cortex COMMAND™

THE GUIDE

OR HOW TO NOT GET YOUR SELF BANNED

MADE BY THE USERS OF DATA REALMS FAN FORUMS FOR NEW MODDERS

CONTRIBUTORS

DAMAN (CORTEX WIKI)
PROJECTTHOR(CORTEX WIKI)
GRIF (LUA TUTORIALS AND CORTEX WIKI)
VAGYR(MAKER OF THIS DOCUMENT)
LORD TIM (TUTORIALS AND CORTEX WIKI)
ROON3 (WALKPATH TUTORIALS)

Table of context

Chapter 1

Introduction on cortex command

1.1) what is cortex command

Cortex Command is the primary project of Data Realms LLC. The game is still in development.

"In Cortex Command, you play as a prospector and explorer in a time where complete cybernetics and whole-body amputations are common practice. Your severed brain is able to control many different types of bodies remotely from its underground bunker: clones, robots, spaceships, defensive turrets, and so on.

A typical scenario starts with a building phase where you get to construct your own bunker complex from scratch. Then you need to mine precious gold from the deformable pixel terrain in order to buy more and better ships, soldiers, weapons, digging tools, and deployable defenses. Use these assets to defend your disembodied brain and destroy or bankrupt your opponent!

Control your team of remote bodies either directly or let the friendly AI do your bidding through real-time strategy elements built into the game. Play with up to four players in split screen -- 2 vs. 2 players, or all four cooperatively against the computer. Eventually, you can play the campaign missions together with friends..."

The game currently features two modes of play: Campaign, and Skirmish.

Skirmish mode supports up to 4 players, on 2 different teams, or 1 player against a single enemy. In a match versus the AI, the game is an endless survival mode. In multiplayer skirmish, the goal of the game is to destroy the opponent's brain.

Campaign mode, in the current version of the game, features a single-player tutorial mission. Later builds will have a much more complete story mode, with co-op gameplay, a planet-wide metagame, and multiple factions.

There are three customizable types of control: Keyboard, Mouse and Keyboard, and Gamepad.

In the future there are plans for a turn based meta game going on in the planetary view while the player builds individual mining bases to generate more income.

The campaign will be nonlinear. Your choice of faction will control tech unlockables, and there will be specific storyline missions.

But the real fun is the modding of this game.
You can easily add your own context or download other mods through the data realms foroum but more on than later.

1.2)Cortex Command history

Cortex Command development started sometime within 2000, by Daniel Tabar (a.k.a. Data), and a working alpha was released in 2004. Since then Cortex Command has gone through numerous changes and updates.As the game carried on he realized that he wasn't exactly the best artist and sought the help of Promster (a.k.a. Prometheus). Promster's graphical changes were implemented on the 27th of July, 2004 and gave Cortex Command the graphical style that it is now known for.In 2004, Data brought Prometheus onto the team, for art and creative help. In 2008, he contracted several members of the community to do work for the game. These contributors includes **capnbubs**, **TheLastBanana**, and **numgun**The project has been in development for seven years. It has been submitted to the Independent Games Festival four times the forth time it won two awards one for technical excellence and the other was the audience award, has a feature on the Great Games Experiment, and was recently featured in Play magazine.Also links to the older versions of cortex command can be found thanks to Grif.

Click on a link to begin download

Build 1
Build 2
Build 3
Build 4
Build 5
Build 6
Build 7
Build 8
Build 10
Build 11
Build 12
Build 13
Build 14
Build 15
Build 16
Build 17
Build 18
Build 19
Build 20
Build 21
Build 22

1.3)Cortex Command factions

Wildlife

The critters that inhabit the gold rich planet you're on. Unfortunately for them, your fighting and mining is killing them and their laying waste to their planet.

Crab

The most common native on the planet.

Mega Crab

A crab that defends the nest and protects it's smaller brethren from harm.

Jumper

A tiny little thing that just jumps around.

Brown Coats

A recently added race, made by Capnbubs. Although little data has been given about this new faction, it has been gathered that they are a heavyset faction, slow to move, but with impregnable armor. No weapons have been specifically introduced for them.

Browncoat Light

A fearsome super soldier!

Browncoat Heavy

A unit that when seen on the battlefield it instills fear in to all that see it,until they shot his helmet off because he has no head as too make it last longer in the field of battle!

White Bots

The Whitebots represent a somewhat elite faction. The current dropship and rocket belong to the Whitebot faction. They also have the brainmech. Although not shown in any game demonstrations to this point, they have been spotted in the opening cinematic in a conflict against the Coalition.

Aliens

Aliens, sure, you really can't tell what the brain is can you? You might be a Zxolophlox controlling those human bodies, because human bodies makes nice organic meat puppets. The opening cinematic briefly shows various alien lifeforms that may be integrated into the game.

Trade star

A giant space station currently orbiting the gold rich planet than the factions of cortex command fight for.
Trade Star is the company that transports their inventory from their shipping station. You are responsible for the safe return of their transportation vehicles. Any damage incured to their equipment will be taken out of your gold return.not many more are known about this faction and it is not known if they will play a more active role later on the game.

Available units

Brain Case

A small case that holds the brain of the player. The brain is the most valuable objective in the game as is what controls all the operations of your army.

Robot 1

Trade Star's standard robot soldier. Suffers from impact damage more than other units.

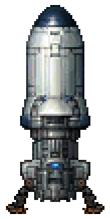
Robot 2

Trade Star's robot soldier redesigned to absorb more bullets,it still suffers from impact damage same as the standard robot but it has been reduced by the rebuild.

Brain Robot

Trade Star's standard robot fited with a brain enclosure for prospecting brains to go set up mining bases or make a quick escape if under seige.

Rocket Mk1

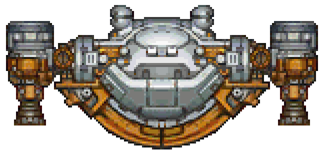


The cheapest Trade Star Rocket, but also the riskiest since the Rocket is larger than a dummy's, has trouble landing, and it's full is kept under high pressure if shot it becomes extremely hard to control. It is however very useful as an offensive weapon, a common strategy is to take control of the rocket and fly it at full speed into a group of enemies, much like a cruise missile.

Rocket Mk2

Trade Star's Rocket Mk2 is a much improved design of the Mk1 for maximum safety, but it is even more tall than the Mk1 making it a even bigger target but it's high speed and maneuverability over compensate for this weakness. The higher speed, larger size, and much higher maneuverability of this craft make it a even better guided missile than the Mk1. the offset is the much higher price.

Drop Ship Mk1



The more expensive and complex craft, this craft keeps its balance, and is easier and safer than a rocket. Assuming no one decides to blast off an engine, causing the craft to fall and spin out of control, crash, and explode, sending its doors nearly into orbit

Avalable Tools & Weapons

light digger

The Light Digger is, as its name suggests, the lightest of the Diggers. It somewhat resembles a handgun or pistol with a slightly extended barrel. It can dig through dirt, inner walls and flesh at an average rate, say like a dustbuster. You can hold the Light Digger and your Jet power is relatively unaffected. Some enemies spawn with the light digger.

medium digger

The Medium Digger is the average tool. It can dig through most materials, yet is not too heavy. It looks more like a submachine gun than a digger, but don't be fooled, the Medium Digger can pack a punch in its short range. Some enemies spawn with the Medium Digger.

Heavy digger

The Heavy Digger is the biggest of the Diggers. It can dig through anything you throw at it, including enemies flung at high speeds and the Impenetrable Bunker wall. Although it is heavy, it is worth every ounce. Some enemies spawn with the Heavy Digger, but rarely. The Heavy Digger is so powerful that if you're digging above you, you won't be able to use your jetpack! By firing down while falling, the Heavy Digger will act as a parachute.

Concrete sprayer

he concrete sprayer is the opposite of the digger. It sprays out concrete a short distance, and can be used to fill in holes and gaps in a bunker as well as to build small walls to use as cover. Enemies do not spawn with the Concrete Sprayer. It is possible to kill enemy or yourself by spraying concrete on the target, but the chance is relatively low. The Concrete Sprayer is excellent when your base has been compromised, and you have to seal the holes, or risk your brain being eliminated.

pusher

As the name suggests the pusher can well push actors and other things like a grenade away from you.Not a realy useful weapon it can be hard to use and difficult to find a real useful situation in order to use it.

Blaster

The blaster is a small light pistol with decent stoping power and a small clip.Buy it when you are low on gold

Laser Rifle

The laser rifle can quicly cut through orgranic matter making it ideal against unarmored targets.The drawbacks are a small clip and the fact than it can be less useful against unarmored targets.

Riot shield

The only available shield in the vanilla game.Use it with a shingle handed weapon but keep in mind than it can brake leaving you open in the enemy gunfire.

Coalition

These are standard, cloned troops. They represent typical military factions, and are usually clad in green. They are strong versus most gun fights, but are slow. They come pre-equipped with helmets, which can stand a single headshot before death. Their weapons come in a variety of types.

Units

Coalition Light Soldier

A Coalition trooper that's fast on its feet but is lightly armored.

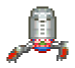
Coalition Heavy Soldier

A Coalition trooper with heavier built armor that protects and slows down this unit.

Coalition Battle Drone

A drone that generates a strong phase field around it for a surprisingly effective radial attack that can rip apart nearby units.

Coalition Medic Drone



This Drone has no offensive abilities BUT it heals units that are close to it.

Coalition Brain Robot

A heavily armored robot that contains the brain in a cavity in it's chest.

WEAPONS AND DEVISES

Coalition concrete sprayer

With this tool, you can create small bridges and walls, fill gaps and create small barricades.A little better than the trade star’s

Pulse digger

Coalition's pulse digger. Digs in wide pulses unlike traditional diggers.Faster than all the other but it cannot penetrate rock or cement.

pistol

Cheap and reliable, the standard sidearm of the Coalition. Quick reload times and good accuracy make up for the lack of stopping power.It is advised to always carry one in order not to find your self defendless.

Auto pistol

Semi-auto not good enough for you? Now with improved ammo capacity over the standard model,it can prove realy usefull in difficult situations.Suffers from low accuracy.

Auto shot pistol

Turn your enemies into swiss cheese with this little beauty, just watch out for the long reload times and the bad accuracy.One of the best choices for a side arm.

Compact assault rifle

Sacrifices stopping power and accuracy for a higher rate of fire.Overall weak and a bad choice if you are going to fight a lot of enemies or a few strong.Can be used as a last option side arm.

Assault rifle

Workhorse of the Coalition army,with a 30 rounds clip and good accuracy you can always rely on this cheap rifle to get you out of difficult situations.A weapon of choice if you want to create cannon folder troopers.

Gatling gun

Coalition's feared heavy weapon that features a large magazine and amazing firepower. Reloading is not an issue because there is enough ammo to kill everyone even remotely close.Can be used for anti dropships duty or for heavy support for your troopers.Just wach out the long reload and the spin up before the fire.It’s heavy weight makes it a weapon for defending duty.

Sniper rifle

Coalition special issue, semi-automatic precision rifle. Complete with scope for long distance shooting.Light weight and with a decent fire rate it can be used for hit and run tactics or for defending a remote difficult to reach area of the map.

Heavy sniper rifle

The sniper rifle's big brother. You only get 4 rounds to a clip, but why settle for a headshot when you can blow the head clean off with a giant .60 caliber bullet?This weapon is heavy and packs a strong punch.Use it for defend purposes and don’t go on a rampage with it.Drawbacks are its slow rate of fire and it’s heavy weight.

shotgun

Standard shotgun. Makes a brutal mess of infantry. 6 Shots, light weight, moderate reload time and small size make it great for close quaters.

Auto shotgun

Fully automatic shotgun. This thing is a blast, but with only a 6 round clip be wary of the reload times!Good for assaults on enemies.

Mauler shotgun

Automatic super shotgun. It launches small spike balls combined with traditional flechettes. This beast will literally rip apart anything in its path.Use it to defend your base.And keep in mind than it is realy heavy.

flamer

Light flamethrower. Grill and scorch your opponents with this pretty weapon. Extremely powerful for close quaters.Exelent weapon for bunker cleaning and for close combat assaults against the enemy bunker.

Napalm flamer

Napalm flamethrower. Cover and cook your foes with burning napalm and fire. The compressed container allows you to spray napalm continuously for several seconds! The only drawback is the short range shared by all flamers.Good for area denial weapon or for tight corridors bunker defence where the enemy cannot evade it.Drawback is the heavy weight making it ineffective for assults

Spike launcher

Fires an explosive charge with a short fuse that rips infantry to shreads and impales them hard with steel spikes. Only holds 2 rounds and takes time to reload.Not a particulary effective weapon for open combat.

Flak cannon

Flak Cannon. A devastating anti-air weapon, it can take out a dropship from a nice distance with easy and little aim. Not too great at close quaters though.Drawback is the heavy weight.

Auto cannon

Auto cannon for your heavy soldiers to use. Devastating power, high rate and lots of rounds to fire. Reloading this thing might take some time though.Use it against very heavy enemies, for light enemies prefer something lighter.

Revolver cannon

Revolver Cannon. A brutal and powerful automatic cannon. Launches heavy slugs at high velocities that smash the living hell out of their targets, and even if they don't, explode after a short delay. You get 6 shots, but before you get to use them all every opponent will be dead and thus the long reload time won't even bother you.Beware of it’s heavy weight.

Uber cannon

Uber Cannon. A shoulder mounted, tactical artillery weapon that is the most powerful weapon in the Coalition infantry arsenal.Crumbersom and heavy it is no a realy usefull weapon.

Rocket launcher

Rocket Launcher. Fires a rocket-propelled grenade. Be careful not to fire at close range if you want to keep your eyebrows intact.Slow firing rocket makes it difficult to actually hit a targer but if you succeed the reward is a decimated enemy.

Homing missile launcher

After a half-second in the air, this missile locks onto a nearby enemy. DO NOT fire at close range!Grate for anti rocket/dropship duties.

Dummies

Cheap and easy to build, these robots can be used for light attack roles. The tutorial mission is your first introduction to these guys, you are given the task of maneuvering your dummies in a small obstacle course to get acquainted with the mechanics of controlling a body. Dummies are well-rounded, and are also light, allowing them to move relatively quickly, however they are easily defeated by stronger enemies and should not be depended upon to hold a base.

UNITS

Dummy Controller

Once these were used by brains as back ups and to expand their control over a area.

Dummy

A cheap, reliable unit but has a bad habit of having it's limbs removed.

