# Chapter 1: Isotope Overview

"Don't walk away in silence..."

## 1.0 Quick Facts

### What is Isotope?

Isotope is an isometric game engine for Pygame written using the Python programming language. It provides the framework for constructing an isometric graphics game with actors who can pick up objects and jump onto platforms. Isotope also provides automated actors who can interact with the player or their environment. The screenshot on the right shows Isotope in action. Isotope is similar to the old game engines for the ZX Spectrum.

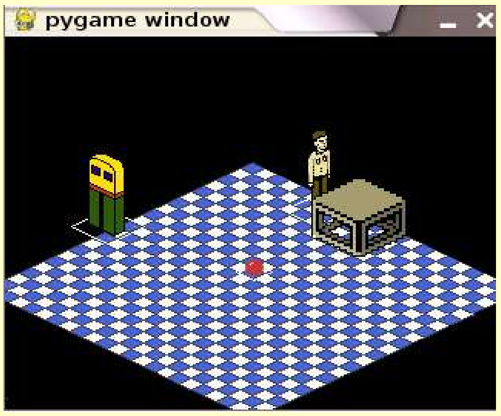


*Figure 1: Isotope screenshot*

### What can I use Isotope for?

Isotope is for creating isometric games primarily but can be used for other isometric applications.

### Features

- Actors: Used for player and monster game objects. Capable of facing, collision response,jumping, automation and inventory.
- Multiframe animation using images
- Simple physics simulation of velocity, gravity, collision and touch detection.
- All objects can be customised and extended using Python.
- Free commented GPL open source code.

### Why should I use Isotope?

You should use Isotope if you need a scripted version of an Isometric engine which is open source and simple. It is useful for understanding how Isometric engines work and for writing python based applications which need a core engine completely written in Python. Isotope consists of simple light weight code to ensure that it can be easily understood and modified. Its reasonably fast and provides a simlar level of speed to the games on the ZX Spectrum (but with no attribute clash).

### What do I need to use it?

Pygame version 1.6 and Python 2.4. Pygame and Python can be downloaded from the Internet.

### What documentation is there and is the code commented?

The code is commented and also Pythons docstring inline help is used - type `help(isotope)` in Python for the top level of the system. This Isotope guide provides an overview, a tutorial, a programmers interface description and a system architecture description of the code itself.

### What rights do I have?

The software is licensed under the GPL so you can use the software for no charge and you can add your own extensions. If you modify the original code you must contribute the changes back to the Isotope project.

### What support can I get?

No commercial support. An email reply if your query is sensible and the author is available. The intention is to have community support if people become interested in the project.

email: simon.gillespie@btinternet.com

### Who created Isotope,why... and how?

Simon Gillespie created Isotope because he could find no other open source Isometric projects available for Python that had clear documentation and simple commented code. There were many great efforts however but they were often focused more on technical achievement.

Simon has dedicated this guide to his family and friends who have helped him so much and to Ian Curtis.

## 1.1 What now ?  Contents of the guide:

## 1.2 High level description of Isotope

Isotope is a Isometric engine written only in Python with the aim of being as simple and concise as possible but capable of the key tasks needed for an Isometric game. It provides the Isometric graphic functions needed to present an image of the physical objects on the screen. It also simulates the physical nature of objects: velocity, collisions and what to do if they are touching.

