Critical Mass Game Bible

Brain Fizzle Games

Team Gnome Toss

Table of Contents

Contacts

Instructions

Create a contact list of all the members on the team. Their role should be which department they work under such Game Dev (GD), Game Art (GA), or Game Masters (GM).

Friday, December 16, 2011
1:06 PM

Name	*Role*	*Email*
Liam Hislop	Producer	liam@fullsail.com
Rod Moye	Design Lead	rmoye@fullsail.com
Joel Carroll	Art Lead	jcarroll@fullsail.com
Patrick Kelly	Scheduling Lead	pkelly@fullsail.com
Mike Lebo	Production Lead	mlebo@fullsail.com
TBD	Audio Lead	TBD@fullsail.com
TBD	External Producer	TBD@fullsail.com
TBD	External Art Director	TBD@fullsail.com

Name	*Role*	*Email*	*Phone*
Josh Villereal	Developer	jadam009@fullsail.edu	832 - 752 - 3197
Ryan Cartier	Developer	ryanc13@fullsail.edu	518 - 534 - 0892

Corey Morehead	Developer	coreyjamesmorehead@gmail.com	919 - 259 - 0153
Evan Wright	Developer	shugoikari@fullsail.edu	407 - 535 - 2408
Alex Garcia	Developer	alexandergenaro@gmail.com	352 - 442 - 8155
Ian Alcid	Developer	Hazards08@fullsail.edu	518 - 788 - 2662
Bob Pasekoff	Developer	bpasekoff@fullsail.edu	505 - 720 - 1996
Thomas Stefl	Developer	t_stefl@yahoo.com	610 - 739 - 2446

Team Charter

Instructions

The game charter provides information about how the team plans to execute its mission (create a video game). Every team member should agree to the contents of the game charter, as it defines the ***expectations***, ***guidelines*** and ***rules*** for the team.

Friday, December 16, 2011
12:23 PM

Objective

To complete a game project over a period of five months

Constraints

Quality, Time, Technology

Attendance

- All team members must contact the team if they are going to be late or miss a class or scheduled work days for any reason.
- Attendance is mandatory for everyone, for classes and scheduled work days.
- Attendance will be recorded by all members of the FP staff for class times.
- Attendance will also be maintained by a team log during scheduled work days through out final project.
- If by any chance a team members means of transportation are unavailable, they should immediately call other team members for a ride.
- All team members are expected to remain productive and on task during all scheduled work hours.
- Failure to remain productive for any reason may result in a verbal and written warning, documentation of absence in the team's documentation.

Dress Code

- Showers and cleanly hygiene are a must for every team member when coming to scheduled work days and class times.

Risks

- A team member may be lost during Final Project due to specific reasons such as being terminated from the project or have to take a leave of absence which as result will increase the work load and being behind schedule.
- Members of the team may not be able to complete the task by the dead line.
- Bugs may be present that do not allow for tasks to be accomplished in time and causing a lag.
- The project is over scoped and there is not enough time to execute the tasks required.
- The computer of one of the team members breaks or is stolen, not allowing them to do the work they are scheduled to do.

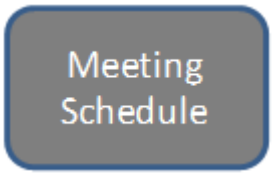
Assumptions

- All tasks will be completed by the deadlines that are set.
- All team members are able to complete the tasks assigned to them.
- Team members will complete all daily and weekly schedules by the end of the day.
- Schedule changes will be reflected and discussed as soon as possible.
- All team members will arrive on time to all meetings and work periods.
- The resources needed to complete a task will be available during the time the task is scheduled.
- Team members will be available to work on non-scheduled work days if needed.
- Team members will seek assistance before falling behind in their work.
- Team members will assist with travel expenses.
- The last hour of work will be used to review each others code for that day.
- If a reviewer does not understand a code segment, the writer will add comments until the requester is satisfied.
- If a team member is perceived to be taking too many breaks, a warning will be issued.
- Deliverables will be met as stated and as scheduled.

- Team members can only checkout items that they are actively working on.
- Team members must pull and verify code is working before every push.
- All team members will arrive on time in the mornings so that members can share information at the morning meeting.

Meeting Schedule

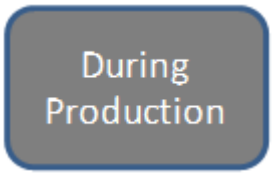
Friday, December 16, 2011
12:58 PM



Monday - Friday: 10am -> 6pm
Saturday: 10am -> 12pm (Code Freeze)
Sunday: Off
Locations: Team Member's Residence
Studio
Open areas on Full Sail Campus

During Production

Friday, December 16, 2011
12:58 PM



Objective
To complete a game project over a period of five months

Constraints
Quality, Time, Technology

Attendance

- All team members must contact the team if they are going to be late or miss a class or scheduled work days for any reason.
- Attendance is mandatory for everyone, for classes and scheduled work days.
- Attendance will be recorded by all members of the FP staff for class times.
- Attendance will also be maintained by a team log during scheduled work days through out final project.
- If by any chance a team members means of transportation are unavailable, they should immediately call other team members for a ride.
- All team members are expected to remain productive and on task during all scheduled work hours.
- Failure to remain productive for any reason may result in a verbal and written warning, documentation of absence in the team's documentation.

Dress Code

- Showers and cleanly hygiene are a must for every team member when coming to scheduled work days and class times.

Risks

- A team member may be lost during Final Project due to specific reasons such as being terminated from the project or have to take a leave of absence which as result will increase the work load and being behind schedule.
- Members of the team may not be able to complete the task by the dead line.
- Bugs may be present that do not allow for tasks to be accomplished in time and causing a lag.
- The project is over scoped and there is not enough time to execute the tasks required.
- The computer of one of the team members breaks or is stolen, not allowing them to do the work they are scheduled to do.

Assumptions

- All tasks will be completed by the deadlines that are set.
- All team members are able to complete the tasks assigned to them.
- Team members will complete all daily and weekly schedules by the end of the day.
- Schedule changes will be reflected and discussed as soon as possible.
- All team members will arrive on time to all meetings and work periods.
- The resources needed to complete a task will be available during the time the task is scheduled.
- Team members will be available to work on non-scheduled work days if needed.
- Team members will seek assistance before falling behind in their work.
- Team members will assist with travel expenses.
- The last hour of work will be used to review each others code for that day.
- If a reviewer does not understand a code segment, the writer will add comments until the requester is satisfied.
- If a team member is perceived to be taking too many breaks, a warning will be issued.
- Deliverables will be met as stated and as scheduled.
- Team members can only checkout items that they are actively working on.
- Team members must pull and verify code is working before every push.
- All team members will arrive on time in the mornings so that members can share information at the morning meeting.

Hours Worked Per Week

Friday, December 16, 2011
12:58 PM

Month One

Monday	*Tuesday*	*Wednesday*	*Thursday*	*Friday*	*Saturday*	*Sunday*
10	10	10	10	10	5	0

Month Two

Monday	*Tuesday*	*Wednesday*	*Thursday*	*Friday*	*Saturday*	*Sunday*
10	10	10	10	10	5	0

Month Three

Monday	*Tuesday*	*Wednesday*	*Thursday*	*Friday*	*Saturday*	*Sunday*
10	10	10	10	10	5	0

Month Four

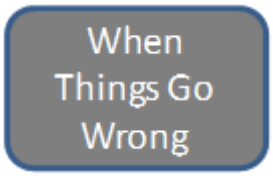
Monday	*Tuesday*	*Wednesday*	*Thursday*	*Friday*	*Saturday*	*Sunday*
10	10	10	10	10	5	0

Month Five

Monday	*Tuesday*	*Wednesday*	*Thursday*	*Friday*	*Saturday*	*Sunday*
10	10	10	10	10	5	0

When Things Go Wrong

Friday, December 16, 2011
12:58 PM



Team Member Illness

1. If a team member is ill and will not be attending the scheduled work meeting, the team member must notify the team via e-mail or phone call as soon as they are aware that they will be unable to attend the scheduled work meeting.

2. Team members who are capable of working are expected to show up and work, but will take proper precautions to prevent further infection.

3. If the illness is too severe for the team member to attend the scheduled work session, the tasks assigned to the sick team member will be completed from his/her personal residence.

a. Accommodations will be made on a case by case basis.

b. Team members working from home are required to join the team in the Skype chat at all times. [See 2: Communication section of the Design Documentation]

1. If a team member is unable to complete his/her tasks due to illness the Team Leads and will develop a plan to distribute those tasks to other team members.

2. If the team suspects that the team member is abusing an illness to avoid completing their work, we reserve the right to ask for a doctor's note that details the illness.

3. The team member that misses scheduled work days and meetings is responsible for contacting team mates for information regarding schedule changes, meeting locations, and tasks to be completed during the next scheduled work day.

Team Member Emergency Protocol

1. An emergency situation can include:

a. Death

b. Major contagious illness

c. Vehicular accidents

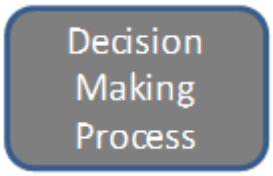
1. If a Team member experiences an emergency situation, they must contact the team as soon as possible via e-mail or cell phone.
2. The team will be informed of an emergency situation either by e-mail, phone call, verbally, or by text message.
3. The team will assume the possibility of an emergency situation if we have not received notification concerning absence from a team mate one hour after the meeting or work session start time.
4. All emergencies will be reported to the EP and the AD via e-mail.

Leaves of Absence

1. If a Team member has to take a leave of absence from Full Sail University during the course of Final Project, all of the tasks assigned to that team member will be redistributed to the remaining Team members.
2. The team member must notify the team if they are planning to take a leave of absence so the team can make the necessary arrangements concerning the schedule as soon as possible.
3. The team is responsible for creating a plan to redistribute the tasks of the team member taking a leave of absence to the remaining team members.

Decision Making Process

Friday, December 16, 2011
12:59 PM



Decision Making Process

1. If an issue arises during any stage of game development that the team cannot agree on through group discussion the following protocol will be followed:
2. A discussion in which any Team member who wishes to contribute to the conversation will state their suggestion for resolving the issue. The team member speaking will explain their reasons for their conclusions and the advantages that it has.
3. Team members will remember that the goal of this discussions is to share the "why" of a suggestion.
4. Other team members are expected to ask questions about their ideas.
5. Once a point is settled into concrete ideas, the choice will be put to a vote.
6. Issues that have already been voted on may be brought up for discussion again if new information or design decisions merit a second discussion and subsequent vote by the team.

Rules of Conduct

Friday, December 16, 2011
1:01 PM

1: General

1. All team members will adhere to the Rules of Conduct.

a. If a team member is discovered not following the terms set in the Rules of Conduct they will first be given a warning by the team.

i. This warning will be documented in a file by the team .

a. Should the aforementioned team member continue to refuse to adhere to the Rules of Conduct after two warnings, they will be reported to the EP, the AD, and the current Course Director via e-mail.

1. All Rules of Conduct are subject to change pending the approval of the team and the notification of all team members.

2. All team members will physically sign the most current iteration of the Rules of Conduct to signify their acceptance and willingness to adhere to the tenants within.

a. The Team Lead is responsible for storing a hard copy of an agreement with all team members' signature agreeing to the hereby stated Rules of Conduct.

b. Added rules must be agreed to by the EP and the AD.

1. Every team member will adhere to the Rules of Conduct as described in the Full Sail University Student Manual. [See Full Sail Student Manual for Rules of Conduct]

2. All team members are expected to behave in a professional manner at all times in accordance with the Full Sail University Student Manual. [See Full Sail Student Manual for Rules of Conduct]

3. Attendance and conduct will be accounted for during all team meetings and reported to the EP and AD.

4. Team members will display respectful behavior towards other team members at all times.

5. If a team member comes into conflict with another team member the two parties will follow the following protocol:

a. First they will discuss the situation amongst themselves and attempt to solve the issue.

b. If team members are unable to settle the dispute themselves, they will make the EP/AD aware of the conflict and give tem the opportunity to assist in the mediation process.

c. If the team judges that the conflict has gotten out of control and is beyond her authority to resolve, she reserves the right to inform the EP and the AD of the situation so they may assist the affected parties in coming to a solution.

1. Any team member that is disrespectful in any manner (through any form of communication – verbal or written) towards another team member will be written up by the team and reported to the EP, AD, and current Course Director via e-mail.

2. Aggressive behavior will not be tolerated under any circumstances.

3. Team members under the influence of any intoxicating substance will be dealt with immediately in a manner consistent with Full Sail's Code of Conduct and will be reported to the EP and the AD via e-mail.

4. Team members are expected to attend meetings at the time and place scheduled.

a. Unless a valid excuse has been given at least one hour prior to the meeting via a preferred communication channel, lateness will be documented.

b. If lateness or a lack of attendance becomes a problem with any team member, the team member will be reported to the EP and the AD.

2: Communication

1. Team members will be respectful to all Full Sail University employees including GP Staff members, receptionists, security officers, and course directors when using any form of communication (e-mail, text message, face to face meetings, Skype calls, and phone calls).

a. All communication with the abovementioned persons will:

i. Lack offensive content

ii. Represent the entire team in a professional and respectable manner

1. Texts will be preferred to communicate urgent messages to team members because it is the most immediate form of communication.

a. All team members are required to have the cell phone numbers of every other team member. [See Contacts section of the Design Documentation]

1. All team members are encouraged to keep their cell phones turned on and in hand in case of an urgent or emergency question is in need of an answer immediately.

2. Through cell phone calls, text messages, and e-mail team members are to communicate with the team about:

a. Arriving late during a workday (including the reason)

b. Health issues that impede the completion of tasks

c. Vehicular accidents

d. Urgent work related messages

e. Any other problem that a Team member considers an urgent matter for the team to know

f. Any questions on tasks that have been assigned

g. Team members are expected to reply to any texts they receive in a timely manner.

1. Cell phone use (including texting, phone calls, and Internet use) during work session is forbidden unless the Team member is expecting an urgent call.

a. If a Team member is expecting an urgent call they will inform the team of the nature of the urgent call and keep the team updated on the situation if necessary.

1. If the team changes the location or status of the meeting and or work sessions, team members will be notified by a cell phone call or text message. They will be informed them of:

a. The new location (if applicable)

b. The new meeting time

1. Team members are required to check their e-mail regularly when outside of team meetings.

a. 'Regularly' is defined as first thing in the morning and last thing before going to bed at night (at minimum). Ideally, team members will check their e-mails at least 3-5 times per day.

b. Team members will CC the team with every e-mail they send to other team members pertaining to the game or Final Project in general.

c. All team members are required to be on Skype if they are working from home, even if they are absent due to illness. [See When Things Go Wrong].

3: Team Attentiveness and Participation

1. All team members are expected to remain productive and on task during all team meeting hours.

a. Failure to remain productive for any reason may result in a verbal and written warning.

b. If a team member is assessed to be habitually non-productive for an excessive amount of time and has failed to return to task after receiving at least three warnings in a two hour time period, then the Team member will be reported to the EP and AD.

1. During team working hours, no team members will be using the Internet for gaming, Facebook, Twitter, Google +, Tumblr, or Youtube unless it applies to the work being accomplished.

a. Failure to adhere to this rule will result in a written report from the team to the EP and AD.

1. Distraction of other Team members from their workload via any means is unacceptable and will fall under the rules for non-productivity. [See Breaks for more information on team member distraction]

4: Breaks

1. Each Team member will be given a 15 minute break every two hours which they may use however they see fit (within the limits of the Full Sail University Manual).

2. If any Team members use this time for any recreational activity that may disturb teammates that have chosen to continue working (such as having conversations with other teammates, playing games with the sound on, etc.) they must move to the hallway or outside.

3. If food is consumed during the break it must be done in the hallway or outside as stated in the Full Sail University Student Manual. [See Full Sail University Student Manual]

4. Team members that return late from breaks (10 minutes after the return time) will be documented in a team log.

5. Smoking breaks and any other breaks that take more than 15 minutes needs to be reported to the team.

6. Each team member is free to take a 1 hour lunch between the end of the morning meeting and up to one hour before the code review.

5: Dress Code

1. Team members will arrive to classes and team meetings in clean clothes.

a. Team members will be expected to wear pants, shirts, and shoes at all times during classes and team meetings.

1. All team members must take daily baths or showers before they show up to work with the group.

2. Dress code for the team will be casual.

Team Roles

Friday, December 16, 2011
1:01 PM

Team Roles

Job Title: Tech/Project Lead
Name: Thomas Stefl
Job Description: Reports to the EP, and serves as the primary point of contact between the team and the EP.

Responsibilities:

- Talk to the EP
- Attend all meetings
- Be present for all meetings and code reviews
- Shares new information with the rest of the team
- Manages/Reviews HandSoft

Job Title: Quality Assurance Lead
Name: Corey Morehead
Job Description: Reviews bug list and ensures that bug reports have all required information

Responsibilities:

- Routinely checks bug report to ensure that it is being properly updated
- If a bug report does not contain adequate information, contacts the submitting team member and advises them on how to correct it.
- Prioritizes bug list

Job Title: Programmer

Name: Ian Alcid, Joshua Villarreal, Bob Pasekoff, Evan Wright, Ryan Cartier, Thomas Stefl, Alex Garcia, Corey Morehead.
Job Description: Writes and troubleshoots code in accordance with current milestone

Responsibilities:
•Ensure that all code is relevant and matches the design principles within the design document
•Oversee all aspects of code including: AI, physics, rendering, quality assurance, collision, animation, audio, special effects, and gameplay
•Develop high quality code for the game build
•Attend all team meetings on time
•Turn in all assigned deliverables on time and in full
•Communicate regularly with team members
•Take the time to help other team members solve problems
•Test created code segments for bugs before committing them to the build
•Maintain a positive professional work attitude no matter what the circumstances
•Contribute daily to the successful completion of Final Project goals and objectives
•Actively engages in code reviews and with the intent of improving our code

Build Procedures

Friday, December 16, 2011
1:01 PM

GP Staff Build Requests

- GP staff will request builds of the game as the team proceeds through the development cycle:

The Project Lead will ensure that the build is ready for review.
The build will be available within 4 hours of request.
A current build will be made at the end of every week and the build will incorporate all of the tasks completed in that week. It is essential that the build is kept as bug free as possible. Any bugs found should be immediately logged with the QA team.

Code Freeze

- 48 hours before a milestone turn in, the team will enter a bug freeze. No new features will be implemented during this period. The QA lead will assign bugs for team members to work. Bug changes must be thoroughly tested before being submitted.
- 24 hours before a milestone turn in, a build will be requested. This build will be turned in for that milestone. There are two exceptions to this rule:
    - Art assets that have not been integrated into the build yet must be added
    - A member of the GP staff requested that more work be done on the build before it is turned in for a milestone
- A code freeze will also be used during code reviews.

Code Review

- All members of the team will participate in the code review. The purpose of this review is to find issues in the coding that might cause larger problems in the future and ensure that everyone

involved understands the agreed upon coding standards and how the creation of the game is progressing.

- The code review will take no longer than 2 hours, but the actual time could be shorter depending on the number of tasks completed that day.
- Tasks will be reviewed in the order of priority - all A priority tasks will be reviewed before B priority tasks
    - If any team member requests, the writer will adjust or comment code to the requesters satisfaction.

Game Vision

Instructions

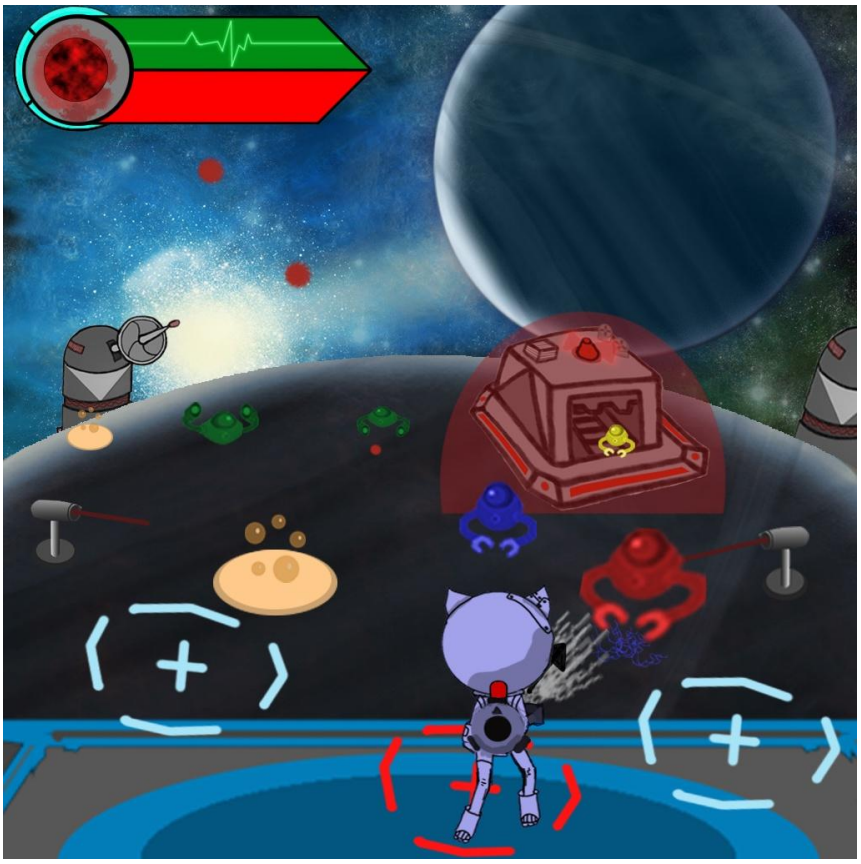
Take the ***Game Vision*** that you created for the ***Pitch*** (or an updated one), and place it in this section. Have a general ***concept*** piece that demonstrates what it would be like to play the game.

Thursday, December 08, 2011
10:43 AM

- **Critical Mass** is a fast paced **3rd person shooter arena** where you will battle waves of killer robots. You play as Kiki who, after having lost his entire regiment in sub-orbit , is left to fight alone against an army of automatons with only his special issue Portable Singularity Device (PSD).
- This unique military weapon behaves as a high power gravitational vacuum, able to inhale the smallest and health pickups. The gun's de-atomizing chamber disintegrates its victims into temporary usable weapon schemas, thereby enhancing its combat capabilities and altering the gun's devastating projectiles at a moment's notice.

- Push, pull, and shoot your way through hordes of homicidal 'bots on the surface of your home world's only moon. Wrest its future from cold mechanized claws and fulfill your duty as an officer of

the Planetary Defense Force (PDF).

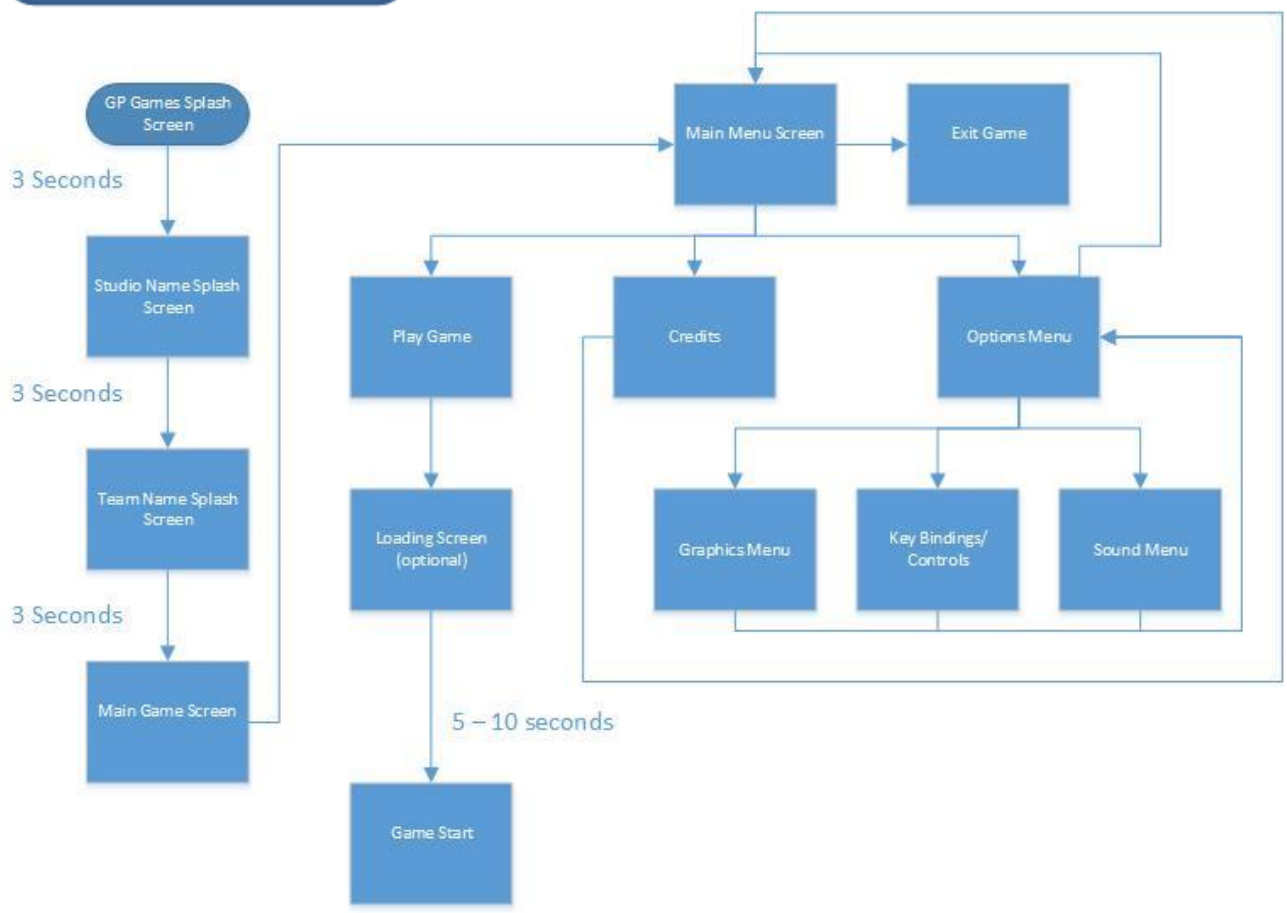


Front-end Flowchart

Instructions

- Create a ***flowchart*** that show the flow starting with the Studio Name screen and ending with your loading screen going into in-game. This should include ***every screen*** you have in the ***front-end*** and the ***logic*** for entering and exiting the screens.
- Thursday, December 08, 2011
- 10:46 AM

Front-end Flowchart



Front-end User Interface

Instructions

Create tabs on the right that list out every screen that is in the front-end of the game. The name of each tab should be the name of the screen. The Main Menu, Options, and Credits are already done for you because they are not optional.

Thursday, December 08, 2011
10:47 AM

Main Menu

Thursday, December 08, 2011
10:48 AM

Main Menu

What menu will look like when an option is highlighted. (Play Game as example)


CRITICAL MASS
PLAY GAME
OPTIONS
CREDITS
QUIT GAME
Ashley Olmstead

Options

Thursday, December 08, 2011
10:49 AM

Options

Main Options Menu. Will be broken up into a Graphics ,Audio and Controls sub-menus.


CRITICAL MASS
OPTIONS
GRAPHICS
AUDIO
KEY BINDS / CONTROLS
Ashley Olmstead

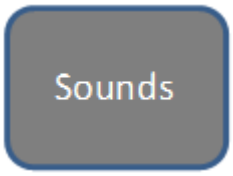
Credits

Thursday, December 08, 2011
10:49 AM

List of all the people that worked on the game.

Sounds

Friday, December 16, 2011
11:10 AM



- Need some sort of background music.
- When scrolling through the choices a type of short tone.
- When a choice has been entered an accepting sound.

Loading Screens

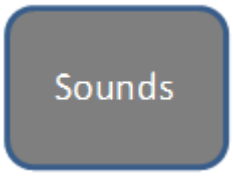
Thursday, December 08, 2011
10:51 AM

- Bar moves across as game is being loaded.
- Dots appear in order, then disappear all at once. This repeats till the game fully loads.

Sounds

Friday, December 16, 2011

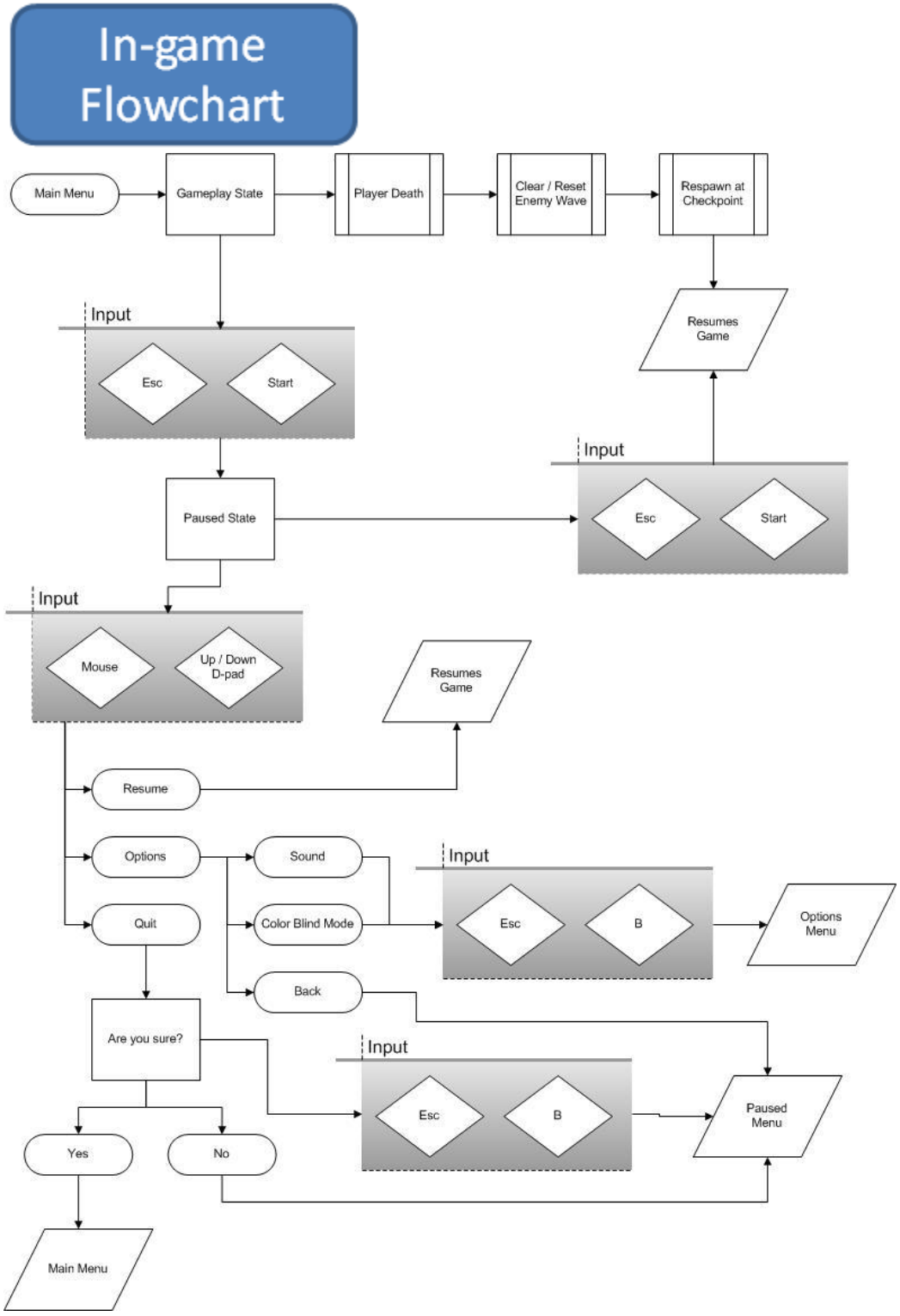
11:10 AM



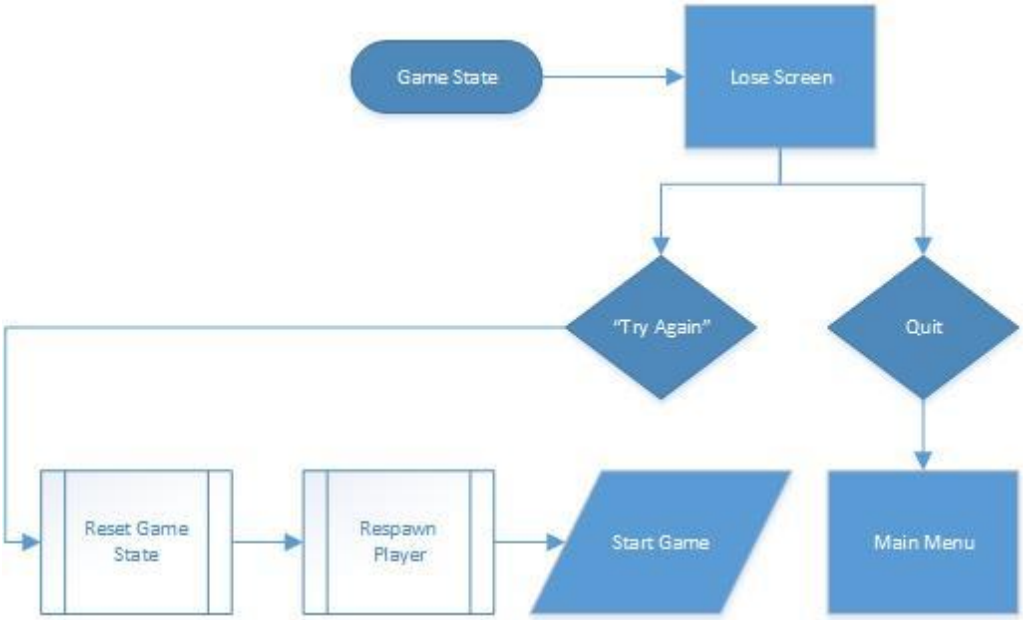
Background Music	Music from main menu continues to play when the loading screen starts and stops when the game begins.

In-game Flowchart

Thursday, December 08, 2011
10:52 AM



Pause Screen



Lose Screen

In-game User Interface

Thursday, December 08, 2011
10:53 AM

In-game User Interface

Design Concept:

-Same Design as Mani Menu

-Pause Menu appears

- Screen fads a little
- Action is paused

-Win Screen appears

- "Victory Accomplished" centered
- Few seconds later, Credits screen

-Lose Screen appears

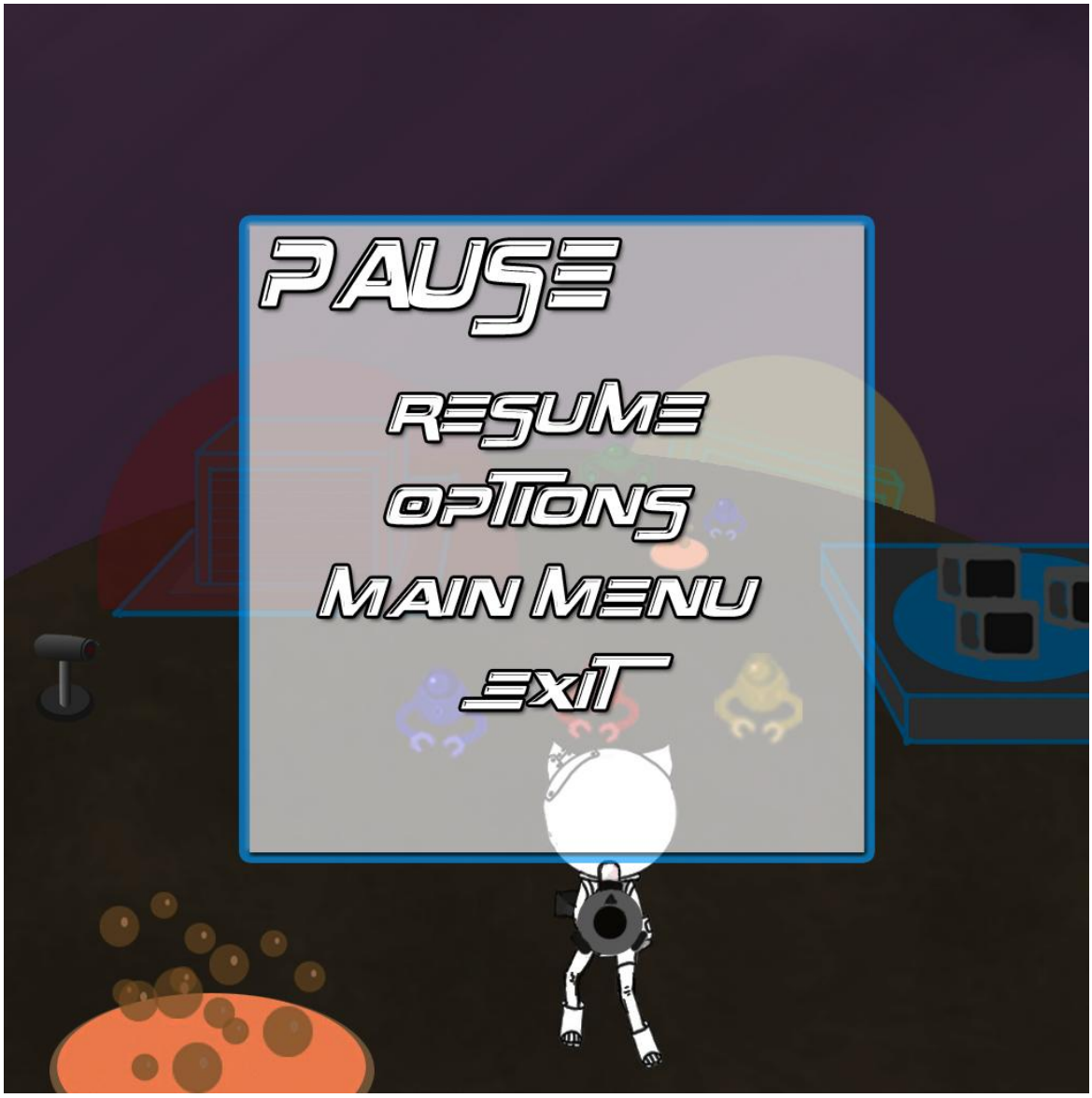
- "Game Over" centered
- Two options are given
  - Try again
  - Main Menu

-Same characteristic's as Main Menu

Win Screen

Lose Screen

Pause Menu



Thursday, December 08, 2011
10:54 AM

Pause Menu

Design Concept:
-Same Design as Mani Menu

-Pause Menu appears

Screen fads a little

Action is paused

In-game HUD

Instructions

The HUD system for the game includes things like ***health bars***, ***mini-maps***, ***combo meters***, ***message system*** (showing damage coming off an enemy), etc… These items are drastically different depending upon the game genre (FPS, RTS, RPG, Puzzle, etc). It is basically every interface piece that is present when the player is actually playing the game. Create ***concept images*** of every piece. Create tabs on the right that list out every different piece of the HUD.

Thursday, December 08, 2011

10:55 AM

Mini map.

Total health remaining.

The circle shows the power up you currently have or if you have something in your chamber it shows what that is.

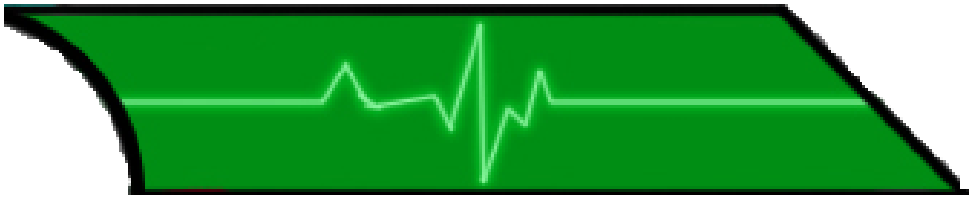
Outside the circle are three bars that represent your lives remaining and the total amount of energy left for your power up.

Health Bar

Wednesday, June 19, 2013

9:14 AM

Health bar when player is at full hit points.



Health bar when the player is at half hit points.

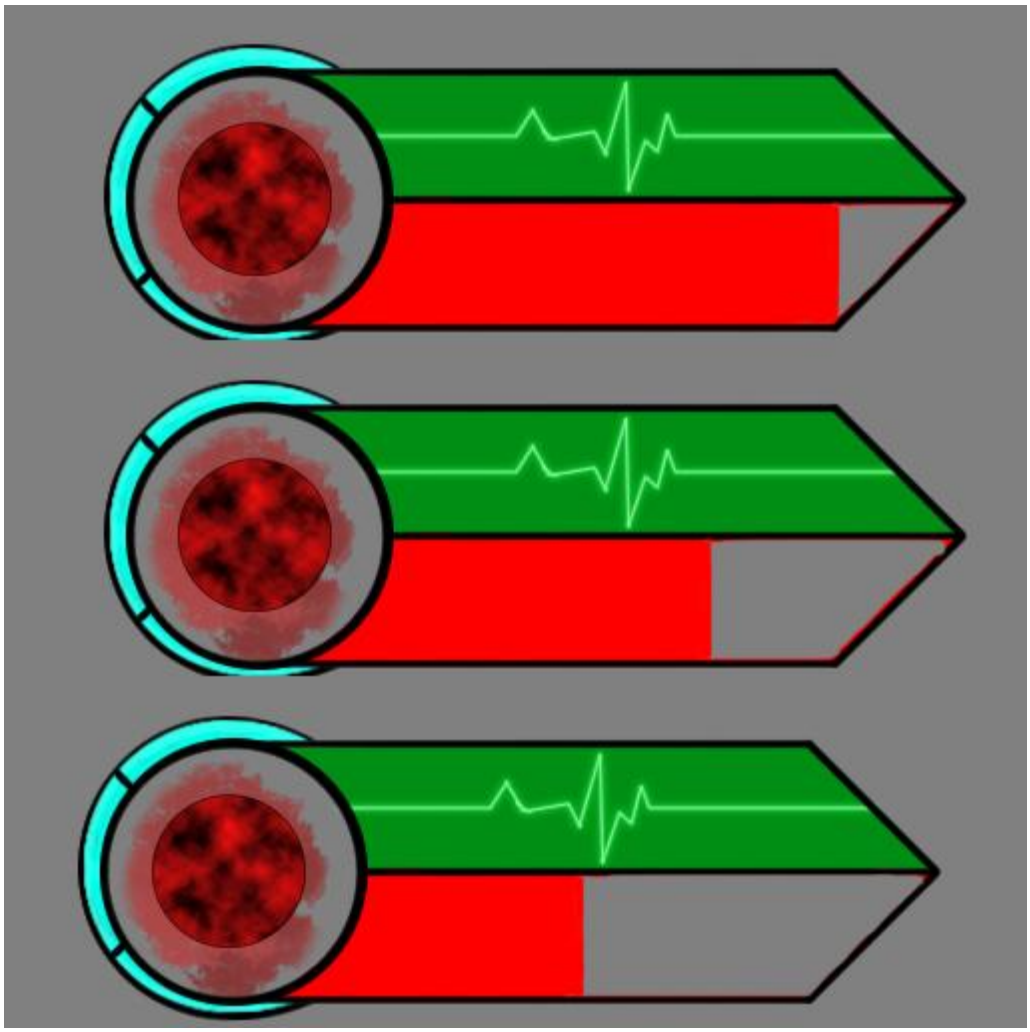
Health bar when the player is at low hit points.

Ammo Bar

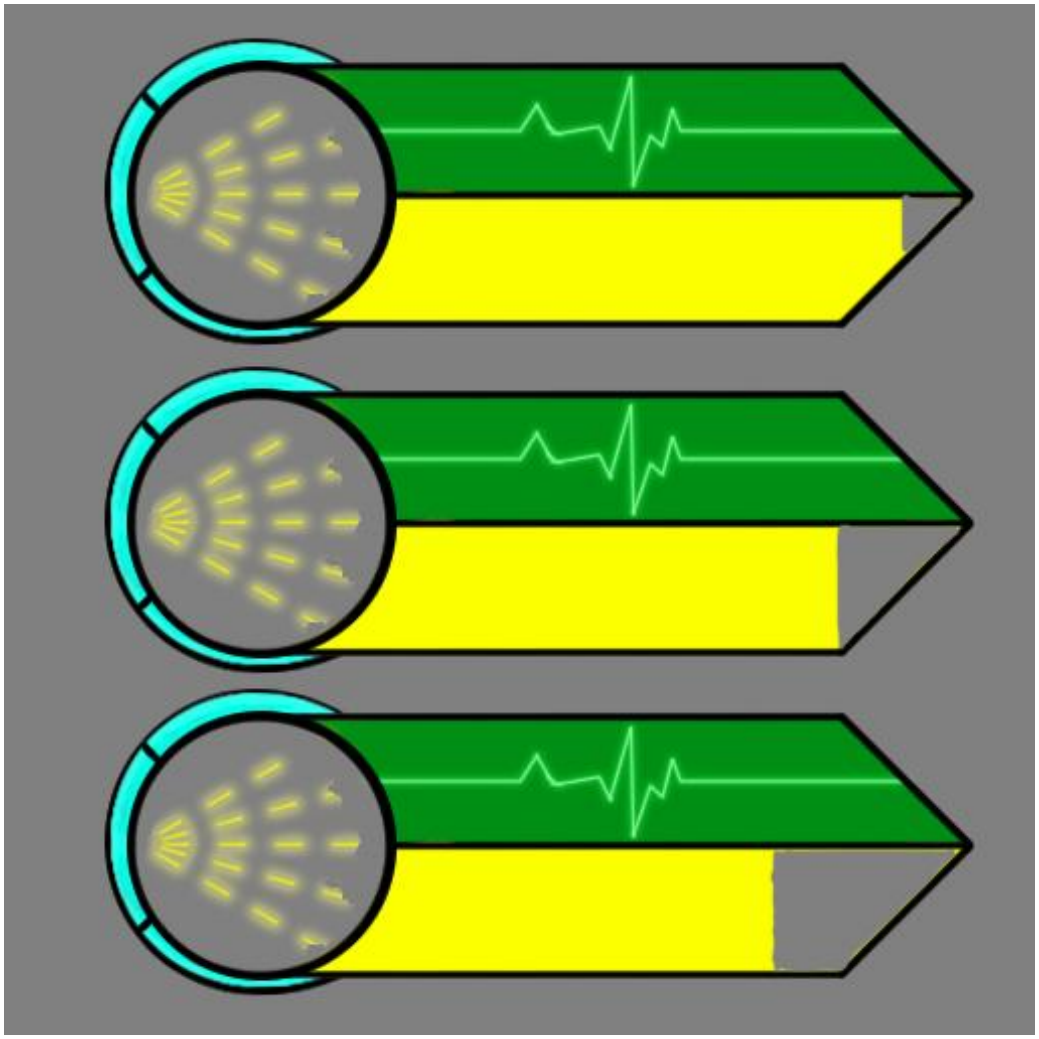
Ammo bar when the player has the red power up.

