GAMES TECHNOLOGY - COURSEWORK

UG-Rehman, Abdul

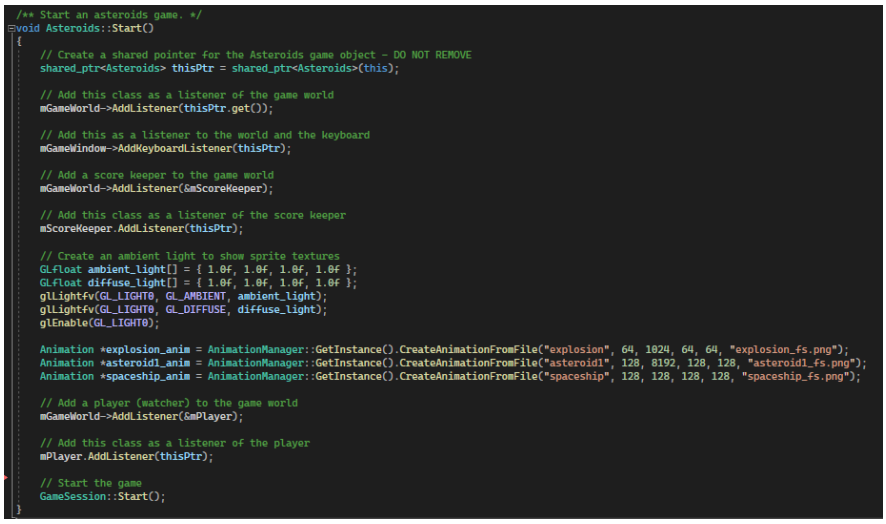
IN2026

Contents

Part I

A start screen.

To Implement start, screen I removed the code that created spaceship, asteroids and GUI containing lives and score from the Start method in the Asteroids.cpp class. This is what the method should look after removing the code:



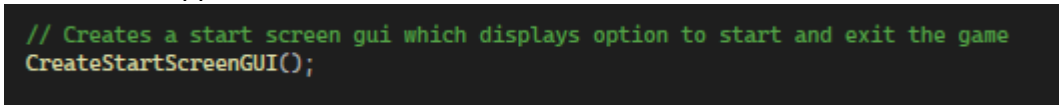
```
/** Start an asteroids game. */
void Asteroids::Start()
{
    // Create a shared pointer for the Asteroids game object - DO NOT REMOVE
    shared_ptr<Asteroids> thisPtr = shared_ptr<Asteroids>(this);

    // Add this class as a listener of the game world
    mGameWorld->AddListener(thisPtr.get());

    // Add this as a listener to the world and the keyboard
    mGameWindow->AddKeyboardListener(thisPtr);

    // Add a score keeper to the game world
    mGameWorld->AddListener(&mScoreKeeper);

    // Add this class as a listener of the score keeper
    mScoreKeeper.AddListener(thisPtr);

    // Create an ambient light to show sprite textures
    GLfloat ambient_light[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat diffuse_light[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_light);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse_light);
    glEnable(GL_LIGHT0);

    Animation *explosion_anim = AnimationManager::GetInstance().CreateAnimationFromFile("explosion", 64, 1024, 64, 64, "explosion_fs.png");
    Animation *asteroid1_anim = AnimationManager::GetInstance().CreateAnimationFromFile("asteroid1", 128, 8192, 128, 128, "asteroid1_fs.png");
    Animation *spaceship_anim = AnimationManager::GetInstance().CreateAnimationFromFile("spaceship", 128, 128, 128, 128, "spaceship_fs.png");

    // Add a player (watcher) to the game world
    mGameWorld->AddListener(&mPlayer);

    // Add this class as a listener of the player
    mPlayer.AddListener(thisPtr);

    // Start the game
    GameSession::Start();
}
```

This method will start an empty game session with all the listeners needed for the game to run as well as animations.

Then I created two new GUI method in the asteroids.cpp to create the gui which will be displayed on the empty game session to provide instruction of how to start and quit the game.

