CollisionTest.cpp


```
#include "CollisionTest.h"

#include "Matrix.h"
#include "Vector.h"
#include "Polygon.h"

using namespace blaster;

const CollisionTest CollisionTest::registerThis;

void CollisionTest::runTest( std::ostream & out )
{
    Vector verts[6];

    verts[0] = Vector( 0, -1 );
    verts[1] = Vector( -1, 1 );
    verts[2] = Vector( 1, 1 );

    verts[3] = Vector( 0, -1 );
    verts[4] = Vector( -1, 1 );
    verts[5] = Vector( 1, 1 );

    Matrix m1;
    Matrix m2;

    Polygon p1( 0, 1, 2, &verts[0], &verts[1], &verts[2] );
    Polygon p2( 3, 4, 5, &verts[3], &verts[4], &verts[5] );

    m1 *= Matrix::translateMatrix( 0, 0, 0 );
    m2 *= Matrix::translateMatrix( 0, 0, 0 );

    p1.transform( m1 );
    p2.transform( m2 );

    assert( p1.collides( &p2 ) == TRUE );

    m1.loadIdentity();
    m2.loadIdentity();

    m1 *= Matrix::translateMatrix( 5, 0, 0 );
    m2 *= Matrix::translateMatrix( 7, 0, 0 );

    p1.transform( m1 );
    p2.transform( m2 );

    assert( p1.collides( &p2 ) == TRUE );

    m2 *= Matrix::rotateZMatrix( M_PI );

    p2.transform( m2 );

    assert( p1.collides( &p2 ) == FALSE );

    out << "CollisionTest completed successfully" << std::endl;
}
```

CollisionTest.h


```
#ifndef COLLISIONTEST_H
#define COLLISIONTEST_H

#include "TestBase.h"

namespace blaster
{

    class CollisionTest:public TestBase
    {
      public:
        CollisionTest( void ):TestBase(  )
        {
        }

        virtual void runTest( std::ostream & out );

      private:
        static const CollisionTest registerThis;
    };

}

#endif
```

MessageTest.cpp


```
#include "MessageTest.h"
#include "ClientNetworkController.h"
#include "ServerNetworkController.h"
#include "TestMessage.h"
#include "TestMessageMaker.h"

using namespace blaster;

const MessageTest MessageTest::registerThis;

void MessageTest::runTest( std::ostream & out )
{
    ClientNetworkController *client =
        ClientNetworkController::getInstance(  );

    ServerNetworkController *server =
        ServerNetworkController::getInstance(  );

    server->startServer( 1337 );

    bool connected = client->connect( "localhost", 1337 );

    assert( connected );

    client->think( 1.0f, 0.0f );
    server->think( 1.0f, 0.0f );

    NetMessage *msg = new TestMessage( 1.0f );

    client->UDPSendMessage( msg );

    server->think( 2.0f, 0.0f );

    assert( TestMessage::recieved == TRUE );

    client->disconnect(  );

    out << "MessageTest completed successfully" << std::endl;
}
```

MessageTest.h


```
#ifndef MESSAGETEST_H
#define MESSAGETEST_H

#include "TestBase.h"

namespace blaster
{

    class MessageTest:public TestBase
    {
      public:
        MessageTest( void ):TestBase(  )
        {
        }

        virtual void runTest( std::ostream & out );

      private:
        static const MessageTest registerThis;
    };

}

#endif
```

PlayerAddTest.cpp


```
#include "PlayerAddTest.h"

#include "Player.h"
#include "GameController.h"

using namespace blaster;

const PlayerAddTest PlayerAddTest::registerThis;

void PlayerAddTest::runTest( std::ostream & out )
{
    Player *player = new Player(  );

    player->setID( 42 );

    GameController *game = GameController::getInstance(  );

    game->addPlayer( player );

    assert( player == game->getPlayer( 42 ) );

    game->removePlayer( 42 );

    assert( game->getPlayer( 42 ) == NULL );

    out << "PlayerAddTest completed successfully." << std::endl;
}
```

PlayerAddTest.h


```
#ifndef PLAYERADDTEST_H
#define PLAYERADDTEST_H

#include "TestBase.h"

namespace blaster
{

    class PlayerAddTest:public TestBase
    {
      public:
        PlayerAddTest( void ):TestBase(  )
        {
        }

        virtual void runTest( std::ostream & out );

      private:
        static const PlayerAddTest registerThis;
    };

}

#endif
```

ShipDeathTest.cpp


```
#include "ShipDeathTest.h"

#include "Player.h"
#include "Level.h"
#include "Ship.h"
#include "GameController.h"

using namespace blaster;

const ShipDeathTest ShipDeathTest::registerThis;

void ShipDeathTest::runTest( std::ostream & out )
{
    GameController *game = GameController::getInstance(  );

    Player *p = new Player(  );

    p->setID( 42 );

    game->addPlayer( p );

    // Give the player a chance to spawn a ship
    game->think( 1.0f, 0.0f );

    // Now the player ought to have a ship
    assert( p->getShip(  ) != NULL );

    // Now lets take it away from him
    float damage = p->getShip(  )->energyCapacity(  ) + 1.0f;

    // bam!
    p->getShip(  )->takeDamage( damage );

    // Give the ship a chance to remove itself
    game->think( 1.1f, 0.0f );

    // Now there should be no ship
    assert( p->getShip(  ) == NULL );

    // Advance the time below the respawn threshold
    game->think( 1.2f, 0.0f );

    // Still no ship, right?
    assert( p->getShip(  ) == NULL );

    // Let the player respawn a ship
    game->think( 5.0f, 0.0f );

    // Make sure that the respawn works
    assert( p->getShip(  ) != NULL );

    out << "ShipDeathTest completed successfully" << std::endl;

    delete p;
}
```

ShipDeathTest.h


```
#ifndef SHIPDEATHTEST_H
#define SHIPDEATHTEST_H

#include "TestBase.h"

namespace blaster
{

    class ShipDeathTest:public TestBase
    {
      public:
        ShipDeathTest( void ):TestBase(  )
        {
        }

        virtual void runTest( std::ostream & out );

      private:
        static const ShipDeathTest registerThis;
    };

}

#endif
```

TestBase.cpp


```
#include "TestBase.h"

using namespace blaster;

Tests TestBase::tests;

TestBase::TestBase( void )
{
    tests.push_back( this );
}

void TestBase::runAllTests( std::ostream & out )
{
    for ( Tests::iterator i = tests.begin(  ); i != tests.end(  ); i++ )
    {
        ( *i )->runTest( out );
    }
}

int main( int argc, char **argv )
{
    TestBase::runAllTests( std::cout );

    return 0;
}
```

TestBase.h


```
#ifndef TESTBASE_H
#define TESTBASE_H

#include <assert.h>
#include <list>
#include <iostream>
#include <string>

namespace blaster
{

    class TestBase;

    typedef std::list < TestBase * >Tests;

    class TestBase
    {
      public:
        TestBase( void );

        virtual void runTest( std::ostream & out ) = 0;

        static void runAllTests( std::ostream & out );

      private:
        static Tests tests;
    };

}

#endif
```