Dummy Dreadnought

A four-legged mobile turret can take lot of gunfire, but it can't aim up at high angles.

Dummy Small Turret

It may be small and unable to move, but it's generally mowing you down by time you knock its armor off with a standard weapon.

Dummy Drop Ship

An orange drop ship used by the Dummy faction. It has much less of its engines exposed, so it's less likely to lose one. Its doors can be destroyed by gunfire. It's also a bit cheaper, and harder to control than a Drop Ship Mk1.

Dummy Rocket

A small orange rocket low capacity for cargo. It's harder to control than a full size rocket and easier to shoot down, but it's cheaper. It makes an excellent projectile when crashed in to a target, especially when filled with bombs.

Weapons And Tools

Turbo digger

Dummy mining tool. Works as a powerful close range weapon too.

shielder

Materializes a temporary energy shield in front of the user for protection and/or slowing down enemy pursuers.

Rail pistol

A compact sidearm for a good price and decent performance!

nailgun

A powerful sidearm that fires heated nails at high velocities. Worth every gold ounce you will pay.

Dummy blaster

Energy based sub machine gun. Has a much shorter range than ballistic weapons, but its power and fast reloading make it an effective weapon.

repeater

Effective rapid fire support weapon. Doubles as a good assault weapon due to its large clip, but users should be warned of the long reload time.

Nailer cannon

Rapid fire version of the Nail Gun. Fire lots of heated nails at an incredible rate!

Dummy sniper rifle

Long range rifle with a scope. It has large ammo capacity and a steady rate of fire.

Dummy grenade launcher

Devastating grenade lobbing weapon. Use this weapon's slow firing velocities to shoot over walls and hills.

Destroyer cannon

This cannon fires bolts of slowly advancing energy that mow down multiple enemies in a row without slowing.

annihilator

Destructive heavy laser cannon. Mow down your opponents with a fat, satisfying laser beam!Heavy weight and a long reload makes this weapon crumbersome and difficult to use properly.

Disruptor grenade

Area denial grenade. Sets a deadly field upon detonation that lasts for 10 seconds.

Impulse grenade

Standard dummy grenade. Explodes into a devastating kinetic blast that will knock away or even tear apart its target.Good for taking out a lot of enemies.

Undead

First encountered in a "Zombie Cave" mission, these putrid smelling foes stagger towards you in the shape of Zombies, which range from fat and skinny ones, to bare skeletons. Essentially, a massing party. The weapons this faction uses can be guaranteed unsafe, bad, and most importantly of all of all cheap.

Skeleton

A reanimated skeleton for use as a cheap troop and cannon fodder.

Zombie Thin

A half baked clone that is just a little more fleshy than a skeleton.

Zombie Medium

This zombie is as close to the real deal as it will get...but thats not saying much.

Zombie Fat

A zombie that was left in the tube a wee bit to long and is now slow*er* and fat, but that fat makes it a bit more durable for a zombie that is.

Weapons

blunderpop

A single shot shot pistol...
Not much to say about it···

Blunder buss

another single shot gun but with more power.

Blue bomb

A decent and cheap grenade.

RONIN

A collection of "rebels". They use antique weapons, and recruit civilians, prisoners, and the poor. Effectively, cannon fodder. Each type of soldier has unique characteristics which help it slightly with combat and effectiveness.

Dafred

Dafred is simply awesome.

Mia

Mia is flexible and runs like the wind.

Dimitri

Dimitri is tall and can see pretty far.

Brutus

Brutus is a tough guy, although heavy and slow

Sandra

Sandra's sexy hair can make any soldier hesitate.

Gordon

Gordon can soak up bullets, but don't crash into anything if you like living.

Weapons and devises

Foam sprayer

shovel

Lady pistol

glock

Hak 20

luger

Desert Eagle

peacemaker

Uzi

Tommy gun

Yak 47

Yak 4700

M 16

M 1600

Long rifle

Ronin sniper rifle

Pumpgun

shotgun

Spaz 12

Spaz 1200

bazooka

RPG M17

a rocket propelled chainshaw...

honestly

Pineapple grenade

Stick grenade

Molotov cocktail

stone

Distractor device

Chapter 2

Useful links

1)devlog (http://www.datarealms.com/devlog/)

Where the developer Daniel Tabar (a.k.a DATA) will post the newest development of data realms projects (cortex command).

2)cortexwiki(http://www.datarealms.com/wiki/index.php/Main_Page)

The official wiki for the cortex command containing tutorials and information for cortex command.Updated from time to time it can provide answers for many questions you might have.

3)FAQ(http://www.datarealms.com/forum/faq.php)

The FAQ for the datarealms forums containing a useful troubleshooting in case something goes wrong with your account or if you have any questions about some aspect of the forum. View it at lest once.

4)SEARCH(http://www.datarealms.com/forum/search.php)

The datarealms forums can be a bit big if you visit them for the first time as a user. Instead of opening a new thread go in the search and type in what you want to find. It is user friendly and able to give you results before you know it. It is advised to use it before you open a new thread.

5)FORUM RULES(http://www.datarealms.com/forum/viewtopic.php?f=61&t=10345)

The data realms forum rules,read them carefully if you want to stay in the forums.

Chapter 3

What does data realms forums has?

Data realms forum is divided in 3 main categories

1)Cortex command

2)mods and scenes

3)Data Realms general

1)"Cortex command" category is divided in 4 sub-categories

A)"dev log" a link which takes you to the development log we analyzed before.

B)"Wiki(link)" again a link which takes you to the cortex Wiki main page.

C)"General" contains discussions about cortex command and general announcements, contains 1 sub-category named "Media" which holds sprites, sounds, art and other media related to cortex command.

D)"Support" this is the sub-category which you can use to find help for technical problems related to cortex command for example if you cannot start the game or you have license problems.

2)”mods and scenes” category is divided in 3 sub-categories

A)”mod making” you can use this sub-category to post you unfinished mod in order to get some feedback(what you should improve etc) before you release it or if you have some problems/questions related with modding.this sub-category is further divided in 2 more sub-categories named “Lua scripting” which you can use to discuss Lua related things or to find some help should your mod requires some form of Lua and you had a problem coding it or you wanted to request a Lua code.The other sub-category is named “Requests” where you can request sprites sounds or codes for your mod.

B)”mod releases” is the sub-category where you can post you completed mods and find other users mod.
This category is the part you will most likely visit the most due to the fact than the whole fun you can get from cortex command comes from mods and the innovation of their creators.

C)”scene releases” this sub-category holds the completed user generated scenes you can use.It is divided in a sub-category named “scene making” which you can use to find help and advises should you be making a new scene.

3)”data realms general” this category is divided in 2 sub categories

A)”General discussion” talk about everything with the community there. It is divided in 3 sub-categories which are easy to understand their use so I will not analyze them.

B)”forum news/feedback/forum help” here you can ask questions and make suggestions for the forums.divided in 2 self explaining sub-categories.

Chapter 4

Getting started with mods

(this Chapter takes as granded than you know how to download a mod/scene).

Mods and scenes comes in files which have the extencion .rte.

The mods and the scenes you will download will be always in either .rar or .zip format.

In order to open these files you will need winzip which can be found and be downloaded from here (http://www.winzip.com/index.htm).

Once you have downloaded and installed the latest version of winzip or an similar program to your computer you must double click the mod you have downloaded (for example mod.rte) and extract it on the directory you have installed cortex command.

(default for windows is C:\Program Files\data realms\cortex command)

Or you can first extract the .rar file in you desktop and drag and drop it where you have installed cortex command.

And that is all about how to install a mod to your computer.

To deleted you simply delete the file of the mod.

Chapter 5

start modding

6.1)what you will need to mod

Modding in cortex command is pretty simple and requires very few actual resources.

You will need a text reader to open the .ini files, a calculator if you want to do waklpaths and offsets and lastly a .bmp editing program like paint.

That is all you will ever need to use in order to start modding.

6.2)introduction in .ini and vocabulary

Modules loaded in Cortex Command are simply directories in the Cortex Command directory with an *.rte* extension. On loading, the game will check for any modules in its home directory.

In order for the game to actually load your module, the folder must contain a file called **index.ini** This is pretty self-explanatory. It is an index for all files to be referenced in the module.

Within the index.ini file are a variety of properties and attributes--examples of which are given below:

```
DataModule
	Author = Me
	ModuleName = My Mod
	IconFile = ContentFile
		FilePath = Example.rte/Icon.bmp

	IncludeFile = Example.rte/Content1.ini
	IncludeFile = Example.rte/Example/Content2.ini

	//Comments are not read.

	/*
	Block comments are also ignored.
	*/
```

DataModule

- This will always be the first line. The only purpose of this line is to inform the game's interpreter that the file being read is the index of a module. If this line is not included, the interpreter will throw an error: *Could not find DataModule.*

Author

- This is a place for the creator to show who created the file. It has no bearing on the actual mod. It is an optional property.

ModuleName.

- This property gives your mod a unique identifying name. It *must* be unique. Mods cannot share ModuleNames. This is a mandatory property.

IconFile.

- This property gives your mod a uniqe banner, flag, or icon to identify it in the buy menu. The property is optional, but the attribute must be **ContentFile** if it is included.

FilePath

- If you opt to create an IconFile, your next declaration will be FilePath. This is pretty straight forward. The FilePath is a parent of the IconFile. In English, it says where the image file resides in the module. The attribute of this property will always be **YourMod.rte/folder/file.extension** You must always include the name of the module (the .rte) that the file is contained in. This property is optional, but must be included if the IconFile property is declared.

IncludeFile

- Finally, the meat of an index. These properties tell the interpreter where to locate other module files to include. Generally, mod contents are not contained in the index, but are organized into other .ini files for organizational purposes. If you opt to add module content to other documents, you must call IncludeFile to tell the game where to look. The attribute for this property is the file location within the module. Again, this must follow the **YourMod.rte/folder/file.extension** format.

The number of includes is essentially limitless. This allows you to organize your files as you see fit and increases the interpreter's flexibility. Many larger mods have separate includes for playable actors, crafts, weapons, devices, and scenes. The complexity of your project is a large factor in how you organize your includes.

If you have experience in object-oriented programming and scripting, you already know at least half of what is required to create mods for this game. For now, we will assume you do not have this experience. For now, lets start with a vocabulary primer:

Vocabulary

You will see these words used quite frequently when you are creating mods. They are integral to the object-oriented programming design cycle. Learn them.

Classes

- A class is a conceptual structure that is used to define an item. If you want a weapon, you use the HDFirearm class. Classes cannot be changed or edited. Their structure is unmodifiable.
- Real-world example: A car is a class of transportation. Don't think about it too hard, just take that as a fact.
- In-game example: An HDFirearm is a class of held device.

Objects

- An object is an instance of a class. Simply put, an object is a class that you can edit and modify to your needs. It is created for you to make a specific type of class.
- Continuing our real-world example: A Volkswagen Beetle, a Toyota Camry, or a Ford Taurus would be objects of the *car* class. Does that make sense? A Beetle, Camry, or Taurus are a "type" of car. The *type* of car would be the object.
- In-game example: A Yak 47, Rail Pistol, and the Gatling Gun are objects of the HDFirearm class.

Properties

- A property is another fancy, object-oriented word for variable. An object has certain attributes, and a property defines what those attributes are.
- Again, using the car analogy: A Volkswagen Beetle (the object) can have several properties. They include (but are not limited to): *color*, *engine size*, or *transmission.*

- In-game example: RateOfFire, FullAuto, and Mass are properties of the YAK-47 object of the HDFirearm class.

Attributes

- Attributes are simply the values of properties. Mass = 16. Mass is the property, 16 is the attribute. Fairly simple.
- Real-world example: Red (attribute) is the color (property) of the Beetle (object) that is a car (class).
- In-game example: 4 (attribute) is the mass (property) of the Rail Pistol (object) that is an HDFirearm (class).

Fundamentals

To begin, the Cortex Command interpreter has very strict rules. Proper capitalization and tabbing are absolutely critical. Look at the two examples below:

Example one:

```
AddDevice = HDFirearm
	PresetName = SMG
```

- Example two:

```
Adddevice = hdfirearm
Presetname = SMG
```

Example one is properly capitalized and tabbed. Example two is not, and will throw an error. One of the first errors you should check should you experience errors is proper spelling, tabbing, and capitalization.

Additionally, it is considered good form to properly comment and document your code. This is accomplished through a variety of means, primarily code commenting. There are two methods to do this: Line comments and block comments.

```
// This is a line comment, the double forward slash denotes this.  It can only
                              occupy one line.

                                    /*
                           This is a comment block.
                             Anything between the
                              forward slash and
                            asterisk is a comment
                                    */
```

Those are basic conventions implemented by the interpreter. If you don't follow these, your code (and as a result, your mod) will not load and the game will crash.