Wednesday, June 19, 2013

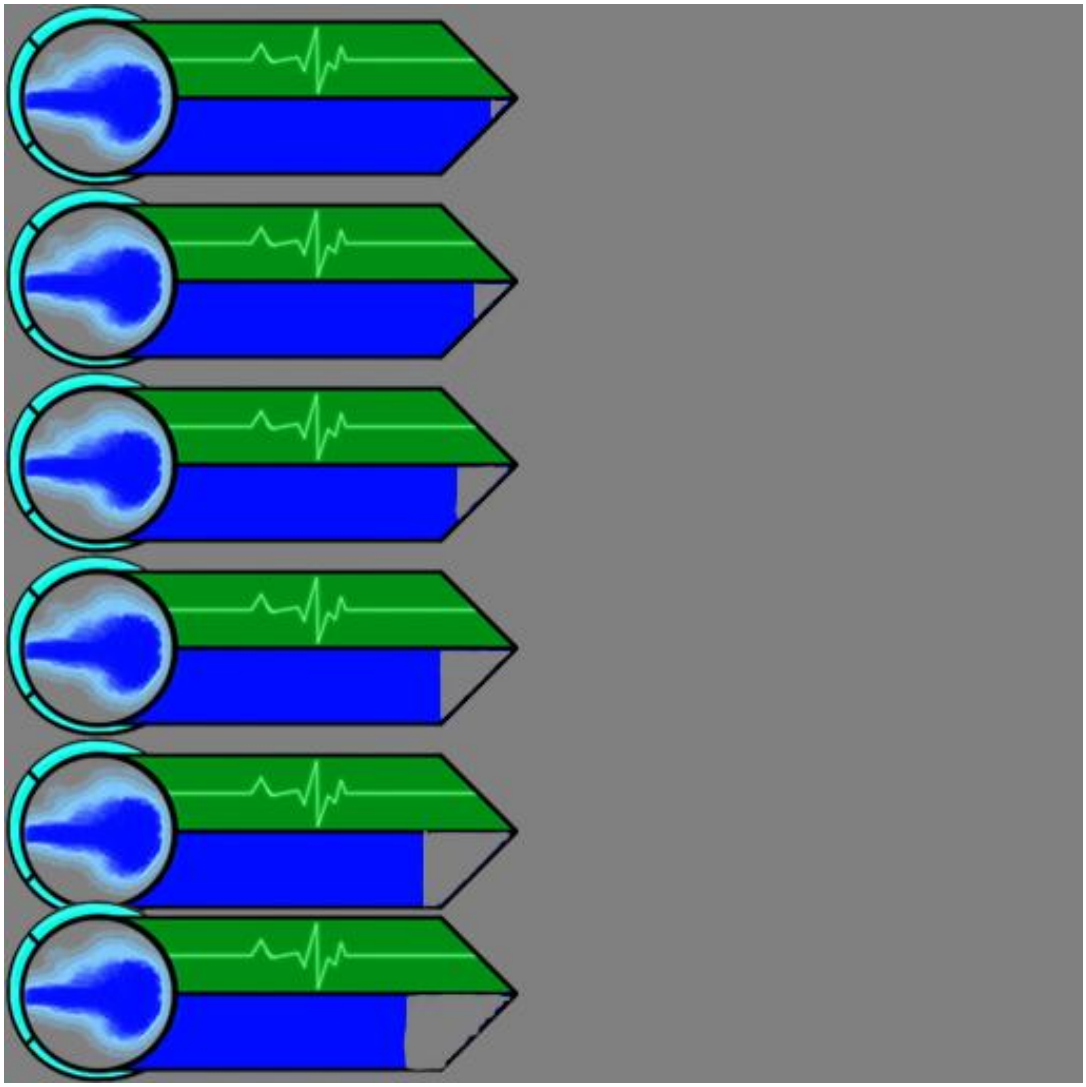
9:15 AM



Ammo bar when the player has the yellow power up.



Ammo bar when the player has the Blue power up.

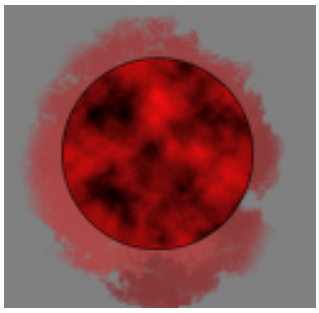


Ammo Icon

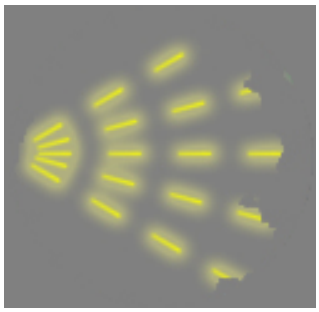
Icon for the Red plasma bomb power up.

Wednesday, June 19, 2013

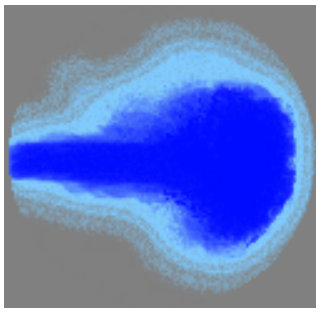
9:15 AM



Icon for the Yellow shotgun power up.



Icon for the Blue laser beam power up.

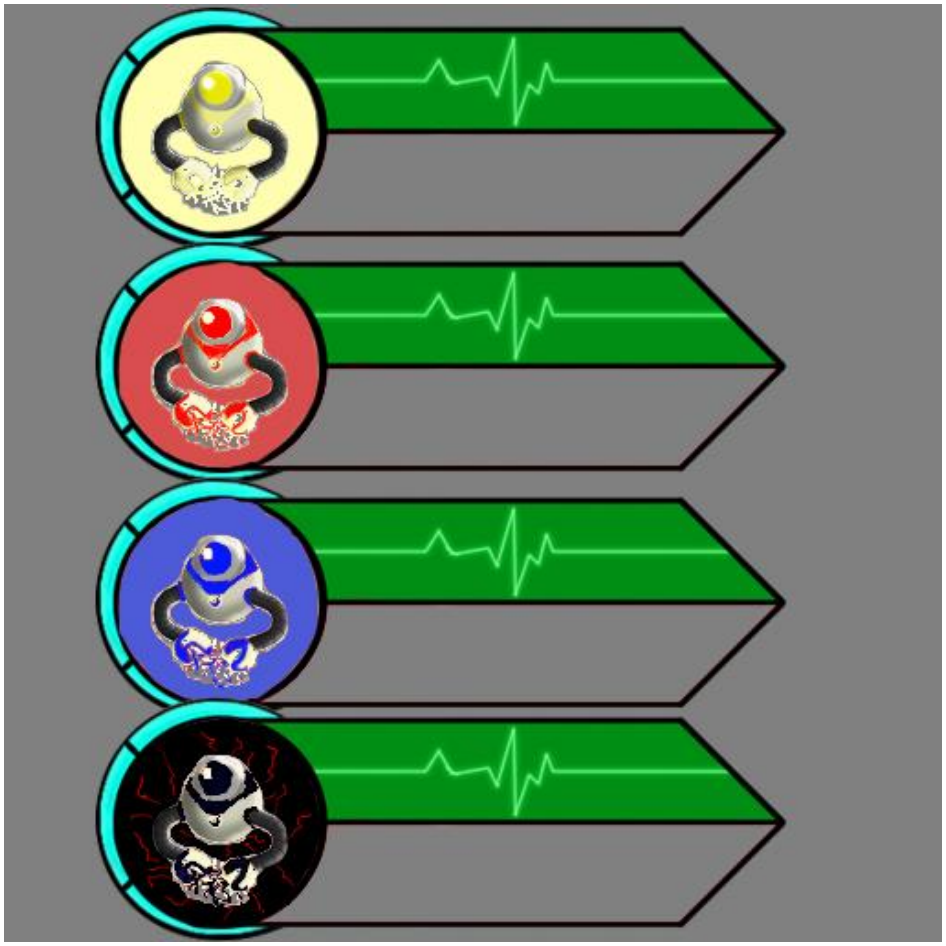


Singularity Chamber Icon

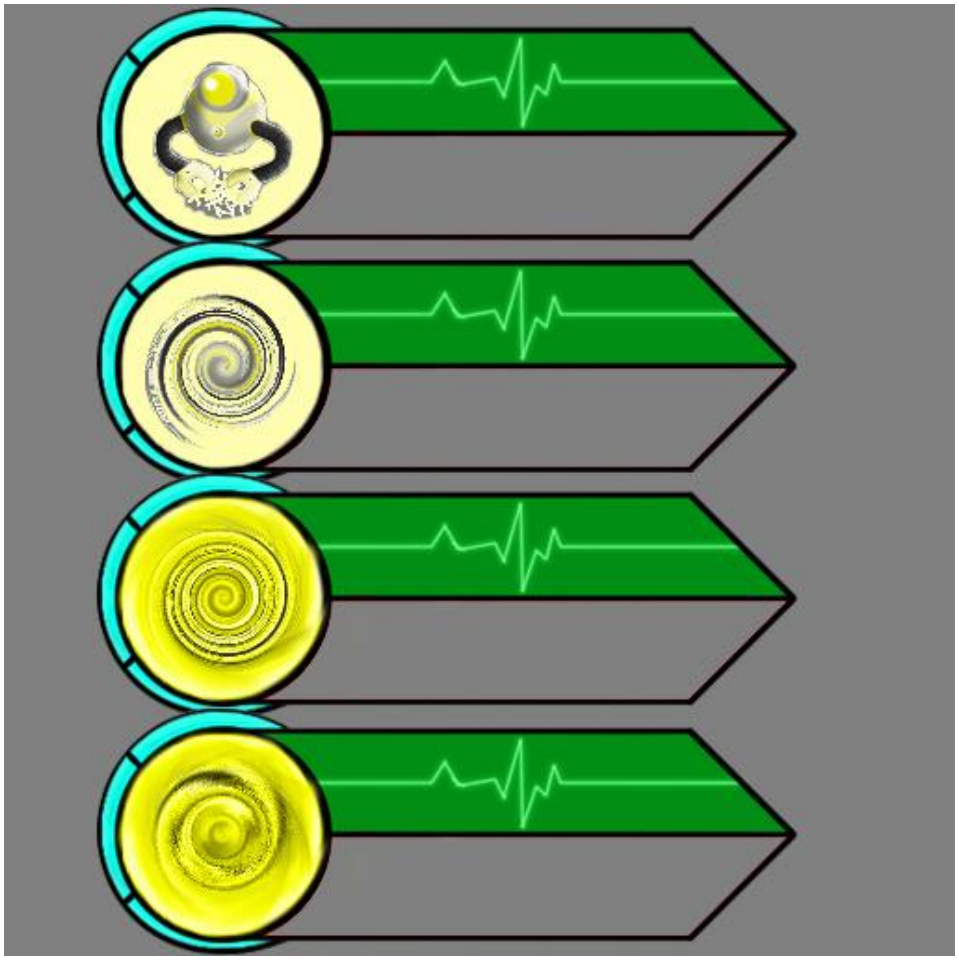
Thursday, June 20, 2013

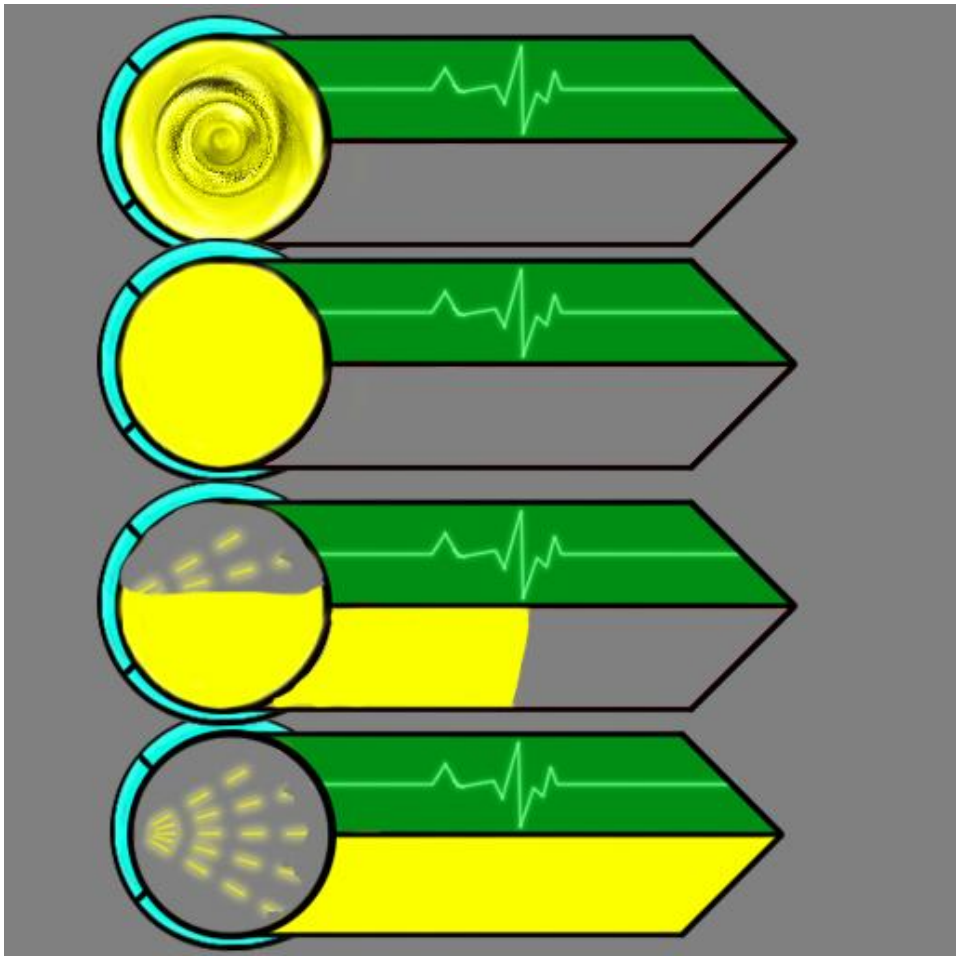
11:36 AM

Icons for each of the 4 small enemies that you can pull into the gun, to represent what you currently have waiting to be consumed.

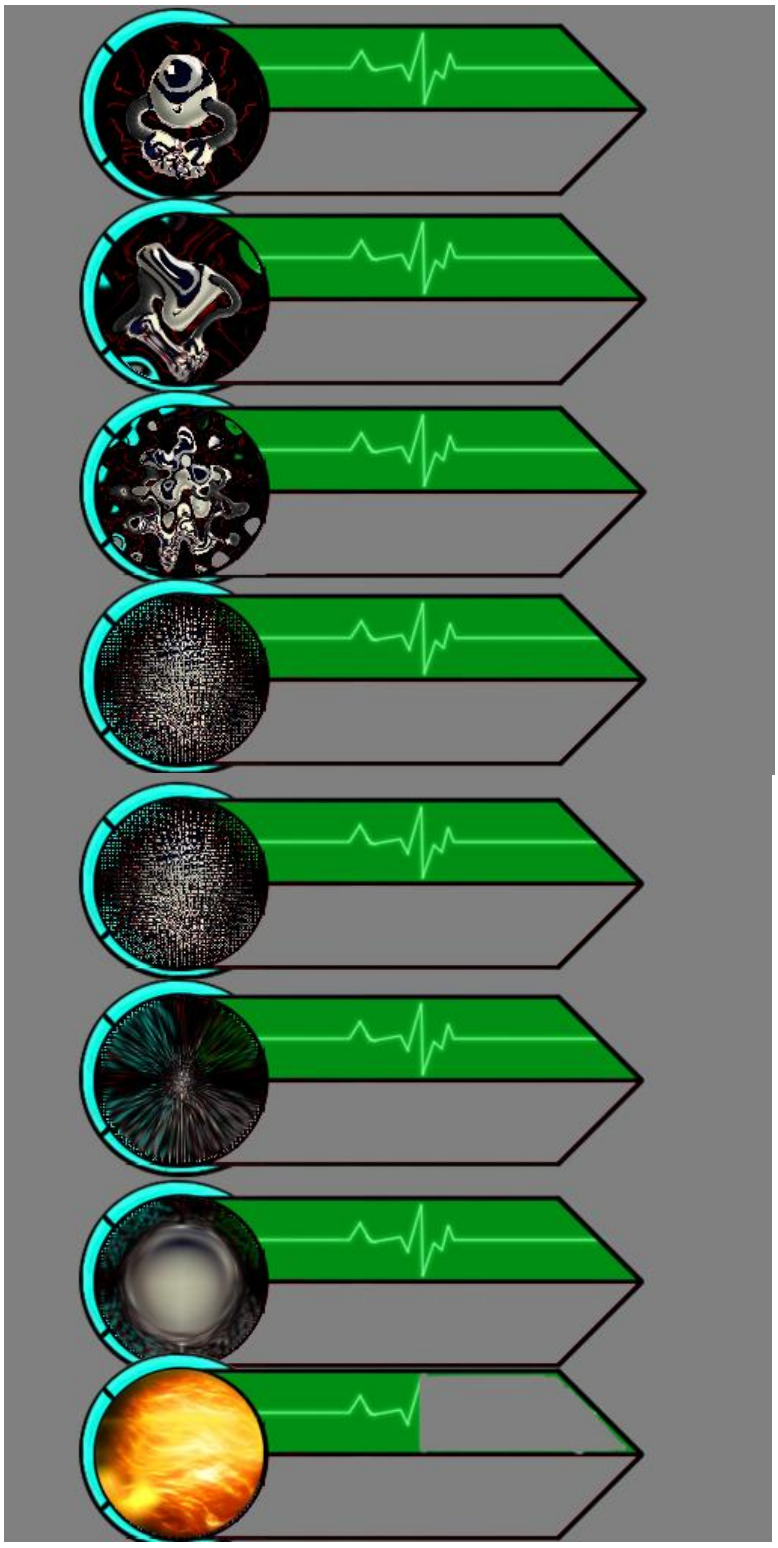


Power up being consumed.





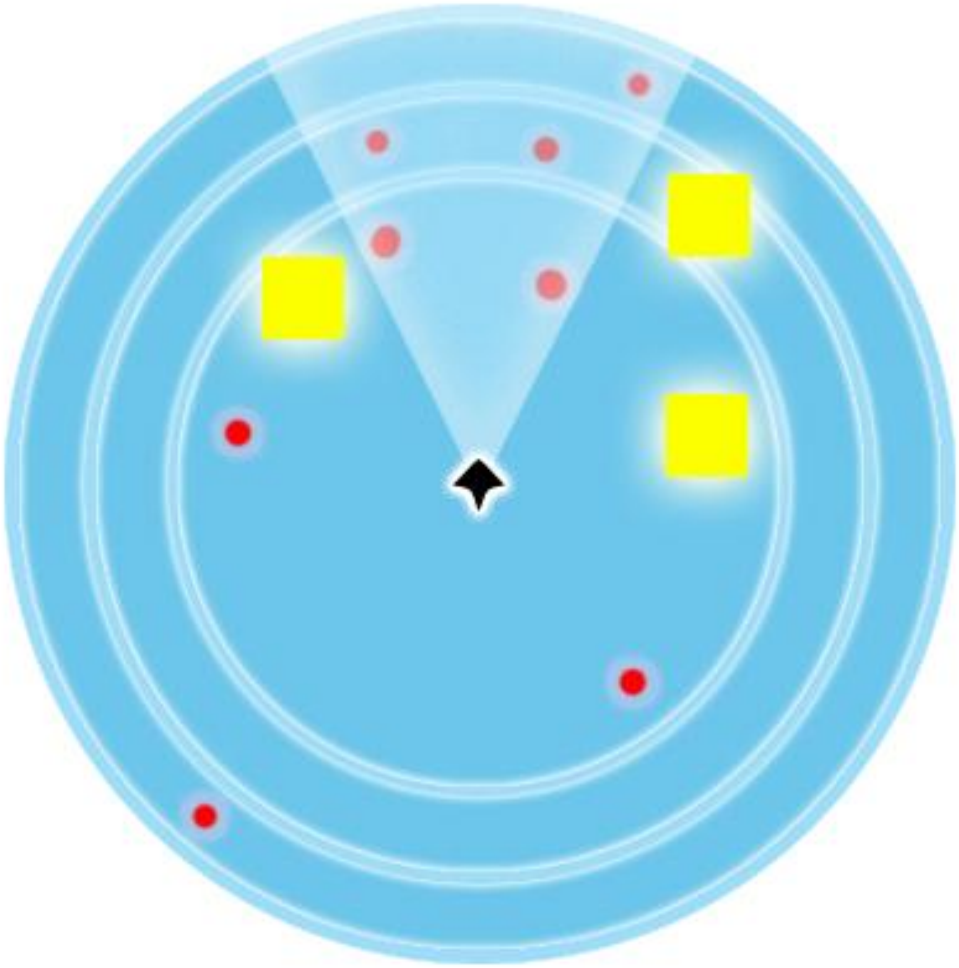
Bomb enemy exploding in the chamber.



Mini Map

- The red dots represent the enemies.
- The yellow squares represent the objectives.
- The dots and squares inside the circle represent enemies and objectives that are within a radius of the players.
- The dots and squares outside the circle represent enemies and objectives outside of the radius around the player, used to show the player where to find enemies and objectives they can't see.

Wednesday, June 19, 2013
9:15 AM



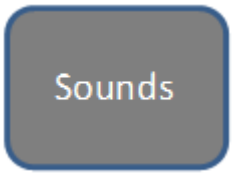
Sounds

Instructions

List out all the sounds that are needed for the HUD elements.

Friday, December 16, 2011

11:11 AM



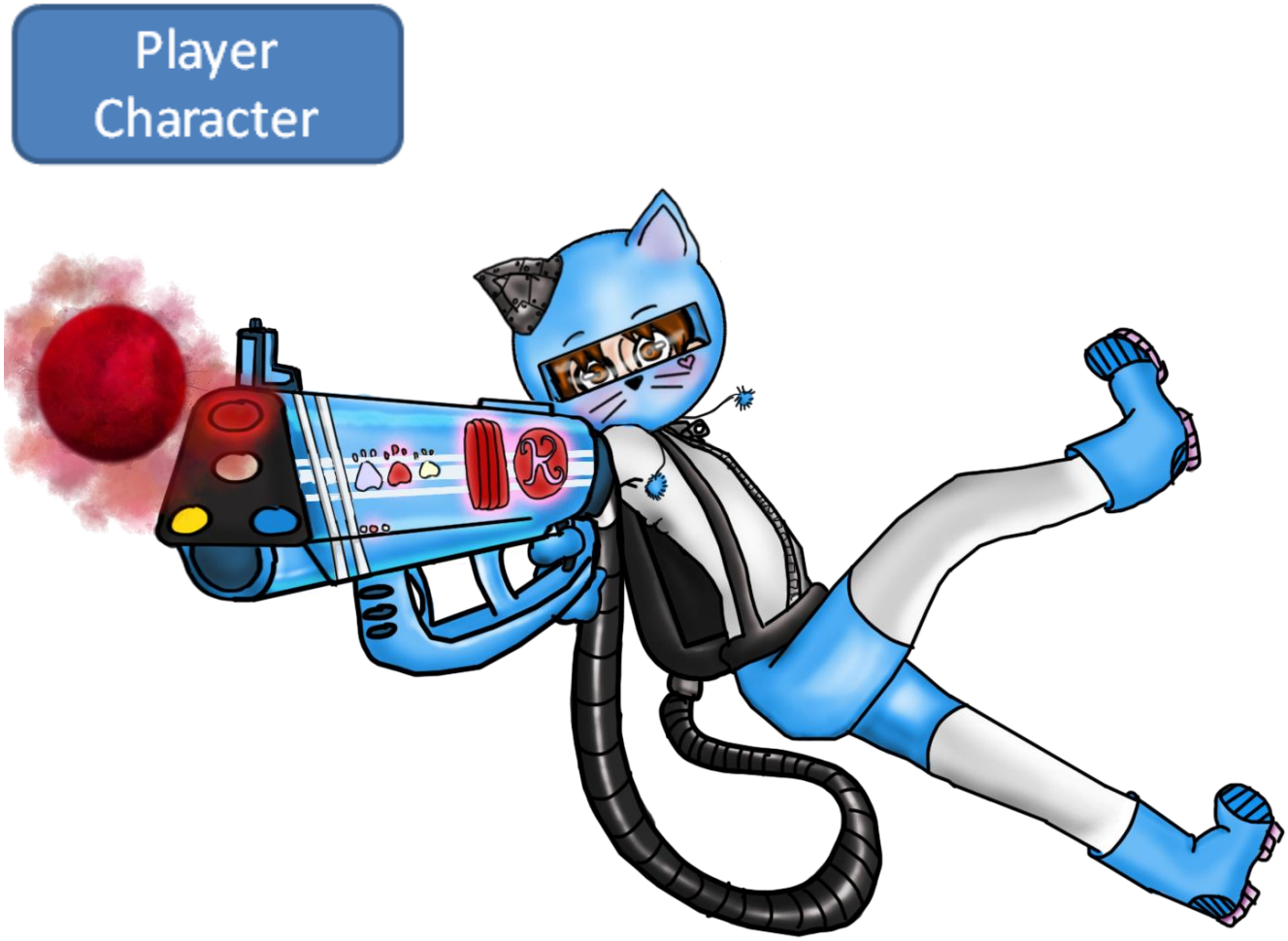
Sound	Description
Low health sound.	Blinking sound similar to borderlands.

Player Character Overview

Instructions

Describe the character in terms of ***general appearance*** (young girl, gritty old man, etc) and list out ***general attributes*** (fast, slow, etc).

Thursday, December 08, 2011
10:56 AM



Play as Kiki, the daughter of the late Admiral Hoover and an aspiring cadet in the Planetary Defense Force. Hoping to live up to her family name, Kiki takes up her missing father's unfulfilled mission to investigate and safeguard a close orbiting moon to one of the colonized worlds. Discovering it to be covered in a hostile mechanize army, Kiki must use her training as a cadet to defeat the invaders.

Armed with the uniquely advanced weapon of the PDF, Kiki's Portable Singularity Device is a multipurpose anti-gravity siphon. Its dual polarity allows Kiki to pull and vacuum up objects or push

them away at high speeds. The weapon's Singularity Chamber can swallow small objects whole and then disintegrate them into usable weapon schemas that can change how the weapon behaves on the fly.

The player can use both of the guns functions throughout the game. Pull can be used to reel in and consume smaller enemies and pick-ups. The affected can they be "eaten" which destroys the object and grants a power up. The power up granted is dependant on the color of the enemy consumed.

Push is used to grant space between the player and their enemies. It can even stun unshielded enemies. When an object has been taken into the Singularity Chamber Push can be used to expel the item at high speed, turning it into a deadly projectile.

Kiki is an average height and light weight. Her hair and eyes are a deep brown. She wears a stylized enviro-suit of white and blue.

Gender:	Female
Age:	23
Height:	5'8" (~2.5cm)
Basic Weapon:	Portable Singularity Device (PSD)
Alternate Fire Mode:	Push
Alternate Fire Mode:	Pull

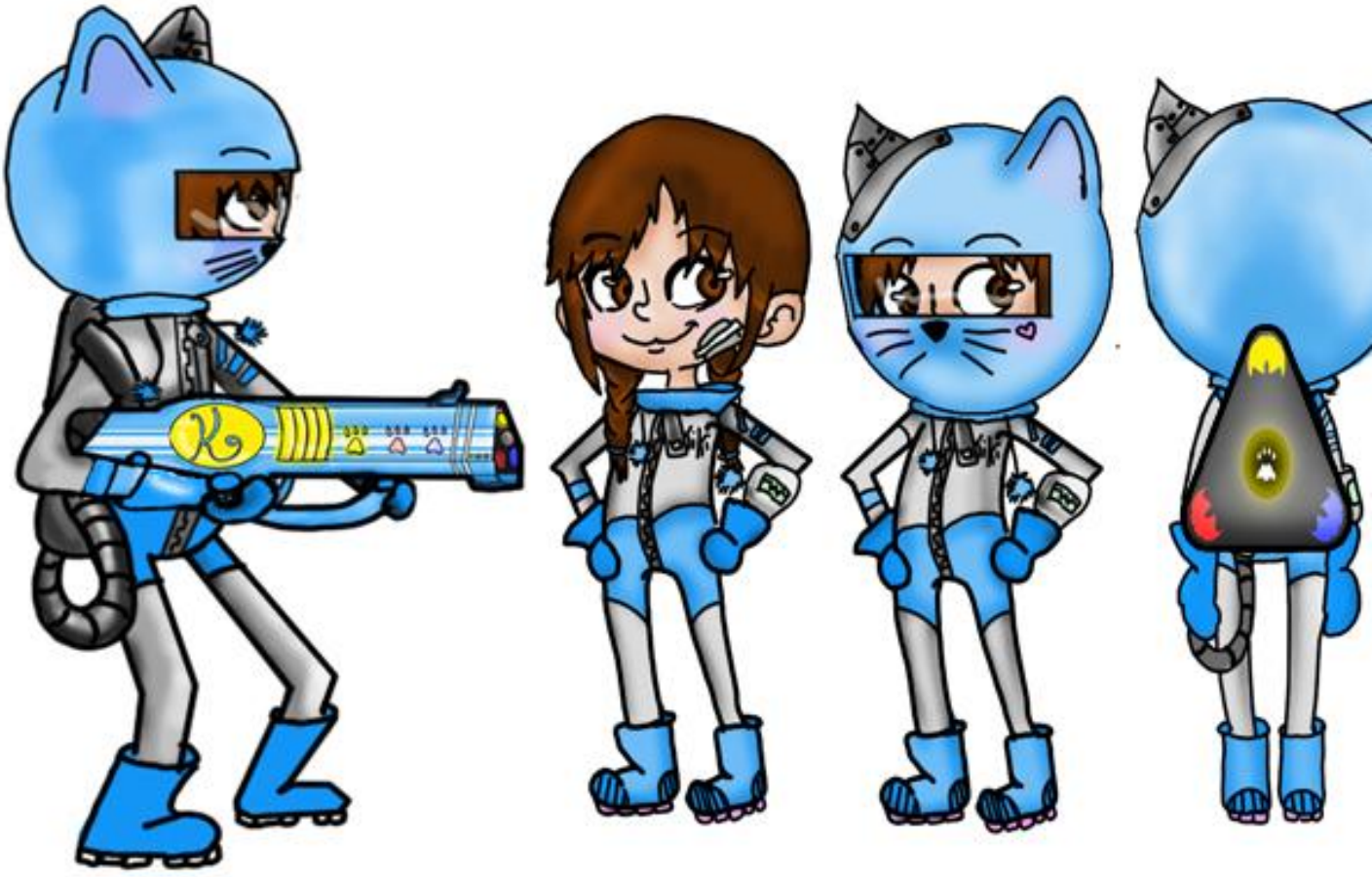
Concept

Instructions

Create front, back, side, and top down view of the character.

Thursday, December 08, 2011
10:57 AM

Concept



Scale

Instructions

Create a scale based on the units the game is based on and show the character's relation to it.

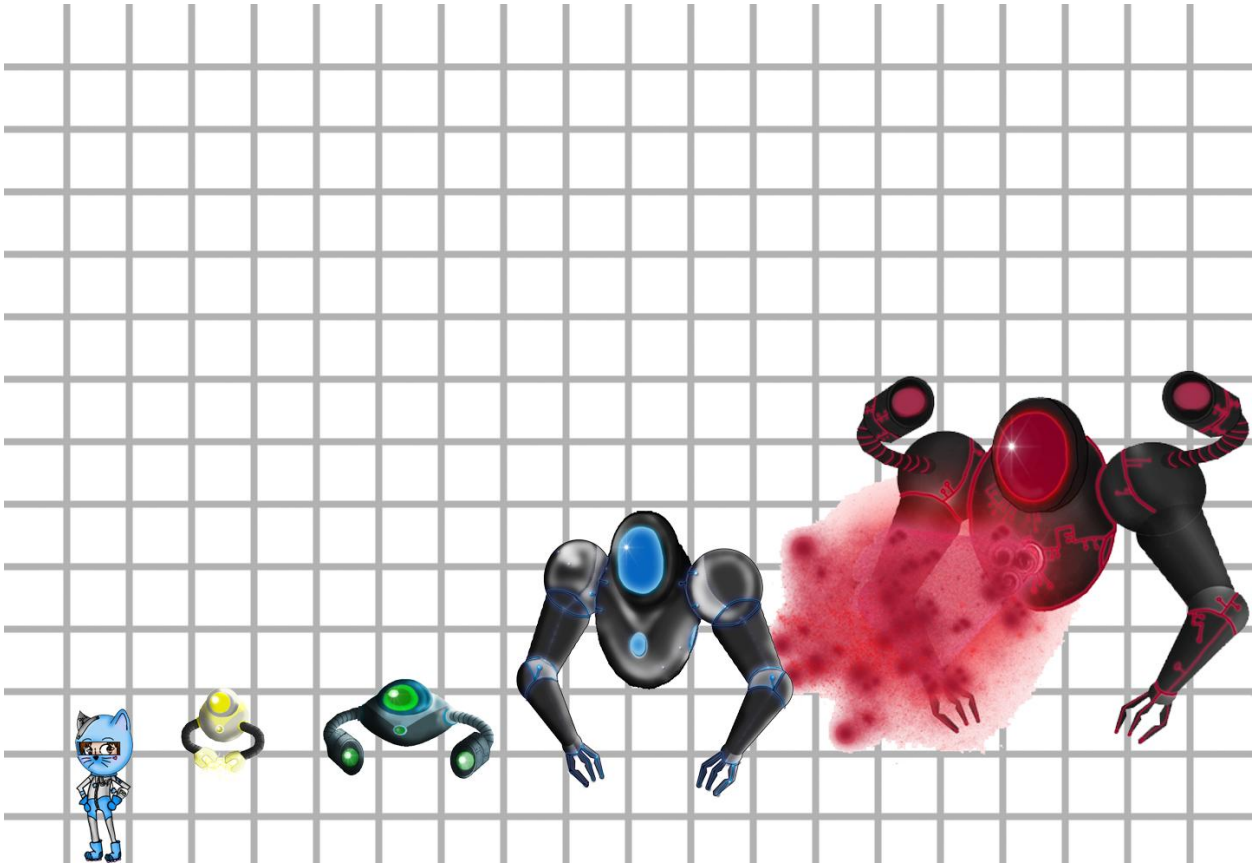
Thursday, December 08, 2011

10:58 AM

Scale

The player is approximately 2.5cm

The moon is 42cm in diameter.



Attributes

Instructions

List the attributes that makeup the character. Attributes like speed, health, damage, etc.

Thursday, December 08, 2011
10:58 AM

Player Attributes

Health	Remaining Health of the player. When at 0, the player dies.
Ammunition	Represented as a bar sized by the percentage of remaining ammo.
Lives	When a player is killed a life is consumed. When at 0 lives, Game Over.
Footprint	The player occupies roughly 1 square cm when standing.
Height	The player is 2.5 cm tall.
Speed	The player moves at roughly 4 cm/s

Weapon Attributes

Weapon	**Speed**	**Damage**	**Notes**
Plasma Cannon	Slow	Heavy	Fires a slow moving red ball of light. Explodes on impact dealing a medium amount of splash damage to all within a local radius.
Laser Beam	Fast	Very Small	Projects a straight beam of blue light. Damage is small but is dealt very quickly. When fired the beam can be sustained and moves with the player.
Shockgun	Medium	Medium	Fires a collection of yellow buckshot charged with electricity. Damage is inversely related to the distance between the muzzle of the gun and the target. (I.E. the close the target, high damage)
Push	Slow	N/A	A defensive capability of the weapon used to repel smaller enemies. Can stun unshielded enemies and repel the plasma

			shots of enemies. Has a high cool down of 5 seconds.
Pull	See Note	N/A	A primary sustained fire capability of the weapon. Used to pull small objects to the wielder. When the object comes in contact with the muzzle of the gun it is placed inside the guns chamber.

Note: The speed at which an object is pulled from their location to the gun can be manipulated. While sustained, the player can repeatedly hit the 'A' button to speed up the rate at which an object accelerates towards the gun. The speed of the pull starts at the half the player's run speed or 2cm/s.

Actions + Animations

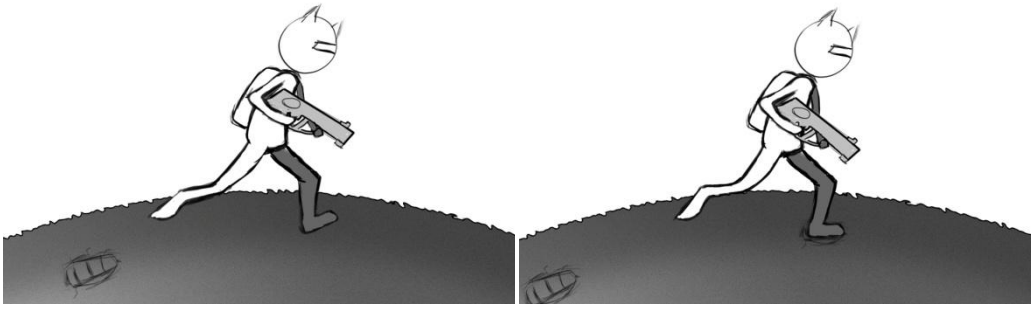
Instructions

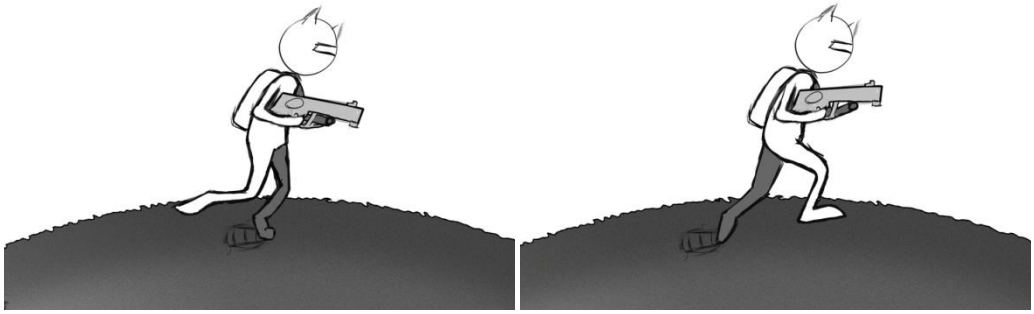
List out the actions the character does, how the player executes each one (what button do I press), and give concepts of the actions being played out.

Thursday, December 08, 2011
11:06 AM

Movement:

Player runs over the soft surface of the moon. Behind her, she leaves foot prints in the moon dust. The play moves using the W A S D keys on the keyboard or by using the left joystick on the controller. A and D keys as well as Left and Right on the joystick will cause the player to strafe sideways, maintaining their current facing direction.





Running

Strafing

Camera:

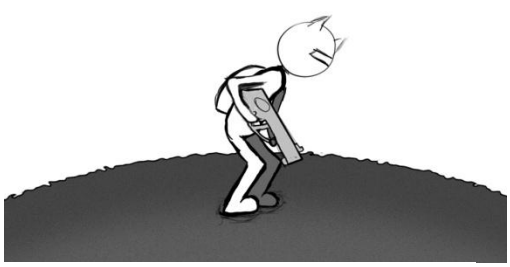
or

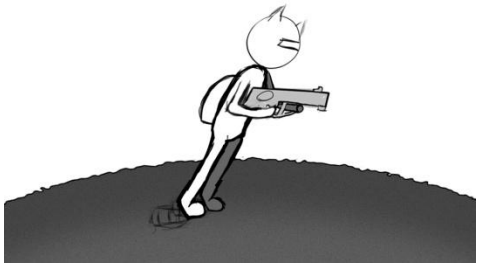
The player can manipulate the camera in any direction. They player's avatars serves at the camera's center pivoting point. The player can manipulate the camera using either the Right Joystick on the controller or my moving the Mouse. The player's firing reticle will always be in the center of the camera.

or

Jump:

The player can jump by pressing the SPACE BAR on the keyboard or the A button on the controller. The player can use jump to traverse obstacles as well as avoiding enemy fire and hazards. The moon's weakened gravity allows the player to jump higher than usual. This is an exaggerated and longer lasting jump, keeping the player airborne for a little more than a full second.

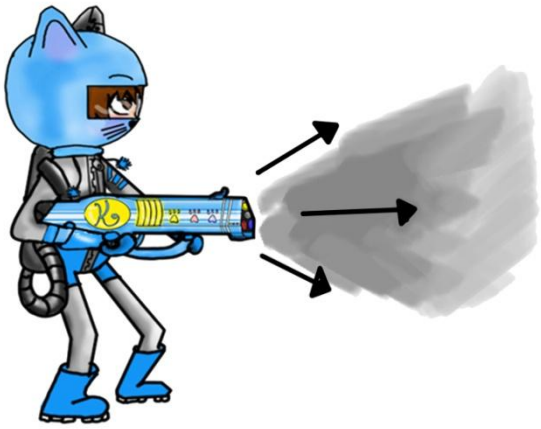


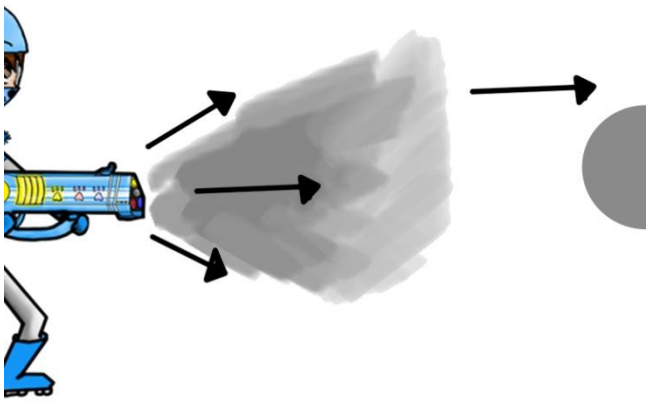


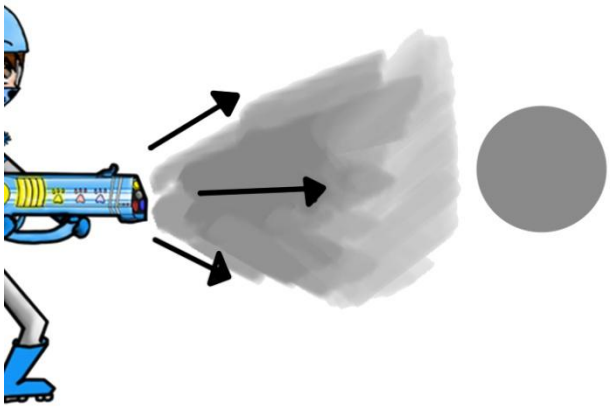
or

Push:

The primary defensive game mechanic. The player can activate the push by pressing the Right bumper on the controller or by pressing Shift. This will produce a short range forward conical blast of influence. All small objects and smaller enemies affected are pushed away from the source of the blast. The player may also use this to deflect direct ranged projectiles fired at them.





