

```

1  /*
2  EnemyManager.cpp
3  Author(s): Antoine Campbell
4  Undead Arcade 2010(c)
5  */
6
7  #include <EnemyManager.h>
8  #include <irrXML.h>
9  #include <vector>
10
11 using namespace FatalTendencies;
12 using namespace irr;
13 using namespace io;
14
15 EnemyManager::EnemyManager(ISceneManager* smgr, IGUIEnvironment* guienv)
16 {
17     m_smgr = smgr;
18     m_guienv = guienv;
19 }
20 // loads enemies for a level from an xml file
21 void EnemyManager::LoadEnemiesXML(char* xmlFileName)
22 {
23     IrrXMLReader* xml = createIrrXMLReader(xmlFileName);
24     while(xml && xml->read())
25     {
26         std::string XMLgravity, XMLhealth, XMLspeed, XMLtype, XMLposx, XMLposy, XMLposz,
XMLmodelName, XMLrotx, XMLroty, XMLrotz;
27         switch(xml->getNodeTypes())
28         {
29             case EXN_ELEMENT:
30                 {
31                     //Capture enemy node data
32                     if(!strcmp("enemy", xml->getNodeName()))
33                     {
34                         XMLgravity = xml->getAttributeValue("gravity");
35                         XMLtype = xml->getAttributeValue("type");
36                         XMLhealth = xml->getAttributeValue("health");
37                         XMLspeed = xml->getAttributeValue("speed");
38                         XMLposx = xml->getAttributeValue("posx");
39                         XMLposy = xml->getAttributeValue("posy");
40                         XMLposz = xml->getAttributeValue("posz");
41                         XMLrotx = xml->getAttributeValue("rotx");
42                         XMLroty = xml->getAttributeValue("roty");
43                         XMLrotz = xml->getAttributeValue("rotz");
44                         XMLmodelName = xml->getAttributeValue("modelName");
45                     }
46
47                     //Create an enemy from this data and place enemy in the list
48                     if(!strcmp(XMLtype.c_str(), "zombie") || !strcmp(XMLtype.c_str(), "harpy"))
49                     {
50                         f32 posx, posy, posz, rotx, roty, rotz, speed;
51                         s32 health;
52                         bool gravity;
53                         stringw type = (stringw)XMLtype.c_str();

```

```

54     stringw modelFileName = (stringw)XMLmodelName.c_str();
55     vector3df position;
56
57     posx = (float)atof(XMLposx.c_str());
58     posy = (float)atof(XMLposy.c_str());
59     posz = (float)atof(XMLposz.c_str());
60     rotx = (float)atof(XMLrotx.c_str());
61     roty = (float)atof(XMLroty.c_str());
62     rotz = (float)atof(XMLrotz.c_str());
63     speed = (float)atof(XMLspeed.c_str());
64     health = (int)atof(XMLhealth.c_str());
65     gravity = atof(XMLgravity.c_str());
66
67     position.X = posx;
68     position.Y = posy;
69     position.Z = posz;
70
71     //Add a new enemy to the enemyList
72     EnemyObject* enemy = AddEnemyObject(health, speed, type);
73
74     //call load content on the enemyObject
75     if(type=="harpy")
76     {
77         ((Harpy*)enemy)->LoadContent(modelFileName, position);
78     }
79     else if(type=="zombie")
80     {
81         ((Zombie*)enemy)->LoadContent(modelFileName, position);
82     }
83     else
84     {
85         enemy->LoadContent(modelFileName, position);
86     }
87     }
88 }
89 }
90 }
91 }
92 // Add enemies to list hard-coded style this is temporary
93 EnemyObject* EnemyManager::AddEnemyObject(s32 startHealth, f32 speed, stringw type)
94 {
95     EnemyObject* enemy;
96     //Detemine what kind of enemy it is and then create it
97     //then add it to the list
98     if(type == "zombie")
99     {
100         enemy = new Zombie(startHealth, m_smgr, m_guienv, speed, "zombie");
101     }
102     if(type == "corpse")
103     {
104         enemy= new Corpse(startHealth, m_smgr, m_guienv, speed, "corpse");
105         listOfCorpses.push_back(enemy);
106     }
107     if(type == "harpy")