6.3)variables

- `InstanceName`
- `FilePath`
- `Priority`
- `Mass`
- `LifeTime`
- `Sharpness`
- `HitsMOs`
- `GetsHitByMOs`
- `Color`
- `R`
- `G`
- `B`
- `Atom`
- `Material`
- `CopyOf`
- `TrailColor`
- `TrailLength`
- `RestThreshold`

- `SpriteFile`
- `FrameCount`
- `SpriteOffset`
- `X`
- `Y`
- `AngularVel`
- `AtomGroup`
- `AutoGenerate`
- `Resolution`
- `Depth`
- `DeepCheck`
- `Framerate`
- `EffectStartTime`
- `EffectStopTime`
- `EffectStartStrength`
- `EffectStopStrength`
- `ScreenEffect`
- `EffectAlwaysShows`
- `DeepGroup`
- `JointStrength`
- `JointStiffness`
- `DrawAfterParent`
- `BurstTriggered`
- `EmittedParticle`
- `ParticlesPerMinute`
- `BurstSize`
- `Spread`
- `MaxVelocity`
- `MinVelocity`
- `PushesEmitter`
- `EmissionEnabled`
- `EmissionsIgnoreThis`
- `BurstScale`
- `BurstSpacing`
- `EmissionDamage`
- `FlashOnlyOnBurst`
- `EmissionSound`
- `LoopSetting`
- `BurstSound`
- `EndSound`
- `Flash`
- `EmissionCountLimit`
- `BurstDamage`
- `EntryWound`
- `ExitWound`

- `DetonationSound`
- `Path`
- `StanceOffset`
- `StartThrowOffset`
- `EndThrowOffset`
- `TriggerDelay`
- `ParticleNumberToAdd`
- `GibImpulseLimit`
- `GibParticle`
- `Count`
- `InheritsVel`
- `ParticleCount`
- `Particle`
- `Shell`
- `FireVelocity`
- `ShellVelocity`
- `Separation`
- `RoundCount`
- `RTTRatio`
- `RegularRound`
- `TracerRound`
- `GoldValue`
- `JointOffset`
- `SharpStanceOffset`
- `SupportOffset`
- `SharpLength`
- `Magazine`
- `ParentOffset`
- `FireSound`
- `EmptySound`
- `ReloadStartSound`
- `ReloadEndSound`
- `RateOfFire`
- `ReloadTime`
- `FullAuto`
- `FireIgnoresThis`
- `ShakeRange`
- `SharpShakeRange`
- `NoSupportFactor`
- `ParticleSpreadRange`
- `ShellSpreadRange`
- `ShellAngVelRange`
- `MuzzleOffset`
- `EjectionOffset`
- `GibWoundLimit`

- `OneHanded`
- `Offset`
- `Discardable`
- `BreakWound`
- `PinStrength`
- `BodyAnimMode`
- `BodyAnimDuration`
- `Status`
- `Health`
- `Team`
- `CharHeight`
- `Door`
- `OpenOffset`
- `ClosedOffset`
- `OpenClosedAngle`
- `AngleDegrees`
- `DoorMoveTime`
- `ClosedByDefault`
- `ResetDefaultDelay`
- `SensorInterval`
- `StartOffset`
- `SensorRay`
- `SkipPixels`
- `Rotation`
- `DoorOpenSound`
- `OpenClosedOffset`
- `OpenAngle`
- `ClosedAngle`
- `HFlipped`
- `DeathSound`
- `GibSound`
- `AimAngle`
- `AimDistance`
- `HolsterOffset`
- `HeldDevice`
- `Hand`
- `MaxLength`
- `IdleOffset`
- `MoveSpeed`
- `Foot`
- `ExtendedOffset`
- `ContractedOffset`
- `BodyHitSound`
- `PainSound`
- `DeviceSwitchSound`

- `Head`
- `Jetpack`
- `JumpTime`
- `FGArm`
- `BGArm`
- `FGLeg`
- `BGLeg`
- `HandGroup`
- `FGFootGroup`
- `BGFootGroup`
- `StrideSound`
- `StandLimbPath`
- `StartSegCount`
- `SlowTravelSpeed`
- `NormalTravelSpeed`
- `FastTravelSpeed`
- `PushForce`
- `WalkLimbPath`
- `CrouchLimbPath`
- `CrawlLimbPath`
- `ArmCrawlLimbPath`
- `ClimbLimbPath`
- `JumpLimbPath`
- `DislodgeLimbPath`
- `MountedMO`
- `ImpulseDamageThreshold`
- `AimRange`
- `Turret`
- `LFGLeg`
- `LBGLeg`
- `RFGLeg`
- `RBGLeg`
- `LFootGroup`
- `RFootGroup`
- `LStandLimbPath`
- `LWalkLimbPath`
- `LDislodgeLimbPath`
- `RStandLimbPath`
- `RWalkLimbPath`
- `RDislodgeLimbPath`
- `EmissionOffset`
- `MinThrottleRange`
- `MaxThrottleRange`
- `EmissionAngle`
- `LifeVariation`

- `StableVelocityThreshold`
- `LThruster`
- `RThruster`
- `ULThruster`
- `URThruster`
- `LHatchDoor`
- `RHatchDoor`
- `HatchDoorSwingRange`
- `HatchDelay`
- `HatchOpenSound`
- `Velocity`
- `CrashSound`
- `CanLand`
- `WillIdle`
- `Position`
- `RLeg`
- `LLeg`
- `MThruster`
- `RaisedGearLimbPath`
- `LoweredGearLimbPath`
- `LoweringGearLimbPath`
- `RaisingGearLimbPath`
- `WrapX`
- `WrapY`
- `ScrollRatio`
- `BitmapFile`
- `GoldCost`
- `MaterialFile`
- `BGColorFile`
- `BitmapOffset`
- `FGColorFile`
- `DebrisFile`
- `DebrisPieceCount`
- `DebrisMaterial`
- `TargetMaterial`
- `OnlyOnSurface`
- `MinDepth`
- `MaxDepth`
- `DensityPerMeter`
- `LocationOnPlanet`
- `GlobalAcceleration`
- `Terrain`
- `BackgroundTexture`
- `FrostingMaterial`
- `MinThickness`

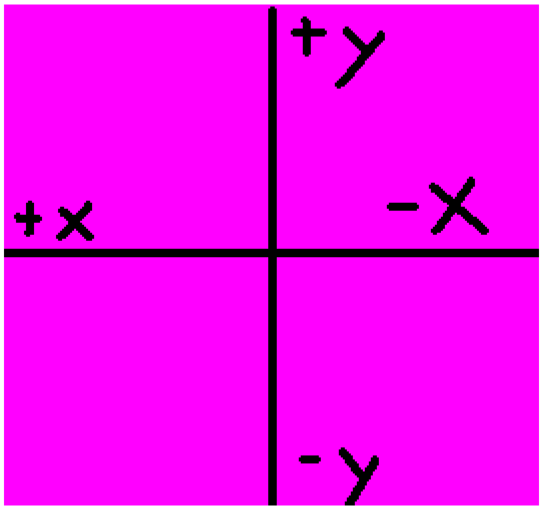
- `MaxThickness`
- `InAirOnly`
- `Location`
- `AIMode`
- `SceneName`
- `TeamCount`
- `PlayerCount`
- `TeamOfPlayer1`
- `FundsOfTeam1`
- `CPUTeam`
- `Difficulty`
- `SpawnIntervalEasiest`
- `SpawnIntervalHardest`

```
Sound
ContentFile
MOPixel
Color
Atom
Material
MOSRotating
AtomGroup
MOSParticle
Attachable
AEmitter
Emission
TDExplosive
Gib
Round
Magazine
HDFirearm
HeldDevice
ADoor
Sensor
Actor
Arm
Leg
AHuman
Vector
Turret
ACrab
ACDropShip
Exit
ACRocket
SceneLayer
TerrainObject
SOPlacer
TerrainDebris
```

6.4)offsets

Due to large amounts of requestsone, I am going to make a guide about how to use offsets correctly and not just guess and hope to get lucky. I'm not the best at explaining things, so bear with me. If you don't know the basics of modding or even what offsets are, get out. Seriously.

OFFSETS



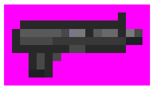
What you see here is the X and Y axis for Cortex Command

-X = Right
+X = Left

-Y = Down
+Y = Up

The first thing you'll probably notice is that the X axis is switched.
Rather than -X meaning left like in
the conventional grid, the kind you'll see in Algebra and the sorts, -X mean right and vice versa.

The most time you're probably going to need this is when you're changing offsets for that new gun you've just sprited. Most people try not to sprite actors larger or smaller than usual.



Imagine that this is your new weapon you've just sprited. You want it so that your clone will hold it properly. Meaning you'll need to fill out these key chucks of code. (the meanings will be in parenthesis)

Code:
```
   SpriteOffset = Vector (This sets the origin of the sprite, should be in the very middle of your sprite)
      X = ?
      Y = ?
```

Code:
```
   JointOffset = Vector (The placement of the main hand)
      X = ?
      Y = ?
```

Code:
```
   StanceOffset = Vector (The placement of the shoulder when carrying the gun)
      X = ?
      Y = ?
   SharpStanceOffset = Vector (The above but when aiming down sights)
      X = ?
      Y = ?
   SupportOffset = Vector (Placement of secondary hand)
      X = ?
      Y = ?
```

Code:
```
   Magazine = Magazine
      CopyOf = Magazine SMG
      ParentOffset = Vector (Where the magazine
goes)
         X = ?
         Y = ?
```

Code:
```
   MuzzleOffset = Vector (Where the bullets fire
from)
      X = ?
      Y = ?
   EjectionOffset = Vector (Where the shells, if
any, eject)
      X = ?
      Y = ?
```

So, how are we going to find the correct coordinates?

Easily.

First of all, we'll need to set the Sprite offset.
Our sprite here is 18 by 10 pixels.

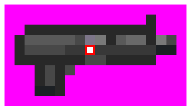
Therefore...

Code:
```
   SpriteOffset = Vector
      X = -9
      Y = -5
```

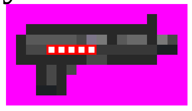
Just take half of your sprites dimensions, negate it, and plug it in. That's all you need to know for Sprite Offsets. Simple, right? This bit of code will help you a lot later on.
Note: If your sprite Offset has a decimal, just round up. There's no such thing as half a pixel.

Now, lets move onto the Joint offset.

This will be where the main hand goes. Setting this one up is a bit more trickier.
First, make a mark off where you're Sprite offset is, which should be in the middle



This is where we will start counting. Start marking how far you want your hand to the left or right.



I counted 4.

Code:
```
   JointOffset = Vector
      X = 4
      Y = ?
```

Remember, +X means left and -X means right.

Now count how much you want to go up or down.

I counted 3.

Code:
```
   JointOffset = Vector
      X = 4
      Y = -3
```

And for whatever reason, **JointOffsets and Parent Offsets (Which is what we are currently working on) need to be the opposite of what you want.** Why, I do not know.

So:

Code:
```
   JointOffset = Vector
      X = -4
      Y = 3
```

There, you're done. That's all you have to do to set an Offset.

Things you might have missed:

- After setting the Coordinates for JointOffsets and Parent Offsets, negate them. (Negatives turn to positives, and vice versa)
- The -X mean right and +X means left
- If you've followed this tutorial and the Offset still doesn't match up, negate for good measure.

6.5)templates of a gun with comments

```
AddEffect = MOPixel // This is the instance call. We're adding an effect, it's an MOPixel
    PresetName = Bullet Blaster Pistol // This is the instance name, so the
instance can be recalled upon later
    Mass = 0.25 // This is the mass of the object, in kilograms
    LifeTime = 650 // This is how long the particle will last before disappearing, in
milliseconds.
    Sharpness = 3 // This is how sharp the MOPixel is; how well is knocks terrain loose
and penetrates objects.
    HitsMOs = 1 // This boolean value defines if the MOPixel hits other things.
    GetsHitByMOs = 0 // This boolean value defines if the MOPixel gets hit by other
things.
    Color = Color // This starts the RGB of the MOPixel's color.
            R = 255
            G = 255
            B = 255
    Atom = Atom // This starts the definition of the MOPixels one and only Atom.
            Material = Material // This starts the definition of the Atom's material.
                    CopyOf = Bullet Metal // This copies a previously defined material
to use.
            TrailColor = Color // This starts the RGB of the trail effect's color.
                    R = 255
                    G = 255
                    B = 255
            TrailLength = 30 // This modifies the length of the trail.

AddAmmo = Round // The instance call, defines the next text as parts of a round.
    PresetName = Round Blaster Pistol // The name used to refer to the instance
later.
    ParticleCount = 2 // The number of particles created when this round is used. This
is what makes a shotgun a shotgun
    Particle = MOPixel// Defines the particle that will make up the round. Does not
need to be a MOPixel.
            CopyOf = Bullet Blaster Pistol// Copying a previously defined and
named object.*
Shell = MOSParticle // Defines the Shell that will eject as the round fires. Does not need to
be a MOSParticle.
          CopyOf = Casing // Copying a previously defined and named object.
FireVelocity = 80 // This is the velocity at which the round particle will be fired.
ShellVelocity = 10 // This is the velocity at which the shell will be ejected. Separation = 5 //
This is the variance of the X position from the MuzzleOffset of the HDFirearm.

AddAmmo = Magazine   // The instance call, defines the next text as parts of a Magazine.
    PresetName = Magazine Blaster Pistol // The name used to refer to the
instance later.
    Mass = 1 // The mass in kg for example; 0.01 would be 10g, 0.1 would be 100g and 1
would be 1kg.
    HitsMOs = 0 // Boolean value that determines if this particle can hit by other particles.
```

```
	GetsHitByMOs = 0 // Boolean value that determines if this particle can hit by other particles.
	SpriteFile = ContentFile // Tells The Magazine to use the sprite specified
		FilePath = Base.rte/Devices/Pistols/MagazineEnergyA.bmp
	FrameCount = 1// How many frames to use.
	SpriteOffset = Vector
		X = -2
		Y = -2
	ParentOffset = Vector
		X = 1
		Y = 1
	EntryWound = AEmitter // The emitter to use for where the bullet enters the weapon.
		CopyOf = Dent Metal
	ExitWound = AEmitter // The emitter to use for where the bullet exits the weapon
		CopyOf = Dent Metal
	AtomGroup = AtomGroup
		AutoGenerate = 1
		Material = Material // Defines the material that the round will be made up of.
			CopyOf = Military Stuff // Reference to the material you want the round to be made of.
		Resolution = 2
		Depth = 0
	DeepGroup = AtomGroup
		AutoGenerate = 1
		Material = Material
			CopyOf = Military Stuff
		Resolution = 3
		Depth = 1
	DeepCheck = 0
	JointStrength = 200 // How much force is required to be slung out of your grip.
	JointStiffness = 1 // How much force is required to move the joint.
	JointOffset = Vector
		X = 0
		Y = 0
	DrawAfterParent = 0 // Boolean value that determines if it is drawn in front of the parent a.k.a weapon(1) or not (0).
	RoundCount = 10 // How many rounds are in the magazine.
	RTTRatio = 0 // RoundToTracerRatio, how many rounds:tracers.
	RegularRound = Round // Defines which round instance to use as the regular round.
		CopyOf = Round Blaster Pistol
	TracerRound = None // Defines which round instance to use as the tracer round.
```

```
AddDevice = HDFirearm // required for an HDFirearm
PresetName = blaster // what you want the gun's name to be in-game
Description = A compact sidearm for a good price and decent performance! // in-game
description of gun
AddToGroup = Weapons // what section the gun will be in
Mass = 4 // the weight of the weapon
HitsMOs = 0
GetsHitByMOs = 1
SpriteFile = ContentFile // what file to use for the appearance of the gun
FilePath = base.rte/blaster
    FrameCount = 2
    SpriteOffset = Vector // the offset of the gun while being held X = -5 Y = -4
EntryWound = AEmitter
        CopyOf = Dent Metal
ExitWound = AEmitter
        CopyOf = Dent Metal
GoldValue = 10 // the value of the gun in-game
AtomGroup = AtomGroup
AutoGenerate = 1
Material = Material
        CopyOf = Military Stuff
Resolution = 4
Depth = 0 DeepGroup =
AtomGroup AutoGenerate = 1
Material = Material CopyOf =
Military Stuff Resolution = 4
Depth = 10 DeepCheck = 0
JointStrength = 75
JointStiffness = 0.5
JointOffset = Vector X = -1 Y = 2
DrawAfterParent = 0
OneHanded = 1
StanceOffset = Vector X = 12 Y = 0
SharpStanceOffset = Vector X = 13 Y = -2
SupportOffset = Vector X = -1 Y = 3
SharpLength = 90
Magazine = Magazine // which magazine to use
CopyOf = Magazine blaster
Flash = Attachable // what muzzle flash to use
CopyOf = Muzzle Flash Pistol
 FireSound = Sound // the sound the gun to make when it fires
AddSample =
ContentFile FilePath = Dummy.rte/Devices/Weapons/Rail Fire.wav
EmptySound = Sound // the sound the gun makes while the magazine is empty and the trigger is
pressed
AddSample =
ContentFile
FilePath = Base.rte/Devices/EmptyClick3.wav
ReloadStartSound = Sound // the sound that plays when you start to reload
AddSample = ContentFile FilePath = Base.rte/Devices/ReloadStart.wav ReloadEndSound =
Sound // the sound that plays when reloading is done
```

```
AddSample = ContentFile
FilePath = Base.rte/Devices/ReloadEnd.wav
RateOfFire = 55 // the rate of fire in milliseconds
ReloadTime = 1600 // the reload time in milliseconds
FullAuto = 0 // whether or not the gun is automatic or not
FireIgnoresThis = 1
ShakeRange = 7
SharpShakeRange = 4
NoSupportFactor = 1.8
ParticleSpreadRange = 0
ShellSpreadRange = 8
ShellAngVelRange = 2
MuzzleOffset = Vector // the offset for the muzzle flash
	X = 7
	Y = -1
EjectionOffset = Vector // the offset for the bullet casings
	X = -3
	Y = -1
GibWoundLimit = 2 // the number of hits the gun takes before breaking
AddGib = Gib
	GibParticle = MOPixel
CopyOf = Spark Yellow 1
	Count = 3
	Spread = 2.25
MaxVelocity = 20
MinVelocity = 8
```

