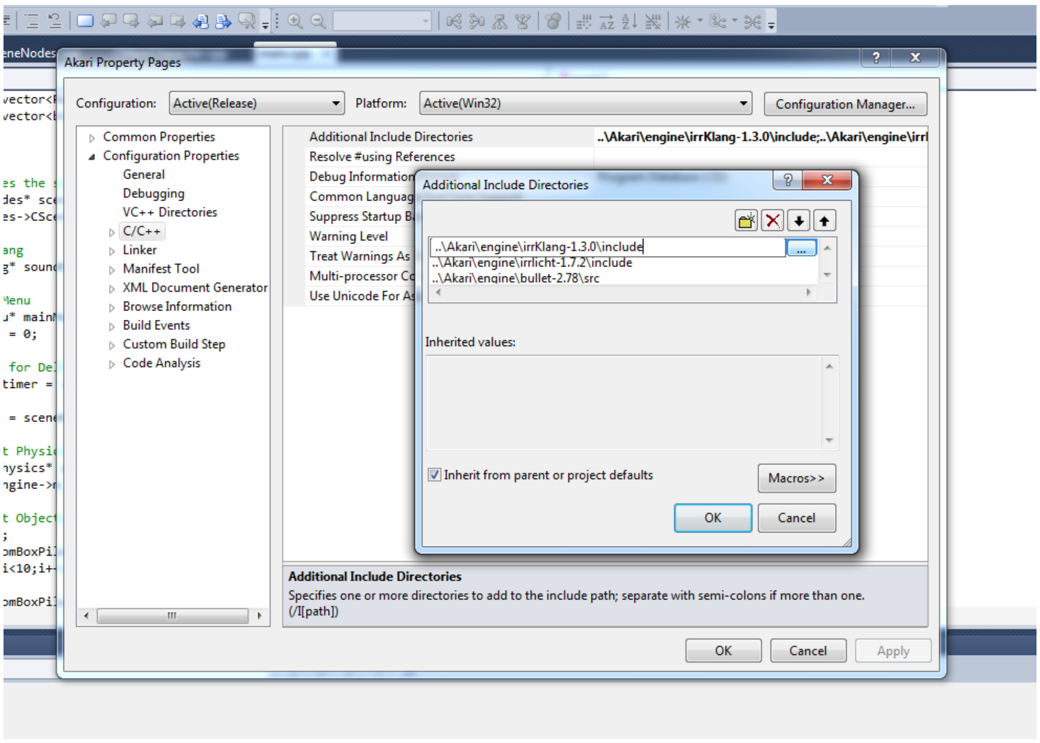
[lx] Adobe, 2012, 3D design | Adobe Photoshop CS6 [Online] Available at: http://www.adobe.com/products/photoshopextended.html
[lxi] Animazoo, 2012, Animazoo Motion Capture Systems and Technology [Online] Available at: http://www.animazoo.com/motion-capture-systems/animazoo-operating-software/
[lxii] Autodesk, 2012, MotionBuilder [Online] Available at: http://usa.autodesk.com/adsk/servlet/pc/index?id=13581855&siteID=123112
[lxiii] Mocappys, 2012, Mapping Optical Data using MotionBUilder Actor[Online] Available at: http://mocappys.com/?p=14
[lxiv] Animazoo, 2012, Animazoo Motion Capture Systems and Technology [Online] Available at: http://www.animazoo.com/
[lxv] Scriptspot, 2012, ghostTown Lite [Online] Available at: http://www.scriptspot.com/3ds-max/scripts/ghosttown-lite
[lxvi] Irrlicht, 2012, Getting Started With Bullet [Online] Available at: http://www.irrlicht3d.org/wiki/index.php?n=Main.GettingStartedWithBullet
[lxvii] Irrlicht, 2012, Irrlicht Engine wiki [Online] Available at: http://www.irrlicht3d.org/wiki/index.php
[lxviii] Irrlicht3D, 2012. Free Camera Rotation Around Target [Online] Available at: http://irrlicht.sourceforge.net/forum/viewtopic.php?t=34911 [29th April 2012]

Appendices

Appendix 1 – Linking Libraries

Once your libraries have been chosen and compiled, they should be placed in a suitable location, preferably in your game directory.

The libraries can then be linked to your project by specifying the URL in the Include Directories box in the project properties.