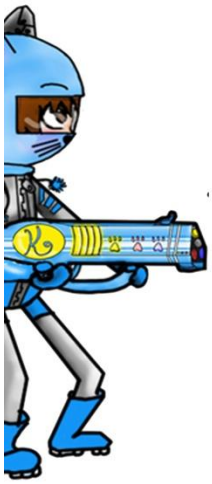


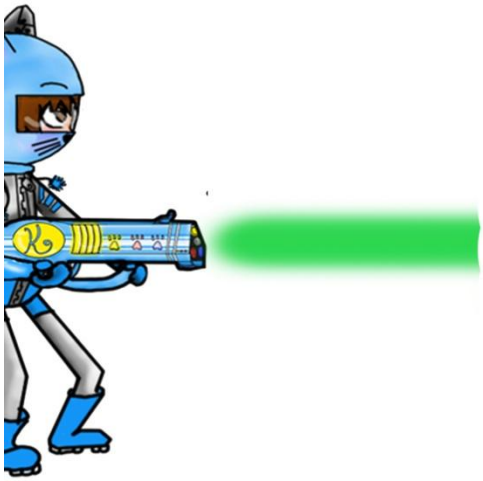
;

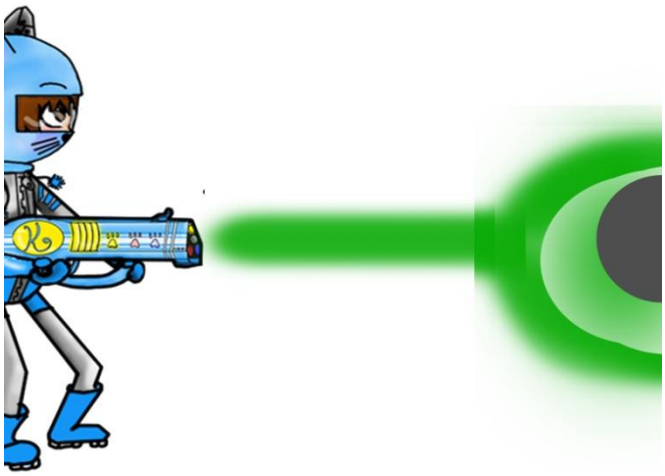
or

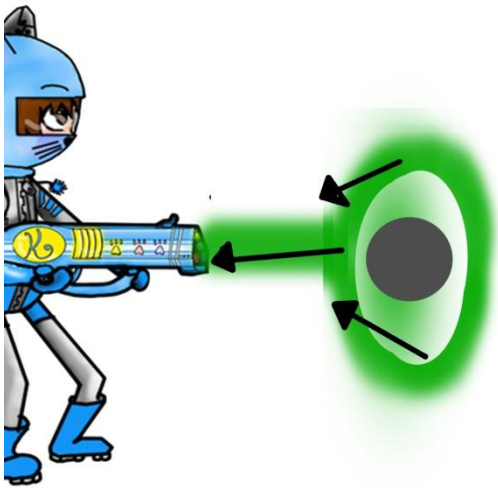
Pull:

The offensive utility aspect of the gun. The player envelops their target in a beam from the muzzle of the gun. This can be activated by pressing and holding the Left Trigger button on the controller or by pressing the Right Mouse Click. The target is pulled towards the player's gun. The player can repeatedly press the A button on the controller or SPACE BAR on the keyboard to increase the speed at which the affected is pulled towards the gun. Can be used on small enemies as well as pickups.





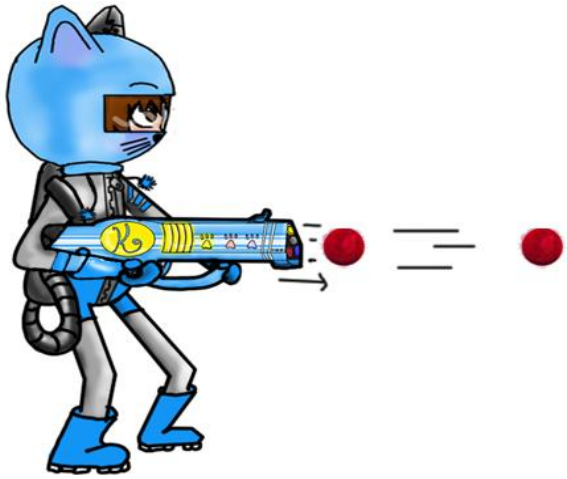




or

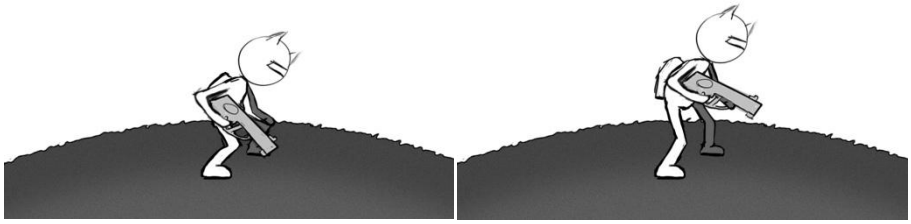
Shoot:

The primary offensive ability of the player. The player can fire various projectiles from her weapon by pressing the Right Trigger or the Left Mouse Click. The projectile is fired down the camera's reticle at whatever it is looking at. The player is able to shoot three different ammo types as well as what ever object is currently being held in the gun's chamber.



Taking Damage

The player takes damage when she collides with an environmental hazard or is hit with an attack from an enemy. The amount of damage dealt to the player is determined by the severity of the hit .

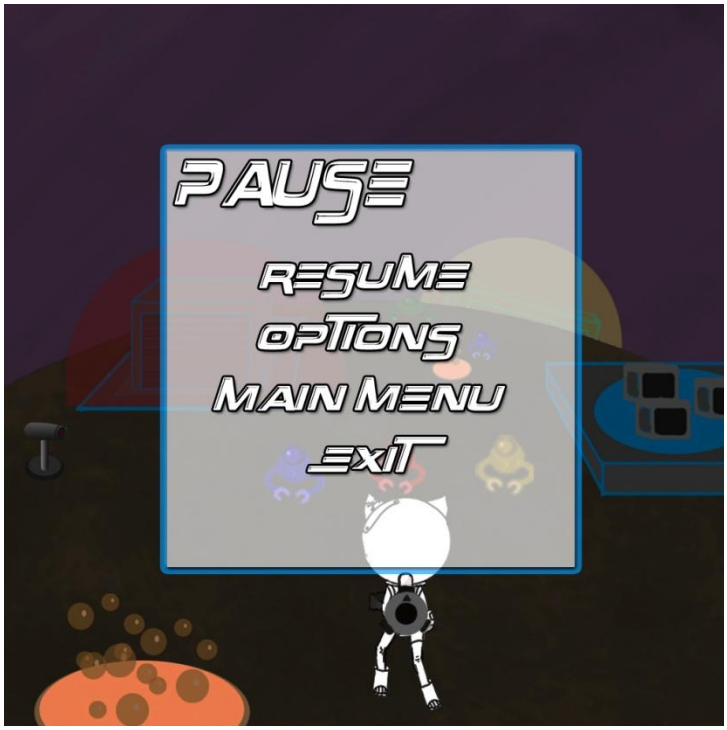


or

ESC

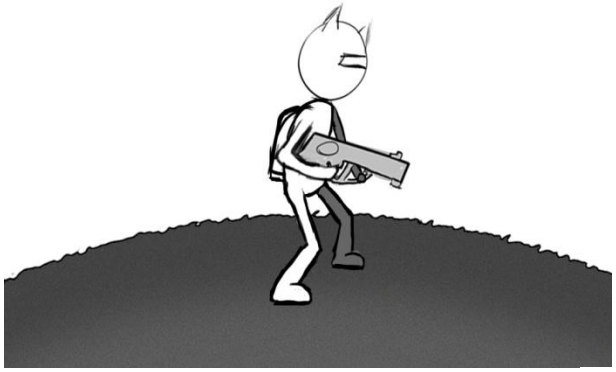
Pause:

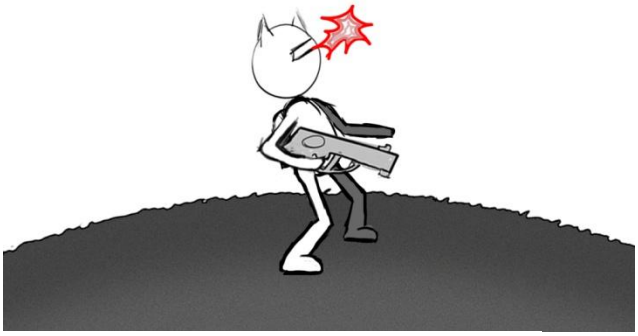
At any time during the game play the player may pause the game. This will stop the game actions and open the Pause Menu for the player to browse and adjust various options as well as quit the current game session.

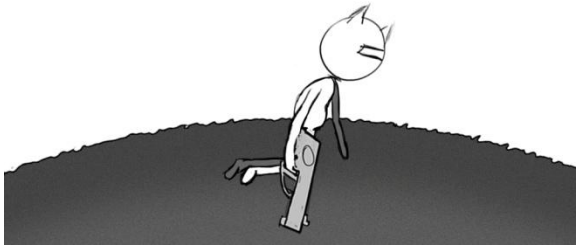


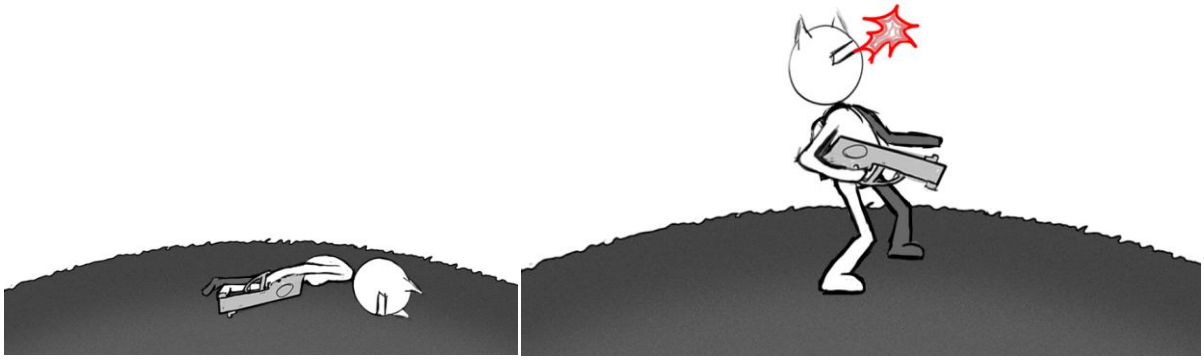
Death

Once the player's health bar reaches 0 (or is empty) the player falls down. This consumes a "life" from the player's HUD. The player will then respawn at one of the landing pads at either the North or South pole of the moon.









Prop(s)

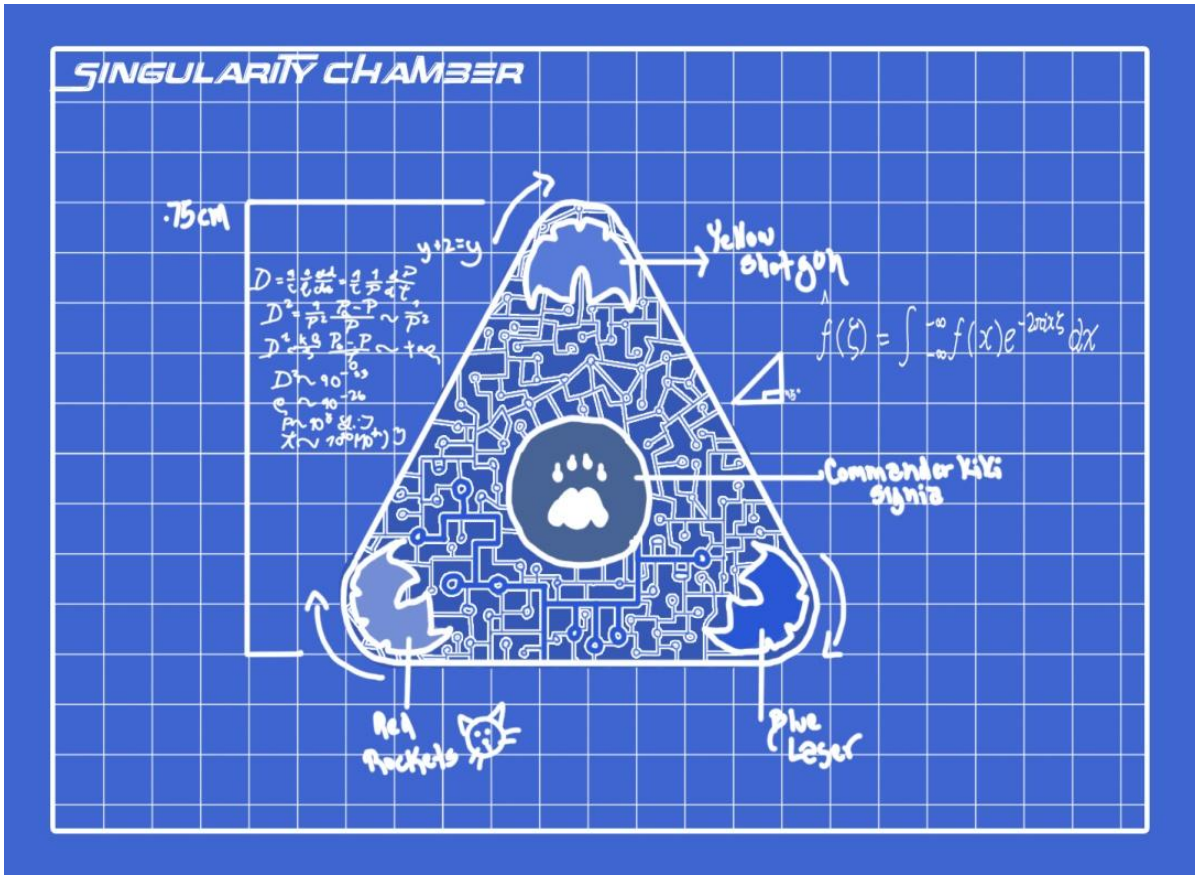
Instructions

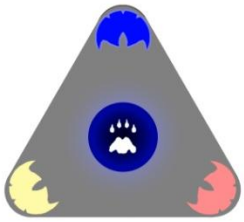
Props are anything the character carries like ***weapons***. List out each one and it's ***attributes*** such as damage, speed, range, etc. ***Concept*** of the prop by itself and another with the character holding it is also required.

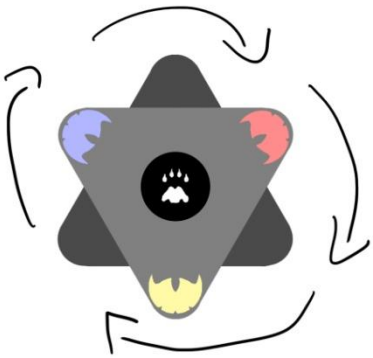
Thursday, December 08, 2011
11:06 AM

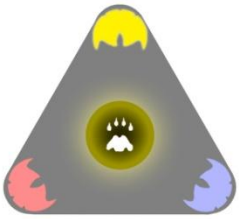
Prop(s)

Player Backpack

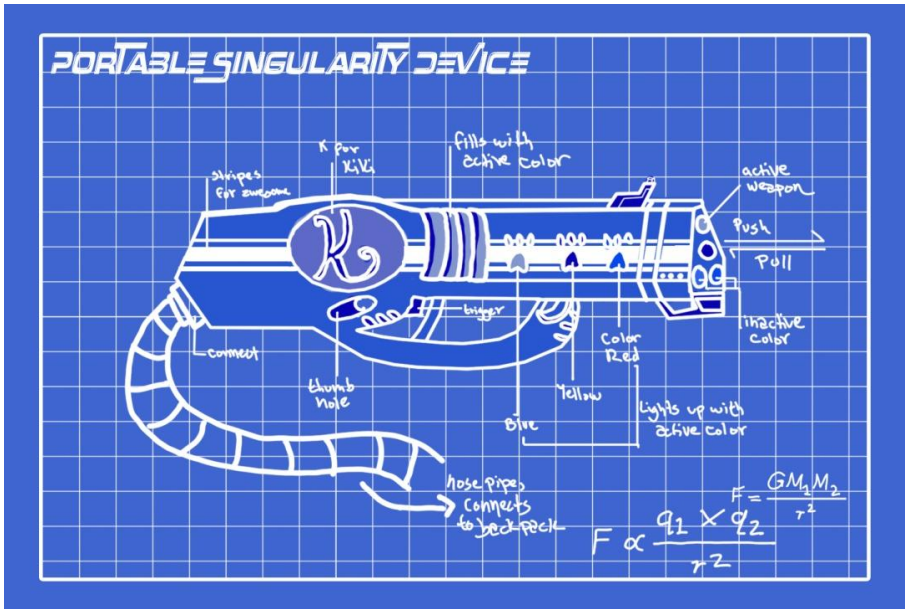


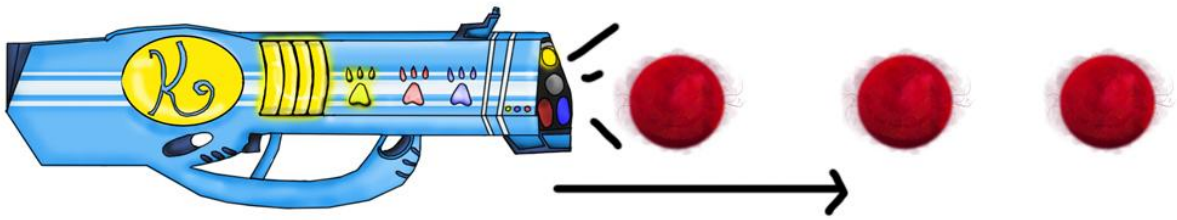






Player Gun

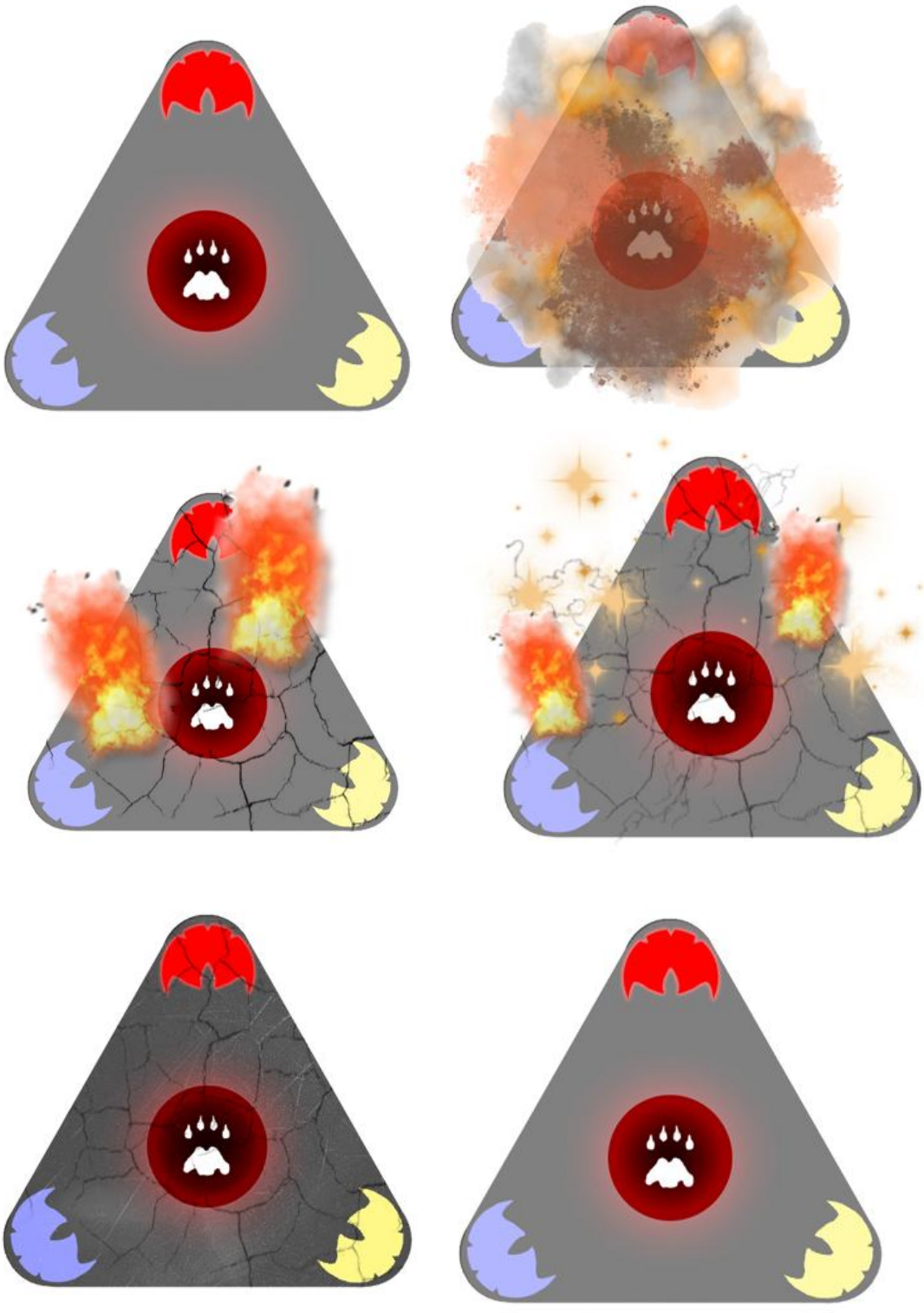




Backpack Damage Repair Protocol

If the PSD sustains any internal damage it will smoke and crack. Its nano-bot repairing system will immediately begin to repair the damage over time (about 5 seconds). During this time, the wielder will be unable to utilize the offensive capabilities of the weapon.

This type of damage would occur when the players takes in a bomb type enemy and fails to expel them before they detonate within the backpack. At that point, the player suffers a high amount of damage as well as having their backpack damaged. The repair protocols automatically activate, but the player will be unable to shoot, push or pull until the repairs are complete. This process will take about 5 seconds.



Effects

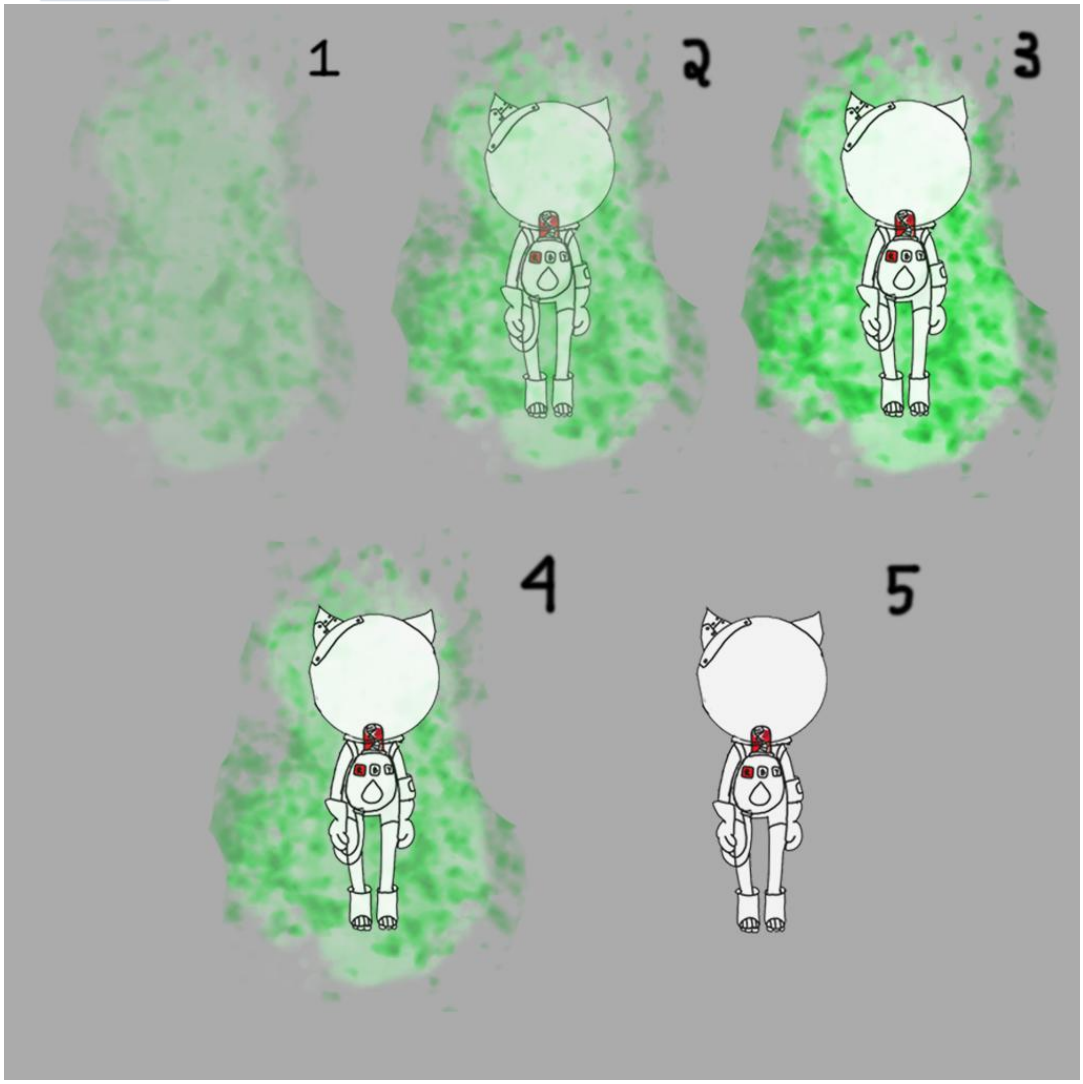
Instructions

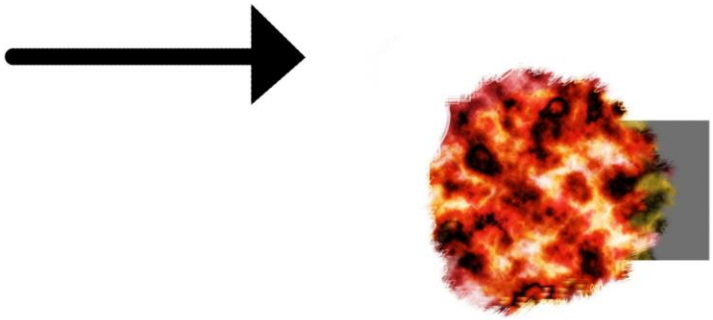
List out all the effects that are needed for the creation of the character. This includes things like particle effects. There should also be concepts for each of the effects.

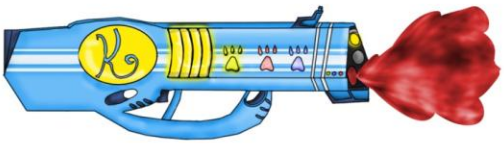
Thursday, December 08, 2011

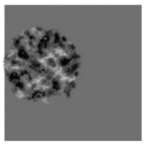
11:07 AM

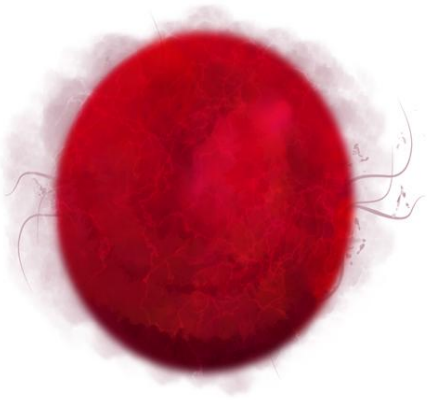
Effects

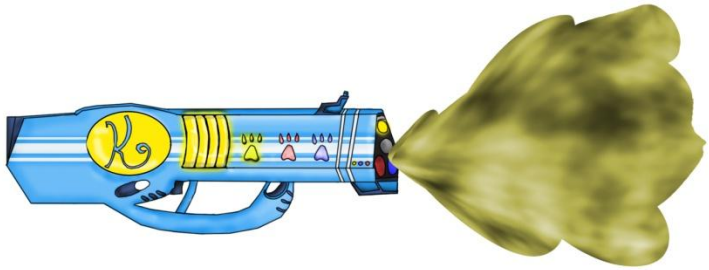


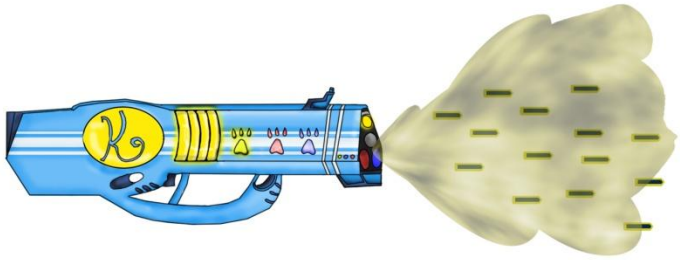


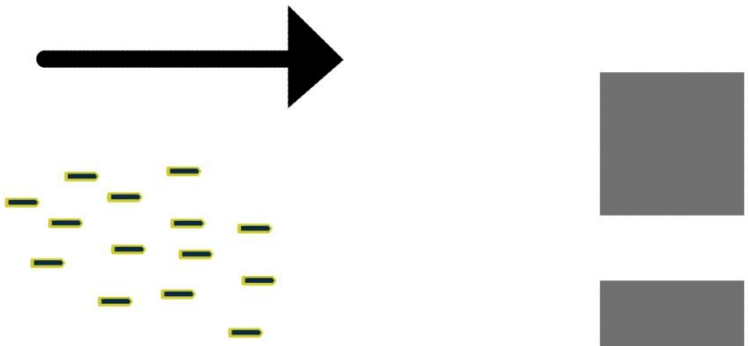


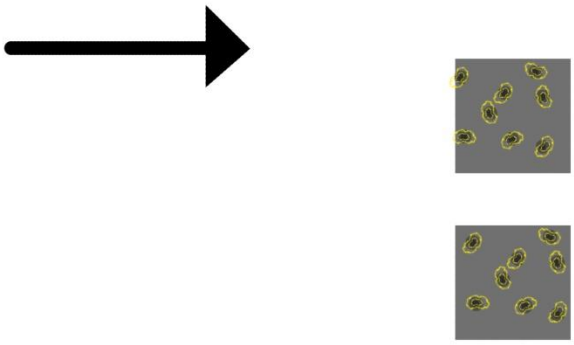


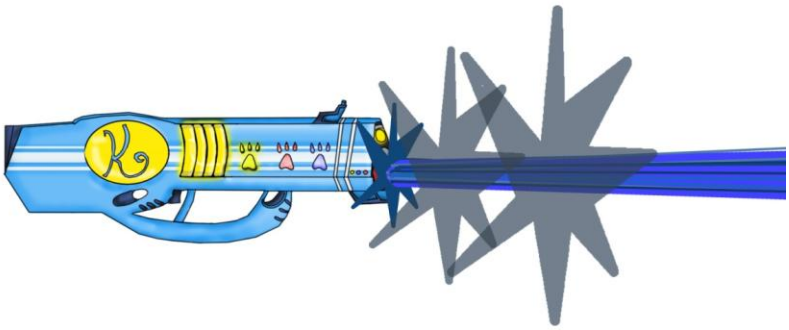


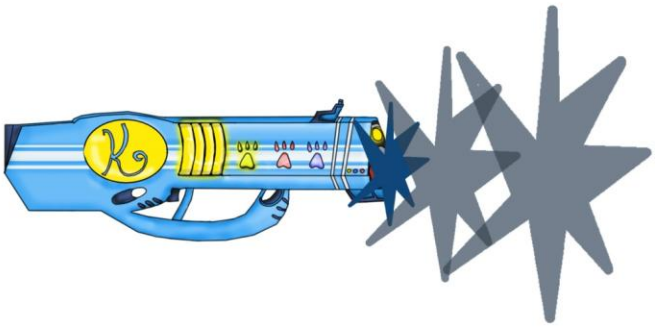


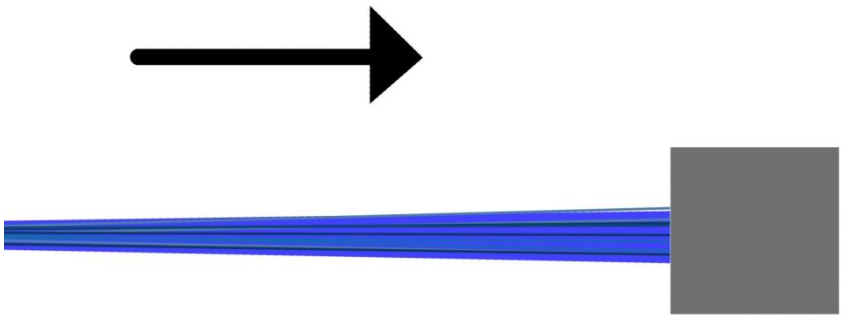


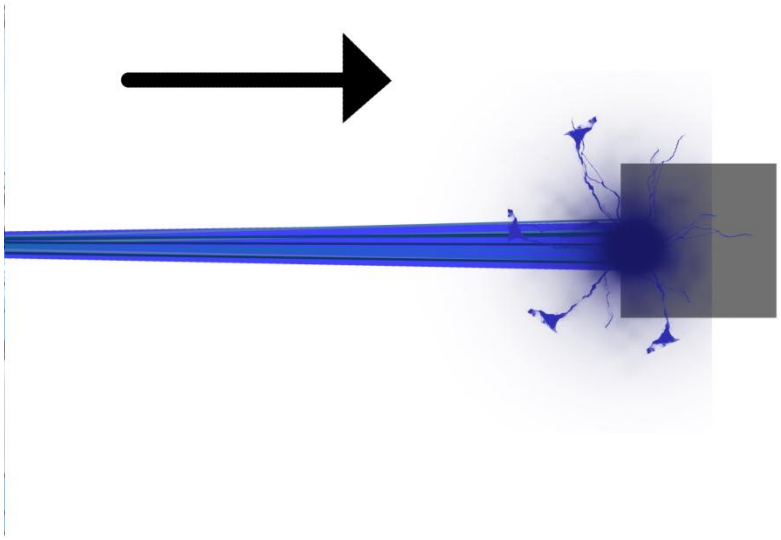


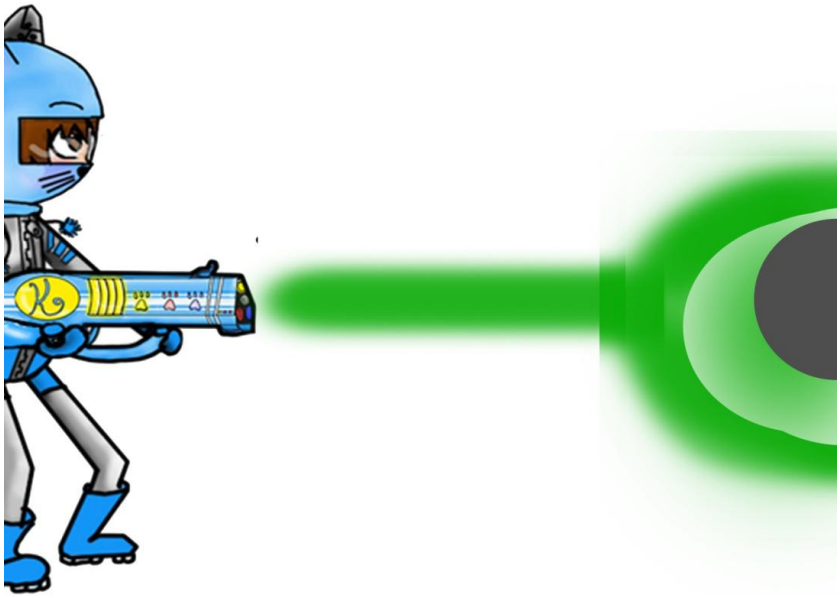


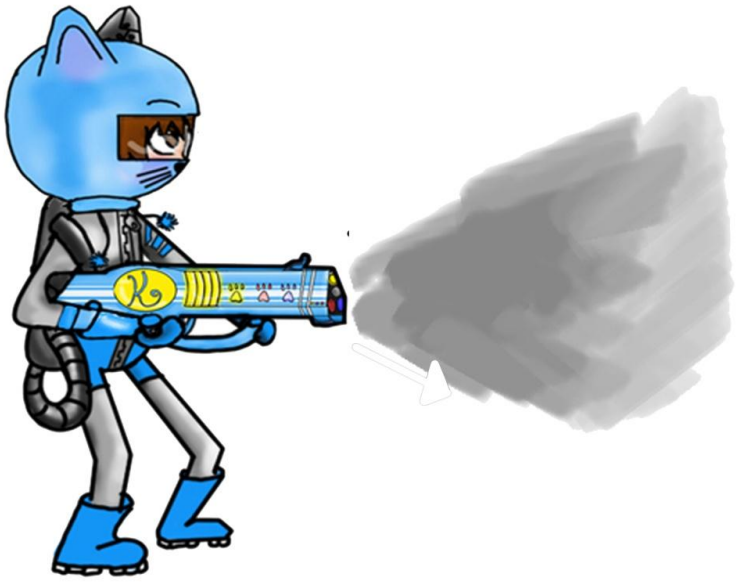












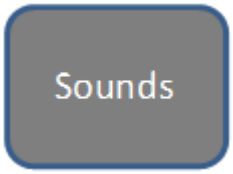
Sounds

Instructions

List out all the sounds that are needed for the creation of the character. This includes things like footsteps, attacks, commentary, etc...

Friday, December 16, 2011

11:11 AM



Player running/foot patters

Player is running on soft dirt or sand like substances on the planet's surface. This would sounds like running through dry sand.

Player jump

Similarly to the above the player kicks off of the ground with their feet. This could be simulated with a louder player footstep noise that is quickly muted to illustrate the player leaving contact with the surface.

Player landing

In the opposite fashion of jump have a quick and loud step onto the sand that is slowly muffled to illustrate the addition of weight as the player lands from their descent.

Player take damage from melee hit

A hollow metal tube hitting a pliable softer surface to illustrate the metal robot's arms bashing the player.

Player taking shock damage

A quick buzzing sounds of discharging electricity.

Player take damage from ranged hit

A sci fi like energy discharge or pop. The enemy projectiles are glowing spheres of energy that burst upon impact.

Pull ray

A sustained and shimmering or wobbling energy sound. A kind of warping tone to show that this effect is futuristic and strange. The pull alters the gravity around the target, hence the warping sounds.

Consumes and enemy

When an enemy is taken into the guns chamber. A cartoonish and rubbers stretch and pop sound.

Enemy swallowed

When an enemy in the waiting chamber is destroyed, giving the player a power up. There is a steam his and an muted energy pulse that starts off loud and quickly quiets. This should give the impression of compressions taking place within the back pack.

Power up Gained

A bubbling and increasing tone. This sound will be further masked by a hiss of compressed steam.

Ammo Change/Refill

Liquid filling a small container. This effect should be modified to sound electronic.

Backpack Shifting

When a new power up is gained the backpack rotates to the correct position. This would sound like a robotic limb with a muffled gear grind noise.

Push

A quick and muffled pulse sound. The push is a quick burst of force.

Plasma ball source

An amplifying and growing discharge of energy. This is the sound the plasma ball will make as it is building and leaving the gun.

Plasma ball travel

A buzzing and light tone. This will play as the ball is traveling to its target.

Plasma ball explosion

A fuzzy and electric sounding pop. The plasma ball explodes and deals area damage.

Blue laser beam

A light and soft vibrato sound to show the pulsing energy of the sustained laser.

Blue laser beam contact

Fizzling and burning noise to show how the energy gives off a lot of heat to deal damage.

Shockgun blast

Typical 12 gauge sound off. Loud quick blast of discharged energy.

Shockgun contact

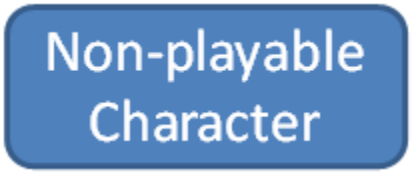
The sound of the individual buck shot particles hitting the target surface. A dull metallic thump.

Gaining health pick up

Rising bubbling tone from low to high frequency in a short time. Shows the building of energy and to simulate the recovery of a resource.

Non-player Character Overview

Thursday, December 08, 2011
10:56 AM



SMALL ENEMY

The small robot enemy has several variants which give the player her different ammunition types. The red small enemy is the version of the small robot enemy that will give the player the plasma ball ammunition type. The blue small enemy is the version of the small robot enemy that gives the player the laser beam ammunition type. The yellow small enemy is the version of the small robot enemy that gives the shotgun ammunition type. Finally, the black small enemy is a version of the small enemy robot which will try and blow itself up on the player. The small enemy travels in groups of five as they roam the planet in search of the player. The small enemy will try and attack the player with a prod when they are within range of the player. The player must pull them in and consume them in order to gain the different ammunition types. These enemies are easy to destroy will all ammo types, however, if the player is feeling overwhelmed then it is recommended to use the shotgun ammunition type.

MEDIUM ENEMY

The medium enemy is the medium ranged type enemy in the game. The enemy will try and kite the player into being attacked by its plasma bolts and EMP bolt. It will fire the EMP bolt every 8th shot in a volley of 5 bolts. The medium enemy is 3 times as strong as the small enemy and can mess up the player's game with its barrage of EMP bolts. The Player is recommended to use the laser beam ammunition type because it will take down this enemy n a quick fashion, However, it can be damaged by all ammo types.

LARGE ENEMY

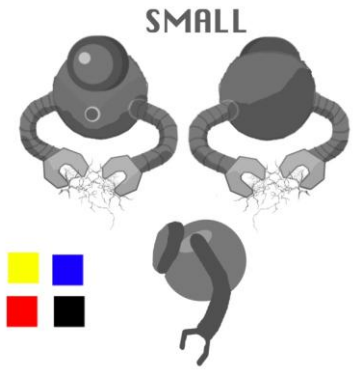
The large enemy has 2 phases, the first of which being a short range melee attack, and the second phase being an aggressive melee attack. The player must dodge the enemy's attacks and a tractor beam the enemy will shoot towards the player. In the first phase, the player will need to take out the Large enemy's shield, which will be color coded, by shooting the appropriate ammo type at the enemy. Then the second phase the player will be forced to dodge the oncoming attacks while shooting it with any ammo type to finish the job.

BOSS ENEMY

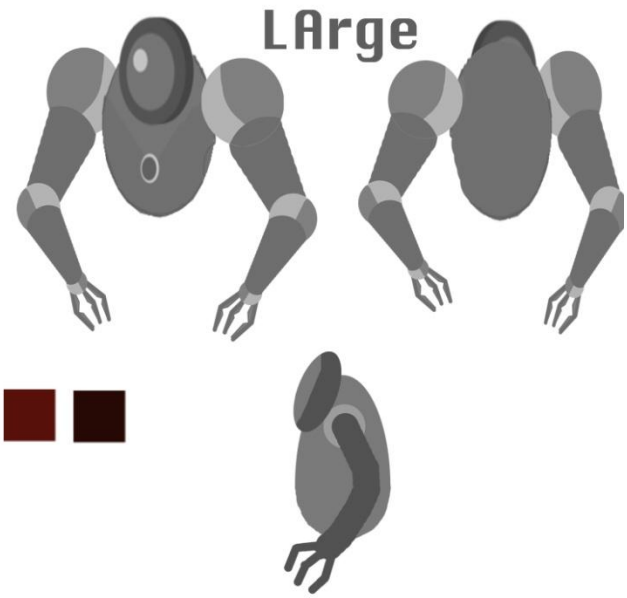
The boss enemy has 3 phases: one that resembles the medium enemy except it chases you, the second being a faster and tougher version of the first phase, and finally using the large enemy's tactics with the tractor beam and hitting you with a melee attack. The first two phases incorporate a shield, and he will be spawning small enemies throughout each phase.

Concept

Thursday, December 08, 2011
10:57 AM



Medium Enemy Hover

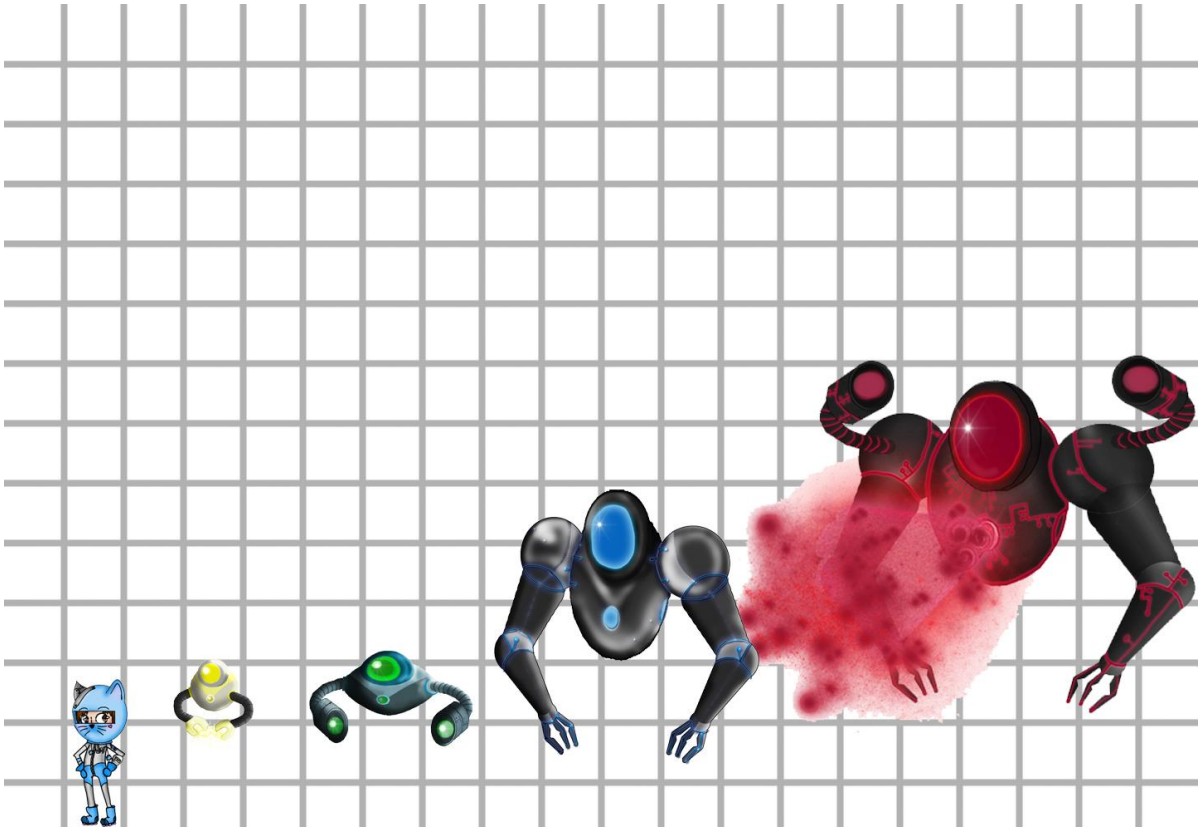


BOSS

Scale

Thursday, December 08, 2011
10:58 AM

Scale



Attributes

Thursday, December 08, 2011
10:58 AM

Small enemy

Basic Stats:

- Health – 5 Hits
- Width – 1 Unit
- Height – 1 Unit
- Speed – 2 Units/second

The small enemy travels in a group of 5 to try and swarm around the player to attack. They attack the player by approaching the player and within melee range of the player they will attack using a prod melee attack. They come in 4 variants which only differ in color except for the black color variant. All variants except for the black variant will give the player different ammunition types that vary in range and damage. The black variant will not give an ammunition type because this variant is a bomb and will try and blow itself up to attack the player. The player can defeat these types in various ways such as: using any and all ammunition types, pulling and consuming them to gain the ammunition type, pulling the enemy in and expelling the enemy with the push mechanic of the gun, or pulling the enemy into the acid pool hazard.