Chapter 1

spriting

7.1)basic introduction on sprites

Giving your creations a colorful look is a difficult process for the beginner. Sprite art is tricky to shade properly if you don't know what you're doing. This is the difference between a hobbyist, a newbie, and a professional: Most will use extremely different forms of sprite shading. It's take hours to cover all the possible pixel art coloring techniques in-depth.
Here are brief descriptions of several of the main types of sprite shading for graphics. You can find many examples online if you look around. These three are typically used in games and animated series.

Pillow Shading

In one word: Don't do it! This is typically where the inside of an object is brighter than the rest, and you make it darker in colors the deeper it goes and along the outlines. It looks horrible unless all the other pillow shade sprites are the same style. Even then, it doesn't look all that good!

Comic Shading

This is typically when your sprites have dark black edges. The inside colors also have outlines, and usually only have two or three different shades in their associated parts. It looks a lot like what you'd find on a cheaply animated cartoon. However, unlike pillow shading, sprites with comic shading do look quite good if everything matches the style.

Natural Shading

This is the most difficult, and most effective type of sprite shading to pull off. Generally speaking: Your outlines are all a somewhat darker color of the other on the sprite in that area. For example, someone in blue pants. Their outlines would be dark blue. You then shade all the inner pieces with lighter colors as if sun was shining in a direction, typically north west.

Pro tip: It is with natural shading that other colors make a large difference. Purples are fantastic for shadows. Using light green helps add a vibrant "alive" mood to the sprite graphics.

What is Pixel Art?

First let's learn what a pixel is.

A pixel is the smallest part of an image. The word pixel is based on a contraction of *pix* (for "pictures") and*el* (for "element") Pixels are normally arranged in a regular 2-dimensional grid, and are often represented using dots, squares, or rectangles.

Now we know what a pixel is, how is it art?

Well, you have probably seen a basic pixel image before. Most of online games are made with pixel art, like Habbo Hotel. The art of pixeling is basically making the eye think a color but blend them together. Ok, OK! I know that wasn't too clear, what I mean is by making each pixel a slight variation of that color, the eye then sees it as one color changing.

Pixel Art - How is it generally made?

Pixel Art is normally made almost one pixel at a time! (that's why its RARE to see LARGE pixels.) Using a graphic program of choice, you don't use any fancy filters or softening smudges or any of those tools. It seems simple enough, but it really is difficult to master. Here is an example of a complex Pixel:

This is a complex Pixel, so much detail is in this it's hard to believe!

Here are a group of pixel game characters from the online chat game, Habbo Hotel. (www.habbohotel.com)

What are some of the uses of Pixeling Art?

Pixeling Art can be applied almost anywhere computer graphics are used (because there basicly the same thing, but done manually and not with code filters etc). Pixels normally have a more "edgy" look to them, but they are appeling and high in demand!
Pixeling can be used in all of these cases:

- Logos
- Cortex command sprites!!!!!!!!!
- Site Banners
- Signatures
- Avatars
- Word and other print-outs (business cards, flyers, etc)
- Computer Games
- Flash movies
- Flash Websites
- and many more!

MISTERXMAN

A very good Pixel Runescape Signature.

I want to do Pixel Art! What must I do to start?

So you want to try your hand at pixeling? It is A LOT harder then it looks. But do not fear, practice makes perfect. The best platform for pixeling (and my choice) is Microsoft Paint (if your on a windows computer, you have it!). Another free option you might wish to use (Don't cheat!) is Paint.net.

Paint.Net is a super verison of MS Paint, it is hosted for free and can be downloaded at: www.paint.net (the right one, not the left!)

If set correctly all of these other programs will allow you to pixel with them:

- Photoshop (all versions)
- Macromedia Fireworks(all versions)
- Corel Painter Essentials (all versions)
- Ms Paint (all versions)
- Paint.Net (all versions)
- there are a TON more out there, almost all programs that allow you to create an image you can pixel with.

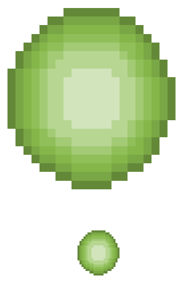
7.2)cortex command palette

Cortex command uses a very specific palete.
In order for you to make cortex command sprites you should always edid a pre exsisting sprite.
Keep in mind than the pink is the background and than it will not show in the game nor will it cound as a part of your sprite.

7.3)giving your sprites correct shading

by kitted - http://kitted.deviantart.com

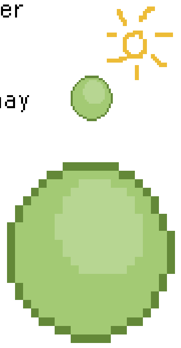
Pixel Art Shading Tutorial



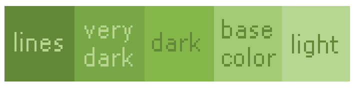
Light and shadow. Those are the basics. Light sources are all around you. Overhead lights. Lamps. The sun. Your own monitor. What these all have in common is that they cast **light** on nearby things. When light is cast on something, a shadow is cast by the object. These are the basics of light and shadow, and shading in art is meant to emulate these things for realism.

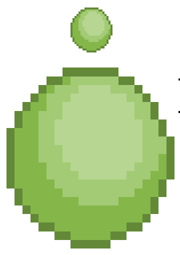
Pictured above is a common mistake made by artists - shading around the edges, paying no attention to a light source. While this is easier and tempting to do it is not realistic. It makes it seem as if there is a very strong light source in the exact middle of the object. Realistically a light source is never exactly in the middle, and more than one light source is common in reality.

So, let's pick a light source! Pictured below you'll see I've chosen to make my light source in the upper right, and a small distance away from the circle. So I apply a light color to a darker color which I call my base color.

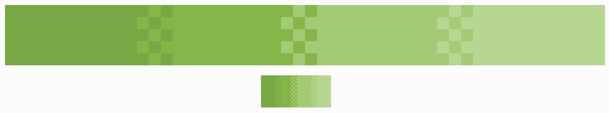


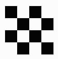
Also, let's pick out a simple color palette. You may or may not end up using all of them.



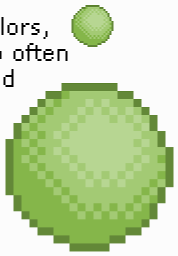


Taking the dark color, I apply it around the edges of the base color. It's a little dark, isn't it? Why don't we try a little dithering?

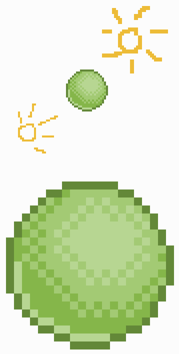




See this checkerboard thing? This is dithering. Do this with two different colors to make them blend better.



Dithering is a technique primarily used to blend colors, and it's very common in pixel art since pixel art so often deals with limited color palettes so it has to be used to blend the colors better. In this case, let's try dithering around the edges of the dark color and the light color. This makes it blend much better and makes the colors softer.



But we're not done yet! Realistically there's usually more than one light source in a picture. So let's pretend there's another light source, this time a weak one in the lower left. Take the light color and make small (very small) lighting on the lower left edge of the circle on the dark color. You do not usually dither this light source.

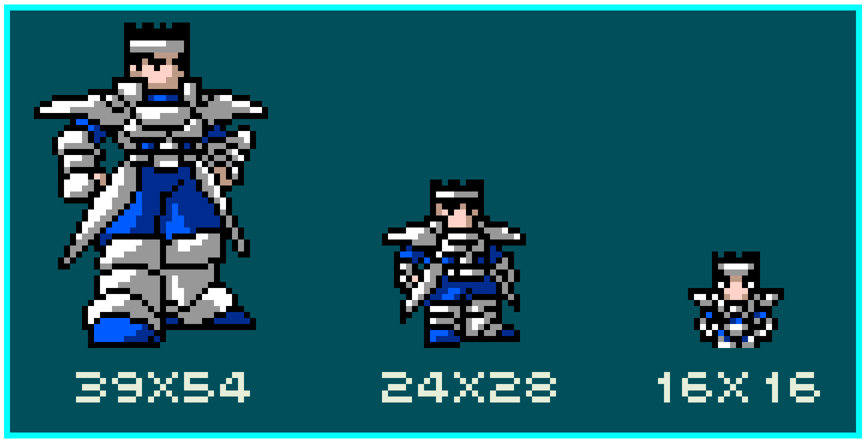
You're finished!

7.4)sprite figures introduction and how to make one

The main character's sprite...Possibly THE most important part of your game. Yeah, you can argue that "Oh no, the programming is more important! Without it there's nothing else!" or "No the scrolling engine! It makes things smooth!" or "It's the music! Music is more important than art! The music sets the mood in the background!"...But no one will see the programming, a nice scrolling engine doesn't matter if you're scrolling over a picture of stickmen, and the music only sets up half the mood. If you're putting out advertisements or anything, making the back of your game's box because it's so amazing that it's going to consoles or whatever, you can't put music on it...Nor programming...You're down to screenshots. Screenshots are what people see first, and that's what they base their opinions on instantly, regardless of whether they realize it or not. If they check out a shot and see stickmen walking on circles of tiled grass, they have in their mind "Ugh, what an ugly game..." and when they play it (IF they play it...A lot of people won't even download or check out a game if the screenshots are ugly) they go into it thinking "This is an ugly game..." and the wonderful music and scrolling engine and programming you have are all ruined because the player is playing your game only half caring. Art is EXTREMELY important in making your game, and yeah, you CAN get away with ugly stickmen...But you'd better have one incredible plot, because you're going to have to make up for that. And even if you have the most amazing plot in the world, it's not going to matter if people don't sit through your ugly graphics long enough to find out about it.

Now specifically, the most important sprite is the main character's sprite. "Why that one? What about my super evil arch-nemesis bad guy set to take over the world for some unknown reason?" Well, he's important too, being an important character in the plot, but he's not as important as the main character. Why? Because the main character is what the player is going to be looking at for the entire game. They might walk through 40 different types of tilesets and journey all over the world and everything, but the one constant graphic is going to be that main character's sprite walking along. If the player doesn't find the main sprite appealing, and even finds it kind of ugly, he's not going to be very enthusiastic about playing because he doesn't care about the character...And if he doesn't care, the plot won't have as much effect on him, and he might not even finish playing the game because he just hates looking at the main character's sprite so much. Yeah, it's not a very bright way of playing a game, but why would I play a game where I hate looking at "myself" when I could go play another one where I'm a wicked looking knight or a mysterious looking mage or something?

Okay, so I think I've made my point...The main character's sprite (and everyone else in the party if you have more than one) is the most important. So where do we go from here? Well, let's talk about limits...and then how to get around them. Here are some quick sprites I did up from that one up above:

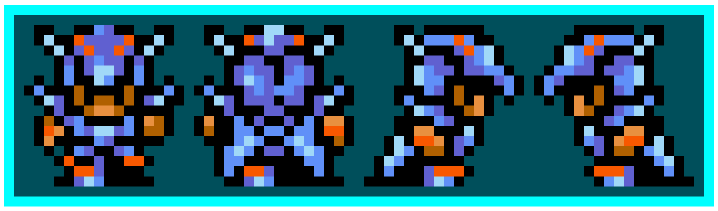


Okay, so you're Mr. Game Artist...And you're working with a programmer who wants you to make up the sprites. So you come up with the first one...39x54, but it's really pretty irrelevant. This first sprite is the equivalent of drawing a rough sketch of something on paper to plan out what you're thinking. From this big sprite we can see how all the armor goes and everything. But of course, unless you're working in SVGA, a sprite this huge is probably not going to be used (it'd take up way too much of the screen). So you decide to chop it down a bit...Okay, about in half, down to 24x28. Now it's a more realistic size, and the programmer won't be laughing his head off and then kill you for expecting him to use huge sprites and keep the engine running fast. The main problem with this sprite, however, is that 24x28 is a weird number (you'd have to have, maybe 24x24 tiles or something...Using rectangular tiles gets hard to work with for complex tiles, and if you have a character sprite that's a certain size, you usually want to make your tiles close to that size (at least the width, height isn't as important) otherwise you have to have doors made of 4 tiles and such, which get aggrivating when making maps, and you can get messed up when you try to align things)). SO, you decide to whittle the sprite down some more...Down to good old 16x16.