```

```
108     {
109         enemy = new Harpy(startHealth, m_smgr, m_guienv, speed, "harpy");
110     }
111
112     m_enemyList.push_back(enemy);
113     return enemy;
114 }
115 // return the enemyList
116 std::vector<ISceneNode*> EnemyManager::GetEnemies()
117 {
118     std::vector<ISceneNode*> list;
119     //Cycle through enemyList
120     for(unsigned int i=0; i<m_enemyList.size(); i++)
121     {
122         list.push_back(m_enemyList[i]->getNode());
123     }
124     return list;
125 }
126 // Update all the enemies in the list
127 void EnemyManager::Update(f32 frameDeltaTime)
128 {
129     //Cycle through enemyList and call updates or destructors
130     for(unsigned int i=0; i<m_enemyList.size() ;i++)
131     {
132         if(m_enemyList[i]->kill == true)
133         {
134             RemoveEnemy(m_enemyList[i]->getNode());
135         }
136         else if(m_enemyList[i]->dead==true)
137         {
138             m_enemyList[i]->getNode()->setPosition(vector3df(0,-300,0));
139             m_enemyList[i]->kill = true;
140         }
141         else if(m_enemyList[i]->enemyType==0)
142         {
143             ((Zombie*)m_enemyList[i])->Update(frameDeltaTime);
144         }
145         else if(m_enemyList[i]->enemyType==1)
146         {
147             ((Corpse*)m_enemyList[i])->Update(frameDeltaTime);
148         }
149         else if(m_enemyList[i]->enemyType==2)
150         {
151             ((Harpy*)m_enemyList[i])->Update(frameDeltaTime);
152         }
153     }
154 }
155 //Use collision manager to set up collision lists for each enemy
156 void EnemyManager::SetCollisionList(CollisionManager* collisionManager)
157 {
158     m_collisionManager=collisionManager;
159
160     //If the listOfCorpses is not empty call the function to add them to the
    collisionManager
```

```
161     if(listOfCorpses.size() > 0)
162     {
163         EnemyManager::AddCorpsesToCollisionManager(collisionManager);
164     }
165     //Cycle through enemyList and sends them to the collision manager
166     for(unsigned int i=0; i<m_enemyList.size(); i++)
167     {
168         m_enemyList[i]->SetCollisionList(collisionManager->PopulateCollisionList(
169     m_enemyList[i]->getNode()->getTransformedBoundingBox()));
170     }
171     //If the listOfCorpses is not empty it will pop all of them onto the collision list
172     void EnemyManager::AddCorpsesToCollisionManager(CollisionManager* collisionManager)
173     {
174         std::vector<ISceneNode*> tempList;
175
176         //Cycle through listOfCorpses and push them onto the enemy list
177         for(unsigned int x=0; x<listOfCorpses.size(); x++)
178         {
179             tempList.push_back(listOfCorpses[x]->getNode());
180         }
181
182         collisionManager->AddCharacters(tempList);
183
184         listOfCorpses.clear();
185     }
186     //Sets the sceneNodes of the players in list format
187     void EnemyManager::SetPlayerList(std::vector<ISceneNode*> players)
188     {
189         listOfPlayers=players;
190
191         //Cycle through enemyList and give each enemy the players
192         for(unsigned int x=0; x < m_enemyList.size(); x++)
193         {
194             m_enemyList[x]->SetPlayerList(listOfPlayers);
195         }
196     }
197     //Gets the players flag so it can be passed to the harpies
198     void EnemyManager::SetPlayerFlag(Flag* flag)
199     {
200         playerFlag=flag;
201
202         //Cycle through enemyList and if its a harpy gives it flag info
203         for(unsigned int x=0; x < m_enemyList.size(); x++)
204         {
205             if(m_enemyList[x]->enemyType==2)
206             {
207                 ((Harpy*)m_enemyList[x])->SetPlayersFlag(flag);
208             }
209         }
210     }
211     //Remove single enemy
212     void EnemyManager::RemoveEnemy(ISceneNode* node)
213     {
```

```

214     std::vector<EnemyObject*>::iterator it;
215     int i = 0;
216     //Cycle through enemyList and remove a specific enemy
217     for(it=m_enemyList.begin(); it!=m_enemyList.end(); it++)
218     {
219         if(m_enemyList[i]->getNode() == node)
220         {
221             m_collisionManager->RemoveCharacter(node);
222             m_enemyList[i]->RemoveEnemy();
223             m_enemyList[i] = NULL;
224             it = m_enemyList.erase(it);
225             return;
226         }
227         i++;
228     }
229 }
230 //Remove all enemies
231 void EnemyManager::RemoveEnemies()
232 {
233     //Cycle through enemyList and remove all enemies
234     for(unsigned int i=0; i<m_enemyList.size(); i++)
235     {
236         m_collisionManager->RemoveCharacter(m_enemyList[i]->getNode());
237         m_enemyList[i]->RemoveEnemy();
238     }
239     m_enemyList.clear();
240 }
241 //Remove corpse
242 void EnemyManager::RemoveCorpse(ISceneNode* node)
243 {
244     std::vector<EnemyObject*>::iterator it;
245     int i=0;
246     //Cycle through enemyList
247     for(it=m_enemyList.begin(); it!=m_enemyList.end(); it++)
248     {
249         if(node == m_enemyList[i]->getNode() && m_enemyList[i]->getNode()->getName() ==
"corpse")
250         {
251             m_collisionManager->RemoveCharacter(node);
252             m_enemyList[i]->RemoveEnemy();
253             m_enemyList[i] = NULL;
254             it = m_enemyList.erase(it);
255             return;
256         }
257         i++;
258     }
259 }
260 //Remove all corpses
261 void EnemyManager::RemoveAllCorpses()
262 {
263     std::vector<EnemyObject*>::iterator it;
264     int i = 0;
265     //Cycle through enemyList
266     for(it=m_enemyList.begin(); it!=m_enemyList.end(); it++)

```

```
267     {
268         ISceneNode* node = m_enemyList[i]->getNode();
269         //if it is a corpse remove it from the list
270         if(node->getName() == "corpse")
271         {
272             RemoveCorpse(node);
273         }
274         i++;
275     }
276 }
277 //Return specific enemy object;
278 EnemyObject* EnemyManager::GetEnemy(ISceneNode* node)
279 {
280     //Cycle through enemyList
281     for(unsigned int i=0; i<m_enemyList.size(); i++)
282     {
283         if(node == m_enemyList[i]->getNode())
284         {
285             //if its the right node return it
286             return m_enemyList[i];
287         }
288     }
289     return NULL;
290 }
```