

```
1 #include "Lujan.h"
2 #include "Map.h"
3
4 Timer Lujan::waveTimer = Timer();
5 int Lujan::waveNum = 1;
6 float Lujan::waveTime = 0.0f;
7 float Lujan::maxWaveNum = 0.0f;
8
9 Lujan::Lujan(ISceneManager* smgr, bool humanPlayer)
10 {
11     enemyStartLine = 500.0f;
12     this->smgr = smgr;
13
14     IAnimatedMeshSceneNode* lujan = smgr->addAnimatedMeshSceneNode(smgr->getMesh(
15         "media/irrlischt/dwarf.x"));
16     lujan->setPosition(vector3df(0,0,1000));
17     lujan->setScale(vector3df(15));
18     lujan->setRotation(vector3df(0,0,0));
19     //lujan->setMaterialFlag(video::EMF_LIGHTING, false);
20     lujan->setMaterialFlag(video::EMF_NORMALIZE_NORMALS, true);
21     //lujan->addShadowVolumeSceneNode(0, -1, true, 10000);
22     lujan->setFrameLoop(0,0);
23
24     isHuman = humanPlayer;
25     waveNum = 1;
26     waveTime = 60.0f;
27     waveTimer = Timer(waveTime);
28     scoreAccumulated = 0;
29
30     //initialize fuzzy logic variables
31     creationWaitWeight = 0.6f;
32     minEnemies = 0;
33     maxEnemies = 16;
34     numEnemiesWeight = 0.2f;
35     waveTimeWeight = 0.2f;
36     minWaveNum = 1;
37     maxWaveNum = 5;
38     waveNumWeight = 0.25f;
39     randChanceWeight = 0.35f;
40
41     //create points to be added each wave
42     //80, 100, 120, 140, 160
43     pointsPerWave = new int[(int)maxWaveNum];
44     pointsPerWave[0] = 80;
45     for (unsigned int i = 1; i < maxWaveNum; i++)
46     {
47         pointsPerWave[i] = pointsPerWave[i-1] + 20;
48     }
49     enemyPoints = pointsPerWave[(int)minWaveNum - 1];
50     enemyPrices = vector3di(1, 2, 3);
51     creationTimer = Timer(0.75f);
52 }
53
54 std::vector<Enemy*> Lujan::GetEnemies()
55 {
56     return enemies;
57 }
58
59 int Lujan::GetWaveNum()
60 {
61     return waveNum;
62 }
63
64 int Lujan::GetWaveTimer()
65 {
66     if (waveNum < maxWaveNum)
```

```
66     {
67         return (int)waveTime - (int)waveTimer.ElapsedSeconds();
68     }
69     return 0;
70 }
71
72 bool Lujan::IsDefeated()
73 {
74     return waveNum >= maxWaveNum && enemyPoints == 0 && enemies.size() == 0;
75 }
76
77 int Lujan::CityDamage()
78 {
79     float damage = 0;
80     for(unsigned int i=0; i<enemies.size(); i++)
81     {
82         if(enemies[i]->AttackCityDamage())
83         {
84             damage += enemies[i]->GetStrength();
85
86             //--> This would cause popup damage bubbles at the wall.
87             //vector3df pos = enemies[i]->GetMesh()->getPosition();
88             //scoreBubbles.push_back(new ScoreBubble(smgr, pos, 1, 5));
89         }
90     }
91     return (int) damage;
92 }
93 }
94
95 int Lujan::GetScoreAccumulated()
96 {
97     int hold = scoreAccumulated;
98     scoreAccumulated = 0;
99     return hold;
100 }
101
102 void Lujan::update(float elapsedTime)
103 {
104     //advance to next wave
105     if (waveNum < maxWaveNum && waveTimer.Ready())
106     {
107         enemyPoints += pointsPerWave[waveNum];
108         waveNum++;
109     }
110
111     if (isHuman)
112     {
113         //do some player controls stuff
114     }
115     else
116     {
117         if (enemyPoints > 0 && creationTimer.Ready() &&
118             CalcFuzzyNumEnemies(Bears) * creationWaitWeight < RandomChance())
119         {
120             switch ((int)ChooseEnemy())
121             {
122                 case (int)Bears:
123                     enemies.push_back(new Bear(smgr, vector3df(-1500.0f + rand() % 3000,
124 0, enemyStartLine)));
125                     enemyPoints -= enemyPrices.Z;
126                     break;
127                 case (int)Monkeys:
128                     enemies.push_back(new Monkey(smgr, vector3df(-1500.0f + rand() % 3000,
129 , 0, enemyStartLine)));
130                     enemyPoints -= enemyPrices.Y;
131                     break;
```

```
130         case (int)Bunnies:
131             enemies.push_back(new Bunny(smgr, vector3df(-1500.0f + rand() % 3000,
132             0, enemyStartLine)));
133             enemyPoints -= enemyPrices.X;
134             break;
135         }
136     }
137
138     //update enemies
139     for(unsigned int i = 0; i < enemies.size(); i++)
140     {
141         enemies[i]->update(elapsedTime);
142
143         if(enemies[i]->JustKilled())