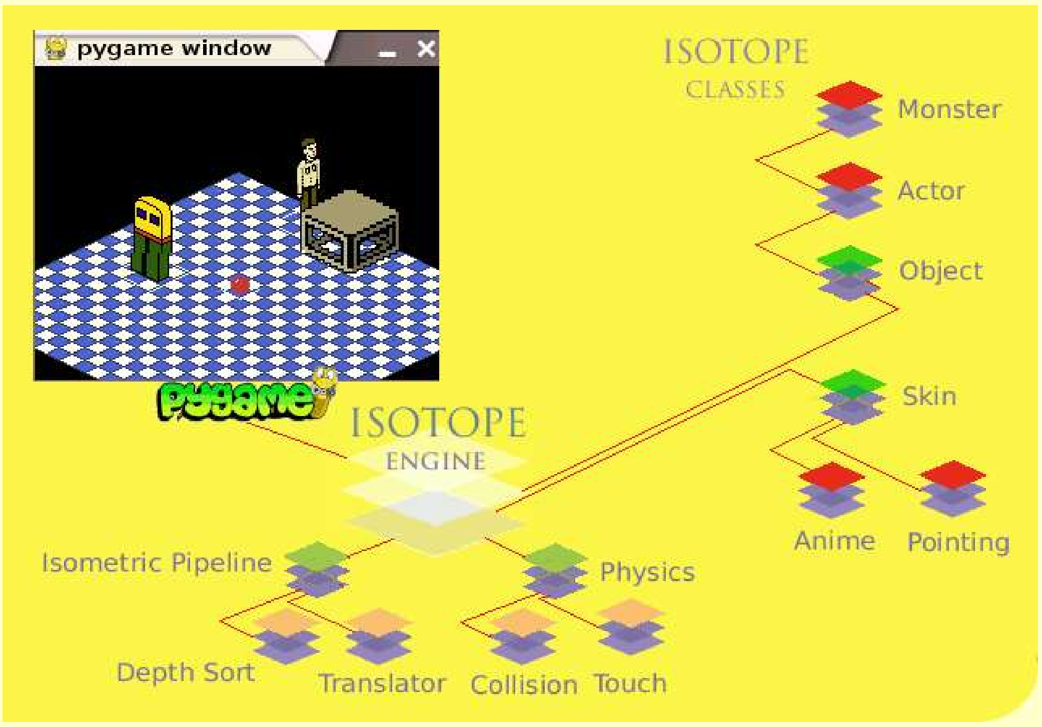


*Figure 2: The Isotope classes and the Isotope engine*

Isotope has two branches of classes which the programmer uses to talk to Isotope: objects and skins. The objects provide the physical information and the skins provide the animation images to plot on the screen. Figure 2 shows a diagram of how the Isotope classes and the Isotope engine fit together.

The objects class has been subclassed (specialised) to give an actor classes. The actor classes can perform the complex functions associated with a person: Movement, Facing, Jumping, picking up objects (inventory). For example, actor class has been further subclassed into a monster class which provides an automated actor. It is programmed to turn away from a collision.

The skin class has two standard subclasses: Pointing and Anime. Pointing provides a skin for the actor class which needs images for its facing directions and also its animation when walking. The Anime class simply cycles through its images like a movie clip.

The Isotope engine itself is based on two subsystems: the physics simulator and the isometric pipeline. The object classes are used by the physics simulator while the Isometric pipeline translates the objects state using the skin classes into sprite information. This sprite information is then passed to pygame to draw on the screen.

The isometric pipeline consists of a translator module and a depth sort algorithm. This produces the coordinates for the screen and sorts the sprites into a drawing order which provides the illusion of 3 dimensional depth. The physics simulator has two components, a touch and a collision processor. The collision processor can deal with multiple objects touching and movement of objects to ensure they dont exist in the same space. The touch processor allows an object to know if its touching on any of its faces and if its not touching anything at all.

ISOTOPE
THE ISOMETRIC ENGINE FOR PYTHON

# *Chapter 2*
# *Installation & Tutorial*

*"But if you could just see the beauty..."*

## 2.0 How to install and run Isotope

First make sure you have pygame installed. Run python and try to import the pygame module:

```
    > python

    Python 2.4.1 (#2, Mar 30 2005, 21:51:10)
    [GCC 3.3.5 (Debian 1:3.3.5-8ubuntu2)] on linux2
    Type "help", "copyright", "credits" or "license" for more information.
    >>> import pygame
```

If you get an error then you need to download pygame from the pygame website and install it.

Exit from python and download the tar file from the website into the isotope directory using your web browser and untar the file:

```
    > tar -xvf isotope.tgz
```

Copy the isotope package into your python directory, your python libraries may be in another directory so you need to check this.

```
    > sudo cp -R isotope/isotope /usr/lib/python2.4/site-packages
```

Test your installation by running the Joy Divison world tutorial:

```
    > cd isotope/tutorial
    > python tutorial.py
```

You should see the tutorial world for the next section: The Joy Divsion World. Press keys o,p,q,a and m to move about and jump.g picks up and object and d drops it.