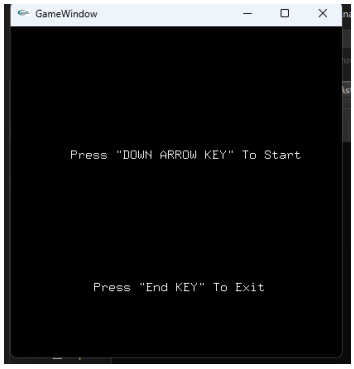
```
void Asteroids::CreateStartScreenGUI() {
    // Create a new GUILabel and wrap it up in a shared_ptr
    mStartGame = make_shared<GUILabel>("Press \"DOWN ARROW KEY\" To Start");
    // Set the vertical alignment of the label to GUI_VALIGN_BOTTOM
    mStartGame->SetVerticalAlignment(GUIComponent::GUI_VALIGN_BOTTOM);
    // Add the GUILabel to the GUIComponent
    shared_ptr<GUIComponent> start_game_component = static_pointer_cast<GUIComponent>(mStartGame);
    mGameDisplay->GetContainer()->AddComponent(start_game_component, GLVector2f(0.18f, 0.6f));

    // Create a new GUILabel and wrap it up in a shared_ptr
    mExitGame = make_shared<GUILabel>("Press \"End KEY\" To Exit");
    // Set the vertical alignment of the label to GUI_VALIGN_BOTTOM
    mExitGame->SetVerticalAlignment(GUIComponent::GUI_VALIGN_BOTTOM);
    // Add the GUILabel to the GUIComponent
    shared_ptr<GUIComponent> exit_game_component = static_pointer_cast<GUIComponent>(mExitGame);
    mGameDisplay->GetContainer()->AddComponent(exit_game_component, GLVector2f(0.25f, 0.2f));

}
void Asteroids::RemoveStartScreenGUI() {
    mStartGame->SetVisible(false);
    mExitGame->SetVisible(false);
}
```

Then added the CreateStartScreenGUI method to the start method of the asteroids.cpp class.



```
// Creates a start screen gui which displays option to start and exit the game
CreateStartScreenGUI();
```

After making the empty session and adding the start screen Gui. The game window's appearance looks like this:



To implement the start and exit I added another case to the OnSpecialKepPressed method of the asteroids.cpp class.

```
case GLUT_KEY_DOWN:
    // Create a spaceship and add it to the world
    mGameWorld->AddObject(CreateSpaceship());
    // Create some asteroids and add them to the world
    CreateAsteroids(10);
    //removes start screen gui
    RemoveStartScreenGUI();
    //Create the GUI
    CreateGUI();
    break;
```

I am using the down arrow key as the means to start the game. The code in the above snippet will create and add spaceship, asteroid objects which would start the game. It will also remove the start screen Gui and add the game Gui which display the lives and score of the player.

```
case GLUT_KEY_END:
    Stop();
    break;
```

The case above is to exit the game once the End Key is pressed.

There is one problem so far. When I run the game everything works fine except that if I press the down arrow key again once the game has started it will create another set of spaceship and asteroids.

To fix this issue I created a bool variable game start and set it to false. Changed the down arrow key case so it will only start one time because after the key is pressed the bool variable will be set to true meaning nothing happens if the key is pressed again. Here is the snippet of the down key pressed case:

```
case GLUT_KEY_DOWN:
    if (!gamestart) {
        // Create a spaceship and add it to the world
        mGameWorld->AddObject(CreateSpaceship());
        // Create some asteroids and add them to the world
        CreateAsteroids(10);
        //removes start screen gui
        RemoveStartScreenGUI();
        //Create the GUI
        CreateGUI();
        //set the game start bool to true
        gamestart = true;
        break;
    }

    break;
```

I added another break line after the if statement because if there is no break after the if statement it will close the game if the down key is pressed again.

Part II

A high score table.

In order to create a high score table, you must store the players name and their score in to a txt file.

Before I wrote any code I added the following libraries in the following file.

Asteroids.cpp:

```
#include <vector>
#include <algorithm>
#include <fstream>
```

Asteroids.h:

```
#include <map>
```

First, I created an empty txt file in the BIN directory.