144         {
145             vector3df pos = enemies[i]->GetMesh()->getPosition();
146
147             int addedScore = (int)((enemyStartLine - Map::GetCityBorder()) - enemies
148 [i]->GetMesh()->getAbsolutePosition().Z);
149             scoreAccumulated += addedScore;
150             scoreBubbles.push_back(new ScoreBubble(smgr, pos, addedScore, 5000));
151         }
152         else if(enemies[i]->IsDead())
153         {
154             enemies[i]->GetMesh()->remove();
155             enemies.erase(enemies.begin() + i);
156
157             i--;
158         }
159         else if(enemies[i]->GetMesh()->getAbsolutePosition().Z <= Map::GetCityBorder
160 ())
161         {
162             enemies[i]->SetAttackCity(true);
163         }
164     }
165
166     for(int i=0;i<(int)scoreBubbles.size();i++){
167         scoreBubbles[i]->Update(elapsedTime);
168
169         if(scoreBubbles[i]->remove){
170             //scoreBubbles.erase(scoreBubbles.begin() + i);
171             //i--;
172         }
173     }
174
175     Lujan::EnemyType Lujan::ChooseEnemy()
176     {
177         float bestFuzzyValue = 0.0f;
178         EnemyType bestType;
179
180         for (int i = (int)Bunnies; i <= (int)Bears; i++)
181         {
182             if ((i == (int)Bears && enemyPoints < enemyPrices.Z) ||
183                 (i == (int)Monkeys && enemyPoints < enemyPrices.Y) ||
184                 (i == (int)Bunnies && enemyPoints < enemyPrices.X))
185             {
186                 continue;
187             }
188
189             float fuzzy = 0.0f;
190             fuzzy += CalcFuzzyNumEnemies((EnemyType)i) * numEnemiesWeight;
191             fuzzy += CalcFuzzyWaveTime((EnemyType)i) * waveTimeWeight;
192             fuzzy += CalcFuzzyWaveNum((EnemyType)i) * waveNumWeight;
```

```
193         fuzzy += RandomChance() * randChanceWeight;
194
195         if (fuzzy >= bestFuzzyValue)
196         {
197             bestFuzzyValue = fuzzy;
198             bestType = (EnemyType)i;
199         }
200     }
201
202     return bestType;
203 }
204
205 float Lujan::RandomChance()
206 {
207     return (rand() % 100) / 100.0f;
208 }
209
210 float Lujan::CalcFuzzyNumEnemies(EnemyType t)
211 {
212     float fuzzy;
213     //make 1.0 = avg enemies for monkeys
214     if (t == Monkeys)
215     {
216         float avgEnemies = (minEnemies + maxEnemies) / 2.0f;
217         fuzzy = 1.0f - abs((enemies.size() - avgEnemies) / avgEnemies);
218     }
219     else
220     {
221         //make 1.0 = max enemies for bears
222         fuzzy = (enemies.size() - minEnemies) / (maxEnemies - minEnemies);
223     }
224     if (fuzzy < 0.0f) { fuzzy = 0.0f; }
225     else if (fuzzy > 1.0f) { fuzzy = 1.0f; }
226
227     //make 1.0 = min enemies for bunnies
228     if (t == Bunnies) { fuzzy = 1.0f - fuzzy; }
229
230     return fuzzy;
231 }
232
233 float Lujan::CalcFuzzyWaveTime(EnemyType t)
234 {
235     float fuzzy;
236     float wTimer = (float)GetWaveTimer();
237     //make 1.0 = middle of wave for monkeys
238     if (t == Monkeys)
239     {
240         float avgWaveTime = waveTime / 2.0f;
241         fuzzy = 1.0f - abs((wTimer - avgWaveTime) / avgWaveTime);
242     }
243     else
244     {
245         //make 1.0 = beginning of wave for bunnies
246         fuzzy = wTimer / waveTime;
247     }
248     if (fuzzy < 0.0f) { fuzzy = 0.0f; }
249     else if (fuzzy > 1.0f) { fuzzy = 1.0f; }
250
251     //make 1.0 = end of wave for bears
252     if (t == Bears) { fuzzy = 1.0f - fuzzy; }
253
254     return fuzzy;
255 }
256
257 float Lujan::CalcFuzzyWaveNum(EnemyType t)
258 {
```

```
259     float fuzzy;
260     //make 1.0 = avg wave for monkeys
261     if (t == Monkeys)
262     {
263         float avgWaveNum = (minWaveNum + maxWaveNum) / 2.0f;
264         fuzzy = 1.0f - abs((waveNum - avgWaveNum) / avgWaveNum);
265     }
266     else
267     {
268         //make 1.0 = last wave for bears
269         fuzzy = (waveNum - minWaveNum) / (maxWaveNum - minWaveNum);
270     }
271     if (fuzzy < 0.0f) { fuzzy = 0.0f; }
272     else if (fuzzy > 1.0f) { fuzzy = 1.0f; }
273
274     //make 1.0 = min wave for bunnies
275     if (t == Bunnies) { fuzzy = 1.0f - fuzzy; }
276
277     return fuzzy;
278 }
```