## 2.1 Tutorial: A Joy Division World

Now that you have the demonstation running, its a good time to look at how a world in created in Isotope. The world consists of the scene, object and skin definitions. If you open the joy_demo.py file you will see the definitions for the objects after the pygame initialisation code.  Two main structures are defined: an scene group then a skin group. These are lists of pointers to the scenes with their objects and skins.

For the demonstration we create Ian Curtis for the player to control. He can move between two rooms, the bedroom and the lounge. In the bedroom we define a guitar for him to pick up, and a bed to jump onto. In the Lounge we provide a sofa and an amplifier. In both rooms we make some ground and walls so he does not fall into empty space.

Heres the code to set up the scene group : note the scene types 0 and 1 are used to match their background skins.

```
joy_world=[]

bedroom=isotope.scene.scene(0)
lounge=isotope.scene.scene(1)
joy_world.append(bedroom)
joy_world.append(lounge)
```

it simply defines a list for the scenes: joy_world. Two scenes to put the objects into: bedroom and lounge.

Now lets focus on the first object definition in the bedroom, the guitar:

```
guitar=isotope.objects.object_portable([60,0,40],[20,12,20],3)
bedroom.append_object(guitar)
```

The first line creates a object named guitar using the subclass of object_3d called an object_portable. Its important because an object_portable class can do more things than the object_3d and its used to represent objects that can be carried. The guitars starting position is defined as the first vector in the initialisation parameters:pos. pos is a vector based on pixel measurements from the origin point at 0,0,0. The next vector is size which defines the size of the object, the space it occupies using its position as an offset. The last value defines the object type, this is a specific code which identifies how the object appears. It is the same number as the skin number in the skin group. The last line places the guitar into the bedroom scene at the end of the list

An important object which allows the lead actor Ian Curtis to jump between scenes is the scene.portal object.

```
door=isotope.scene.portal([180,80,0],[10,30,56],5,lounge,[10,90,0])
bedroom.append_object(door)
```

The fourth argument 'lounge', links the door to the next scene from the bedroom.  The fifth argument is the coordinate to place the lead actor. You will find another door definition in the lounge for a door which links to the bedroom.

Next we can look at the skin definitions. First we define a skin group using a list:

```
skin_group=[]
```

Take a look at skin definition number 3, as the skins are numbered starting from 0, its for the guitar object above:

```
guitar_image=isotope.skins.load_images(["guitar.png"])
skin_group.append(isotope.skins.skin(guitar_image,"Amp"))
```

First we load the image for the guitar then append it to the skin group. Note that we can load multiple images at once if the skin has animations or different faces. This is very useful for actors that turn around. You can see this in the skin definition for Ian Curtis. We use a pygame trick to mirror the images to create the different facing directions.

```
# Mirror the images to complete the animation
for i in range(3,9):
   ian_curtis_images[i]=pygame.transform.flip(ian_curtis_images[i],1,0)
```

Finally we create and launch the engine using our scene and skin groups:

```
# Create an isotope engine using the skin_group and the scene_group
joy_engine=isotope.isotope.engine(ian_curtis,skin_group,surface)

# Start the isotope engine
joy_engine.start()
```

Thats just about it! If you go back to the start of the joy_demo.py code. we imported a number of modules which will be discussed in the next chapter in detail.

```
import isotope.isotope
import isotope.scene
import isotope.objects
import isotope.actors
import isotope.skins
```

These five modules form the Programmer Interface for Isotope. Continue on to chapter 3 if you want to know more about this.

ISOTOPE
THE ISOMETRIC ENGINE FOR PYTHON

# *Chapter 3*
# *Programmer Interface*

*"When routine bites hard..."*

## 3.0 The Isotope Programmer Interface

The Isotope programmer interface is the set of modules and classes used to access and communicate with the Isotope engine. The modules and classes are thoroughly documented in the python help docstring interface. Type `help(isotope.[`***`module name]`***`)` in the python command line after importing the module for the descriptions. The module help is also available from the shell using pydoc: `pydoc isotope.[module name]`

The following page provides a helpful summary sheet of the classes and some details associated with their use.

