SceneGraphを用いたゲームプログラムの為のテスト作成手法

055722G 小林 佑亮　指導教員：河野真治

1　研究背景・目的

当研究室では学生実験において、PlayStation3 を用いた並列的なゲームプログラムの作成を行っている。そこで使用されるのが当研究室で開発した Cerium レンダリングエンジンである。Cerium を用いたゲームプログラムの例題として PlayStation2 で動作していた超弾帝 (スーパーダンディ) というゲームの移植を行った。
その際、オブジェクトの描画が正常に行われない、衝突判定の位置ずれが生じる等の不具合が発生した。超弾帝には約 100 種類に及ぶオブジェクトが存在し、今後も同様な不具合を修正していくとすると、学生実験の進行に大きな支障をきたす事となる。
本研究では Cerium の中でも SceneGraph に着目し、SceneGraph 単位でのテストを行うことで、、オブジェクトごとの振る舞いや描画をチェックする。これにより、ゲームのデバッグを容易にし、今後の学生実験におけるゲームの移植や改良、作成を円滑にすることができる。

2　SceneGraphによるゲームフレームワーク

2.1　SceneGraphの特徴

ゲームの中の一つの場面 (Scene) を構成するオブジェクトやその振る舞い、ゲームのルールの集合を SceneGraph とする。SceneGraph のノードは親子関係を持つ Tree で構成される。(図 1)

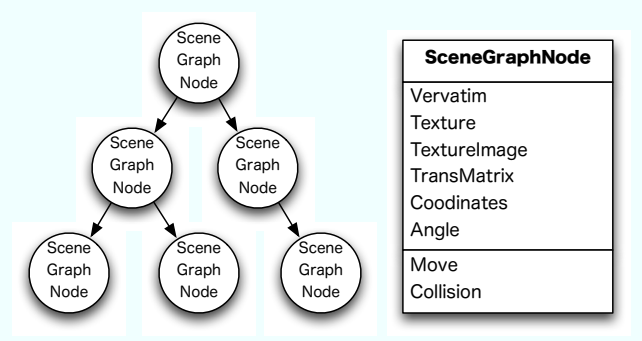

図 1: SceneGraph

親子関係とは、親オブジェクトの回転や並行移動等の行列計算による頂点座標の変更が子オブジェクトにも反映する関係の事である。これは子に対してスタックに積まれた親の変更行列を掛ける事で実現できる。

2.2　SceneGraphを用いたゲームプログラム

ゲーム内で使用するオブジェクトは Blender というオープンソースの 3 次元コンピュータグラフィックスソフトウェアで作成する。作成したオブジェクトモデルは python スクリプトにより xml ファイルに変換され、それを元に SceneGraph が生成される。生成した SceneGraph は set_move_collision という関数によって move や collision が実装される。

2.2.1　move

オブジェクトの動作や状態遷移を記述する。各オブジェクトは複数の move 関数を持っており、set_move_collision 関数によって状態遷移する。簡単なプログラム例を以下に示す。

```
static void
boss1_move_right(SceneGraphPtr node, int screen_w, int screen_h) {
  node->xyz[0] += node->stack_xyz[0];
  if(node->xyz[0] > screen_w-280) {
    node->set_move_collision(boss1_move_left, boss1_collision);
  }
}

static void
boss1_move_left(SceneGraphPtr node, int screen_w, int screen_h) {
  node->xyz[0] -= node->stack_xyz[0];
  if(node->xyz[0] < 280) {
     node->set_move_collision(boss1_move_right, boss1_collision);
  }
}
```

オブジェクトは boss1_move_left によって左に移動する。しかし、そのままだといずれ画面外に消えてしまう。そこで boss1_move_left 関数中で条件分岐を立て、画面外ぎりぎりの座標になったら set_move_collision で move_right に切り替え、右移動に切り替える。しかし、このままでもいずれ画面外に消えてしまう為、再び set_move_collision に切り替える。これを繰り返す事により、オブジェクトが常に画面内に描画されている状態を保つ事が出来る。

2.2.2　collision

オブジェクトの衝突判定を記述する。オブジェクト同士が触れた時に行う動作を決定する。例えばシューティングゲームなら弾丸が敵や自機に当たった時にダメージを受ける、ゲームオーバーになる、などの判定を行う。こちらも条件分岐により、set_move_collision で状態遷移する。