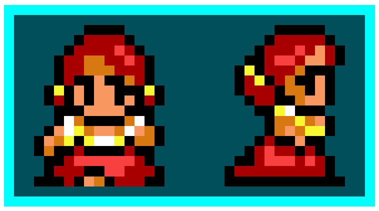
16x16 is the old "general" size of sprites. This is what Final Fantasy and tons of other RPGs used to use. It has something to do with sprite sizes that are powers of 2 being quicker to display or something...I don't really know, heh. But it was a nice number because it would allow somewhat large tiles to allow detail, but not too large to slow the engine down. Through time, it's just become a tradition to use 16x16 for some reason (the limits these days are pretty non-existant when it comes to this kind of thing...You could probably get away with using the first huge sprite on the left if you wanted and not have to worry). Anyway, I've redrawn him yet again, shrunk down to 16x16 on the right...So now he still "feels" like the large one, but he's nice and small. However, what happened to the sides of his armor? The 3 sections to his gloves? The triple boot thing? His belt buckle? They're all gone!

That's right, they ARE gone...We've run into the limit of sprite size. Now really, this isn't much of an issue these days, as I've said...As you can make sprites pretty much whatever size you want. However, having an understanding of how to work with tiny sprites will make working with large sprites not only MUCH easier, but it will help you develop a style and learn how to make them look better. Check out the middle size sprite...It's basically ugly (heh, I didn't intend for that, but the more that I look at it the more I hate it). The character is stumpy and off balance looking...It just looks cluttered, and it's hard to see what goes where. The reason for all this is that everything has an outline. EVERYTHING. Every piece of armor and such. And it looks bad because there's too much black crowding in places and in general it's just not working...

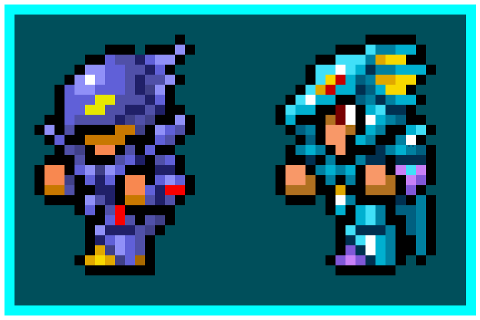
So the big question is "Well if I don't outline everything, how am I going to show different parts of the sprite?" The answer to that is to use colors strategically. If you're drawing a character with a cape that comes around the front, you don't need to outline the cape with black to split it apart from the clothes underneath. All you have to do is figure out what color you can use to contrast it the right way. Basically what you want is to SUGGEST details. Just like you don't have to show each individual finger to represent a hand, you don't have to show each individual section of clothing to represent it. You just need to suggest the feeling of it. And this is used everywhere...Though nowadays with sprite sizes being pretty much unlimited and most of it all switching to 3d anyway, it's not really a common thing to see anymore, and if it IS used, it's probably just a style choice of the artist. In the old days though, when it was necessary to have smaller sprites but still show off detail, it was used often. For instance, let's check out some sprites from Final Fantasy 2 and 3:



First up is the coolest character in any Final Fantasy game...Kain, the Dragoon Knight! Heh...Now check out that sprite...It looks good. You can tell he's got dragon-spikey-cool armor going on. But take a look at his leg, in the front view. He's got red (uhh...Pants? Spikes? I don't know, heh), and then blue boots...But they're not divided by a black line. And yet, you can still see there's a difference between them. This is a basic example of using color division (ooo, I made up a term, heh) in sprites. Also check out his feet. There's no black line along the bottom of them...They just suddenly "end". The reason is that there doesn't NEED to be a black line there. Because there are black lines on the sides of the leg, your eye automatically draws an "invisible" line connecting the left side to the right side, using the end of the colors as an edge. Now it may not SOUND like much, but that means an entire row of pixels was saved...That row can be used for doing details on the face or other parts of the body, instead of wasting it just to outline that foot, when it's not even necessary. Another small part would be the red "eyes" on his helmet. They're just red on blue, no black. A final section is the top of his helmet. On the front and back views, there's no black line on the very top. Again, that would take an entire row for it and it's not necessary. Your eye says that because the blue in the middle sticks up above the blue beside it, it's higher, and since it's got black surrounding it, it's also flat and a part of his helmet, and not just a random blue pixel flying by behind him. Kain isn't a great example of extreme color division (usually cloth requires it more than armor does because armor is so segmented and usually doesn't have much color difference on the sections), but he's cool, and so he gets to be on here, heheh...Next up:



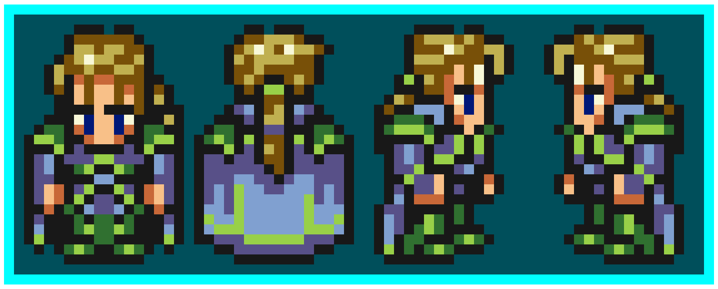
A random NPC from a town! I only grabbed these two views of her because the others didn't have much that I wanted to show. Now the MAJOR place that you'll see color division is hair. Check out her hair...There's no black line that divides the red of her hair from the peach of her skin...But you can tell her hair isn't just sunburnt skin, right? There's enough difference between the two colors that a black line isn't necessary. Plus if there was a black line, imagine what that would do to the eyes! You wouldn't be able to tell what's part of the eye and what's part of the hair! By using color divisions, the hair/face fall into the background, becoming less important than the eyes, which are used for expressions and such. Now check out her torso chunk (from the neck to the waist, including the arms). The only black lines there are the ones that outline the full piece...They're on the "outside" of the area. No black lines come in and cross the "inside". Again, there's no lines outlining her blouse and her skin, but you can tell she's wearing something, right? See, when you're working with bodies that are like 2 pixels high, you don't have the extra space to outline everything...And like I said before, sometimes outlining everything just makes the sprites look crowded with black, and the black can overpower the color, making your sprites look more jumbled. The last point is her hair, where there's the yellow tying thing (heh, I don't know what it's called, just go with me here)...You know what it is, and you know it's there, even though it's not outlined. Am I getting the point across? Heh...Now there ARE some black lines that cut through the inside, namely the line of the jaw and the waist...THESE are necessary, because they segment the character. If you have NO black lines dividing pieces, then it looks like a large colored blob of pixels...This is a mistake a lot of people make...Not ENOUGH division of the sections. Be careful on this...It generally just takes practise, learning what parts of the sprite to emphasize and what parts to blend. Next let's take a look at the battle sprites...



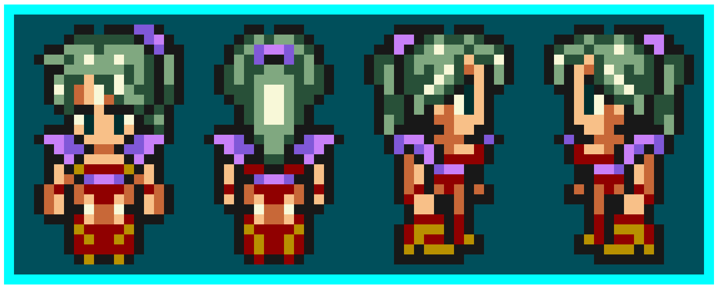
Cecil and Kain, snatched from the battle scenes of FF2. These sprites are 16x24, unlike the map sprites which are 16x16...Because of this, they're much more detailed...And YET...They still use color division...See, Mr. Artist gets a whole 8 more pixels to work with now. "Ooo!! That means I can put in all the black outlines!" Well, you COULD do that...But what you'd end up with is basically your 16x16 sprite, but with black outlines separating everything...And really, it wouldn't look much better than the 16x16 sprite, which means you might as well save your programmer some work and just use 16x16 sprites. "I don't want to do that! I want fancy sprites!" Okay then, listen up and study these sprites...

First up is Cecil on the left...Now the main part that sticks out is his helmet...Check it out, notably the piece that covers his eyes...Follow the black line. Follow...follow...oop! Where'd it go? It just stops! That's right. From just past the middle of his face, the line becomes a dark purple line, goes through and shapes his helmet, then attatches back up to the black line at the top. "So what was the point of THAT?" Well, it has to do with contrast. If you glance at his head, the first thing your eyes focus on is the front of it, his facial area. Then your eyes scan around to the back and take in the whole thing. The reason your eyes point to one spot first, is because of the collection of black that's there. If that purple line were black, it would stand out and fight the face for attention, which would make the face seem less important (I know, this all sounds hokey, but trust me here, heh...). Because it's purple, it fades in with the rest of the purple armor. Squint your eyes and look at the sprite a bit...The purple areas all seem to blend together, because they're similar, but the black areas still stand out on their own. Okay, so you don't have to get THIS technical and philisophical about the lines...But just keep in mind that you don't always need BLACK outlines. Now check out his arm (the one that's closest to us)...Again, you can see there's no black lines from the shoulder down to the hand...It's divided with colors, and you can see there are divisions. Also on his chest armor, his legs, and so on. It's all done with color separations...Dark colors used instead of thick black to divide things.

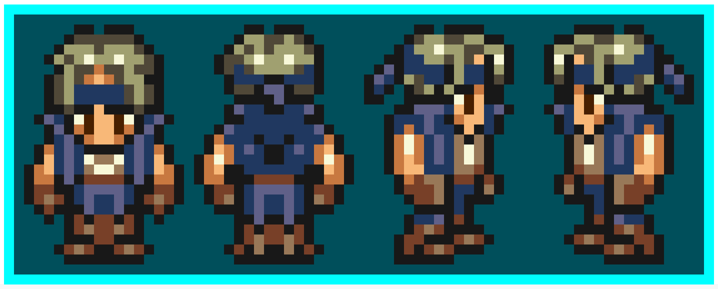
Now Kain, because he's also wearing armor, is similar to Cecil in where the colors are used. In fact, if you look close, his legs are exactly the same as Cecil's except that the colors are changed, heh...While we're on the subject of the legs...Check out Cecil's, then Kain's. Notice the difference? The bottom right corner on Cecil has an orange spot and a corner pixel. Now if you look at them, those TWO pixels change the look of the boot almost completely. Kain has rounded solid boots, whereas Cecil has an orange part sticking back (possibly his heel?) and the armor is sharp-cornered, like jagged. That orange pixel has no black outline...It's just an orange pixel, but it changes the look of the armor. Okay, so now check out Kain's helmet...The dragony fin part...The black outlines the back, and the inside is just alternating aqua ad yellow. There's enough difference in the colors that they're separate looking, and there's no black outlines on them, which would crowd up that section of the helmet and make the sprite look ugly. Kain has some more divisions (most notably on his forearms), and the black lines make his armor feel like it's solid on top...Cecil's purple armor, right on the wrists, seems more like a sleeve, while Kain's is a separate segment, thanks to the black line. Black can divide things up, but it can also make it hard to tell what's going on. Again, it'll take practise to learn how to use black like "the masters", heh. So now we'll check out Final Fantasy 3:



It's good old Edgar...Fully decked out in his fancy clothes. For a bit of useless information, the character sprites in FF3 are 16x24, the same size as the battle sprites in FF2. Also, in FF2 they switched between 16x16 sprites for the map and 16x24 sprites for the fights, whereas in FF3, they just use the 16x24 sprites for everything. I guess they made the battle sprites in FF2 and fell in love with their look or something, heheh...Anyway, these sprites use massive color division. In fact, oddly enough, the larger sprites use it more than the smaller sprites do, heh...The lesson from that? Using bigger sprites doesn't mean you don't have to still think about what you're doing. All those black lines you blob around each individual strand of hair could be used as shading or details...Think about it as you do it. So now, to get to Edgar...The main thing to check out is his hair...Specifically where it falls down on the face...Just dark strands, no black outlines. As well, note the very top of his head. He has a part in his hair...That one black pixel is missing. Logic tells us that if we want a bump down in the hair (for the part), we should have the black line bump down...But we don't need that. All we need is to eliminate a pixel and break up the line. When your eye is travelling along the top line of the head, it comes to the break, dips down until it finds color, and comes out again, following the shape of the pixels...That creates the dip in his hair, and doesn't take a big fat black pixel. Watch the green lining on his clothing too, it's just there, no black outlines...Black outlines would make the lining (especially on the cape in the back view) stand out too much and start to look like a separate piece of clothing/armor. On an interesting note, there IS no black line on here, technically. In FF3 they went with colored lines...If you get close up to the outlines, you can see a tinge of green in it. Because of this green, the sprite looks a bit more "realistic" than the FF2 sprites, which look like "drawings" because of the harsh contrast of black. This green softens the look out. The final note for Edgar (there are other things but I'm assuming that by this point you're already looking at the sprites and finding things on your own, for that's how you'll learn) is that there's no separation between his legs in the front view. This is, unfortunately, an inconvenience of using even numbers for sprite sizes (16 for the width). Programming-wise, calculations are faster if the numbers are even, so it was standard practise in the old days to use even numbers on the sprite sizes. The problem with this is that there's no middle column of pixels, and thusly no actual center point on the sprite...If the sprite had a middle column, you could have a line going up it at the legs to separate them evenly, and maybe even throw in a nose between the eyes...But because there are two middle columns, you can't have a line dividing legs unless you either shrink your whole sprite's width so that there's a column of blank pixels on one side, or you have uneven widths for legs. The double column problem also affects the faces, which is why if you check out Edgar, he's got one short eyebrow and one long one...If both eyebrows were long, it would turn into a unibrow, heheh. If both were short, he'd look happy-go-luck, and he's usually burdened with stuff...So the one pixel there makes it look sort of like he's trying to figure something out. Okay, that's enough of Edgar...Let's check out the next character, Terra:

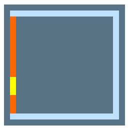


So by now you should have the usual "checkpoints" memorized...Check out her hair, the top of her head with the missing pixel, her legs, and her torso. Something to note, after you're done all that, is her chin...There's no black line across it, dividing it from the body. If you remember the NPC from FF2 above, there IS a black line...So why the difference? The NPC looks like she's an overweight woman, like a generic medieval mother. Terra, however, appears to be thinner and in better shape. Part of the reson for this is her chin. If you think about a fat person's head, there's usually a double chin dividing the head from the chest...And if you have some fat, and fold it, you get a crease...a dark line. So on the NPC, there's a dark line that separates her head more from her chest, making it feel more segmented, and more like there's a large flat line dividing it...a line of fat, basically. Now in Terra, she's thinner, so instead of a thick line, she's just got a few darker pixels. It shows that there IS a chin there, it's just not a very definate chin. A black line could also be used on characters with square jaws (usually big barbarian type people), to show the definition. The lack of a heavy line, much like a pointed chin, makes the character look more feminine. Now we'll move on to Locke:

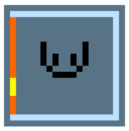


The main things in Locke are, like the others, the hair, the legs, the torso, the headband, etc. Now Locke seems to be more divided because he has that funky jacket on...But yet, if you look close, drawing a curved line following his body from one hand to the other, there are no black pixels! The color change is so drastic between the blue and the peach, however, that it feels like it's divided. Now if the dark blue on his jacket was replaced with black, his jacket wouldn't be a wide blue jacket, it would be a black jacket with a thin blue stripe going up it...Another plus of not overdoing the black on a sprite. On the back and side views, the tied parts of his headband are outlined with black, while the part around his head isn't...Why? Because the tied parts stick out from his head, and they're different segments of him...The headband is tight around his head, so it doesn't need to be defined separately. Now do you see where all this is going? If you haven't got it drilled into your head now, then let me state it again..."You do NOT need to outline everything." Now the next time you're playing some games, and see some sprites, check out where outlines are used and where color separations are used...You'd be surprised how many sprites DO use color separations.

