```
//==========================================================================
//  File:     BuildStage.mel
//  Author:   Ryan Durand
//  Project:  Tad Studbody
//  Desc:     This harvests information from the current scene to build a Lua
//            script that will build the scene in game. Also generates a list of
//            meshes and materials for the loading script to use.
//==========================================================================

//==========================================================================
//    getNodes: Gets all transform nodes with the passed string in their name.
//==========================================================================
proc string[] getNodes(string $mask)
{
  string $objs[] = `ls -fl -type transform`;
  string $result[];

  int $i, $count = 0;

  // Find reference names.
  for($i=0;$i<size($objs);$i++)
  {
    int $match = `gmatch $objs[$i] $mask`;

    if ($match > 0)
    {
      // Throw out non-base objects.
      $match = `gmatch $objs[$i] "*|*"`;

      if ($match == 0)
      {
        $result[$count] = $objs[$i];
        $count++;
      }
    }
  }

  return $result;
}

//==========================================================================
//    rotateVec
//==========================================================================
proc vector rotateVec(vector $v, float $rx, float $ry, float $rz)
{
  matrix $rotMatX[3][3], $rotMatY[3][3], $rotMatZ[3][3];
  vector $tv = $v;

  // X-axis rot mat.
  $rotMatX[0][0] = 1; $rotMatX[0][1] = 0;        $rotMatX[0][2] = 0;
  $rotMatX[1][0] = 0; $rotMatX[1][1] = cos($rx); $rotMatX[1][2] = -sin($rx);
  $rotMatX[2][0] = 0; $rotMatX[2][1] = sin($rx); $rotMatX[2][2] = cos($rx);

  // Y-axis rot mat.
  $rotMatY[0][0] = cos($ry);  $rotMatY[0][1] = 0; $rotMatY[0][2] = sin($ry);
  $rotMatY[1][0] = 0;         $rotMatY[1][1] = 1; $rotMatY[1][2] = 0;
  $rotMatY[2][0] = -sin($ry); $rotMatY[2][1] = 0; $rotMatY[2][2] = cos($ry);

  // Z-axis rot mat.
```

```
  $rotMatZ[0][0] = cos($rz);  $rotMatZ[0][1] = sin($rz); $rotMatZ[0][2] = 0;
  $rotMatZ[1][0] = -sin($rz); $rotMatZ[1][1] = cos($rz); $rotMatZ[1][2] = 0;
  $rotMatZ[2][0] = 0;         $rotMatZ[2][1] = 0;        $rotMatZ[2][2] = 1;

  $tv = $tv * $rotMatX; $tv = $tv * $rotMatY; $tv = $tv * $rotMatZ;

  return $tv;
}

//==========================================================================
//    inArray: Check for a string in an array of strings.
//          Return: The index in the array, or one past the end if not found.
//==========================================================================
proc int inArray(string $myArray[], string $key)
{
  int $i;

  for ($i=0;$i<size($myArray);$i++)
    if ($myArray[$i] == $key) return $i;

  return $i;
}

//==========================================================================
//    writeLua: Write the stage's Lua script for our engine.
//==========================================================================
proc writeLua(string $stageName, string $path)
{
  string $objs[], $archetype, $result, $matName;
  string $attrList[], $scriptList, $scripts[];
  string $archetypes[], $materials[];
  float $transX, $transY, $transZ, $scaleX, $scaleY, $scaleZ, $rotX, $rotY, $rotZ;
  vector $look, $up, $center;
  int $match, $i, $strlen;

  // Write script preamble.
  $result = "local Stage = Stage\nlocal Tad = Tad\n\nmodule(..., ";
  $result += "RegisterGeneralScript);\n\nfunction ";
  $result += $stageName;
  $result += "()\n  local stageId = Entity.Create()\n  local curEnt = ";
  $result += "Stage.entityNum\n  local curId = {}\n  local msg\n\n";
  $result += "  Stage.stageID = stageId\n\n  -- Stage objects --";

  // Get renderable object list and write it out.
  $objs = getNodes("x_*");

  for($i=0;$i<size($objs);$i++)
  {
    // Construct archetype name.
    $archetype = substring($objs[$i], 3, size($objs[$i]) - 3);

    // Collect unique meshes from archetypes.
    $archetypes[inArray($archetypes, $archetype)] = $archetype;

    // Collect unique materials from objects if they have the attribute.
    if (size(`listAttr -st "Material" $objs[$i]`) == 1)
    {
      $matName = getAttr($objs[$i] + ".Material");
      $materials[inArray($materials, $matName)] = $matName;
```

```
   }

   $result += "\n  curEnt = curEnt + 1\n  Stage.entityList[curEnt] = ";
   $result += "Archetype.CloneFrom(\"";
   $result += $archetype;
   $result += "\")\n";

   // Get attribute data.