Directory Path:

File:

Then I added Instruction for the player to type their name. so I added entername gui label in Asteroids.h

```
shared_ptr<GUILabel> mEnterName;
```

Then added the text and its properties in the CreateGUI method in Asteroids.cpp.

```
// Create a new GUILabel and wrap it up in a shared_ptr
mEnterName = shared_ptr<GUILabel>(new GUILabel("Please Type Your Name - Press 1 When Done"));
// Set the horizontal alignment of the label to GUI_HALIGN_CENTER
mEnterName->SetHorizontalAlignment(GUIComponent::GUI_HALIGN_CENTER);
// Set the vertical alignment of the label to GUI_VALIGN_MIDDLE
mEnterName->SetVerticalAlignment(GUIComponent::GUI_VALIGN_BOTTOM);
// Set the visibility of the label to false (hidden)
mEnterName->SetVisible(false);
// Add the GUILabel to the GUIContainer
shared_ptr<GUIComponent> play_again_component
    = static_pointer_cast<GUIComponent>(mEnterName);
mGameDisplay->GetContainer()->AddComponent(play_again_component, GLVector2f(0.5f, 0.1f));
```

Then set visibility to true when the game is over. Add the userline code in the SHOW_GAME_OVER if statements in OnTimer method in Asteroids.cpp.

```
if (value == SHOW_GAME_OVER)
{

    spaceshipAlive = false;
    mGameOverLabel->SetVisible(true);
    mScoreLabel->SetVisible(false);
    mLivesLabel->SetVisible(false);
    mEnterName->SetVisible(true);
```

After that I added a bool and string to store the player's name. the bool enterName is used to allow the player to type their name.

```
//if true will allow the player to enter name
bool enterName = false;
//holds players name
string name;
```

The bool is changed to true when the game ends so the player is able to type. Add the underline code. To the On Timer method in the Asteroids.cpp.

```
if (value == SHOW_GAME_OVER)
{

    spaceshipAlive = false;
    mGameOverLabel->SetVisible(true)
    mScoreLabel->SetVisible(false);
    mLivesLabel->SetVisible(false);
    mEnterName->SetVisible(true);
    enterName = true;

}
```

To make the enter name back to false I used the number 1 key. SO when the player is done typing their name they can press one to stop.

```
//stop player from typing their name
case '1':
   enterName = false;
```

Once the player can type. We must register all the letter the player types. We can do that by adding cases for all the letters in the alphabet and storing them in a string name shows above. This code is added to the OnKeyPressed methods of the Asteroids.cpp.

```
//all the cases below will add whichever leter is pressed to a string to hold players
name.
       case 'a':
              if (enterName) {
                     name += "a";
              }
              break;

       case 'b':
              if (enterName) {
                     name += "b";
              }
              break;

       case 'c':
              if (enterName) {
```

```
                name += "c";
        }
        break;

case 'd':
        if (enterName) {
                name += "d";
        }
        break;

case 'e':
        if (enterName) {
                name += "e";
        }
        break;

case 'f':
        if (enterName) {
                name += "f";
        }
        break;

case 'g':
        if (enterName) {
                name += "g";
        }
        break;

case 'h':
        if (enterName) {
                name += "h";
        }
        break;

case 'i':
        if (enterName) {
                name += "i";
        }
        break;

case 'j':
        if (enterName) {
                name += "j";
        }
        break;

case 'k':
        if (enterName) {
                name += "k";
        }
        break;

case 'l':
        if (enterName) {
                name += "l";
        }
        break;

case 'm':
        if (enterName) {
```

```
                name += "m";
            }
            break;

        case 'n':
            if (enterName) {
                name += "n";
            }
            break;

        case 'o':
            if (enterName) {
                name += "o";
            }
            break;

        case 'p':
            if (enterName) {
                name += "p";
            }
            break;

        case 'q':
            if (enterName) {
                name += "q";
            }
            break;

        case 'r':
            if (enterName) {
                name += "r";
            }
            break;

        case 's':
            if (enterName) {
                name += "s";
            }
            break;

        case 't':
            if (enterName) {
                name += "t";
            }
            break;

        case 'u':
            if (enterName) {
                name += "u";
            }
            break;

        case 'v':
            if (enterName) {
                name += "v";
            }
            break;
#
        case 'w':
            if (enterName) {
```

```
                name += "w";
            }
            break;

        case 'x':
            if (enterName) {
                name += "x";
            }
            break;

        case 'y':
            if (enterName) {
                name += "y";
            }
            break;

        case 'z':
            if (enterName) {
                name += "z";
            }
            break;
```

