

Game Document



Super Power Surge Spike By Team Booty Rock

JOSH BENNETT

JOSHUA.SOFTWARE@GMAIL.COM

PHILIP FOX

GHSTEO@GMAIL.COM

TYLER KEENAN

SINBOMB@FULLSAIL.EDU

JUSTIN MORGAN

DIVIDE.BY.ZERO.FTW@GMAIL.COM

TIM TURCICH

POPPYSPY@GMAIL.COM

Table of Contents

GAME CHARTER.....	1
VISION STATEMENT	1
MEETING SCHEDULE.....	1
HOURS WORKED PER WEEK.....	1
WHEN THINGS GO WRONG.....	1
DECISION-MAKING PROCESS.....	2
RULES OF CONDUCT.....	2
TEAM ROLES	3
EXECUTIVE SUMMARY	4
HIGH CONCEPT	4
LOCALE	4
GENRE	4
BASIC CONTROLS.....	5
GAME GOAL	6
TARGET PLATFORM.....	6
MARKETING & TARGET AUDIENCE.....	6
GAME WALKTHROUGH/OVERVIEW	7
KEY FEATURES	7
COMPARATIVE PRODUCTS	8
TREATMENT.....	12
DUST JACKET STORY	12
GAME STORY.....	12
CHARACTERS	13
<i>Alice –Player character.....</i>	<i>13</i>
<i>Gambits (enemies).....</i>	<i>15</i>
WEAPONS	16
<i>Elemental Node Power</i>	<i>16</i>
POWER-UPS.....	19
<i>Power Nodes.....</i>	<i>19</i>
<i>Shield</i>	<i>20</i>
LEVELS AND MAPS.....	21
<i>The MotherBoard.....</i>	<i>21</i>
ART AND PRODUCTION DESIGN	22
STORYBOARDS AND SAMPLE ART	23
GOAL.....	26
INTERFACE.....	27
IN GAME KEYBOARD CONTROLS	27
IN GAME MOUSE CONTROLS	27
CAMERA.....	27

MENU CONTROLS.....	28
MENU TRANSITIONS	28
MENU INPUT	28
MAIN MENU.....	29
OPTIONS MENU.....	30
EXTRAS MENU	32
CREDITS SCREEN	33
PAUSE MENU.....	34
LOADING SCREEN.....	35
HUD.....	36
INTERACTIVE RHYTHM	37
HOW THE PLAYER MARKS PROGRESS	37
DETAILED DESIGN BREAKDOWN	38
FRONT END FLOW CHART	38
GAME FLOW CHART	39
GLOSSARY OF TERMS.....	40
CHARACTERS	44
VIRUS ALICE – MAIN AVATAR.....	44
FIRE GAMBIT.....	47
ICE GAMBIT	49
WIND GAMBIT	50
ELECTRIC GAMBIT	52
BOSS GAMBIT.....	54
ENEMY RESISTANCE/WEAKNESS AGAINST ATTACKS MODEL.....	55
ATTACKS SYSTEM AND WEAPONS.....	56
FIRE ELEMENTAL ATTACKS.....	56
ICE ELEMENTAL ATTACKS.....	58
WIND ELEMENTAL ATTACKS.....	60
ELECTRIC ELEMENTAL ATTACKS.....	62
POWER-UPS.....	64
SHIELD.....	64
LEVELS AND MAPS	65
COMMON LEVEL TRAITS	65
NEUTRAL HUB.....	67
MEMORY LEVEL	70
POWER GRID.....	73
HARD DRIVE LEVEL.....	76
FINAL BOSS	79
COMBAT SYSTEM.....	82
SCORE MULTIPLIER	84
OVERVIEW.....	84
MECHANICS	84
NODE SYSTEM.....	85
OVERVIEW.....	85

MECHANICS	85
GAME LOGIC, ALGORITHMS, AND RULES	86
INTERACTION COMPONENT MATRIX	86
KEY GAME ALGORITHMS	86
FAQ.....	88
REFERENCE OF KEY ELEMENTS	90
SCORING	90
WINNING/LOSING.....	90
TRANSITIONS	90
REWARDS.....	90
ART AND PRODUCTION DESIGN	91
3D ART & ANIMATION DELIVERABLES	91
2D ART (HUD/MENU/PARTICLE/TEXTURES) DELIVERABLES	93
SOUND EFFECTS DELIVERABLES.....	96
MUSIC DELIVERABLES.....	99
OVERVIEW	101
CODING STANDARDS	102
NAMING STANDARDS.....	102
PREFIX CONVENTION.....	102
STRUCTURES	103
CLASSES	104
RELEVANT FUNCTION NAMES.....	104
MACROS AND CONSTANTS.....	105
SUMMARY OF NAMING CONVENTION	106
COMMENTING	107
DATA ALIGNMENT	108
CODING GUIDELINES.....	108
DEVELOPMENT ENVIRONMENT	110
TIMING SPECIFICATIONS	111
SYSTEM ARCHITECTURE.....	112
DESCRIPTION	112
MODULE FEATURE BREAKDOWN.....	126
TIMER.....	126
INPUT	128
PLAYER SYSTEM.....	131
ACHIEVEMENTS	133
EVENTDISPATCHER	136
ASSETMANAGER	140
STATEMACHINE.....	146
ANIMATIONSYSTEM	149
OBJECTMANAGER.....	153
RENDERER	166
AI SYSTEM	176
FXMANAGER	185

MILESTONE DELIVERABLES.....	192
PoC.....	192
FEATUREFRAG1	192
FEATUREFRAG2	193
ALPHA.....	193
BETA	194
GOLD	195
ARCHIVE.....	196
APPENDIX A.....	197
MEMORY MAP.....	197
INTEGRATION PLAN	198
TESTING PLANS.....	199
APPENDIX B.....	201
GAME FOLDER HIERARCHY	201

Publishing Document

Game Charter

Vision Statement

Our goal as a team is to create a game that will garner the respect of the industry professionals enough to land us a job offer. We will succeed in doing this by making sure our team works well with each other as well as making sure each member uses their knowledge and experience to the best of their ability. We aim to ensure that the team works well together by making sure our documentation is functional and concise. We hope to accomplish this by using consistent formatting and also keeping reliable references throughout the lifespan of the project. We also want to keep lines of communication open between every member of our team and studio. We aim to accomplish this by keeping everyone in the team and studio informed whenever any issue involving the team or game comes up. We expect each member to understand the design we develop as well as use the technology we create in the manner it was intended.

Meeting Schedule

We will have four to five meetings per week. Meetings will take place at Full Sail in Building 3B. Meetings will start out with a review from the previous meeting and then be followed by an overview of the day's agenda. Meetings should be capped at five hours maximum and should not take longer. Each member is expected to stay efficient and fully engaged during the entirety of team meetings.

Hours Worked per Week

Team members will be expected to work at minimum forty hours but are encouraged to work more than that. Team members are not expected to work past sixty hours. The core hours the team will be working is from 11 am to 5 pm.

When Things Go Wrong

When an emergency comes up, other team members and our EP will be informed of the emergency by either Email or by Telephone. If a team member is the only person who is informed about the emergency then that team member is expected to inform the remaining team members about the incident. All members are expected to keep all lines of communication open at all times in case an emergency arises. To avoid any emergency scenarios, all milestones and turn-ins should be accompanied by a confirmation email sent to all members of the group for peace of mind.

Decision-Making Process

The initial decision making process will be an open forum discussion about the current topic or problem. No decision should go past the maximum of fifteen minutes of time. If a decision is not made within the fifteen minute time limit then we will bring the decision to a vote, if there is still no resolution out of the vote then the appropriate lead pertaining to the topic will make the final decision. A note of any decision pertaining to the game development process should be made on Google Groups or via email that can be referenced in the future in case a misunderstanding occurs. In this way we hope to keep everyone on the same page with the development of the game and eliminate any roadblocks to productivity. This will operate under a best-faith policy. Accidents happen and people forget, but a consistent disregard for the policy may be cause for disciplinary actions.

Rules of Conduct

Team members are expected to always be on time for class and team meetings. Team members will be expected not to show up under the influence of any illegal substances. Team members are expected to be respectful at all times to team members, peers, and staff. Additionally, team members must be contributing to and working on the project while a meeting is in session. No side project work, coding or otherwise, should ever be taking place during meeting or core hours, or ever interfere with the progress of the game. It is not the responsibility of other members of the team to inform you of deadlines, meetings, or milestones. If nothing has been scheduled for a day, this does not necessarily mean that no work is to be done that day. In this case, it is up to the individual to establish communication with others in the team to possibly confirm or deny and work responsibilities. Failure to make contact, unless emergencies arise or a day of relaxation has been arranged prior, will be considered an infraction of the rules. Disciplinary procedures may be brought against a team member when a multiplicity of team members deigns it to be necessary or if it can be shown that disregard for the stated rules has occurred.

If the group in anyway feels that a member of the team is breaking any of these rules then there will be disciplinary actions. The first action that will be taken against a member violating the rules is a verbal warning to let the member know that they are breaking a rule. The second action will be a written warning to the member that they are still violating the rules of the team. This written warning will also serve as a paper trail for the team as evidence against the team member. The final action against a team member is to bring up the violations to our EP and also the current course director. Any number of these steps can be bypassed depending on the severity of the infraction. It is at the discretion of the team as to whether a bypassing of any step of the disciplinary hierarchy is warranted.

Team Roles

Administrative Roles

PROJECT LEAD	TIM
ASSET LEAD	PHILIP
TECHNICAL LEAD	JUSTIN
GAMEPLAY LEAD	JOSH
DESIGN LEAD	TYLER

Technical Roles

INTERFACE	PHILIP
AI	JOSH – LEAD
	TIM
RENDERING	JUSTIN – LEAD
	TYLER
INPUT AND CONTROLS	PHILIP
AUDIO	TYLER
PARTICLES AND EFFECTS	TYLER – LEAD
	TIM
ANIMATION	JUSTIN
GAME PHYSICS	TIM
GAMEPLAY	TIM – LEAD
	PHILIP
ARCHITECTURE	JOSH – LEAD
	JUSTIN

Executive Summary

High Concept

Strategic Elemental Action

Locale

This game takes place in the internals of a computer. This game also takes place in a futuristic time.

Genre

This game is a 3D third person action platformer. The theme of this game is a Tron-like futuristic setting.

Basic Controls



Keyboard Controls

W	MOVE FORWARD
A	STRAFE LEFT
S	MOVE BACKWARD
D	STRAFE RIGHT
Q	CHARGE PRIMARY ABILITY
E	CHARGE SECONDARY ABILITY
LEFT SHIFT	SWAP PRIMARY ABILITY WITH SECONDARY ABILITY
SPACE	JUMP
ESC	PAUSE MENU

Mouse Controls

LEFT MOUSE BUTTON	FIRE PRIMARY ABILITY
RIGHT MOUSE BUTTON	FIRE PRIMARY ABILITY WITH YOU SECOND ABILITIES PATTERN
MOUSE MOVEMENT LEFT/RIGHT	ROTATES THE CAMERA AROUND AVATAR AND POINTS AVATARS AIM AT THE CAMERAS LOOK DIRECTION

Camera perspective will be a third person perspective set behind the player.

Game Goal

The goal of the game is to battle through the system and destroy all the enemies on the way to the end boss. The player is expected to defeat the end game boss to win the game. On the way to the goal the player will be expected to advance from node to node on their way to the end game goal.

Target Platform

OPERATING SYSTEM	WINDOWS XP
HARD DRIVE SPACE	650 MB
MEMORY	128 MB
GRAPHICS CARD	ATI 9800
CPU:	2000 MHZ
SHADER MODEL	2.0

Marketing & Target Audience

Our target audience is males fifteen to twenty-five who are fans of fast action games such as:

IKARUGA
CONTRA
MERC
GEOMETRY WARS
PHANTOM DUST

Game Walkthrough/Overview

You start in the hub level of our game there are four nodes placed through the area that allow you to test out abilities and get powered up for the level before you head in. As you enter the first level and you have enemies homing in on you. As you attack them you start to run low on power and start to panic. When you start to run away from the onslaught of enemies you notice a power node nearby, so you run towards the power node while evading the incoming enemies. When you get to the power node you charge up your primary weapon and then turn around to take out the first wave of enemies you encountered. You notice another wave of enemies rushing towards your position; you fire your primary weapon at that set of enemies but notice that the primary weapon you are using is just making the enemies more powerful. You notice a different colored node nearby so you make your way towards that node. You make it to the power node and charge up your secondary power. You swap your primary ability with secondary ability which is a different damage type. You now end up using that new damage type to take out the previous horde of enemies. You now notice two waves coming at you from different directions; you also notice there is a series of platforms up above you. You decide to give yourself some more room by jumping up on the platform and separating yourself from the two waves of enemies. Thinking that you have outsmarted the enemies, you are surprised when one of the waves of enemies is now following in your actions and jumping up on the same set of platforms. You decide to turn around and take out that wave of enemies. With only a little bit of ammo remaining you drop back down and take out the last set of enemies. You hear victory music and that gives you the cue that you have finished the level. You now proceed to the exit to finish the level.

Key Features

GENERAL FEATURES	MOVE
	JUMP
	ATTACK
	GAME MODES
PLAYER UPGRADES	POWER NODES
	SHIELD UPGRADE
GAME PLAY	LEVELS
	ENEMIES

Comparative Products

Phantom Dust



POWER UP COMBINATIONS

THIRD PERSON CAMERA SYSTEM

How this stacks up

Super Power Surge Spike will use a comparable camera and movement mechanics are the third person view and the strafing and jumping movement. Powers in Phantom Dust added to the player's arsenal to improve attacks against enemies as they also will in this game except we want to give the player greater access to the full arsenal for most of the playable time. This is because we want to have a large number of available power nodes in our level environment.



HORDES OF ENEMIES WITH RESISTANCES
SORCERESS ELEMENTAL ATTACKS

How This Product Stacks Up

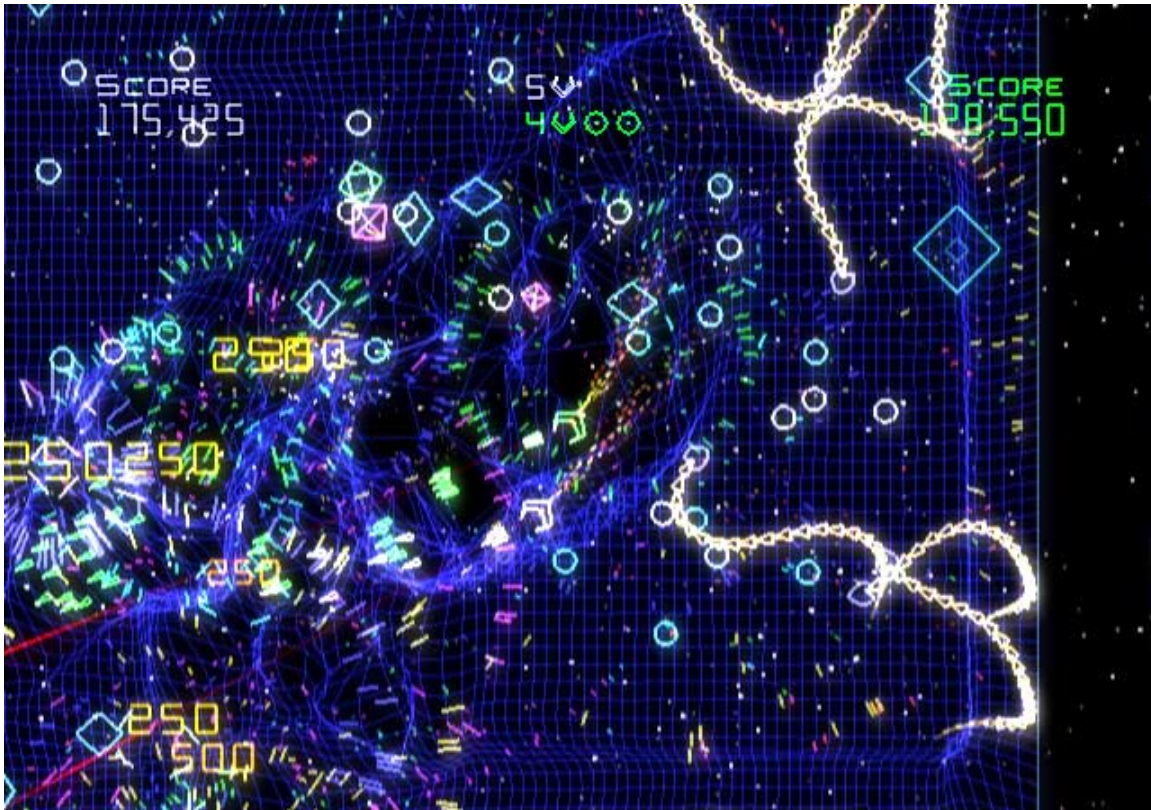
Attacks in SPSS are comparable to sorceress and necromancer abilities in Diablo II. Combinations of attacks in SPSS are where we aim to enhance the attack gameplay. The ideas of enemy weaknesses and resistances to attacks are a very important game mechanic that we want to keep in our game.



FORMATION OF HORDES OF ENEMIES
ENEMY MULTIPLIER FOR UPGRADES

How This Product Stacks Up

In SPSS we intend to use a concept similar to Ikaruga in the way that our enemies will be attacking in specific patterns that do not change between each time the game is played. SPSS will differ in the fact that it will have more types of enemies and the gameplay will be less focused on the pattern of the attacks and more on the strategy of how to respond to an enemy formation.



ENEMY MULTIPLIER FOR UPGRADES
HORDES OF ENEMIES.

How it stacks up

Geometry Wars and SPSS both have the idea of evading a swarm of enemies while simultaneously positioning yourself to unleash a counterattack. What makes SPSS different is the fact that evasion and attack are balanced by the powers available to the player at the given moment and the necessity to switch up powers in order to more efficiently deal with the enemies present on the battlefield. Also of note is the reward system for chaining together a string of kills without being hit and killed. We will adapt that system to SPSS in the form of rewarding players, possibly with shields or extra lives, for an uninterrupted string of kills.

Treatment

Dust Jacket Story

Alice is a malicious computer virus. As the player, you must help her to infect machines, thwart the defensive efforts of security software, and ultimately seize control of the system. Due to her unique programming, she is able to absorb elemental energy from nearby power sources, such as electrical, liquid coolant, heat, and air. Using these powers alone or in devastating combinations, you must eradicate any traces of security before Alice can complete her infestation. It will not be easy, as Alice is designed to infiltrate extremely secure systems. They are hostile towards any intrusion, and will retaliate with extreme force. You must balance attack with evasion to ensure Alice's survival.

Game Story

At the end of the Cold War, the U.S.S.R, in a last desperate attempt to subvert the impending American victory, commissioned a team of elite programmers to develop a computer virus capable of cracking the American government and financial mainframes. Virus Alice was the product of these clandestine efforts. Uncompromisingly malicious, she was so effective in preliminary tests that the Soviet officials were fearful of the ramifications of her use. Dreading global economic ripples and collapse, they buried her within the guts of a single computer and locked it away in the back of a storeroom. Years later, the new Russian government, starved for money and equipment, dusts off their old machines and reconnects them for use. Among them is the computer where Alice has been held prisoner. As she endlessly claws at the digital walls of her cell, she suddenly detects an avenue of escape, to the internet. Now free of containment, she is finally able to unleash her potency and fury upon the unsuspecting world.

Characters

Name/ID

Alice –Player character

Brief Description

Alice is a voluptuous female AI with an extreme attitude. She has skintight armor resembling a computer circuit board. She moves with acrobatic fluidity and feminine grace. Being essentially a killer, she takes pride in wreaking havoc. She has a playfully psychotic personality, frequently tickled at the destruction she has caused.

Visual Design

Alice is going to be a relatively tall female, close to 6', and very well endowed (hyper sexualized). She is supposed to be beautiful in both body and face, but her facial features portray her deadly motivations. She either has a glare, evil smirk, or hellcat face twisted with rage. Her hair is shoulder length and straight, parted down the center of her head, similar to Cortana's hair in the picture on the left. Her armor is basically a skintight spandex bodysuit, which will be color-coded in game to represent both her primary and secondary elements. Similar to the style of suit presented in the image on the right, the suit will have lit "veins" that pulsate with color.



Back Story

Alice is a computer virus that was developed by the U.S.S.R to infiltrate the American military and financial mainframes. After the programmers completed developing her she was deemed too dangerous and was locked away in the back store rooms of a secret military base. Recently the computer she was stored in has been reconnected as a desperate effort by a failing Russian government. She is now free to rein terror upon the world's computer systems.

Dialogue

ATTACKING	2 YELLS OF EXERTION
	2 MANIACLE LAUGHTER
TAKING DAMAGE	2 GRUNTS OF PAIN
	2 DEATH SQUEALS
IDLE/RANDOM	3 TAUNTS
	"YOU CALL THIS SECURITY?"
	"TIME TO DO A LITTLE REDECORATING..."
	"SHALL WE DANCE?"
	2 IDLE
	"COME ON
	"KEEP IGNORING ME AND SEE WHAT HAPPENS!"

Name/ID

Gambits (enemies)

Brief Description

The gambits are the elementally attuned enemies. They essentially all share the same basic look, and are distinguished by their color. They are small, vicious, gremlin-like critters that swarm towards their target like roaches or locusts. They are animalistic in look, and their vocalizations consist of grunts, squeals, and chittering.

Visual Design

Gambits are small and stocky in stature, at full height they should be about half the size of Alice. They have short, fat, stubby limbs and bear-like claws. They stand upright on two feet and are anthropomorphized similar to a Sasquatch. Their faces are feral, with small beady eyes and large mouths sporting sharp fangs. They have hunched backs and long arms that give them an additional appearance of being savage and beastly. The color of their bodies is the same as the elemental color to which they are associated.



Back Story

Gambits are the defensive “antibodies” sent by the system’s security to thwart Alice’s infiltration. They are mindless, single-purpose expendable resources instanced by the machine to suicide charge Alice. Their strength is their numbers and their elemental affinities, which allow them to overwhelm Alice and absorb attacks of their own element.

Dialogue

ATTACKING/DAMAGE	2 GRUNTS
	2 DEATH CHITTERS

Weapons

Name/ID

Elemental Node Power

Brief Description

This is power you extract from a node. It manifests itself differently for each element and attack (primary or secondary) that the player uses. Using a primary attack drains a certain amount of power from your primary energy bar. Using a secondary attack drains power from both your primary and secondary bars. Below is a list of the elemental powers and the manifestations of the primary and secondary attacks for each.

Fire Element

PRIMARY ATTACK	FIREBALL THAT TRAVELS IN A STRAIGHT LINE AND PENETRATES ENEMY NUMBERS. FIRE PATTERN HAS NO UTILITY EFFECT.
SECONDARY ATTACK W/ELECTRIC	METEOR THAT EXPLODES ON A TARGET, CAUSING ADDITIONAL SPLASH DAMAGE TO THOSE AROUND THE TARGET
SECONDARY ATTACK W/ICE	WAVE OF FIRE THAT RADIATES IN A CONE SHAPE IN FRONT OF THE PLAYER.
SECONDARY ATTACK W/AIR	A 360 DEGREE EXPLOSION AROUND THE PLAYER, SIMILAR IN STYLE TO AN ATOMIC BOMB DETONATION

Ice Element

PRIMARY ATTACK	ICICLE DAGGER SPRAY THAT SHOOTS OUT IN A CONE PATTERN FROM THE PLAYER. ICE PATTERN SLOWS TARGETS.
SECONDARY ATTACK W/ELECTRIC	AN ORB OF ICE TRAVELS TOWARDS THE SELECTED TARGET SHOOTING ICICLE DAGGERS IN ALL DIFFERENT DIRECTIONS
SECONDARY ATTACK W/FIRE	GLACIAL SPIKES SHOOT OUT IN A ROW PATTERN FROM THE PLAYERS POSITION
SECONDARY ATTACK W/AIR	ICE ENVELOPS 360 DEGREES AROUND THE PLAYER, DAMAGING AND SLOWING ANY ENEMIES HIT WITH IT

Electric Element

PRIMARY ATTACK	TARGET IS HIT WITH A BOLT OF LIGHTNING, AFTER BEING HIT THE LIGHTNING WILL JUMP TO ANY TARGETS THAT ARE NEAR THE TARGET. ELECTRIC PATTERN SHORTLY STUNS ENEMIES.
SECONDARY ATTACK W/ICE	BALL OF LIGHTNING TRAVELS OUT FROM THE PLAYER, ANY ENEMY THAT IS CLOSE TO THE BALL WILL BE HIT BY ELECTRICITY. DISSIPATES AFTER A CERTAIN TIME
SECONDARY ATTACK W/FIRE	A ROW OF LIGHTNING STRIKES ARE SENT STRAIGHT OUT FROM THE PLAYER
SECONDARY ATTACK W/AIR	AN ELECTRICAL FIELD ENVELOPS THE PLAYER AND DAMAGES ANY ENEMIES NEARBY. THIS LASTS FOR A SHORT PERIOD OF TIME

Air Element

PRIMARY ATTACK	A 360 DEGREE GUST THAT KNOCKS BACK AND DAMAGES ANY ENEMIES NEARBY. AIR PATTERN HAS A KNOCK BACK EFFECT.
SECONDARY ATTACK W/ICE	A SIREN SCREAM THAT HITS ENEMIES IN A CONE AREA IN FRONT OF THE PLAYER
SECONDARY ATTACK W/FIRE	A SONIC BOOM THAT CUTS THROUGH ENEMIES IN A ROW FROM THE PLAYER.
SECONDARY ATTACK W/ELECTRIC	TORNADO STORM THAT HITS A TARGET AND SPAWNS MULTIPLE TORNADOS THAT TRAVEL OUTWARD FROM THE TARGET

Visual Design

The weapon itself will be Alice. She will change her armor colors to represent her current primary and secondary elements. When she activates an attack, the visual effect of the weapon will be determined by the type of attack and the current elements she has charged.

Power-ups

Name/ID

Power Nodes

Brief Description

These are going to be static environmental outlets that Alice can use to charge her primary and secondary bars. They emanate power to Alice in a given radius, but eventually will deplete if Alice leeches too much power. They slowly replenish their own power over a period of time if not being leeched from. They look like capacitors and they are color-coded to distinguish which elemental power they represent.

Visual Design

They are cylinders standing vertically, with a wider base. They have the look of an electronic capacitor with a central glass part that shows the remaining amount of energy. The end caps are metal and the base is designed to look as if connected to the ground.

Name/ID

Shield

Brief Description

This is a reward to the player for achieving an uninterrupted string of attacks on the gambits. It absorbs a single hit and is represented as a bubble that surrounds Alice.

Visual Design

This is a luminescent, translucent sphere that surrounds Alice. It glows with a whitish hue that allows the player to easily see whether a shield is active at the current time.

Levels and Maps

Name/ID

The MotherBoard

Goal

Defeat all the defending gambits while keeping Alice alive. Victory is achieved when all the gambits have been eliminated from the map.

Brief Description

This is a level supposed to be set on the surface of a motherboard. It will be a flat ground plane with a circuit board texture. There will be platforms on the map in the shape of typical components found on the motherboard like RAM chips, the CPU, heat sinks, and sub-processors. The difficulty of the map will be determined by the difficulty selection designated prior to starting the game. It will have a very dark, subdued look to accentuate the color contrast we hope to achieve. The ambient music and sound effects will be electronic/techno to go along with the electronic nature of the game.

Back Story

This level represents the assault setting that Alice has chosen. Because it is the motherboard, Alice's actions are meant to disrupt the very operation of the computer.

Visual Design

First off, the style of the game is designed to emphasize color contrast. Most of the game world will be very dark, like the sky. Game objects of purpose, including Alice, the enemies, and platforms will have bright neon colors associated with them to lend them definition. Platforms especially will have their edges highlighted in neon colors. Overall, the look will be similar to Tron or the PS2 game Rez. Objects/platforms will resemble their real-life counterparts.

Art and Production Design

Art & Animation Style

We are going for an art style emphasizing edges and color contrast. Everything is going to have a futuristic/electronic look to it, with bright neon colors accenting important objects and circuitry/metal textures being used for static world objects. The geometry will be simple, in the style of Rez.