Color	Speed	Type	Range	Damage
Red	3 Units/second	Plasma Bolt	20 Units	10 Hits
Blue	10 Units/second	Laser	20 Units	3 Hits/Second
Yellow	15 Units/second	Shotgun	5 Units long 4 Units Wide	5 Hits

Medium enemy

Basic Stats:

- Health – 15 hits
- Width – 3 Units
- Height – 1 Unit
- Speed – 1 Unit/Second

The medium enemy is the medium enemy in the game. The medium enemy will stay 7 - 10 Units away from the player and try and pull the player to try and attack them from a distance with either a plasma bolt or EMP bolt. The medium enemy's primary fire is the Plasma bolt which will do 20 Units of damage to the player. Every 8th shot from the medium enemy will be an EMP bolt. The EMP bolt does not do damage to the player but it will consume all of the ammunition the player has. The player can also use the pull mechanic to pull herself toward the enemy to get in for a close range attack or use the push mechanic to stun the enemy and then attack. The medium enemy can be destroyed by any ammo type but it is recommended to use the laser ammunition type to be able to attack the medium enemy from afar if the player chooses not to use the pull mechanic method.

Type	Damage	Speed	Range
Plasma Bolt	10% max player health	3 Unit / Second	20 Units
EMP Bolt	No damage but Consumes all Ammunition	5 Units/second	3 x 3 Units around a radius of 8 Units around the enemy

Large enemy

Basic Stats:

- Health - 30 hits
- Width - 3 unit
- Height - 4 units
- Movement Speed - 0.5 units per sec

The large enemy is a very large and slow enemy that only does melee attacks. This enemy acts like a mini-boss in a way, in that it has 2 phases. The first phase consists of the large enemy being slow and cumbersome, and equips a shield. The large enemy will try to latch on to the player with this tractor beam in this phase. The player, when caught in the beam, has limited range of movement and has the current ammo start to drain at a slow but steady rate. The player can use his push while within a certain range from the large enemy to temporarily disorient the large enemy and release the tractor beam. The player can shoot while under the tractor beam's influence. When the player gets within a close range to the enemy, it will do a moderately damaging melee attack, pushing the player back and releasing the tractor beam. The player will want to find the nearest small enemy with the corresponding ammo type to the large enemy's shield color and use that ammo type until the shield is destroyed.

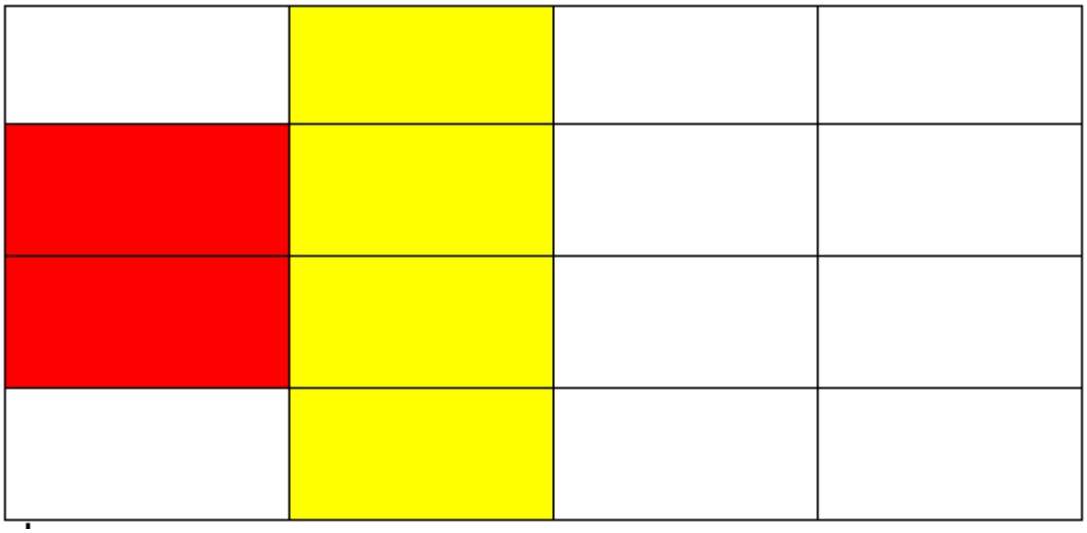
Once the shield is down, the second phase begins. The large enemy becomes less defensive and more offensive as he starts charging towards the player, doing a melee swipe while trying to trample over the player. This does slightly more damage than the attack in phase 1. The player will be able to damage the large enemy with any ammo type at this time, however the Plasma Ball from the Red small enemy is highly recommended. Both melee attacks done by the large enemy will also affect the small enemies if they are within collision range, as they are not the smartest processor in the factory and don't know to move out of the way.

Attacks:

Description:

- Once the player is within range, the large enemy will swipe horizontally at the player, dealing significant damage. At full health, the player can only take 4 of these hits before dying.
- The Player's best defense to combat the swipe is to jump above the attack. If the player can disable the tractor beam before the attack started, they should by using the push to break the tractor beam.

Swipe	Speed	Damage	Effect
Player	0.5 sec	25% max health	Flinch (Off Balance)
Small enemy	0.5 sec	30% max health	Flinch



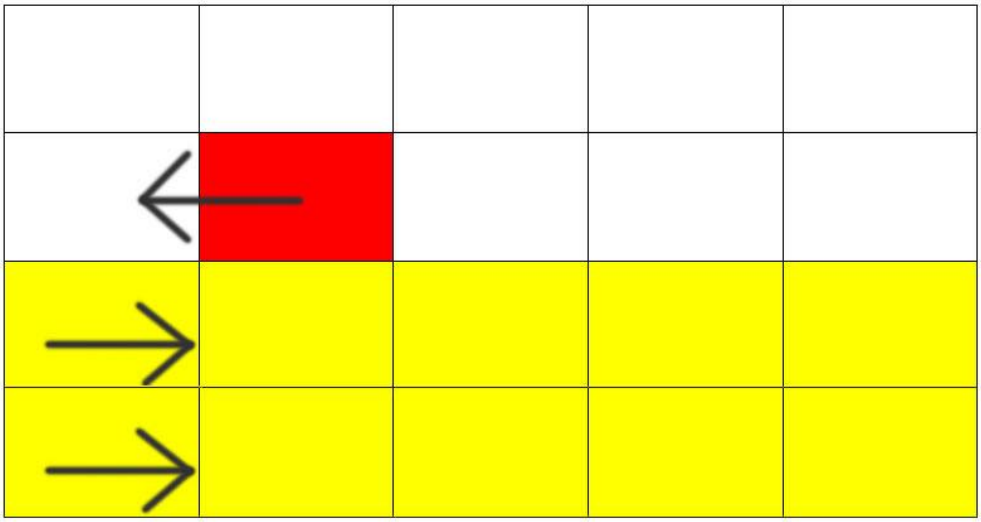
^ RED IS BOSS POSITION

^ YELLOW IS ATTACK ZONE

Description:

- The large enemy will lean forward and there will be a charging noise. After 2 seconds, the large enemy will charge towards the player until it either collides with the player or runs past it.
- The player's best defense to combat the charge is to jump out of the enemy's path.

Ground Slam	Speed	Range	Damage	Effect
Player	1.5 sec	4 units after player position	30% max health	Flinch (Off Balance)
Small enemy	1.5 sec	N/A	33% max health	Flinch



Boss

Basic Stats:

- Health - 90 hits
- Width - 6 unit
- Height - 6 units
- Movement Speed - 0.25 units per sec

The boss has 3 phases, emulating the medium and heavy enemies. The first phase consists of the Boss being slow and shooting the Plasma Bolt and EMP Bolt shots that the medium enemies have, and equips a shield. The Boss will slowly chase the player in this phase. The boss will periodically and dynamically have small enemies spawn at the nearest spawn depot. The player will want to find the nearest small enemy with the corresponding ammo type to the Boss's shield color and use that ammo type until the shield is destroyed.

Once the shield is down, the second phase begins. The shield will reappear, but in a flickering and visibly weakened state. The boss will start to move faster and shoot faster. Any and all environmental hazards will be active if they weren't and more small enemies will spawn. The player will want to find the

nearest small enemy with the corresponding ammo type to the Boss's shield color and use that ammo type until the shield is destroyed.

The Boss will enter the third phase when the shield is completely destroyed. The Boss will start using the tractor beam from the large enemy to make the player closer. The player, when caught in the beam, has limited range of movement and has the current ammo start to drain at a slow but steady rate. The player can use his push while within a certain range from the Boss to temporarily disorient the Boss and release the tractor beam. The player can shoot while under the tractor beam's influence. If the player comes in close proximity, the Boss will use a melee swipe attack that is slightly more powerful than the Large enemy's version. The boss can be damaged by all weapon types at this point. Once the player does enough damage to kill the boss, the waves will end and the boss will explode. A cutscene will happen and there will be a win screen.

Attacks:

Description:

- Once the player is within range, the Boss will swipe horizontally at the player, dealing significant damage. At full health, the player can only take 3 of these hits before dying.
- The Player's best defense to combat the swipe is to jump above the attack. If the player can disable the tractor beam before the attack started, they should by using the push to break the tractor beam.

Swipe	Speed	Damage	Effect
Player	0.5 sec	35% max health	Flinch (Off Balance)
Small enemy	0.5 sec	50% max health	Flinch

^ RED IS BOSS POSITION

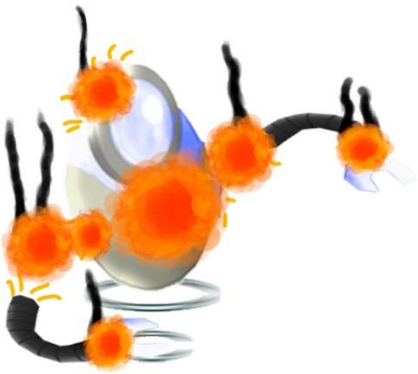
^ YELLOW IS ATTACK ZONE

Actions + Animations

Thursday, December 08, 2011
11:06 AM

Small enemy ATTACK:

Small enemy DEATH:



Player Shock

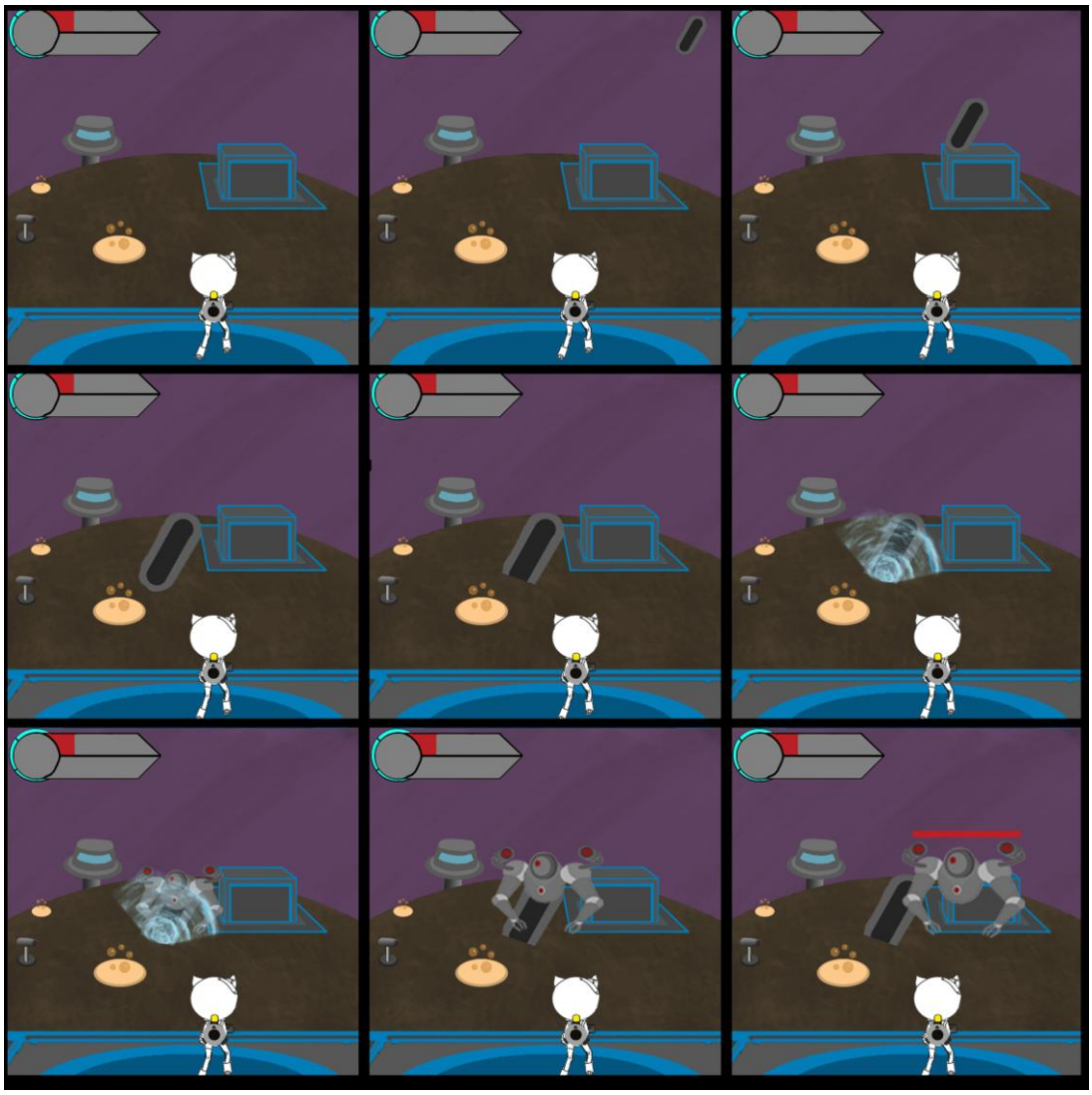
Medium Enemy Firing Plasma Bolt

Medium Enemy Explosion

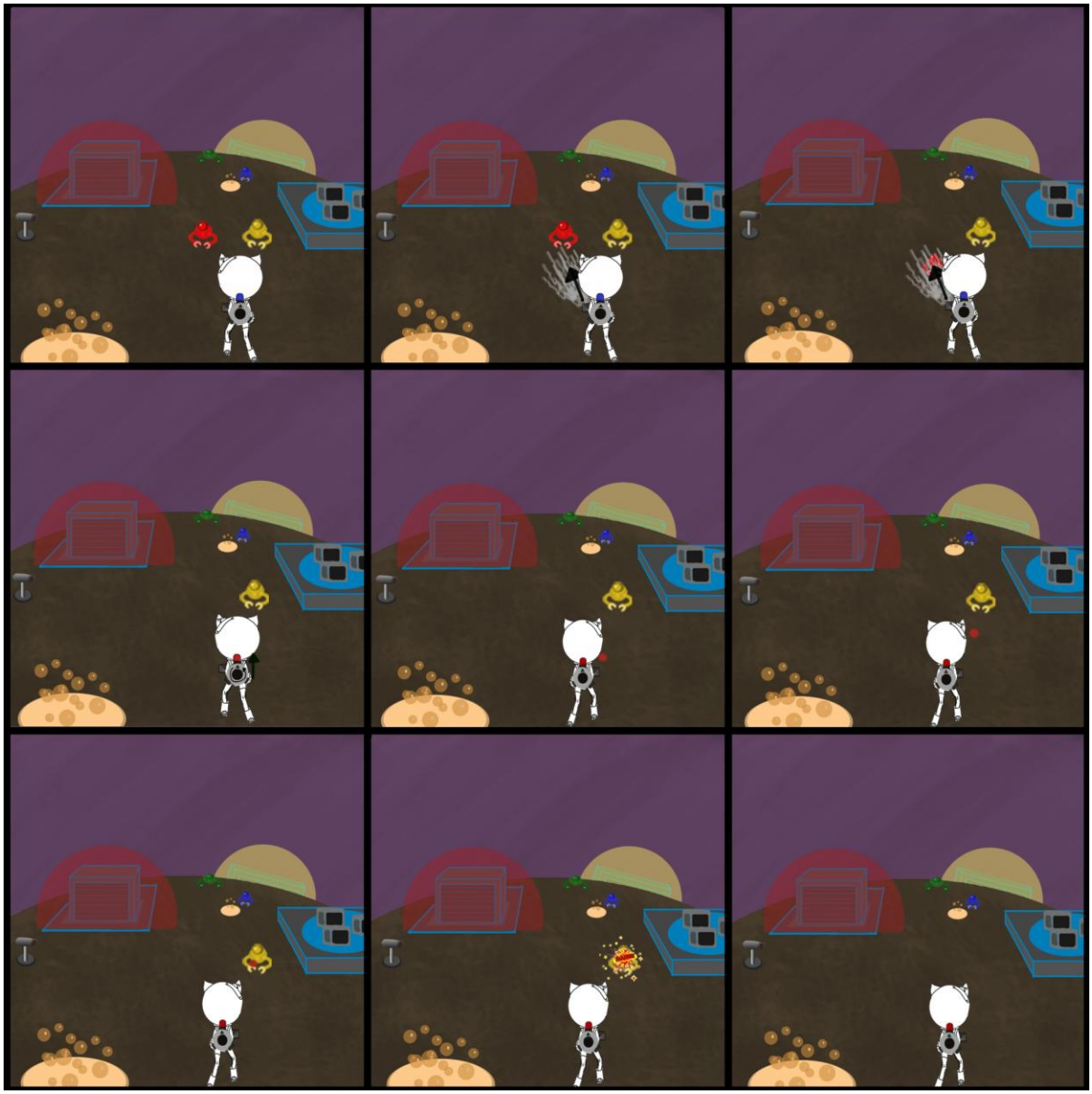
Large TRACTOR BEAM:

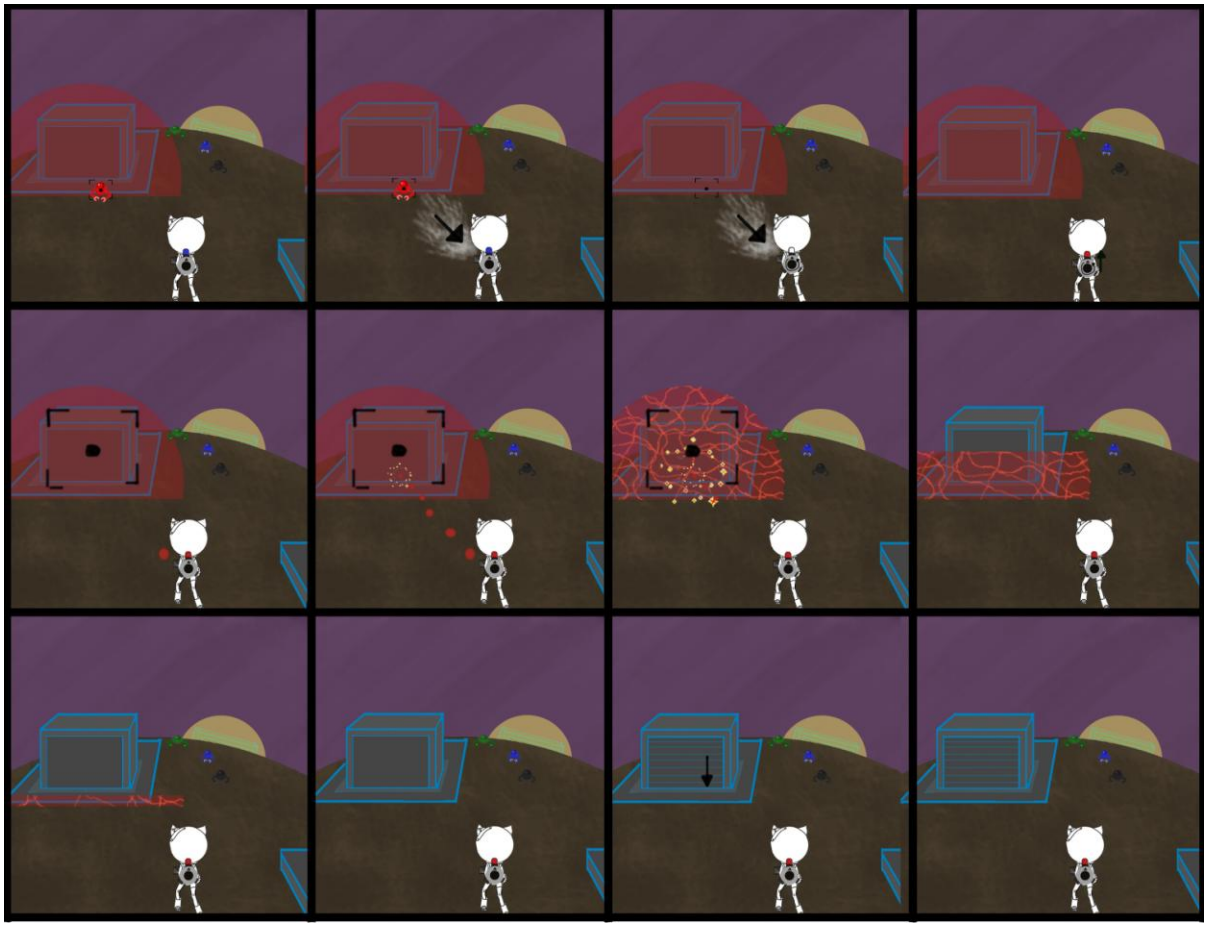
Large DEATH:

Boss Entrance

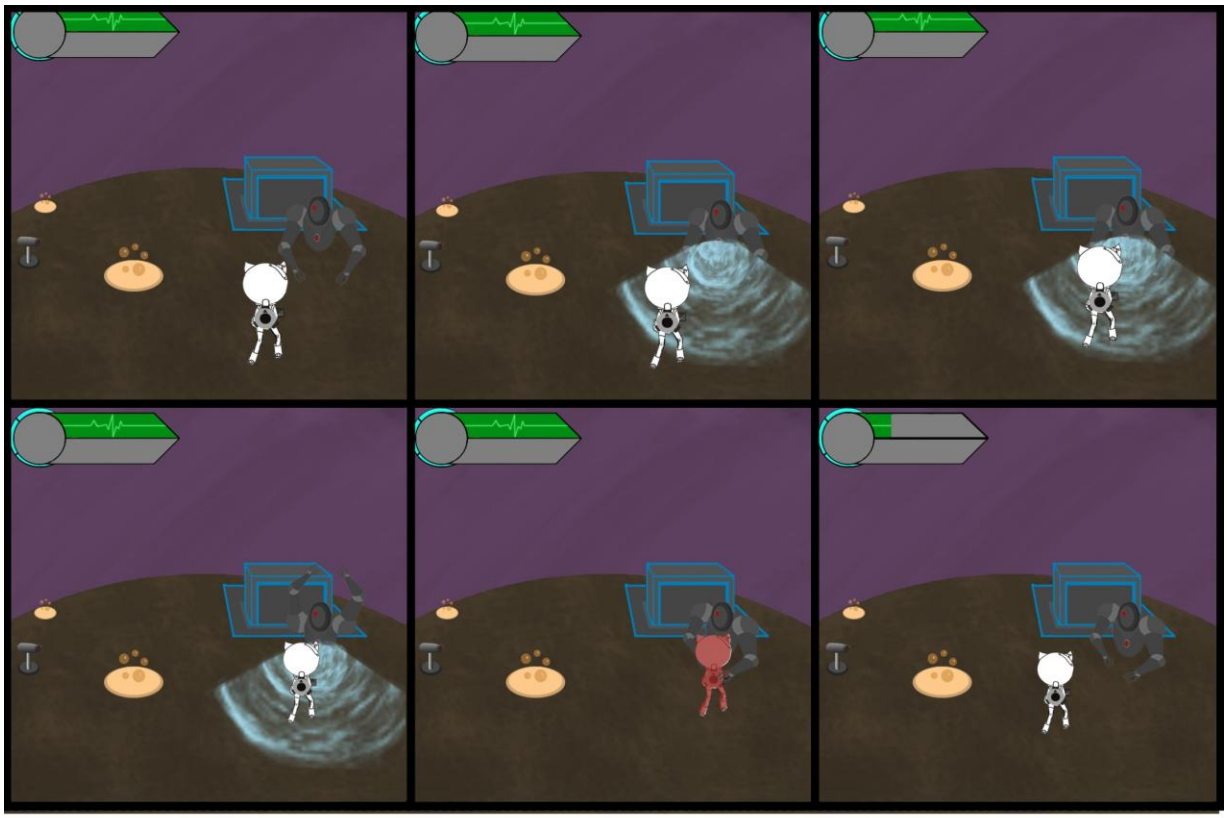


Fighting Smaller Enemies



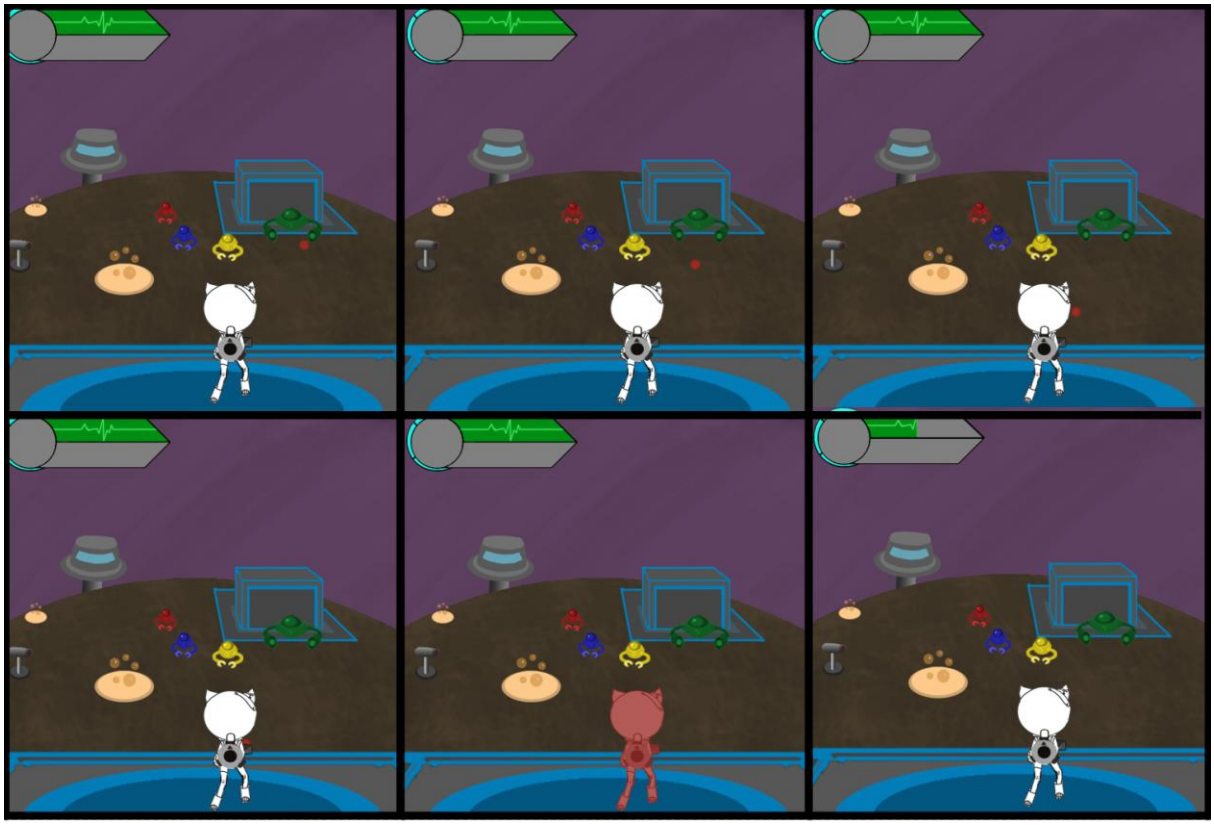


Large enemy tractor beam attack

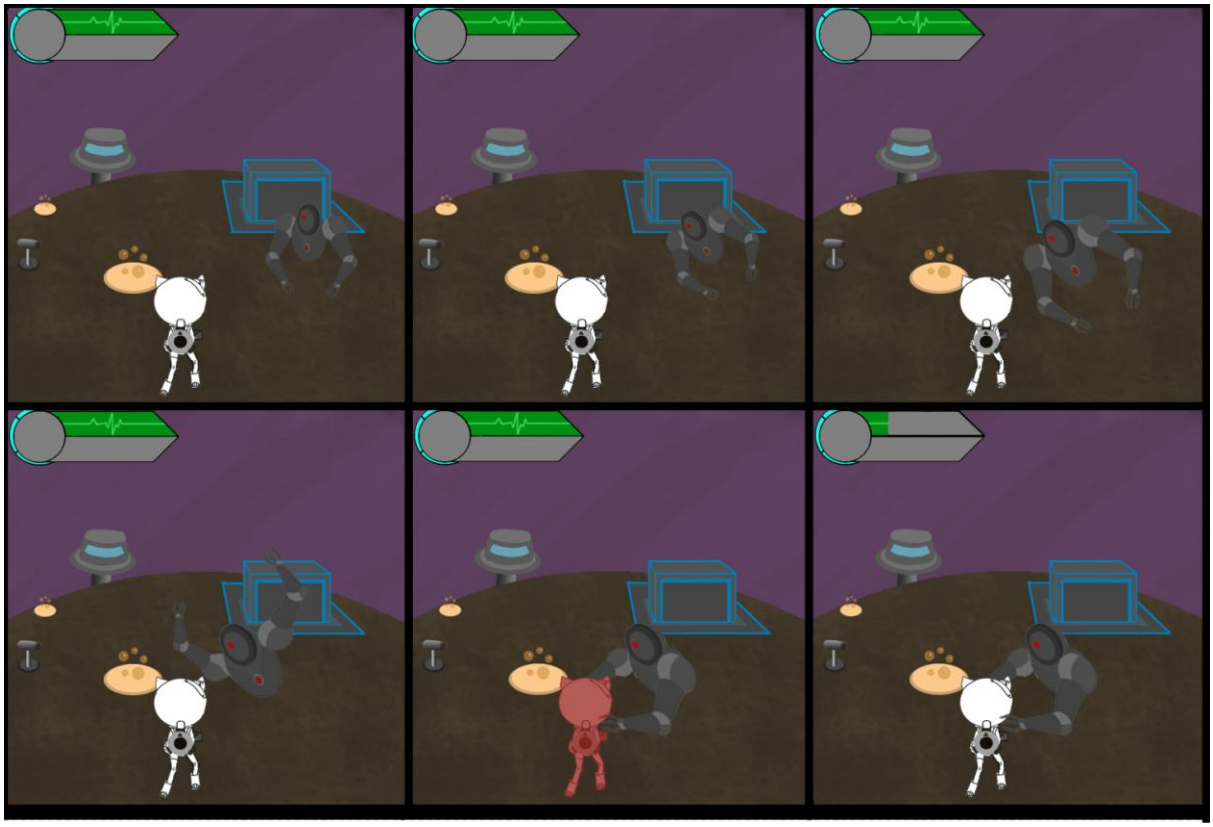


Player Taking Damage from the Bomb type Enemy

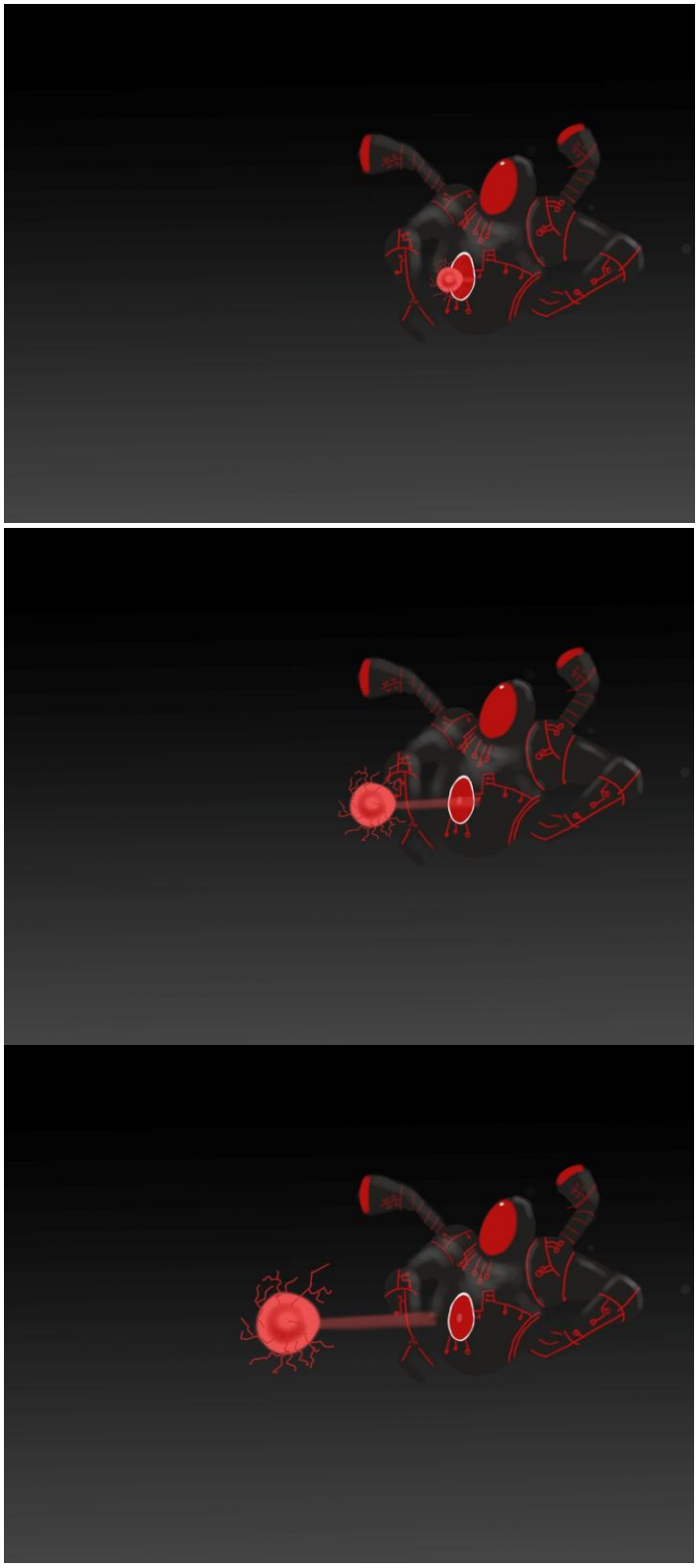
Medium Enemy Attacking



Large Enemy Charge Attack



Boss Shooting Plasma Shot



Boss Death

Prop(s)

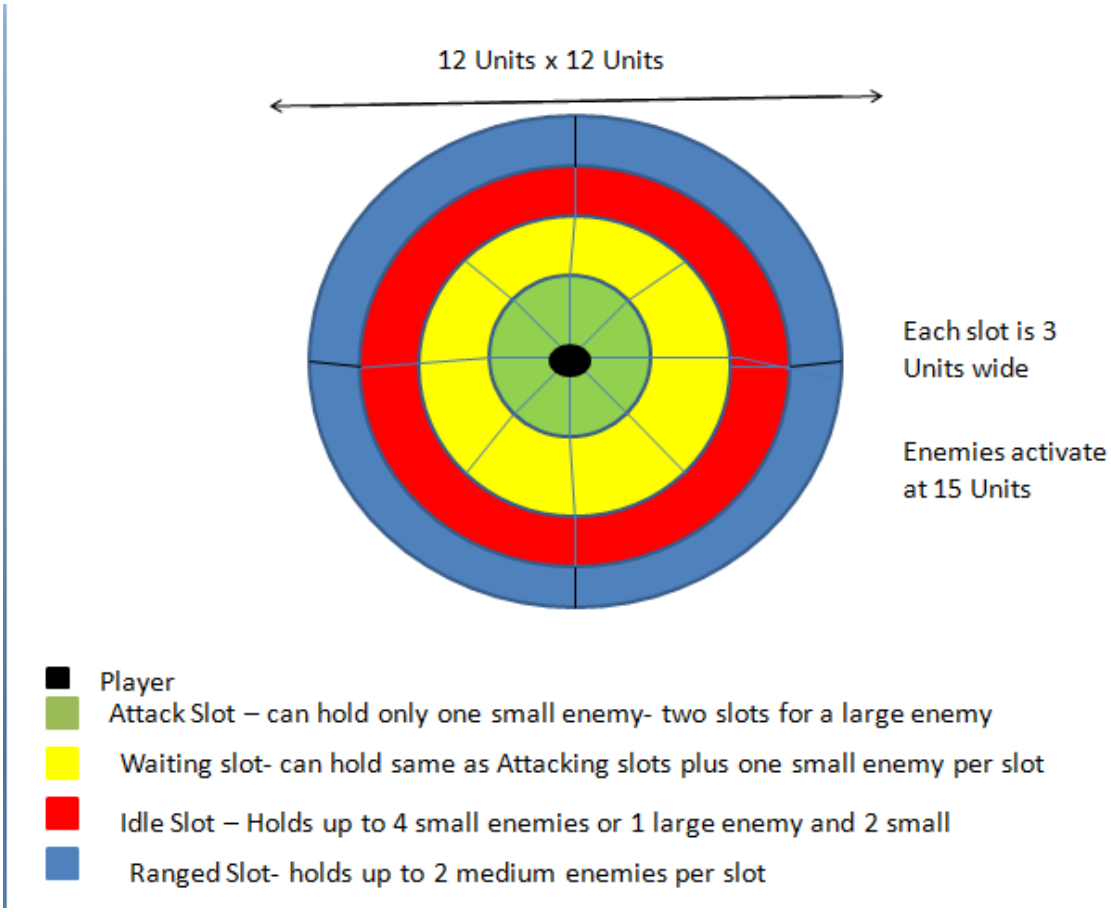
Thursday, December 08, 2011
11:06 AM

All Non-Playable characters either use their limbs for melee or fire projectiles.
No Non-Playable Character has props.

AI Slot System

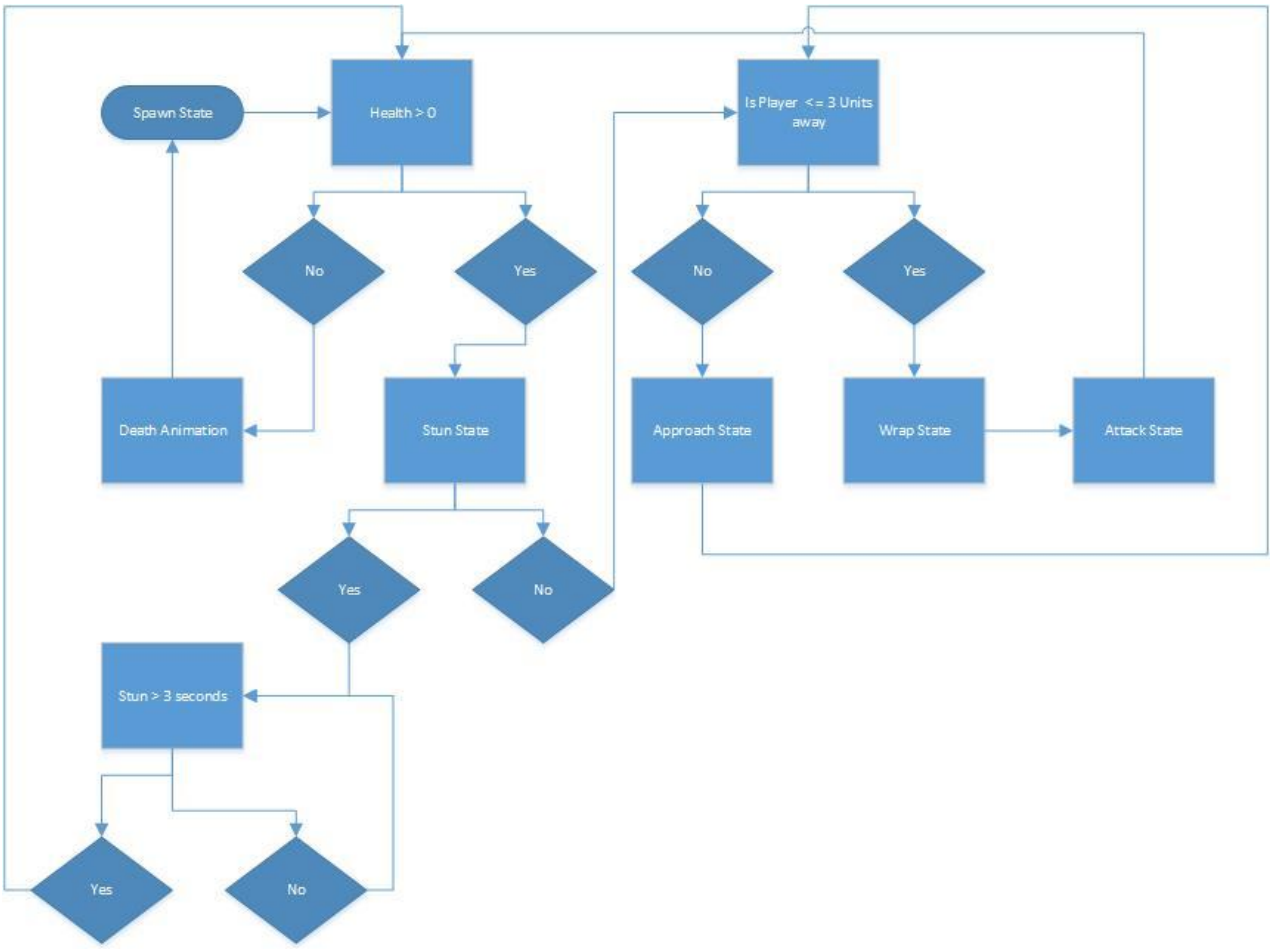
Tuesday, June 25, 2013
4:51 PM

The AI slot system is an implicit section of code designed to control and delegate enemies and their attacks towards the player. This system will act as a catalyst for a "controlled chaotic" feel that the game will have in terms of combat. All enemy types, except for the boss, will use this slot system. With the player in the center, the following diagram will show the basic thought process the system will follow to assign positions for enemies.