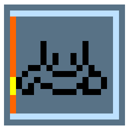
So now that you've read all the other chapters and have done some analyzing of your own, and thinking about how sprites are set up and all (if you haven't done any of this, then go do it because it will make things much easier for you when you get down to making a bunch of your own sprites), it's time to start actually making sprites. We're going to look at a few different styles of sprites, but we'll start off with the classic and most widely used one (for RPGs that is), the Final Fantasy 2 style sprite. Even if you're not interested in doing small 16x16 sprites, and you just want to learn to do bigger ones, it's still a good idea to read it all...You can pick up a lot of skills from doing small sprites that you can use when you're doing bigger sprites. Learn from everything you can. With that said, here we go with FF2 sprites:



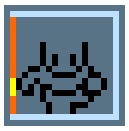
Now the first thing you have to know about FF2 sprites, is that they're all divided up in a certain way...This is a 16x16 box that you see above the text...Along the side is a line of orange, yellow, orange. Those lines are a guide of where everything goes. In an FF2 sprite, the waist down is given 3 pixels (the bottom orange guide), the torso from the neck to the waist is given 3 pixels as well (the yellow guide), and the head is given a massive 10 pixels (the top orange guide). If you talk to people about sprites you'll probably hear the term "SD" come up often...SD stands for "Super Deformed" and is used to describe scenes in anime when a character who's drawn normally will suddenly shrink down into a stubby sort of doll looking state where their head is about as big as their body...Anyway, that's what FF2 uses, and a lot of the beginning RPGs. This is because most of these were made in Japan, where SD is used in anime, and it's also good for expressions. In the west, we tend to want "realistic" looking characters, where the heads are small and in proportion to the bodies. The problem with this in the old days, however, is that when you had a sprite box that was only 16x16 to work with, a realistic head would have to be about 2 pixels wide and 4 tall...While it's in proportion, it doesn't look very good because you end up with faceless characters, and the most important part of a character is its face, because the player identifies with it more. The use of a huge head in sprites was common because it allowed for good expressions as well...With a huge head, you could have them bow their head, shake their head, blink in amazement, grow huge shocked eyes, close their eyes, yawn, shout, and so on. This gives the characters more "life" and makes them more interesting...With a small "realistic" head, you're stuck with maybe changing the shading on the 6 pixels and that's about it. For doing realistic proportions, it's best to use larger sprites (like Phantasy Star IV did). Anyway, now you know why everything was SD for so long, heh...This is all just my speculation however, I could be totally wrong.



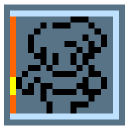
So we'll start with the basic head. This is, pixel for pixel, the basic structure of an FF2 sprite's head...What a lot of artists will do, if they have to make a lot of sprites, is make a basic "dummy" one that would be an average person and then modify that for each sprite. It's a good idea if you have to make a lot of characters (especially townspeople)...But make sure that you modify the dummy enough to make each one look individual and not just color swapped like they did with Mortal Kombat, heh...So the basic head, two pixels in between the eyes, the eyes are two pixels tall, there's a single pixel space between them and the sides of the head, as well as between them and the bottom of the jaw. All in all it looks reasonably proportioned. Note that it doesn't take up the entire space it's allowed...A lot of the room is left for the hair because hair is important, heh...When you have a 16x16 sprite, and you have only 6 pixels for a body, you don't have a lot that differentiates one sprite from another, except for colors and a few pixels. To make the characters stand out from eachother more, everyone usually has an extremely distinct hairstyle. Hair is very versatile and can be short, long, parted on a side, parted in the middle, tied in a ponytail, tied in pigtails, wrapped into a bun, shaved, spiked, and so on...It's a good way of separating one character from another, so it gets a lot of space on the sprite. If you were to fill in all of your characters with black so they were shadows, the only real way of telling them apart would be the shape of their hair. Anyway, moving on, notice that the jaw comes down to the bottom of the top orange guide.



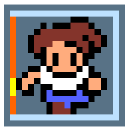
Now we add the torso...I've set it up so that one fist is forwards and one is back. In older games, sprites would constantly switch from the left foot forwards frame to the right foot forwards one...This was a way of avoiding making a third "standing" frame where both feet are in the middle and the hands are at the sides...There are things I want to talk about in this stance, so I'm going with a frame that's already in the middle of walking. Note I didn't separate the arm from the torso...You can if you want (and it'll make it look like a jacket sort of deal), but I'm not going to for this one...This guy will just be a generic shirt and pants character. The yellow guideline defines the distance from the bottom of the jaw to the waist, and really you only get 2 pixels for the chest because of the waistline (unless it's, say, a dress or a shirt that's untucked or something and you don't need that line)...I mention this because if you check out his shoulders, you can see they go up into the orange. The reason for this is that the world you're looking at in the game is not a straight on front view, and it's not a straight on above view, it's a "top down" view ("top down" is usually used to describe the angle of Final Fantasy style games). Basically you're looking down at them at around 45 degrees, which is why you can see the front and tops of houses and such (45 degrees is usually used to describe isometric games (like Diablo, where the tiles run diagnol instead of straight up and down) because a lot of people don't understand what "isometric" is (you'll come across it all the time in a drafting class, heh), so that's why "top down" is used instead of "45 degree" for FF style stuff...confused? Heheh...). Now because it's a top down view, you're going to see part of the top of the characters shoulders and they're going be back more "under" his head...You have to be thinking about this in 3d in your head to get it...The more above you're looking down at something, the more of the shoulders you see and the farther under the head they go. A lot of people make the mistake of doing a flat front view of a character when the maps are all top down...Honestly, it doesn't matter TOO much...You can get away with it...But it stands out a lot more these days. Back during the NES days, they warped perspective so unbelievably that it's hilarious to look at now...But at the time it didn't matter because the graphics were crummy in general and it was the gameplay people wanted anyway. Check out a dungeon map in the NES Zelda sometime...The room is viewed from straight above, but Link is viewed from a top down type view, and the statues and various junk is all viewed from a front view, heh...



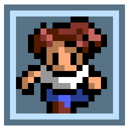
Now we've got some legs on this guy. Again, note that they're in a walking position, not a normal standing one...There are some things I have to point out about this...I didn't draw a black line along the bottom of the foot because I need that bottom row of pixels to show a foot and your eye draws an imaginary line from the left black pixel to the right black pixel on the leg anyway (if you read the earlier chapter, you already know this...If not, go read it). The character's right leg (the one that's forward) doesn't go straight down, but it goes in a bit...The reason for this is in the animation process. If you had the legs just go straight down when they were forwards, and straight up when they were back, the character would just look like he was stomping left and right as he went along...By curving the leg in a bit, it gives them a more natural look to their walk. The leg that's back is just a row of pixels basically. Right now it looks like a stubby hip of some sort...We're going to show that that leg is back when we get into the colors. Also note that the waist is NOT a straight line, it curves, and the legs go up into the yellow guide to connect to it...This is because it's a top down view. A big mistake people do is have a straight waist...It stiffens the character out a lot and gives a sort of cardboard cut-out look, which looks really bad. Curving the line a bit gives them a little more depth, like looking down at a cylinder (think 3d again).



Now he's finally got hair. Hair is something you'll probably spend the most time on...A single pixel can mess up the hair, or make it look great, so you'll be juggling pixels constantly trying to get a good shape. This guy will just have hair that goes to the side...I threw in a missing pixel in the top of his hair to give him parted hair, just for demonstration, heh...Note that his hair doesn't completely cross his face with a black line...With a full black line, it can start to separate the hair from the head too much (like how the torso and legs look like separate segments), so sometimes you want to just use color separation to split it up. I did use a bit of black under the parts that flop down because I want to show that they come out away from his head and aren't slicked down with gel or anything. The general hairline is as from from the eyes as the jaw is, there's a one pixel row dividing them. It's not mandatory as different characters will have different hairstyles, but it's just a guide to go with for starting. At first you'll probably want to be working with color AND black at the same time on the hair to figure out where you don't need black pixels dividing things, because it's easier to tell...I've made a couple zillion sprites so I just do the outline and figure it out in my head, heheh...This guy's hair goes right up to the top of the sprite, but not every sprite has to...Short people or kids wouldn't, nor would bald people because the actual head only goes up a pixel or two under the hair. Of course, if you had shorter people or kids, you'd probably mess with all the proportions and give only 4 pixels for the body instead of 6.

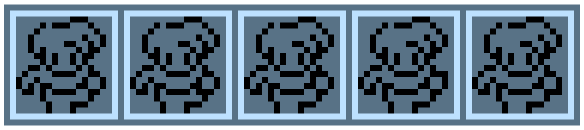


Here I've filled in the sprite with flat colors (no shading). Take note of his hair, shirt, and shoes. I've used the reddish-brown of his hair to define part of his head, and because there's no black under it, it seems like it's more flat, closer to his head than his flopping down hair on the other side. Also I left a blank pixel where the black line separates for his part...If I had filled in that pixel with the hair color, it would create the effect that he's got a strand of hair that sticks up there...Filling it in with a black pixels makes his hair look flat there, and leaving it blank gives the part. One pixel can make a lot of difference. His shirt seems to be a muscle shirt (no sleeves)...To turn it into a jacket I would have black pixels that outline in, and to turn it into a t-shirt, I would put a white pixel on his arm, joined to the rest of the white. That one white pixel will create a "sleeve" on his arm. To give him long sleeves, I just need to fill his arm in with white except for the last pixel. I'll demonstrate a bunch of this stuff later.



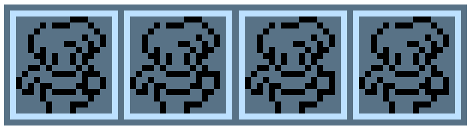
Now we throw in the shading and we've got our finished sprite. When you're shading your sprites, try to keep the light at a consistant point...Right now the light it above and to the right of the sprite, so all the shadows are on the left and bottom sides. A reflex way of shading that a lot of people do is to shade the left and right, leaving the middle brighter...This means the light sources is right in front of them...It's okay to do, but it can make the sprite look a bit dull because they have no "left and right", they just have "center"...It's weird to explain, but it makes the sprite more symmetrical and uninteresting...With shading on just one side, it gives the sprite a definate left and right. Anyway, there's no real "standard" for how dark the shade should be. You just have to play with what looks good. There are some things you can keep in mind though...If there are bright lights (say, middle of the day, standing by a fire, etc.), the shadows will be very dark, high contrast from from the normal color. If the lights are dim (inside a house, night time, etc.), the contrast will be a lot less. I'm not saying you need different light on your sprites depending on where they are, just keep in mind what sort of effect your shading can have. Another good thing to keep in mind is that if you squint your eyes and look at your sprite and you can't tell the light shading from the dark shading, then you can probably use some more contrast. If you can't see a difference, then why bother shading, right? This comes up a LOT when people use yellow...They just slightly dim it and it's hard to even notice. The face, being an important part of the character in showing expressions and all usually has less contrast than the rest of the sprite...This is simply because if you have a lot of contrast, it can start to break up the colors and becomes distracting to the eye. If you have just a big of contrast, you can see that it's shaded, but it will fit together a bit...Granted, this contradicts my last point, but like I say, there's no "law" in how to do this stuff...Go with what looks good. Some games will even have the face completely one color, so that the features (well, the eyes, heh) stand out well. I've used pretty high contrast in the skin on this sprite, and it's a little bit distracting. The last part to check out is his lower section. The leg that's behind is completely dark and you can't see the foot.