   $transX = eval("getAttr " + $objs[$i] + ".translateX;");
   $transY = eval("getAttr " + $objs[$i] + ".translateY;");
   $transZ = eval("getAttr " + $objs[$i] + ".translateZ;");

   $scaleX = eval("getAttr " + $objs[$i] + ".scaleX;");
   $scaleY = eval("getAttr " + $objs[$i] + ".scaleY;");
   $scaleZ = eval("getAttr " + $objs[$i] + ".scaleZ;");

   $rotX = eval("getAttr " + $objs[$i] + ".rotateX;");
   $rotY = eval("getAttr " + $objs[$i] + ".rotateY;");
   $rotZ = eval("getAttr " + $objs[$i] + ".rotateZ;");

   // Write positional info.
   $result += "  Entity.OffsetPosition(Stage.entityList[curEnt], ";
   $result += "{x=" + $transX + ",y=" + $transY + ",z=" + -$transZ + "})\n";

   // Write scale info.
   $result += "  Entity.SetScale(Stage.entityList[curEnt], ";
   $result += "{x=" + $scaleX + ",y=" + $scaleY + ",z=" + $scaleZ + "})\n";

   // Calculate look and up vectors.
   $rotX  = deg_to_rad($rotX);
   $rotY  = deg_to_rad($rotY);
   $rotZ  = deg_to_rad($rotZ);

   $look = rotateVec(<<0,0,1>>, $rotX, $rotY, $rotZ);
   $up = rotateVec(<<0,1,0>>, $rotX, $rotY, $rotZ);

   // Write orientation information.
   $result += "  Entity.SetOrientation(Stage.entityList[curEnt], ";
   $result += "{x=" + $look.x + ",y=" + $look.y + ",z=" + $look.z + "}, ";
   $result += "{x=” + $up.x + ",y=" + $up.y + ",z=" + $up.z + "})\n";
}

$result += "\n  -- Tangible(Environmental physics) objects --\n  curEnt = ";
$result += "curEnt + 1\n  Stage.entityList[curEnt] = Entity.Create()";
$result += "\n  cId = Component:Build(\"EnvironmentCollisionComponent\")\n";
$result += "Entity.Attach(Stage.entityList[curEnt], cId)";

// Get tangibles(physics) object list and write it out.
$objs = getNodes("tan_*");

for($i=0;$i<size($objs);$i++)
{
   // Get attribute data.
   $center = eval("getAttr " + $objs[$i] + ".center;");
   $transX = $center.x;
   $transY = $center.y;
   $transZ = -$center.z;

   $rotX = eval("getAttr " + $objs[$i] + ".rotateX;");
```

```
  $rotY = eval("getAttr " + $objs[$i] + ".rotateY;");
  $rotZ = eval("getAttr " + $objs[$i] + ".rotateZ;");

  // Rotate back to axis aligned before getting bounding box.
  eval("rotate " + -$rotX + " " + -$rotY + " " + -$rotZ + " " + $objs[$i]);

  $scaleX = eval("getAttr " + $objs[$i] + ".boundingBox.boundingBoxSizeX;") / 2;
  $scaleY = eval("getAttr " + $objs[$i] + ".boundingBox.boundingBoxSizeY;") / 2;
  $scaleZ = eval("getAttr " + $objs[$i] + ".boundingBox.boundingBoxSizeZ;") / 2;

  // Rotate back.
  eval("rotate " + $rotX + " " + $rotY + " " + $rotZ + " " + $objs[$i]);

  // Calculate look and up vector from rotation.
  $rotX  = deg_to_rad($rotX);
  $rotY  = deg_to_rad($rotY);
  $rotZ  = deg_to_rad($rotZ);

  $look = rotateVec(<<0,0,1>>, $rotX, $rotY, $rotZ);
  $up = rotateVec(<<0,1,0>>, $rotX, $rotY, $rotZ);

  // Is this a ground plane?
  if (`gmatch $objs[$i] "*ground*"` > 0)
  {
    $result += "\n  Message.Post(Message:Build(\"AddGroundBox\", {x=";
  }
  // No, so just add a collision box.
  else
  {
    $result += "\n  Message.Post(Message:Build(\"AddEnvironmentBox\", ";
  }

  $result += ("{x=" + $up.x + ",y=" + $up.y + ",z=" + $up.z + "}, ");
  $result += ("{x=" + $look.x + ",y=" + $look.y + ",z=" + $look.z + "}, ");
  $result += ("{x=" + $scaleX + ",y=" + $scaleY + ",z=" + $scaleZ + "}, ");
  $result += ("{x=" + $transX + ",y=" + $transY + ",z=" + $transZ + "}))");
}

$result += "\n\n  -- Intangible objects --";

// Get intangibles(Scripting and dummy) object list and write it out.