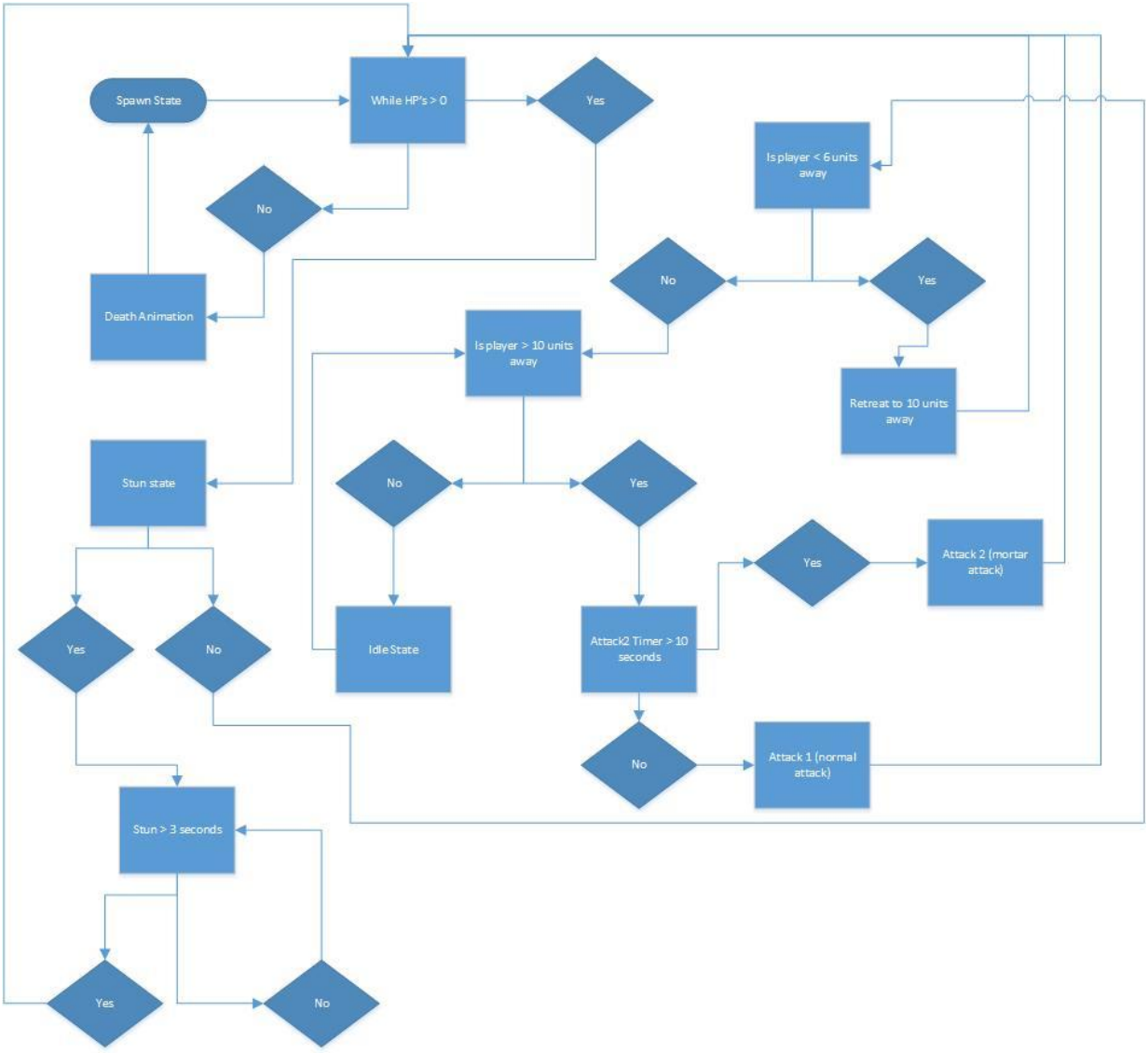


AI Flowcharts

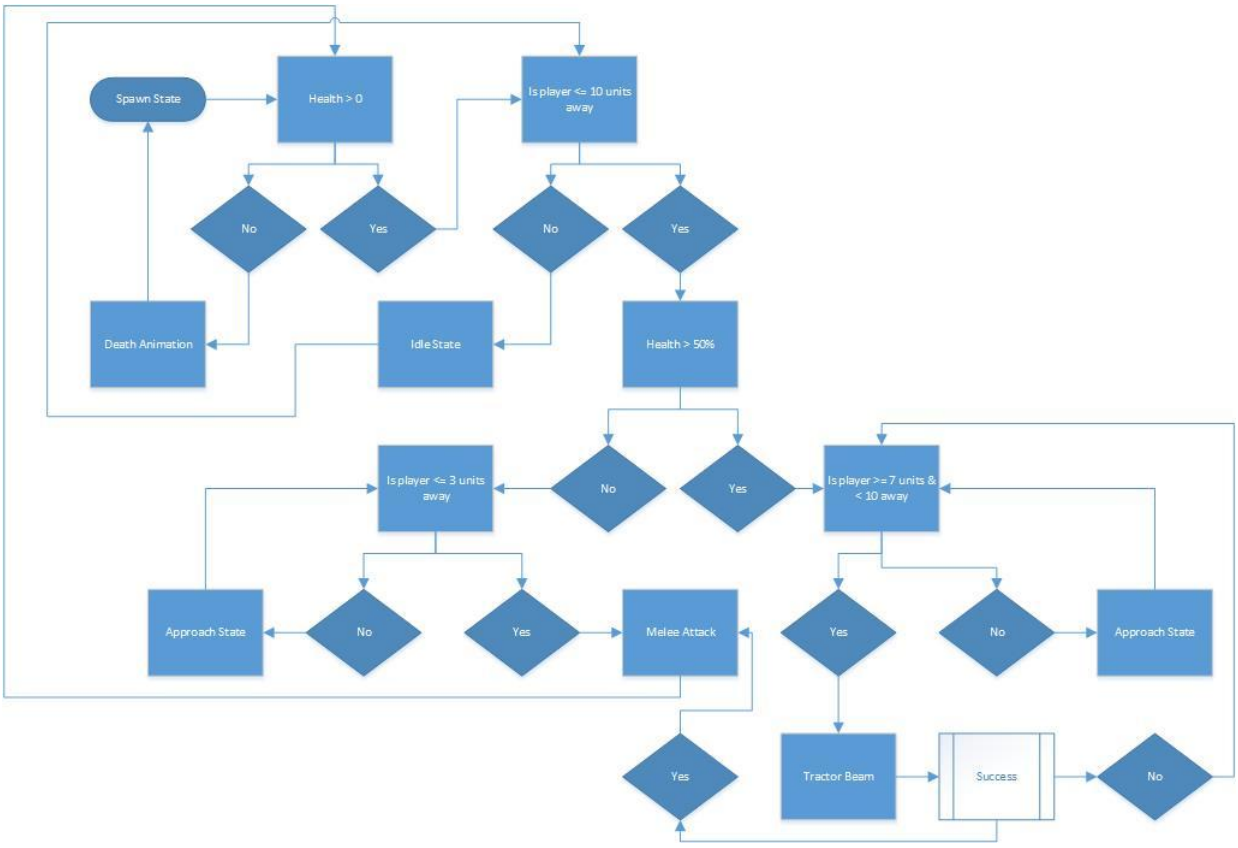
Thursday, December 08, 2011
11:07 AM



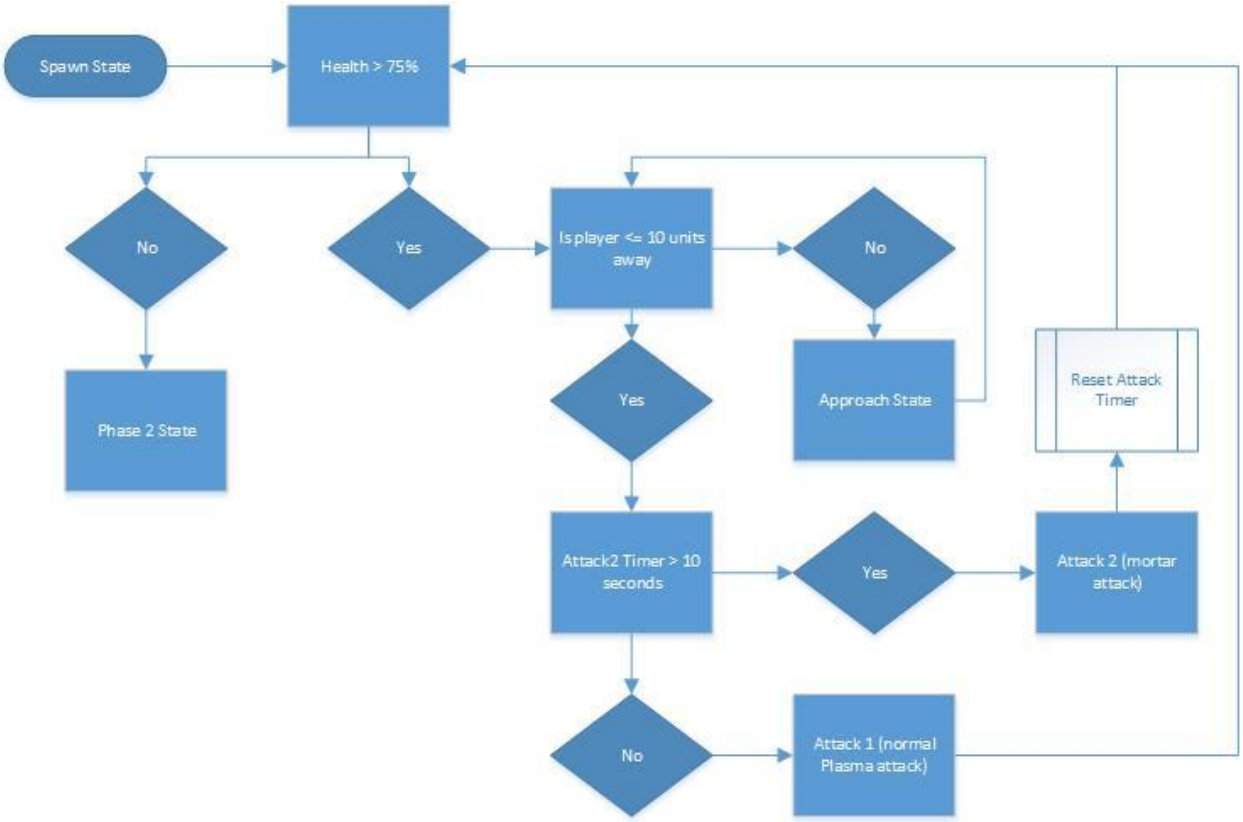
Small Enemy



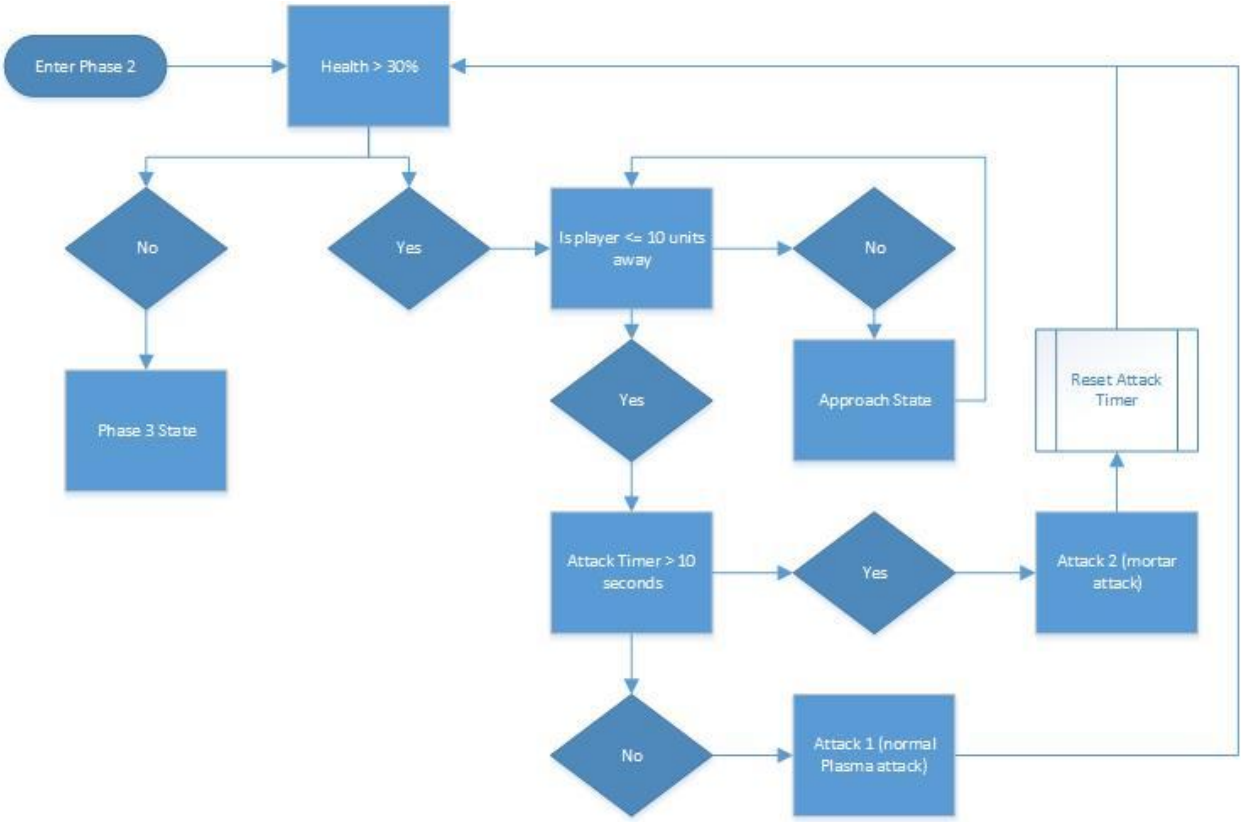
Medium Enemy


Spawn State
Health > 0
Is player <= 10 units away
No
Yes
No
Yes
Death Animation
Idle State
Health > 50%
Is player <= 3 units away
No
Yes
Is player >= 7 units & < 10 away
Approach State
No
Yes
Melee Attack
Yes
No
Approach State
Yes
Tractor Beam
Success
No

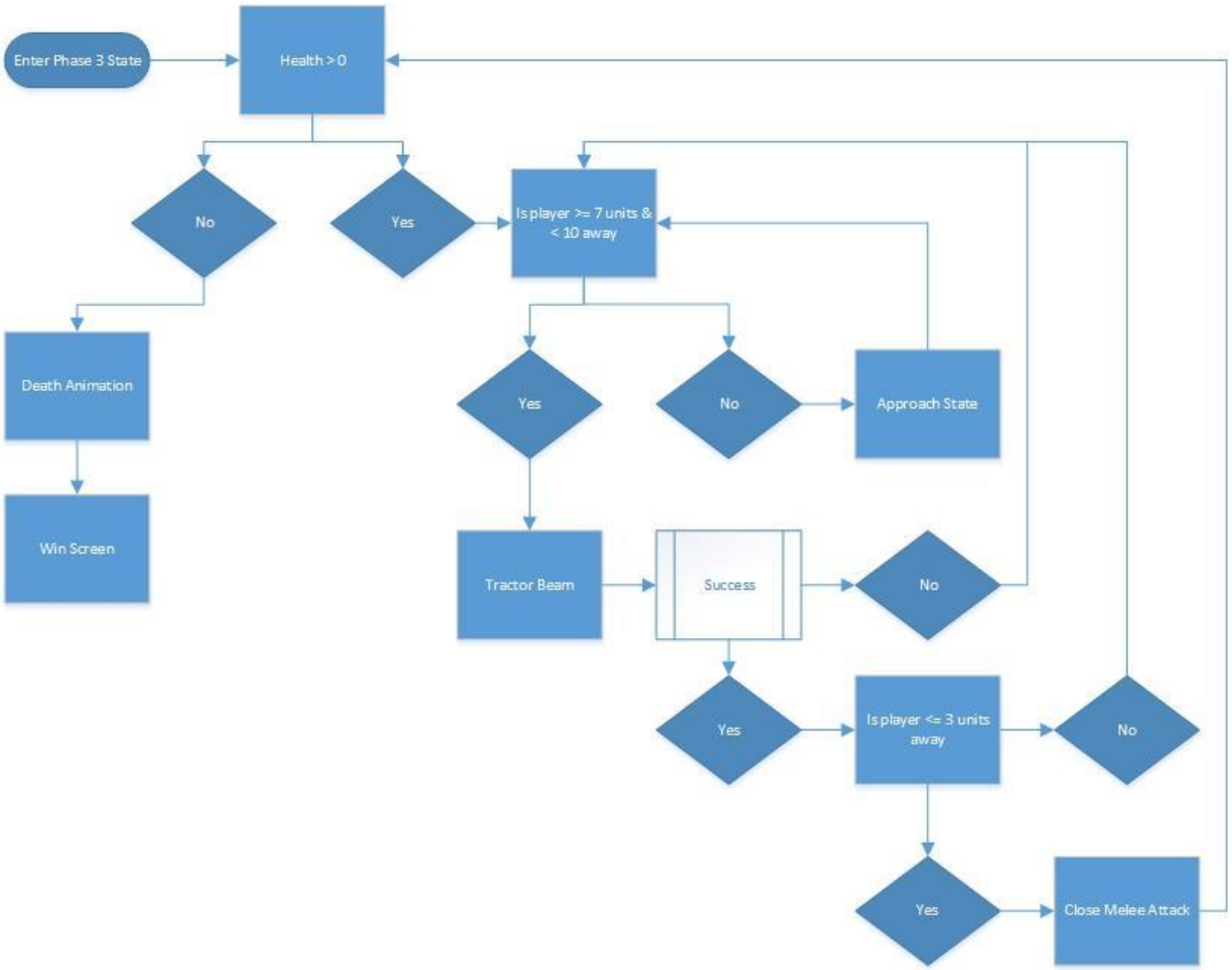
Large Enemy


Spawn State
Health > 75%
No
Yes
Is player <= 10 units away
No
Yes
Approach State
Reset Attack Timer
Phase 2 State
Attack2 Timer > 10 seconds
Yes
Attack 2 (mortar attack)
No
Attack 1 (normal Plasma attack)

Boss Phase 1


Enter Phase 2
Health > 30%
No
Yes
Is player <= 10 units away
No
Yes
Approach State
Reset Attack Timer
Phase 3 State
Attack Timer > 10 seconds
Yes
Attack 2 (mortar attack)
No
Attack 1 (normal Plasma attack)

Boss Phase 2

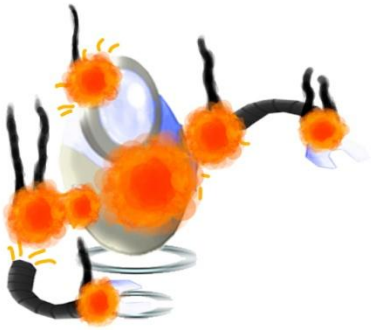

Enter Phase 3 State
Health > 0
No
Yes
Is player >= 7 units & < 10 away
Death Animation
Win Screen
Yes
No
Approach State
Tractor Beam
Success
No
Yes
Is player <= 3 units away
No
Yes
Close Melee Attack

Boss Phase 3

Effects

Thursday, December 08, 2011
11:07 AM

Effects



Small Enemy Shock Attack

Small Enemy Hover

small:

- Death explosion
- Floating effect
- Shock attack effect

medium:

- Death Explosion
- Floating effect
- Firing Plasma Bolt
- Firing EMP Bolt

large:

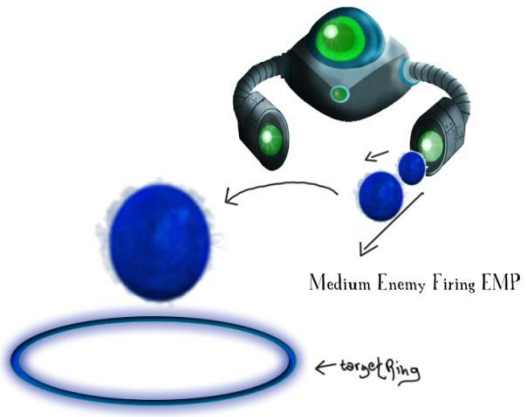
- Death Explosion
- Floating Effect
- Tractor Beam
- Shield

boss:

- Death Explosion
- Floating Effect
- Tractor Beam

Medium Enemy Firing Plasma Bolt

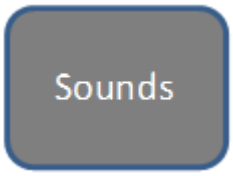
Medium Enemy Explosion



Medium Enemy Hover

Sounds

Friday, December 16, 2011
11:11 AM



Small enemy:

- Death explosion
- Floating effect sound
- Shock attack effect sound

Medium enemy:

- Death Explosion
- Floating effect sound
- Firing Plasma Bolt sound
- Charging EMP Bolt sound
- Firing EMP Bolt sound

Large enemy:

- Death Explosion
- Floating Effect sound
- Tractor Beam lock on sound
- Tractor Beam continuous sound
- Swipe sound
- Charge build up sound

Boss:

- Death Explosion (different)
- Floating Effect (different)

Everything else for Boss is duplicates

Environment Levels Overview

Instructions

Describe the environments the game takes place in and give a general concept of the art style of your game.

Thursday, December 08, 2011
11:10 AM

Moon

The game takes place on the moon over Kiki's home world. The surface of the moon is visually similar to that of Earth's moon, save that it is dotted with structures. It is significantly smaller than Earth's moon.

Landing Pads

Located on the poles of the planet, these act as spawn points for the player. They also receive supply crate drops in between each wave and the following wave. Landing pads are round and constructed of heavy metal. They are similar to real-world helipads.

Mining Depots

Spaced equally over the surface of the moon, mining depots are metal structures that emerge from the their foundations when activated, similar Terran supply depots in Star craft II. They spawn act as spawn points for all enemies in the game, except for the boss. When a mining depot is activated, it pops up from is lowered position. Once it is up, after a short pause, its shield activates. After another short pause, enemies begin pouring out of it.

Mining depots also serve as the objectives for each wave. At the beginning of each successive wave, a greater number of mining depots activates. To complete the wave, the player must deactivate all active mining depots. To deactivate each one, the player must obtain the correct ammunition type to damage it, depending on its shield color, and discharge her weapon into its shield until the shield goes down. The mining depot will then descend back into the ground in its inactive state.

Communications Towers

Spaced evenly over the surface of the moon, communications towers are tall enough to see from further away than other buildings. They are constructed of metal trusses, similar to mini Eifel Towers, and are topped by a large satellite dish that scans slowly back and for the across space. The purpose of communications towers is to create a more interesting skyline on the moon and to provide simple

physical obstacles for the player and the enemies to negotiate. The player hears faint radio chatter when she gets close to a tower.

Light Poles
Taller than other buildings, but shorter than communications towers, light poles provide additional sources of light in the scene beyond what comes in from space. They also add to the skyline and provide physical obstacles for the player and enemies.

Waste Pools
A byproduct of the enemies' mining operations, waste pools are highly corrosive pools of glowing orange liquid. They are scattered haphazardly over the surface of the moon. Should the player or an enemy come into contact with the liquid, she or it will steadily take damage.

Spinning Lasers
Planted near the mining depots by the enemies as an extra layer of defense, these permanent fixtures emerge from metal plates on the ground. Once up, they extrude solid beams of laser light, much like the light sabers in Star Wars. These rotate, damaging anything they touch. The beams are roughly equivalent to the player's height (about 2-2.5cm in length).

Crates
Supply crates are dropped in from orbit between waves to help Kiki recover her health. They land on the landing pads located on the poles of the moon. Constructed of heavy plastic, crates can be broken by any weapon or by using PUSH. Once shattered, they reveal health orbs.

Health Orbs
These are glowing, glittering green orbs of light that restore health to the player. They come in two sizes, the larger of which restores 30% of the player's health and the smaller of which restores 10%. Health orbs display a plus(+) sign in their centers.

Attributes

Environment Piece	Damage	Speed	Notes
Spinning Laser	Medium	Instant	Spinning Laser turrets rotate clockwise at a speed of one rotation per 2 seconds. A player takes damage when initial contact per rotation is made with the laser beam. Enemies are not harmed by these lasers.
Acid Pool	Very Small	Fast	A slow boiling pool of orange waste. Player will take small amounts of damage very quickly while standing within the bounds of the acid pool. Enemies are not harmed as they are

			able to float above the acid pool.
Explosive Barrel	High	Instant	These environment pieces are scattered around the moon's surface. Any form of damage dealt to them will cause them to explode and deal damage to all objects around them in a radius of 3cm. This could cause other surrounding barrels to explode themselves. Explosions will damage both the player and any enemies within the damage radius.

Game Progression

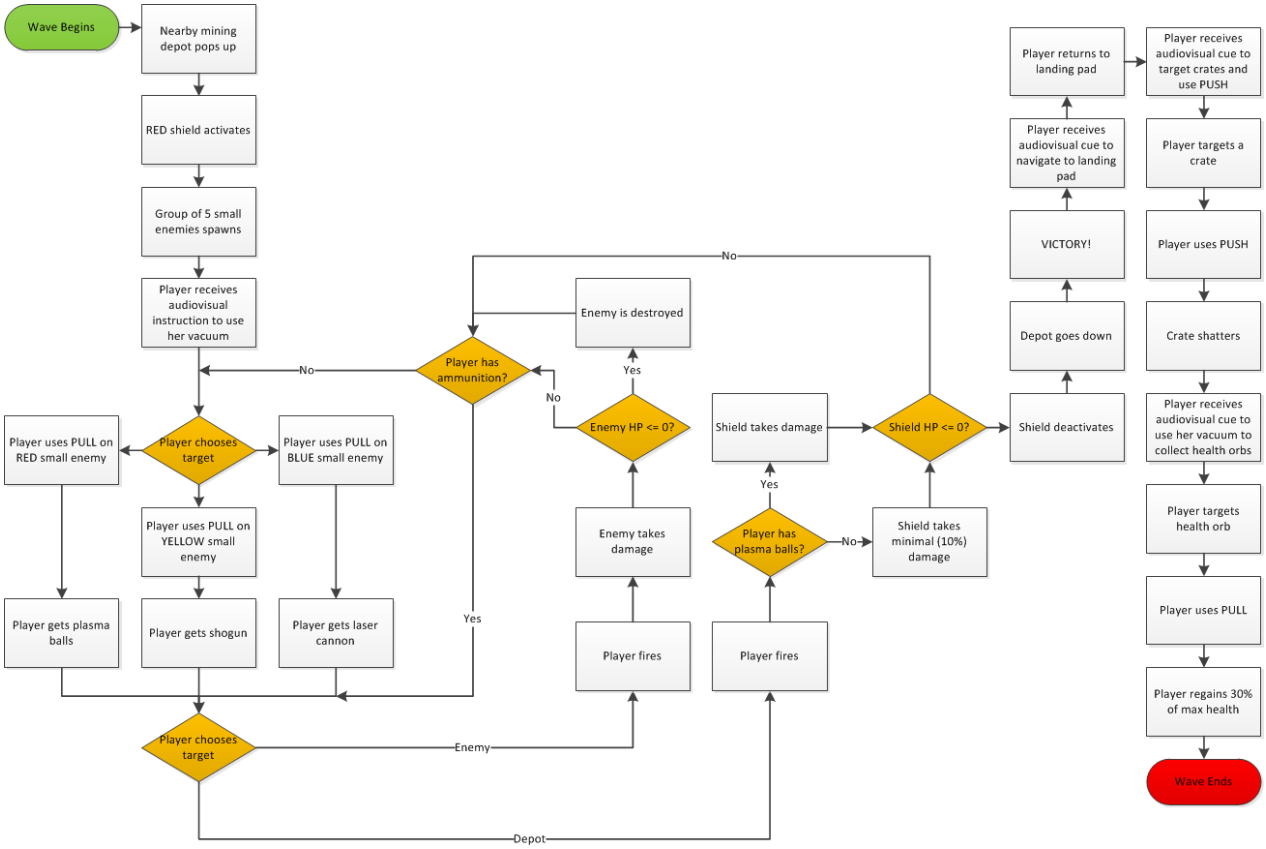
Instructions

Create a flowchart for how the player experiences the game progression and what makes the game progress from one section to the next. This should include a beginning and end section and as well win and lose scenarios.

Thursday, December 08, 2011
11:15 AM

General Notes:
On all waves, small enemies spawn until all objectives are met.

Wave 1
Win condition: Deactivate the active mining depot.
Loss condition: Lose all health.
Notes: There are no BLACK small enemies in this wave.

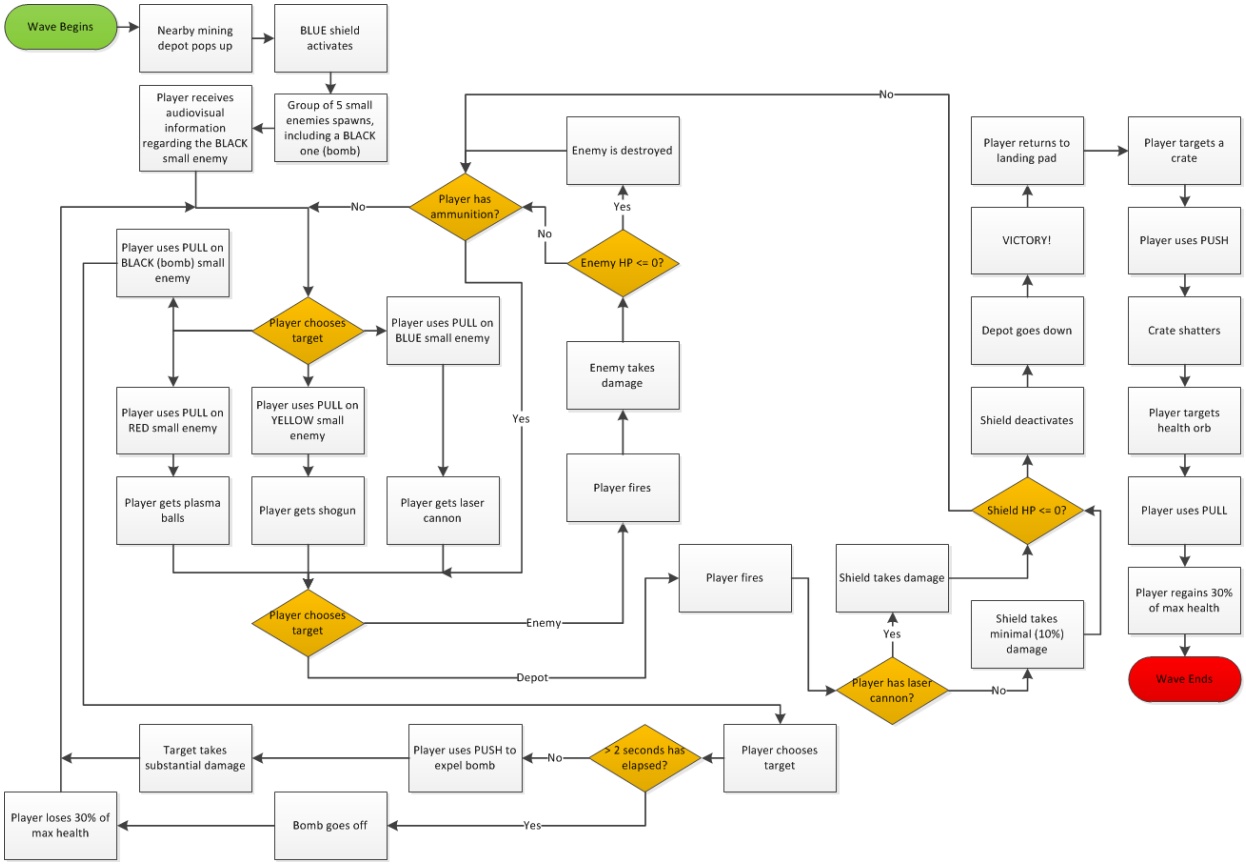

Wave Begins
Nearby mining depot pops up
RED shield activates
Group of 5 small enemies spawns
Player receives audiovisual instruction to use her vacuum
Player has ammunition?
No
Yes
Player chooses target
Player uses PULL on RED small enemy
Player uses PULL on YELLOW small enemy
Player uses PULL on BLUE small enemy
Player gets plasma balls
Player gets shogun
Player gets laser cannon
Player chooses target
Enemy
Depot
Enemy is destroyed
Enemy HP <= 0?
Enemy takes damage
Player fires
Shield takes damage
Shield HP <= 0?
Shield takes minimal (10%) damage
Player has plasma balls?
Player fires
Shield deactivates
Depot goes down
VICTORY!
Player returns to landing pad
Player receives audiovisual cue to navigate to landing pad
Player receives audiovisual cue to target crates and use PUSH
Player targets a crate
Player uses PUSH
Crate shatters
Player receives audiovisual cue to use her vacuum to collect health orbs
Player targets health orb
Player uses PULL
Player regains 30% of max health
Wave Ends

Wave 2

Win condition: Deactivate the active mining depot.

Loss condition: Lose all health.

Notes: This wave introduces the BLACK (bomb) small enemy.

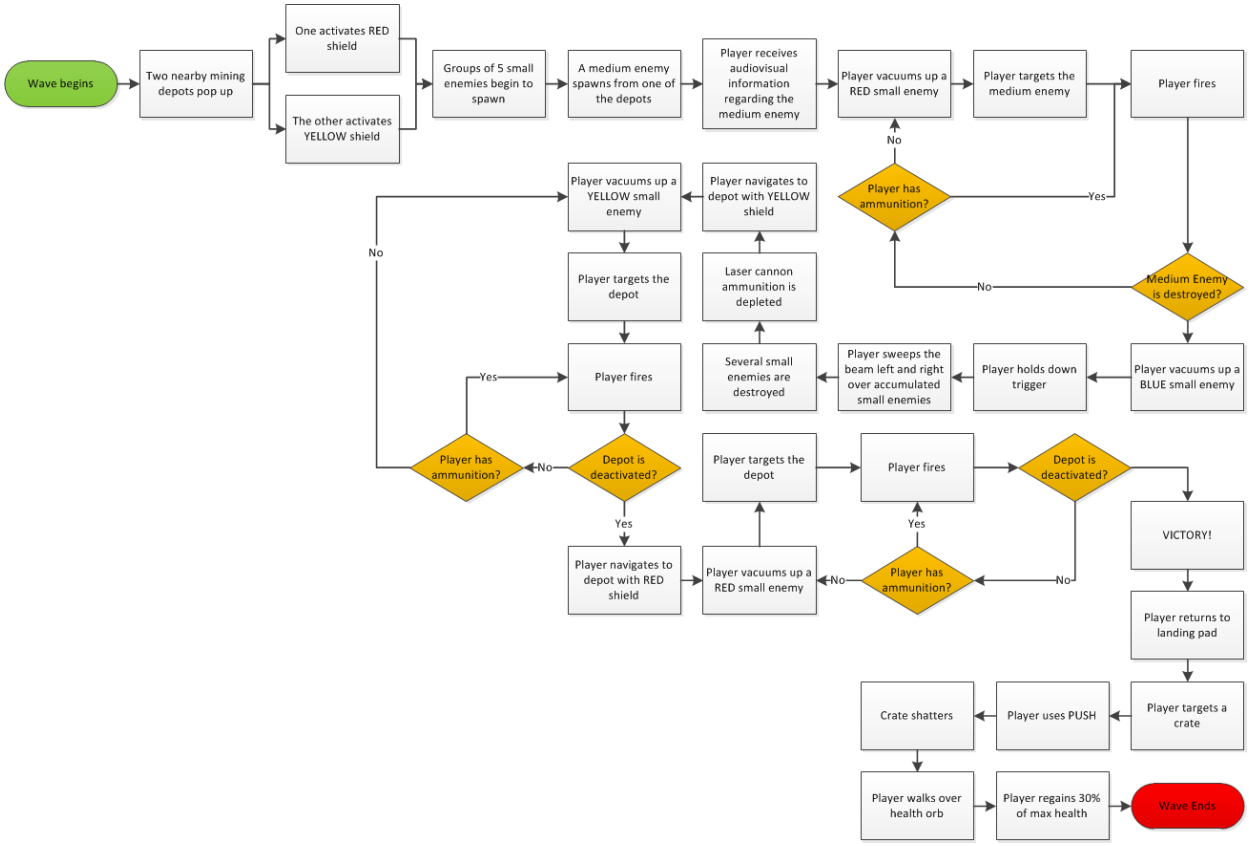


Wave 3

Win condition: Deactivate both active mining depots.

Loss condition: Lose all health.

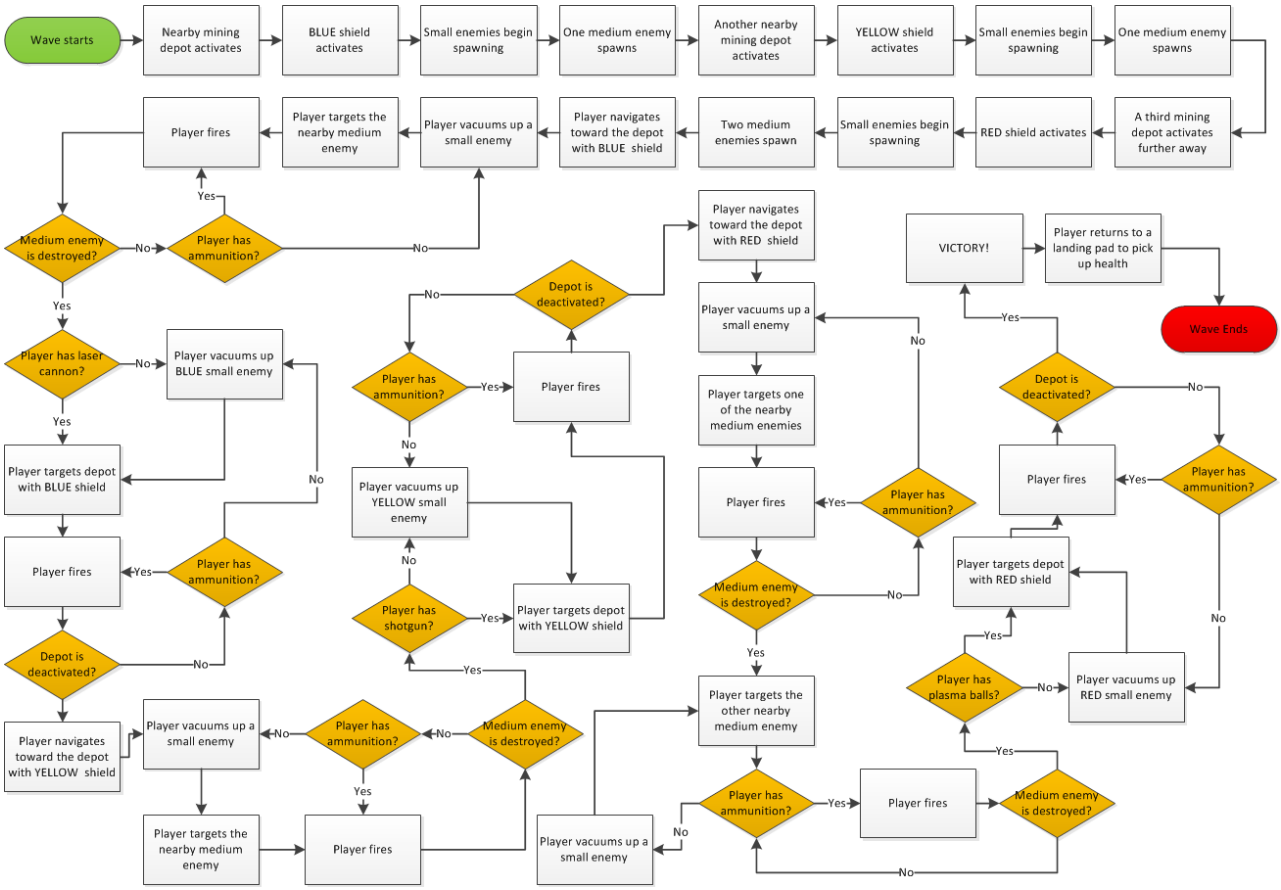
Notes: This wave introduces the medium enemy.



Wave 4

Win condition: Deactivate all three active mining depots.

Loss condition: Lose all health.

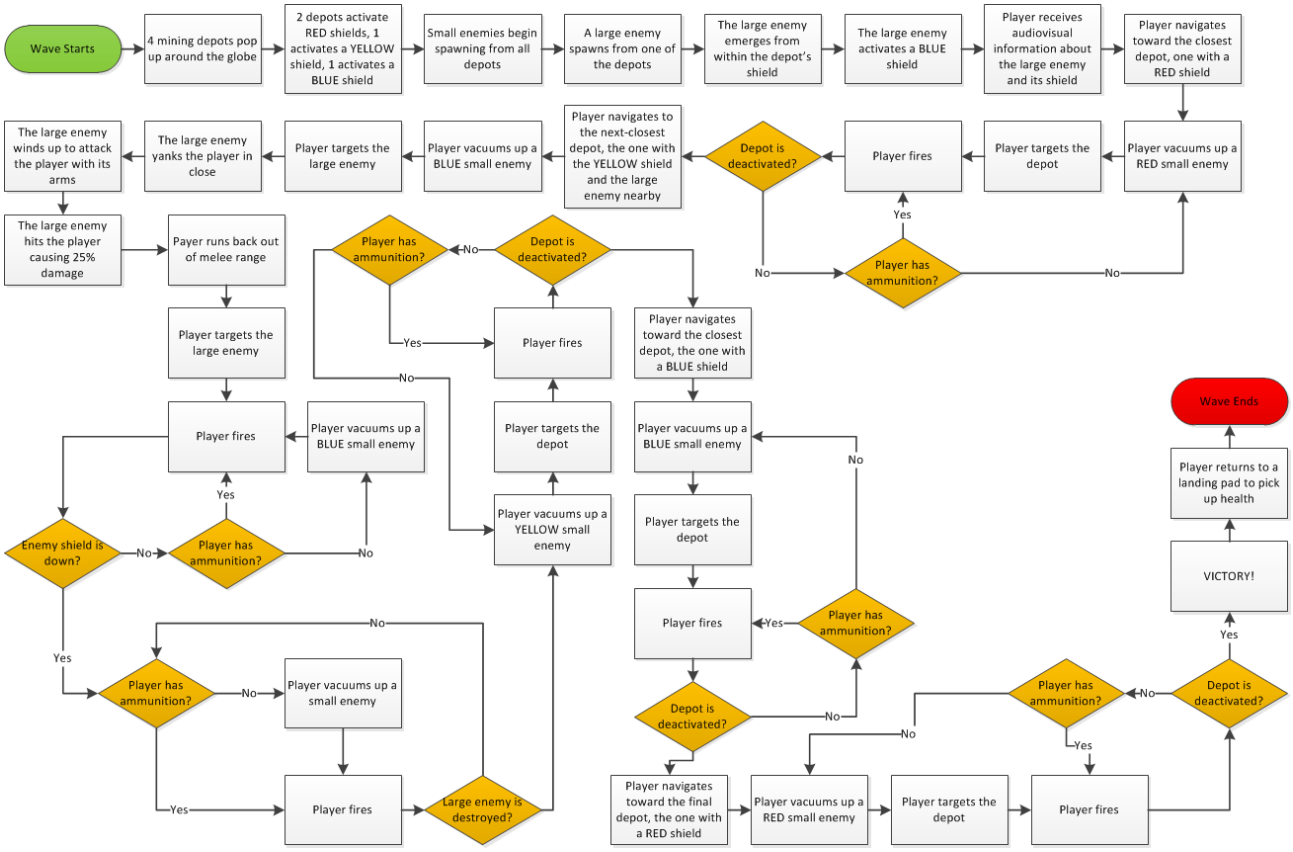


Wave 5

Win condition: Deactivate all four active mining depots.

Loss condition: Lose all health.

Notes: This wave introduces the large enemy.

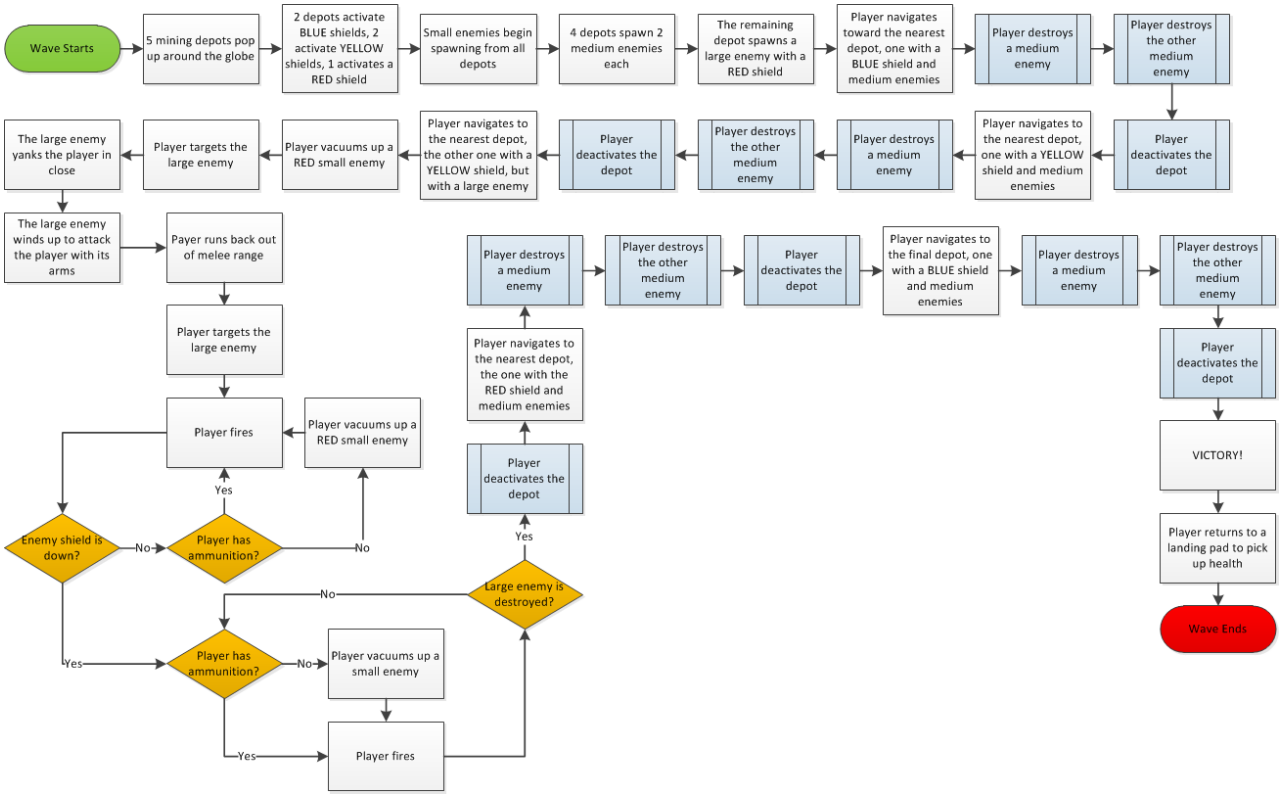


Wave 6

Win condition: Deactivate all five active mining depots.

Loss condition: Lose all health.

Notes: This wave combines all regular enemy types. The player is assumed to be familiar with the procedure for destroying small and medium enemies and deactivating depots. As such, these actions are displayed in the flow chart below as sub-processes.

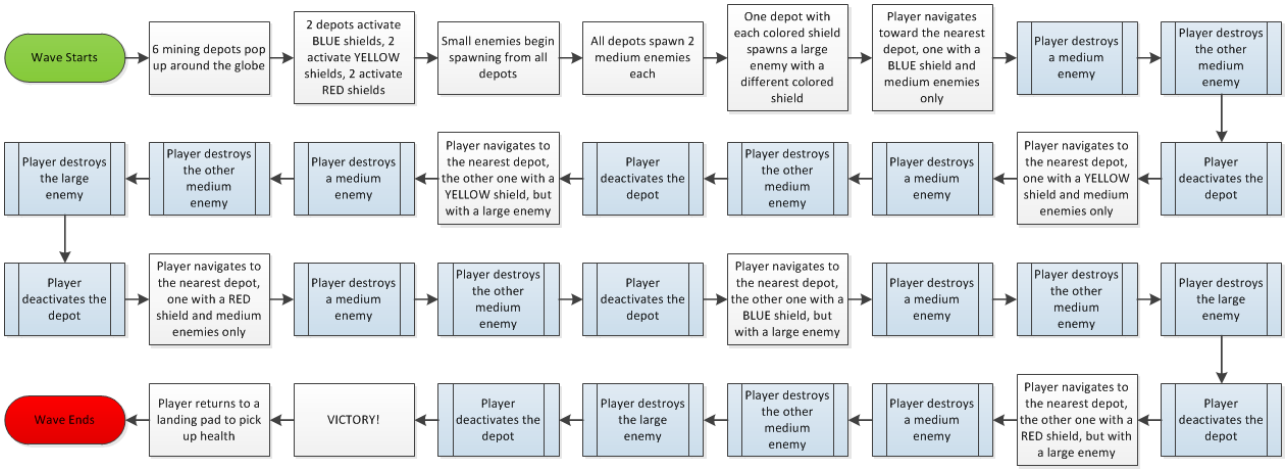


Wave 7

Win condition: Deactivate all six active mining depots.