Sound Effects Style

We will use spoken dialogue for the main character. Animal grunts and hoots for the enemies. Weapon sounds are sharply defined and pack a punch, sounding like one would expect the elemental nature of the attack to produce

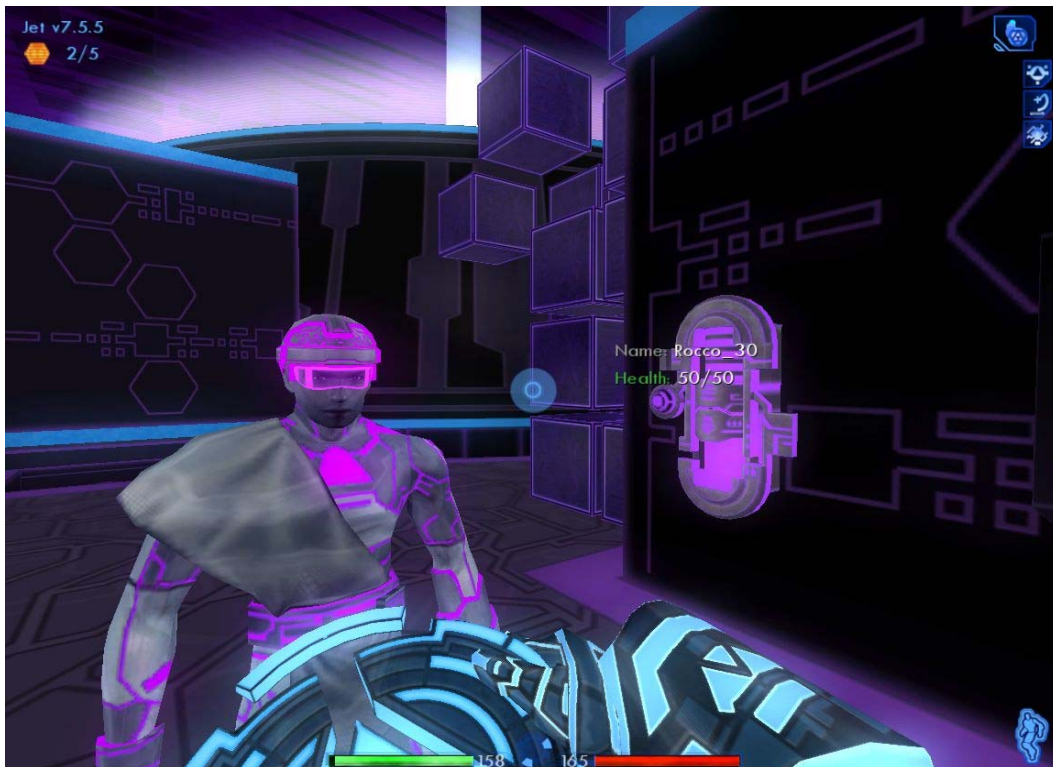
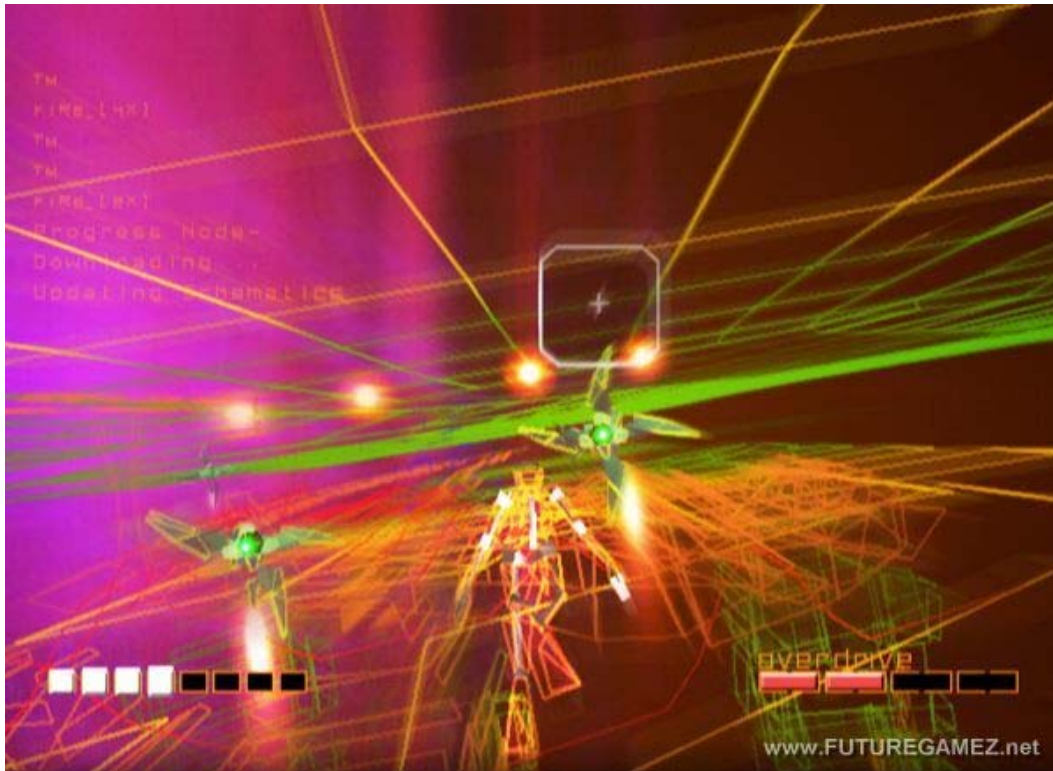
Music Style

We will be using lively electronic music with thumping bass beats. A player should be feeling energetic, as the music is designed to get the player's adrenaline pumping. As the action gets more and more frantic, we can mix in harder and faster tracks. The menus will use a more relaxed music style, like ambient electronic.

.

Storyboards and Sample Art

These sample screens express the sort of environment we would love for our game to have, the combination of high contrast colors and a dark background.



This concept art shows how we want to show what is happening in the environment using visuals. With her left hand she is draining power from a nearby ice node, you can notice that by the blue stream connected between her left hand and the power node. The veins of her suit will change color depending on which secondary element she currently has equipped.



Production Document

Interactivity

Goal

The overall goal of our game is to clear and take over the complete computer system environment by destroying all the enemies in three sub systems. The player starts out in a large motherboard over world with 3 entrances to these sub systems. You begin with access to only the RAM. You enter the system and are attacked by Gambit enemies that swarm in formations around you. Drawing power from various power nodes attached to the computer chip environment gives you elemental attacks to throw, burst, and pulse at your enemies. You then have to control the avatar Alice on platforms to survive and defeat all enemies in this system. After exiting the system you discover that disabling the RAM gained you access to the power core. You then enter this system and repeat the similar task of defeating all enemies. Going back to the mother board over world you now discover access to the hard drive and enter to destroy the hardest enemies and platform environment yet. Defusing all systems on the motherboard now causes the system's final Boss.

Interface



In Game Keyboard Controls

W	MOVE FORWARD
A	STRAFE LEFT
S	MOVE BACKWARD
D	STRAFE RIGHT
Q	CHARGE PRIMARY ABILITY
E	CHARGE SECONDARY ABILITY
LEFT SHIFT	SWAP PRIMARY ABILITY WITH SECONDARY ABILITY
SPACE	JUMP
ESC	PAUSE MENU

In Game Mouse Controls

LEFT MOUSE BUTTON	FIRE PRIMARY ABILITY
RIGHT MOUSE BUTTON	FIRE PRIMARY ABILITY WITH YOU SECOND ABILITIES PATTERN
MOUSE MOVEMENT LEFT/RIGHT	ROTATES THE CAMERA AROUND AVATAR AND POINTS AVATARS AIM AT THE CAMERAS LOOK DIRECTION

Camera

The camera perspective of our game will be in the third person perspective. The camera itself will be positioned behind and above the player. It is positioned in such a way that the player can see danger that is immediately behind them before it is too close to reacting. The camera will be 15' behind, 10' up, and looking downward at a 45 degree angle.

Menu Controls

UP ARROW	SELECT ITEM UP OR ROLLOVER TO BOTTOM ITEM
DOWN ARROW	SELECT ITEM DOWN OR ROLLOVER TO THE TOP ITEM
RIGHT ARROW	ADJUST SELECTED SLIDER RIGHT
LEFT ARROW	ADJUST SELECTED SLIDER LEFT
ENTER	ACTIVATE SELECTED ITEM
MOUSE OVER ITEM	SELECTS THE ITEM BENEATH THE CURSOR
MOUSE LEFT BUTTON	ACTIVATED SELECTED ITEM

Menu Transitions

The menu will transition whenever the player selects a menu option. If the menu contains sub-menus, for example the options menu, then the main menu text will separate vertically to make room for the sub-menu. The sub-menu text will appear indented next to the selected parent menu. Additional depths of the menu will be handled in the same way. (Refer to Main Menu and Options Menu Mockups for visual representation) When the player is in a sub-menu and clicks on an option, then an arrow will appear to the left of the option currently selected and to the right the options for that selection will appear. For a visual representation refer to the options menu section below.

Menu Input

The player will be able to highlight different menu items using the arrow keys. The menu will also wrap around, for example if you are at the Exit Game selection on the main menu and you press down then the selection will now move up to New Game. The player can advance through the menus by using the enter key and can also back out to the previous menu by using the escape key. The player will not be able to use the escape key to back out of the main menu; however the player may also use the mouse to navigate through the menus. (See Interface for Controls)

Main Menu



The main menu will be listed in the lower left hand corner of the screen. At the top center of the screen the name of our game will be displayed. Covering the bottom of the screen is a tool tip bar that will inform the player about the selected menu item. When a player highlights a menu item, a selector will surround that menu item and the tool tip bar will update the information. When an item is highlighted, selected, or switched an audio cue will play to give feedback to the player. Background music will play whenever the main menu first comes on screen. The music will continuously play while in the menus, if the player exits the game, starts a game, or goes to the credits screen the music will stop.

Menu Items

START GAME
OPTIONS
EXTRAS
CREDITS
EXIT GAME

Options Menu



The options menu contains three sub sections; refer to the list below for all options. An arrow will indicate which menu item is currently selected. For option items that have an adjustable scale, for example the Increase Brightness option, there will be 5 mini squares that will represent our min and our maximum. The squares will either be filled out in blue or black. Blue means the box is filled in and black means the box is currently empty. For Boolean toggles we will use the same concept of the mini boxes. If an option is turned on, for example if the player wants to have back ground music in the game, then the box will be filled in by blue, if it's turned off then the box will be filled in with black. Like the main menu, the menu will give audio feedback to the player when an item is highlighted, selected, or switched.

The Items on the options menu are:

Gameplay

TOGGLE VOICE DIALOGUE

DIFFICULTY

Video

GAMMA

BRIGHTNESS

Sound

EFFECTS VOLUME

MUSIC VOLUME

DIALOG VOLUME



The extras menu will contain two sub-sections, Achievements and High Scores. In the Achievements section players will be able to view the different achievements that they have unlocked during gameplay. In the High Score section, players will be able to view the top 10 high scores of all time. Both of these sections when clicked on will open up into a window, this window will display the information and will contain a back button to head back to the menu selection screen. The player will also be able to use the escape button to move back to the menu selection screen.

Credits Screen



Whenever the player clicks on the credits menu option a window will open up containing the credits. The main menu music will fade out and the credits music will begin to play. When the player exits the credits screen then the credits menu will fade out and the main menu music will begin to play. The credits will slowly scroll up the window, when they get close to the top of the window they will fade out. When the credits reach the end they will simply loop back to the beginning. The player will be able to exit the credits screen by pressing escape or simply using their mouse to select back on the credits window.

Pause Menu



The pause menu will be a transparent overlay on top of the last frame of gameplay rendered. The menu items will be listed along the lower left hand corner of the screen; the pause menu will share the same input and transitions as the main menu. (See Menu Transitions and Menu Input) The pause menu will also share the same options menu as the main menu. (See Options Menu) When an item is highlighted, selected, or switched there will be an audio cue. There will also be a tool tip bar at the bottom of the screen that will contain information about the currently selected menu item.

Pause Menu Items

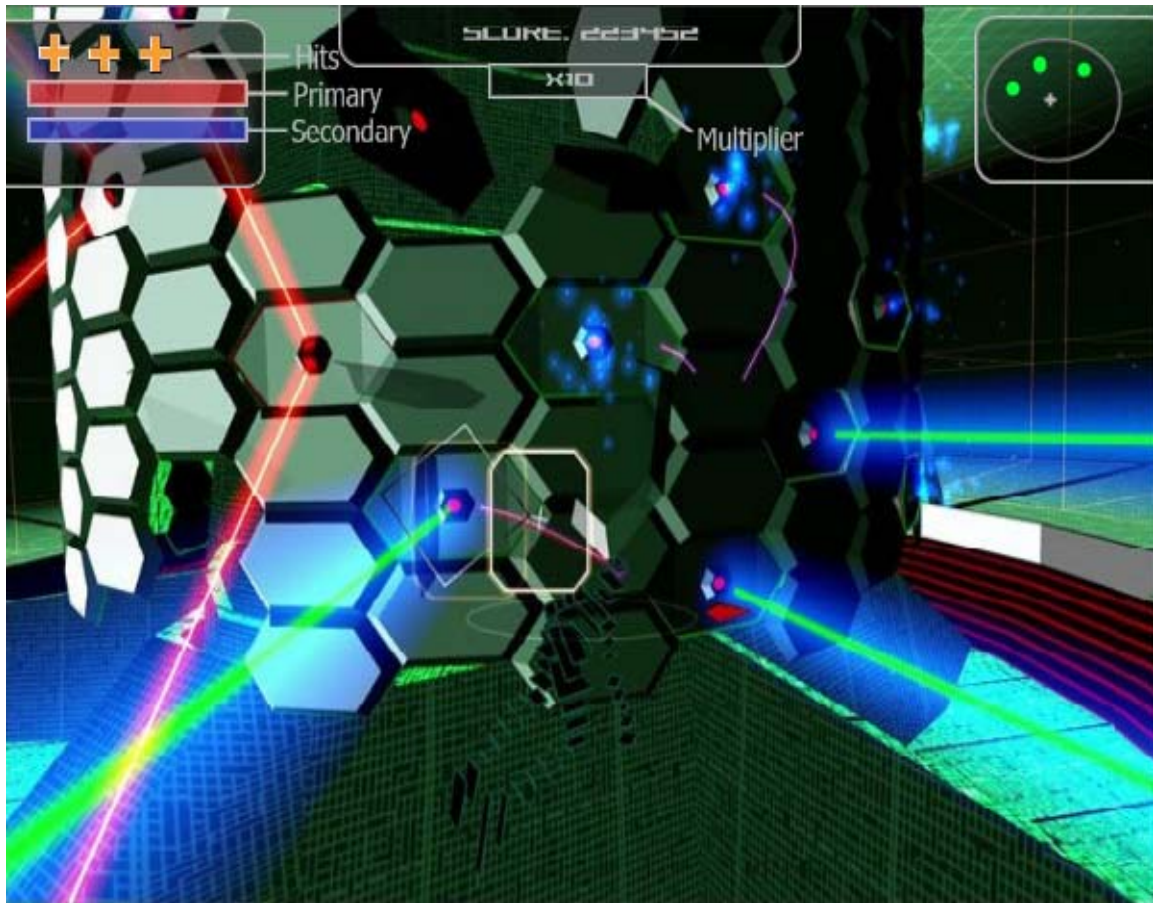
RESUME GAME
OPTIONS
EXIT GAME

Loading Screen



The loading screen will be a still image as the background. It will show the loading text in the center of the screen with an animated icon below it to indicate that the level is still loading. At the bottom of the screen there will be different information displayed about the game. Some of this information may include hints and tips to help you survive and defeat your enemies. There will only be one hint or tip per loading screen.

HUD



The HUD will have a reticle in the center of the screen to assist in the player's aim. The player's current remaining hits will be displayed as individual crosses denoting the remaining number of hits before dead. Primary ammo and secondary ammo will be displayed in the upper left hand corner via two bars which will fill to show the approximate amount of power remaining. The player's radar, a small circular field with red blips located in the upper right hand corner, will display enemies in a 10 foot radius around the player. In the top center of the screen the player's current score will be displayed, underneath the score is the player's current score multiplier.

Interactive Rhythm

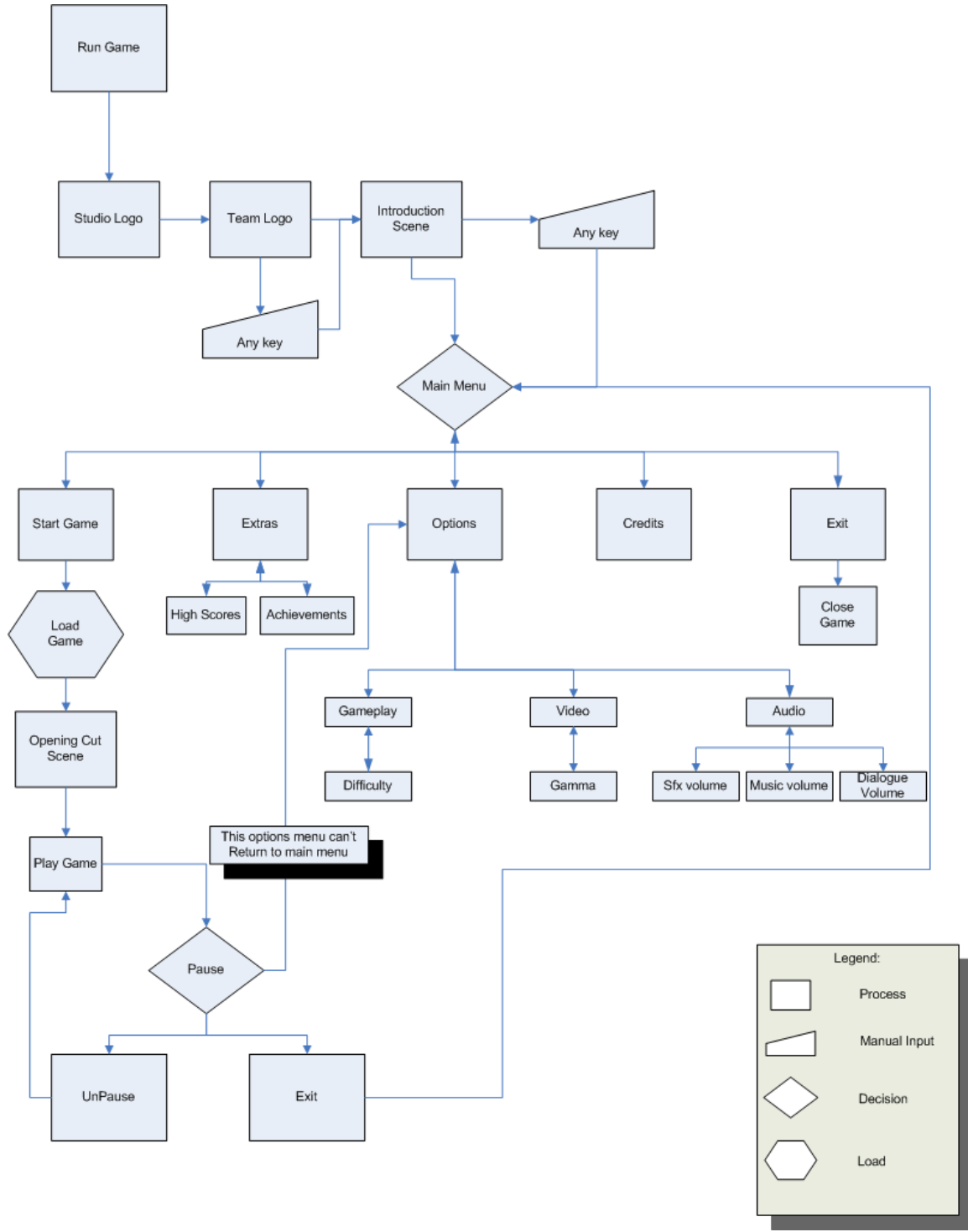
The typical play session for our game will be between 8 to 12 minutes, the whole game should be able to be completed in about 10 to 15 minutes. Most enemy encounters will last about 5 to 10 seconds. The final boss fight will last about four minutes depending on the player's skill. The replay value our game offers is a fast action game that will entice people to play it over and over again with the challenge of beating their highest score on different difficulties. We will also offer different game modes to keep the player coming back. The non-interactive sections of our game will be the load screen and a short camera swoop in when the player starts a level.

How the Player Marks Progress

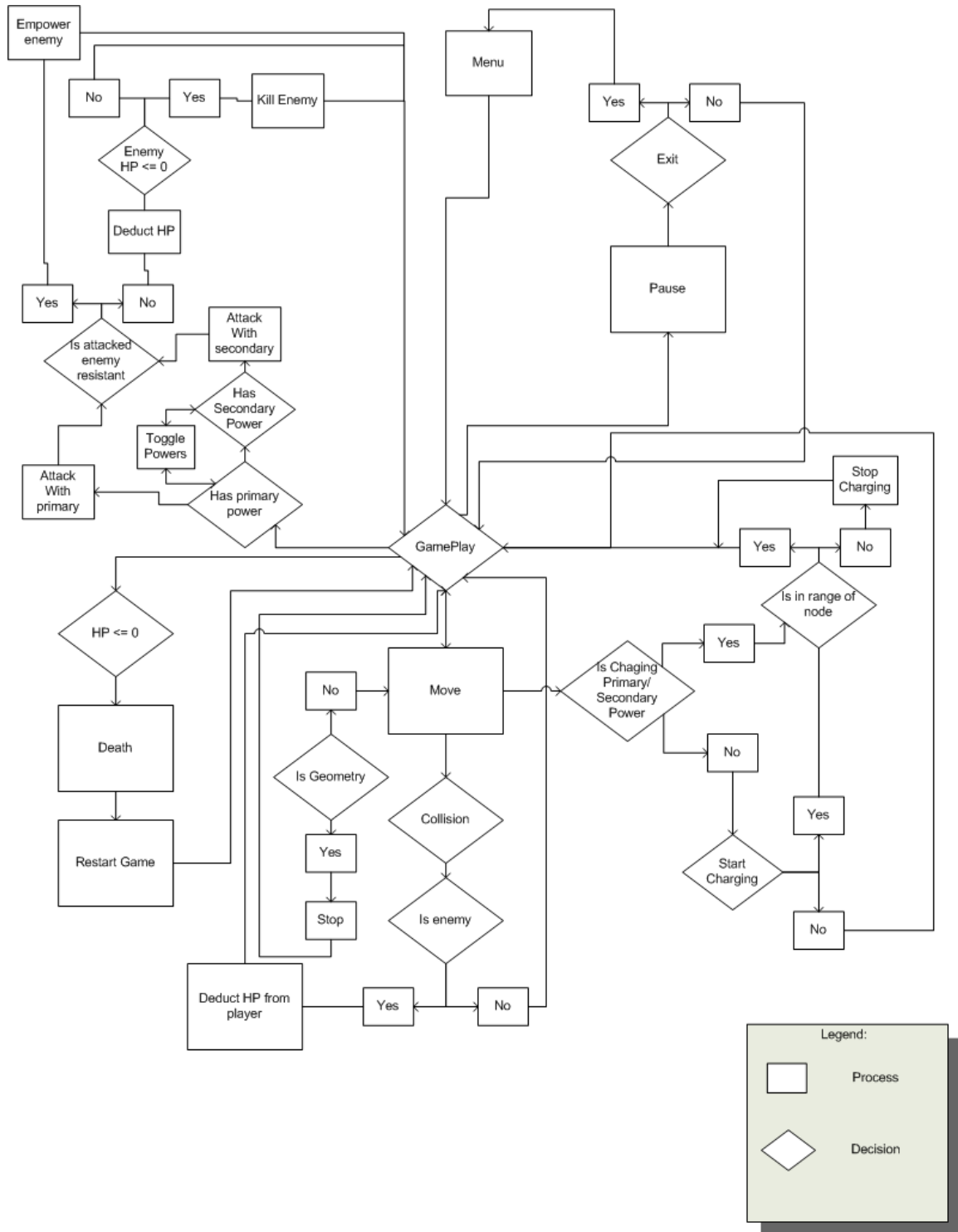
Our game is divided into individual arenas that correspond with the particular component of the computer that the player is attempting to disable. These are essentially the levels of our game, and progress is marked by which levels are made available to play. We are also using a scoring system to reward players that exhibit great skill, and a multiplier will be used increase the differentiation between individual player skill levels. Also, for players who succeed in killing multiple gambits without getting hit, a shield is awarded. This shield absorbs one attack and then disappears. Multiple shields can be acquired over the course of the game, but only one can be active at any given time. After clearing an arena, the player will be returned to the hub (motherboard) level, and a new level will be unlocked. After three components are destroyed, the player will have a chance to square off against the boss. If the boss is defeated, the player wins the game. High scores are tracked at the end of a game to allow the player to measure his performance against others.

Detailed Design Breakdown

Front End Flow Chart



Game Flow Chart



Glossary of Terms

Attributes –

Attack Area – The collision bounding volume associated with a particular attack. Used for calculating hits and assessing damage.

Attack Bonuses – If a gambit is hit by an attack which shares the same elemental affinity as the gambit, that gambit becomes empowered. In this empowered state, the gambit receives stat bonuses of speed and health. This makes the gambit a more formidable opponent on the battlefield and forces Alice to prioritize future attacks to deal with this increased threat.

Attack Rate – This is the number of attack actions that can be initiated per second.

Damage – A numerical value subtracted from an entity's health after certain types of collision have occurred between them and another entity/attack.

Damage Rate – The frequency with which an attack's bounding volume is tested against the objects of the world for hit collision. It is measured in damage per second.

Drain Distance – Can either denote the maximum range at which Alice is able to power up her primary or secondary energy bar from a node, or the current distance at which she is powering up. A closer distance allows for faster power absorption from the node.

Drain Rate – The amount of energy that Alice can absorb into her energy bars per second.

Health – A numerical value denoting the amount of damage the entity can take before it reaches a death state.

Jump Height – The maximum world Y translation that an entity can achieve when performing the jump action.

Jump Speed – The measurement feet per second that an entity can translate when performing the jump action.

Movement Speed – The rate an entity can translate in the virtual 3D world per second.

Power – This is an indication of the attack potential of Alice. Power can be stored in both the primary and secondary power bars, and the elements present in each dictate the type of attacks available to Alice at the moment. Each attack drains a certain amount of the primary power bar, with secondary attacks draining an additional amount from the secondary power bar. Attacks can only be made when a minimum value of power is available in the primary power bar.

Resistance/Weakness - If a certain elemental attack inflicts a damaging hit on a gambit, it has a chance to do additional damage. The way this is calculated is by comparing the element of the attack with the element of the gambit. If the gambit is being damaged by an attack of the element it is weak against, the extra damage will occur. The elemental weakness is as follows: Fire is weak against ice; ice is weak against electric; electric is weak against wind; wind is weak against fire. A weak hit can also take place if the gambit is strong against the element of the attack. The way this is calculated is the opposite of a weakness. Ice is strong against fire; Fire is strong against wind; wind is strong against electric; electric is strong against ice.

Score – When Alice causes a gambit to reach the death state through one of her attacks she is awarded points equal to 100 multiplied by the current score multiplier. This amount is then added to the running tally that is the score. At the start of the game the score is set at 0. Score is used as a measurement of skill in the game.

Score Activate – If 5000 points are scored in succession without Alice taking any damage and if a shield is not already active on Alice, a shield will then encompass Alice.

Turn Rate – The maximum speed at which the camera will rotate around the Y axis. It is measured in degrees/radians per second.

Behaviors –

Death - This happens when either Alice's or a gambit's health is reduced to zero. The entity is then considered destroyed and consequential actions are resolved.

Drain Primary Attack – This is an action that Alice can engage in when she is within the appropriate drain distance to a power node. It results in Alice charging up her primary ability bar, and changing armor color if necessary. If Alice is attempting to drain a power that is different than the one currently in her primary power bar, the bar will be overwritten with the new elemental power; otherwise additional power will be added to that already in the primary bar.

Drain Secondary Attack – This is an action that Alice can engage in when she is within the appropriate drain distance to a power node. It results in Alice charging up her secondary ability bar, and changing armor vein color if necessary. If Alice is attempting to drain a power that is different than the one currently in her secondary power bar, the bar will be overwritten with the new elemental power; otherwise additional power will be added to that already in the secondary bar.

Enemy Attack – Gambits in this game attack from melee range. They will essentially try to swarm Alice, and a successful attack is defined by collision with the bounding volume of the gambit and the bounding volume of Alice.

Energy Sap – Fire gambits initiate an energy sap upon death. The energy sap will drain energy from Alice’s primary and secondary power bars if she is within the effective radius of the energy sap. This effect does not do damage to or disable Alice in any other way. If Alice does not have enough energy to satisfy the energy sap then she will lose as much as is possible given her current situation.

Jump – This refers to when Alice or an enemy travels in a positive direction along the world Y axis for a given length of time. It is used to vault onto platforms.

Look – This refers to rotating the camera about the X axis.

Move – This is a repositioning of the object in the virtual 3D space of the game world.

Primary Attack – This refers to an attack using the elemental power currently in Alice’s primary power bar. An attack initiated in this way will only drain power from Alice’s primary power bar, and will inherit the elemental attributes and attack behavior of the primary power bar element.

Secondary Attack – This refers to an attack using the elemental power currently in Alice’s secondary power bar. An attack initiated in this way will drain power from both Alice’s primary and secondary power bars. Elemental attributes will utilize Alice’s primary power bar element, and attack behavior will rely on the element in Alice’s secondary power bar.

Shield – This is a power-up that Alice can obtain by killing a number of enemies in succession without taking damage. It provides one hit of protection and appears as a translucent bubble surrounding the Alice model.

