Using Milkshape and A* algorithm

CS134

Chris Pollett

Nov. 22, 2004.

Outline

- Will look at some people's Seige Programs
- Talk about getting MD2 models out of Milkshape and into Pop.
- Talk about A* algorithm

Using Milkshape

- Milkshape needs a little help in order to export MD2 models.
  - You need to copy the MD2.qc file from Milkshape's directory over to the directory with your model
  - You need to assign all vertices in your model to some joint before you export. This is true even though the joints don't get exported in the MD2 format.

Getting Your Model into Pop

- Biggest headache:By default, Pop wants to animate your model between different frame numbers. If you don’t have enough frames (look in the source comments for the ranges) nothing appears.
- Solution: Either make your model with enough frames for Pop (not default in Milkshape) or override imagedraw and call the Animate function yourself.

Example imagedraw

```
void cSpriteQuakeFrame::imagedraw(cGraphics
   *pgraphics, int drawflags)
{
   if (!pgraphics->is3D()  || !_pModel)
   {
       cSprite::imagedraw(pgraphics, drawflags);
       //Use the baseclass draw (a circle)
       return;
   }
   _pModel->Animate(pgraphics,
   _curFrame,_curFrame, 1); /*_curFrame is field
   of our class -- can set to what want*/
}
```

More on Getting Your Model into Pop

- Skin filenames in the MD2 file itself are ignored by Pop.
- When you use the 3 argument constructor, you must supply a skin name: Even if you model doesn't use skins.

A* Algorithm

Problem: Have a monster and want it to track a player, but there are obstacles in the way. Have a terrain map and what to send robot troops to some against known player positions along cheapest route.

Solution: Want to use search to find the best path around the obstacles to the player.

Tweaks: If monster is not omniscient you might get it to search to last known position of player. Then do a breadth first from there.

More A*

Idea: Obstacles induce a graph onto scene. Want to find a path to the player along this graph.

- To search the graph keep an active priority queue of nodes to search from.
- Initially, this list just has the Monster's position.

* Pull the top item off our queue. Check if is at player. If yes, output path. Otherwise, check a list of already seen nodes to see if have already visited this node.

- If have, pull another item off queue. Go back to (*) above.
- If not, look at all vertices one edge away from this edge.
- For each vertex compute the cost of getting to that vertex + an estimated heuristic cost of going from that vertex to the solution.
- Add each of these vertices to the priority queue using this heuristic. Go to item (*).

Example on the board.