Loss condition: Lose all health.

Notes: The player is now assumed to be familiar with the procedure for destroying all regular enemy types. These actions are displayed accordingly in the flow chart.

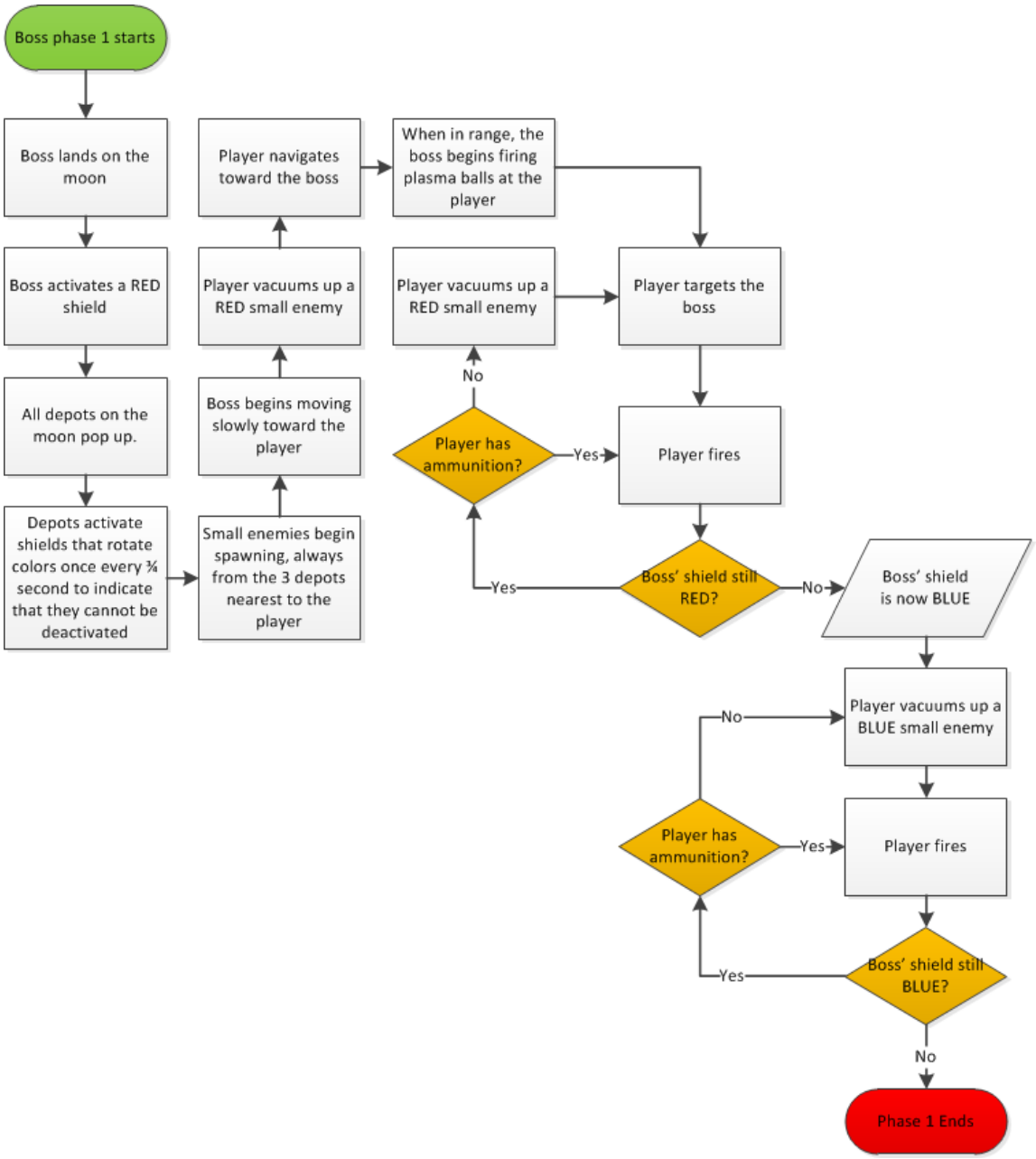


Wave 8

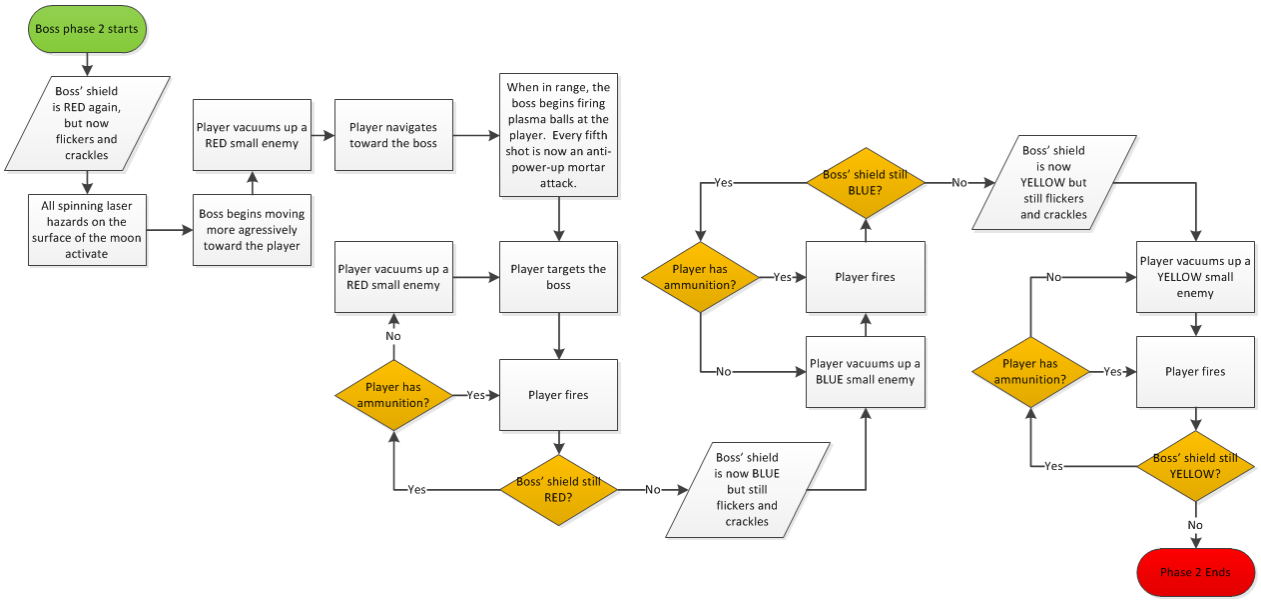
Win condition: Defeat the boss.

Loss condition: Lose all health.

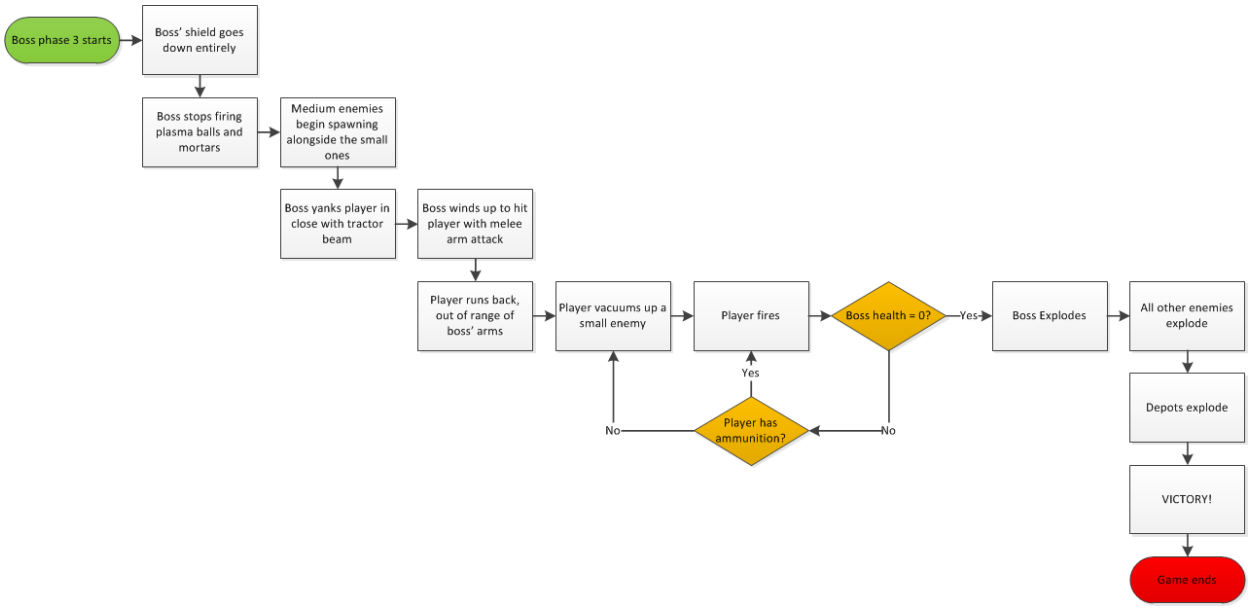
Phase 1



Phase 2



Phase 3



Concept

Instructions

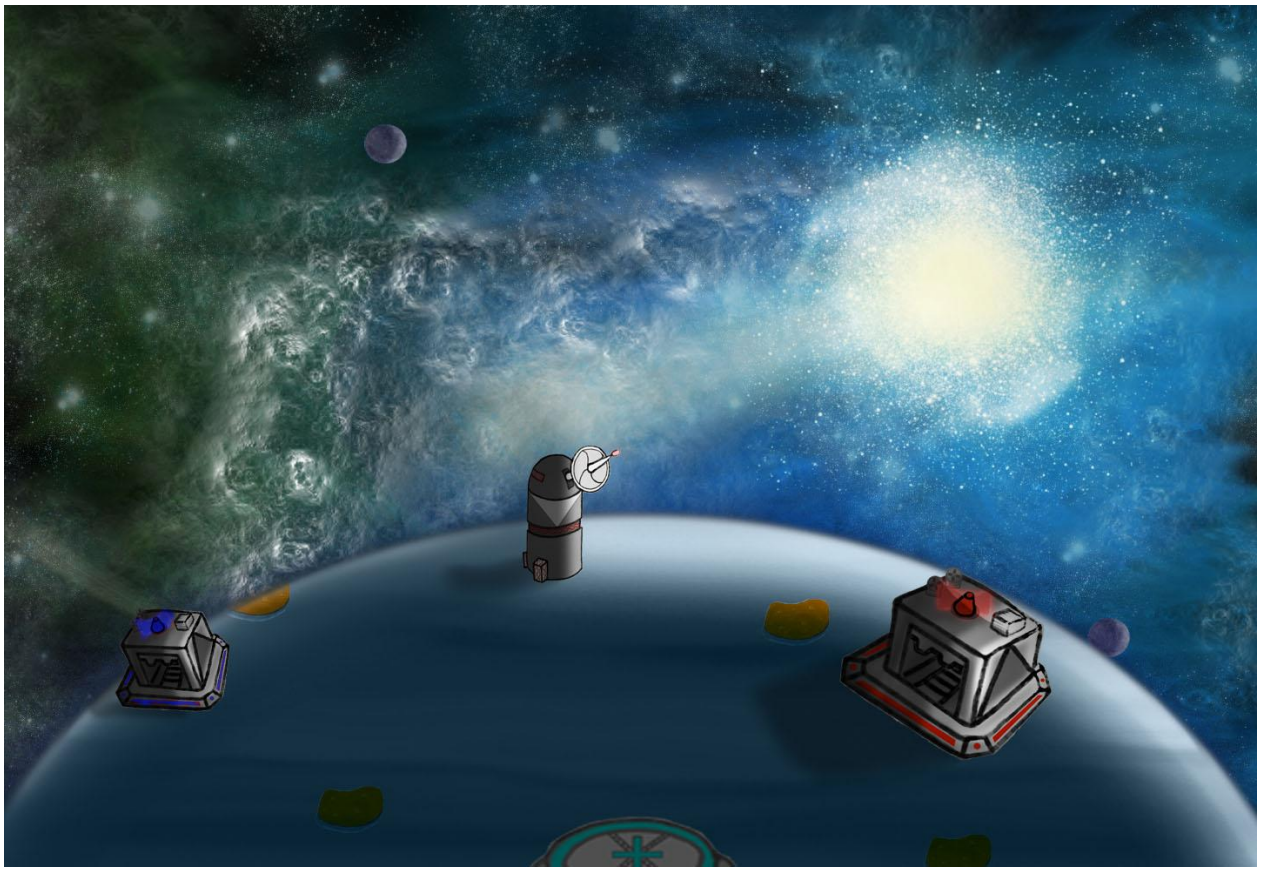
Create concept pieces for each environment. This should be from the gameplay camera angle. Also include topdown views for each.

Thursday, December 08, 2011
11:14 AM

Concept

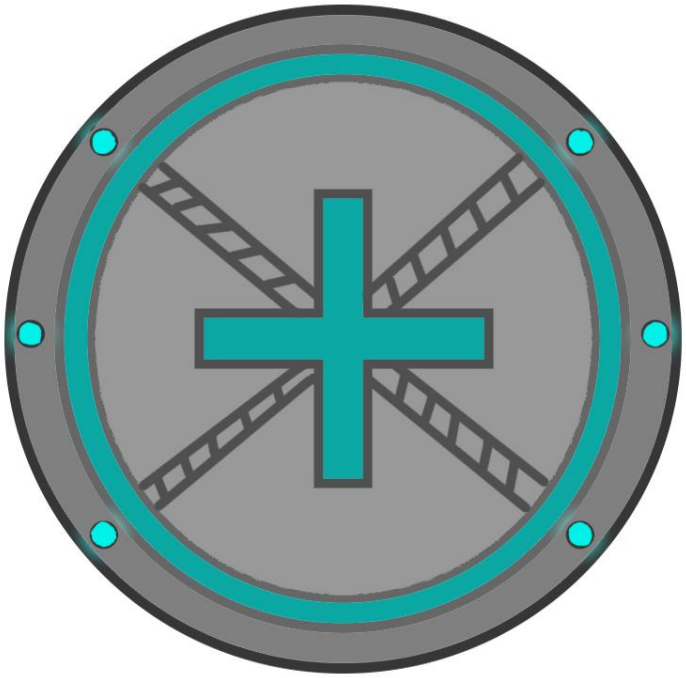
Moon

In-game view:

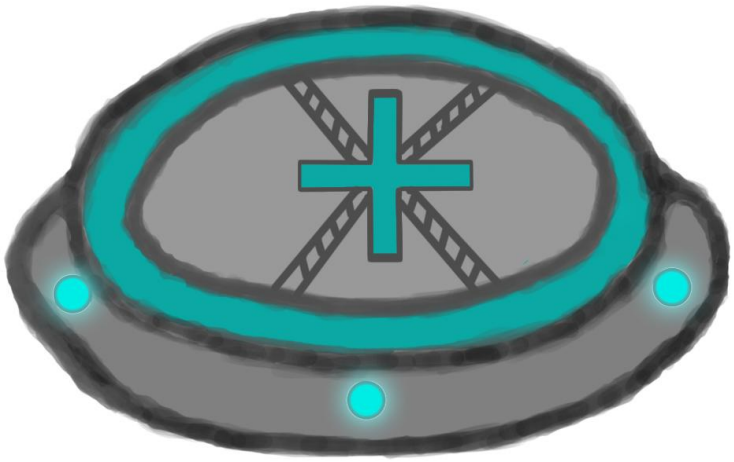


Landing Pad

Top view:

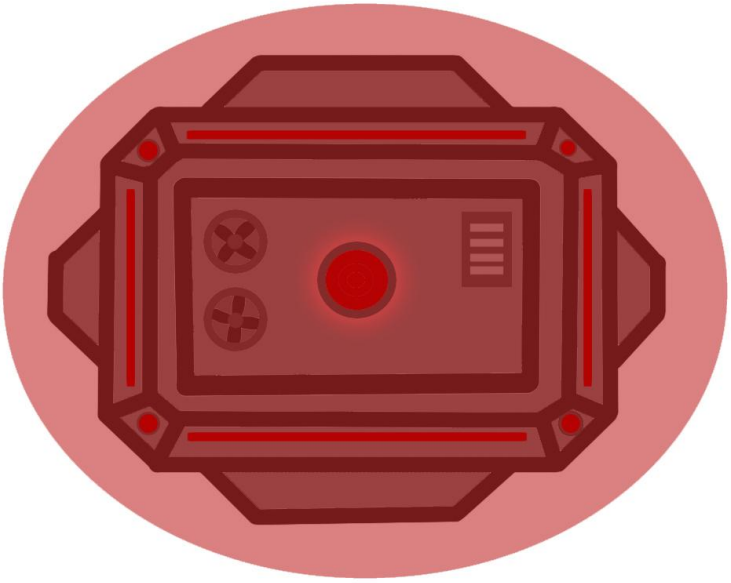


In-game view:

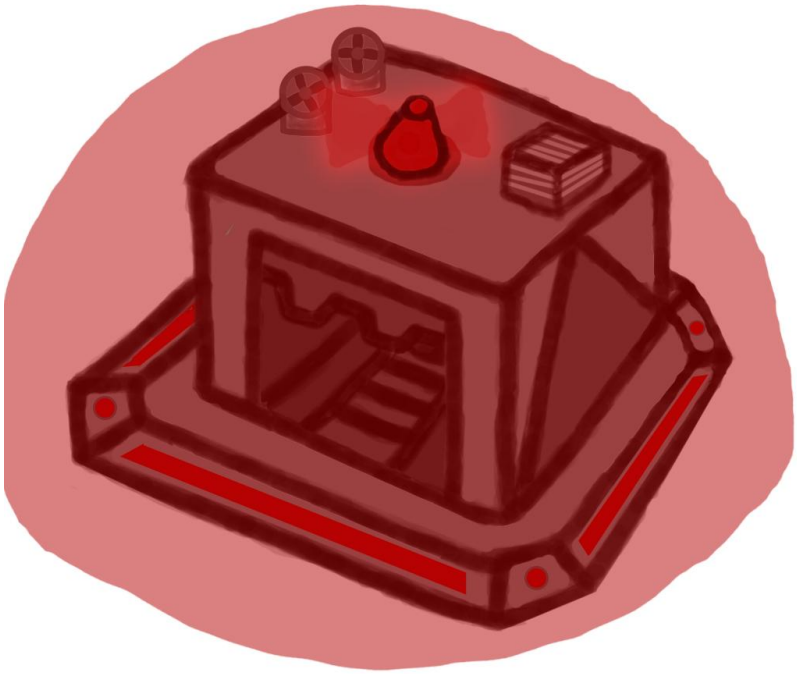


Mining Depot

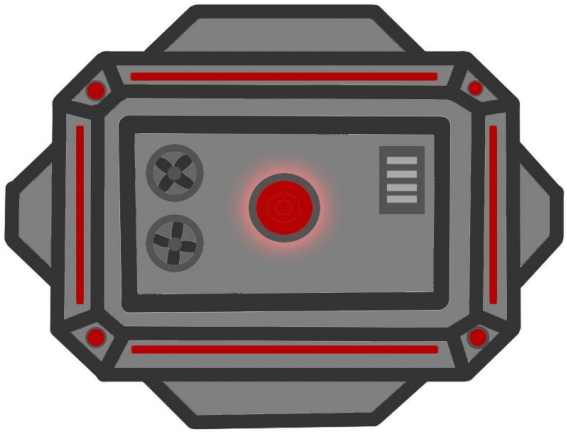
Top view in the above-ground position with shield up:



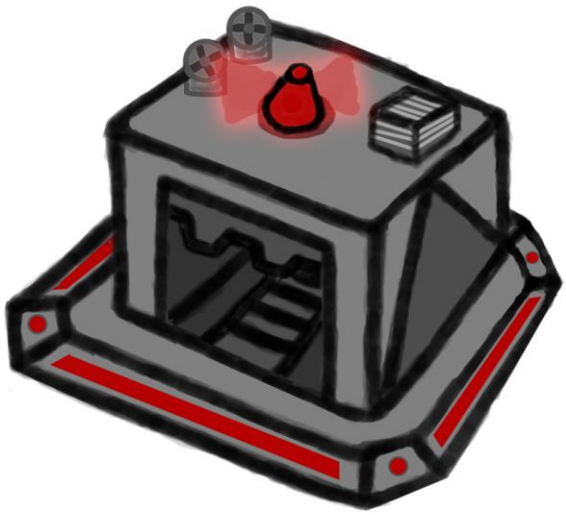
In-game view in the above-ground position with shield up:



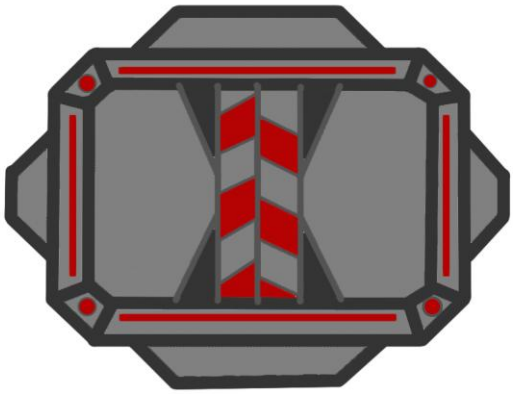
Top view in the above-ground position without the shield up:



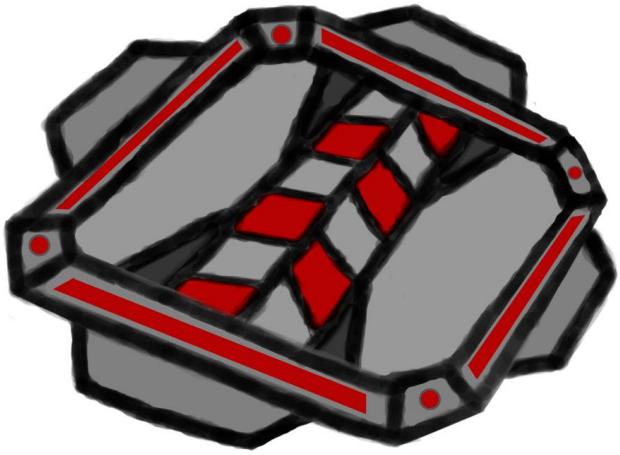
In-game view in the above-ground position without the shield up:



Top view in the below-ground position:

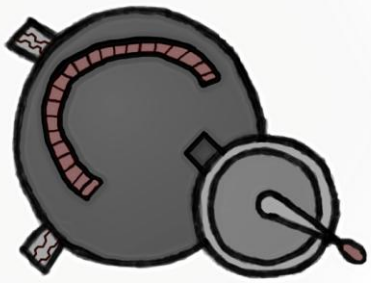


In-game view in the below-ground position:

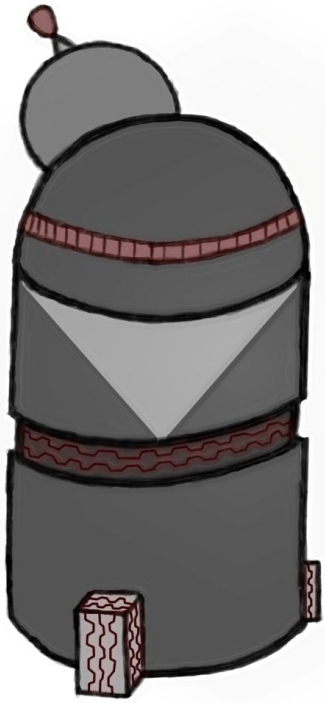


Communications Tower

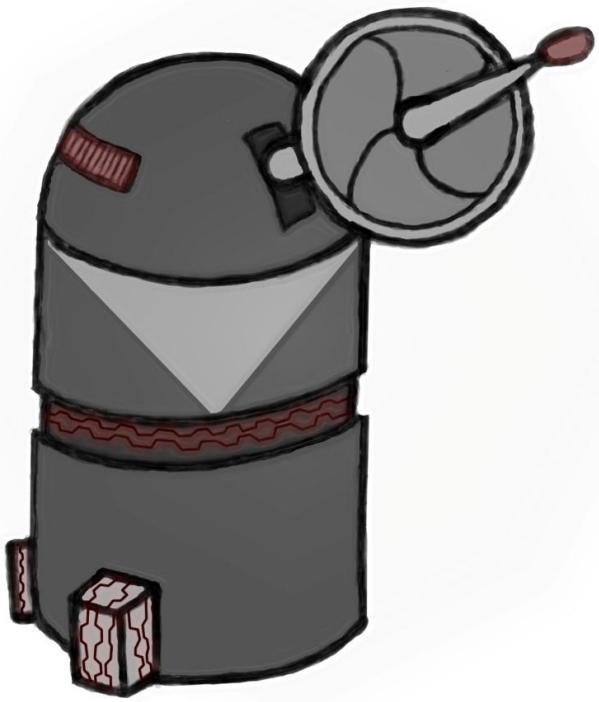
Top view:



In-game view from the back:

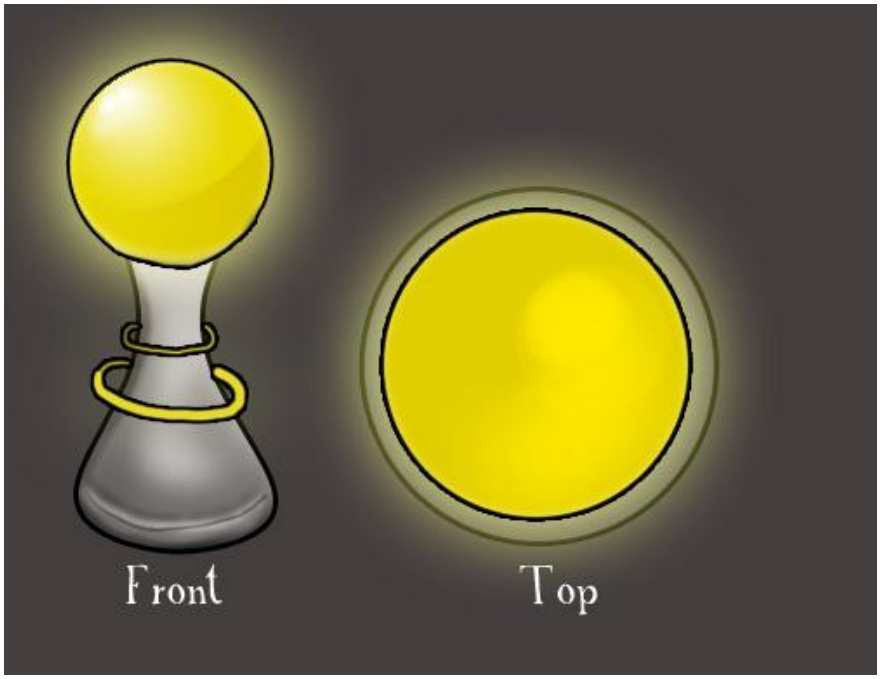


In-game view from the font:



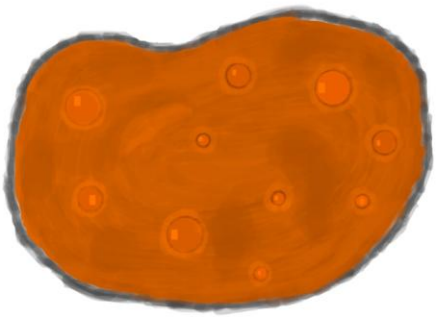
Light Poles

Top and in-game views:



Waste Pool

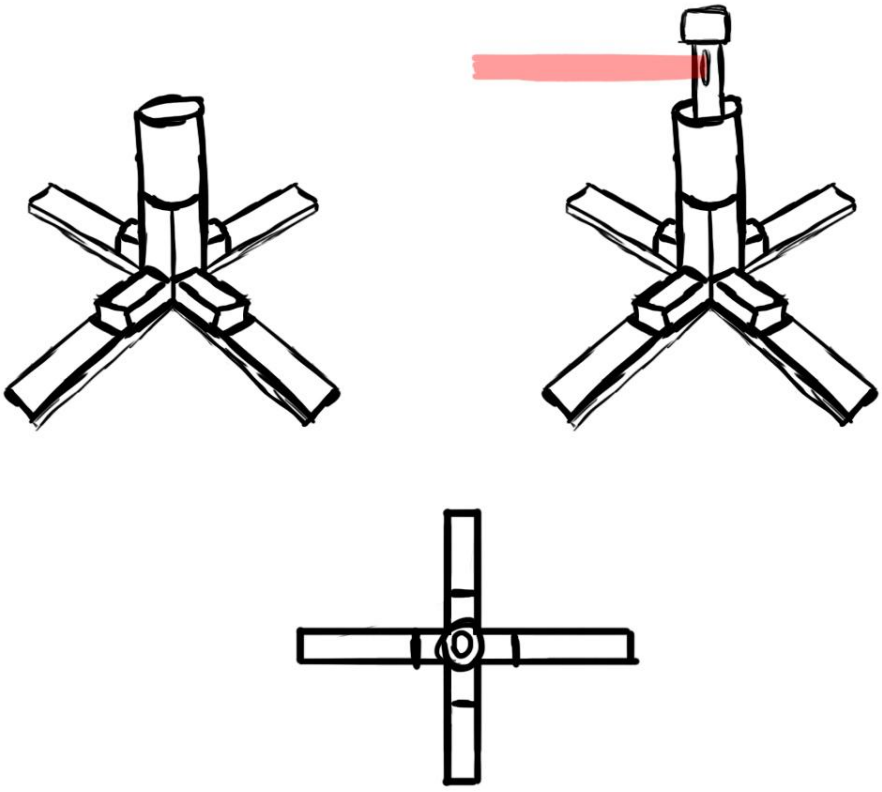
Top view:



In-game view:

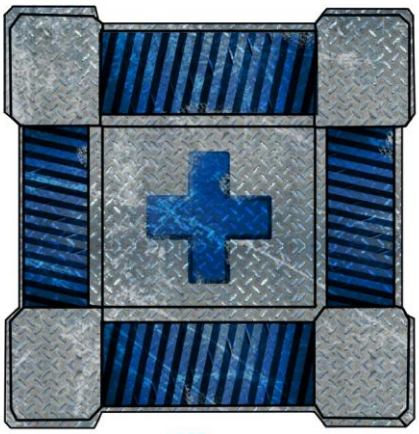
Spinning Laser

(Clockwise) In-game view in the down position, in-game view in the active position, top view:



Crate

Top and side views:

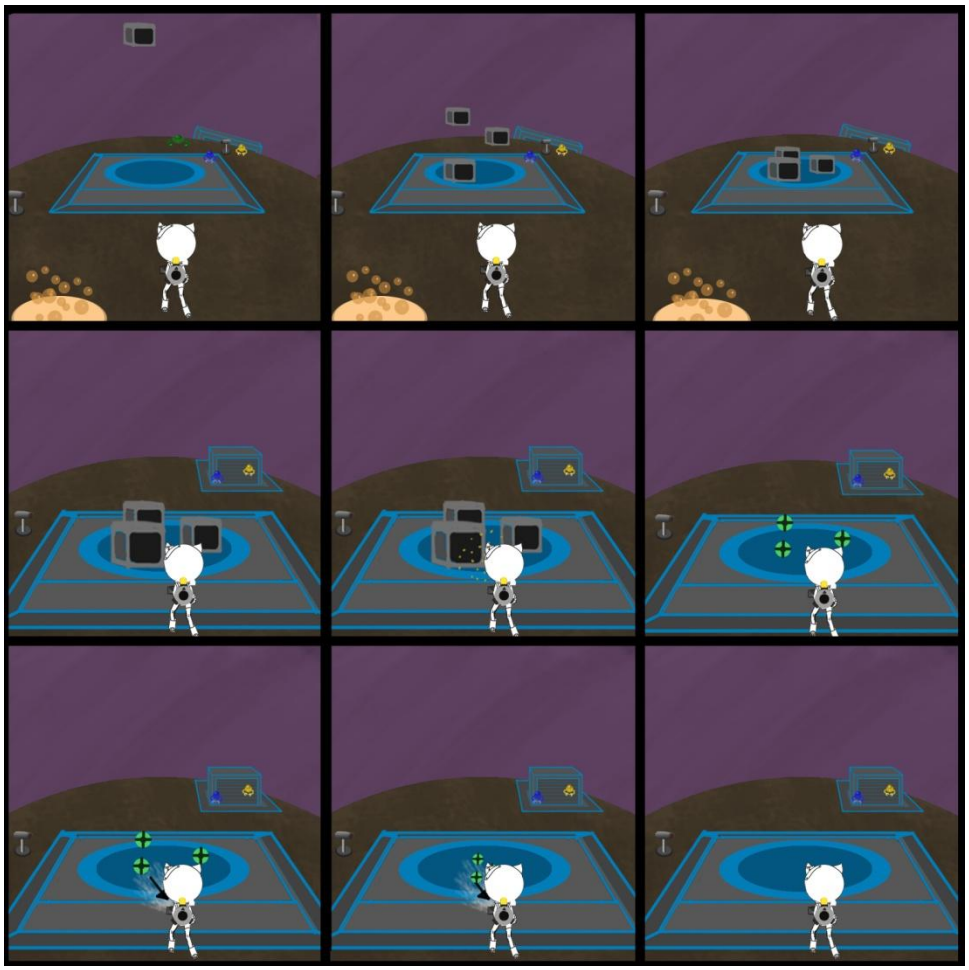


Top

Side

Health Orb

In-game view:



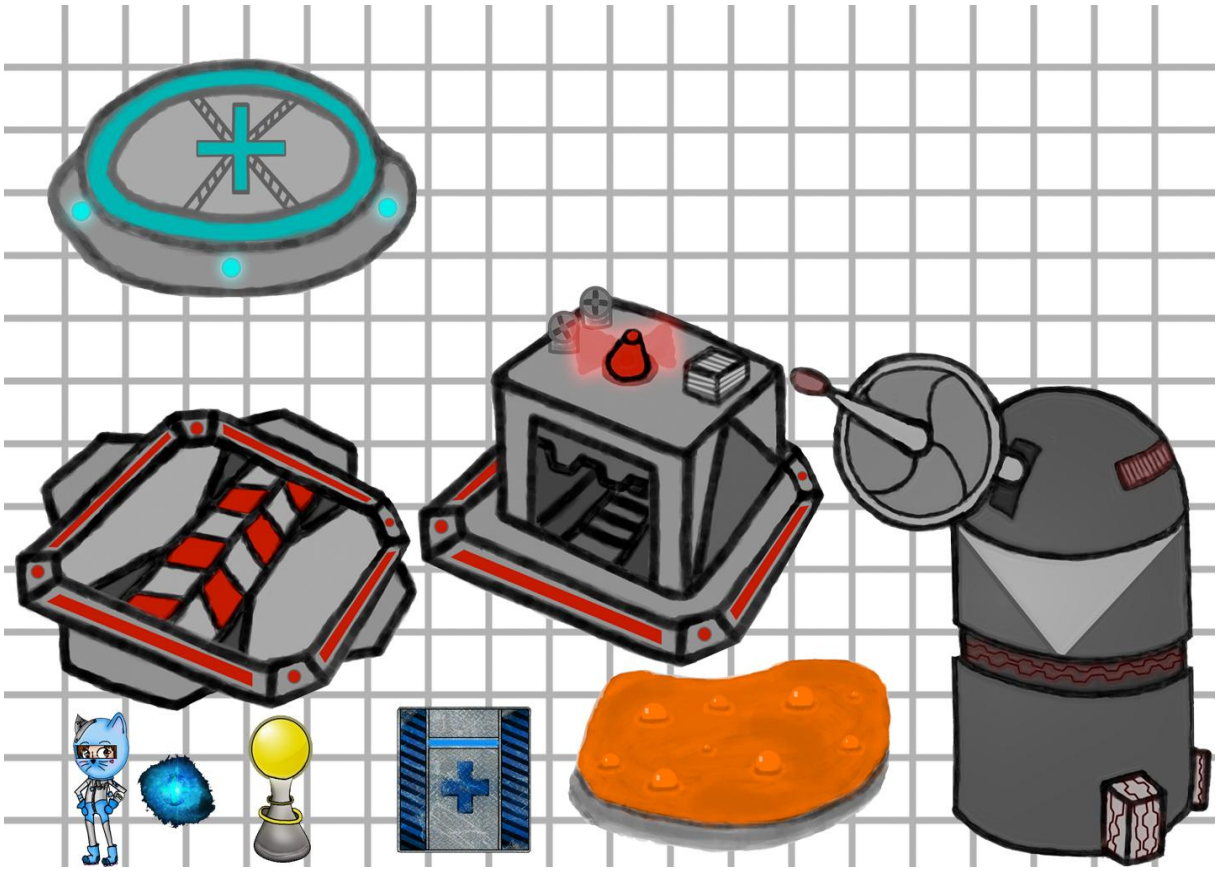
Scale

Instructions

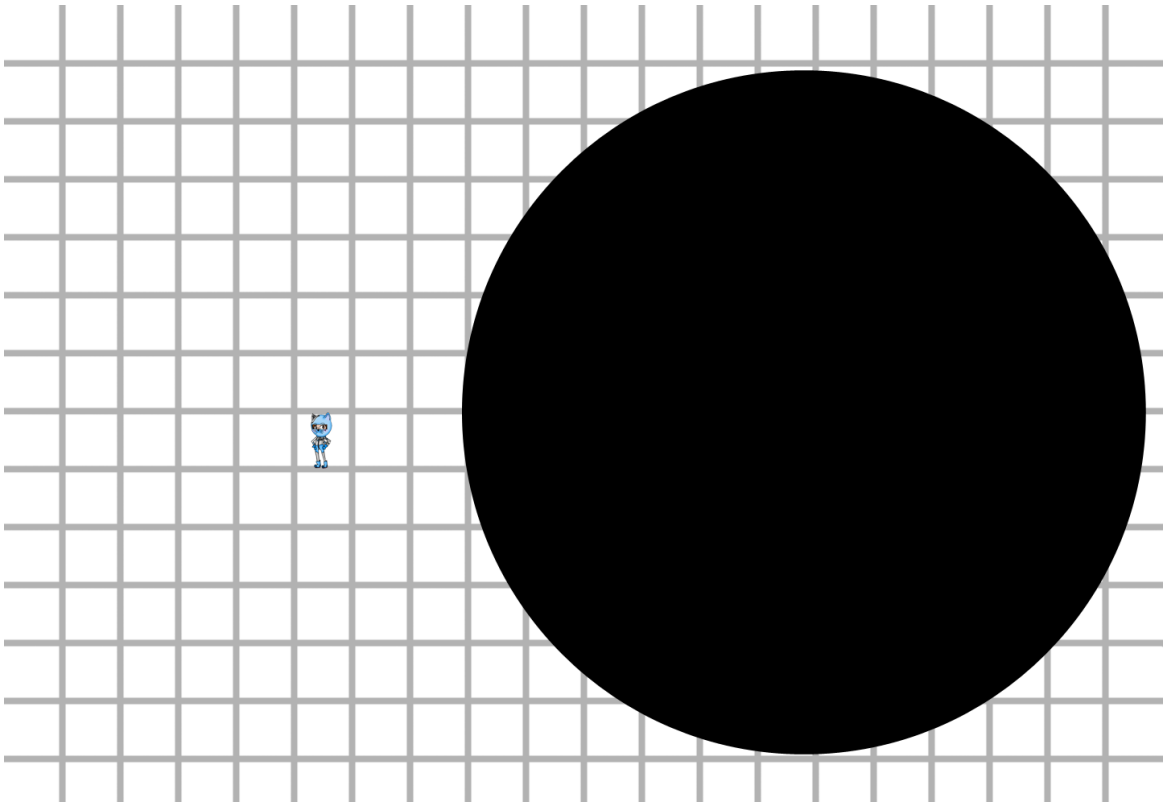
Create a scale based on the units the game is based on and show the environments relation to it.

Thursday, December 08, 2011

11:15 AM



Player To Environment Pieces



Player To Moon

Effects

Instructions

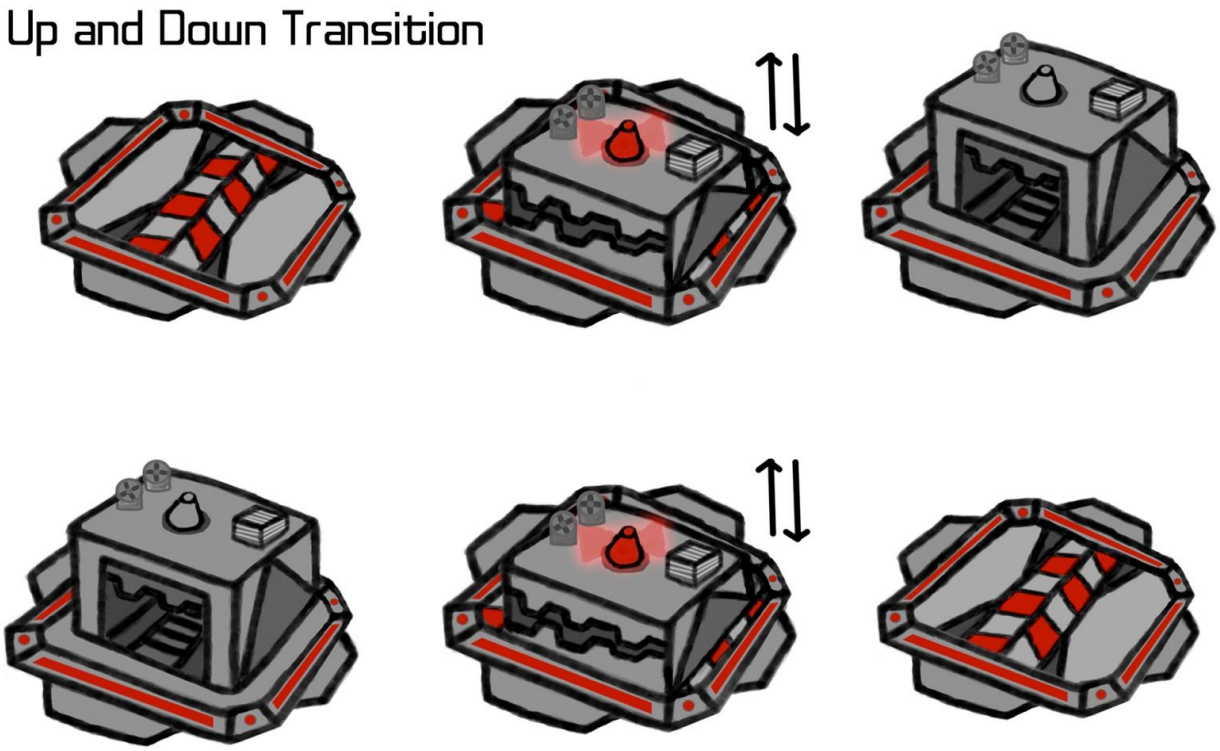
List out all the effects that are needed for the creation of each environment. This includes things like particle effects. There should also be concepts for each of the effects.