$objs = getNodes("intan_*");

for($i=0;$i<size($objs);$i++)
{
  $result += "\n  curEnt = curEnt + 1";
  $result += "\n  Stage.entityList[curEnt] = Entity.Create()\n";

  // Get attribute data.
  $transX = eval("getAttr " + $objs[$i] + ".translateX;");
  $transY = eval("getAttr " + $objs[$i] + ".translateY;");
  $transZ = eval("getAttr " + $objs[$i] + ".translateZ;");

  $scaleX = eval("getAttr " + $objs[$i] + ".scaleX;");
  $scaleY = eval("getAttr " + $objs[$i] + ".scaleY;");
  $scaleZ = eval("getAttr " + $objs[$i] + ".scaleZ;");

  $rotX = eval("getAttr " + $objs[$i] + ".rotateX;");
  $rotY = eval("getAttr " + $objs[$i] + ".rotateY;");
```

```
   $rotZ = eval("getAttr " + $objs[$i] + ".rotateZ;");

   // Set positional info.
   $result += "  Entity.OffsetPosition(Stage.entityList[curEnt], ";
   $result += "{x=" + $transX + ",y=" + $transY + ",z=" + -$transZ + "})\n";

   // Set scale info.
   $result += "  Entity.SetScale(Stage.entityList[curEnt], ";
   $result += "{x=" + $scaleX + ",y=" + $scaleY + ",z=" + $scaleZ + "})\n";

   // Calculate look and up vector from rotation.
   $rotX  = deg_to_rad($rotX);
   $rotY  = deg_to_rad($rotY);
   $rotZ  = deg_to_rad($rotZ);

   $look = rotateVec(<<0,0,1>>, $rotX, $rotY, $rotZ);
   $up = rotateVec(<<0,1,0>>, $rotX, $rotY, $rotZ);

   // Output orientation information.
   $result += "  Entity.SetOrientation(Stage.entityList[curEnt], ";
   $result += "{x=" + $look.x + ",y=" + $look.y + ",z=" + $look.z + "}, ";
   $result += "{x=" + $up.x + ",y=" + $up.y + ",z=" + $up.z + "})\n";

   // Check for scripts.
   $attrList = eval("listAttr -st notes " + $objs[$i] + ";");
   if (size($attrList) == 1)
   {
     $scriptList = eval("getAttr " + $objs[$i] + ".notes;");
     $scriptCount = `tokenize $scriptList ";" $scripts`;

     // Attach scriptable component.
     $result += "  Entity.Attach(Stage.entityList[curEnt], ";
     $result += "Component:Build(\"LuaScriptable\"))\n";

     // Attach all scripts.
     for($j=0;$j<size($scripts);$j++)
     {
       $result += "  Entity.AttachScript(Stage.entityList[curEnt], \"";
       $result += "$scripts[$j] + "\")\n";
     }
   }
}

$result += "\n  Stage.entityNum = curEnt\nend";

// Write tables for the materials and meshes for this stage.
$result += ("\n\n" + $stageName + "Meshes = {");
for ($i=0;$i<size($archetypes);$i++)
{
  $result += ("\n  \"" + $archetypes[$i] + ".FBX\"");
}

$result += ("\n}\n\n" + $stageName + "Mats = {");
for ($i=0;$i<size($materials);$i++)
{
  $result += ("\n  \"" + $materials[$i] + "\"");
}
$result += "\n}";
```

```
  // Open file and write.
  int $fileID = `fopen $path "w"`;
  fprint $fileID $result;
  fclose $fileID;
}


global string $stageNameInput;
global string $win;
//=============================================================================
//    start
global proc startBuildState()
{
  global string $stageNameInput;
  global string $win;

  string $stageNameText = `textField -q -text $stageNameInput`;

  if( $stageNameText != "" )
  {
    string $path = (`internalVar -uwd` + "Room_" + $stageNameText + ".lua");
    print $path;

    writeLua($stageNameText, $path);
    deleteUI $win;
  }
  else
  {
    error "No stage name entered.";
  }
}



//=============================================================================
//    BuildStage
//=============================================================================
global proc BuildStage()
{
  global string $stageNameInput;
  global string $win;

  $win = `window -resizeToFitChildren true -title "Build Stage.."`;
      columnLayout -columnAttach "both" 5 -rowSpacing 10 -columnWidth 250 ;
      rowColumnLayout -numberOfColumns 2;
  text -label "Stage Name:";
  setParent ..;
  $stageNameInput = `textField -enterCommand "startBuildState();"`;
      button -label "Write Lua" -command "startBuildState();";

  showWindow $win;
}

BuildStage();
```