**By Dino Gambone**

**Object Selection**  
**Revision 2.0**  
**by** [**Dino M. Gambone**](mailto:dmgambone@yahoo.com)

**Introduction**

The purpose of this article is to explain an efficient method of selecting an object based on a pixel point in an isometric game. The previous version of this article provided a method of doing just that but it was inefficient and very slow. Thanks goes out to Michael Duffy for tipping me off to this new method.

**The basic idea**

Most isometric games, if not all, draw the screen starting from the top-left corner to the bottom-right corner. The drawing routine moves from left to right and top to bottom. This ensures that all objects are drawn with the proper perspective. The more distant images are covered over by images of objects closer to the screen. Using this basic idea, then a record of where all objects are can be stored. Each record consists of a bounding rectangle and an object ID.

When trying to determine what object, if any, is under a given pixel point the list of records is traversed in reverse order. If the point is in the bounding rectangle then an actual pixel text is done. In the following article, I will walk through the steps to used to determine what object is located under a given pixel point.

**The display record: tagDISPINFO**

In order for this method to be successful, we need to keep track of what objects were drawn where. The method I am going to describe to you uses a struct called tagDISPINFO. This struct will store all the information about items being drawn to the screen. Here's the basic struct:

typedef unsigned int GAMEOBJID;

struct RECT{

long left;

long top;

long right;

long bottom;

};

struct tagDISPINFO{

GAMEOBJID ObjID; //Unique game object ID

RECT BoundRect; //Bounding rectangle

};

In this struct are two key members. Your struct may have additional members depending on how you will implement and handle the list of tagDISPINFO. The firstof these key members is GAMEOBJID ObjID. GAMEOBJID ObjID is a unique ID for the game object that is currently being drawn. RECT BoundRect is the bounding rectangle of the image. If you are using DirectDraw chances are that you have this already calculated. Simply copy the rectangle you use when drawing the image into BoundRect.

There may be more members to this struct depending on exactly how you implement the storing of this information. Linked lists, for example, will require a pointer to the next node. Here's the struct now:

struct tagDISPINFO{

GAMEOBJID ObjID; //Unique game object ID

RECT BoundRect; //Bounding rectangle

tagDISPINFO \*NextNode; //Next item in the list

tagDISPINFO \*PrevNode; //Previous item in the list

};

**Producing the list**

For this method to work properly, we need to update the list every time the screen is drawn. This means that we need to add to the drawing routine that draws BitBlts the actual image to the screen some extra code. This code simply adds to the end of the list of records information about the image just drawn. This information is the object ID that the image represents and the bounding rectangle. Populating the list is actually the easiest part of this method. It may become a bit harder if you are using a linked list rather than an array, but it is generally quite easy to do.

*Tip*: Don't add items to the list that can't be selected such as tiles. It is, in my opinion, pointless.

**Selecting the object**

Ok... now we have a list of objects and we now want to know which object is under a given pixel point. We first start out at the very last item in our selection list. This object was the last object to be drawn and thus the topmost object on the display. We first check to see if the pixel point within the bounding rectangle for this object. If isn't, then we move to the previous item in our selection list and check that. We continue until we reach the beginning of the list or we find an object that contains our pixel point.

Now that we have an object that is under our pixel point, we need to figure out if our pixel actually is over a non-transparent portion of our image. We first need to convert our display pixel point to a point that is relative to our image. A simple calculation will do this:

//Pt is our pixel point

//sel\_rec is the selection record that contains the bounding rect

Pt.x -= sel\_rec->BoundRect.left;

Pt.y -= sel\_rec->BoundRect.top;

With our point now made relative to the image, we can determine whether or not the point is over a non-transparent portion of our image. This can be done in many different ways. The easiest way is to use GetPoint() to get the color and match it to the transparent color. Other methods are more complex but are faster. GetPoint() is an extremely slow routine and shouldn't be used often.

**Conclusion**

I have tried to be specific about how to perform object selection without being too specific to any game engine. This article will hopefully give you, the reader, a better idea on how to perform object selection in your isometric game.