Thursday, December 08, 2011

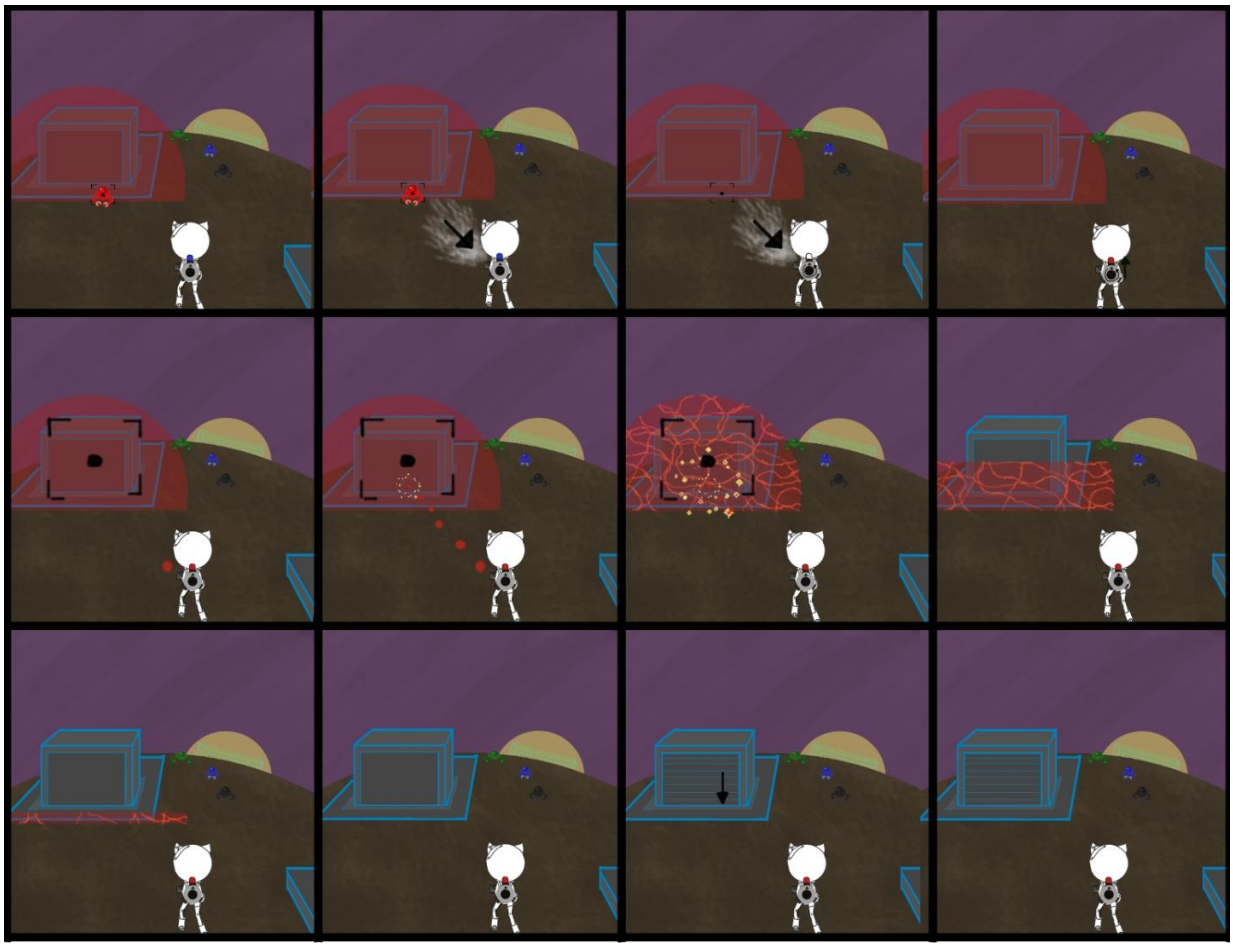
11:16 AM

Mining Depots

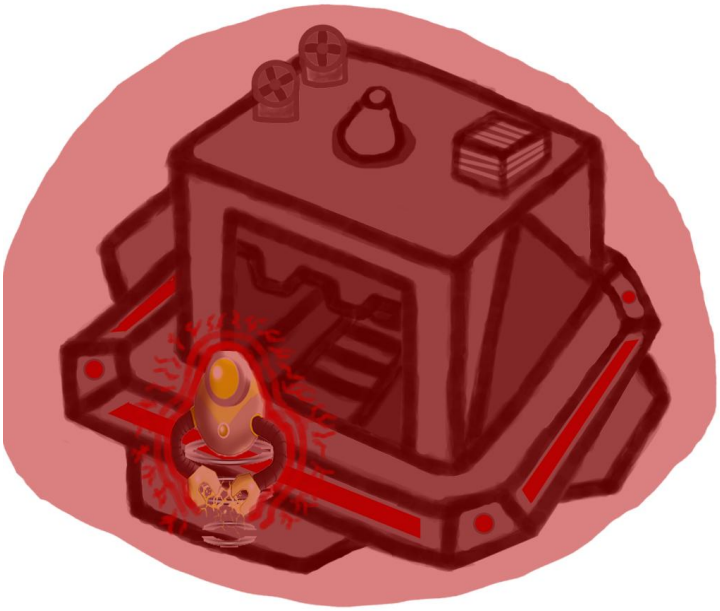
Moving up and down:



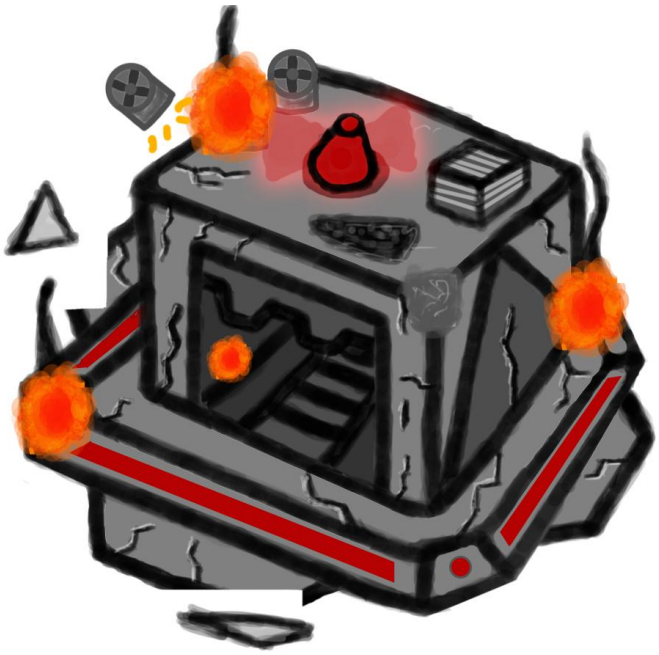
Shield deactivating:



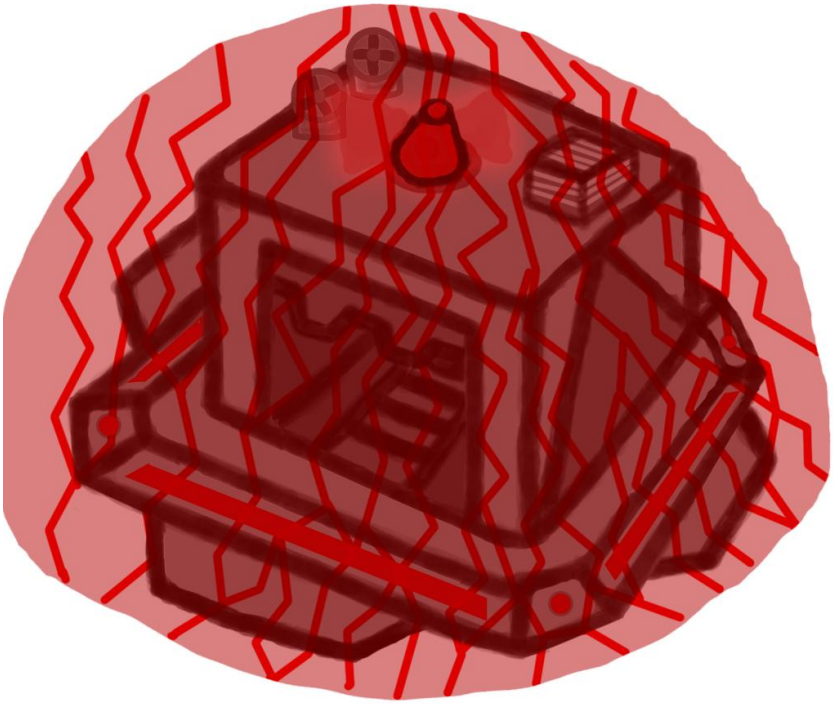
Enemy moving through the shield:



Depot taking damage:

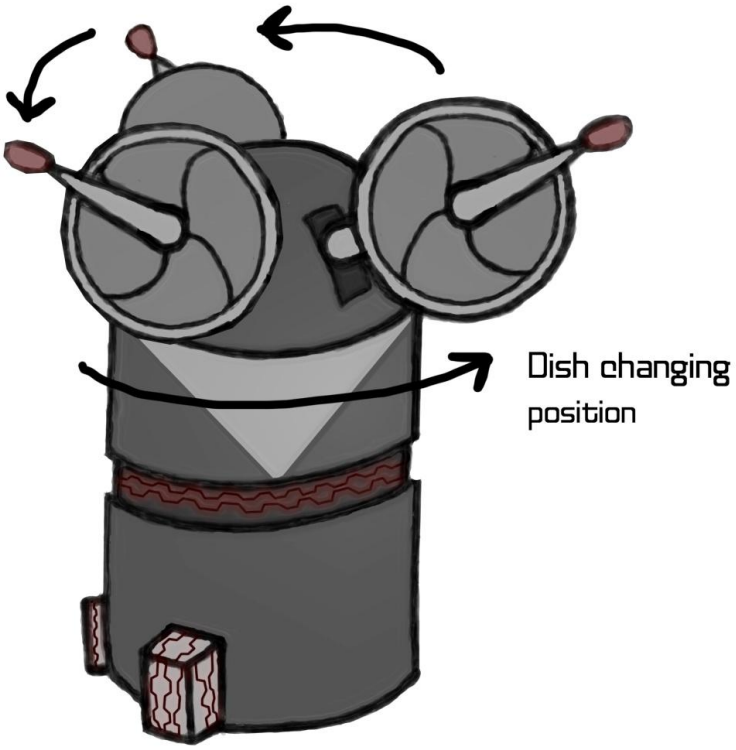


Shield taking damage:



Communications Towers

Satellite dish slowly changing position:



Waste Pools

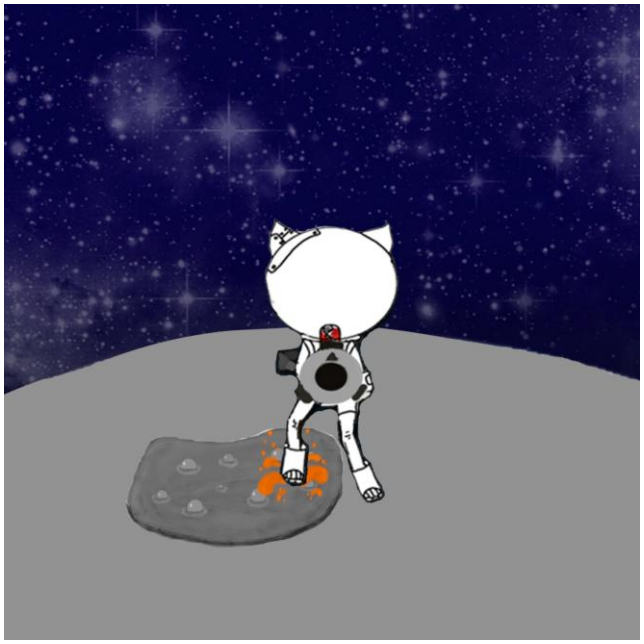
Bubbling / splashing waste:

Hit by plasma ball:

Hit by laser cannon:

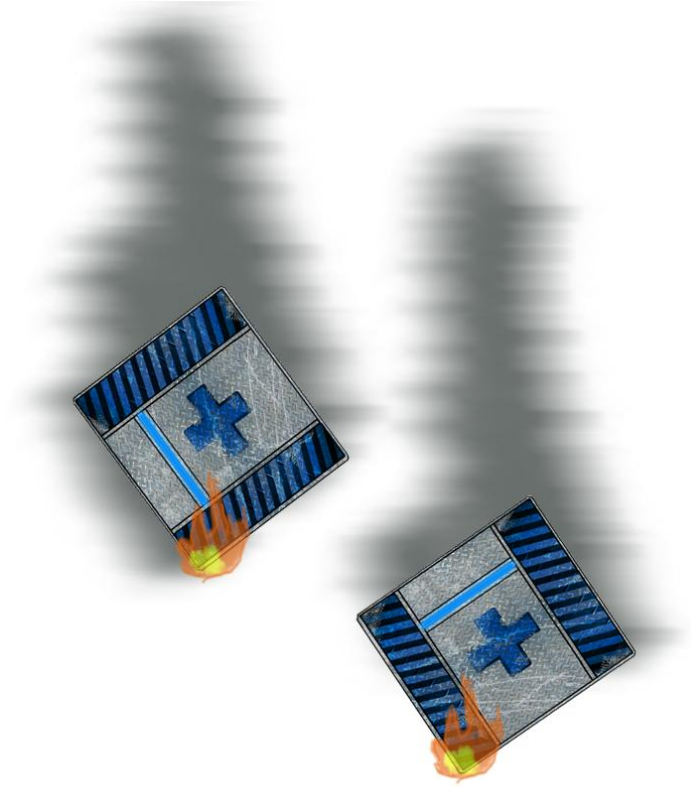
Hit by shotgun:

Player or enemy splashes in pool:

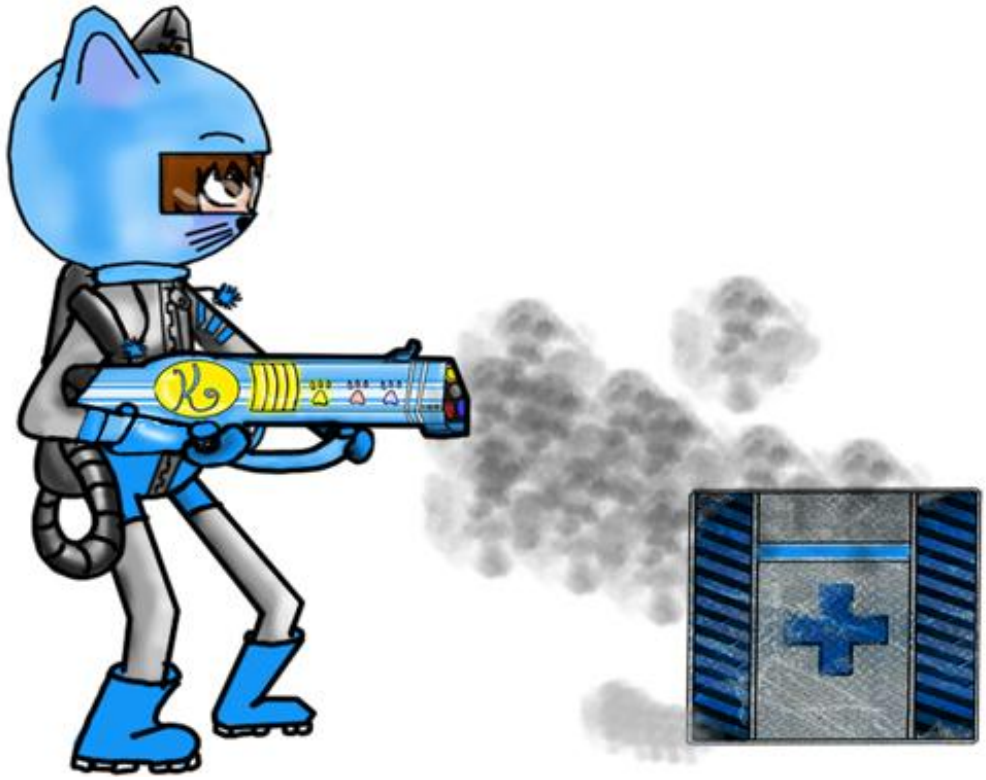


Crates

Dropping in from orbit:



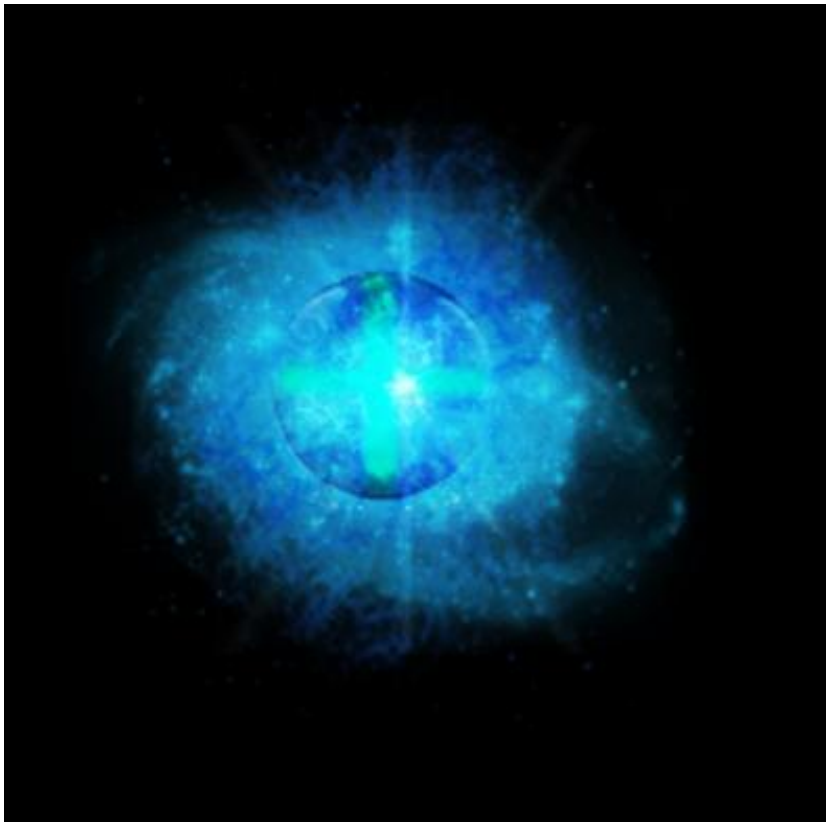
Hit by PUSH:



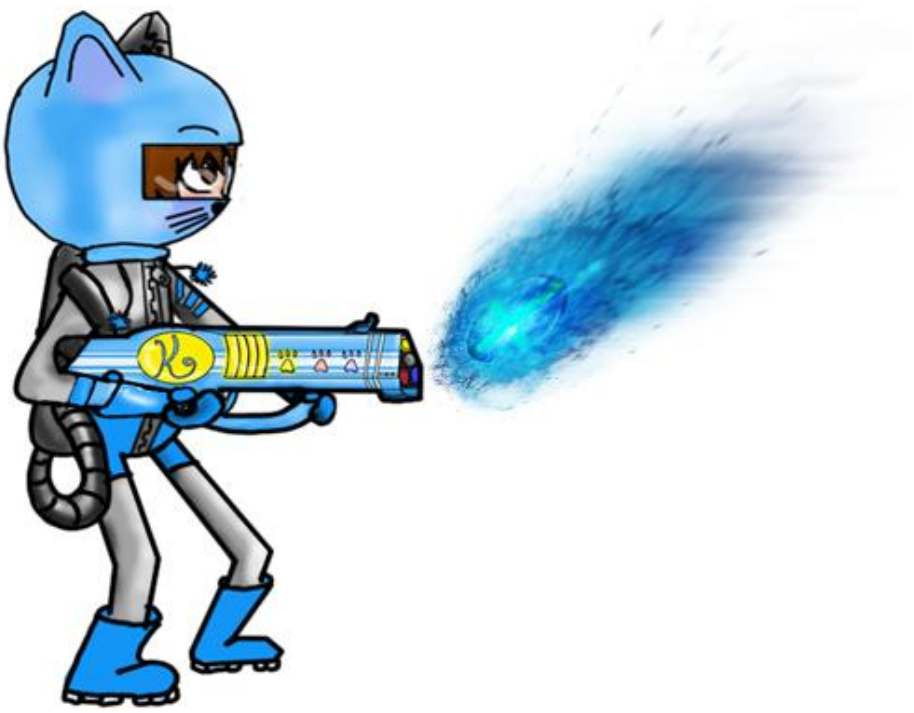
Breaking open:

Health Orbs

Glowing / sparkling:



Getting picked up / vacuumed up:



Sounds

Instructions

List out all the sounds that are needed for the creation of each environment. This includes things like ambient sounds (birds, wind, etc…). Basically anything that makes a noise in the environment should be listed here.

Friday, December 16, 2011
11:12 AM

Sounds

Ambient
Environmental music for regular waves
Environmental music for boss battle
Meteorites hitting moon surface

Mining Depots
Hydraulic noise of mining depot moving or down
Steam / air escaping when depot does up or down
Shield activating

Shield deactivating
Enemy moving through shield
Mining depot taking damage
Shield taking damage

Communications Tower
Motor noise for satellite dish changing position
Radio chatter when the player get close

Waste Pool
Bubbling waste
Splashing waste
Pool hit by plasma ball
Pool hit by laser cannon
Pool hit by shotgun
Player or enemy splashes in pool

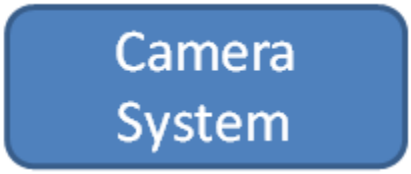
Spinning Laser
Popping up
Going down
Laser beam buzzing
Beam hits something

Crate
Dropping in from orbit
Hit by PUSH
Breaking open

Health Orb
Glowing / sparkling
Getting picked up / vacuumed up

Camera System

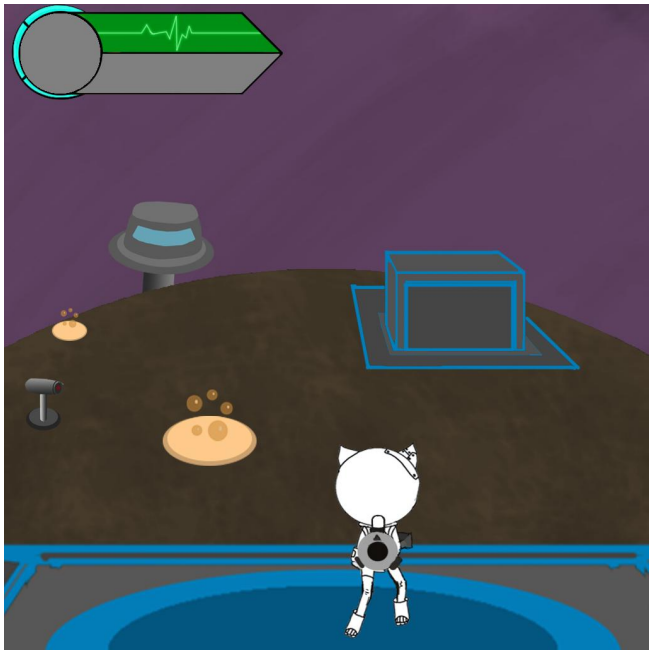
Thursday, June 20, 2013
10:17 AM



Camera Perspective
3rd person camera that cannot zoom in or out. The player will be take up 30% of the screen with the remaining 70% being the environment and enemies. In the very center of the screen will be the aiming reticle.

Camera Behavior
The player will lock to the cameras current view direction, unless they are not moving or shooting. If they aren't doing those things the camera can be rotated fully around the player, allowing for you to see the character model fully. Once the player moves or shoots the character will lock to where the camera is facing. If the player backs into an object that would otherwise obstruct his view the object goes 80% transparent, allowing for the player to still see themselves but also see the obstruction.

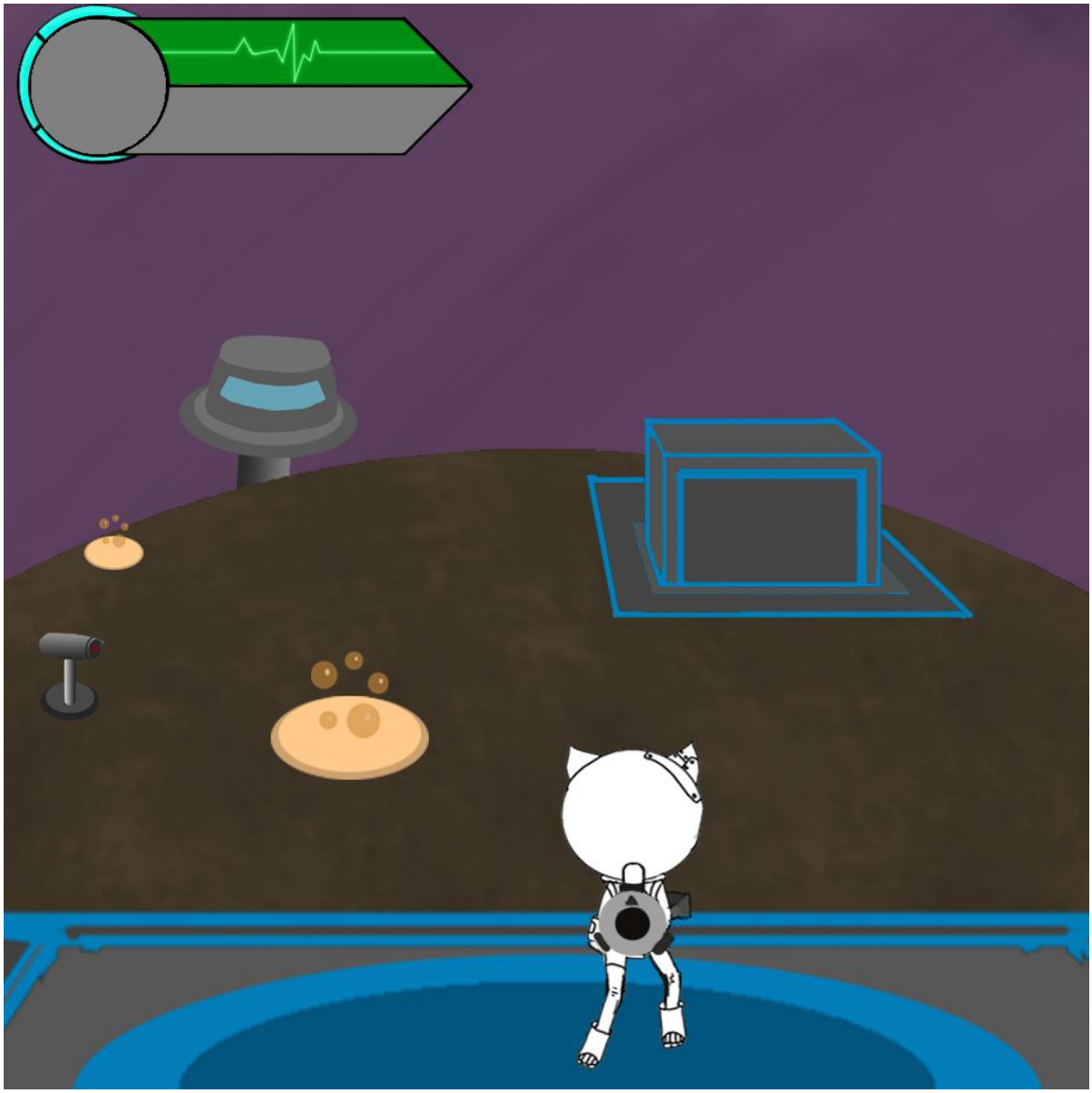
Scripted Events
There are several scripted camera events in Critical Mass. Both of these events basically put the camera on rails, forcing it to look and move in the direction we choose. The first event is when the player lands on the planet. The second is when the boss arrives. The last is when the boss is destroyed and the player wins.



Concept

Thursday, December 15, 2011
10:19 AM

Planet



Technical Document

Critical Mass

Team Gnome Toss

Ian Alcid	Hazards08@fullsail.edu
Ryan Cartier	ryanc13@fullsail.edu
Corey Morehead	coreyjamesmorehead@gmail.com
Bob Pasekoff	bpasekoff@fullsail.edu
Thomas Stefl	t_stefl@yahoo.com

Evan Wright shugoikari@fullsail.edu

Technical Design Overview

We as a team will use this document to lay the grounds for a coding structure to be followed in the months to come. This document is meant to be as extensive and thorough as possible in order to allow this window of time to be spent on design allowing us to later work entirely on actual code and debugging rather than spending time designing architecture. The team's entirety wants to make a fun game which plays well within the system constraints we have. Ultimately this document is a preplanned map to unify the teams architectural design process.

Milestone Deliverables

Core

- ***The Moon (spherical world)***
    - Collision with surface objects
- ***Player Movement***
    - The player avatar
    - The player moving across the surface of a sphere
- ***Shooting/Targeting***
    - Third person camera
    - Picking
    - Reticle
    - Single projectile color/type (Red)
- ***Small Enemies***
    - Pack and flocking algorithms
    - Slot system
    - Moves toward player when nearby
    - Collision with player triggers attack
    - *Colorized
- **Medium Enemies**
    - Attack
        - Standard ranged attack.
        - Mortar ranged attack.
    - Behavior States
        - Staying at a distance from the player.
- ***Color System***

    - Implement on the small enemies
    - Allow player to get all three weapon types
    - Damage affects colorized shields
- ***Pull Mechanic***
    - Able to pull and consume targeted enemy
    - Gain the power up from the affected target
- ***The Depots***
    - Colorized shields
    - Needed to complete a wave
- ***Win State***
    - Must be able to destroy depots in order to complete wave(s).
- ***Lose State***
    - Game Over will appear when player loses all of their lives.
- ***HUD***
    - Display health, lives, wave, ammo and type
- ***Sounds***
    - Sounds can be loaded in
    - Sounds can be triggered and played
- ***Camera***
    - User can look around
    - Camera tracks player model
    - Projectiles shoot based off camera perspective (reticle)
    - Objects in between player and camera become transparent
    - Camera itself must have collision with objects (and is an object itself)
- ***Push***
    - Push small enemies away

- Deflect enemy projectiles

Alpha

- **Large Enemies**
  - Attack
    - Tractor beams and melee attack.
- **Shotgun (Yellow)**
  - Small radius aoe/cone effect short distance away from player
- **Laser Cannon (Blue)**
  - Able to pierce enemies.
- **Spinning Laser Hazard**
  - **Collision with player**
- **Visual FX**
  - Rendering particles
  - Manages all active and inactive particle effects
  - Basic billboarded sprites

Boss Event

- **Boss spawning**
  - Phase 1
    - Shields 100% up
    - Spawn small enemies
    - Shoots you with plasma balls
  - Phase 2
    - All actions speed up
    - Spinning laser traps activate
  - Phase 3

        - Shields down, just large enemy

- ***Menus***
    - Main Menu
        - Play Game
        - Options
        - Quit
    - Options
        - SFX
        - VFX
        - *Difficulty
    - Pause
        - Stop gameplay
        - State can be added or removed from state machine

Beta

- ***Environment Pieces***
    - Exploding Barrel Hazard
        - Splash damage
        - Visual Effects
    - Acid Pools Hazard
        - Placement
        - Collision with Player
    - Light Posts
        - Placement and lighting for the individual posts
    - Communication Towers

        - Place these pieces on the surface.
- ***Credits***
    - Contains all of the credits for the game
- **Feedback Integration**
- **Asset Integration**
    - Application of textures
    - Integration of models

Gold

- ***Game Balance***
    - Wave balancing
    - Damage balancing
    - Resilience of player and enemies
- ***Visual Polish***
    - Any remaining unwanted artifacts are removed
    - Team is happy with the final product
- ***Q/A***
    - Completed bug list.

Development Environment:

Programs:

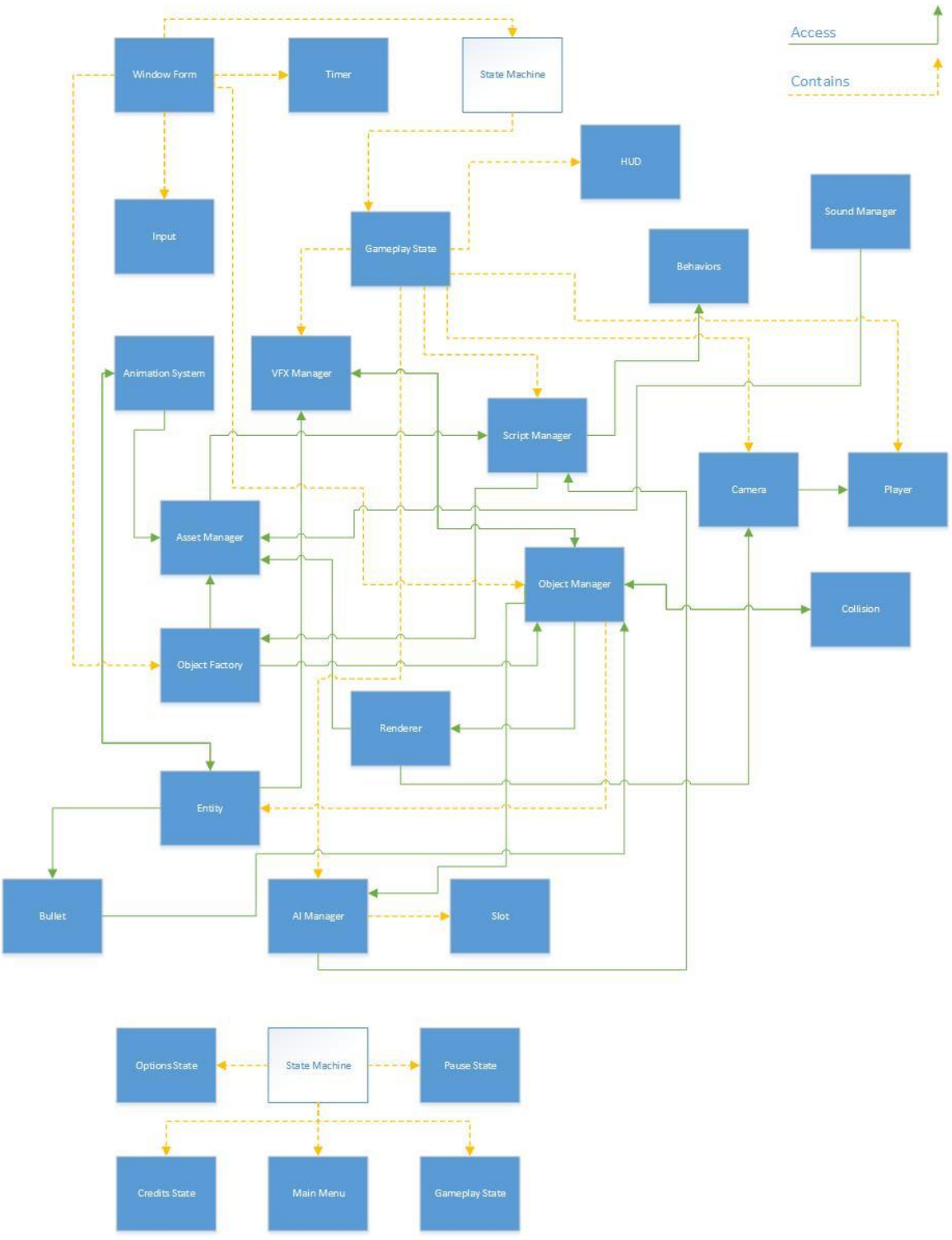
- IDE
    - Primary: Microsoft Visual Studio 2010
    - Secondary: Microsoft Visual Studio 2012
- Shader creation
    - Primary: Notepad++ v6.3.2
    - Secondary: NShader 1.3
- Source Control
    - Perforce P4V 4.8.3
- Texture Editor
    - GIMP 2.8
- Task and Time Management
    - Hansoft v7.0767

Third Party Libraries:

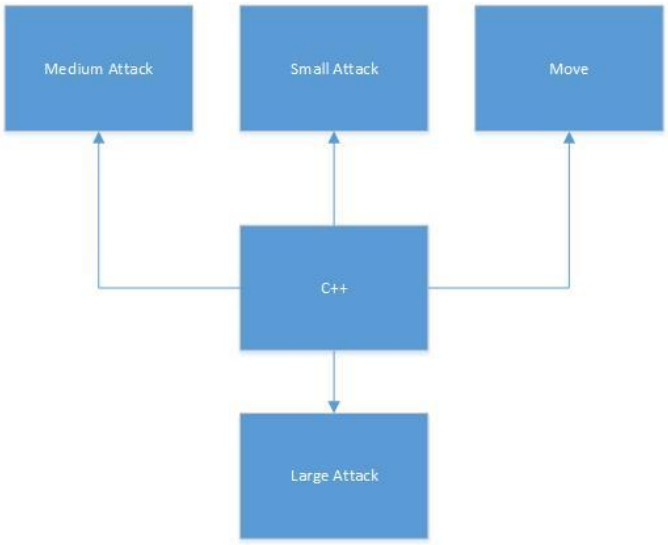
- Sound
    - WWise 2013.1.1
- Rendering
    - OpenGL 3.0
    - OpenGL Extension Wrangler (GLEW) 1.9
- Mesh/Particle Editor
    - Autodesk Maya 2013
- Parse XML
    - TinyXML 2.0
- Scripting

- Lua 5.2

System Architecture Flowchart



Lua Flowchart


Medium Attack
Small Attack
Move
C++
Large Attack

System Architecture Context Model Description

- **Windows Form**
  - Input
    - Init
  - Timer
    - Reset
    - Update
    - GetDeltaTime
  - Renderer
    - Init
    - Shutdown
    - Update
  - SoundManager
    - Init
    - LoadAllSounds
    - Shutdown
  - StateMachine
    - Init
    - Input
    - Update
    - Render
    - Shutdown
- **A.I. Manager (contained within A.I. Manager)**
  - ScriptManager
    - GetAIBehavior

- **Slot (contained within A.I. Manager)**
    - N / A
- **VFX Manager (contained within Gameplay State)**
    - AssetManager
        - LoadTexture
        - LoadShader
        - LoadMesh
        - LoadAnimation
    - Renderer
        - AddVFXToRenderList
- **HUD (contained within Gameplay State)**
    - Renderer
        - AddToUIList
        - ClearUIList
        - DrawUI
    - AssetManager
        - GetTexture
        - GetMesh
- **Object Factory (contained within Gameplay State)**
    - N / A
- **Object Manager (contained within Gameplay State)**
    - Entity
        - Update
        - Check Collision
    - Renderer
        - AddToRenderList

        - ClearRenderList
        - DrawRenderList
    - Collision Library
        - CheckCollisions
    - ObjectFactory
        - AddObject
        - RemoveObject
    - AI Manager
        - AddDepot
- **State Machine (contained within Windows Form)**
    - State
        - OnEnter
        - OnExit
- **Gameplay State (sub of State Machine)**
    - State Machine
        - ChangeState
        - PushState
    - Sound Manager
        - PlaySound
        - PauseSound
        - ResumeSound
        - StopSound
    - Renderer
        - AddToRenderList
        - ClearRenderList
        - DrawRenderList

        - ChangeRenderState
        - PopulateLightList
        - ClearLightList
        - DoShading
        - DoPostEffect
        - AddToUIList
        - ClearUIList
        - DrawUI
    - Object Manager
        - CheckCollisions
        - UpdateObjects
        - RenderObjects
    - Input
        - GetInput
    - AI Manager
        - Init
        - Shutdown
        - Update
- Script Manager
        - Initialize

- **Pause State (sub of State Machine)**
    - State Machine
        - ChangeState
        - PopState
        - PushState
    - Sound Manager

        - PlaySound
        - PauseSound
        - ResumeSound
        - StopSound
    - Renderer
        - AddToUIList
        - ClearUIList
        - DrawUI
- **Options State (sub of State Machine)**
    - State Machine
        - PopState
    - Sound Manager
        - PlaySound
        - PauseSound
        - ResumeSound
        - StopSound
        - ChangeVolume
    - Renderer
        - AddToUiList
        - ClearUIList
        - DrawUI
- **Main Menu State (sub of State Machine)**
    - State Machine
        - ChangeState
        - PushState
    - Sound Manager

        - PlaySound
        - PauseSound
        - ResumeSound
        - StopSound
    - Renderer
        - AddToUiList
        - ClearUIList
        - DrawUI
- **Intro State (sub of State Machine)**
    - State Machine
        - ChangeState
    - Sound Manager
        - PlaySound
        - PauseSound
        - ResumeSound
        - StopSound
    - Renderer
        - AddToUiList
        - ClearUIList
        - DrawUI
- **Entity**
    - SoundManager
        - PlaySound
        - PauseSound
        - ResumeSound
        - StopSound

- AI Manager
    - AIUpdate
- Renderer
    - Render
- Collision Entity(Sub of Collision Library)
    - CreateColEnt

- **Player (derived from Entity) (contained within Gameplay State)**
    - Input
        - Update
    - Camera
        - ChangeYaw
        - ChangePitch
        - ChangeRoll
        - GetViewMatrix
    - Object Factory
        - AddObject
- **Enemy (derived from Entity)**
    - Object Factory
        - AddObject
- **Projectile (derived from Entity)**
    - Object Manager
        - AddEvent

- **Asset Manager (contained within Gameplay State)**
    - N / A

- **Renderer (contained in Game)**
    - Asset Manager
        - GetTexture
        - GetMesh
        - GetShader
        - GetAnimation
- **RenderState(Sub of Renderer)**
    - N / A
- **Sound Manager (Static Class contained within Game)**
    - wWise
        - PlaySound
- **Collision Library**
    - Entity
        - GetCollisionEntity
        - HandleReaction
    - Collision Entity
        - GetType
- **Collision Entity (Sub of Collision Library)**
    - N / A
- **Input Manager (contained within Windows Form)**
    - N / A
- **Script Manager (contained within Gameplay State)**
    - Object Factory:
        - CreateObject
    - LuaScript:
        - Move

        - SmallAttack
        - MediumAttack
        - LargeAttack
- **Timer (contained within Windows Form)**
    - N / A
- **Animation System**
    - Asset Manager
        - GetAnimation
    - Entity
        - UpdateAnimation

LUA CMD (These are the Modules made in and for LUA)

- Small Attack
    - Execute
- Medium Attack
    - Execute
- Large Attack
    - Execute
- Move
    - Execute

Module Breakdown

Entity

Contains the basic information for a renderable object. It will be used for the base of all objects in our game. It has a position, a pointer to the model data contained within Asset Manager and collision bounding volume. It handles it's own collision reactions and will behave based on what type of entity it is, such as; player, enemies, boss, crates, bullets, etc.

Dependencies

- Access to the following
    - Sound
    - Asset Manager
    - AI Manager
    - VFX Manager
- Accessed by the following
    - AI Manager
    - Collision Library
    - Object Factory
    - Renderer

Return	Name	Parameters	Description
bool	Initialize	void	Initializes an object. Returns whether or not the initialize was successful.
void	Release	void	Cleans up what needs to be cleaned.
void	Update	float _elapsedTime	Updates the object

void	HandleReaction	Entity* _entity	Object reacts according to their type.
CollisionEntity*	GetCollisionEntity	void	Returns the abstract collision entity.
int	GetType	void	Returns the objects type
Model*	GetModel	void	Returns the pointer to the model.
Vec3f	GetPosition	void	Returns the entities position.
Matrix4x4f	GetWorldMatrix	void	Returns the world matrix
void	SetType	int _type	Sets the entities type.
void	SetModel	Model* _model	Sets the entities model.
void	SetPosition	Vec3f _position	Set the position
void	SetWorldMatrix	Matrix4x4f _worldMatrix	Set the world matrix.

Module Author(s)

Evan Wright

HUD

Displays important data to the player, like: player health, player ammo, current weapon, minimap, etc. The info will be organized in the most appropriate way possible. This information will be updated by a pointer passed into it.

Dependencies

- Access to the following:
    - Player via pointer
- Accessed by the following:
    - Gameplay State

Return	**Name**	**Parameters**	**Description**
bool	Initialize	void	Initializes the data. Returns whether or not the initialize was successful.
void	Release	void	Cleans up what needs to be cleaned.
void	Render	void	Gets the current state of the controller
void	Update	float _elapsedTime, CPlayer* _player	Updates the hud according to player data.

Module Author(s)

Evan Wright

Enemy

Derived from Entity. This class will take care of holding relative information for each enemy object. This class is responsible for making decisions for its entity such as updating its position based on its velocity and its remaining health points based on the amount of damage the object has received in game.

Dependencies

- Access to the following:
    - See Entity
- Accessed by the following:
    - See Entity

Return	Name	Parameters	Description
int	GetType	void	Returns the object's type.
void	Initialize	void	Initializes the variables.
void	Release	void	Cleans up memory.
void	Render	void	Draws the art assets for the enemies.
void	Update	float fElapsedTime, CEnemy enemy	updates the enemy objects
vec3f	GetPosition	void	returns the the position of the enemy
int	GetDamage	void	returns the damage taken
void	SetDamage	int damage	sets the damage on the enemy

void	SetPosition	vec3f position	sets the position of the enemy

Module Author(s)

Corey Morehead

Player

Derived from Entity. This module will contain all relevant information for the player character (health, ammo, lives..). It will also be responsible for updating its actions and positions based on information gathered from the Input module.

Dependencies

- Access to the following
    - Input
    - Entity
    - Projectile
- Accessed by the following
    - Collision Library
    - Object Factory
    - Object Manager
    - State Machine

Return	Name	Parameters	Description
int	GetType	void	Returns the object's type.
bool	Initialize	void	Initializes an object. Returns whether or not the initialize was

			successful.
void	Release	void	Cleans up what needs to be cleaned.
void	Update	float _elapsedTime	Updates the Player.
void	Input	void	Handles all Player input.
int	GetAmmoType	void	Returns the type of Ammo.
void	SetAmmoType	void	Sets the type of ammo.
int	GetHealth	void	Gets the current health amount

Module Author(s)

Joshua Villareal

Projectile

Derived from Entity. This is the base class for the plasma balls, mortars, shotgun, and beam. There is a list of projectiles maintained inside Object Manager. This list is separated into two lists of active and inactive projectiles that will be checked accordingly for collision.

Dependencies

- Access to the following
    - Asset Manager
    - Timer
    - Entity

- Accessed by the following
    - Collision Library
    - Object Factory

Return	Name	Parameters	Description
vec3f	GetVelocity	void	Returns the projectiles current velocity.
Void	SetVelocity	vec3f vel	Sets the bullets velocity to the input.
Int	GetType	void	Returns the bullets type as an int ID.
Void	SetType	int Type	Sets the bullets type ID to the parameter.
Void	Release	void	Cleans up necessary memory.
Void	Update	float DeltaTime	Updates bullets properties such as lifetime, velocity, and position based on in game events and time.
int	GetDamage	void	Returns the amount of damage the bullet causes.
Void	SetDamage	int Damage	Sets the bullets damage to the parameter.

Sub-Classes: PlasmaBall,Mortars,Shotgun,Beam.

Module Author

Tom Stefl

VFX Manager

Handles all visual effects in the game. On startup allocates a pool of memory to use for all emitters in the game. Uses particles and meshes to create unique and appealing effects for the game, such as; explosions, sparks, gaining power-ups, etc. When an emitter is finished it's memory is recycled back into the VFXs memory pool for use when another emitter is needed.

Dependencies

- Access to the following:
    - Emitter
- Accessed by the following:
    - Renderer
    - Entity

Return	Name	Parameters	Description
bool	Initialize	void	Initializes the data. Returns whether or not the initialize was successful.
void	Release	void	Cleans up what needs to be cleaned.
void	Update	float _elapsedTime	Updates the current active VFX.

void	Emitter	enum::VFX_TYPE _type, Vec3f _position	Starts a VFX with the sent in type using the emitter.