## 3.1 Interface Files

isotope.py
The engine is a combines, simulation, isometric view elements with keyboard response and an information display.

scenes.py
A scene or location in a game with objects inside the scene is represented by the scene class. Portals are also defined in the scenes and can represent doors or teleport systems.

objects.py
Items with simple behaviour in the game can be represented using definitions in the objects file. They can have gravity and be carried by actors.

actors.py
Actors which can move about and jump are defined in the actors file. A lead actor and a monster type are also defined. Lead actors can change their scene and carry objects. The monster class is an example of an automated actor with a program to turn around when it hits something. The facing direction is important for actors and is defined using a unit vector. An actor will jump and put down objects in the direction they are facing.

special_objects.py
This file contains example definitions of advanced objects which perform programmed actions similar to the monster class.

skins.py
The skins file defines the way an object is represented graphcally in the isometric view. The skin objects match the correct image to the state of the object. Designing graphics for isometric games requires special care as optimisations are used to ensure that object movement is smooth and quick. To calculate the required object size for a graphic you should use a graphics program to measure the x, y and z coordinates of the sprite. The z coordinate is the same but the x and y coordinates must be divided by 1.18.

Color key transparency is also important. Total white is used as the color key. This means that if a pixel with a value of RGB:255,255,255 is found they it is transparent and will show whatever is behind it.

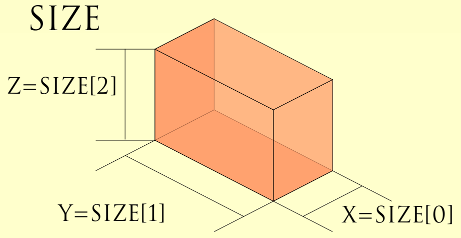
isotope_elements.py
The view and simulation elements are defined in this file. It can be used by advanced programmers to provide more control than the isotope engine allows.

# ISOTOPE
THE ISOMETRIC ENGINE FOR PYTHON

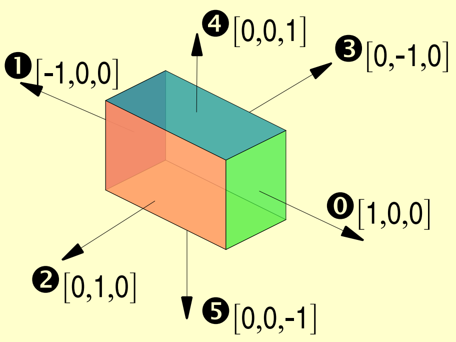
*Programmer Reference Sheet*

## OBJECTS

### SIZE



### FACE NUMBER AND FACING



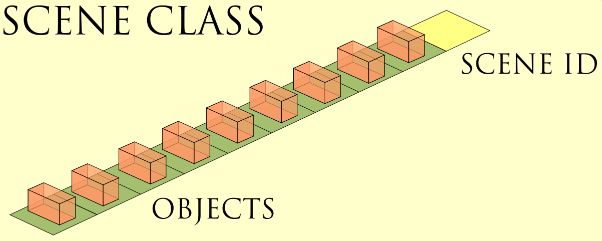
## OBJECT CLASSES

### SYMBOLS:

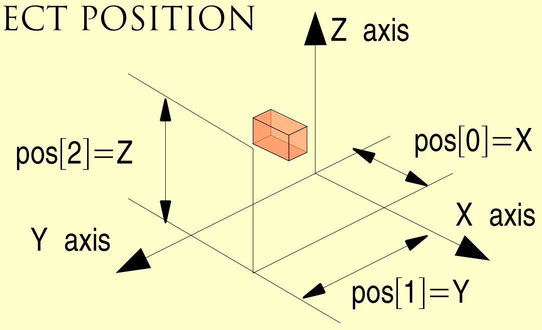
| | | | |
|---|---|---|---|
| ✚ | Position | ❄ | Disolves after time |
| ➔ | Velocity | ✡ | Explodes on touching |
| ✍ | Portable | ☝ | Invertory |
| ✈ | Change scene | ∩ | Teleport, |
| ○ | Gravity | ➚ | Jumping |
| ➲ | Reversal | ◈ | Create Object |
| ✦ | Facing | | |