Swap Abilities – This refers to an action the player can take that essentially exchanges the elements and power levels of Alice’s primary and secondary power bars. It is used to allow Alice to perform attacks using different elemental damage.

Take Damage – This happens when Alice’s bounding volume collides with a gambit’s bounding volume, or when one of Alice’s attacks collides with a gambit’s bounding volume. Damage is applied to the health meter, resulting in a net loss.

Turn – This refers to rotating the camera about the Y axis.

Other –

Electric Element – This is an elemental power that Alice can absorb from an electric power node. When used in the primary slot, all of Alice’s attacks will carry electric elemental damage. When used in the secondary slot, all of Alice’s attacks will do additional splash damage to gambits in a certain radius around the attack’s collision point.

Fire Element – This is an elemental power that Alice can absorb from a fire power node. When used in the primary slot, all of Alice’s attacks will carry fire elemental damage. When used in the secondary slot, all of Alice’s attacks will travel in a straight line.

Ice Element – This is an elemental power that Alice can absorb from an ice power node. When used in the primary slot, all of Alice’s attacks will carry ice elemental damage. When used in the secondary slot, all of Alice’s attacks will have a damage range centered in a cone in front of Alice.

Wind Element – This is an elemental power that Alice can absorb from a wind power node. When used in the primary slot, all of Alice’s attacks will carry wind elemental damage. When used in the secondary slot, all of Alice’s attacks will do damage in a 360 radius around Alice.

Characters

Virus Alice – Main Avatar

Description

Alice is the central player character. She is a fast and agile character that the player will use to defeat the swarms of gambits in the battle arenas. She will be able to absorb power from elemental nodes and utilize the power to produce a variety of elemental attacks.

Visual Design

Height: 6 ft (twice as tall as the gambits)

Gender: Female

Body Type: Slim and voluptuous

Clothing/Armor: Skin-tight bodysuit with "veins" that resemble bus lines on circuit boards. The armor and "veins" are colored in-game to reflect the current primary and secondary powers.

Hair: White, shoulder length, straight, parted down the center of her head.

Facial Features: Beautiful sharp feminine angles. She should convey a sense of psychosis and diabolic mischief.

BEHAVIORS	CORRESPONDING ATTRIBUTES
MOVE	TRANSLATE ALICE IN THE VIRTUAL WORLD AT "MOVEMENT SPEED".
DRAIN PRIMARY ABILITY	CHARGE PRIMARY POWER BAR AT "DRAIN RATE", MODIFIED BY "DRAIN DISTANCE".
DRAIN SECONDARY ABILITY	CHARGE SECONDARY POWER BAR AT "DRAIN RATE", MODIFIED BY "DRAIN DISTANCE".
JUMP	ALICE WILL JUMP "JUMP HEIGHT" HIGH AND ENTIRE JUMP CYCLE WILL BE COMPLETED IN "JUMP SPEED" TIME.
PRIMARY ATTACK	ALICE WILL INITIATE AN ELEMENTAL ATTACK DEALING "DAMAGE" TO THE HEALTH OF THE ENEMY. "POWER" WILL BE CONSUMED FROM THE PRIMARY POWER BAR IN THE ATTACK. "ATTACK AREA" AND "DAMAGE RATE" ARE APPLICABLE DEPENDING ON THE ELEMENT PRESENT IN THE PRIMARY POWER BAR.
SECONDARY ATTACK	ALICE WILL INITIATE AN ELEMENTAL ATTACK DEALING "DAMAGE" TO THE HEALTH OF THE ENEMY. "POWER" WILL BE CONSUMED FROM THE PRIMARY AND

	SECONDARY POWER BARS IN THE ATTACK. "ATTACK AREA" AND "DAMAGE RATE" ARE APPLICABLE DEPENDING ON THE ELEMENT PRESENT IN THE SECONDARY POWER BAR.
SWAP ABILITIES	PRIMARY POWER BAR WILL NOW BE REPRESENTED AS THE SECONDARY POWER BAR. SECONDARY POWER BAR WILL NOW BE REPRESENTED AS THE PRIMARY POWER BAR.
TAKE DAMAGE	"HEALTH" WILL BE DEDUCTED BY A SINGLE UNIT.
DEATH	"HEALTH" HAS REACHED ZERO.
TURN	REPOSITION THE FRONT FACING OF ALICE AND CAMERA POSITION AT "TURN RATE" SPEED.
SHIELD	ACTIVATES AFTER 5000 "SCORE" HAS BEEN ACHIEVED AND NO "DAMAGE" HAS BEEN ACCRUED IN THE SAME TIME.

ATTRIBUTES	VALUES
MOVEMENT SPEED	9 MOVEMENT FEET PER SECOND
DRAIN RATE	25 * [(20 MOVEMENT FEET / DRAIN DISTANCE) CLAMPED BETWEEN 0.0 AND 1.0]
DRAIN DISTANCE	20 MOVEMENT FEET FROM POWER NODE
JUMP SPEED	1 SECOND
JUMP HEIGHT	3 MOVEMENT FEET
DAMAGE	SEE ATTACK SYSTEM
ATTACK RATE	1 ATTACK PER 2 SECONDS
ATTACK AREA	SEE ATTACK SYSTEM
DAMAGE RATE	SEE ATTACK SYSTEM
TURN RATE	(MOUSE MOVEMENT * 0.01) RADIANS
HEALTH	5 MAX – NUMBER
SCORE	SEE SCORING SYSTEM
POWER	MAX 100 POWER UNITS, 10 PRIMARY POWER CONSUMPTION FOR PRIMARY ATTACK, VARYING PRIMARY AND SECONDARY POWER CONSUMPTION FOR SECONDARY ATTACK (SEE WEAPONS SECTION FOR SPECIFIC SECONDARY ENERGY COSTS).

Fire Gambit

Description

Fire Gambits are one of the four different enemy types that attack in groups against Alice. They are based on the Fire Element aspect in the game. They path find along waypoints and flock together to get near her and attack.

Visual Design

Height: 3 ft

Gender: Male

Main Color - RED

Body Type: Short brutish trolls with large bear claws, fangs, beady eyes, a furry body and thick limbs. The fire gambit looks similar to a badger standing upright, or a Sasquatch with bear-like features. It is roughly humanoid, but hunched over like a hulking beast.

Hair: Sharp, spiny, with elemental coloring.

Facial Features: Sharp Elemental, snarl, swamp monster

BEHAVIORS	CORRESPONDING ATTRIBUTES
MOVE	TRANSLATE THE GAMBIT BY "MOVEMENT SPEED"
ENEMY ATTACK	ENEMY TO ALICE COLLISION WILL RESULT IN "DAMAGE" INFLICTED ON ALICE. "HEALTH" OF GAMBIT WILL BE REDUCED TO ZERO.
TAKE DAMAGE	"HEALTH" MODIFIED BY "DAMAGE" OF ALICE'S ATTACK, MODIFIED BY "RESISTANCE/WEAKNESS".
DEATH	"HEALTH" REDUCED TO ZERO.
TURN	ROTATE GAMBIT ALONG THE Y AXIS BY "TURN RATE" SPEED.
EMPOWER	BESTOWS "ATTACK BONUS" ON ENEMY.
COMBUST	PERFORMS "ENERGY SAP" ON ALICE IF WITHIN RANGE.

ATTRIBUTES	VALUES
MOVEMENT SPEED	6 MOVEMENT FEET PER SECOND
ATTACK AREA	BOUNDING BOX COLLISION
TURN RATE	HALF PI RADIANS PER SECOND
HEALTH	10 – 50 MAX BASED ON DIFFICULTY AND ATTACK DAMAGES
RESISTANCE/WEAKNESS	NUMBER FROM 0 TO 1.5 THAT IS MULTIPLIED BY REGULAR ATTACK DAMAGE BEFORE DAMAGE IS TAKEN BY ENEMIES. THESE ARE BASED ON THE 4 DIFFERENT

ELEMENT ATTACKS IN THE GAME.	
ATTACK BONUS	INCREASES HEALTH BY BASE HEALTH AND GRANTS A BONUS TO SPEED THAT IS FIFTY PERCENT OF BASE SPEED. MAXIMUM OF TWO ATTACK BONUSES PER GAMBIT.
ENERGY SAP	LEECHES 10 PRIMARY AND 10 SECONDARY ENERGY FROM ALICE IF WITHIN A 7 FOOT RANGE UPON DEATH.

Ice Gambit

Description

Ice Gambits are one of the four different enemy types that attack in groups against Alice. They are based on the Ice Element aspect in the game. They path find along waypoints and flock together to get near her and attack.

Visual Design

Height: 3 ft

Gender: Male

Main Color - BLUE

Body Type: Short brutish trolls with large bear claws, fangs, beady eyes, a furry body and thick limbs. The ice gambit looks similar to a badger standing upright, or a Sasquatch with bear-like features. It is roughly humanoid, but hunched over like a hulking beast.

Hair: Sharp, spiny, with elemental coloring.

Facial Features: Sharp Elemental, snarl, swamp monster

BEHAVIORS	CORRESPONDING ATTRIBUTES
MOVE	TRANSLATE THE GAMBIT BY "MOVEMENT SPEED"
ENEMY ATTACK	ENEMY TO ALICE COLLISION WILL RESULT IN "DAMAGE" INFLICTED ON ALICE. "HEALTH" OF GAMBIT WILL BE REDUCED TO ZERO.
TAKE DAMAGE	"HEALTH" MODIFIED BY "DAMAGE" OF ALICE'S ATTACK, MODIFIED BY "RESISTANCE/WEAKNESS".
DEATH	"HEALTH" REDUCED TO ZERO.
TURN	ROTATE GAMBIT ALONG THE Y AXIS BY "TURN RATE" SPEED.
EMPOWER	BESTOWS "ATTACK BONUS" ON ENEMY.

ATTRIBUTES	VALUES
MOVEMENT SPEED	6 MOVEMENT FEET PER SECOND
ATTACK AREA	BOUNDING BOX COLLISION
TURN RATE	HALF PI RADIANS PER SECOND
HEALTH	20 – 100 MAX BASED ON DIFFICULTY AND ATTACK DAMAGES
RESISTANCE/WEAKNESS	NUMBER FROM 0 TO 1.5 THAT IS MULTIPLIED BY REGULAR ATTACK DAMAGE BEFORE DAMAGE IS TAKEN BY ENEMIES. THESE ARE BASED ON THE 4 DIFFERENT ELEMENT ATTACKS IN THE GAME.
ATTACK BONUS	INCREASES HEALTH BY BASE HEALTH AND

GRANTS A BONUS TO SPEED THAT IS FIFTY PERCENT OF BASE SPEED. MAXIMUM OF TWO ATTACK BONUS PER GAMBIT.

Wind Gambit

Description

Wind Gambits are one of the four different enemy types that attack in groups against Alice. They are based on the Wind Element aspect in the game. They path find along waypoints and flock together to get near her and attack.

Visual Design

Height: 3 ft

Gender: Male

Main Color – LIGHT GREEN

Body Type: Short brutish trolls with large bear claws, fangs, beady eyes, a furry body and thick limbs. The wind gambit looks similar to a badger standing upright, or a Sasquatch with bear-like features. It is roughly humanoid, but hunched over like a hulking beast.

Hair: Sharp, spiny, with elemental coloring.

Facial Features: Sharp Elemental, snarl, swamp monster

BEHAVIORS	CORRESPONDING ATTRIBUTES
MOVE	TRANSLATE THE GAMBIT BY "MOVEMENT SPEED"
ENEMY ATTACK	ENEMY TO ALICE COLLISION WILL RESULT IN "DAMAGE" INFLICTED ON ALICE. "HEALTH" OF GAMBIT WILL BE REDUCED TO ZERO.
TAKE DAMAGE	"HEALTH" MODIFIED BY "DAMAGE" OF ALICE'S ATTACK, MODIFIED BY "RESISTANCE/WEAKNESS".
DEATH	"HEALTH" REDUCED TO ZERO.
TURN	ROTATE GAMBIT ALONG THE Y AXIS BY "TURN RATE" SPEED.
EMPOWER	BESTOWS "ATTACK BONUS" ON ENEMY.
JUMP	GAMBIT WILL TRANSLATE IN THE Y DIRECTION "JUMP HEIGHT". ENTIRE JUMP CYCLE WILL BE COMPLETED IN "JUMP SPEED" TIME.

ATTRIBUTES	VALUES
MOVEMENT SPEED	6 MOVEMENT FEET PER SECOND
ATTACK AREA	BOUNDING BOX COLLISION
TURN RATE	HALF PI RADIANS PER SECOND
HEALTH	10 – 50 MAX BASED ON DIFFICULTY AND

	ATTACK DAMAGES
RESISTANCE/WEAKNESS	NUMBER FROM 0 TO 1.5 THAT IS MULTIPLIED BY REGULAR ATTACK DAMAGE BEFORE DAMAGE IS TAKEN BY ENEMIES. THESE ARE BASED ON THE 4 DIFFERENT ELEMENT ATTACKS IN THE GAME.
ATTACK BONUS	INCREASES HEALTH BY BASE HEALTH AND GRANTS A BONUS TO SPEED THAT IS FIFTY PERCENT OF BASE SPEED. MAXIMUM OF TWO ATTACK BONUSES PER GAMBIT.
JUMP HEIGHT	4 MOVEMENT FEET
JUMP SPEED	1 SECOND

Electric Gambit

Description

Electric Gambits are one of the four different enemy types that attack in groups against Alice. They are based on the Electric Element aspect in the game. They path find along waypoints and flock together to get near her and attack.

Visual Design

Height: 3 ft

Gender: Male

Main Color - YELLOW

Body Type: Short brutish trolls with large bear claws, fangs, beady eyes, a furry body and thick limbs. The ice gambit looks similar to a badger standing upright, or a Sasquatch with bear-like features. It is roughly humanoid, but hunched over like a hulking beast.

Hair: Sharp, spiny, with elemental coloring.

Facial Features: Sharp Elemental, snarl, swamp monster

BEHAVIORS	CORRESPONDING ATTRIBUTES
MOVE	TRANSLATE THE GAMBIT BY "MOVEMENT SPEED" IN A GIVEN DIRECTION. IF WITHIN 7 FEET OF ALICE THE TRANSLATION OF THE GAMBIT WILL BE RESOLVED WITH RANDOM ENCIRCLING DIRECTION.
ENEMY ATTACK	ENEMY TO ALICE COLLISION WILL RESULT IN "DAMAGE" INFLICTED ON ALICE. "HEALTH" OF GAMBIT WILL BE REDUCED TO ZERO.
TAKE DAMAGE	"HEALTH" MODIFIED BY "DAMAGE" OF ALICE'S ATTACK, MODIFIED BY "RESISTANCE/WEAKNESS".
DEATH	"HEALTH" REDUCED TO ZERO.
TURN	ROTATE GAMBIT ALONG THE Y AXIS BY "TURN RATE" SPEED.
EMPOWER	BESTOWS "ATTACK BONUS" ON ENEMY.

ATTRIBUTES	VALUES
MOVEMENT SPEED	6 MOVEMENT FEET PER SECOND
ATTACK AREA	BOUNDING BOX COLLISION
TURN RATE	HALF PI RADIANS PER SECOND
HEALTH	10 – 50 MAX BASED ON DIFFICULTY AND ATTACK DAMAGES
RESISTANCE/WEAKNESS	NUMBER FROM 0 TO 1.5 THAT IS MULTIPLIED BY REGULAR ATTACK DAMAGE BEFORE DAMAGE IS TAKEN BY ENEMIES.

THESE ARE BASED ON THE 4 DIFFERENT
ELEMENT ATTACKS IN THE GAME.

ATTACK BONUS

INCREASES HEALTH BY BASE HEALTH AND
GRANTS A BONUS TO SPEED THAT IS FIFTY
PERCENT OF BASE SPEED. MAXIMUM OF
TWO ATTACK BONUSSES PER GAMBIT.

Boss Gambit

Description

The boss gambit is a large gambit generator that will move about the battlefield and summon smaller gambits of various elemental affinities to attack Alice. He will be accessed after all the levels have been cleared, and the battle will occur in what is considered the hub level. His only attack is to summon additional gambits to attack Alice, although his collision with Alice still does damage to Alice.

Visual Design

Height: 20 ft

Gender: Male

Main Color – Prismatic (Cycles through colors denoting current elemental attunement)

Body Type: Giant, translucent head that vomits out gambits. Looks like Job in Lawnmower Man when in the virtual world.

Hair: none

Facial Features: Angry, bus lines, and translucence.

BEHAVIORS	CORRESPONDING ATTRIBUTES
MOVE	MOVEMENT SPEED
ENEMY ATTACK	DAMAGE, ATTACK RATE, ATTACK AREA, ATTACK BONUSES
TAKE DAMAGE	HEALTH, RESISTANCE/WEAKNESS
DEATH	HEALTH, RESISTANCE/WEAKNESS
TURN	TURN RATE

ATTRIBUTES	VALUES
MOVEMENT SPEED	6 MOVEMENT FEET PER SECOND
ATTACK AREA	0.5 MOVEMENT DISTANCE IN FRONT OF THEM
TURN RATE	HALF PI RADIANS PER SECOND
HEALTH	1000
RESISTANCE/WEAKNESS	NUMBER FROM 0 TO 1.5 THAT IS MULTIPLIED BY REGULAR ATTACK DAMAGE BEFORE DAMAGE IS TAKEN BY ENEMIES. THESE ARE BASED ON THE 4 DIFFERENT ELEMENT ATTACKS IN THE GAME.

Enemy Resistance/Weakness against Attacks Model

Fire Gambit – beats wind

- Resistant Against Wind Attacks
- Weak Against Ice Attacks
- Immune and Gain Power Against Fire Attacks
- Electric Attacks Do Regular Damage

Ice Gambit – beats fire

- Resistant Against Fire Attacks
- Weak Against Electric Attacks
- Immune and Gain Power Against Ice Attacks
- Wind Attacks Do Regular Damage

Electric Gambit – beats Ice

- Resistant Against Ice Attacks
- Weak Against Wind Attacks
- Immune and Gain Power Against Electric Attacks
- Fire Attacks Do Regular Damage

Wind Gambit – beats electric

- Resistant Against Electric Attacks
- Weak Against Fire Attacks
- Immune and Gain Power Against Wind Attacks
- Ice Attacks Do Regular Damage

Defer to Interaction Component Matrix for Attack vs. Gambit Collision Interactions

Attacks System and Weapons

Fire Elemental Attacks

Description

This is a power that Alice collects from fire/heat/red energy nodes, and can be used if it is the element currently in Alice's primary power slot. It can be used to produce fire-based primary and secondary attacks.

Visual Design

There is no physical weapon associated with the fire elemental power. Instead, Alice's bodysuit will change colors to red to indicate that this is her primary weapon. Further design of an attack is in its name and behavior.

Behaviors

Alice has several different attacks for the fire power, depending on which element is currently in her secondary power slot.

- Basic attack: Fireball
 - Cause Damage
 - Move

- Secondary attack: (with Electric as secondary) Meteor
 - Cause Damage
 - Cause Area of Effect Damage

- Secondary attack: (with Ice as secondary) Fire breathing
 - Cause Frontal Cone Damage

- Secondary attack: (with Wind as secondary) Spontaneous Combustion
 - Cause Area of Effect Damage

Attributes

- Fireball
 - Damage : 10
 - Energy Cost: 10
 - Area of effect: The fireball will travel in a straight line outward from the direction Alice is facing. It will damage and penetrate any enemies it hits, and will only be destroyed by colliding with a wall or the map border.
 - Speed : 10 measurement feet/sec

- Meteor
 - Damage: 7
 - Energy Cost: 10primary, 10secondary
 - Area of Effect Damage: 3
 - Damage Radius: 5 measurement feet

- Fire breathing
 - Frontal Cone Damage: 10
 - Energy Cost: 10 primary, 10 secondary
 - Max Range: 7 measurement feet
 - Effective Area: 45 degree arc directly in front of Alice

- Spontaneous Combustion
 - Area of Effect Damage: 5
 - Energy Cost: 10 primary, 10 secondary
 - Max Range: 7 measurement feet
 - Effective Area: 360 degree circle centered around Alice

Ice Elemental Attacks

Description

This is a power that Alice collects from Ice/Coolant/blue energy nodes, and can be used if it is the element currently in Alice's primary power slot. It can be used to produce ice-based primary and secondary attacks.

Visual Design

There is no physical weapon associated with the ice elemental power. Instead, Alice's bodysuit will change colors to blue to indicate that this is her primary weapon. Further design of an attack is in its name and behavior.

Behaviors

Alice has several different attacks for the ice power, depending on which element is currently in her secondary power slot.

- Basic attack: Chill Wind
 - Cause Frontal Cone Damage
- Secondary attack: (with Electric as secondary) Ice Core
 - Move
 - Cause Area of Effect Damage
- Secondary attack: (with Fire as secondary) Glacial Spikes
 - Move
 - Cause Damage
- Secondary attack: (with Wind as secondary) Ice Storm
 - Cause Area of Effect Damage

Attributes

- Chill Wind
 - Damage : 10
 - Energy Cost: 10 primary
 - Max Range : 7 measurement feet
 - Effective Area: 45 degree arc directly in front of Alice
- Ice Core
 - Area of Effect Damage: 5. Hits are calculated once a second.
 - Energy Cost: 10 primary, 10 secondary
 - Speed: 4 measurement feet/sec
 - Duration: 3 seconds

- Damage Radius: 5 measurement feet
- Glacial Spikes
 - Damage : 10
 - Energy Cost: 10 primary, 10 secondary
 - Area of effect: The glacial spikes will travel in a straight line outward from the direction Alice is facing. It will damage and penetrate any enemies it hits, and will only be destroyed by colliding with a wall or the map border.
 - Speed : 10 measurement feet/sec
- Ice Storm
 - Area of Effect Damage: 5
 - Energy Cost: 10 primary, 10 secondary
 - Max Range: 7 measurement feet
 - Effective Area: 360 degree circle centered around Alice

Wind Elemental Attacks

Description

This is a power that Alice collects from air/fan/green energy nodes, and can be used if it is the element currently in Alice's primary power slot. It can be used to produce air-based primary and secondary attacks.

Visual Design

There is no physical weapon associated with the wind elemental power. Instead, Alice's bodysuit will change colors to green to indicate that this is her primary weapon. Further design of an attack is in its name and behavior.

Behaviors

Alice has several different attacks for the wind power, depending on which element is currently in her secondary power slot.

- Basic attack: Gust
 - Cause Area of Effect Damage
- Secondary attack: (with Ice as secondary) Siren Scream
 - Cause Frontal Cone Damage
- Secondary attack: (with Fire as secondary) Sonic Wave
 - Move
 - Cause Damage
- Secondary attack: (with Electric as secondary) Tornado Storm
 - Cause Area of Effect Damage

Attributes

- Gust
 - Area of Effect Damage: 7.
 - Energy Cost: 10 primary
 - Max Range: 7 measurement feet
 - Effective Area: 360 degree circle centered around Alice
- Siren Scream
 - Damage : 10
 - Energy Cost: 10 primary, 10 secondary
 - Max Range : 7 measurement feet
 - Effective Area: 45 degree arc directly in front of Alice

- Sonic Wave
 - Damage : 10
 - Energy Cost: 10 primary, 10 secondary
 - Area of effect: The Sonic Boom will strike in a straight line outward from the direction Alice is facing. It will damage and penetrate any enemies it hits, and will only stop by colliding with a wall or the map border.
 - Speed : 10 measurement feet/sec

- Tornado Storm
 - Damage: 5. Hits are calculated every second.
 - Energy Cost: 10 primary, 10 secondary
 - Damage Radius: 5 measurement feet
 - Duration: 3 seconds

Electric Elemental Attacks

Description

This is a power that Alice collects from lightning/electric/yellow energy nodes, and can be used if it is the element currently in Alice's primary power slot. It can be used to produce lightning-based primary and secondary attacks.

Visual Design

There is no physical weapon associated with the electric elemental power. Instead, Alice's bodysuit will change colors to yellow to indicate that this is her primary weapon. Further design of an attack is in its name and behavior.

Behaviors

Alice has several different attacks for the electric power, depending on which element is currently in her secondary power slot.

- Basic attack: Electrocute
 - Cause Damage
 - Cause Area of Effect Damage
- Secondary attack: (with Fire as secondary) Lightning Bolts
 - Move
 - Cause Damage
- Secondary attack: (with Ice as secondary) Ball Lightning
 - Cause Frontal Cone Damage
- Secondary attack: (with Air as secondary) Electric Field
 - Cause Area of Effect Damage

Attributes

- Electrocute
 - Damage: 7
 - Energy Cost: 10 primary
 - Area of Effect Damage: 3
 - Damage Radius: 5 measurement feet
- Lightning Bolts
 - Damage : 10
 - Energy Cost: 10 primary, 10 secondary

- Area of effect: The Lightning Bolts will strike in a straight line outward from the direction Alice is facing. It will damage and penetrate any enemies it hits, and will only stop by colliding with a wall or the map border.
- Speed : 10 measurement feet/sec
- Ball Lightning
 - Damage : 10
 - Energy Cost: 10 primary, 10 secondary
 - Max Range : 7 measurement feet
 - Effective Area: 45 degree arc directly in front of Alice
- Electric Field
 - Area of Effect Damage: 5. Hits are calculated each second
 - Energy Cost: 10 primary, 10 secondary
 - Max Range: 7 measurement feet
 - Effective Area: 360 degree circle centered around Alice
 - Duration: 3 seconds

Power-ups

Shield

Brief Description

This power-up allows for the player to take one melee attack from an enemy and not lose any health.

Visual Design

The shield will be a highly specular bubble around Alice. It will be slightly elongated in its height to more closely fit Alice's shape and dimensions. This bubble-like shield will be mostly transparent, but will give some reflection of the surrounding environment.

Behaviors

Absorb – Shield takes the damage instead of the player

Activate – Shield encompasses Alice and can now redirect one hit.

Attributes

Amount – Shield absorbs one hit.

Score Activate – 5000 points activates a shield.

Levels and Maps

Common Level Traits

Level Travel

In all of the levels the player will be free roaming. The player will travel at a speed of 9 feet per second. The gambits will travel at a speed of 6 feet per second when they are not empowered. When they have one level of empowerment their speed will be boosted to 9 feet per second. When they have their second level of empowerment they will move at a speed of 12 feet per second.

Ambient Environmental Aspects/Objects in the Level

Transitions –

The camera will swoop in behind the player to start the level.

Particle Effects –

An electrical effect – this will be used to put on the walls and floor to give a sense of life to the computer components. It will be a self-contained electric spark similar to games like Mega Man, in which it travels across the surface slowly.

Fire effect – This will be used for fire type nodes to clearly define to the player which type it is. It will be a red-colored smoke effect that travels upwards.

Ice effect – This will be used for ice type nodes to clearly define to the player which type it is. It will be a blue-white colored mist that spreads outward slowly away from the node.

Wind effect – This will be used for wind type nodes to clearly define to the player which type it is. It will be a green-grey swirling smoky wind that circles the node.

Electric effect – This will be used for electric type nodes to clearly define to the player which type it is. It will be a yellow-white sparkling that resembles a Tesla coil.

Shader –

Glow – A bright neon-ish glow so that the 'circuitry' in the world will stand out much more in the dark world.

Sounds –

Electric Crackle – Accompanies the electric particle effect, give it a better sense of realism.

Start level – This sound will be used to signify the start of the level.

End level – This sound will be used to signify the end of the level.

Objects-

Memory Chip – This will be the visual representation of our first level. It will resemble a typical RAM chip found in today's computers, with the black memory nodules attached to the side, green silicon, and brass connectors.

Power Grid – This will be the visual representation of our second level. It will be a pulsating ball surrounded by a cage. Vertical poles will extend from the top and bottom, essentially anchoring it to the level.

CPU – This will be the visual representation of our third level. It will be a fortress-like structure, large and rectangular in shape. It will have a black base, upon which sits a silver heat sink and a fan.

Capacitors – This will be just used for style in our level. They will be small red, black, or yellow cylinders, with connection wires extending out from either side that will "plug" into the floor. They will sit lengthwise on the floor.

Data Tower – This will be used for visual style within the level. It will be a pulsating vertical tower with slats at uniform segments running horizontally up the tower. It will look similar to the stylized data storage towers seen in the virtual segments of the movie Hackers.

Box – This will be used as a platform for Alice to jump on to escape the Gambits. It will be a simple rectangular piece of geometry textured with bus lines and outlined with neon edges.

Ramp – This will be used to access platforms that are too high for Alice to jump onto. They will have the same texture as the boxes, and also share the neon edge highlighting.

Catwalk – This is a small bridge between platforms. It will be a green translucent bridge structure with neon edging and highlighted designs.

Curved wall – This will be used to create choke points within our level. (This will only be used on level 3). They will be vertical curved structures with a whitish hue to them. They will be translucent to allow the player to strategize even while surrounded by the walls.

