the zoidberg engine

Search this site

Welcome to the zoidberg engine
Infosphere
**Project Definition**
Resources
Game Ideas
Documentation
The Team
To-Dos
Project updates
Files
Release Dates
Sitemap

**136**
days since
**Project Due Date**

**Recent site activity**

removed stuff
edited by Joe Balough

The Team
edited by rmb5299

removed stuff
edited by Joe Balough

cliCreator
edited by Joe Balough

Testing Framework
edited by Joe Balough

View All

# Project Definition

## Don't know what to put in here? Look at the Project Definition To-Do! It'll give you something to do.

**Contents**

## Abstract

The zoidberg engine is an open source game development kit developed specifically for the Nintendo DS game system. The goal is to provide a relatively easy way for people interested in game development to create games of their own. The zoidberg engine provides such people an outlet to develop their ideas on the Nintendo DS game platform regardless of the genre of the intended game.

The design of the zoidberg engine is such that a person can play any number of zoidberg engine games using the same zoidberg engine executable on their Nintendo DS. All of the game data is stored in a custom zbe datafile that gives the game engine needed graphical data and tells the game engine how to build the levels of the game.

To show the features that have been added to the zoidberg engine an original 2D game has been designed. The game is a playable demo of what the zoidberg engine is capable of doing for Nintendo DS game designers who are interested in using the zoidberg engine for there project.

The zoidberg game engine is designed to be genre independent so that many game types can be designed using the engine. The engine has the ability to support hop and bop 2D side-scroller games similar to the popular Super Mario Brothers and Sonic the Hedgehog but also have the ability to work run and gun side-scrollers like Metal Slug. If the designer would like to make an RPG such as The Legend of Zelda, all the tools necessary to support such a game are included in the engine as well.

A Java-based editor is included with the zoidberg engine that allows the user to design all aspects of their game from the graphics to the level layout to the sound effects. The editor is designed to be easy to use and will provide a series of Creative Commons licensed components so that the user can reuse parts of other people's games without fear of copyright issues. The Creative Commons components are to be implemented so that the user does not need to design all aspects of their game from scratch. The editor creates a final XML file that can be parsed by the cliCreator to build the final zbe datafile that can be loaded and run by a zoidbergengine.nds executable running on the DS.

The zoidberg engine is built upon the DevkitARM toolchain using the MaxMod, libnds, libwifi, and libfat libraries. These libraries allow some abstraction from the direct Nintendo DS hardware to make programming for the platform much easier. The zoidberg engine is written in C++ in order to utilize the powerful object-orientated abstraction and speed it provides. The project is Open Source under the GNU General Public License v.3 and the zoidberg engine files and its dependencies are hosted on github.com and are publicly available for anybody to view and/or modify.

## Features

- Written in C++
- Open source using GPL v.3 license
- Using git and github.com for source control
- 2D side-scrolling platform game engine

- Compiles to .NDS file, uses custom zbe file to build game content
- Genre independent -- can be used for:
  - run and gun
  - hop and bop
  - puzzle
  - comical action
  - cinematic
  - etc.
- Enemy and non-playable character customization
- Decapodian Physics Engine
- Use of custom sprite animations
- Java based game editor

## Version Road map

- √ 0.0.1 - Hello World
  - Does nothing more than say hello to the world.
- √ 0.1.0 - Basic implementation of Object class
  - Shows simple block to represent object
- √ 0.2.0 - Basic implementation of Hero class
  - Allows for a simple object to be added that is under the player's control, further development of all other classes
- √ 0.3.0 - Basic implementation of level load routines
  - Allows the addition of objects and heroes based on the zbe datafile, further development of all other classes
- √ 0.4.0 - Completion of level load routines, beginning work on physics engine
  - Levels will now be constructed entirely from external data sources using libfat
- √ 0.5.0 - Implementation of scrolling backgrounds
  - Allows for large levels to be shown how large by having backgrounds that scroll as the hero moves
- 0.6.0 - Completed physics engine
  - Allows for the hero to stack boxes
- 0.7.0 - Sound support
  - Allow the inclusion of sound effects and background music using the maxmod library
  - **Start Public Releases** - Hop and bop engine elements complete
    - Allow for creation of a hop and bop style game
- 0.8.0 - Full run and gun support, begin work on cinematic engine, assessment of AI scripting feasibility
  - Allow for creation of a run and gun style game
- 0.9.0 - **Feature Freeze - 1.0 Release Candidate Series** - Cinematic engine complete, [AI scripting done]
  - This series of releases will act as 1.0 release candidates. No new features should be implemented and the focus should be on making the engine as stable as possible. The Cinematic engine should be complete, though not necessarily bug-free and the AI scripting engine should be in a similar state if it is feasible.
  - During this period, we will begin work on a Java based editor using the swing gui api. This will ensure that users of all platforms will be able to easily design their own games using the zoidberg engine.
- 1.0.0 - **First Major Release** - all bugs squashed, editor done
  - Major release so no crashing, easy to use, decided-upon scripting language, etc. The editor should also be complete and should be able to make creating a zoidberg engine game easy.