The black outline below it helps set it further behind...That bottom line could be just dark brown instead, but it would look more like he's just raising his leg. The leg that's in front HAS to have more light shading on it, so it looks like it's in front. His boot is separated just with color. Also check out his arms...To have his forwards arm look like it's coming forwards, there's only one pixel of shading on it...The more shading, the further back it looks...To make his back arm look more like it's behind him, I should have shaded the entire arm with the dark shade. I left it a bit light though, and it looks like it's more just at his side than it is behind him. There's more shading on it than there is on his fist though, so it looks behind the point where his fist is, which is good. I've said before that a single pixel can make a big difference, and now we're going to look at that:

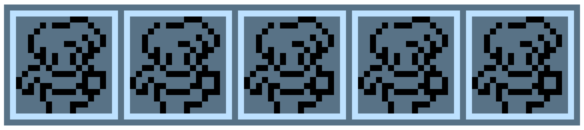


So now you're probably thinking "What the...?" At first glance, all of those sprites look the same...But they all have one pixel of difference...Check out the forwards fist. I'll explain what this one measly pixel can do...The basic fist in the first one is just a round ball basically, no corner pixels. As a result it looks like a delicate fist, like someone with small hands would have. You don't really know much about it, it could be rotated in any direction and you wouldn't know it changed. In the second one, there's a corner pixel in the bottom left of the fist...Now the fist looks like it's dipping down pointing towards his legs. He could even be wading through water or climbing over a rock or something with this fist because the point is down...In the third sprite, the corner pixel is at the top left. Now it looks like he's pointing towards his face, but with a finger because the bottom right pixel is still rounded which means it can't be an elbow since an elbow is pointed. So basically he's going "Who, me?" in a way. The fourth sprite has the pixel in the top right and this makes it look sort of like he's motioning

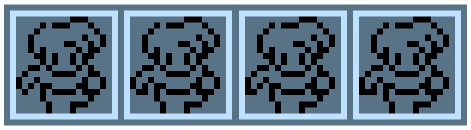
"Come on and get me!" with his hand...He has no elbow and his fingers are up by his head but out a bit. He's sort of doing an uppercut out to the side...or flipping the player off, heh. The last sprite has the pixel in the bottom right, and now it look like an elbow because that's a natural position for the elbow to be in (in the other ones, like the third one, you can't really mistake the pixel for an elbow because that would be extremely weird having the elbow up there, so your brain assumes it's a finger or hand)...Now with the curve, the curve looks like a fist, opposite the pointed elbow...So now he's doing an inward uppercut. Maybe starting out a dragon punch or something, heheh. Now that's just ONE pixel:



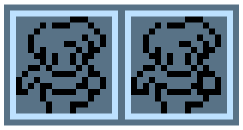
Here I start messing with two pixels...In the first sprite, he's got the flat side facing in. Two pixels make the fist look much flatter, like maybe he's slapping or something...You know, so he's got a definate flat edge (a very tight fist maybe). in the first sprite, he's got it inside and it just looks pretty normal. But say he wants to hold something like a vase up for everyone to see his strength. So now he rotates his pixels to the top...Now his flat edge is on the top and it looks like he's holding his hand up, and he could be balancing a pot or a lamp or something on it. In the third one he's got the flat parts out which looks...well...pretty bad. Heh...The only way to justify this one would be if he had some sort of square armor forearm piece and he's got his arm at the side, and you'd have to show that by color usage when you were doing the colors...For the last one, with the flat side on the bottom, he could be playing basketball or something with that arm, or squishing someone. Now this is all interesting and everything, but it's only TWO pixels, heh...Let's try out three:



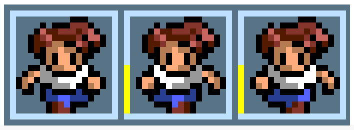
Now three pixels does something different from two pixels. Three pixels starts to add "weight" to the fist...Where one pixel is a delicate fist, three pixels starts turning into a hamfist, like an ogre or big tough blacksmith would have. In these sprites, they follow the same look as the one pixel ones do, but now they look like big meaty hands, ready to crush something. The last sprite, with all four corners filled in makes it look like a straight on view of the fist, as if he's punching at you. Now we'll just take a quick glance at the other arm:



Now we're looking at the back arm, and checking out the pixels on it. In the first sprite, it has no corner pixels and it looks like maybe a tentacle or something...A loose hand, I guess. No real definition...It acts pretty much like a tail, just flailing there. In the second one, his fist in in towards his body, like he's flexing his muscles making a half circle with his arm. In the third one, the pixel is on the outside and it makes it look more like his hand is way out to the side, unlike the one before it where his hand feels closer to his body. The second sprite is like he's a tough guy walking around, and the third sprite is more like he's running around with his arms swinging in a running pose. In the last one he's got a flat line for the bottom of the fist and it looks like he's playing basketball again, or he's got a gun for an arm, some sort of armor, a large hamfist, etc. Now to just look at the two semi-extremes for a second:



The guy on the left seems light and agile with his arms, not a lot of definition in the fists...The guy on the right is more buff looking because he's got huge square fists going. Now you should understand what I mean when I say that one pixel can make so much difference...That's what's interesting (to me) about working with small sprites like this. Juggling pixels around to figure out how to change someone's expression or movement with just a pixel or two. A few pixels can make someone look huge like an ogre, or dainty like a fairy. Those few pixels of difference make your sprites look different and give them more life. This is why you can start with a dummy, but you have to make sure that you alter pixels on it to change the feeling of the body. I just used the fists as an example for pixels, but you can change a lot with single pixels, especially facial expressions. Work them them, play around and figure out what looks best for your sprite's pose. Practise, heh...Practise a lot, and soon it'll become natural.



Chapter 8

Advanced modding

8.1)actors templates

AHumans are a subclass of units derived from the Actor class. Their name is an amalgam of *Actor* and *Human.*

AHumans are bipedal, with two arms for device manipulation. They are also commonly equipped with a Jetpack, granting them higher mobility than those not equipped with such equipment. The choice of adding a jetpack or even limbs is completely optional, and depends on the mod's requirements and needs. However, a head is necessary for the actor to live.

AHumans are capable of equipping, dropping, picking up, and storing devices in their inventory, unless their foreground arm is missing. Lacking a background arm may affect the accuracy of HDFirearms used, and will disable them from using multiple devices simultaneously, such as a shield and one-handed weapon, or dual wielding. Missing one leg will affect walking ability, generally surprisingly little. Missing both legs disables their walking ability entirely.

They can board transport craft.

```
AddActor = AHuman
	PresetName = Dummy
	Description = Standard dummy soldier.  Quite resilient to impacts and 
falls, and very agile.
	AddToGroup = Actors
	Mass = 32
	GoldValue = 80
	HitsMOs = 1
	GetsHitByMOs = 1
	SpriteFile = ContentFile
		FilePath = Dummy.rte/Actors/Dummy/TorsoA.bmp
	FrameCount = 1
	SpriteOffset = Vector
```

```
                X = -4
                Y = -16
        AngularVel = 0
        EntryWound = AEmitter
                CopyOf = Wound Bone Entry
        ExitWound = AEmitter
                CopyOf = Wound Bone Exit
        AtomGroup = AtomGroup
                AutoGenerate = 1
                Material = Material
                        CopyOf = Civilian Stuff
                Resolution = 4
                Depth = 0
        DeepGroup = AtomGroup
                AutoGenerate = 1
                Material = Material
                        CopyOf = Civilian Stuff
                Resolution = 6
                Depth = 3
        DeepCheck = 0
        BodyHitSound = Sound
                CopyOf = Bone Crack
        PainSound = Sound
                CopyOf = Bone Crack
        DeathSound = Sound
                CopyOf = Bone Crack
        DeviceSwitchSound = Sound
                CopyOf = Device Switch
        Status = 0
        Health = 100
        ImpulseDamageThreshold = 2600
        AimAngle = 0
        AimDistance = 30
        Perceptiveness = 0.7
        CharHeight = 100
        HolsterOffset = Vector
                X = -6
                Y = -8
        Head = Attachable
                CopyOf = Dummy Head A
                ParentOffset = Vector
                        X = -1
                        Y = -13
        Jetpack = AEmitter
                CopyOf = Jetpack
                ParentOffset = Vector
                        X = -6
                        Y = -1
```

```
	JumpTime = 4 // Secs
	FGArm = Arm
		CopyOf = Dummy Arm FG A
		ParentOffset = Vector
			X = 0
			Y = -8
	BGArm = Arm
		CopyOf = Dummy Arm BG A
		ParentOffset = Vector
			X = 4
			Y = -9
	FGLeg = Leg
		CopyOf = Dummy Leg FG A
		ParentOffset = Vector
			X = 0
			Y = 1
	BGLeg = Leg
		CopyOf = Dummy Leg BG A
		ParentOffset = Vector
			X = 2
			Y = 1
	HandGroup = AtomGroup
		CopyOf = HandGroup
	FGFootGroup = AtomGroup
		CopyOf = Foot
	BGFootGroup = AtomGroup
		CopyOf = Foot
	StrideSound = Sound
		CopyOf = Robot Stride
	StandLimbPath = LimbPath
		PresetName = Dummy Stand Path
		StartOffset = Vector
			X = 1
			Y = 17
		StartSegCount = 0
		SlowTravelSpeed = 0.1
		NormalTravelSpeed = 0.5
		FastTravelSpeed = 1.5
		PushForce = 1800
	StandLimbPathBG = LimbPath
		CopyOf = Dummy Stand Path
		StartOffset = Vector
			X = 4
			Y = 17
	WalkLimbPath = LimbPath
		PresetName = Dummy Walk Path
		StartOffset = Vector
			X = 10
```

```
				Y = -2
			StartSegCount = 3
			AddSegment = Vector
				X = 0
				Y = 2
			AddSegment = Vector
				X = 0
				Y = 2
			AddSegment = Vector
				X = 0
				Y = 5
			AddSegment = Vector
				X = 0
				Y = 5
			AddSegment = Vector
				X = -6
				Y = 4
			AddSegment = Vector
				X = -4
				Y = 0
			AddSegment = Vector
				X = -4
				Y = 0
			AddSegment = Vector
				X = -4
				Y = 1
			AddSegment = Vector
				X = -3
				Y = 1
			AddSegment = Vector
				X = 0
				Y = -2
			SlowTravelSpeed = 1.5
			NormalTravelSpeed = 2.6
			FastTravelSpeed = 4.5
			PushForce = 8000
		CrouchLimbPath = LimbPath
			PresetName = Dummy Crouch Path
			StartOffset = Vector
				X = 10
				Y = 0
			StartSegCount = 0
			SlowTravelSpeed = 0.1
			NormalTravelSpeed = 0.5
			FastTravelSpeed = 1.5
			PushForce = 5000
		CrawlLimbPath = LimbPath
			PresetName = Dummy Crawl Path
		StartOffset = Vector
			X = -12
			Y = -8
		StartSegCount = 2
		AddSegment = Vector
			X = 12
			Y = 0
		AddSegment = Vector
			X = 8
			Y = 2
		AddSegment = Vector
			X = 0
			Y = 8
		AddSegment = Vector
			X = 0
			Y = 10
		SlowTravelSpeed = 1.5
		NormalTravelSpeed = 1.8
		FastTravelSpeed = 4.5
		PushForce = 8000
	ArmCrawlLimbPath = LimbPath
		PresetName = Dummy Arm Crawl Path
		StartOffset = Vector
			X = -8
			Y = -5
		StartSegCount = 2
		AddSegment = Vector
			X = 0
			Y = -4
		AddSegment = Vector
			X = 3
			Y = -3
		AddSegment = Vector
			X = 4
			Y = 0
		AddSegment = Vector
			X = 4
			Y = 4
		AddSegment = Vector
			X = 0
			Y = 10
		SlowTravelSpeed = 1.5
		NormalTravelSpeed = 1.5
		FastTravelSpeed = 4.5
		PushForce = 6000
	ClimbLimbPath = LimbPath
		PresetName = Dummy Climb Path
		StartOffset = Vector
```

```
                X = -8
                Y = -10
        StartSegCount = 6
        AddSegment = Vector
                X = 0
                Y = -4
        AddSegment = Vector
                X = 3
                Y = -3
        AddSegment = Vector
                X = 4
                Y = 0
        AddSegment = Vector
                X = 4
                Y = 4
        AddSegment = Vector
                X = 6
                Y = 8
        AddSegment = Vector
                X = 0
                Y = 14
        AddSegment = Vector
                X = -5
                Y = 5
        AddSegment = Vector
                X = -3
                Y = 0
        SlowTravelSpeed = 1.0
        NormalTravelSpeed = 1.5
        FastTravelSpeed = 4.5
        PushForce = 5000
JumpLimbPath = LimbPath
        PresetName = Dummy Jump Path
        StartOffset = Vector
                X = 0
                Y = 8
        StartSegCount = 3
        AddSegment = Vector
                X = 0
                Y = -10
        AddSegment = Vector
                X = 0
                Y = 4
        AddSegment = Vector
                X = 0
                Y = 4
        AddSegment = Vector
                X = -8
                Y = 14
        SlowTravelSpeed = 3
        NormalTravelSpeed = 6
        FastTravelSpeed = 7
        PushForce = 5000
DislodgeLimbPath = LimbPath
        PresetName = Dummy Dislodge Path
        StartOffset = Vector
                X = 2
                Y = -10
        StartSegCount = 0
        AddSegment = Vector
                X = 0
                Y = 6
        SlowTravelSpeed = 1.5
        NormalTravelSpeed = 2.5
        FastTravelSpeed = 4.5
        PushForce = 10000
AddGib = Gib
        GibParticle = MOSRotating
                CopyOf = Dummy Rib Cage Gib A
        Offset = Vector
                X = -2
                Y = -3
        Count = 1
        Spread = 0
        MinVelocity = 0
        MaxVelocity = 0
```

ACrabs are quadrupedal actors derived from the Actor class. Their name is an amalgam of *Actor* and *Crab*, although real crabs typically have more than four legs. The term crab is used because this actor type was originally made for crabs, which are depicted in CC as quadrupedal, likely for the sake of simplicity.

An ACrab has the benefit of added ground mobility, having four legs and no upright constraints, but they are not typically equipped with jumping or jetpack abilities. ACrabs are capable of having a jetpack, but it is uncommon. Their lack of upright constraints affects their jetpack use.

Their angle of fire can be limited, making upper and lower bounds to the angle they can aim (like real-world tanks). However, the bounds are not in relation to the rotation of the actor.