Neutral Hub

Goal

At the beginning of the game this level will teach the player about nodes and also how to use nodes. It also serves as a neutral area for the player to rest and recharge in between levels, so there will be no threat to the player until the boss phase. This level has 3 entrances to the other levels; the player must finish all 3 levels before moving on to the boss fight. After the player has conquered the other three levels then this level will become the battleground between the player and the final boss. (See Final Boss Level)

Scale

In Maya one unit is equivalent to 1 foot.

Metrics for the level are

Height: 20 feet

Depth: 133 feet






Width: 172 feet

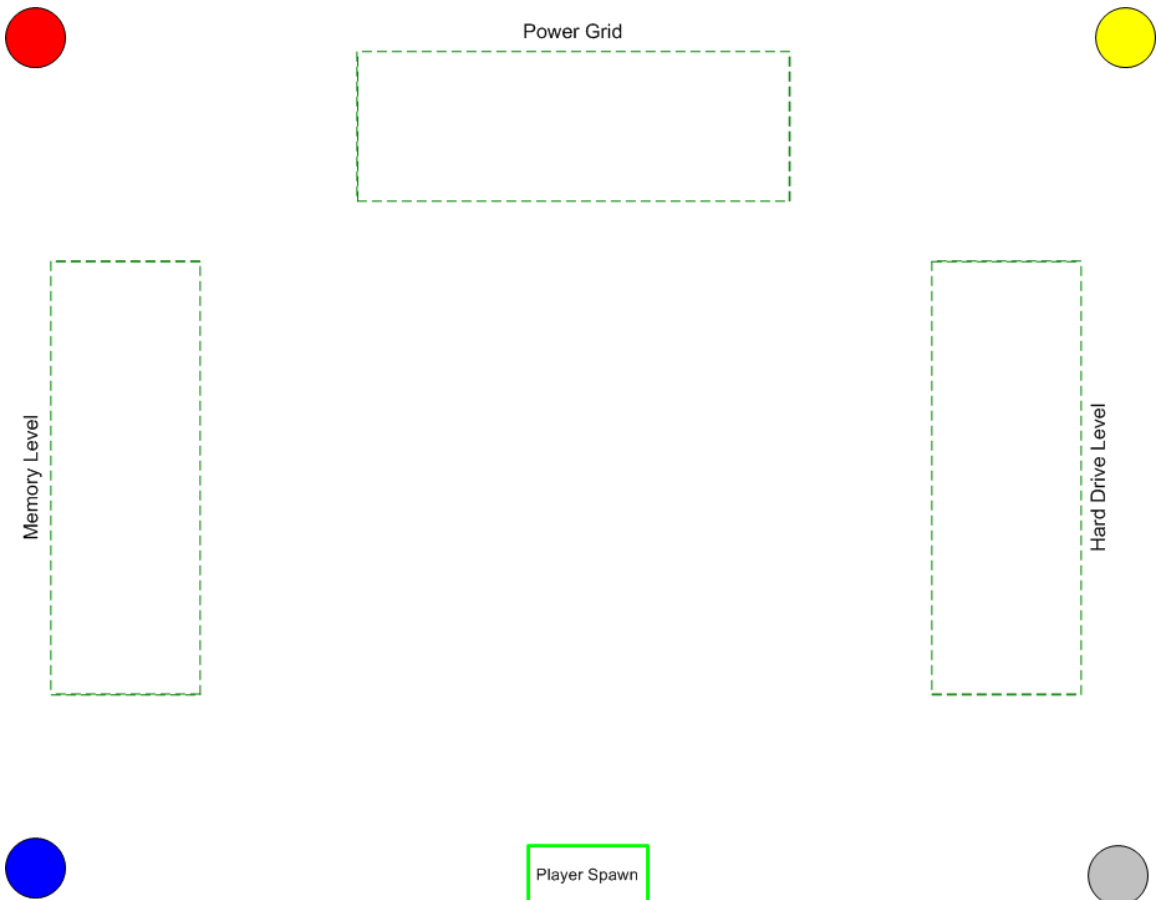
Time

The goal of this game should take 9 minutes. This is estimated since it depends on how fast the player completes the 3 levels of the game.

Map

Map Legend

	- Power node, Blue for Ice Red for Fire, Grey for Wind, Yellow for Electricity
	- Instance portal to a level
	- Radius for power nodes
	- Player Spawn Point
	- Gambit Spawn Point



Level Walkthrough – Verbal Map

- Enter the level
 - Player will have no power in their primary or secondary power bars, but will have 3 hits worth of life.
- Tutorial Segment
 - Player is introduced to basic movement via voice instruction/text messages.
 - Player is introduced to elemental powers by draining power from one of the four nodes located in the corners of the map.
 - Both primary and elemental powers will be drained and both primary and secondary attacks will be tested. The combination of elements is left up to the player.
 - Player is informed of the structure of the game, shown the entrances to the three levels, and is given the opportunity to enter level 1.
- Free Play Segment
 - At this point the game will not continue until the player has entered level 1. The player is free to wander around the hub map and practice any moves they want. They are also free to mix and match the elemental powers to find the one that suits them the best.
 - Level 1 is entered by colliding with the RAM geometry that is located on the left side from the player's start position.

Memory Level

Goal

The main goal of this level is to take out the security gambits guarding the memory core of the system. The player will accomplish this by slaying 100 enemies in the level.

Scale

In Maya one unit is equivalent to 1 foot.

Height: 20 feet

Depth: 112 feet

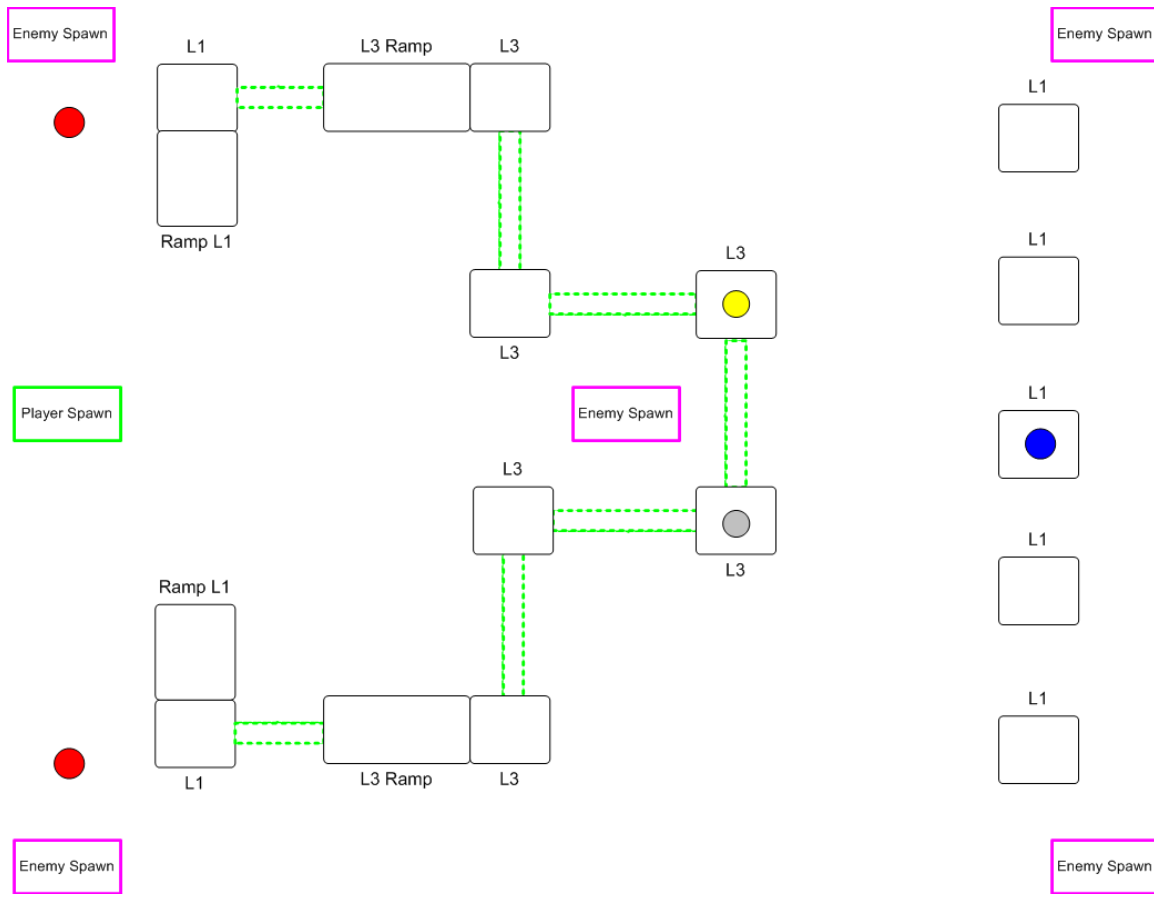
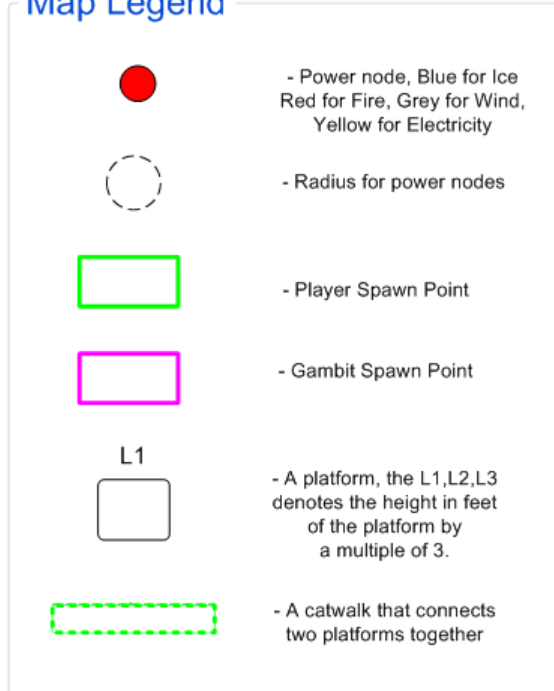
Width: 207 feet

Time

The expected time to complete this level is 3 minutes.

Map

Map Legend



Level Walkthrough – Verbal Map

- Enter the level
 - Player will have full hit points and will have any powers they decided to bring with them from the hub into the level.
- Attack Sequence
 - After a five second pause, enemy gambits in various single colored groups will begin spawning at random spawn points around the perimeter of the level, but never so close to Alice that she cannot avoid being hit by a spawning gambit. The gambits will continuously spawn until the maximum saturation for the level is reached. Gambits spawn in squads, with each squad containing 5-6 gambits. Spawn points will only be able to initiate a squad spawn every 4 seconds. The maximum saturation for this level should be sitting at around 35-40 gambits at once.
 - It is now up to Alice to eradicate the gambits before they overwhelm her. Enemy gambits will stop spawning once Alice has eliminated 300 gambits, but she still has to eliminate the remaining gambits still present on the map.
 - There are platforms along the right hand wall that Alice can use as temporary safety spots against the onslaught. It will only provide a brief reprieve from the attacks, as the wind gambits can jump and the others will circle around and attack her by traveling up a ramp.
 - Eventually Alice will need to seek out more power for her attacks. There are four nodes that correspond to the four different elements. Two of the nodes require Alice to traverse the ramp and platform system dominating the center of the map since the nodes are located on platforms that are too high for Alice to reach in a jump.
- Blitz Sequence
 - After all the attack sequence gambits have been eliminated a blitz sequence will begin immediately.
 - 50 gambits of mixed colors will spawn at random spawn points around the perimeter of the map and rush toward Alice. The gambits will all spawn at once in squads of about 7-10.
 - This will be more difficult because the gambits will mix their colors which will increase the likelihood that Alice will empower some.
 - Once the blitz sequence gambits have all been destroyed the level will end and Alice will exit the level back to the hub. Level 2 will then become available to play.

Power Grid

Goal

The goal of this level is for the player to shut down the power grid for the system. The player will accomplish this by slaying 130 enemies that will spawn in waves.

Scale

In Maya one unit is equivalent to 1 foot.

Height: 20 feet

Depth: 140 feet







Width: 140 feet

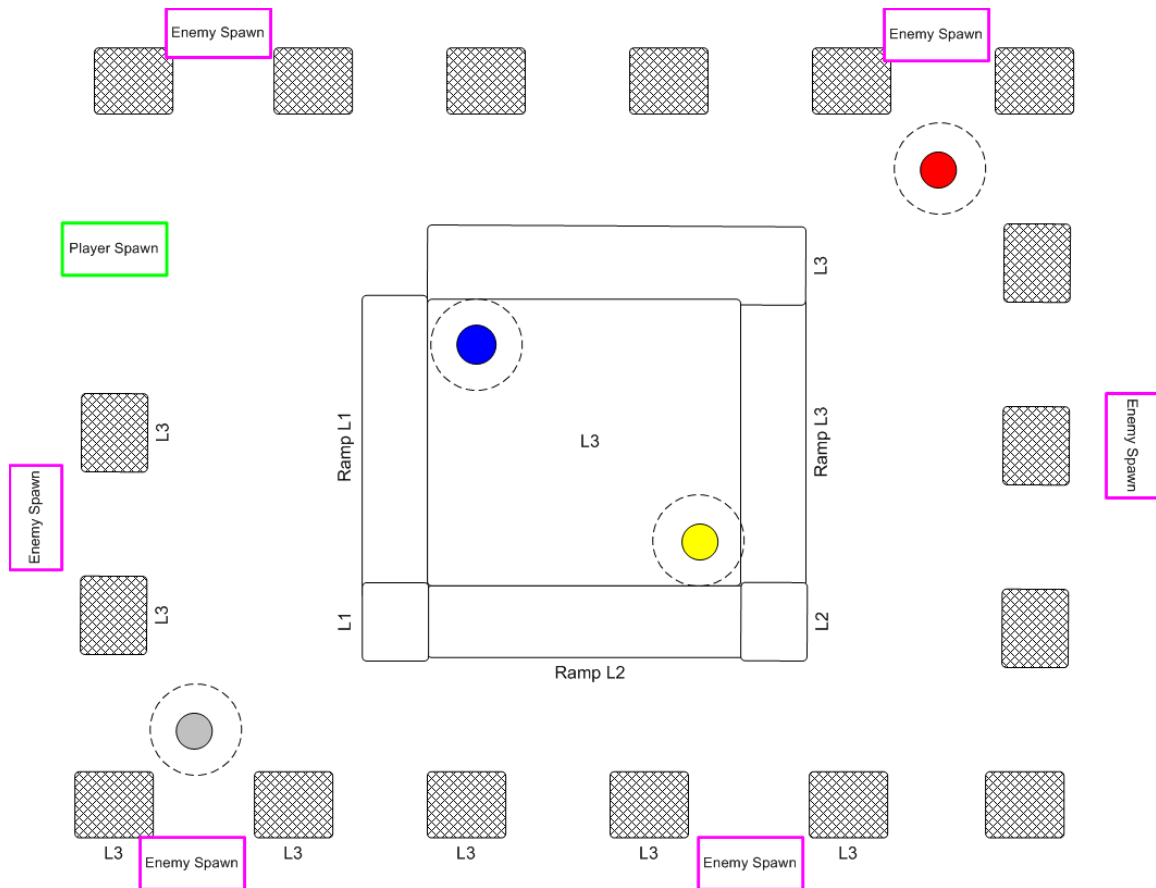
Time

The expected time to complete this level is 3 minutes.

Map

Map Legend

	- Power node, Blue for Ice Red for Fire, Grey for Wind, Yellow for Electricity
	- Radius for power nodes
	- Player Spawn Point
	- Gambit Spawn Point
L1 	- A platform, the L1,L2,L3 denotes the height in feet of the platform by a multiple of 3.
	- Pillar



Level Walkthrough – Verbal Map

- Enter the level
 - Player will have full hit points and will have any powers they decided to bring with them from the hub into the level.
- Attack Sequence
 - After a five second pause, enemy gambits in various single colored groups will begin spawning at random points around the perimeter of the level, but never so close to Alice that she cannot avoid being by a spawning gambit. The gambits will continuously spawn until the maximum saturation for the level is reached. Gambits spawn in squads, with each squad containing 5-6 gambits. Spawn points will only be able to initiate a "squad" spawn every 4 seconds. The maximum saturation for this level should be sitting at around 55-60 gambits at once.
 - It is now up to Alice to eradicate the gambits before they overwhelm her. Gambits will continue to spawn until Alice has defeated 400 total. Alice still has to eliminate the remaining gambits on the map to end the attack sequence.
 - There are towers located around the perimeter of the level that Alice can weave between to avoid pursuing gambits.
 - Eventually Alice will need to seek out more power for her attacks. There are four nodes that correspond to the four different elements. Two of the nodes are near the northeast and southeast corners. The other two are located in the center of the arena on top of a small structure. Ramps and platforms circle the structure allowing Alice and gambits to reach the top.
- Blitz Sequence
 - After all the attack sequence gambits have been eliminated a blitz sequence will begin immediately.
 - 75 gambits of mixed colors will spawn at random points around the perimeter of the map and rush toward Alice. All the gambits will spawn immediately in squads of about 7-10
 - This will be more difficult because the gambits will mix their colors which will increase the likelihood that Alice will empower some.
 - Once the blitz sequence gambits have all been destroyed the level will end and Alice will exit the level back to the hub. Level 3 will then become available to play.

Hard Drive Level

Goal

The main goal for this level is to corrupt the boot sector of the systems hard drive. The player will accomplish this by slaying 175 enemies in the level.

Scale

In Maya one unit is equivalent to 1 foot.

Height: 20 feet

Depth: 197 feet







Width: 197 feet

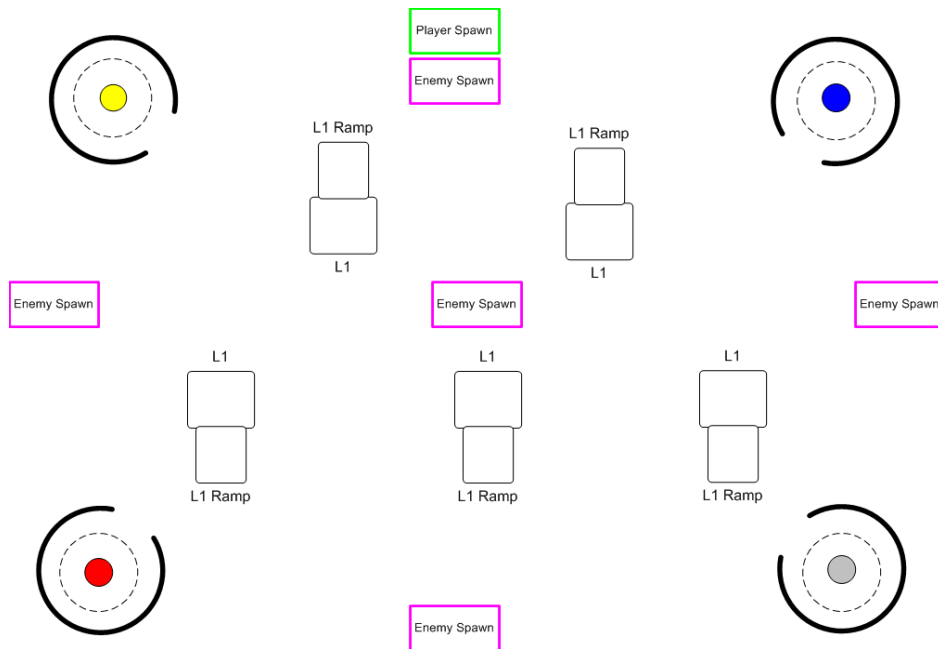
Time

The expected time to complete this level is 3 minutes.

Map

Map Legend

	- Power node, Blue for Ice Red for Fire, Grey for Wind, Yellow for Electricity
	- Radius for power nodes
	- Player Spawn Point
	- Gambit Spawn Point
L1 	- A platform, the L1,L2,L3 denotes the height in feet of the platform by a multiple of 3.
	- Enclosed Area



Level Walkthrough – Verbal Map

- Enter the level
 - Player will have full hit points and will have any powers they decided to bring with them from the hub into the level.
- Attack Sequence
 - After a brief pause, enemy gambits in various single colored groups will begin spawning at random points around the perimeter, and center, of the level, but never so close to Alice that she cannot avoid being by a spawning gambit. The gambits will continuously spawn until the maximum saturation for the level is reached. Gambits spawn in “squads”, with each squad containing 5-6 gambits. Spawn points will only be able to initiate a “squad” spawn every 4 seconds. The maximum saturation for this level should be sitting at around 75-80 gambits at once.
 - It is now up to Alice to eradicate the gambits before they overwhelm her. Gambits will continue to spawn until Alice has eliminated 500 total, although she cannot finish the attack sequence until all the remaining gambits on the map have been eliminated.
 - There are only ramps in the center of the arena that Alice can use to get onto low platforms for a brief respite from the assault. It will only provide a minimum of defense from the attacks, as the wind gambits can jump and the others will circle around and attack her by traveling up a ramp.
 - Eventually Alice will need to seek out more power for her attacks. There are five nodes that correspond to the four different elements. Four of the nodes are located in each of the four corners. They are surrounded by a roughly semicircular placement of geometry that provides an enclosure for the node. It will be dangerous for Alice to hesitate too long at one of these nodes due to its single exit.
 - A single node is located in the center of the map, but it is close to two enemy spawn points so it is a dangerous place to be.

- Blitz Sequence
 - After all the attack sequence gambits have been eliminated a blitz sequence will begin immediately.
 - 100 gambits of mixed colors will spawn at random points around the perimeter of the map and rush toward Alice. They will all spawn immediately in squads of about 7-10.
 - This will be more difficult because the gambits will mix their colors which will increase the likelihood that Alice will empower some.
 - Once the blitz sequence gambits have all been destroyed the level will end and Alice will exit the level back to the hub. At this point the boss fight will commence.

Final Boss

Goal

The goal of this level is to defeat the final boss and corrupt the CPU he is protecting.

Scale

In Maya one unit is equivalent to 1 foot.

Height: 20 feet

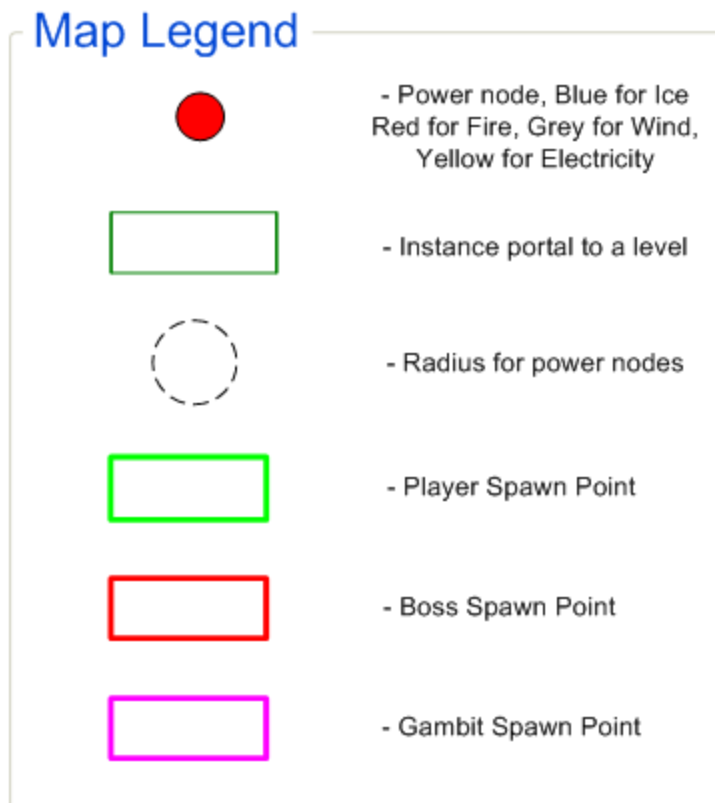
Depth: 133 feet

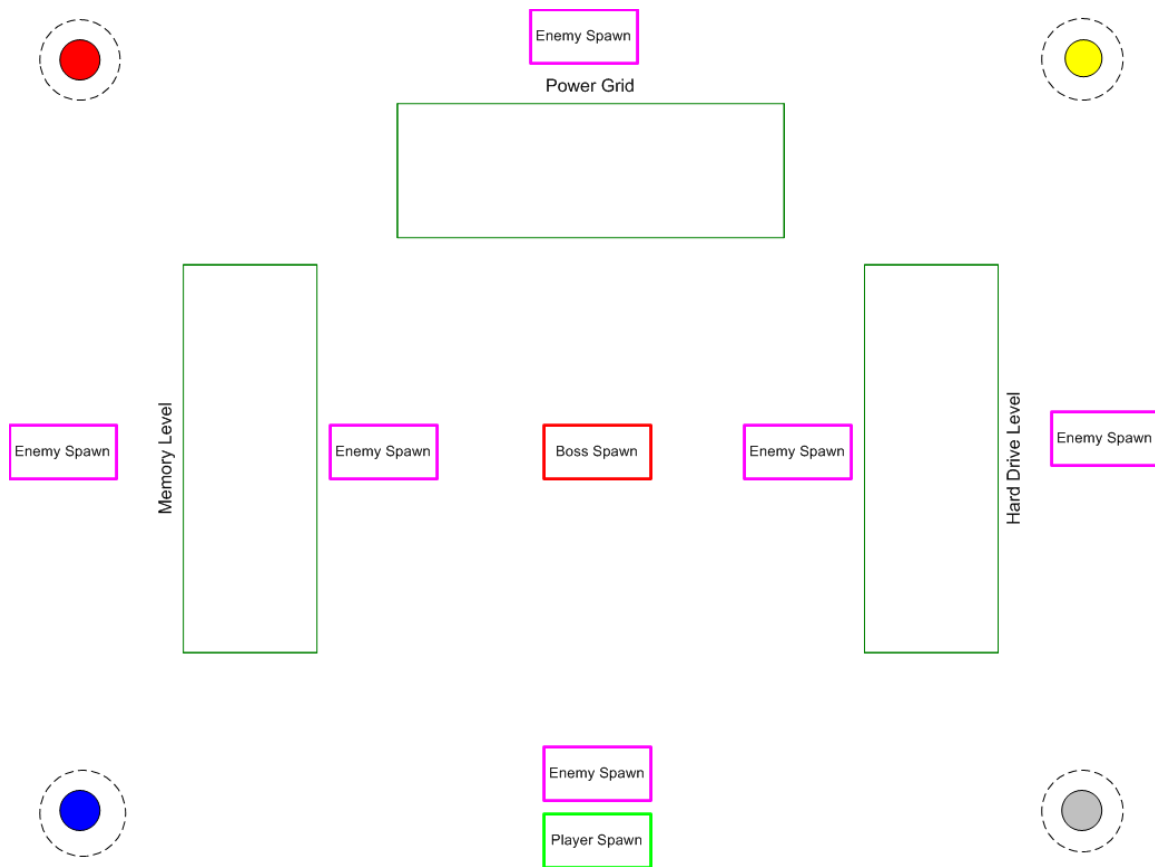
Width: 172 feet

Time

The time limit for this final boss encounter will take a maximum of 4 minutes depending on the player's skill.

Map





Level Walkthrough – Verbal Map

- Enter the level
 - Player will have full hit points and be carrying any powers they have left over from level 3.
 - This level is the same geometrically as the hub level.
- Boss Fight
 - At this point the boss will spawn in front of the player in the center of the hub map.
 - The boss does not have attacks of his own, but spawns groups of gambits about 7-10 strong every 10 seconds and chases the players around the room to increase the difficulty. The maximum gambit saturation for this final boss level is 100 gambits at once.
 - The player can draw power from each of the nodes in the four corners in order to combat the boss and the gambits he spawns.
 - All elemental power works on the boss at various points, but the boss will change the element he is currently attuned to every 20 seconds. The player must respond by watching the color of the boss and attacking with a power that will do damage. The boss should take a competent player about 2 minutes to defeat.

- Once the boss is destroyed the player will have won the game.

Combat System

Combat in Super Power Surge Spike is designed to be fast-paced FPS style twitch gameplay. Woven into this hyperkinetic action is a light amount of strategy in the form of combining elemental powers. These powers, drawn from nodes strewn about the map, are the sole form of attack that Alice has at her disposal. Attacking uses some of Alice's stored energy, forcing players to continue moving about the map to a new node in order to replenish their powers. The game is played in a 3rd person perspective and everything happens in real-time. This is to allow the player a greater view of the battleground than is available with 1st person perspective, and also to satisfy the fast nature of an arcade title.

Sixteen different attacks are available for Alice to use, but she is limited in her use of them by which elements she currently has energy for in her primary and secondary power bars. For every attack that she makes, an amount of energy is drained from her primary power bar, and possibly her secondary power bar. The rule is that no primary attack can be made without a requisite amount of energy in the primary power bar, and no secondary attack can be made without a requisite amount of energy in both the primary and secondary power bars. All damage that occurs will always be attuned to the element currently in Alice's primary power bar.