## Class Structure

The engine is broken into several fundamental classes that will be expanded using C++'s characteristic inheritance.

### Basic Classes

#### Object Class

The most basic class in the zoidberg engine. An object is anything in game that would be represented using the Nintendo DS' hardware sprite engine using a sprite entry. Objects move by having impulses give the object a horizontal and/or vertical velocity. Acceleration is counteracted by gravity which is implemented on a per-object basis, meaning that all objects can "fall" in different directions. In order for objects to interact in a realistic fashion, all objects have a weight value that is used in collision resolution

Objects are used in-game to represent any object that the sprites can use or interact with using the physics engine but does not provide anything to the sprite. Used in this way, they can be used to act as a platform.

### Assets Class

The assets class manages the current game's zoidberg engine zbe datafile. It is in charge of loading all necessary graphical data from the disk into the DS's memory and keep track of what data can be removed from memory. It tracks this information to allow a game to use as many different sprites as possible without running out of memory. Upon initialization, the assets class parses the datafile building arrays of object and level metadata.

When a level is initialized, it asks the assets class to make sure that all the graphical data needed for that level is loaded into memory. The level class then uses the metadata provided by the assets class to build the vectors of all the objects used in that level for the level class. The level then uses those vectors to update all objects.

### Background Class

The background class handles loading and scrolling backgrounds using the DS's background hardware. Because the DS is limited to strict background sizes, the background class initializes its background to be the smallest size that is larger than the DS screen. When scrolling, it swaps out parts of the background that are not in view but may come into view if the hero moves far enough in that direction.

### Game Class

Contains relevant data about the game currently being played. This includes, but is not limited to, the creator of the game, it's name, description, etc.
Compared to the other main engine classes, the Game class is amongst the simplest. It is mostly in charge of initializing the assets class and running a level.

### Level Class

This class runs what is commonly referred to as the main game loop. It contains a list of pointers to all the objects that appear in its level that it iterates through telling all of those objects to update. Based on which objects have changed during the update, collision detection is run using the CollisionMatrix class to get a list of the fewest objects that could be interacting with it.

As mentioned above, when the level class is initialized, It uses metadata provided by the assets class to properly populate the heroes, villains, items, objects, switches, etc. lists with what it finds indicated in the level data file.

### CollisionMatrix Class

The collisionMatrix class divides the screen into smaller squares that contain game objects. It does this by using a 2-Dimensional array of object pointer lists. Upon object initialization, a pointer to it is added to the matrix by figuring out which square it should go into and adding it to the list at that coordinate. All level objects are always present in the collisionMatrix.

When an object moves, it removes itself from its list in the collisionMatrix and re-adds itself to the collisionMatrix in order to make sure that it is always present in the correct list.

When collision detection needs to be run because an object moved, it queries the collisionMatrix for a list of all possible objects it may have collided with. When getting such a list, the object's x and y coordinates are provided, the collisionMatrix will give a list of objects in the colliding object's group along with all the objects in the groups above, to the left, and to the upper left. These four groups contain all the objects that the object could possibly be colliding with.

### Sprite Classes

#### Hero Class : public Object

The character that the player controls. Expanded to include usable items, weapons, and powerups. The design of the Hero class is such that more than one can be dropped into a level and can interact with other Heroes. This will be advantageous for future improvements to engine that would allow multiplayer gaming in either a cooperative or competitive way. For right now, however, this would result in undefined behavior because the hero object updates the screen's viewport coordinates.

### The Cinematic Engine Classes

The cinematic engine breaks cinematic scripts down into three parts: Script -> step -> action. A script describes the cinematic event which is composed of several steps. Each step can have multiple actions defined to occur simultaneously.

It should be noted that though the cinematic engine is designed and mostly programmed, it is not fully implemented and is completely untested. These class descriptions remain to explain how the implemented code will function.

#### Script Class

The Script class contains a list of Steps that should be performed to complete the script parsed by the Cine class. There are two ways to run a script: interrupting gameplay and not interrupting gameplay.

When a script interrupts gameplay, the script class starts its own "main game loop" and doesn't return execution to the level class until the script is complete, essentially pausing the game while the script runs. In this style of script, the script class iterates through the list of steps and runs them, then returns.

When a script does not interrupt gameplay, it keeps track of which step is currently running and runs it when the level class tells the script to update. The results in game events continuing to occur while the cinematic script runs.

#### Step Class

The step class simply defines a list of actions to run. When told to update, it will go through the list of actions and tell each of them to update providing a value representing how complete they should be.