Module Author(s)

Evan Wright

Emitter (Submodule of VFX Manager)

This will be responsible for updating the point sprites and mesh particles in the game. On startup allocates a pool of memory to use for all particles in the emitter. The memory is recycled when a particle dies, back into the emitters memory pool, and reused when another particle is activated. This is also what will be used to render the particles, through the Renderer.

Dependencies

- Access to the following:
    - Renderer
- Accessed by the following:
    - VFX Manager

Return	Name	Parameters	Description
void	Update	float _elapsedTime	Allocates static particles when necessary and updates position, velocities, rotation, and life times of all particles already allocated.

void	Render	void	Renders the currently active and visible particles, using the Renderer.
void	AllocateParticles	int _numParticles	Grabs statically allocated particles from the array and initializes them.
void	LoadXML	string _fileName	Loads the XML file sent in and re-initializes the emitter to be the loaded in settings.

Module Author(s)

Evan Wright

Sound Manager

This system sets up the wWise sound engine and allows the user to play and stop sounds in 3D. The sound manager does not require access to any other modules. It will only be accessed by the State System, for sounds and music on menus, and by the Entities module for in game sounds (Player, Enemy, Music, and SFX sounds.)

· **Dependencies**

- Access to the following
    - wWise
- Accessed by the following
    - State System
    - Entities

Return	Name	Parameters	Description

bool	InitSound	HWND hWnd //main window handle	Initializes the components of wWise interface and all accompanying codecs needed to play sounds.
Void	Shutdown	Void	Stops all sounds from playing and shuts down wWise along with all initialized codecs.
Void	LoadSoundBank	Void	Used to load the list of sounds in wWise so sounds can be played.
Void	UnloadSound	Int SoundID //The ID of the sound to unload	Will remove the sound from the list of currently loaded sounds and free up the sound ID
Void	PlaySound	Int SoundID //The ID of the sound to play	Will play the sound based off the unique sound ID passed in
Void	PauseSound	Int SoundID //The ID of the sound to pause	Will pause the sound based off the unique sound ID passed in
Void	ResumeSound	Int SoundID //The ID of the sound to resume playing	Will resume a paused sound based off the unique sound ID passed in
Void	StopSound	Int SoundID //The ID of the sound to stop playing	Will completely stop a sound based off the unique sound ID passed in

Void	SetVolume	AkRtpcID setID
		// ID for the volume
		Float fVolume
		// the volume for the sound
			Used to set the volume levels for Rtpc WWise objects

Module Author(s)

Alexander Garcia, Corey Morehead

Gameplay State

Large state contained in State Machine. Holds a majority of the managers and uses them to manage the process of gameplay. This will use the Object Manager to update the states objects. It will create a render list to give to the Renderer.

Dependencies

- Access to the following
    - State Machine
    - Renderer
    - HUD
    - Player
    - AI Manager

- Access by the following
    - State Machine

Pause State

This class will stop the gameplay update, and show a menu to resume play, how to play or exit back to the main menu. It is pushed on top of Gameplay state on the state stack.

Dependencies

- Access to the following
    - State Machine
    - Sound Manager
    - Renderer
- Access by the following
    - State Machine

Options State

This class will have give the player options to change the resolution, volume.

Dependencies

- Access to the following
    - State Machine
    - Sound Manager
    - Renderer
- Access by the following
    - State Machine

Main Menu State

This class will have give the player options to change the state from options, game, credits, exit.

Dependencies

- Access to the following
    - State Machine
    - Sound Manager
    - Renderer
- Access by the following
    - State Machine

Intro State

This class will show the player the splash screens before the main menu.

Dependencies

- Access to the following
    - Main Menu State
    - Sound Manager
    - Renderer
- Access by the following
    - State Machine

Return	Name	Parameters	Description

void	Initialize	void	initializes variables
void	Release	void	shutdown the class safely, changes to new state.
void	Update	float _elapsedTime	updates objects
void	Render	void	draws the assets
void	Input	void	receives players input

Module Author(s)

Corey Morehead

Input Manager

Input supports Keyboard and Mouse or XBOX Gamepad and is contained within the State Machine. This class is a wrapper for the SGD Input Wrapper which encapsulates all key/button checks. At the beginning of Game Main's input method Read Devices will be called, which will update each registered player's bit flag. Keyboard and Gamepad 0, (If present,) bit flags will always be updated so that they can be read during menus.

Dependencies

- Access to the following:
    - Game Main
    - Player
    - OptionsState
    - PauseMenu
    - PlayGameState
    - MainMenu

- IntroState
- CharacterSelectState
- CollisionLibrary

- Access by the following:
  - Windows Form

· Methods

Return	Name	Parameters	Description
void	Read Devices	void	Encapsulates all the key/button checks for each active player and fills out a bit flag for use
int	GetInput	void	Accessor that returns the bitflag
int	PollDevice	void	Checks what button was pressed
Input*	GetInstance	void	Accessing the singleton

- Risks
    - Most laptops can’t read 4 keys at once

Timer

This class calculates time using the query performance counter/frequency. This class can also be used as a stopwatch timer, allowing the user to get accurate time. It includes windows.h and will be passed to all necessary classes via their update functions. This will mainly be used in State system for Update function’s elapsed time. It is also used to calculate frames per second.

It is held in the Windows Form.

· **Dependencies**

- Access to the following
    - N/A
- Accessed by the following
    - State Machine

Return	Name	Parameters	Description
void	Init	HWND _hWnd //main window handle	Initializes the timer class
Void	Shutdown	void	Cleans up and Shuts down the Timer class
Void	Update	void	This function does the entire math involved with time calculation.
Void	Start	void	Starts the timer from when this function is called.

Void	Stop	void	Stops the timer from when this function is called.
Void	Reset	void	Resets the timer from when this function is called.
float	GetFPS	void	Returns the number of frames the game has updated this second.
float	GetElapsedTime	void	Returns the time between last update to this point.

Object Manager

This module holds all the currently active and inactive objects given to it from the factory. The Object Manager will call each active entities update function passing along the Timer's delta time. It will also call their respective renders. It will also use the Collosion library to check

Dependencies

- Access To the following
    - Renderer
    - Object
    - Collision
    - Entity
    - Object Factory
- Accessed by the following
    - GameState

- Object Factory

Return	Name	Parameters	Description
void	AddObject	IEntity* _entity	Adds objects from the factory
void	RemoveObject	IEntity* _entity	Finds and removes the specified object from the list and adds them back to the object factory.
void	UpdateObjects	float _elapsedTime	Loops through the object list, calling the updates on the object.
void	RemoveAllObjects	void	Removes the objects from the list and adds them back into the object factory.
void	CheckCollisions	void	Goes through the object list calling the collision library's CheckCollision
void	RenderObjects	void	Loops through the object list added them to the render context.

Collision Library

This module handles the math for detecting collision in multiple forms. The object manager will iterate through all the objects and try all pairs of objects in the CheckCollision function, passing

in two entities at a time. The library will then detect which collision object types those hold, and perform the appropriate collision check on those objects. The library handles moving spheres and capsules, and will handle all reactionary tasks, whether that means dealing damage to the entities, moving their positions to correct tunneling, or setting them to dead instead of alive.

· **Dependencies**:

- Access to the following:
  - Entities
  - Collision Entities
- Accessed by the following:
  - Object Manager

Return	Name	Parameters	Description
Void	CheckCollision	CEntity* _lhs, CEntity* _rhs	The generic function that the object manager will call, passing in objects. This will determine which type of bounding volume the objects are and call the appropriate collision function. If collision is detected, the function will call both objects' reaction functions.

Module Author(s)

Alex Garcia

Collision Entity (SubModule of Collision Library)

This module is the base object for all collision volumes that the Entities use for collision detection. The classes that will inherit from the collision volumes will be spheres (for enemies, the player, and most ammo), AABBs(for environmental objects),OOBBs(for crate pickups), and capsules(Laser and other long shaped objects), which are the types of bounding volumes the game objects can use. These child classes will have personalized data members based on which type of bounding volume they represent. Entities will only have access to the Collision Object class, which will be a data member inside each entity. The collision library will take collision objects into their functions, then downcast into their appropriate child class, and handle the collision reactions.

· **Dependencies**:

- Access to the following:
  - N/A
- Accessed by the following:
  - Entities
  - Collision Library

Return	Name	Parameters	Description
CollisionEntity*	CreateColEnt	enum _type, …	Pseudo- constructor making an object of the passed in type, and then setting its primary variables. The function definition will have guidelines for properly passing in the information of the bounding volume.
Enum	GetType	void	Returns the type of the object.

Module Author(s)

Alex Garcia

Renderer

The renderer uses OpenGL to draw the scene, the HUD, VFX, particles, lighting, post-processes, etc. to the screen. It will be contained within the Windows Form. This module initializes all OpenGL objects. It is a forward renderer with deferred lighting and supports diffuse mapping, normal mapping, and specular mapping. Other objects, including entities, VFX, and post-process effects, get passed into the renderer's Add methods by their respective owners. The renderer adds their geometry to its render lists each frame, sorting by render state, then sorting spatially. It gathers that information, along with the relevant textures and shaders, from the objects' internal pointers. When all of the relevant objects for a particular frame have been added to each list, the Draw methods are called, rendering all of the objects to the screen. Each time an object in a render list is reached that has different state information from the previous object, ChangeRenderState is called to update the renderer's state information. The goal of sorting the objects is to prevent this from happening any more frequently than absolutely necessary.

The CPostEffect class is defined in within the Renderer module, and is used for adding full-screen post-process effects. The CUIElement class is a sub-module of HUD and is used to pass information from the HUD to the renderer for drawing. The CRenderState class is a sub-module of the renderer and is used by any object that needs to be drawn to pass shader and texture information to the renderer.

Dependencies

- Access to the following
    - Camera
    - Windows Form
- Accessed by the following
    - Entity
    - VFX Manager
    - HUD

Return	Name	Parameters	Description
bool	Initialize	void	Sets up the renderer.
void	Shutdown	void	Releases all dynamic memory held by the renderer.
void	AddToRenderList	CEntity* _entity	Adds the passed-in entity pointer to the render list, sorted first by render state and then by spatial position.
void	AddVFXToRenderList	CVFX* _pEffect	Adds the passed-in effect pointer to the render list, sorted the same as if it were an entity.
void	ClearRenderList	void	Empties the contents from the render list.
void	DrawRenderList	void	Renders the contents of the render list.
void	ChangeRenderState	CRenderState* _pRenderState	Updates the render state, based on the state pointer that is passed in from an object. This method is called by DrawEntities and DrawEffects when the next entity in the render list uses different state.
void	PopulateLightList	void	Calls Collision on each active light in the scene to check it for visibility, then adds the visible

			active lights to the light list.
void	ClearLightList	void	Empties the contents from the light list.
void	DoShading	void	Loops through the light list and applies shading to the scene.
void	DoPostEffect	CPostEffect* _pPostEffect	Applies a post-process effect to the scene.
void	AddToUIList	CUIElement* _pUIElement	Adds a UI element to the UI element list.
void	ClearUIList	void	Empties the contents from the UI element list.
void	DrawUI	void	Renders all of the UI elements in the list.
void	Update	float _dT	Updates the renderer based on the passed-in elapsed time.

Module Author(s)

Robert Pasekoff

Camera

This module will handle the camera's frustum position/orientation based on the player's input. The player's position and orientation will be manipulated by the camera's frustum data. Renderer will use this to decide whether or not to render the current active objects.

- **Dependencies**:
  - Access to the following:
    - N/A
  - Accessed by the following
    - Renderer
    - Entity

Return	Name	Parameters	Description
void	ChangeYaw	float _change	Changes the yaw of the camera by _change degrees
Void	ChangePitch	float _change	Changes the pitch of the camera by _change degrees
Void	ChangeRoll	float _change	Changes the roll of the camera by _change degrees
Matrix4x4f*	GetViewMatrix	void	Returns the view matrix based on the frustum data
Matrix4x4f*	GetProjMatrix	void	Returns the projection matrix based on the frustum data

Module Author

Alex Garcia

State Machine

The module holds all of the currently registered states and call the respective states Update and Render functions for the states on the state stack.

Dependencies

- Access To the following
    - State
- Accessed by the following
    - Game Main
    - State

Return	Name	Parameters	Description
void	PushSate	const char* _name	Adds a state to the top of the state stack.
void	PopState	void	Removes the top of the state stack.
void	Update	float _elapsedTime	Updates the top of the state stack.
void	Render	void	Calls render on all the states in the state stack.
void	RegisterState	const char* _name, State _state	Adds a new state into the state map.

void	ChangeState	const char* _name	Removes all the states in the state stack, and pushes the new state onto it.

A.I Manager

The module holds the Slotting System for the AI and designates whether the slots are vacant or not. The module also calls update on the slots to designate what the slots should do next. the Slots will handle when the enemies can attack and at what turn they will be at depending on what slot they occupy.

Dependencies

- Access To the following
    - Slot
- Accessed by the following
    - Enemy

Return	Name	Parameters	Description
void	PushSlot	Slot* _pSlot	Adds non-vacant slots to the non vacant list
void	PopSlot	State*_pState	Finds and removes the specified slot from the list and adds them back to the vacant slot list
void	UpdateSlots	float _fElapsedTime	Loops through the slot list, calling the updates

			on the slots.
void	ClearSlots	void	Removes the objects in all slots from the list and adds the slots back into the vacant slot list.
void	CheckSlots	Slot* _pSlot	Goes through the object list checking to see if the slot is vacant
void	OccupySlot	void	sets the Slot to non-vacant
void	RenderSlots	void	Loops through the slot list and renders them. FOR DEBUG USE ONLY

Module Author(s)

Joshua Villarreal

Slot (Sub of A.I. Manager)

The module is the structure for the A.I Manager which controls the order of attacking for enemies and idling.

Dependencies

- Access to the following
    - Timer
- Accessed by the following
    - A.I Manager

Return	Name	Parameters	Description
bool	IsVacant	void	returns true if the current slot is vacant
void	InitSlot	int _numSlots	Initializes the number of slots passed in and splits them up by type: IE: idle,attacking,rangedidle, rangeattacking
vecf	GetPosition	void	returns the position of the slot
void	Update	float _fElapsedTime	checks to see if the timer on the slot has finished to see if the enemy can advance to the next slot or go back to idling
void	Clear	void	Clears the slot

Module Author(s)

Joshua Villarreal

Animation System

This module controls the various animations of the game. The actual data for each animation is held by the Asset Manager while the object itself is in the object manager. The processor uses these to find the current frame for the relevant animation. While the processor has no data members itself, its functions are static and are accessible through the Resolution Operator (::).

Dependencies

- Access To the following
    - Asset Manager
    - Object Manager
- Accessed by the following
    - Entities

Return	Name	Parameters	Description
void	LoadMeshFromFile	const char* _FileName	Imports the binary Mesh file.
void	LoadAnimFromFile	const char* _FileName	Imports the binary animation file.
void	AddTime	float _Time	Adds the elapsed time from the last frame to the m_CurTime.
void	Process	void	Interpolates between the animations key-frames based on the Interpolator's m_CurTime.

Necessary Structures / Classes:

struct KeyFrame

Transform* pKeyFrames;

unsigned int unCurrFrame;

unsigned int unNumFrames;

class Mesh

Vec3f m_Verts;

Vec3f m_Norms;

vector<unsigned int> m_Indicies;

vector<vector<float>> m_Weights;

Accessors for each.

class Model

vector<Mesh> m_Meshes;

class Animation

float m_Duration;

vector< KeyFrame > m_Keyframes

Assessors for both members.

class Interpolator

const Animation *m_Animation

Keyframe m_CurKeyFrame;

float m_CurTime;

float m_AngleTotal

float m_Angle;

Accessor and Setter for the m_CurKeyFrame.

Accessor for Animation.

Asset Manager

This module will be responsible for loading and containing texture and shaders assets as well as animations and meshes. It will contain an array of data for every game object where assets must be loaded in. All of the assets are loaded at the beginning of the game in order to cut down or even eliminate long loading times during gameplay. Accessors will be available to access all the elements in an array of object data and will use an int ID to get the correct index for data.

Dependencies

- Access To the following
    - Renderer
    - Accessed by the following
    - VFX Manager
    - HUD

Return	Name	Parameters	Description
unsigned int (index)	LoadShader	char* _FileName, unsigned int _shader	Loads a shader from a file at the specified index.
unsigned int (index)	LoadTexture	char* _FileName, enum_wrapMode, enum _filter	Loads the texture using the file name, wrap mode, and filter type.
unsigned int (index)	LoadAnimation	char* _FileName	Loads the animation data from an exported maya file.
unsigned int (index)	LoadMesh	char* _FileName	Loads in an exported mesh data from maya file.
unsigned int (OpenGL handle)	GetTexture	unsigned int index	Returns an OpenGL handle to based on the index passed in.
Animation *	GetAnimation	unsigned int (index)	Returns an animation pointer based on the index passed in.
CAssetManager*	GetInstance	void	Used to get access to data within the manager. Singleton.
void	Shutdown	void	This function is used to unload all of the data loaded into the manage and free memory upon

			the shutdown.
void	LoadMesh	char* _FileName	Loads the mesh with the appropriate values.
void	LoadAnimation	char* _FileName	Loads the animation with the correct keyframes.

Necessary Structures:

ObjectData

vector<Animation> vAnims;

unsigned int unNumVerts;

unsigned int unNumTris;

unsigned int hVertBuff;

unsigned int hIndexBuff;

unsigned int hShader;

unsigned int hTexture;

Windows Form

This is the most external class. Windows form is responsible for startup initializations of the state machine as well as some of the more globally necessary modules such as Input and the Sound Manager and Timer. During this initialization, Asset Manager will then front load all of the assets from the libraries.

Dependencies

- Access To the following
    - State Machine
    - Renderer
    - Input
    - Timer

    - Sound Manager
- Accessed by the following
    - N/A

Return	Name	Parameters	Description
void	Initialize	HInstance _hWnd, int _width, _height	Initializes the class members.
void	Release	void	Deletes all of the dynamic memory that is used.
void	Run	void	Calls the RenderFrame and UpdateFrame.
void	RenderFrame	void	Calls the State Machine's render function.
void	UpdateFrame	void	Calls the State Machine's update function.
void	WndProc	HWND _hWnd, UINT _msg, WPARAM _wParam, LPARAM _lParam	Initializes the window.
int	GetHeight	void	Returns the height of the window.
int	GetWidth	void	Returns the width of the window.

Script Manager

The Script Manager is Responsible for binding to LUA to pass data back and forth between the main modules and the scripts. It also registers any functions that are needed for LUA to call. Scripting will be available to creating Waves of enemies, AI behaviors, and visual effects

Dependencies

- Access To the following
- Accessed by the following
    - AI Manager

Return	Name	Parameters	Description
void	Initialize	void	Initializes the class members.
void	Execute	void* _ptr, const char* name	Executes the script that is passed into it.
lua_State*	GetLuaState	void	Returns the lua state.

Sample Script

--Movement Behavior

```
info = {};


function Init()


	table.getn(arg)

	-- arg 0-2 holds players position x,y,z  arg 3 is a bool for if the player is in range

	--arg 4-6 is the enemies own position arg 7 is a function passed in and arg 8 is a bool to see if the behavior is switching from movement to slotting

	info.playerposition = {arg[0],arg[1],arg[2]};

	info.inrange			= arg[3]

	info.myposition	= { arg[4],arg[5],arg[6]};

	info.alighnmentfuc = arg[7];

	info.slotting			=arg[8];

	Run();


end

function Run()

--based on the information passed into the info table is whether the position of this enemy

--should move or not and also if they aren't about to go into an attack slot


if info.inrange == true && info.slotting == false then

	if info.playerposition[0] < info.myposition[0] then

		info.myposition[0] = info.myposition[0]-1;

	else
```

```
                info.myposition[0] = info.myposition[0]+1;
        end
        if info.playerposition[3] < info.myposition[3] then
                info.myposition[3] = info.myposition[3]-1;
        else
                info.myposition[3] = info.myposition[3]+1;
        end

        info.alighnmentfunc(); -- this is the function that was sent in to the behavior to clamp the enemy into the proper positioning
        return info;
else
        data = {};
        if info.inrage == true then
        data.inrange = true;
        else
        data.inrange = false;

        if info.slotting == true then
        data.slotting = true;
        else
        data.slotting = false;
        return data -- if the player is out of range or about to go into an attack slot then return to the script manager and go into the proper script

end

end
```

Lua Module Breakdown

Move

Lua based entity that is called from the C++ side if a decision for the enemy to change direction in movement and/or speed if the decision from C++ side dictates that a change in the movement has occurred.

Return	Name	Parameters	Description
LUA TABLE	Execute	LUA TABLE	Based on the information that the table contains will determine whether the Direction of Movement is changing or if the speed of the movement increased or decreased.

Module Author(s)

Joshua Villareal

Small Attack

Lua based entity that is called from the C++ side if a decision is made to make a small attack, the melee attack. Based on the C++ side, the decision is made to change or keep the current state of attack, holds the amount of damage, and speed of the attack.

Return	Name	Parameters	Description
LUA TABLE	Execute	LUA TABLE	Based on the information that the table contains will determine whether the state of the attack is new or will stay in the same state, be able to see how much damage the attack makes, and also how fast the attack should be

Module Author(s)

Joshua Villareal

Medium Attack

Lua based entity that is called from the C++ side if a decision is made to make a small attack, the melee attack. Based on the C++ side, the decision is made to change or keep the current state of attack, holds the amount of damage, and speed of the attack.

Return	Name	Parameters	Description
LUA TABLE	Execute	LUA TABLE	Based on the information that the table contains will determine whether the state of the attack is new or will stay in the same state, be able to see how much damage the attack makes, and also how fast the attack should be

Module Author(s)

Joshua Villareal

Large Attack

Lua based entity that is called from the C++ side if a decision is made to make a small attack, the melee attack. Based on the C++ side, the decision is made to change or keep the current state of attack, holds the amount of damage, and speed of the attack.

Return	Name	Parameters	Description
LUA TABLE	Execute	LUA TABLE	Based on the information that the table contains will determine whether the state of the attack is new or will stay in the same state, be able to see how much damage the attack makes, and also how fast the attack should be

Module Author(s)

Joshua Villareal

Integration Plan

What Method of Source control software are you going to use?

The Source control software method is going to be perforce.

What marks a module or file as complete for integration?

The file or module is going to be marked as complete after the author feels it is has met the parameters required for completion. It will then be reviewed by either Tech Lead or Q/A Lead prior to integration. This process carried out by the Leads will screen for class A or B level bugs as these render the work unacceptable for program integration.

A,B, and C Class Level Bugs are defined further down in testing plans.

What are the steps of integration?

After the author has successfully integrated their module in their own build, Any Issues the module has will be addressed on the author's build so that it is executing in a similar way to the main build. Bugs that will be acceptable are low level B classified bugs and C bugs but they must be logged into Hansoft's bug log. Tech Lead will have also reviewed the module before integration.

What if a module breaks?

If a module has a Break [j1] then the author will have to locate where and what is causing the module to break and what, if any, other systems it may have broken. If there is a problem with a module not created by the author than the original creator shall handle the break in the module if they do not grant authority to change code in that module. If both modules are owned by the author, the author may decide to proceed however they deem fits the situation. if the conflicting module is the one chosen to change instead of the current one up for integration than the author must send a notification to the rest of the team that a change will occur and check for any conflicts in doing so.

Break – A Class A bug or code that cannot compile

What happens when a bug is found?

The QA lead will be notified and the bug will be logged through Hansoft. Bugs are categorized and defined below as well as under the Integration Planning section of the document:

Class A Bug- Any bug or error in the program that disqualifies a module from being integration ready. These are bugs that cause a system to fail, the program to crash or prevents the program from compiling at all. This also includes any bug that causes the game to become unplayable. If a class A bug has made it into the main build undetected by the author, Tech, or Q/A Leads then the code's author will and must be notified. Class A bugs are always of the highest priority and must be rectified immediately. In order of delegation, the codes author will need to address the bug first. If unavailable, Q/A Lead or members with temporarily suspendable tasks will be set to correct bugs of this level.

Class B Bug – Any bug that does not necessarily cause the build to break or prevent it from compiling, but may cause unwanted behavior within the game such as frame skipping or a faulty mechanic. If a module causes a Class B bug in a different module than that bug must be resolved before qualifying for integration. In the occasion that a class B makes it through to the main build and later affects another module, that bug must be fixed within a 24 hours time. After 24 hours, the bug will be reported to the QA lead for a decision to be made.

Class C Bug- Any cosmetic bugs that are temporarily acceptable in the main build. These must be reported to the QA Lead. The author will handle the bug if the cause is within the author's own module. Otherwise the delegation for the task of fixing the bug is entirely the discretion of the Q/A Lead.

When will a rollback occur?

If a Class A bug is introduced into the main build and not resolved within the hour, the QA Lead must be informed in which the he will have 5 hours to fix or remove the bug from the main build.

After the time limit has passed the QA Lead will inform the Team Lead of the situation and a rollback will take place. The QA Lead will use perforce to instantiate a rollback or revert back to a known working version. The other case for a rollback will be if a B class bug creates problems with game play. The same procedures will take place as before to correct the problem.

Will there backups and when?

Labels will be created each day at the end of the day after the final module integration if any that day.

How many Files can be checked out by one person and can more than one person check out the same file?

As many files as that one person needs but no more than one person per file checkout.

When will files be checked back in?

Files must be checked in at the end of the day before the build backup. Exceptions to this are modules with class A bug and/or unknown class that cause breaks or if the module is still being created.

Unknown Class – A bug that cannot be located that either is or has potential to break the build

If there is any ambiguity, questions should be addressed to the Tech Lead, if the Tech Lead cannot resolve the question then the question will be addressed to the Team Lead.

Testing Plans

Corey Morehead is the QA Lead and will be in charge of maintaining the bug database for our team. The person who finds the bug is responsible for logging the bug into Hansoft. Individual bugs will be assigned to the team member who was in charge of that section of the project. In cases where that team member is currently working on another bug, the bug will be assigned to a member who is not working on an assignment or bug. If a team member cannot figure out how to fix a bug, another member, who is not currently working on a bug, will be assigned to help until the bug is fixed.

Game bugs will be categorized in three ways.

· Category A bugs will be detrimental to the game, most likely causing the game to crash or rendering it unplayable.

· Category B bugs will be related to functionality and certain less important gameplay aspects.

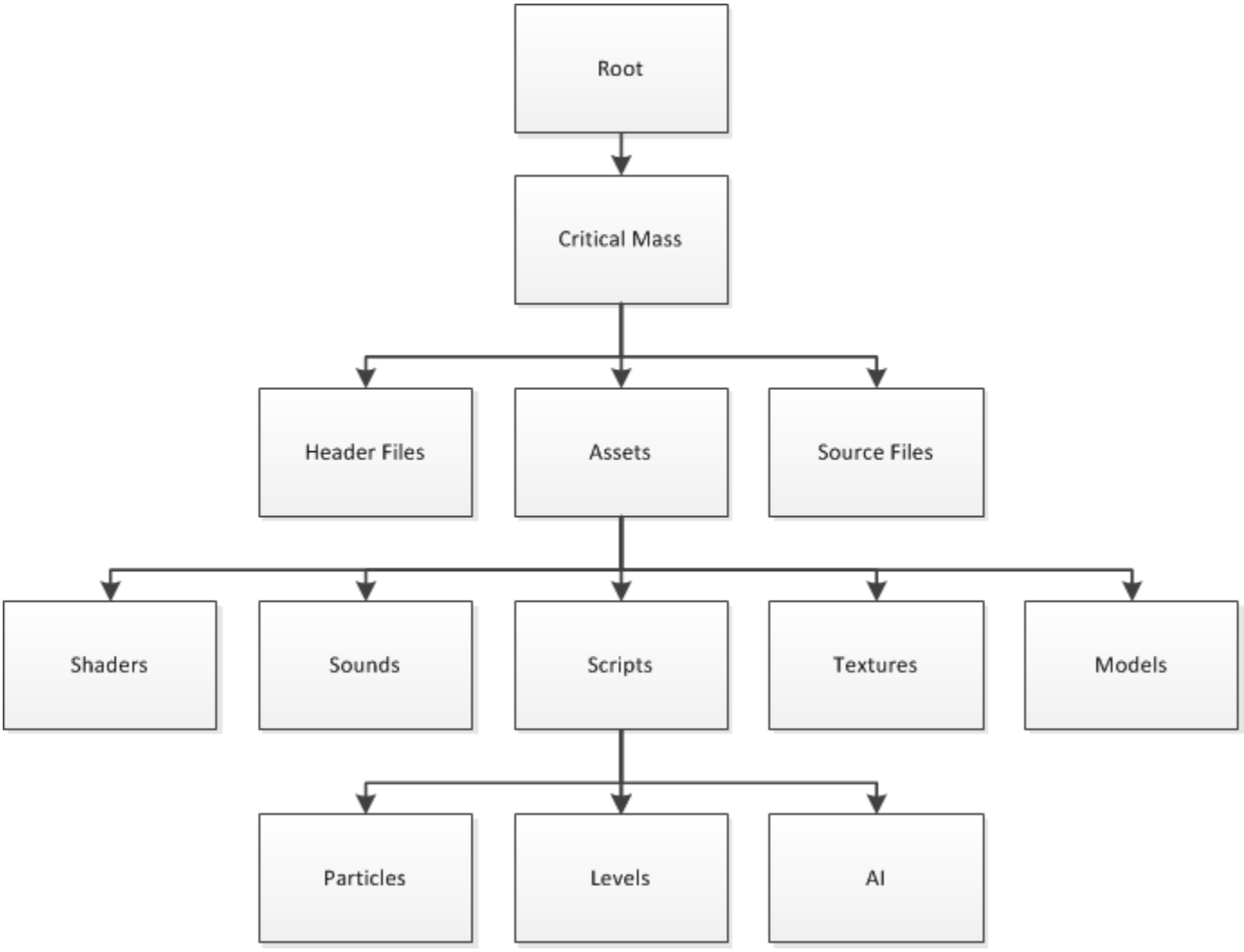
· Category C bugs will be related to cosmetic and non crucial aspects of the game.

The bug database will be updated daily and will be assigned at the end of the day to be worked on at home. If a bug is not fixed by the next meet, the team member will continue to work on the bug until finished. When balancing both normal tasks and bug fixing, team members will prioritize their next task in this order:

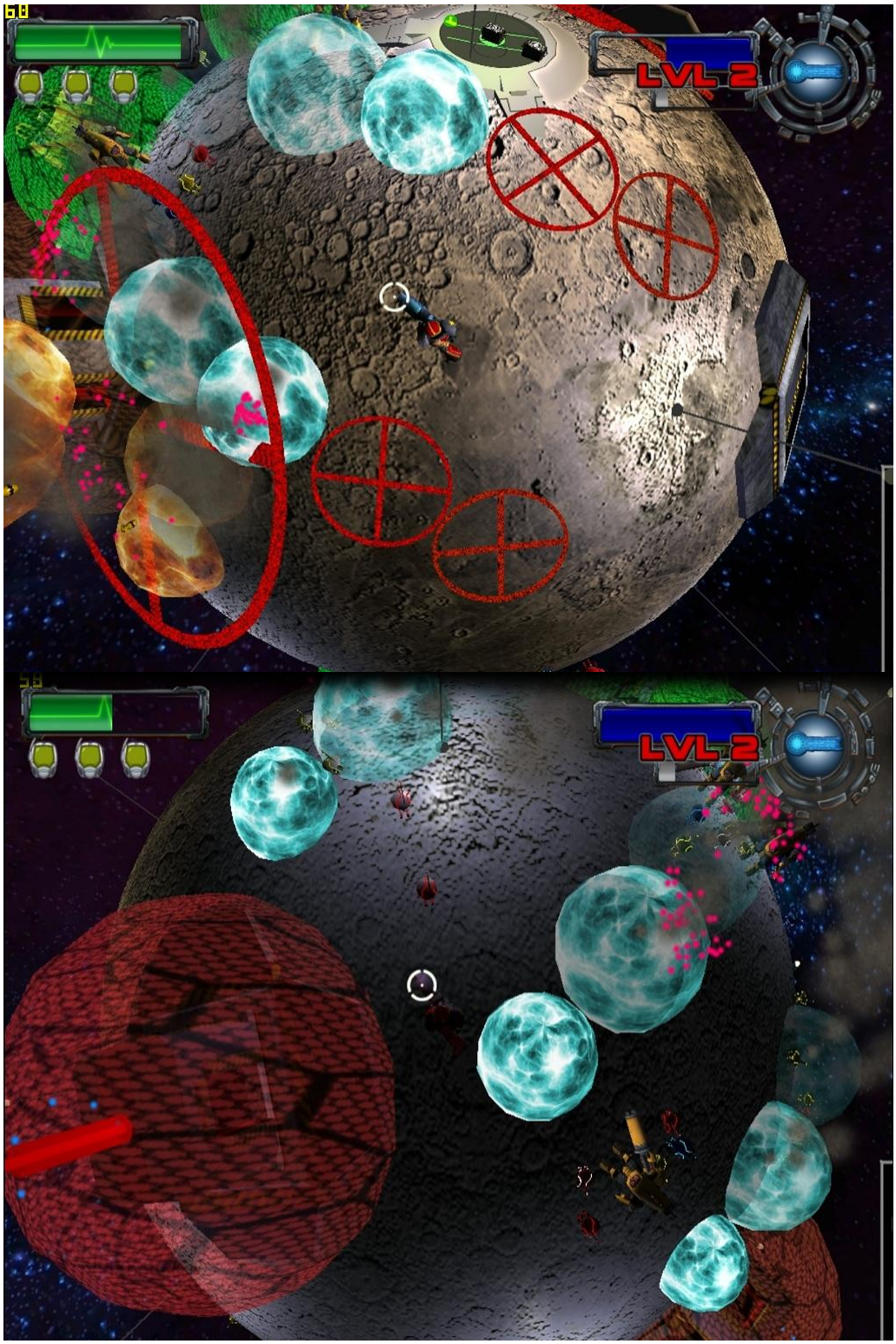
1. category A bugs
2. category A tasks
3. category B bugs
4. category B tasks
5. category C tasks
6. category C bugs

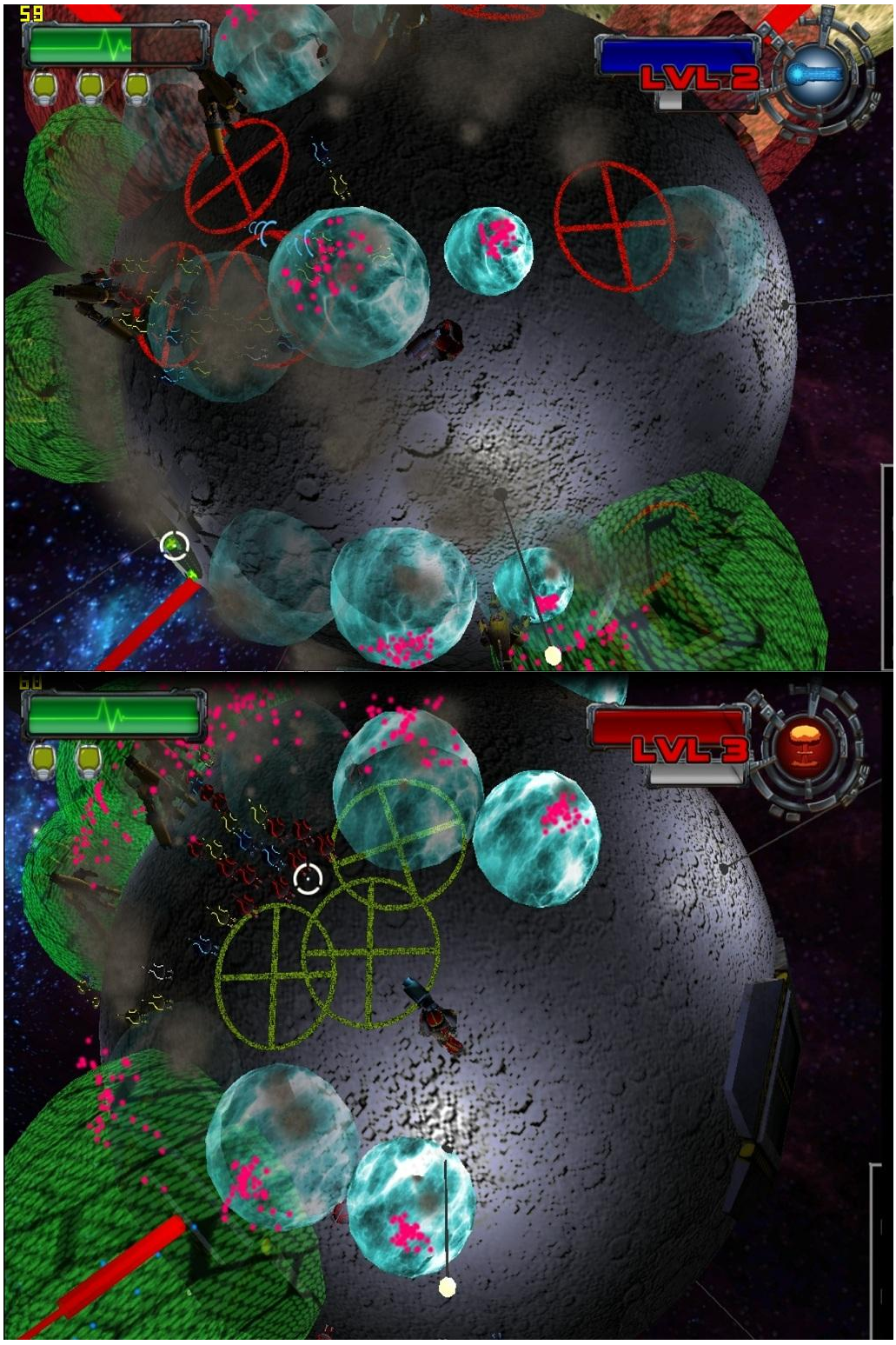
This bug database will be represented by the Quality Assurance section of Hansoft. In addition to testing done by the Gaming Project Studio staff, there will be 3-4 designated time slots, happening once a week, where outside testing will occur in a controlled environment. People willing to test will be given a survey asking overall satisfaction and for any bugs found while testing. Once the testing group has ended, the QA lead will review all the surveys and add bugs from those surveys to the second bug list. The team will also meet once a week on a different day and spend 30 minutes testing the game in areas they did not manage directly, and add those bugs to the Quality Assurance section of Hansoft.

Game Folder Hierarchy



Screenshots




59
LVL 2
LVL 3

Code Samples

Ian Alcid

```
if(_player->GetGun().GetConsumed() != EMPTY)
{
	float curr = (float)_player->GetGun().GetExperience();
	float need = (float)_player->GetGun().GetNeededExperience();
	float tempRatiolvl = curr / need;

	if( _player->GetGun().GetWeaponLevel() >= 3 )
		m_HUDElements[HUD_WEAPON_LVLBAR].GetRect().SetWidth(214.0f);
	else
		m_HUDElements[HUD_WEAPON_LVLBAR].GetRect().SetWidth(tempRatiolvl
* 214.0f);
	m_HUDElements[HUD_WEAPON_LVLBAR].BuildGeometry(m_Renderer-
	>GetScreenWidth(),	m_Renderer->GetScreenHeight());
}
```

Ryan Cartier

```
/*****************************************************************************
* Ryan Cartier
*
*
*
* Summary: Shader used to speed up the animated meshes in the game by passing
* skinning matricies and the joint influences onto the GPU and doing the
* translations there instead of doing them on the CPU which would be
* exponentially slower.
*
*
*****************************************************************************/

uniform mat4 mvpMatrix;
uniform mat4 mSkin[30];

in vec4 vVertex;
in vec2 vTexCoord0;
in uvec4 nJointIndex;
in vec4 fJointWeight;

out vec2 vTex;
out vec4 vColor;

void main( void )
{
	vTex = vTexCoord0;
	vColor = vec4(0.0f, 0.0f, 0.0f, 1.0f);

	vec4 newPos;
	vec4 pos = vec4(vVertex.xyz, 1.0f);

	newPos = mSkin[nJointIndex[0]] * pos * fJointWeight.x;
	newPos += mSkin[nJointIndex[1]] * pos * fJointWeight.y;
	newPos += mSkin[nJointIndex[2]] * pos * fJointWeight.z;
	newPos += mSkin[nJointIndex[3]] * pos * fJointWeight.w;

	gl_Position = mvpMatrix * newPos;
}
```

Corey Morehead

```
//SHOOTING
static bool pressed = false;
if( _input.IsLeftClicked() ||
_input.GetController().GetControllerState().Gamepad.bRightTrigger )
{
	if (m_bWeaponBlue)
		m_bShoot = true;

	dynamic_cast<CPlayer*>(m_Player)->GetGun().Shoot(m_OF,
		(CPlayer*)m_Player, 0 );
	pressed = true;
}
else
	pressed = false;

if (((CPlayer*)m_Player)->GetGun().GetConsumed() != EMPTY)
{
	m_bPull = true;
}

if (((CPlayer*)m_Player)->GetGun().GetConsumed() == RED)
{
	m_bStep1 = true;
	m_AIManager.DestroySmall();
}
if (((CPlayer*)m_Player)->GetGun().GetConsumed() == YELLOW)
{
	m_bStep1 = true;
	m_AIManager.DestroySmall();
}
if (((CPlayer*)m_Player)->GetGun().GetConsumed() == BLUE)
{
	m_bStep1 = true;
	m_AIManager.DestroySmall();
}
if (m_bStep1)
{
	if (((CPlayer*)m_Player)->GetGun().GetConsumed() == EMPTY)
	{
		//check to see if any enemies are on screen.
		m_bStep2 = true;
		if (!m_bWeaponRed)
		{
			m_bWeaponRed = true;
			m_WeaponCounter++;
		}
		else if (!m_bWeaponYellow)
```

```
            {
                  m_bWeaponYellow = true;
                  m_WeaponCounter++;
            }
            else if (!m_bWeaponBlue)
            {
                  m_bWeaponBlue = true;
                  m_WeaponCounter++;
            }
      }
}
if (m_bStep2)
{
      if (m_AIManager.GetNumSmallEnemies() == 0 && m_bTab)
      {
            CMessageSystem::GetInstance()->SendMessage(new
                  CCreateEnemyMessage(m_DepotRot[0], ET_SMALLENEMY));
      }
      m_bStep1 = false;
      m_bStep2 = false;
}

if (((CPlayer*)m_Player)->GetGun().GetConsumed() == EMPTY)
{
      if (m_AIManager.GetNumSmallEnemies() == 0)
      {
            m_bStep2 = true;
      }
}
```

```
bool CRenderer::_CheckFBOStatus( GLenum _readOrWrite )
{
	GLenum result = glCheckFramebufferStatus( _readOrWrite );

	switch( result )
	{
	case GL_FRAMEBUFFER_UNDEFINED:
	cout << "Current FBO binding is 0, but no default frame buffer exists
.\n";
		return false;
	case GL_FRAMEBUFFER_COMPLETE:
		// OK
		break;
	case GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT:
	cout << "One of the buffers enabled for rendering is incomplete.\n";
		return false;
	case GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT:
		cout << "No buffers are attached to the FBO.\n";
		return false;
	case GL_FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER:
	cout << "One of the buffer attachments enabled for rendering does not
 have a buffer attached.\n";
		return false;
	case GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER:
	cout << "One of the buffer attachments enabled for reading does not h
ave a buffer attached.\n";
		return false;
	case GL_FRAMEBUFFER_UNSUPPORTED:
	cout << "The attempted combination of internal buffer formats is unsu
pported.\n";
		return false;
	case GL_FRAMEBUFFER_INCOMPLETE_MULTISAMPLE:
	cout << "The number of samples or the value for GL_TEXTURE_FIXED_SAMP
LE_LOCATIONS for all buffers does not match.\n";
		return false;
	case GL_FRAMEBUFFER_INCOMPLETE_LAYER_TARGETS:
	cout << "Not all color attachments are layered textures or bound to t
he same target.\n";
		return false;
	};
	return true;
}
```

Tom Stefl

```
m_RenderNode.SetPosition(
      ((CBaseEntity*)m_SourceEntity)->GetRenderNode().GetPosition());

Vec3f planetCenter = {0.0f,75.0f,0.0f};
Vec3f A;
glsSubtractVectors3(A,m_ClosestCratePos,planetCenter);

Vec3f N;
glsCopyVector3(N,((CBaseEntity*)m_SourceEntity)-
>GetRenderNode().GetUpVector() );

float AdotN = glsDotProduct3(A,N);
Vec3f B;
//Vec3f B = A - (AdotN) * N;
glsScaleVector3(N,AdotN);

glsSubtractVectors3(B,A,N);
```

Evan Wright

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Summary: Traverses its child nodes in order, from left to right. The Selector 
succeeds
             as soon as ANY child succeeds. The Selector fails only if ALL 
children fail.
             Same as Sequence, Selector saves "running" child and traverses 
it first on
             the next pass.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
Selector(unsigned int _length, ...)
      : length(_length), current(0)
{
      // Creates an array of nodes based on the sent in length
      behaviors = new BehaviorNode*[length];
      // Grab all the nodes being sent in.
      va_list args;
      va_start(args,_length);
      {
            // Adds them as a behavior in this selector
            for(unsigned int i = 0; i < length; ++i)
                  behaviors[i] = va_arg(args, BehaviorNode*);
      }
      va_end(args);
}
```