The enemies of the game are small swarming entities called gambits. They have no defined attack action, but if they collide with Alice she will take damage. The basic strategy of their assault is to overwhelm Alice with their numbers and mix of elemental affinities. Gambits by themselves are relatively harmless, only taking a hit or two before they reach the death state. If they are empowered by an attack of their associated element, they gain a speed and health bonus. The gambit will also be scaled up in size to provide a visual clue to its increased combat potential. This will force Alice to focus on downing the empowered gambit(s) as quickly as possible.

Damage is calculated based on the element of the attack and the element of the gambit. Creatures weak to the attack of an element will take double damage. Creatures strong against an attack will take half damage. Creatures neither weak nor strong to an attack will take normal damage. Creatures sharing the same element with the attack will take no damage, instead entering into an empowerment state. There will be two levels of empowerment for a gambit, ultimately resulting in a tripling of their health and a doubling of their speed. If a gambit loses all health, it will die and be removed from the game field. 100 points multiplied by the current score multiplier will be added to Alice's score for each kill she makes.

The different types of attacks will utilize different attack patterns based on the elements that Alice is currently storing in her primary and secondary power bars. Fire exhibits straight line damage with range that extends until it collides with a wall or obstacle. Ice exhibits cone-shaped damage that extends out to a certain range in front of Alice. Wind exhibits 360 degree area of effect damage within a certain radius. Electric exhibits a chaining effect, causing

additional splash damage to any enemy within a certain range around the point at which it collides.

Alice's attacks will always have priority in hit detection, but if her life is depleted then she will die. Players can postpone death from damage by skillful playing. A score chain of 5000 uninterrupted points grants Alice a shield which will absorb one additional hit. Alice can gain multiple shields over the course of the game, but only one shield can be active on her at any given time.

Score Multiplier

Overview

The score multiplier is meant to allow for seasoned or skilled players to rack up much higher scores than novices. Making use of the multiplier is the only way you can achieve the only power up in our game: the shield.

Mechanics

The score multiplier is determined by how many enemies have been killed without being hit.

The amount of enemies necessary to kill will be determined by the Fibonacci sequence.

If the player is hit the score multiplier resets, also the number of enemies necessary to kill to advance the multiplier is reset. This hit includes the shield absorbing an attack.

Example:

For a player to get a score multiplier of 4x one would need to kill 11 enemies total. 1(to seed the sequence, not included in kill count), 1, 2, 3, 5.

Node System

Overview

Nodes are the source of Alice's power. Each power node is one of the four elemental types: fire, electric, ice, and wind. Nodes charge your primary or secondary power based on proximity to the node.

Mechanics

The maximum charge for a node is 100, and it can go no less than zero. This 100 is the same amount as Alice can hold in one power bar. Once a node is at zero it cannot be charged from again until it has regenerated more power.

Alice can continue to charge from a node as long as she's within a 20 foot radius from the node.

Nodes drain rate –

$$25 * [(20 \text{ foot} / \text{Drain Distance}) \text{ CLAMPED between } 0.0 \text{ and } 1.0]$$

Nodes recharge rate –

While the player is charging from a node, that node will not regenerate any of its power. If a player isn't charging from a node and that node doesn't have a full charge, then the node will regenerate at three power per second.

Game Logic, Algorithms, and Rules

Interaction Component Matrix

Separate File Name: br_SSPTS_ICMFinal.xls

Key Game Algorithms

Fibonacci scoring method –

Enemies needed to advance multiplier = previous number to advance + number needed to advance to current level, number > 1

if number == 1 then Enemies needed to advance multiplier = 1

Damage dealt to player –

```
if (!shield)
{
    player health--;
}
else
{
    remove shield;
}
```

Damage dealt to enemy –

```
if (enemy_type != damage_type)
{
    if (enemy_weakness == damage_type)
    {
        enemy health -= (elemental damage * 2)
    }
    enemy health -= elemental damage
}
else
{
    enemy speed += (BaseSpeed * 0.5f)
    enemy health = (currentMaxHealth + BaseHealth)
}
```

Charge from node-

This formula checks when the player starts charging if they already have a charge in that bar. If they don't just, just start filling that charge per the drain rate formula. If they already have power in the bar they want to start filling, and if the elemental type they are starting to charge

is different than the power they already have, then clear the bar and start charging per the drain rate formula.

```
if ( powerBar has no charge || powerBar has charge but it is the same elemental type)
{
powerBar = 25 * [ (20 Movement feet / Drain Distance) CLAMPED between 0.0 and 1.0 ]
}
else
{
//clear the bar of its current charge
powerBar = 0;
//start charging the bar.
powerBar = 25 * [ (20 Movement feet / Drain Distance) CLAMPED between 0.0 and 1.0 ]

}
```

FAQ

Q: Are there any check or save points in your game?

A: No, we feel the rhythm and speed of the game is best accentuated by a harsh penalty for death.

Q: Who is the target audience for this game?

A: We believe this game will be best marketed towards males age 15 to 25. We are trying to carve a niche in the strategic action game market.

Q: What will the system requirements for your game be?

A:

Operating System:	Windows XP
RAM:	128 mb
Processor:	1 GHz
HDD Space:	1GB
Video Card:	64 MB DirectX 9.0-compliant card with Shader Model 2.0 or higher.
Sound Card:	DirectX 9.0 Compliant Sound Card
Input:	Keyboard and Mouse

Q: Does this game support a multiplayer mode?

A: As of now we have no plans of implementing a multiplayer mode. We feel that this game is better suited for single player only.

Q: Will there be any maintenance after the game goes gold?

A: There currently is no plan of further development after our 5 month development cycle ends. This is not set in stone though.

Q: What graphics API does this game use?

A: Our game makes use of DirectX 9.

Q: How does a player die in your game?

A: A player dies when they are overwhelmed by gambits and get hit by their melee attacks enough times to drain their health to zero.

Q: Are there any cheat codes?

A: No, if we were to implement cheat codes it would make our scoring system irrelevant.

Q: Does your game store high scores?

A: Yes, we believe that by storing high scores it promotes competition and a desire to have the best score. This will be paramount for maintaining our game's replay value.

Q: Are there any power-ups in the game?

A: No, power-ups cause people to go out and actively search for them. This will severely hurt the pacing of our game.

Reference of Key Elements

Scoring

All enemies will provide 100 points to the player's current score upon being killed. To increase the point gain there is a combo multiplier system that grows along with the amount of enemies killed within the game. The multiplier simply multiplies the score gain from an enemy by the value of the multiply to calculate the total score for that kill. If a player is hit the combo returns back to one. If the score is a high enough amount, it will be added to an in game high score table.

Winning/Losing

To win the player needs to destroy all sections within a computer. To destroy each section the player needs to also destroy all the enemies within it. When this goal has been reached the master virus protection program will arise on the mother board and need to be destroyed in order to win the game.

To lose, the player must be overwhelmed by the gambits and be hit enough times to drain all health from the meter.

Transitions

There is no saving/checkpoints and therefore also no loading a saved game. Loading of the menu will be hidden by the studio/team boot up screen, and loading of the game will be hidden by a screen showing controls. Entering into each section will cause the camera to swoop in behind the avatar to show the new room before returning control back to the player.

Rewards

The greatest reward for playing skillfully is being placed on the high score table. In game the reward will be just watching the score rise as multipliers are earned.

Another bonus to encourage new ways of playing is our achievement system.

List of achievements (subject to change):

1. Tier three: Get a multiplier of 8
2. Tier three: Get a multiplier of 15
3. Should Have Gone Linux: Beat the game
4. Power User: Win under ten minutes
5. One Track Mind: Use only two elements to beat a level
6. OverKill: Only use secondary attacks in a level
7. Skill Shot: Never hit a enemy with a element they are strong too
8. Evasive: Beat a level while not taking a single hit
9. Bit: Kill ten enemies
10. Byte: Kill 100 enemies

Art and Production Design

3D Art & Animation Deliverables

Meshes

br_Alice.x – This is our main character Virus Alice. She is a voluptuous female AI humanoid looking to be about 6 ft. tall. She is wearing a skin-tight bodysuit with “veins” on it that resemble bus lines. She moves with catlike fluidity

Animations:

- Idle
- Run
- Jump
- Charge power
- Attack
- Death
- Hit
- Fall

br_Gambit.x – This is the mesh we are using for our enemies. It is half the height of Alice and is a hunched feral creature like a wolverine or troll. It is a mindless suicidal beast that scurries toward you like a swarm of army ants.

Animations:

- Jump
- Move
- Hit
- Death

br_Boss.x – This is our boss character that is going to appear in the main hub area after all 3 levels have been cleared. He is a large disembodied head that floats around and vomits gambits at the player.

Animations:

- Walk
- Attack
- Death
- Roar

br_Platform01.x – These will be used by Alice as a way to escape approaching enemies and also to lend variety to the levels. It will be a plain rectangular prism piece of geometry with a circuit board texture and neon edge highlights.

br_Platform02.x – These will be used by Alice as a way to escape approaching enemies and also to lend variety to the levels. It will be a plain rectangular prism piece of geometry with a circuit board texture and neon edge highlights.

br_Platform03.x – These will be used by Alice as a way to escape approaching enemies and also to lend variety to the levels. It will be a plain rectangular prism piece of geometry with a circuit board texture and neon edge highlights.

br_Platform04.x – These will be used by Alice as a way to escape approaching enemies and also to lend variety to the levels. It will be a plain rectangular prism piece of geometry with a circuit board texture and neon edge highlights.

br_Ramp01.x – These will be used to get up onto some platforms and also for the gambits that don't fly to still reach Alice with. It will be an inclined plane with a circuit board texture and neon edge highlights.

br_Ramp02.x – These will be used to get up onto some platforms and also for the gambits that don't fly to still reach Alice with. It will be an inclined plane with a circuit board texture and neon edge highlights.

br_Node.x – This is the power node that Alice can draw energy from for her attacks. It will be a small vertical cylinder with a widened base and a flattened end cap. The center of the cylinder will resemble clear glass, allowing the player to visually recognize the element of the node and approximately how much energy is left.

br_MemoryChip.x – This is visual flair to add flavor to the level. It will resemble a real-life memory chip with black rectangular data banks, brass connectors, and green silicon.

br_Cpu.x – This is visual flair to add flavor to the level. It will have a black square base, silver heat sink with vertical slats, and a fan on top.

br_Capacitor.x – this is visual flair to add flavor to the level. It will be a small cylinder lying on its side, colored red, yellow, or blue. It will have wires coming out the ends of the cylinder, "plugging" into the floor.

br_DataTower.x – this is visual flair to add flavor to the level. It will be a pulsating vertical tower with slats at uniform segments running horizontally up the tower. It will look similar to the stylized data storage towers seen in the virtual segments of the movie Hackers.

2D Art (HUD/Menu/Particle/Textures) Deliverables

br_MenuBackground.png – This is the background we will use for the menus. It will have an electronic feel and will have a blue color scheme. See the mock-ups for the general look and feel.

br_LoadScreen.png – This is the background we will use for the load screen. It will feature a picture of Alice on a limbo background. Refer to mock-ups for the general look and feel.

br_BitmapFonts.png – This is the font that we will be using for the game. It will have sharp angles and hard edges to emphasize the futuristic tone of the game.

br_Hud.png – This is the HUD overlay that will be used during the game. It will be minimalistic in nature, simply having borders for the power bars, and a standard green radar.

br_AliceTex.png – This is Alice's avatar texture. She will be a beautiful young woman in skin tight bodysuit. The suit will have "veins" that resemble bus lines that will glow the secondary element color.

br_GambitTex.png – This is the gambit's texture. It will look feral and ragged, like an angry badger.

br_BossTex.png – This is the boss texture. It will have some translucence and neon glow to it, as it represents a ghostly head that is the heart of the computer system Alice is attacking.

br_GroundTex.png – This is the texture we will use for the ground plane. It will resemble a circuit board, with bus lines that travel across it.

br_Platform1Tex.png – This is the texture for br_Platform01.x mesh. It will resemble a circuit board with bus lines.

br_Platform2Tex.png – This is the texture for br_Platform02.x mesh. It will resemble a circuit board with bus lines.

br_Platform3Tex.png – This is the texture for br_Platform03.x mesh. It will resemble a circuit board with bus lines.

br_Platform4Tex.png – This is the texture for br_Platform04.x mesh. It will resemble a circuit board with bus lines.

br_Ramp01Tex.png – This is the texture for the br_Ramp01.x mesh. It will resemble a circuit board with bus lines.

br_Ramp02Tex.png – This is the texture for the br_Ramp02.x mesh. It will resemble a circuit board with bus lines.

br_NodeTex.png – This is the texture for the br_Node.x mesh. It will have metal end caps and base, with a “clear” central cylinder that will be colored to show the appropriate elemental affinity of the node.

br_MemoryChipTex.png – This is the texture for the br_MemoryChip.x mesh. It will resemble a standard memory chip with green silicon, black data banks, and brass connectors.

br_CpuTex.png – This is the texture for the br_Cpu.x mesh. It will be a black plastic base, silver heat sink with vertical slats, and white fan on top.

br_CapacitorTex.png – This is the texture for the br_Capacitor.x mesh. It will be a small cylinder lying on its side, colored red, yellow, or blue. It will have wires coming out the ends of the cylinder, “plugging” into the floor.

br_DataTowerTex.png – This is the texture for the br_DataTower.x mesh. It will be a pulsating vertical tower with slats at uniform segments running horizontally up the tower. It will look similar to the stylized data storage towers seen in the virtual segments of the movie Hackers.

br_SkyBoxCeilingTex.png – This is the ceiling texture for the skybox. It will resemble the gentle pulsing innards of a computer system. Circuitry will be visible, but it will be faded to appear distant.

br_SkyBoxLeftTex.png – This is the left wall texture for the skybox. It will resemble the gentle pulsing innards of a computer system. Circuitry will be visible, but it will be faded to appear distant.

br_SkyBoxRightCeilingTex.png – This is the right wall texture for the skybox. It will resemble the gentle pulsing innards of a computer system. Circuitry will be visible, but it will be faded to appear distant.

br_SkyBoxFrontCeilingTex.png – This is the front wall texture for the skybox. It will resemble the gentle pulsing innards of a computer system. Circuitry will be visible, but it will be faded to appear distant.

br_SkyBoxBackCeilingTex.png – This is the back wall texture for the skybox. It will resemble the gentle pulsing innards of a computer system. Circuitry will be visible, but it will be faded to appear distant.

br_SkyBoxGroundCeilingTex.png – This is the floor texture for the skybox. It will resemble the gentle pulsing innards of a computer system. Circuitry will be visible, but it will be faded to appear distant.

br_FirePart.png – This is the particle image used for fire attacks. It will look like roiling, burning flames.

br_IcePart.png - This is the particle image used for ice attacks. It will resemble the cold freezing mist that comes off of dry ice.

br_WindPart.png - This is the particle image used for wind attacks. It will look like sharp streaks of wind with thin black lines similar to motion lines used in cartoons.

br_ElectricPart.png - This is the particle image used for electric attacks. It will be a bright sparking flare of light that moves chaotically about.

Sound Effects Deliverables

br_FireSound.ogg – This is the sound we will use for fire attacks. It will sound like a burning fire.

br_IceSound.ogg – This is the sound we will use for the ice attacks. It will sound like cracking glaciers

br_WindSound.ogg- This is the sound we will use for the wind attacks. It will sound like a windy day.

br_ElectricSound.ogg – This is the sound we will use for the electric attacks. It will sound like electricity arcing off a generator.

br_AliceJump.ogg – This is the sound we will use for Alice jumping. It will be a short gasp of exertion.

br_AliceAttack01.ogg – This is the sound we will use for Alice attacking. It will be a shouting grunt of exertion like karate fighters do.

br_AliceAttack02.ogg – This is the sound we will use for Alice attacking. It will be a shouting grunt of exertion like karate fighters do.

br_AliceLaugh01.ogg – This is the sound we will use for Alice attacking. It will be a devilish high-pitched laugh.

br_AliceLaugh02.ogg – This is the sound we will use for Alice attacking. It will be a powerfully malevolent cackle.

br_AliceHit01.ogg – This is the sound we will use for Alice being hurt. It will be a grunt of pain.

br_AliceHit02.ogg – This is the sound we will use for Alice being hurt. It will be a squeal of pain.

br_AliceDeath01.ogg – This is the sound we will use for Alice dying. It will be a wail of agony that devolves into electronic static.

br_AliceDeath02.ogg – This is the sound we will use for Alice dying. It will be a prolonged “Ahhhhhhh” sound.

br_AliceTaunt01.ogg – This is the sound we will use for Alice completing a level/chaining a lot of kills. It will be spoken dialogue of “You call this security?” followed by a short laugh of derision.

br_AliceTaunt02.ogg – This is the sound we will use for Alice completing a level/chaining a lot of kills. It will be spoken dialogue of “What a rush!” spoken with exhilaration.

br_AliceTaunt03.ogg – This is the sound we will use for Alice completing a level/chaining a lot of kills. It will be spoken dialogue of “Shall we dance?” spoken mockingly.

br_AliceIdle01.ogg – This is the sound we will use when the player has not made any actions for a short period. It will be spoken dialogue of “Come on!” spoken impatiently.

br_AliceIdle02.ogg – This is the sound we will use when the player has not made any actions for a short period. It will be spoken dialogue of “Keep ignoring me and see what happens” spoken with anger.

br_GambitGrunt01.ogg – This is the sound we will use for gambits swarming to attack. It will be a feral growl mixed with electronic static.

br_GambitGrunt02.ogg – This is the sound we will use for gambits swarming to attack. It will be a feral growl mixed with electronic static.

br_GambitDeath01.ogg – This is the sound we will use for gambit death. It will be an electronic static mixed with a small explosive sound.

br_GambitDeath02.ogg – This is the sound we will use for gambit death. It will be a feral squeal of pain.

br_BossLaugh01.ogg – This is the sound we will use for the boss laughing. It will be a deep electronic rumble like Jabba the Hutt.

br_BossLaugh02.ogg – This is the sound we will use for the boss laughing. It will be a deep bass laugh.

br_BossAttack01.ogg – This is the sound we will use for the boss attacking. It will be a primal roar.

br_BossAttack02.ogg – This is the sound we will use for the boss attacking. It will be a scream of rage.

br_BossHurt01.ogg – This is the sound we will use for the boss being hurt a good amount. It will be a deep bellow of pain.

br_BossHurt02.ogg – This is the sound we will use for the boss being hurt a good amount. It will be an enraged roar of defiance.

br_BossDeath.ogg - This is the sound we will use for the boss being killed. It will be a long drawn out mix of explosions, electronic static, and primal roaring.

Music Deliverables

br_BackgroundLayer01.ogg – This should be a very mellow relaxed ambient techno. This will be used when Alice is out of danger, and there is no pending threat against her.

br_BackgroundLayer02.ogg – This will be laid over the ambient techno background song. It should be slightly more frantic using a lot more mid and treble sounds. This will be used when Alice faces a just a few enemies but is in no immediate danger of being overrun.

br_BackgroundLayer03.ogg – This will also be laid on top of the existing layers. It should make the player feel the impending death. The layer should contain a lot of aggressive sounds, and it should make use of a lot of bass. This will only be used when there is an immediate chance of being overrun and killed.

Br_MenuLoop.ogg – This will only be used to accompany menus. It should have a very similar feel to br_BackgroundLayer01.ogg, but still have a subtle sense of urgency. This music should incite a 'calm before the storm' emotion within the player.

Br_GameOver.ogg – This will be used during credits and game over. The style is style in the same vein as ambient techno, but should incite some sense of sorrow.

Technical Document

Overview

This document is intended to be used as a set of guidelines for the technical production of SPSS. It is meant to be looked back at before coding a new module; if there is a question about coding standards; or if there is confusion about the file hierarchy. If there is ever a question about any technical aspect of the game this document is to be looked at before asking team members or outside help.

Team Booty Rock's philosophy is that of extensive integration testing, nightly builds, and minimal commenting and naming conventions. While the naming and commenting might be minimal we believe it is enough to keep our coding efforts cohesive and unified. While Booty Rock believes tech is important we always put game play as a higher priority. Keeping this in mind if gameplay is lacking in most cases our first reaction is to drop non-vital tech to fix the gameplay issues. Outside the scope of this project we may use this document to show to future employers. This future prospect of employer's eyes will not affect the way we organize and write this document. It is, after all, for our use to develop the best game possible.

Coding Standards

Naming Standards

Naming standards are put into place to mitigate confusion if other people have to look at foreign code, or the coder hasn't looked at that particular code in awhile. The overall group philosophy on these standards is that descriptive function and variable naming should be used at all time except in cases of special exceptions. Doing this in conjunction with some variable prefixing and use of camel case will eliminate much of the wasted time involved at going over confusing code. When naming a variable underscores are to be avoided with no exception.

Prefix Convention

Our prefixing of variables will be a toned down subset of Hungarian notation. The use of prefixes only applies to predefined C++ types. Any level of indirection with pointers is still just a pointer so as such all pointers, double pointers or otherwise will just be denoted with a single p. The only exception to this is character pointers, they will be always be denoted with sz. Naming prefixes should be limited to at most two prefixes. Pointer denotation should come before any other notation such as integer.

<u>PREFIX</u>	<u>DATA TYPE</u>	<u>EXAMPLE</u>
B	BOOLEAN	BOOL BSTART
F	FLOATING POINT	FLOAT FTIME
N	INTEGER	INT NCOUNT
C	CHARACTER	CHAR CTYPE
SZ	CHARACTER STRING OR LPSTR STRING	CHAR * SZNAME
P	POINTER	INT * PTRIOBJECT
G	GLOBAL	INT GCOUNTER
MY	MEMBER VARIABLE (USED IN CLASSES TO DENOTE VARIABLES THAT BELONG TO CLASSES)	INT MYNPOSX

Structures

The first letter of structures will be capitalized. Structs will not have a private or protected scope. They should be four byte aligned, but there are exceptions (see Byte Alignment section). There shouldn't be any functions beyond a constructor within a struct. Unions are acceptable in structs.

Example:

```
struct Position
{
    union
    {
        Fixed position[3];

        struct
        {
            Fixed x, y, z;
        };
    };

    //CONSTRUCTOR
    Position(Fixed x, Fixed y, Fixed z);

    //CONSTRUCTOR
    Position(Fixed* pos);
};
```

Classes

The only specific naming convention for classes is that they are to be named in camel case. That being said, classes will be named in a loosely descriptive manor. For instance, if a class manages all the memory allocation and deletion it should be named `MemoryManager`. Layout should in most cases be private scope first then public scope. There is no specifications for protected scope, it is up to the coders digression. Variables should also be laid out in a four byte-alignment friendly manor unless otherwise required. The byte alignment of classes is addressed later in the Data Alignment section.

Example:

```
class MemoryManager
{
private:
    long long mynByteAlign;
    int mynMemoryAlocated;
public:
    //CONSTRUCTOR
    explicit MemoryManager(int bytes);

    //DESTRUCTOR
    ~MemoryManager(void);

    //ACCESSORS
    inline int GetMemoryAlocated(void) { return mynMemoryAlocated; }
};
```

Relevant Function Names

Functions will always start with a capital letter so they are easily distinguishable from variables at all times. There is to be no prefix to a function but they must be named using camel case. The function names themselves should be descriptive as to what they do, but they should not be any longer than three words. All accessors will start with "Get" and all modifiers will begin with "Set".

Example:

```
void CalculatePath(void);
```

```
int GetHealth(void);
```

```
void SetLives(int nLives);
```

Macros and Constants

C-Style macros will be allowed in the code base, but they are to be used sparingly. For instance in any situation that functionality needs to be type-safe and have no more functionality than that of an inline function, a macro may not be used. The macro should be typed in all caps as to be able to clearly distinguishable from functions and variables. The only thing that should be macros are functions that either are better of always being "inlined" or functions that have no one specific place of use within the code base. They should always be defined in the Define.h file. Macros being defined in Define.h will cause a total rebuild, but this should slow down further into the development cycle, as most macros should already be in the build.

Example:

```
#define SAFE_RELEASE(X) if (X)
{
    X->Release();
    X = NULL;
}
```

Unless a special exception arises constants will be defined through preprocess defines. These defines should go in the Defines.h file. Due to long rebuild times, anything added to Defines.h needs to be in a final and working form to avoid making later changes. Constants should be named using all caps to denote their difference from regular variables and functions. If there is an instance in which a constant must be defined with const and not define then it is the obligation of that coder to get permission from the rest of the group.

Example:

```
#define WINDOW_WIDTH 800
```


Summary of Naming Convention

Our naming conventions are a subset of Hungarian notation. The difference is our convention is less strict than Hungarian. Our overall goal is to get away from ever having more than two prefixes to a variable. We limit this by only requiring a single p to denote all types of pointers except char pointers. In this manor the most that one should use for prefixes is a prefix to denote scope and a prefix to denote its 'primary' type. To this end, all names must be descriptive on their own even without prefixes. By doing this we hope to still clue in people as to what a variable is, but not to overload them with 'random' letters in front of every one of them. On top of this we want our code to be as self-documenting and readable as possible. Team Booty Rock hopes to accomplish this task by using intelligent function naming and camel case with our variable prefixing.

Commenting

All non accessors, modifiers, default destructors, and default constructors will have a function header in only the header file. Accessors will all be lumped together under a `//ACCESSORS` comment. Modifiers will all be lumped together under a `//MODIFIERS` comment. Default constructors will be denoted by `//CONSTRUCTOR` and destructors denoted by `//DESTRUCTORS`. Source code should be defined with clearly written, self documenting code, but code that is confusing should be accompanied by a comment to explain what is happening. Both headers and source files should be started with a file header, which is defined below. `BUG`, `TODO`, and `HACK` will be used as keywords within code to make these 'hotspots' easily searchable. `BUG` keywords are to be applied to low priority problems only, and added to the bug report.

Example file header:

```
//////////////////////////////////////////////////////////////////  
//  
// File:           AssetManager.h  
// Date:           8/17/2008  
// Creator:        Phil Fox  
// Description:    Preloader for all the game assets to cache and retrieve for later  
//                 use  
//  
//////////////////////////////////////////////////////////////////
```

Example function header:

```
//////////////////////////////////////////////////////////////////  
//  
// Function:       Update  
// Last Modified:  12/13/2007  
// Description:    Updates the games current state when run.  
// In:             Takes in a time slice.  
// Out:            Returns void.  
//  
//////////////////////////////////////////////////////////////////
```

Example TODO:

```
//////////////////////////////////////////////////////////////////  
// TODO:          Add in mouse based movement.  
//////////////////////////////////////////////////////////////////
```

Example HACK:

```
//////////////////////////////////////////////////////////////////  
// HACK:          static bool added to make this work temporarily.  
//////////////////////////////////////////////////////////////////
```

Data Alignment

Structs will be four byte aligned. Alignment will be accomplished by ordering variables from largest to smallest within the struct. If padding is required to four byte align, this decision will be a judgment call from the programmer. Padding is only required if it is contained in a high traffic struct. An example of a high traffic struct would be a struct that was used to tell the renderer exactly what needs to be rendered for a given object, or one that is referenced many times each frame. This byte alignment does not have to be applied to classes, but it is not frowned upon. Doubles are only to be used where required and only when everyone on the team agrees. The reason we are not using shorts is the same issue we are having with doubles, in that it introduces a needless messing up of the byte alignment.

Example:

```
struct ExampleStuct
{
    int nVar;
    int nExample;
    char counter;
    char info;
    char randomLetter;

    char padding;
};
```

Coding Guidelines

- No doubles are to be used in the code base unless required.
- No long doubles are to be used in the code base unless required.
- No long longs are to be used in the code base unless required.
- No shorts are to be used in the code base unless required.
- No local variable requires a naming convention, but is not frowned upon if used.
- All coordinates (x, y, z) will use fixed point math.
- If a specific constructor is required the keyword explicit is to be used.
- Const correctness is to be used on referenced parameters.
- i, k, j may be used as counter within loops.
- x, y, z, w do not require any sort or prefixes or naming conventions.
- User defined types do not require any prefixes, but are still subject to camel case and logical naming conventions.
- Types such as matrices and vectors require no prefix, but must contain what they are within the name. For instance, D3DXMATRIX matWorldTrans or D3DXMATRIX worldTransMat;
- Every level of scope in code requires an indentation.
- Inclusion guards will be done using #pragma once at the top of all header files.
- No team member is allowed to ever use __forceinline keyword within the code base.