| | | | |
|---|---|---|---|
| OBJECTS.PY | OBJECT_3D ✚➔ | OBJECT_GRAVITY ✚➔○ | |
| | OBJECT_PORTABLE ✚➔○✍ | | |
| ACTORS.PY | ACTOR ✚➔○➚✦ | MONSTER ✚➔○➚➲✦ | LEAD_ACTOR ✚➔○➚☝✈✦ |
| SPECIAL OBJECTS.PY | FACTORY ◈ | DISOLVER ✚➔○❄ | EXPLODER ✚➔○✡ |
| SCENES.PY | PORTAL ✚➔∩ | | |

## SCENE CLASS

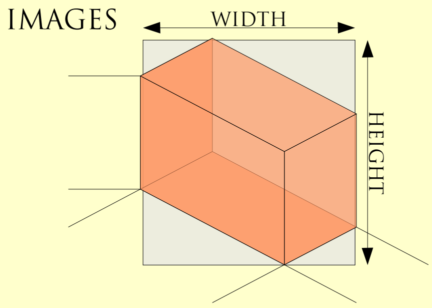


## OBJECT POSITION



## SKINS

### IMAGES



PIXEL DIMENSIONS ARE 1.18 TIMES THE OBJECT SIZE
FOR X,Y AND EQUAL FOR Z COORDINATE

## SKIN CLASSES

### SYMBOLS

| | |
|---|---|
| ✚ | Basic Image: Images[0] |
| ➔ | Animation: Cycle through all images |
| ✦ | Facing: Image number is the same As the face number above |

| | | | |
|---|---|---|---|
| SKINS.PY | SKIN ✚ | ANIME ✚➔ | POINTING ✚➔✦ |

ISOTOPE
THE ISOMETRIC ENGINE FOR PYTHON

# Chapter 4
# System Architecture
# How Isotope works

*"More than this... There is nothing"*

## 4.0 Technical Specification

Required Software: Pygame module v1.6, Python v2.4.

Simulation: touch (proximity) and collision detection based on boundary box (aligned to the axes, not direction flexible), no world boundary or false ground plane. Uses objects to define a ground plane. Object tick, collision and touch events simulate velocity and gravity and can be extended by the programmer using the Python class system Collision detection does not register objects that pass completely through an object in a single frame therefore object speed and size must be controlled.

Animation: multiple images translated to final display using skins. Game time clock provides frame sychronisation. Pygame is used to plot sprites to the display.

Drawing: Isometric representation in a pygame window, fully ordered sprites method. No tilling system. Allows objects to move in all x,y,z.

## 4.1 Isotope Engine Description

The core elements of the Isotope engine are defined in the isotope_elements.py file. These elements integrate the functions into classes which can be driven by the engine. The main function files are described below.

### Isometric Pipeline: isometric.py module

The Isometric pipeline uses two pipelines to process the object information: The Isometric pipeline to draw a view of the objects and a Simulation pipeline which models the movements and actions of the objects.

Isometric pipeline (Drawing the veiw):
skins.Update_skin_images(skin_group, object_group)
isometric.iso_order(object_group)
isometric.iso_group_transform(object_group,sprite_group,view_size,inversion_height)
sprites.ordered_draw(sprite_group,draw_order,screen) : Note: This function is in sprites.py

The skin class has a physical body (called an object) and a representation of the body to the computer in the Isometric view (called a skin). It relates the ideal of the simulation to the practical visual representation on the computer screen. The skin class has a method: get_image(), to translate the correct image to display based on the state of the object/body. The view position on the screen is calculated from the body/object positions (object group) and size values and the view parameters. The order of drawing the images on the screen is also calculated from the objects position and size values. The final result is an ordered sprite group which can be plotted into the window.

### Simulation pipeline: physics.py module

Simulation pipeline:
movement(), object.tick() and player keypress
objects.touch_processor(object_group)
objects.collision_processor(object_group)

The physical world is represented by the object_group. It undergoes a movement phase which are AI (objects tick routine) and key presses for player objects. Then the object group is passed through a collision detection processor to ensure that objects are not overlapping in space and finally a touch detection processor which allows the objects to react to nearby objects.