3 CppUnit を用いた SceneGraph のテスト

3.1 CppUnit とは

CppUnit は xUnit の一つであり、C++の単体テストを自動化する frame work である。 CppUnit の特徴は、テストケースを増やす事が容易であり、1 つの事象に対して様々なテストケースを同時にテストする事が出来る。また、羅列したテストケースは一括で実行、結果の表示ができる。(図 2)

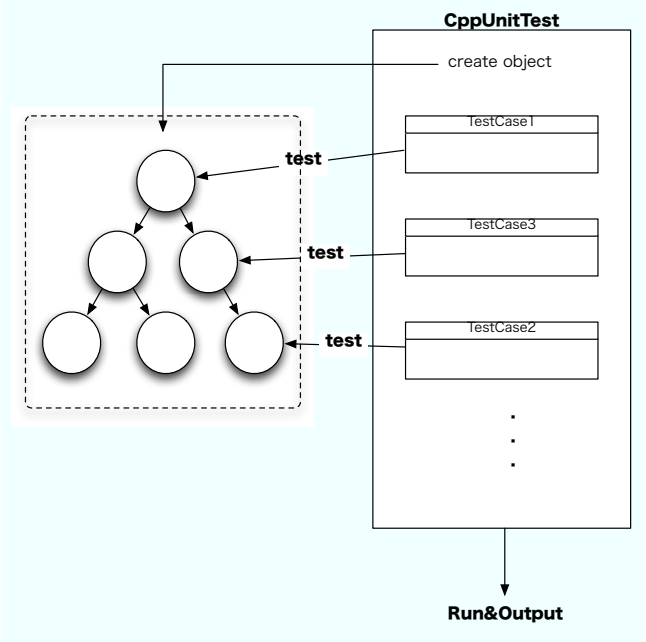

図 2: CppUnitTest

3.2 ゲームプログラムのテスト

3 つの SceneGraph を持つオブジェクトのテストを行った。このオブジェクトは本体の他に左右にパーツを 1 つずつ持つ。本体を Tree の root として左右のパーツがその子供になっている。

ここで、各オブジェクトの SceneGraph はその親や子、兄弟に対するアドレスを保持している。従って、パーツオブジェクトのテストを行いたい場合は、そのオブジェクトの Root である本体から辿れば、パーツオブジェクトの各パラメータを参照する事が出来る。

そこで、Root のアドレスを取得する getSGP() という関数を作成した。getSGP() 関数によって取得してきた Root のアドレスを使って、各オブジェクトの座標を取得し、その初期化が正しいか、状態遷移において正しい値を保持いるかテストした。

```
void
sgTest::rootTest() {
  test_init();

  sg_root->print_member();
  CPPUNIT_ASSERT_EQUAL((float)width/2, sg_root->xyz[0]);
  CPPUNIT_ASSERT_EQUAL(0.0f, sg_root->xyz[1]);
  CPPUNIT_ASSERT_EQUAL(-100.0f, sg_root->xyz[2]);
}

void
sgTest::childTest() {
  while (sg_root) {
    if(sg_root->children != NULL) {
      sg_root->children->print_member();
...
      sg_root = sg_root->children;
    } else if(sg_root->brother != NULL) {
      sg_root->brother->print_member();
      CPPUNIT_ASSERT_EQUAL(0.0f, sg_root->brother->xyz[0]);
...
      sg_root = sg_root->brother;
    } else {
...
```

テストの結果、全てのオブジェクトが正常に初期化されているのが確認出来た。しかし、move,collision 中の各オブジェクトの座標は確認出来なかった。これはテストを走らせた時点で全てのテストケースを同時にテストしている為、全てのテストケースにおいて初期化がなされた時点での座標の情報しか参照できない為である。

4 評価

CppUnit によるテストはオブジェクトの座標などの初期値はテスト出来るが、move,collision によって変化した値をテストするのには不向きである。よって CppUnit は move,collision の各関数を抜き出して状態遷移をするかどうかの条件分岐に対して適用するのが適当である。

参考文献

[1] 宮國 渡, 河野 真治, 神里 晃, 杉山 千秋: Cell 用の Fine-grain Task Manager の実装, 第 108 回システムソフトウェアとオペレーティング・システム研究会 (2008)

[2] 高橋 寿一:知識ゼロから学ぶソフトウェアテスト, 翔泳社 (2005)

[3] 伊藤 喜一:CppUnit 入門, (http://www.ogis-ri.co.jp/otc/hiroba/technical/CppUnit/)