Instead of arms like AHumans have for operating devices, ACrabs have a turret with a mounted device. This limits them to that single predetermined device. They can not switch devices, pick up devices, or drop devices. Their one and only device can be and generally is destructible. While they can hold devices in their inventory, they cannot access them, they will only serve to weigh them down, and they will only be retrievable upon the ACrab gibbing. ACrabs oddly do not suffer from recoil forces, this was likely an oversight in the creation of their means of handling a device.

```
AddActor = ACrab
	PresetName = Dreadnought
	Description = Armored tank on 4 legs.  Armed with a machine gun and covered with multiple layers of armor.
	AddToGroup = Actors
	Mass = 21.71
	GoldValue = 200
	HitsMOs = 1
	GetsHitByMOs = 1
	SpriteFile = ContentFile
		FilePath = Dummy.rte/Actors/Dreadnought/MountMobileA.bmp
	FrameCount = 2
	SpriteOffset = Vector
		X = -6
		Y = -6
	EntryWound = AEmitter
		CopyOf = Leaking Machinery Light
```

```
        ExitWound = AEmitter
                CopyOf = Leaking Machinery Light
        AtomGroup = AtomGroup
                AutoGenerate = 1
                Material = Material
                        CopyOf = Military Stuff
                Resolution = 4
                Depth = 0
        DeepCheck = 0
        BodyHitSound = Sound
                CopyOf = Bone Crack
        PainSound = Sound
                CopyOf = Bone Crack
        DeathSound = Sound
                CopyOf = Bone Crack
        DeviceSwitchSound = Sound
                CopyOf = Device Switch
        Status = 0
        Health = 100
        ImpulseDamageThreshold = 2100
        AimAngle = 0
        AimRange = 0.5
        AimDistance = 30
        CharHeight = 110
        Turret = Turret
                CopyOf = Dummy Turret Large
                ParentOffset = Vector
                        X = 0
                        Y = -5
        LFGLeg = Leg
                CopyOf = Dummy Crab Leg FG
//              CopyOf = Dummy Leg FG A
                ParentOffset = Vector
                        X = -5
                        Y = 1
        LBGLeg = Leg
                CopyOf = Dummy Crab Leg BG
//              CopyOf = Dummy Leg BG A
                ParentOffset = Vector
                        X = -5
                        Y = 1
        RFGLeg = Leg
                CopyOf = Dummy Crab Leg FG
//              CopyOf = Dummy Leg FG A
                ParentOffset = Vector
                        X = 5
                        Y = 1
        RBGLeg = Leg
```

```
		CopyOf = Dummy Crab Leg BG
//		CopyOf = Dummy Leg BG A
		ParentOffset = Vector
			X = 5
			Y = 1
	LFootGroup = AtomGroup
		CopyOf = CrabFootGroup
	RFootGroup = AtomGroup
		CopyOf = CrabFootGroup
	StrideSound = Sound
		CopyOf = Robot Stride
	LStandLimbPath = LimbPath
		PresetName = Dummy Crab Stand Path Left
		StartOffset = Vector
			X = -6
			Y = 6
		StartSegCount = 0
		SlowTravelSpeed = 0.1
		NormalTravelSpeed = 0.5
		FastTravelSpeed = 1.5
		PushForce = 1800
	LWalkLimbPath = LimbPath
		PresetName = Dummy Crab Walk Path Left
		StartOffset = Vector
			X = -13
			Y = -12
		StartSegCount = 4
		AddSegment = Vector
			X = 10
			Y = 0
		AddSegment = Vector
			X = 5
			Y = 8
		AddSegment = Vector
			X = 0
			Y = 3
		AddSegment = Vector
			X = 0
			Y = 3
		AddSegment = Vector
			X = 0
			Y = 3
		AddSegment = Vector
			X = -2
			Y = 2
		AddSegment = Vector
			X = -13
			Y = 0
		SlowTravelSpeed = 1.5
		NormalTravelSpeed = 2.5
		FastTravelSpeed = 4.5
		PushForce = 5500
	LDislodgeLimbPath = LimbPath
		PresetName = Dummy Crab Dislodge Path Left
		StartOffset = Vector
			X = -1
			Y = -10
		StartSegCount = 0
		AddSegment = Vector
			X = 0
			Y = 6
		SlowTravelSpeed = 1.5
		NormalTravelSpeed = 2.5
		FastTravelSpeed = 4.5
		PushForce = 10000
	RStandLimbPath = LimbPath
		CopyOf = Dummy Crab Stand Path Left
		PresetName = Dummy Crab Stand Path Right
		StartOffset = Vector
			X = 6
			Y = 6
	RWalkLimbPath = LimbPath
		CopyOf = Dummy Crab Walk Path Left
		PresetName = Dummy Crab Walk Path Right
		StartOffset = Vector
			X = -2
			Y = -12
	RDislodgeLimbPath = LimbPath
		CopyOf = Dummy Crab Dislodge Path Left
		PresetName = Dummy Crab Dislodge Path Right
		StartOffset = Vector
			X = 2
			Y = -10
	AddGib = Gib
		GibParticle = MOSRotating
			CopyOf = Gib Metal Grey Small B
		Offset = Vector
			X = 0
			Y = -1
```

ACDropShips are a subclass of actors that are ships that deploy cargo by dropping it. Their name is an amalgam of *Actor*, *Craft*, and *Drop Ship*.

ACDropships have 2 engines that are used for primary thrust and stabilization. The two stabilize by emitting more or less based on the tilt of the craft. When Up is pressed, both engines thrust more. When Down is pressed, both engines thrust less. With Right is pressed, both engines tilt clockwise and increase power, not affecting lift significantly, but adding rightward thrust. Left is the opposite of right. When one engine is destroyed, stabilization is almost always lost, resulting in a spinning plummet as the remaining engine still thrusts, ending in an gibbing, either due to impact or auto-scuttling.

To support in stabilization, ACDropShips also have retro thrusters, which actually defy the lift force, but greatly help in keeping the ship stable. While a dropship can be made without retro thrusters, it will likely roll over and lose balance when attempting much horizontal movement. Their help in stabilization can even make a single engine dropship possible, if used right.

Being a type of craft, ACDropShips can hold actors and devices in their inventory, and can eject them and board them through exits. The exit opening for dropships is visually represented by two attachable doors rotating apart to open. These doors can be used as armor for the bottom of the craft, and their loss only has no functional detriment. The exits, when open and not still dropping, have 'tractor beams' of variable length, width, and power, that can pull actors and devices in to the exit. They are visually represented by dotted yellow lines, with dots moving inward. Dots moving outward is a sign that cargo is still being dropped. Crafts can be flown above the top of the level to be returned to Trade Star Midas, selling the for the value of them and their contents, which is affected by health. Crafts also generally gib into a lethal mess of falling debris, killing occupants and actors below. Crafts also have a scuttling ability, accessibly through the pie menu, which self destructs them. When a craft senses that it is as good as dead, it automatically scuttles. This sense is currently flawed.

```
AddActor = ACDropShip
	PresetName = Drop Ship
	Description = Heavily armored aerial transport. Very reliable and
stable.
	AddToGroup = Craft
	Mass = 1300
	HitsMOs = 1
	GetsHitByMOs = 1
	SpriteFile = ContentFile
		FilePath = Dummy.rte/Crafts/Dummy Dropship/DropshipHullA.bmp
	FrameCount = 1
	SpriteOffset = Vector
		X = -67
		Y = -26
	EntryWound = AEmitter
		CopyOf = Dent Metal
	ExitWound = AEmitter
		CopyOf = Dent Metal
	GoldValue = 110
	AtomGroup = AtomGroup
		AutoGenerate = 1
		Material = Material
			CopyOf = Military Stuff
		Resolution = 10
		Depth = 0
	DeepGroup = AtomGroup
		AutoGenerate = 1
		Material = Material
			CopyOf = Military Stuff
		Resolution = 20
		Depth = 4
	DeepCheck = 1
	BodyHitSound = Sound
		CopyOf = Metal Body Blunt Hit
	Status = 2
	Health = 100
	ImpulseDamageThreshold = 9000
	StableVelocityThreshold = Vector
		X = 220
		Y = 220
	CharHeight = 100
	LThruster = AEmitter
		CopyOf = Dummy Dropship Engine A
		ParentOffset = Vector
			X = -53
			Y = -12
		Mass = 100
	RThruster = AEmitter
```

```
		CopyOf = Dummy Dropship Engine B
		ParentOffset = Vector
			X = 53
			Y = -12
		Mass = 100
	ULThruster = AEmitter
		CopyOf = Dummy Dropship Retro Thruster
		ParentOffset = Vector
			X = -23
			Y = -22
	URThruster = AEmitter
		CopyOf = Dummy Dropship Retro Thruster
		ParentOffset = Vector
			X = 23
			Y = -22
	LHatchDoor = Attachable
		CopyOf = Dummy Dropship Door A
		ParentOffset = Vector
			X = 0
			Y = -12
	RHatchDoor = Attachable
		CopyOf = Dummy Dropship Door B
		ParentOffset = Vector
			X = 0
			Y = -12
	HatchDoorSwingRange = Matrix
		AngleDegrees = 28
	HatchDelay = 250
	HatchOpenSound = Sound
		AddSample = ContentFile
			Path = Base.rte/Actors/Rockets/HatchOpen.wav
	AddExit = Exit
		Offset = Vector
			X = 0
			Y = 28
		Velocity = Vector
			X = 0
			Y = 4
		Radius = 22
		Range = 40
//	ExitInterval = 1250;
	CrashSound = Sound
		CopyOf = Metal Body Blunt Hit Large
	CanLand = 0
	GibImpulseLimit = 18000
	GibWoundLimit = 26
	GibSound = Sound
		CopyOf = Ship Explosion
	AddGib = Gib
		GibParticle = AEmitter
			CopyOf = Fuel Fire Trace Gray
			PresetName = Ship Explosion
			LifeTime = 175
		Count = 8
		Spread = 2.25
		MaxVelocity = 20
		MinVelocity = 5
		LifeVariation = 0.25
```

8.2)how to make correct walkpaths

Things you'll need:

Calculator - Optional
MSPaint - or any other program that shows offsets
NotePad - Obviously
Registered version of CC - You'll need the actor veiwer

Lets take a look at the walkpath code:

Code:
```
StartOffset = Vector
```

Defines the 0,0 point for future offsets, its start offset is the actor's hip joint.

Code:
```
StartSegCount = N
```

Causes the game to skip the first N AddSegment lines when not needed, (N being any number larger than zero) so actors can climb steep hills without having to raise their foot all the time. (Only when climbing)

Code:
```
AddSegment = Vector
```

Adds a segment in the walk "animation", will be explained in the next sections.

Code:
```
SlowTravelSpeed =
NormalTravelSpeed =
FastTravelSpeed =
```

Not sure what slow and fast do, but NormalTravelSpeed is the speed the actor moves his legs. (The others can be ignored, they're a feature Data wanted to add to CC)

Code:
```
PushForce =
```

Don't know...

How do we start:

Start by copying the coalition light's torso code into a new a .ini, scroll down to the walkpath area and delete all the AddSegment lines. **Do not erase the StartSegCount and StartOffset lines!**
Change StartSegCount to 0. (We won't be using this for this tutorial, I might add a section about this var in the future)

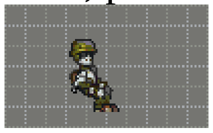
Open the actor veiwer and start working, reload actor data whenever you make a change:

Now we begin by adding one add segment like so:

Code:
```
      AddSegment = Vector
         X = 0
         Y = 5  //You don't normally have to use
five, I will in this tut
```

This is the first part of the walking animation, our actor will his foot by 5 pixels.

Now, press the move right key and have him raise his foot and take a pic:



Now we want out actor to move his foot backwards and take it forward again so we draw a dotted "line" following the path like so:

(The first dot should be in the center of the foot)

Now here comes the part where I think most people fail:
Look at the offset of the 1st dot: 50,50
And now the 2nd: 48,55
So the X drops by 2 and the Y gets added 5 so your next segment should look like this:

Code:
```
      AddSegment = Vector
         X = -2
         Y = 5
```

Now for the next segment:
2nd offset: 48,55
3rd offset: 44,56
So now:

Code:

```
      AddSegment = Vector
         X = -4
         Y = 1
```

And the one after that:
3rd offset: 44,56
4th offset: 37,57
So:

Code:
```
      AddSegment = Vector
         X = -7
         Y = 1
```

And finally:
4th offset: 37,57
5th offset: 28,58

Code:
```
      AddSegment = Vector
         X = -9
         Y = 1
```

And you're done, that was as hard you thought it was right? Play around with these variables and make your own awesome walkpaths!

If you find any spelling/math errors/know what other vars do, please tell me.
Sorry if my bad grammar is confusing and hope I was able to help.

(Note: The walkpath in this tut is very basic and probably looks bad, don't expect to much)

Chapter 1

lua modding

DATATYPES

Number

A Number in Lua is any variable of type Interger, Float, or Fixed.

Vector

A Vector in Cortex Command is an array with values x, y specifying a position.

String

A String is any number of typed charicters, enclosed in " "

Boolean

Boolean has two values, True, or False.

Table

Lua has a general-purpose aggregate datatype called a table. Aggregate data types are used for storing collections (such as lists, sets, arrays, and associative arrays) containing other objects (including numbers, strings, or even other aggregates). Lua is a unique language in that tables are used for representing most all other aggregate types. <ref>http://lua-users.org/wiki/LuaTypesTutorial</ref>

Function

In Lua, functions are assigned to variables, just like numbers and strings. Functions are created using the function keyword. Here we create a simple function which will print a friendly message.

nil

nil is a special value which indicates no value. If a variable has the value nil then it has no value assigned to it and therefore will no longer exist (or doesn't exist yet). By setting a variable to nil you can delete a variable.

Userdata

Userdata values are objects foreign to Lua, such as objects implemented in C. These typically come about when an object in a C library is exposed to Lua. An example of a userdata in Cortex Command value is an Activity.

BASIC SYNTAX

Variables

A variable stores a value. The value can be any of a few datatypes. Here are examples to assign variables *x*, *y*, and *z* number, boolean, and string values respectively:

```lua
x = 21 --Integer
x = true --Boolean
x = "Cortex" --String
```

Comments

In programming, comments are parts of code that are completely ignored by the reading program. In Lua, this is accomplished with the use of two hyphens right next to each other: "--"

```lua
--This is a proper comment.
function() --Comments can also be placed after a statement.

--[[Or this format can be used
to make a block comment.
That spans multiple lines.
]]--
```

Variable Manipulation

Variables of any datatype can be manipulated in a variety of ways. We will explain the methods of computing, comparing, and assigning variable values.

Math

The math operators are common math signs:

Math Operators	
Symbol	**Purpose**
+	Addition
-	Subtraction
*	Multiplication
/	Division

Assignment & Math

These operators are shorthand; they first add the inputs, and then assign the result to the first.

Math Operators	
Symbol	**Purpose**

+=	Addition
-=	Subtraction
***=**	Multiplication
/=	Division

Example:

```
x = 1 --x is 1
x = x + 1 --x is 2
x += 1 --shorthand form; x is 3
```

Equality

These operators are usually used with control structures and return a boolean value.

Comparison Operators	
Symbol	**Purpose**
==	is equal to
~=	is not equal to
>=	is greater or equal

<=	is less or equal
>	is less than
<	is less than

Example:

```
x = 5 --assignment
x == 5 --true
x ~= 5 --false
```

Control Structures

If/then

One of the simplest logical constructions, the first statement is evaluated as a boolean. Equality operators are useful to this effect. If the first statement is true the statement after *then* is executed.

Example:

```
x = 1
if x == 1 then y = 1 end
```

or

```
x = true
if x then y = true end
```

For

While

Functions

A Lua function is a set of instructions invoked as a group by name with the syntax:

```
function_name(arguments)
```

In Lua, the trailing parenthesis are not necessary for control structures such as *if* or *for*, but may be useful to maintain logical flow while reading code. If in doubt, parenthesize. Another important facet of functions is their *return value*. A function can be used in the stead of a datatype as long as its return value is of that datatype.

Member Functions

You may have wondered at the term "member function," but this is simply a function that is organized into a group of functions and variables which in Cortex Command are the Manager objects. Proper syntax for accessing member functions in Lua:

```
object_name:function_name(arguments)
```