Akari Property Pages

Configuration: Active(Release) Platform: Active(Win32)

Additional Include Directories: ..\Akari\engine\irrKlang-1.3.0\include;..\Akari\engine\irrl

Additional Include Directories

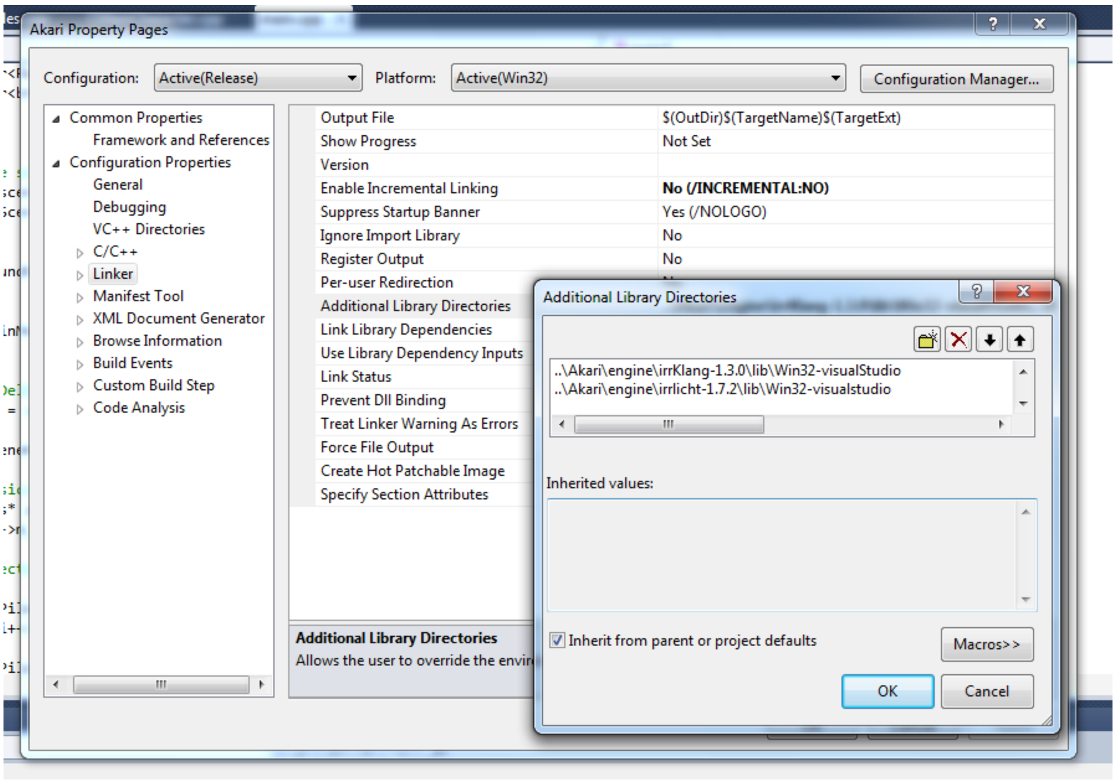
```
..\Akari\engine\irrKlang-1.3.0\include
..\Akari\engine\irrlicht-1.7.2\include
..\Akari\engine\bullet-2.78\src
```

Inherited values:

[✓] Inherit from parent or project defaults

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

The links should direct the project to the libraries header files.


Akari Property Pages

Configuration: Active(Release) Platform: Active(Win32) Configuration Manager...

- Common Properties
  - Framework and References
- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step
  - Code Analysis

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	**No (/INCREMENTAL:NO)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	
Additional Library Directories	
Link Library Dependencies	
Use Library Dependency Inputs	
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	
Create Hot Patchable Image	
Specify Section Attributes	

Additional Library Directories
Allows the user to override the envir

Additional Library Directories

```
..\Akari\engine\irrKlang-1.3.0\lib\Win32-visualStudio
..\Akari\engine\irrlicht-1.7.2\lib\Win32-visualstudio
```

Inherited values:

☑ Inherit from parent or project defaults

Macros>> OK Cancel

The links to the .lib files should be specified under Library Directories.

The bullet libraries are contained within their own projects. Any supporting projects can be included in the Solution by adding them via the solution explorer, but your game's project should be set as the start up project.