#### Action Class

Used to cause one thing to happen in a cine script. This will contain a pointer to an object to manipulate and data about what it will do in what time. The action class itself is essentially an abstract that does not define any one action to occur. Specific cinematic events are coded by inheriting the action class and implementing the action in that class' update function.

## Cinematic Engine

Using a pre-rendered video on the DS would be nothing but a headache. The hardware is not that powerful and video codecs are a very complicated beast with which to tangle. Instead, cut scenes will be implemented in the zoidberg engine by using a simple scripting language that provides camera commands and hero input actions. The scene will be constructed by building the level indicated in the script like it would for in-game play. All hero-object interactions will be preserved. In every way, the regular in-game zoidberg engine will be running while a cut scene is playing.

However, instead of having the user's input control the hero, commands listed in the cut scene script will control the hero. Since the Hero class is designed to allow for multiple heroes in one level, if a scene wants to have a hero interact with a villain, that villain will be made into a hero for the cut scene only and its actions provided by the script. That way, the design of the Villain class would not need to be impeded or changed when in cut scenes.

Camera commands in the script will allow for the camera to zoom in, zoom out, lock (don't follow hero), unlock, move, and eventually perform effects like fade to white, fade to black, various wipes, et cetera. There will also be the ability to insert messages that wait for user input before proceeding. These would pop up at the bottom of the screen and, generally, look like speech bubbles containing text coming from a character.

Finally, the script will have full abilities to add, remove, move, rotate, etc. any object in game.

These cut scenes can either be triggered in game by the user running into an invisible switch or upon completion of a level. It should also be possible to use this cinematic engine in-game leaving the player in control of their character while things happen around them (like items flying around them in a circle).

## Game Editor -- zbeCreate

The main goal of the zoidberg is a lightweight and powerful game engine that other people can use to make games of their own. Ideally, this would not be limited to people who can program the necessary customizations into their game or people who are comfortable using command-line tools. For the first major release, we will include a Java based game editor that anybody can use to quickly and easily create a zoidberg engine game.

### Level Data Tile Designer

The Nintendo DS has a powerful 2D graphics engine that allows for fast drawing of a tile based world. This means that all background level data is broken down into a list of small tiles that are then arranged on the screen to make the level. For example, there would be a tile representing one type of wall's vertical section. Whenever that wall has a vertical section the vertical section's tile would be used for every small increment of the vertical face. When rendered on the screen, these tiles can be flipped horizontally, vertically, and colored based on a palette.

The tile designer will contain a means for the user to edit and preview the tiles to make sure that they not only look good but are seamless when placed next to a different tile. For example, the user would want to be able to easily see that a vertical section of a wall looks seamless placed below the top piece of that wall. There would also be a way for the user to edit and preview different palettes for these tiles.

The user would not be forced to create their own unique tiles, however. There will be a Creative Commons Collection of tiles that the user can drag and drop into their own level tile design.

### Level Tile Layout

Using the tiles selected in the Tile Designer, the user will now be able to simple "draw" what their level will look like. Along the right side of the screen, there will be a "Tilebox" that contains all the tiles defined using the Tile Designer. The user simply needs to select a tile from the Tilebox and click in any boxes in the designer that they want to use that tile.

The tiles don't have any behavior associated with them by default, however, so after drawing the level, they will have to associate behavior with these tiles. For example, the user can click on the "Wall Tool" and highlight all the tiles that should be made into walls. Platforms will be indicated using a similar tool.

### Sprite Designer

The sprite designer will be similar to the Tile Designer but the user will not be limited to the small size of the tiles since sprites can be any size. The Sprite Designer will let the user draw a sprite and assign AI to it if it is a Villain. Like the other Designers, the Sprite Designer will contain a Creative Commons Collection allowing the user to quickly add a sprite without having to create one from scratch.

### Sprite Layout

Sprites and Objects are not a part of the Level's tile layout since they will be implemented using the hardware Sprite engine. To insert these objects into the level data, the user will switch to a "Populate" mode in order to insert Sprites and Objects. Similar to the "Draw" mode, there will be boxes on the right of the screen, only these will be "Villainbox", "Itembox", "Switchbox", et cetera. The user can select a Villain from the villainbox and indicate where in the map that villain should be spawned at the start of the level.

When the user adds a Switch from the Switchbox, an editor will automatically come up allowing the maker to tell the engine what that switch should do. It could trigger a Cinematic Event or open a door. In any case, the action will be simple and the user needs only indicate what door it should open or what script to run.

Subpages (2): Project Definition To-Do removed stuff

## Attachments (1)

Project Definition turned in on 1.25.10.pdf - on Jan 26, 2010 12:40 PM by Joe Balough (version 1)
89k View Download

## Comments (1)

**Joe Balough** - Jan 25, 2010 2:43 PM
wow that editor was a lot of good fluff. haha