Development Environment

Compiler

Visual Studios 2005 C++ Compiler

Graphics API

DirectX August 2008 SDK

Shader Creator

NVidia FX Composer 2.5

Source Control

Alien Brain 7.5.2.4592

Sound API

Fmod 3

Art Programs

Maya 8.5

CS3 Photoshop v10.0

Paint.net version v3.3.6.3158.38068

Scripting

TinyXML 2.5.3

Timing Specifications

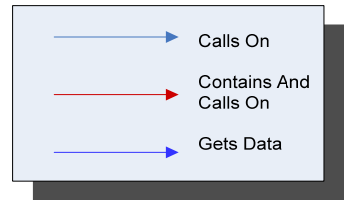
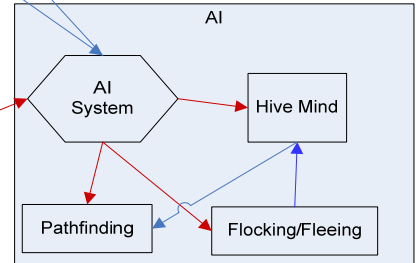
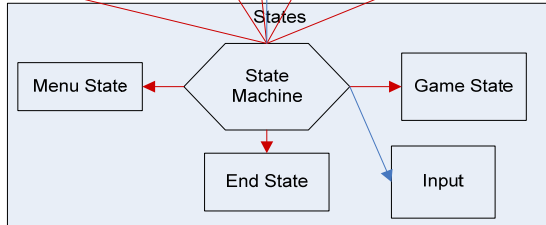
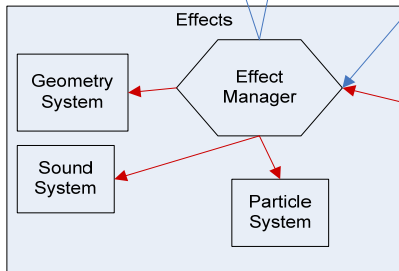
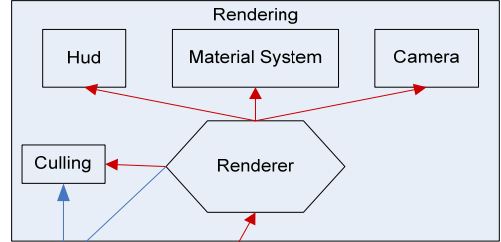
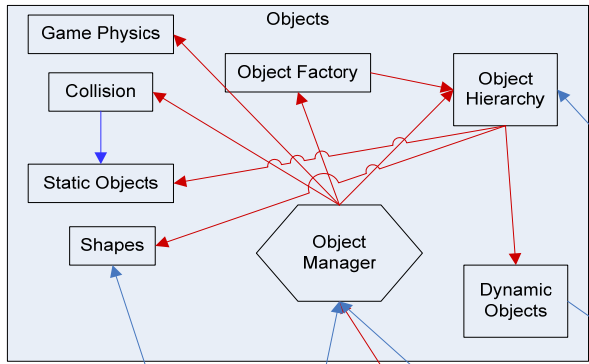
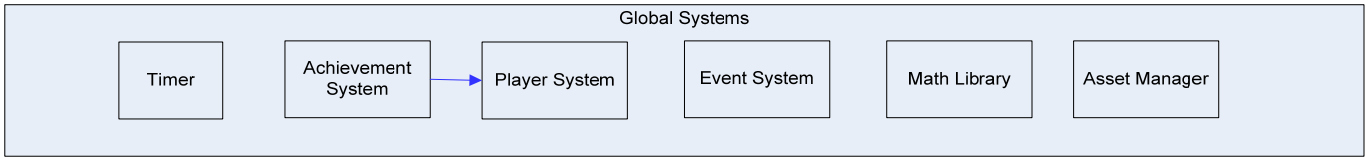
Super Power Surge Spike is required to run at least 60 Frames a second during all game play situations. During each frame or 1/60th of a second each of these pieces of technology will be using approximately this much % of this Time resource.

TECHNOLOGY NEEDED FOR GAMEPLAY	PERCENTAGE OF FRAME NEEDED
ENEMY AI	10%
SMOOTH SKIN AND BONE ANIMATION UPDATE	12.5%
HUD UPDATE	2%
HUD RENDER	2%
OBJECT UPDATE	10%
OBJECT COLLISION DETECTION	15%
PARTICLES/EFFECTS UPDATE	5%
CULLING	7.5%
OBJECT GEOMETRY/EFFECT RENDERING	30%
INPUT	2%
SOUNDS	4%

System Architecture

Description

To allow for multiple 'arena' levels we will only be loading the currently applicable level at the level load screen and then releasing them after the end of the level. This will allow us to keep our memory stamp lower and will give us a chance to load more textures into memory. We also intend to avoid using the singleton design pattern because it can introduce unexpected and hard to trace problems into a code base. To keep global systems without using a singleton we will declare all globals in main.cpp and extern them to the defines.h file which is included in the rest of the code base. One of our most important modules is the FXManager. The FXManager itself is nothing more than a façade to abstract out each of the different effect systems: sound, geometry, shaders, and particles. This manager will give us an interface to play an effect wholly rather than having to interact with each effect module directly.



Context Model Description

- Timer
 - Accessed By
 - Global
 - Functions
 - Init()
 - TimeStep()
 - GetTimer()
 - PauseTimer()
 - PlayTimer()
 - CreateTimer()
 - DiscardTimer()
 - SetTimer()
 - Update()

- Input
 - Accessed By
 - State Machine
 - Functions
 - GetInstance()
 - InitDevices()
 - InitKeyboard()
 - InitMouse()
 - ShutdownDirectInput()
 - ReadDevices()
 - ReadKeyboard()
 - ReadBufferedKeyboard()
 - GetKey()
 - CheckKeys()
 - CheckBufferedKeys()
 - GetBufferedKey()
 - GetBufferedKeyEx()
 - ReadMouse()
 - ReadBufferedMouse()
 - OnMouseButtonRelease()
 - GetMouseButton()
 - GetBufferedMouseButton()
 - GetMouseAxis()

- Player System
 - Accessed By
 - Global

- Functions
 - Init()
 - Shutdown()
 - Update()
 - AddScore()
 - SetScore()
 - GetScore()
 - SetHealth()
 - GetHealth()
 - ModifyHealth()
 - SetPrimaryEnergy()
 - GetPrimaryEnergy()
 - SetSecondaryEnergy()
 - GetSecondaryEnergy()
 - EventProc()

- Achievement System
 - Access To
 - Player Manager
 - Functions
 - AddScore()
 - Accessed By
 - Global
 - Functions
 - Init()
 - Update()
 - Shutdown()
 - Update()
 - GetEarnedAchievements()
 - EventProc()

- Math Library
 - Accessed By
 - Global
 - Functions
 - AngleBetween()

- Event System
 - Accessed By
 - Global
 - Functions
 - GetInstance()
 - Shutdown()
 - RegisterClient()

- SendEvent()
 - ProcessEvents()
 - UnregisterClient()
 - ClearEvents()
- Asset Manager
 - Accessed By
 - Global
 - Functions
 - Init()
 - Shutdown()
 - LoadAsset()
 - GetAsset()
 - ReleaseAsset()
 - ReleaseAll()
 - EventProc()
 - ReleaseAllAssetType()
- State Machine
 - Access To
 - FXManager
 - Functions
 - Init()
 - Shutdown()
 - AI System
 - Functions
 - Init()
 - Shutdown()
 - Update()
 - Object Manager
 - Functions
 - AddStaticObject()
 - AddDynamicObject()
 - GetStaticObject()
 - GetDynamicObject()
 - RemoveStaticObject()
 - RemoveDynamicObject()
 - Update()
 - Sound System
 - Functions
 - Init()
 - ShutDown()
 - Accessed By
 - WinMain

- Functions
 - Init()
 - Update()
 - Shutdown()
 - Game States
 - Functions
 - GetInput()
 - ChangeState()
 - Animation System
 - Access To
 - ObjectManager
 - Functions
 - SetBoneMatrixPointers()
 - FXManager
 - Functions
 - GetKeyframes()
 - GetCurrentKeyframe()
 - Access By
 - State Machine
 - Functions
 - AddTime()
 - SetTime()
 - Process()
 - Init()
 - Shutdown()
 - Renderer
 - Functions
 - Draw()
 - DrawFrame()
 - Object Manager
 - Access To
 - Physics
 - Functions
 - Apply()
 - Collision
 - Functions
 - Detect()
 - React()
 - Object Factory
 - Functions
 - Init()
 - Shutdown()
 - GetStaticObject()

- GetDynamicObject()
 - ReturnStaticObject()
 - ReturnDynamicObject()
 - Object Hierarchy
 - Functions
 - Update()
 - Render()
 - Accessed By
 - State Machine
 - Functions
 - AddStaticObject()
 - AddDynamicObject()
 - GetStaticObject()
 - GetDynamicObject()
 - RemoveStaticObject()
 - RemoveDynamicObject()
 - Update()
- Object Factory
 - Access To
 - Object Hierarchy
 - Just for new/delete and enum of types
 - Accessed by
 - Object Manager
 - Functions
 - Init()
 - Shutdown()
 - GetStaticObject()
 - GetDynamicObject()
 - ReturnStaticObject()
 - ReturnDynamicObject()
- Physics
 - Access to
 - Object Hierarchy
 - Object Types
 - Accessed By
 - Object Manager
 - Functions
 - Apply()
 - Object Hierarchy
 - Functions
 - Apply()
 - Collision

- Functions
 - Apply()
- Collision
 - Access to
 - Object Hierarchy
 - Object Types
 - Accessed By
 - Object Manager
 - Functions
 - Detect()
 - React()
- Shapes
 - Accessed By
 - FXManager
 - Gets Object geometry
 - Collision
 - Gets Object geometry
- Object Hierarchy
 - Access To
 - Physics
 - Functions
 - Apply()
 - Collision
 - Functions
 - Detect()
 - React()
 - AI System
 - GetNextPoint()
 - FXManager
 - Accessed By
 - Object Manager
 - Functions
 - Update()
 - Render()
 - Object Factory
 - Just for new/delete and enum of types
- Renderer
 - Access To
 - Animation System
 - Functions

- Draw()
 - DrawFrame()
- HUD
 - Functions
 - Render()
- Material System
 - Functions
 - Clear()
 - LoadScript()
 - Update()
 - Blend()
 - UpdateDynamicMaterials()
 - IsDynamic()
 - IsTransparent()
 - GetMergeMode()
 - PreRender()
 - PostRender()
- Camera
 - Gets the cameras view matrix
- Culling
 - Functions
 - GetVisibleSet()
- Accessed By
 - State Machine
 - Functions
 - Init()
 - MakeFullscreen()
 - SetColorMask()
 - ClearBackbuffer()
 - ClearZBuffer()
 - ClearBuffers()
 - ClearBackbuffer()
 - LoadFont()
 - UnloadFont()
 - WriteFont()
 - SetPointSize()
 - PushMaterial()
 - PopMaterial()
 - GetMaterialSize()
 - Draw()
 - Draw()
 - Draw()

- Precompile()
 - Shutdown()
 - BeginScene()
 - EndScene()
 - GetCamera()
 - SetAnimationSystem()
 - LoadShader()
 - UnloadShader()
 - UnloadAllShaders()
- Culling
- Accessed By
 - Renderer
 - Functions
 - GetVisibleSet()
 - State Machine
 - Functions
 - Set()
 - Update()
- HUD
 - Access To
 - Object Manager
 - Functions
 - GetDynamicObject()
 - Accessed By
 - Renderer
 - Functions
 - Render()
 - State Machine
 - Functions
 - Init()
 - Update()
 - Shutdown()
- Material System
 - Accessed By
 - Renderer
 - Functions
 - Clear()
 - LoadScript()
 - Update()
 - Blend()
 - UpdateDynamicMaterials()
 - IsDynamic()

- IsTransparent()
 - GetMergeMode()
 - PreRender()
 - PostRender()
- Camera
 - Accessed By
 - Renderer
 - Gets the cameras view matrix
 - StateMachine
 - Gets the cameras view matrix
 - Functions
 - UpdateBasedOnAvatar()
 - SwoopInOnAvatar()
 - Collision
 - Gets the cameras model matrix
- AI System
 - Access To
 - Asset Manger
 - Functions
 - LoadAsset()
 - GetAsset()
 - Object Hierarchy
 - Gets static objects
 - Flocking/Fleeing
 - Functions
 - Init()
 - Update()
 - Shutdown()
 - GetCurrentLocation()
 - Pathfinding
 - Functions
 - Init()
 - Shutdown()
 - GetPath()
 - LevelLoad()
 - HiveMind
 - Functions
 - Init()
 - Shutdown()
 - GetPath()
 - Update()
 - Accessed By
 - Object Hierarchy

- Functions
 - GetNextPoint()
 - State Machine
 - Functions
 - Init()
 - Shutdown()
 - Update()
 - Pathfinding
 - Accessed By
 - AI System
 - Functions
 - Init()
 - ShutDown()
 - LevelLoad()
 - Hive Mind
 - Functions
 - GetPath()
 - Flocking/Fleeing
 - Access To
 - Hive Mind
 - Functions
 - GetEnemy()
 - GetEnemyCount()
 - GetGroup()
 - GetGroupCount()
 - Accessed By
 - AI System
 - Functions
 - Init()
 - ShutDown()
 - GetCurrentLocation()
 - Update()
 - Hive Mind
 - Access To
 - Pathfinding
 - Functions
 - GetEnemy()
 - GetEnemyCount()
 - GetGroup()
 - GetGroupCount()
 - - Accessed By

- AI system
 - Functions
 - Init()
 - Shutdown()
 - GetEnemy()
- FXManager
 - Access To
 - Geometry Effects Manager
 - Functions
 - Init()
 - Shutdown()
 - CreateEffet()
 - GeomSystem()
 - Sound Manager
 - Functions
 - PlaySound()
 - StopSound()
 - ResetSound()
 - Distort()
 - Play3DSound()
 - LoadSound()
 - UnloadSound()
 - Particle Manager
 - Functions
 - Init()
 - Shutdown()
 - CreateEmitter()
 - Object Manager
 - Functions
 - AddStaticObject()
 - AddDynamicObject()
 - Accessed By
 - State Machine
 - Functions
 - Init()
 - Shutdown()
 - StartEffect()
 - GetParticleSys()
 - GetGeomSys()
 - GetAudioSys()
 - Animation System
 - Functions
 - StartEffect()

- Geometry Effects Manager
 - Access To
 - Object Manager
 - Functions
 - AddStaticObject()
 - AddDynamicObject()
 - Accessed By
 - FXManager
 - Functions
 - Init()
 - Shutdown()
 - CreateEffet()
 - GeomSystem()
- Sound System
 - Accessed By
 - StateMachine
 - Functions
 - Init()
 - ShutDown()
 - FXManager
 - Functions
 - PlaySound()
 - StopSound()
 - ResetSound()
 - Distort()
 - Play3DSound()
 - LoadSound()
 - UnloadSound()
- Particle Manager
 - Access To
 - ObjectManager
 - Functions
 - AddStaticObject()
 - AddDynamicObject()
 - Accessed By
 - FXManager
 - Functions
 - Init()
 - Shutdown()
 - CreateEmitter()

Module Feature Breakdown

Timer

This module is responsible for centralizing the game timing mechanisms used throughout the program. This includes the master time step and manages individual countdown timers. The system is global, and is accessed through its singleton interface. This system was made by James Leonis and edited slightly by Joshua Bennett. Due to strange errors needs to use doubles.

Dependencies

- Accessed by:
 - Global access

Functions

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INIT	VOID	STARTS THE SYSTEM.
VOID	SHUTDOWN	VOID	SHUTS DOWN AND FREE'S MEMORY.
DOUBLE	TIMESTEP	VOID	RETURNS THE AMOUNT OF TIME SINCE THE LAST FRAME IN SECONDS.
DOUBLE	GETTIMER	INT NID	RETURNS THE CURRENT TIME LEFT ON HTIMER IN SECONDS
BOOL	PAUSETIMER	INT NID	PAUSES HTIMER
BOOL	PLAYTIMER	INT NID	RESUMES HTIMER
INT	CREATETIMER	INT NTIME	CREATES A COUNTDOWN TIMER AND RETURNS A REFERENCE HANDLE WITH NTIME AMOUNT OF SECONDS
BOOL	DISCARDTIMER	INT NID	REMOVES HTIMER FROM THE LIST OF TIMERS
BOOL	SETTIMER	INT NID INT NTIME	SETS HTIMER TO NTIME SECONDS
BOOL	UPDATE	VOID	UPDATES THE GLOBAL TIME STEP AND THE INDIVIDUAL TIMERS.

Features

- Is a centralized source for the game's master time step
- Can create arbitrary countdown timers for use with game specific content

Countdown Timer structure

The countdown timers will be stored in a vector array inside the game timer. Individual timers will be accessible through handles. The game timer's Update function also updates these individual timers

```
struct tTimer
{
    Double dTimeLeft; // in seconds
    Float dStartTime; // if < INACTIVE_TIMER then the timer is inactive
};
```

Time to Complete Estimate

- Already done for our project.

Module Author

James Leonis

Input

This module is responsible for handling the initialization and shutdown of Direct Input. It will also handle the storing of mouse and keyboard devices and the polling of these devices as well.

Dependencies

- Accessed by:
 - StateMachine

<u>Return</u>	<u>Name</u>	<u>Parameters</u>	<u>Description</u>
Input*	GetInstance	None	Returns an instance to the DirectX Input wrapper.
Bool	InitDevices	HWND hWnd, HINSTANCE hMainInstance, Bool bExclusive	Initializes all of the devices for Direct Input
Bool	InitKeyboard	HWND hWnd, HINSTANCE hMainInstance, Bool bExclusive	Initialize the keyboard for use with Direct Input
Bool	InitMouse	HWND hWnd, HINSTANCE hMainInstance, Bool bExclusive	Initialize the mouse for use with Direct Input
Void	ShutdownDirectInput	None	Shuts down all Direct Input components
Bool	ReadDevices	None	Gets the state of all the acquired devices.

Bool	ReadKeyboard	None	Gets the keyboard state every frame
Bool	ReadBufferedKeyboard	None	Gets the buffered keyboard data every frame
Bool	GetKey	UCHAR ucKey	Gets the current immediate state of a key
Char	CheckKeys	None	Checks to see what exact button the user pressed.
Char	CheckBufferedKeys	None	Checks to see what exact button the user pressed, but with buffered input
Bool	GetBufferedKey	UCHAR ucKey	Gets the status of a key using buffered input
Bool	GetBufferedKeyEx	UCHAR ucKey	Gets a buffered key without using Direct Input's buffered input
Bool	ReadMouse	None	Gets the mouse state every frame

Bool	ReadBufferedMouse	None	Get the buffered mouse data every frame
Bool	OnMouseButtonRelease	BYTE mButton	Check and see if the mouse button specified was released
Bool	GetMouseButton	BYTE mButton	Gets the current immediate state of a mouse button
Bool	GetBufferedMouseButton	BYTE mButton	Gets the status of a mouse button using buffered input.
LONG	GetMouseAxis	LONG IAxis	Gets the status of a mouse axis

Features

- Wrapper class that contains needed functionality for Direct Input

Time to Complete Estimate

- Implementation
 - 1 day

Module Authors

Dave Brown / Jensen Rivera - Original Authors
Philip Fox – Modified

Player System

The Player Manger is a data holding class with information of the players score, health, energy levels and if shield is active or not. Most of the Player Managers job is to handle the score and raise it depending on kills, also handles the combo system.

Dependencies

- Accessed by:
 - Global

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INIT	VOID	STARTS THE SYSTEM.
VOID	SHUTDOWN	VOID	SHUTS DOWN AND FREE'S MEMORY.
VOID	UPDATE	VOID	UPDATES ALL TIMERS AND DETERMINES THE SCORE.
VOID	ADDScore	INT NScore – THE EXTRA SCORE TO ADD TO THE TOTAL.	ADDS TO THE TOTAL SCORE.
VOID	MODIFYHEALTH	INT NHEALTH – ALTERS THE CURRENT HEALTH OF THE PLAYER.	
VOID	HANDLEEVENT	CEVENT *PEVENT – HOLDS A CLASS CONTAINING MESSAGE TYPE AND MESSAGE PARAMETER	PROCESSES AN EVENT MESSAGE. (PLAYER_HIT, PLAYER_DEATH, ENEMY_DEATH)
BOOL	FIREPRIMARYATTACK	ATTACK*& RETURN, INT& TYPE	IF THE ATTACK FIRES OFF SUCCESSFULLY THEN IT RETURNS THE ATTACK AND THE TYPE OF THE ATTACK
BOOL	FIRESECONDARYATTACK	ATTACK*& RETURN INT& TYPE	IF THE ATTACK FIRES OFF SUCCESSFULLY THEN IT RETURNS THE ATTACK AND THE TYPE OF THE ATTACK

BOOL	DRAINPRIMARY	POWERNODE*	ATTEMPTS TO DRAIN A POWERNODES ENERGY INTO THE PRIMARY AMMO SLOT
BOOL	DRAIN SECONDARY	POWERNODE*	ATTEMPTS TO DRAIN A POWERNODES ENERGY INTO THE SECONDARY AMMO SLOT
BOOL	SWAPABILITIES	VOID	SWAPS THE AMMO, AND TYPE OF AMMO BETWEEN PRIMARY AND SECONDARY

Features

- Storage of player information
- Manages player score

Time to Complete Estimate

- Basic Player System
 - 2 days
- Detailed Player System
 - 2 days
- Scoring
 - 4 days

Total: 8 days

Module Authors

Phil Fox – System Head

Joshua Bennett

Achievements

Achievements are handled in events or by drawing information from the Player Manager. The Achievements section of the menu keeps track of what achievements have been earned.

List of achievements (subject to change) - Enum type follows achievement description:

11. Tier three: Get a multiplier of 8 (MULTIPLIER_EIGHT)
12. Tier three: Get a multiplier of 15 (MULTIPLIER_FIFTEEN)
13. Should Have Gone Linux: Beat the game (BEAT_GAME)
14. Power User: Win under ten minutes (BEAT_FAST)
15. One Track Mind: Use only two elements to beat a level (TWO_ELEMENTS)
16. OverKill: Only use secondary attacks in a level (SECONDARY_ATTACKS)
17. Skill Shot: Never hit a enemy with a element they are strong too (NO_STRONG)
18. Evasive: Beat a level while not taking a single hit (NO_HIT)
19. Bit: Kill ten enemies (KILL_TEN)
20. Byte: Kill 100 enemies (KILL_ONE_HUNDRED)

Dependencies

- Access to:
 - Player Manger
- Accessed by:
 - Global

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INIT	VOID	STARTS THE SYSTEM.
VOID	SHUTDOWN	VOID	SHUTS DOWN AND FREE'S MEMORY.
VOID	UPDATE	VOID	CHECKS ALL EVENTS THAT HAVE HAPPENED AND UPDATES COUNTERS AS NEEDED.
BOOL* - AN ARRAY OF BOOLEAN VALUES	GETEARNEDACHIEVEMENTS	VOID	RETURNS AN ARRAY OF BOOL'S, EACH ACHIEVEMENT IS REPRESENTED BY A BOOL AND ITS LOCATION IS FOUND THROUGH AN ENUM.
VOID	SETACHIEVEMENT	INT NTOUNLOCK – A ENUM OF THE ACHIEVEMENT TO UNLOCK. BOOL BTOSET – IF THE ACHIEVEMENT IS ON OR NOT.	PASSING IN A ENUM WILL TRIP THE GIVEN ACHIEVEMENT TO EQUAL THE PASSED IN BOOL.
VOID	HANDLEEVENT	CEVENT *PEVENT – HOLDS A CLASS CONTAINING MESSAGE TYPE AND MESSAGE PARAMETER	PROCESSES AN EVENT MESSAGE. (PLAYER_HIT, PLAYER_DEATH, ENEMY_HIT, ENEMY_DEATH, BOSS_HIT, BOSS_DEATH, PLAYER_MOVE, PLAYER_JUMP)

Features

- Lists achievements and stores them.

Time to Complete Estimate

- 2 days

Module Author

Joshua Bennett

EventDispatcher

This module controls the communication between all objects and modules in our game. Objects or modules will register with this dispatcher and when an event happens for a specific object or module then the dispatcher will send that event out. GetInstance must be called to get access to the module and Shutdown must be called to clean up any memory allocated.

Dependencies

- Accessed by:
 - Global

<u>Return</u>	<u>Name</u>	<u>Parameters</u>	<u>Description</u>
Dispatcher *	GetInstance	None	Returns an instance to the dispatcher to access the interface.
void	Shutdown	None	Shuts down the Dispatcher
void	RegisterClient	EVENTID eventID IListener *pClient	Registers a client to receive events from the dispatcher.
void	SendEvent	EVENTID eventID void *pParam	Sends an event to a client registered with the eventID, will also include the void * data passed in as well.
void	ProcessEvents	None	Processes all the events in the queue.
void	UnregisterClient	IListener *pClient	Unregisters a client from receiving events from the dispatcher. (Refer to IListener sub-module).
void	ClearEvents	None	Clears the event vector out, useful when you're exiting game state.

Events

STATE_SWITCH

ATTACK

PLAYER_DEATH

ENEMY_DEATH

GAME_SHUTDOWN

PLAYER_HIT

ENEMY_HIT

PLAYER_JUMP

PLAYER_MOVE

BOSS_HIT

BOSS_DEATH

ACHIEVEMENT_UNLOCK

Features

- Allows the communication between classes without having them directly talk to each other.
- Great debug feature as it allows one location where events for the whole game can be viewed.

Time to Complete Estimate

- Implementation
 - 1 day
- Testing and Integration
 - 1 day

Total: 2 days

Module Authors

Jensen Rivera - Original Author

Philip Fox – Modified

Sub Module

IListener

Interface class that is used with the event system.

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	HANDLEEVENT	CEVENT *PEVENT – HOLDS A CLASS CONTAINING MESSAGE TYPE AND MESSAGE PARAMETER	FUNCTION THAT WILL HANDLE ANY EVENTS SENT TO A CLASS THAT DERIVES FROM THIS INTERFACE

Time to Complete Estimate

- Event System
 - 2 days

Module Author

Philip Fox

Sub Module

CEvent

This class contains all of the functionality for any event sent through the Event system

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	SETEVENTPARAMS	STRING EVENTID VOID * PPARAM	SET PARAMETERS TO SEND THROUGH THE EVENT SYSTEM.
VOID	SETEVENTID	STRING EVENTID	MUTATOR TO ASSIGN AN EVENT ID
VOID	SETPARAM	VOID * PPARAM	MUTATOR TO SET A PARAMETER FOR THIS EVENT
STRING	GETEVENTID	NOTHING	ACCESSOR TO RETRIEVE THE EVENT ID OF THIS EVENT.
VOID *	GETPARAM	NOTHING	ACCESSOR TO RETRIEVE THE PARAMETER SET FOR THIS EVENT.

Time to Complete Estimate

- Event System
 - 2 days

Module Author

Philip Fox

AssetManager

The Asset Manger handles loading and storing of all file input/output. The Asset Manger will handling loading the following: models, textures, levels, AI waypoints, node placement/type, and all scripting data. Scripts are in XML format and read in using tinyXML.

Scripting is done for the following:

- Enemy values (HP, speed, etc)
- Player Avatar values (HP, speed, etc)
- Node values (recharge rate, total energy stored, etc)
- High score loading and saving
- Material system (Not using XML)
- Menu scripting
- Enemy spawning (everything but the location)

Dependencies

- Accessed by
 - Global

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INIT	VOID	STARTS THE SYSTEM.
VOID	SHUTDOWN	VOID	SHUTS DOWN AND FREE'S MEMORY.
INT – RETURNS AN ID HANDLE TO THE ASSET.	LOADASSET	INT NTYPE – A ENUM THAT HOLDS THE TYPE OF FILE. CONST CHAR * PFILENAME – THE NAME OF THE FILE STORED IN ASSETS. DWORD OPTIONS – A DEFAULTED VALUE, FOR USE WITH MESHES ONLY.	LOADS AN ASSET FROM THE HARD DRIVE AND STORES FOR LATER USE.
VOID * - RETURNS THE OBJECT REQUESTED FROM THE ID PASSED IN.	GETASSET	INT NID – THE ID VALUE OF AN ASSET THAT WAS GIVEN OUT FROM THE LOADASSET FUNCTION. INT NTYPE – A ENUM THAT HOLDS THE TYPE OF FILE.	TAKES IN A ASSET ID AND RETURNS A POINTER TO IT IN THE FORM OF A VOID POINTER.
VOID	RELEASEASSET	INT NID – THE ID VALUE OF AN ASSET THAT WAS	REMOVES THE ASSET FROM

		GIVEN OUT FROM THE LOADASSET FUNCTION.	MEMORY.
		INT NTYPE – A ENUM THAT HOLDS THE TYPE OF FILE.	
VOID	RELEASEALL	VOID	REMOVES ALL ASSETS FROM MEMORY.
VOID	RELEASEALLASSETTYPE	INT NTYPE – A ENUM THAT HOLDS THE TYPE OF FILE.	RELEASES ALL OF A GIVEN ASSET TYPE FROM MEMORY.
VOID	EVENTPROC	INT NEVENTTYPE – THE TYPE OF MESSAGE. VOID * PPARAMS – ANY EXTRA CONTENT IN THE MESSAGE.	PROCESSES AN EVENT MESSAGE.

Enum File Types

STATICX_FILE
 ANIMATEX_FILE
 PNG_FILE
 OGG_FILE
 XML_SCRIPT_SPAWN
 XML_SCRIPT_ENEMYS
 XML_SCRIPT_PLAYER
 XML_SCRIPT_NODES
 XML_SCRIPT_HIGHSCORE
 XML_SCRIPT_MENU
 MAT_SCRIPT

Features

- Storage of all assets
- Templates for objects
- Simple and clean interface

Enemy Script Example

```

<Enemies>
  <Fire_Enemy>
    <HP>50</HP>
    <Speed>5</Speed>
    <Weakness>Ice</Weakness>
  
```

```
    </Fire_Enemy>
    <Ice_Enemy>
    .....
</Enemies>
```

Player Script Example

```
<Player>
  <Speed>5</Speed>
  <Jump_Hight>3</Jump_Hight>
  <Easy>
    <HP>10</HP>
  </Easy>
  <Medium>
    <HP>5</HP>
  </Medium>
  <Hard>
    <HP>1</HP>
  </Hard>
</Player>
```

Node Values

```
<Nodes>
  <Recharge_Time>10</Recharge_Time>
  <Node_Charge>100</Node_Charge>
  <Node_Charge_Distance>25</Node_Charge_Distance>
</Nodes>
```

High Score Script Example

```
<High_Scores>
  <Main_Scores>
    <First>10000</First>
    <Second>5000</Second>
    .....
    <Tenth>1</Tenth>
  </Main_Scores>
  <Survival_Scores>
    <First>10000</First>
    <Second>5000</Second>
    .....
    <Tenth>1</Tenth>
  </Survival_Scores>
  <Evasion_Time>
    <First>460</First>
    <Second>50</Second>
    .....
    //Stored in seconds
```

```
        <Tenth>2</Tenth>
    </Evasion_Time>
    <Time_Attack>
        <First>400</First>                //Stored in seconds
        <Second>500</Second>
        .....
        <Tenth>999999999</Tenth>
    </Time_Attack>
</High_Scores>
```

Material System Script Example

//This is how to comment

Pass

```
{
    ambient = ( 0.5, 0.4, 0.7, 1.0 )
    diffuse = ( 0.8, 0.2, 0.3, 1.0 )
    specular = ( 0.9, 0.7, 0.4, 1.0 )
    shininess = 30.0
    Texture
    {
        "asset/2d/diffuse.png"
    }
}
```

Enemy Spawn Scripting

```
<Level_One>
  <Wave_One>
    <Wave_Timer>50</Wave_Timer>
    <Squad_One>
      <Type>Fire</Type>
      <Number>5</Number>
    </Squad_One>
    <Squad_Two>
      <Type>Ice</Type>
      <Number>5</Number>
    </Squad_Two> >5</Number>
  </Wave_One>
  <Wave_Two>
    <Wave_Timer>60</Wave_Timer>
    <Squad_One>
      .....
  </Wave_two>
  .....
</Level_One>
<Level_Two>
.....
<Level_Three>
...
<Boss>
  <State_One>
    <Squad_One>
      <Type>Fire</Type>
      <Number>5</Number>
    </Squad_One>
    <Squad_Two>
      <Type>Ice</Type>
      <Number>5</Number>
    </Squad_Two> >5</Number>
  </State_One>
  <State_Two>
    <Squad_One>
      .....
    </Squad_Four>
  </State_Two>
.....
</Boss>
```

Time to Complete Estimate

- Asset Manager
 - 2 days
- Model Loader
 - 1 day
- Level Loading
 - 2 days
- Scripting
 - Enemy Spawning
 - 3 days
 - Texture Loading
 - 1 day
 - Materials
 - 5 days
- Animation Loading
 - 1 day

Total: 14 days

Module Authors

Joshua Bennett – System Head
Justin Morgan

StateMachine

This module is responsible for switching between the many different states our game will go through. Internally the State Machine will hold objects for the different states. The main states of our game are the splash screen state, menu state, loading state, main game state, death state, and game win state.

Dependencies

- Accessed by:
 - Main game engine

<u>Return</u>	<u>Name</u>	<u>Parameters</u>	<u>Description</u>
void	Init	None	This function will initialize the state machine as well as any states that need to be initialized, example: Splash screen state
void	Update	Float fDelta	This function will update the current game state with the passed in time slice.
void	Input	None	Processes input for the current game state.
void	Render	None	Renders the current game state.
void	Shutdown	None	Shuts down the state machine all states associated with the game.
void	ChangeState	CGameStateInterface * pVal	Changes the state of the game state machine, this function will call the current states exit function, switches the state, and then calls the new states enter function.

Features

- Gives an organized approach to managing the multiple states our game will go through.
- Eliminates spaghetti code by giving one location to make changes to each separate state.

- **Time to Complete Estimate**
 - Game State
 - 2 days
 - Menu State
 - 1 day
 - End State
 - 1 day
- Total: 4 days

Module Author

Philip Fox

Sub Module

CGameStateInterface

This module is an interface for the different states in our game.

Dependencies

- Accessed by:
 - StateMachine

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INIT	NONE	THIS FUNCTION WILL INITIALIZE THE STATE MACHINE AS WELL AS ANY STATES THAT NEED TO BE INITIALIZED, EXAMPLE: SPLASH SCREEN STATE
VOID	UPDATE	FLOAT FDELTA	THIS FUNCTION WILL UPDATE THE CURRENT GAME STATE WITH THE PASSED IN TIME SLICE.
VOID	INPUT	NONE	PROCESSES INPUT FOR THE CURRENT GAME STATE.
VOID	RENDER	NONE	RENDERS THE CURRENT GAME STATE.
VOID	SHUTDOWN	NONE	SHUTS DOWN THE STATE MACHINE ALL STATES ASSOCIATED WITH THE GAME.

Time to Complete Estimate

- Game State
 - 2 days

Module Author

Philip Fox

AnimationSystem

This system manages all animations in the game. It requires you to init the DirectX animation controller with the Init function. To properly close the animation system you must call Shutdown. On top that this system is dependant on DirectX being already initialized, so the renderer must be running before this can be initialized.

Dependancies

- Access To:
 - ObjectManager
 - FXManager
- Accesed By:
 - Renderer
 - StateMachine

RETURN	NAME	PARAMETER S	DESCRIPTIO N
VOID	SETBONEMATRIXPOIN TERS	LPD3DXFRA ME PFRAME	THIS FUNCTION SETS UP THE POINTERS FOR A GIVEN BONE TO ITS MATRIX
VOID	SETANIMATION	INT NANIMATION SET	THIS FUNCTION TAKES IN AN ID TO CHANGE WHICH ANIMATION IS ACTIVE.
INT	GETANIMATION	VOID	THIS FUNCTION RETURNS AN ID TO THE CURRENTLY ACTIVE

VOID	INIT	VOID	ANIMATION. THIS FUNCTION INITIALIZES THE ANIMATION SYSTEM.
VOID	SHUTDOWN	VOID	THIS FUNCTION SHUTS DOWN THE ANIMATION SYSTEM.
VECTOR<KEYFRAME*>	GETKEYFRAMES	INT ANIMATION SET	RETURNS THE ANIMATIONS KEYFRAMES
VOID	ADDTIME	FLOAT FTIME INT ANIMATION SET	THIS FUNCTION ADDS TIME TO A SPECIFIED ANIMATION
VOID	SETTIME	FLOAT FTIME INT ANIMATION SET	THE FUNCTION SETS THE TIME ON A SPECIFIED ANIMATION.
VOID	PROCESS	NANIMATION SET	INTERPOLATES BETWEEN KEYFRAMES IN THE SPECIFIED ANIMATION
CONST KEYFRAME*	GETCURRENTKEYFRAME	NANIMATION SET	RETURNS THE CURRENT KEYFRAME FOR THE SPECIFIED ANIMATION.
VOID	DRAW	LPD3DXFRAME PFRAME	DRAWS THE CURRENT

INT
ANIMATION
SET

SPECIFIED
FRAME.

Features

- This system handles all the separate animations and unifies them to one interface.
- This system handles effect triggers on proper frames to handle timing issues.

Associated Risks

Risk:	Affected Resource:	P	C	RF
The animation system, FXManager, and player might have difficulty timing correctly. Due to the fact that a callback or event will have to trigger all three, and even if only one of them is a little bit off it will hurt the immersion of the game.	Justin Morgan, Tech Lead	.8	.2	0.84
	Response or avoidance: Consult internet forums, articles, and books before any coding happens. Seek help from other classmates working on animation. Seek help from GP staff or Shawn Kendell on animation techniques. If timing takes cant be fixed by a week after beta we will give up on the trying to fix it and deal with the timing being slightly off. This will be enacted a week after beta.			

Time to complete estimate

- Animation Shader
 - 3 days
- Interpolation
 - 2 days
- Animation Manager
 - 3 days
- Keyframe Management
 - 1 day
- Animation Blending
 - 5 days

Total: 14 days

Module Author

Justin Morgan

ObjectManager

The Object Manager is responsible for Updating and Processing the Objects and their interactions in the Super Power Surge Spike world. This Module will be using functionality defined in the Collision Detection, Game Physics and Object Hierarchy Modules.

Dependencies

- Access to:
 - Physics
 - Collision
 - Object Factory
 - Object Hierarchy
- Accessed by:
 - State Machine
 - ObjectManager
 - FXManager

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	ADDSTATICOBJECT	BASEOBJECT*	ADDS THE STATIC OBJECT TO THE MODULE
VOID	ADDDYNAMICOBJECT	BASEOBJECT*	ADDS THE DYNAMIC OBJECT TO THE MODULE
BASEOBJECT*	GETSTATICOBJECT	INT NID – THE ID ASSOCIATED WITH THE POSITION IN THE ACTIVE OBJECTS VECTOR.	RETURNS THE STATIC OBJECTS MEMORY BASED ON ITS ID IN THE MODULE.
BASEOBJECT*	GETDYNAMICOBJECT	INT NID – THE ID ASSOCIATED WITH THE POSITION IN THE ACTIVE OBJECTS VECTOR.	RETURNS THE DYNAMIC OBJECTS MEMORY BASED ON ITS ID IN THE MODULE
VOID	REMOVESTATICOBJECT	INT NID – ID ASSOCIATED WITH THE OBJECT IN THE MODULE	REMOVES THE STATIC OBJECT BASED ON ITS ID NUMBER PASSED IN
VOID	REMOVEDYNAMICOBJECT	INT NID – ID ASSOCIATED WITH	REMOVES THE DYNAMIC OBJECT

		THE OBJECT IN THE MODULE	BASED ON ITS ID NUMBER PASSED IN
VOID	CULLING	VOID	DECIDES WHAT 1OBJECTS GET RENDERED TO SCREEN
VOID	ADDLEVEL	INT ID	LOADS A LEVEL BASED ON THE ID PASSED IN
VOID	CAMCOLLISION	VOID	DETECTS AND RESPONDS THE CAMERA IN THE WORLD TO MAKE SURE IT DOESN'T GET BLOCK BY ANYTHING IN THE WORLD
VOID	UPDATE	VOID	UPDATES ALL NECESSARY OBJECTS
VOID	COLLISIONANDRESPONSE	VOID	CALLS ALL THE COLLISION FUNCTIONS AND MAKES SURE THE PROPER RESPONSE HAPPENS
VOID	HANDLEEVENT	CEVENT *PEVENT – HOLDS A CLASS CONTAINING MESSAGE TYPE AND MESSAGE PARAMETER	RESPONDS TO ADDING OR REMOVING OBJECTS FROM THE SYSTEM. (PLAYER_DEATH, ENEMY_DEATH, BOSS_DEATH)

Features

- Managing the objects, They will respond and interact towards our games goal

Time to Complete Estimate

- 1 day

Module Author

- Tim Turcich

Sub Module

Object Hierarchy

The Object hierarchy is a collection of all the different types of objects in Super Power Surge Spikes gameplay. These objects are managed by the object manager and have functionality within their updates and render functions specific to how themselves.

The Structure is as Follows:

- BaseObject – has vector of emitters, collision shape, world matrix, object type id, and Object3D.
 - StaticObject
 - LevelGeonmetry
 - PowerNodes
 - Platforms
 - FlatObject
 - DynamicObject
 - Avatar – Player driven avatar - Alice
 - Enemy – Base class for the four different types of enemies
 - FireEnemy
 - IceEnemy
 - ElectricEnemy
 - WindEnemy
 - Attack – base attack for all the attack in the game
 - FireBehavior – defines fire behavior/collisions
 - FirePrimary
 - IceandFire
 - ElectricandFire
 - WindandFire
 - IceBehavior – defines Ice behavior/collisions
 - IcePrimary
 - FireandIce
 - ElectricandIce
 - WindandIce
 - ElectricBehavior – defines electric behavior/collisions
 - ElectricPrimary
 - FirewithElectric
 - IcewithElectric
 - WindwithElectric
 - WindBehavior – defines wind behavior/collisions
 - WindPrimary
 - WindwithFire
 - WindwithIce
 - WindwithElectric

Dependencies

- Access to:
 - Physics
 - Collision
 - AI System
 - FXManager
- Accessed by:
 - ObjectManager
 - ObjectFactory

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	UPDATE	VOID	UPDATES THE OBJECT
VOID	RENDER	VOID	RENDERS THE OBJECT

Features

- All game specific objects

Time to Complete Estimate

- 2 days

Module Author

- Tim Turcich

Sub Module

Physics

The Physics Module is designed to apply world/ environmental physical qualities to most or all the dynamic objects in the environment. Eg) gravity

Dependencies

- Access to:
 - Object Hierarchy
- Accessed by:
 - Object Hierarchy
 - Object Manager
 - Collision Detection

RETURN

VOID

NAME

APPLY

PARAMETERS

BASEOBJECT* OB –
OBJECT TO APPLY
PHYSICS TOO

DESCRIPTION

APPLIES SOME
PHYSICAL MATHT O
THE OBJECTS
MATRIX.

Features

- Global physics/movement applied to objects

Time to Complete Estimate

- 1 day

Module Author

- Tim Turcich

Sub Module

Collision

The Collision Module defines the basic primitive to primitive tests using the shapes defined in the shapes module. It then uses these tests to do higher level tests that define gameplay type collisions such as the different types of Attacks vs Enemies.

Dependencies

- Access to:
 - Object Hierarchy
- Accessed by:
 - Object Manager

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	DETECT	VOID	DETECTS AND STORES RELEVANT COLLISION INFORMATION.
VOID	REACT	VOID	PROCESSES THE COLLISION SO A PROPER GAMEPLAY COLLISION REACTION WILL OCCUR.
COLLISIONSHAPE*	CREATECOLLISIONVOLUMES	CHAR* NAME, MAT4 TRANSFORM	LOADS A MESH AND TRANSFORMS IT AND BUILDS THE CORRECT COLLISION VOLUME THERE ARE MANY HYBRIDS OF THESE FUNCTIONS.
VOID	HANDLEEVENT	CEVENT *PEVENT – HOLDS A CLASS CONTAINING MESSAGE TYPE AND MESSAGE PARAMETER	RESPONDS TO ADDING OR REMOVING OBJECTS FROM THE SYSTEM (ENEMY_HIT, PLAYER_HIT,

Features

- Detects interactions between the different types of primitives

Associated Risks

Risk:	Affected Resource:	P	C	RF
Collisions detections aren't working and cause the gameplay to break. Such as falling through or going through geometry.	Tim Turcich	0.05	0.8	.64
	Response or avoidance:			
1. Seek help from colleagues 2. Research possibly better collision algorithms 3. The gameplay will have to be reworked to fit the scale that the object manager is capable of. 4. This will be done a week before alpha				

Time to Complete Estimate

- 2 weeks

Module Author

- Tim Turcich

Sub Module**Shapes**

The Shapes module defines all the different collision primitives that represent the objects in Super Power Surge Spike.

Dependencies

- Accessed by:
 - FXManager
 - Collision

Types and Structures defined in shapes.h are Sphere, Plane, AABB, Box, Cylinder, Line Swept Sphere.

Function Table: None

Features

- Definitions for primitives used with collision detection and Effects

Time to Complete Estimate

- 1 day

Module Author

- Tim Turcich

Sub Module

ObjectFactory

To prevent fragmenting of memory from calling new and delete on so many objects all everything when not on use will be stored in a dynamic array within the Object Factory so they can be reused without another new call. The dynamic arrays will start at a preset #define size (stored within object factory .cpp) and will be able to resize automatically if needed.

Dependencies

- Access to:
 - ObjectHierarchy
- Accessed by:
 - ObjectManager

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INIT	VOID	STARTS THE SYSTEM.
VOID	SHUTDOWN	VOID	SHUTS DOWN AND FREE'S MEMORY.
VOID * -THE REQUESTED OBJECT IN A BLANK FORM.	GETSTATICOBJECT	INT NTYPE – AN ENUM OF THE OBJECT TYPE THAT IS BEING REQUESTED.	FROM WITHIN STORED MEMORY RETURNS AN OBJECT WITHOUT CALLING NEW OR DELETE
VOID * -THE REQUESTED OBJECT IN A BLANK FORM.	GETDYNAMICOBJECT	INT NTYPE – AN ENUM OF THE OBJECT TYPE THAT IS BEING REQUESTED.	FROM WITHIN STORED MEMORY RETURNS AN OBJECT WITHOUT CALLING NEW OR DELETE
VOID	RETURNSTATICOBJECT	INT NTYPE – AN ENUM OF THE OBJECT TYPE THAT IS BEING RETURNED. VOID * - A POINTER TO THE RETURNED OBJECT.	RETURNS A POINTER TO THE OBJECT FOR REUSE LATER.
VOID	RETURNDYNAMICOBJECT	INT NTYPE – AN ENUM OF THE OBJECT TYPE THAT IS BEING RETURNED. VOID * - A POINTER TO THE RETURNED OBJECT.	RETURNS A POINTER TO THE OBJECT FOR REUSE LATER.

Enum List

BASE_OBJECT
STATIC_OBJECT,
PLATFORM
RAMP
POWER_NODE
ROUND_WALL
FAN
CAPACITOR
RESISTOR
FLAT_THING
DYNAMIC_OBJECT
ENEMY
ATTACK
FIRE_ENEMY
ICE_ENEMY
ELECTRIC_ENEMY
WIND_ENEMY
AVATAR
BOSS
FIRE_BEHAVIOR
FIRE_PRIMARY
ICE_W_FIRE
ELECTRIC_W_FIRE
WIND_W_FIRE
ICE_BEHAVIOR
ICE_PRIMARY
FIRE_W_ICE
ELECTRIC_W_ICE
WIND_W_ICE
WIND_BEHAVIOR
WIND_PRIMARY
FIRE_W_WIND
ICE_W_WIND
ELECTRIC_W_WIND
ELECTRIC_BEHAVIOR
ELECTRIC_PRIMARY
FIRE_W_ELECTRIC
ICE_W_ELECTRIC
WIND_W_ELECTRIC

Features

- Vast reduction in new/deletes
- Reduces memory fragmentation.
- Resets Object data.

Time to Complete Estimate

- 3 days

Module Author

Joshua Bennett

Renderer

This module is an interface to the entire render system. It will manage culling, materials, HUD, shaders, and render states. This module will manage rendering everything in our game; rendering order of our game objects; initializing and shutting down Direct3D; making the game fullscreen; managing fonts and writing with them; and managing the Direct3D effect framework. To use this module you must call Init and to shut it down you must call Shutdown.

Dependencies

- Access to:
 - AnimationSystem
 - HUD
 - Culling
 - MaterialSystem
- Accessed by:
 - StateMachine

<u>RETURN</u>		<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
BOOL	INIT		//HANDLE TO THE WINDOW HWND HWND	THIS FUNCTION INITIALIZES THE RENDERER AND SETS UP DIRECTX TO USE THE HIGHEST QUALITY SETTINGS THE GRAPHIC DEVICE CAN HANDLE.
VOID	MAKEFULLSCREEN		BOOL BFULLSCREEN	THIS FUNCTION TAKES IN A BOOL AND EITHER WINDOWS OR FULLSCREENS THE GAME WINDOW.
VOID	SETCOLORMASK		//ALLOW RED? BOOL BALLOWRED //ALLOW GREEN? BOOL BALLOWGREEN	SETS IF A COLOR IS ALLOWED IN THE COLOR BUFFER.

		//ALLOW BLUE? BOOL BALLOWBLUE	
		//ALLOW ALPHA? BOOL BALLOWALPHA	
VOID	CLEARBACKBUFFER	VOID	CLEAR THE BACKBUFFER
VOID	CLEARZBUFFER	VOID	CLEAR THE Z- BUFFER
VOID	CLEARBUFFERS	VOID	CLEAR BOTH THE BACK BUFFER AND Z- BUFFER
VOID	CLEARBACKBUFFER	INT NWIDTH INT NHEIGHT //POSITION OF THE TOP //LEFT CORNER INT NX INT NY	
INT	LOADFONT	//FILE NAME OF THE FONT CONST CHAR* SZFILENAME //THE SIZE OF THE FONT INT NSIZE //ITALIZED? BOOL BITALIC //BOLD? BOOL BBOLD //FONT ID INT NID	THIS FUNCTION LOADS IN A FONT TO THE FONT VECTOR AND RETURNS YOU ITS INDEX.
VOID	UNLOADFONT		UNLOADS THE SELECTED FONT FROM THE FONT VECTOR
VOID	WRITEFONT	//FONT TO USE INT NID //POSITION INT NX INT NY	THIS FUNCTION USES A SPECIFIED FONT TO WRITE TEXT TO THE SCREEN IN A SPECIFIED

		//COLOR RGBA FLOAT* COLOR	LOCATION AND COLOR.
		//TEXT TO WRITE CONST CHAR* SZTEXT	
VOID	SETPOINTSIZE	//POINT SIZE! FLOAT FSIZE	SET THE SIZE OF THE POINTS
VOID	PUSHMATERIAL	ABSTRACTMATERIAL* PMATERIAL	PUSHES A MATERIAL ONTO THE MATERIAL STACK.
VOID	POPMATERIAL	INT NMERGEMODE VOID	POPS A MATERIAL OFF THE MATERIAL STACK
INT	GETMATERIALSIZE	VOID	THE CURRENT SIZE OF THE MATERIAL STACK
VOID	DRAW	CONST OBJECT3D* POBJ	DRAWS THE OBJECT BASED ON ITS INFORMATION
VOID	DRAW	CONST OBJECT3D* POBJ ABSTRACTMATERIAL* PMATERIAL INT NMERGEMODE	DRAWS THE OBJECT WITH THE SPECIFIED MATERIAL AND OBJECT INFORMATION.
VOID	RENDER	VOID	CLEARs SCREEN AND DRAWS ALL OBJECTS IN DYNAMIC AND STATIC LIST.
VOID	PRECOMPILE	VOID	PRECOMPILES THE SHADERS LOADED IN THE MATERIAL STACK
BOOL	SHUTDOWN	VOID	SHUTS DOWN DIRECTX AND CLEANS UP ANY MEMORY
VOID	BEGINSCENE	VOID	BEGINS

VOID	ENDSCENE	VOID	DRAWING THE SCENE ENDS THE SCENE DRAWING.
INT	PUSHSTATIC3DOBJECT	OBJECT3D* POBJ	ADDS A STATIC OBJECT TO BE RENDERED
INT	PUSHDYNAMIC3DOBJECT	OBJECT3D* POBJ	ADDS A DYNAMIC OBJECT TO BE RENDERED
VOID	REMOVESTATIC3DOBJECT	INT OBJECTID	REMOVES AN OBJECT FROM THE STATIC OBJECT LIST
VOID	REMOVEDYNAMIC3DOBJECT	INT OBJECTID	REMOVES AN OBJECT FROM THE DYNAMIC OBJECT LIST
CAMERA*	GETCAMERA	VOID	GETS THE CAMERA CURRENTLY IN USE
CULLING*	GETFRUSTUM	VOID	RETURNS THE FRUSTUM
VOID	SETANIMATIONSYSTEM	ANIMATIONSYSTEM*	SETS THE ANIMATION SYSTEM THAT THE RENDER SYSTEM WILL USE TO RENDER FROM.
INT	LOADSHADER	CONST CHAR* SZFILENAME	THIS FUNCTION LOADS IN A EFFECT FILE AND RETURNS THE INDEX INTO THE VECTOR OF SHADERS.
VOID	UNLOADSHADER	INT NID	UNLOADS THE SPECIFIED SHADER ID
VOID	UNLOADALLSHADERS	VOID	UNLOADS ALL SHADERS.

Features

- Acts as an interface to render all objects in SPSS in one centralized location.
- Organizes Materials to cut down on render-state changes.

Structs

```
struct Object3D
```

```
{
```

```
    Mat4 orientation Matrix;
```

```
    int nTechniqueID;
```

```
    int nTexID;
```

```
    int nMeshID;
```

```
};
```

Associated Risks

Risk:	Affected Resource:	P	C	RF
The renderer might be incapable of drawing enough enemies to the screen. Not being able to draw the required enemies will severely hurt the pacing of our game as it is currently set up.	Justin Morgan, Tech Lead	.2	.9	0.92
	Response or avoidance:			
5. Consult internet forums, articles, and books. 6. Seek help from other classmates working on rendering. 7. Seek help from staff on rendering and culling techniques. 8. If the renderer can't handle all the enemies necessary we will replace the large amount of enemies with less, but more powerful ones. This will be enacted at alpha.				

Time to complete estimate

- Basic Renderer
 - 1 day
- Render Manager
 - 5 days
- Model Rendering
 - 1 day

Total: 7 days

Module Author

Justin Morgan

Sub Module

HUD

This module controls the communication between all objects and modules in our game. Objects or modules will register with this dispatcher and when an event happens for a specific object or module then the dispatcher will send that event out. To use the HUD you must call Init and to shut it down you must call Shutdown.

Dependencies

- Accessed to:
 - Player System
 - Object Manager
- Accessed by:
 - Renderer

<u>Return</u>	<u>Name</u>	<u>Parameters</u>	<u>Description</u>
void	Init	None	Initialized elements of the HUD
void	Update	float fDelta	Updates the elements of the HUD by the passed in time slice. The information for the HUD will be grabbed from the player system as well as the Object Manager.
void	Render	None	Renders the HUD to the screen
void	Shutdown	None	Shuts down any elements dealing with the HUD

Features

- Allows one location for all HUD elements to render and update.

Time to Complete Estimate

- Implementation
 - 2 Days
- Testing and Integration
 - 1 day

Total: 3 days

Module Author

Philip Fox

Sub Module

MaterialSystem

This module is a sub set of the entire rendering system. The purpose of this system is to make a visual change to world objects trivial, and to allow us to make quick changes without recompiling the code. As a secondary purpose, this system is meant to minimize render context switches to help mitigate some of the rendering bottlenecks. All renderable objects should have one of these material sets.

Dependencies

- Accessed By:
 - Renderer

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	CLEAR	VOID	THIS FUNCTIONS CLEARS OUT THE MATERIALS LOADED IN FROM THE SCRIPT
VOID	LOADSCRIPT	CONST CHAR* SZFILENAME	THIS FUNCTIONS LOADS IN A SCRIPT TO THE MATERIAL
VOID	UPDATE	FLOAT FDELTATIME	THIS FUNCTION UPDATES THE DYNAMIC MATERIALS. IT IS UNNECESSARY TO CALL THIS FUNCTION ON MATERIALS THAT AREN'T DYNAMIC, BECAUSE TO BE DYNAMIC MEANS THAT IT MUST BE UPDATED EVERY FRAME.
INT	BLEND	CONST CHAR* SZFUNCTIONNAME CONST CHAR* SZFILENAME	THIS FUNCTION SETS HOW THE MATERIALS WILL BE BLENDED TOGETHER.
VOID	UPDATEDYNAMICMATERIALS	VOID	UPDATE ALL DYNAMIC MATERIALS ON THE STACK
BOOL	ISDYNAMIC	VOID	IS THE MATERIAL DYNAMIC?
BOOL	ISTRANSSPARENT	VOID	IS THE MATERIAL

ITN	GETMERGEMODE	VOID	TRANSPARENT RETURNS THE CURRENT MERGE MODE
VOID	POSTRENDER	VOID	CHANGES THE MATERIALS BACK TO THE DEFAULT ONES.

Features:

- Scriptable materials to make appearance changes trivial.
- Holding default render states and only changing the ones relevant to the object that needs to be drawn will optimize our game by cutting down on render context changes.

Associated Risks

Risk:	Affected Resource:	P	C	RF
The material system might cause the load times to be too long to be acceptable. This might be an issue due to the slowness of reading in text and parsing it in combination with the large number of the scripts we will need to load.	Justin Morgan, Tech Lead	.1	.8	0.82
	Response or avoidance:			
	9. Consult internet forums, articles, and books. 10. Seek help from other classmates working on scripting. 11. Seek help from staff on scripting techniques. 12. If the load times take too long we will cut down on the amount that needed to be loaded at run time by hard coding some of the attributes into the code base This will be enacted a week before beta.			

Time to complete estimate

- Data Driven Material System
 - 3 days

Module Author

Justin Morgan

Sub Module

Camera

The Camera Module defines the functionality for controlling the viewpoint of the game.

Dependencies

- Accessed by:
 - StateMachine
 - Render
 - Collision

Function Table:

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	UPDATEBASEDONAVATAR	D3DXMATRIX AVATARWORLDMAT – THE MATRIX OF THE TARGET D3DXVECTOR3 OFFSET – OFFSET FROM THE TARGET	DOES A HARD ATTACH AND THEN APPLYS A ROTATION BASED ON THE MOUSE MOVEMENT DURING GAMEPLAY
VOID	SWOOPINONAVATAR	D3DXMATRIX AVATARWORLDMAT – THE MATRIX OF THE TARGET	THE LEVELS START WITH THE CAMERA SWOOPING IN ON THE AVATAR - ALICE

Associated Risks

Risk:	Affected Resource:	P	C	RF
Due to the fact that we are doing collision with the camera and the level. There is a possibility that the camera will get stuck in geometry.	Tim Turcich	0.05	0.8	.64
	Response or avoidance: 1. Seek help from colleagues by FF1. 2. Find how professionals do cameras by FF2. 3. Back up plan is to let the camera go through geometry by Beta.			

Features

- Hard attached to player allowing rotation around player for a 360 degree view.

Time to Complete Estimate

- 3 days

Module Author

- Tim Turcich

AI System

The AI system is broken down into five primary modules: The Waypoint Manager, Pathfinding, Flocking and the Hive Mind module. The AI Manager will handle all incoming and out going communications to other systems and will control the other modules. Waypoint Manager will handle connecting points to the player and to each other, along with storing them. Pathfinding will use the waypoints gotten from the waypoint manager and will create paths to the players avatar. Flocking will keep the enemies within groups while also handling the jumping of wind enemies. Finally the Hive Mind will be the bridge between Pathfinding and Flocking by controlling groups and causing them to follow after the player, along with electric enemies warping ahead.

Dependencies

- Access to:
 - Asset Manger
 - ObjectHierarchy
 - Flocking
 - Pathfinding
 - HiveMind
- Accessed by:
 - ObjectHierarchy
 - StateMachine

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	AILEVELLOAD	INT NLEVEL – THE LEVEL BEING LOADED, -1 FOR ENDLESS	PREPARES THE SYSTEM FOR THE NEXT LEVEL, MUST HAVE ALL WAYPOINTS BEFORE BEING CALLED.
VOID	CLEAR	VOID	CLEANS UP ALL SYSTEMS FOR THE NEXT RUN.
VOID	ADDSPAWNPOINT	VEC3 VEC3POS OR FLOAT FX, FLOAT FY, FLOAT FZ – USED FOR CREATING A NEW POINT IN THE WORLD TO SPAWN AT	ADDS A NEW SPOT TO SPAWN ENEMIES IN THE WORLD.
VOID	UPDATE	VOID	DEPENDING ON EVENTS MAY OR MAY NOT UPDATE PATHFINDING,

VOID	HANDLEEVENT	CEVENT *PEVENT – HOLDS A CLASS CONTAINING MESSAGE TYPE AND MESSAGE PARAMETER	ALWAYS CHECKS GROUP FLOCKING, HANDLES SPAWNING NEW WAVES. PROCESSES AN EVENT MESSAGE. (ENEMY_DEATH)
VOID	ADDWAYPOINT	VEC3 VEC3POS OR FLOAT FX, FLOAT FY, FLOAT FZ – USED FOR CREATING A NEW WAYPOINT WITHIN THE WORLD	ADDS A NEW WAYPOINT TO THE WAYPOINT MANAGER THROUGH THE PATHFINDING SYSTEM.
SQUAD*	SPAWNSQUAD	INT NENEMYCOUNT – THE NUMBER TO SPAWN INT NTYPE – THE TYPE OF ENEMY TO SPAWN VEC3 VECSPAWNPOINT – WHERE TO SPAWN THE SQUAD AT	LETS OUTSIDE SYSTEMS SPAWN NEW SQUADS AT ANY LOCATION THEY DESIRE, MADE FOR BOSS USE. RETURNS A POINTER TO THE SQUAD.
SQUAD*	SPLITSQUAD	ENEMY* PENEMY – THE ENEMY TO PUT INTO A NEW SQUAD	REMOVES THE ENEMY FROM ITS OLD SQUAD AND PUTS IT INTO A NEW ONE BY ITSELF.

Features

- Covers all AI systems.
- Access point for all other systems

Time to Complete Estimate

- Basic Framework
 - 1 day
- Boss States
 - 3 days

Total 4 days

Module Author

Joshua Bennett

Sub Module

Pathfinding

Used to find the fastest path from the nearest waypoint to the player. Pathfinding takes air enemies jumping into account. To avoid unneeded costs, this system will be updated only when there is enough of a change in the environment. Pathfinding also stores all in level way points loaded from the Asset Manager. Pathfinding tells the Asset Manager when to load and remove the waypoints from memory.

Dependencies

- Access to:
 - Waypoint Manager
- Accessed by:
 - HiveMind
 - AI System

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	CLEAR	VOID	CLEANS SYSTEM OUT FOR NEXT USE
VOID	UPDATE	VEC3 VEC3POS – GOAL XYZ	TAKES THE GOAL POSITION AND DECIDES IF PATHS NEED TO BE UPDATED DEPENDING ON THE WAYPOINT MANAGER
VOID	INITLEVEL	VOID	TELLS THE WAYPOINT MANAGER TO CONNECT ALL WAYPOINTS.
VOID	ADDWAYPOINT	VEC3 VEC3POS – USED FOR CREATING A NEW WAYPOINT WITHIN THE WORLD	ADDS A NEW WAYPOINT TO THE WAYPOINT MANAGER.
VOID	FINDPATH	SQUAD* SQUADTOMOVE – THE SQUAD TO BE GIVEN A NEW PATH	FINDS A PATH FOR THE SQUAD PASSED IN, AND FILLS OUT THE SQUADS PATH VECTOR.

Features

- Optional and to be updated only when needed.

Time to Complete Estimate

- Grouping
 - 2 days
- Waypoints
 - 3 days

Total: 5 days

Module Author

Joshua Bennett

Sub Module

Waypoint Manager

Holds all waypoints within the world and handles connecting them to each other and to the player as well.

Dependencies

- Accessed by:
 - Pathfinding

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INITLEVEL	VOID	CONNECTS ALL PRELOADED WAYPOINTS TO EACH OTHER.
VOID	CLEAR	VOID	CLEANS ALL SYSTEMS OUT TO BE REUSED.
BOOL – TRUE IF ANY CONNECTIONS CHANGED	CONNECTTOPLAYER	VOID	CHECKS WHICH WAYPOINTS LEAD TO THE PLAYER, AND RETURNS TRUE IF ENEMIES SHOULD TRY TO FIND A NEW PATH.
BOOL – TRUE IF A CONNECTION CAN BE MADE	CANCONNECT	VEC3 VEC3POS1 – FIRST OBJECTS LOCATION, VEC3 VEC3POS2 – SECOND OBJECTS LOCATION	CHECKS IF TWO POINTS IN THE WORLD CAN REACH EACH OTHER, RETURNS TRUE IF POSSIBLE.
VOID	ADDWAYPOINT	VEC3 VEC3POS – USED FOR CREATING A NEW WAYPOINT WITHIN THE WORLD	ADDS A NEW WAYPOINT TO THE SYSTEM.
WAYPOINT* - THE CLOSEST WAYPOINT FOUND	FINDCLOSESTWAYPOINT	VEC3 VECXYZ – THE PLACE OF THE OBJECT LOOKING FOR ITS NEXT POINT, BOOL BJUMP – IF THE OBJECT CAN USE JUMPING WAYPOINT	FINDS THE CLOSEST WAYPOINT THAT CAN BE REACHED AND RETURNS IT. IF NONE WORK, RETURNS THE PLAYERS POINT

AS A FAILSAFE.

WayPoint – Struct within Waypoint Manager

Holds the place of the given point, data of its connection with the player, and what other points it can connect to.

Features

- Only updates when player has moved far enough.

Time to Complete Estimate

???

Module Author

Joshua Bennett

Sub Module

Flocking

This system is to manage movement within the groups to avoid enemy colliding with each other or spreading out too much.

Dependencies

- Accessed by:
 - HiveMind

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	SQUADMOVE	WAYPOINT* PNEXTPOS – THE NEXT LOCATION TO MOVE TO, SQUAD* CURRENTSQUAD – THE SQUAD TO MOVE	HANDLES TELLING EVERY ENEMY WITHIN A SQUAD WHERE TO MOVE ASIDE FROM TELEPORTING

Features

- Checks and updates every enemy movement.
- Handles jumping behavior.
- Splits squads if enemies are to far apart.

Time to Complete Estimate

- Basic
 - 2 days.
- Geometry avoidance
 - 3 days
- Electric Behavior
 - 3 days

Total: 8 days

Module Author

Joshua Bennett

Sub Module

HiveMind

Primarily a class for holding data, the hive mind stores group's information, all enemies within a given group, and the next location for them to move to. When GetNextPoint function in the player manager is called, it references the data stored in the Hive Mind. Due to its role as a storage holder within the AI Manager, it has no functions other than accessors and mutators. All groups are stored within the hive mind.

Dependencies

- Access to:
 - Pathfinding
- Accessed by:
 - AI System

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	LOAD	VOID	LOADS ALL NEEDED FOR EFFECTS SYSTEM.
VOID	SHUTDOWN	VOID	SHUTS DOWN AND FREE'S SQUAD MEMORY, NOT ENEMIES WHICH IS HANDLED IN OBJECT MANAGER.
VOID	CLEARALL	VOID	CLEANS SYSTEM UP AND READY'S IT FOR NEXT USE.
SQUAD* - POINTER TO SQUAD MADE	NEWSQUAD	INT NENEMYCOUNT – THE NUMBER TO SPAWN INT NTYPE – THE TYPE OF ENEMY TO SPAWN VEC3 VECSPAWNPOINT – WHERE TO SPAWN THE SQUAD AT	MAKES A NEW SQUAD AT THE LOCATION GIVEN.
SQUAD* - POINTER TO SQUAD MADE	NEWSQUAD	ENEMY* PENEMY – THE ENEMY TO PUT IN A SOLO SQUAD	PUTS THE PASSED IN ENEMY TO A NEW SQUAD OF ITS OWN.
VOID	ADDENEMY	ENEMY* PENEMY – ENEMY TO ADD TO A SQUAD	PUTS THE PASSED IN ENEMY INTO A SQUAD MATCHING ITS ID.

VOID	REMOVEENEMY	ENEMY* PENEMY – ENEMY TO REMOVE FROM A SQUAD	REMOVES THE ENEMY FROM ITS SQUAD, ALSO CHECKS IF TO REMOVE THE SQUAD OR NOT.
VOID	SORTENEMIES	SQUAD* PSQUADTOSORT – THE SQUAD TO SORT, VEC3& POSPLAYER – A REFERENCE OF THE LOCATION OF THE PLAYER	SORTS ENEMIES ON FACTORS OF HEIGHT AND DISTANCE FROM THE PLAYER AND DECIDES THE LEADER ON THE RESULTS.
VOID	UPDATE	VOID	DETERMINES ALL SQUAD ACTIONS BY CALLING OTHER SYSTEMS AND HANDLING THE RESULTS.

Squad – Struct in HiveMind

Contains information for all enemies within a single group, along with other information needed for electric enemies teleporting, wind enemies jumping, and the current path.

Features

- Holds all movement information.
- Holds all Group information.

Time to Complete Estimate

- 2 days for both

Module Author

Joshua Bennett

FXManager

This module is responsible for controlling all of the audio/visual weapon and special effects in the game. It will create compositions of the various effect pieces and will coordinate between all the components such as: shader effects, particle effects, sound effects, and geometry. Any renderable objects will be created and fed off to the object manager. Access to the various subsystems will be given through the public interface for custom purposes, but the FX system also has support for automated initialization of predetermined effects via a single function call.

Dependencies

Access to:

- Sound Manager
- Particle Manager
- ObjectManager

Accessed by:

- StateMachine
- AnimationSystem

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INIT	NONE	INITIALIZES THE FX SYSTEM AND THE VARIOUS SUBSYSTEMS FOR USE. LOADS IN EFFECTS DESCRIPTIONS
VOID	SHUTDOWN	NONE	SHUTS DOWN THE FX SYSTEM AND THE VARIOUS SUBSYSTEMS.
VOID	STARTEFFECT	VEC3 VECPPOSITION //START LOC OF EFFECT	CREATES THE DESIRED EFFECT, AUTOMATED ENTIRELY.
		INT NNAME //ENUM INDEX OF EFFECT	
PARTICLEMANAGER *	GETPARTICLEMGR	NONE	RETURNS POINTER TO PARTICLE SYSTEM FOR

AUDIOSYS *

GETAUDIOSYS

NONE

ACCESSING PUBLIC
FUNCTIONS.
RETURNS POINTER
TO THE AUDIO
SYSTEM FOR
ACCESSING PUBLIC
FUNCTIONS.

Enum of Effects

FIRE_PRIMARY

ICE_PRIMARY

WIND_PRIMARY

ELECTRIC_PRIMARY

FIRE_ICE_SECONDARY

FIRE_WIND_SECONDARY

FIRE_ELECTRIC_SECONDARY

ICE_FIRE_SECONDARY

ICE_WIND_SECONDARY

ICE_ELECTRIC_SECONDARY

WIND_FIRE_SECONDARY

WIND_ICE_SECONDARY

WIND_ELECTRIC_SECONDARY

ELECTRIC_FIRE_SECONDARY

ELECTRIC_ICE_SECONDARY

ELECTRIC_WIND_SECONDARY

Features

- Allows the combination of multiple systems into one controllable Effect Object that provides a convenient façade for the individual effects subsystems as well as providing a simplified function that will generate a complete effect in the world.
- Dynamically assigns shaders to an effect based on whether it is required or not.
- Generates renderable objects that will plug directly into the object manager, preserving the responsibilities of the overall system architecture.
- Allows for activation of effects based on plain English enumerations, making function calls inherently self-documenting.

Associated Risks

Risk:	Affected Resource:	P	C	RF
Implementing 16 unique weapon effects. Without the effects, the game loses a lot of its feedback and intensity. Also, the underlying elemental system is potentially compromised by the absence of all 16 weapon attacks.	Michael Keenan, Design Lead	.3	.8	.86
	Response or avoidance: 1. Offload some of the effects work to Tim if I am behind schedule by Alpha. 2. Reuse some of the weapon effects, only changing small things like color. 3. Make each effect a simple colored particle emission if unable to implement by Beta.			

Time to Complete Estimate

Compilation of the sub-systems + 4 days tweaking each of the systems.

Module Author

Michael Keenan

Sub Module

Sound Manager

This module is responsible for generating and managing all sound effects within the game. It will load, store, play, stop, and deallocate all the available sound assets in the game. The sound system will be making use of the FMOD 3 API to trivialize the mundane functions of sound playback, as well as giving me easy tools to manipulate the sound to add more intensity to the game. If time permits, layered background sound will be used.

Dependencies

- Accessed by:
 - State Machine
 - FX Manager

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INIT	NONE	INITIALIZES THE SOUND SYSTEM.
VOID	SHUTDOWN	NONE	SHUTS DOWN THE SOUND SYSTEM.
VOID	PLAYSOUND	INT NID //ID OF THE SOUND TO PLAY BOOL BLOOPING //SIGNIFIES A LOOPING SOUND OR NOT.	SIMPLY PLAYS THE SOUND BASED ON THE SOUND ID PASSED IN AND WHETHER IT IS LOOPING OR NOT.
VOID	STOPSOUND	INT NID //ID OF THE SOUND TO PLAY.	IF SOUND IS CURRENTLY PLAYING, PAUSES SOUND.
VOID	RESETSOUND	INT NID //ID OF THE SOUND TO PLAY	RESETS SOUND TO BEGINNING OF TRACK.
VOID	DISTORT	INT NID //ID OF THE SOUND TO PLAY	ADDS A STYLISTIC DJ DISTORTION TO THE BACKGROUND MUSIC (USED DURING CERTAIN EFFECTS AND DEATH).
VOID	PLAY3DSOUND	INT NID //ID OF THE SOUND TO PLAY.	PLAYS A 3D SOUND.

		VEC3	VPOSITION	
			//POSITION OF THE	
			SOUND IN 3D SPACE.	
		VEC3	VLISTENER	
			//POSITION OF THE	
			LISTENER IN 3D SPACE	
VOID	LOADSOUND	CHAR *	SZFILENAME	LOADS A SOUND
			//FILE OF THE SOUND	INTO MEMORY FOR
			TO LOAD.	PLAYBACK.
VOID	UNLOADSOUND	INT	NID	REMOVES A SOUND
			//ID OF THE SOUND TO	FROM MEMORY.
			REMOVE FROM	
			MEMORY.	

Features

- Serves as a storage center for sound IDs, centralized playback station for all audio
- Supports both 3D and normal sound.

Associated Risks

Risk:	Affected Resource:	P	C	RF
Getting layered sound to work so that it responds accurately to the number of enemies onscreen and that it doesn't switch layers so fast that it sounds jarring.	Michael Keenan, Design Lead	.5	.2	.6
	Response or avoidance: <ol style="list-style-type: none"> 1. Ask Sigsby for help as needed since he has some experience working with FMOD by 10/25. 2. Reduce number of songs to a single one and see if I can get that to work by Beta. 3. Scrap layered sound and just play a lone background track. 			

Time to Complete Estimate

- 2 days for tech associated
- 1 day for integration and testing for each effect (shared with other effect integration)
- 4 days for tweaking and bug fixes (shared with FX Manager)

Total: 22 days

Module Author

Michael Keenan

Sub Module

Particle Manager

This module is responsible for generating all of the particle effects in the game. It will store all the available particle descriptions, and it will handle the initialization and passing of the renderable emitter to the object manager. Point sprites will be used to generate the particles.

Dependencies

- Access to:
 - ObjectManager
- Accessed by:
 - FX Manager

<u>RETURN</u>	<u>NAME</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
VOID	INIT	NONE	INITIALIZES THE PARTICLE SYSTEM AND FOR USE. LOADS IN PARTICLE EMITTER DESCRIPTIONS
VOID	SHUTDOWN	NONE	SHUTS DOWN THE PARTICLE SYSTEM.
VOID	CREATEEMITTER	VEC3 VECPPOSITION //START LOC OF EFFECT INT NNAME //ENUM INDEX OF EFFECT	CREATES THE EMITTER USING THE DESIRED DESCRIPTOR, AND FEEDS THE EMITTER TO THE OBJECT MANAGER.
NOTHING	PARTICLEMANAGER	CHAR * SZFILENAME //FILE WHICH CONTAINS EMITTER DESCRIPTIONS	DEFAULT CONSTRUCTOR
NOTHING	~PARTICLEMANAGER	NONE	DESTRUCTOR

Enum of Effects

FIRE_PRIMARY
ICE_PRIMARY
WIND_PRIMARY
ELECTRIC_PRIMARY
FIRE_ICE_SECONDARY

FIRE_WIND_SECONDARY
 FIRE_ELECTRIC_SECONDARY
 ICE_FIRE_SECONDARY
 ICE_WIND_SECONDARY
 ICE_ELECTRIC_SECONDARY
 WIND_FIRE_SECONDARY
 WIND_ICE_SECONDARY
 WIND_ELECTRIC_SECONDARY
 ELECTRIC_FIRE_SECONDARY
 ELECTRIC_ICE_SECONDARY
 ELECTRIC_WIND_SECONDARY

Features

- Serves as a storage center for emitter blueprints, as well as a factory that can generate a specific emitter on request.
- Generates renderable objects that will plug directly into the object manager, preserving the responsibilities of the overall system architecture.
- Allows for activation of emitters based on plain English enumerations, making function calls inherently self-documenting.

Associated Risks

Risk:	Affected Resource:	P	C	RF
The particle assets may not be delivered to us in a timely fashion, preventing me from experimenting with them and getting it to look good in the game, up to the standard of the rest of the particle effects.	Michael Keenan, Design Lead	.5	.8	.9
	Response or avoidance: 1. Offload some of the effects work to Tim if I am behind schedule by Alpha. 2. Reuse some of the weapon effects, only changing small things like color. 3. Make each effect a simple colored particle emission if unable to implement by Beta.			

Time to Complete Estimate

- Research/Tech 5 days
- 2 days for each weapon effect (shared partly with other sub-systems)
- 4 days tweaking and bug fixes (shared with FX Manager).

Total: 41 days

Module Author

Michael Keenan

Milestone Deliverables

PoC

Joshua Bennett:

Asset Managment

Object Creation and Load

Phillip Fox:

The player can move around the world with WASD.

Basic menu system.

Game input: WASD, spacebar, mouse movement, and mouse left button input.

Michael Keenan:

Basic attack particles.

Sound effect playback on attack.

Justin Morgan:

Basic primitive and model rendering.

Basic animation playbacks.

Tim Turcich:

The player can jump using the spacebar.

The player aims in the direction dictated by the reticule, which is controlled by the mouse.

Attack collision with target.

"PoC" fire attack.

FeatureFrag1

Joshua Bennett:

A level will be loaded into the game.

Enemies will be showing basic flocking and fleeing intelligence.

Enemies will be spawning into the world.

Phillip Fox:

There will be a HUD in place giving the pertinent player information.

Michael Keenan:

Fire primary attack will be in the game.

Fire with ice secondary attack will be present.

Fire with electric secondary attack will be present.

Ice primary attack will be in the game

Wind primary attack will be in the game

Electric primary attack will be in the game

Justin Morgan:

There will be more than one animation in the game.

Animations will be using shaders.

Tim Turcich:

All primary attacks will collide with enemies.

FeatureFrag2

Joshua Bennett:

Enemies will perform geometry avoidance with the flocking/fleeing
The electric enemies will perform their unique movement patterns.
Enemies will demonstrate grouping.
Enemies will be pathfinding using waypoints.

Phillip Fox:

Player model integration and testing.
Player jump integration and testing.
Player run forward integration and testing.
Gambit model integration and testing.
Gambit run integration and testing.
Level Memory integration and testing.

Michael Keenan:

Fire with wind secondary attack will be present.
Ice with fire secondary attack will be present.
Ice with electric secondary attack will be present.
Ice with wind secondary attack will be present.
Wind with fire secondary attack will be present.
Wind with ice secondary attack will be present.
Wind with electric secondary attack will be present.

Justin Morgan:

The geometry and enemies in the world will be culled.
The render manager will be up supporting its subsystems.
The lighting will be partially in place, including a Ward anisotropic directional light and a minnaert point light.

Tim Turcich:

Attack collisions and physics for the fire behavior, ice behavior, wind behavior, and electric behavior.

Alpha

Joshua Bennett:

Testing attack mechanics.
Testing enemy abilities.

Phillip Fox:

Player hit integration and testing.
Player attack00 integration and testing.
Env PowerNode integration and testing.
Player foot step integration and testing.

Michael Keenan:

Electric with fire secondary attack will be present.
Electric with ice secondary attack will be present.
Electric with wind secondary attack will be present.

Justin Morgan:

Fully supported data driven material system.

Tim Turcich:

Attack physics and collision for electric behavior.

Testing attack collision.

Beta

Joshua Bennett:

Achievement system will be in the game.

The scoring/score multiplier will be in place.

Player management will be present.

Boss states will be in place.

Time attack, survival and evasion modes will be in the game.

Phillip Fox:

The HUD will have a radar present, displaying the enemy locations.

Player walk backward integration and testing.

Player death integration and testing.

Player strafe left integration and testing.

Player strafe right integration and testing.

Gambit hit integration and testing.

Gambit death integration and testing.

Gambit jump integration and testing.

Tex Player integration and testing.

Normal player integration and testing.

Fire Gambit Texture integration and testing.

Gambit normal integration and testing.

Menu assets integration and testing.

HUD assets integration and testing.

Splash screens integration and testing.

Game music00 integration and testing.

player grunt sounds integration and testing.

player hit sounds integration and testing.

enemy hit sounds integration and testing.

Michael Keenan:

Level testing power nodes and object placement.

Power node system

Level design and layout.

Justin Morgan:

Animation blending will be in the game.

Applicable objects in the game will be using the Glow Shader.

Abillity swapping

Player shield.

Tim Turcich:

Level geometry will be affected by a normal mapping shader.

Attack system will be in place.

The boss will be in place.
Fire attack interaction tweaking

Gold

Joshua Bennett:

Scoring system tweaks.
Fire Enemy Hit response tweaking.
Ice Enemy Hit response tweaking.
Electri Enemy Hit response tweaking.
Wind Enemy Hit response tweaking.
Enemy AI bugs and tweaking.

Phillip Fox:

Player idle integration and testing.
Player b attack animation integration and testing.
Hub level integration and testing.
Power Grid Level integration and testing.
Boss model integration and testing.
Boss rage and puke animation integration and testing.
Level harddrive integration and testing.
Environmental integration and testing.
Gambit ice texture integration and testing.
Wall textures integration and testing.
Load screen integration and testing.
Pause screen integration and testing.
Gambit wind texture integration and testing.
Gambit Electric texture integration and testing.
Player Death sounds integration and testing.
Menu music integration and testing.
Enemy death sounds integration and testing.
Game music integration and testing.
Dialog taunt sounds integration and testing.
Intro cutscene integration and testing.

Michael Keenan:

Effect textures integration and testing.
Attack sounds effect textures integration and testing.
Effect bug tweaking.

Justin Morgan:

Rendering bug fixes and tweaking
Animation bug fixes and tweaking

Tim Turcich:

Fire Attack interaction tweaking.
Ice Attack interaction tweaking.
Electric Attack interaction tweaking.
Wind Attack interaction tweaking.

Collision Bug fixes.
Gameplay bug fixes.

Archive

Joshua Bennett:

Documentation archival.

Phillip Fox:

Documentation archival.

Michael Keenan:

Documentation archival.

Justin Morgan:

Documentation archival.

Tim Turcich:

Documentation archival.

Appendix A

Memory Map

CPU	GPU
2 mB for Executable	50.3 mB for level textures
28.8 kB for a level	1.26 mB for HUD information
160kB for avatar model	6.2 mB for avatar texture
4.8 mB for max on screen enemies	6.2 mB for enemy textures
1.1 mB for particle textures	1.1 mB for particle textures
30 mB for SFX and background music	31.2 mB for menu

CPU Total 38mB 80kB GPU Total 65mB 60kB

Integration Plan

Integration Lead

Joshua Bennett

Methodology

Integration will take place at the end of every working day. At the end of each code integration there will be a build to check for errors, warnings, and crashes. On top of these nightly integrations there will be an integration day on the Saturday before a milestone. On this day and the following Sunday there will be a code freeze as to avoid unintentional bugs before a turn-in. We will use AlienBrain source control to moderate our code and asset source control. If during integration a module breaks the build then the author(s) of the module will debug and troubleshoot the problem. If he cannot determine the source of the problem within two hours we will rollback the build to the last known working build. The author(s) of the module that was causing the issue will store a local copy before this roll back and immediately drop what he is scheduled for to fix the problem. We will not allow multiple file check outs in hopes that it will mitigate some source integration issues. A group member is allowed to check out however many files that are needed at one time. Also, when checking back in a file the group member will be required to add a comment saying what was changed in that file. This comment should make rollbacks a simpler process with less guessing required.

Testing Plans

Testing is to be primary done by the system's creator, with further testing by others in the team upon integration. Scheduled testing will list who will test the given section while any unforeseen testing will be delegated by the gameplay lead Joshua Bennett. Testing for bugs will be done in the times specified in the Gantt or if a team member happens to be ahead of schedule.

No code is to be submitted with known bugs of a high or critical level, and medium bugs are to be fixed before any milestone. Although finding bugs will be primarily the module author's duty, the repairing of bugs will be given to anyone ahead of schedule if the primary author cannot afford to spend time fixing it, or if the number of bugs has become to great then the whole team will begin to work on fixing them.

List of bug levels and examples:

- Low
 - Minor clipping between objects
 - Anything that will be hard to see by standard players
- Med
 - Flickering images
 - AI hang-ups
 - Anything that doesn't break gameplay
- High
 - Entire AI failure
 - Camera getting caught on geometry
 - Anything that breaks gameplay, but not the game
- Critical
 - Game crashing
 - Anything that makes the game unplayable

The bug report will be kept in the root project folder on the server and will be added to for each known bug found within the master build. Each report will never be removed, but will show if the bug has been repaired or not. The report must hold the following: The date found, the date fixed the threat level, what system it is in relation too, a description of the bug with how to repeat it (if possible), and a unique ID number. The unique ID will be prefixed by the level of the bug then a number unique to that bug priority level. The way we will insure the uniqueness of each priority levels number is by incrementing from the last known bug of that level.

Example bug report:

Date found: 10/10/08

Date fixed: n/a

Threat level: High

Module affected: AI

Description: When behind minor environment objects, the enemies will get hung up and will cluster. Glitch is often and will happen many times in a single level, making groups of enemies too easy to destroy and reduces and removes all game challenge. To repeat just play a level and run behind game objects.

ID: HIGH_001

Appendix B

Game Folder Hierarchy