now the Player is able to type their name in the game. Lets write the name and score to the file.

I created to methods in the Asteroids.h file. WriteScore and ReadScore.

```
//writes the players name and their score to the file
void WriteScore();

//reads the players name and add it to the score.
void ReadScore();
```

In write score method we will be using fstream to write to the file. I added the name then space and then score to separate them. Here is the code for it.

```
//Writes the Current Score to the File to keep record of scores.
void Asteroids::WriteScore() {
    int score = mScoreKeeper.getmScore();
    cout << "WriteScore" << endl;
    try {
        //opens the file
        fstream file("RecordScore.txt", ios_base::out | ios_base::app);
        //write name score to the file
        file << name << " " << score << endl;
        //close the file
        file.close();
    }
    catch (int e)
    {
        cout << "Error number " << e << endl;
    }
}
```

In Read Score method I first added the Title for the Score table to gui. Then to read the name and the score I added two variable int score and string pName to store the score and their respective player name.

Then I used fstream to open the file and added the first word to pNAme and second word to score. Now that the score and name are stored. I added the score to the vector and then added score and name to hashmap to keep track of the player name and their score. Then I sort the nums vector from the high to low score and then create another vector to store top 5 from the nums vector.

Then using the GUI label I displayed the score.

To display the name I used the map to find the player name using the score.

Before this code I initialised the maps and the Gui labels in the Asteroids.h.

```
//High Score Table Labels
shared_ptr<GUILabel> mHighScoreTable;
shared_ptr<GUILabel> mPrintScore;
shared_ptr<GUILabel> mPrintName;
```

```
//keep the players name and their score
map<int,string > playerRecords;
```

Here is the code.

```
//Reads the Top 5 Scores and Display Once the game ends;
void Asteroids::ReadScore() {
	//create vector to store scores
	vector<int> nums;

	//cretae string to stores name
	string pName;

	//displays High score Table Title
	mHighScoreTable = shared_ptr<GUILabel>(new GUILabel("Top 5 Highest Scores"));
	mHighScoreTable->SetHorizontalAlignment(GUIComponent::GUI_HALIGN_CENTER);
	mHighScoreTable->SetVerticalAlignment(GUIComponent::GUI_VALIGN_MIDDLE);
	shared_ptr<GUIComponent> high_score_table_component =
static_pointer_cast<GUIComponent>(mHighScoreTable);
	mGameDisplay->GetContainer()->AddComponent(high_score_table_component,
GLVector2f(0.5f, 0.9f));


	try {
		fstream file;
		file.open("RecordScore.txt");

		int num;

		//first word in pName and second in num
		while (file >> pName >> num) {
			//adds the score to the nums vector
			nums.push_back(num);
```

```
                        //adds the name and the score
                        playerRecords[num] = pName;
                }
                //closes the file
                file.close();
                cout << "Success";

        }
        //catches exception
        catch(int e){
                cout << "Error No.: " << e << endl;
        }

        //sorts the vector high to low score
        sort(nums.begin(), nums.end(), std::greater<int>());

        //adds top 5 scores from the nums
        vector<int> top_five(nums.begin(), nums.begin() + min(5, (int)nums.size()));

        //y position of the gui
    float yPos = 0.8f;
        //displays top 5 score
        for (int i = 0; i < top_five.size(); ++i) {
                //displays score
                mPrintScore = shared_ptr<GUILabel>(new
GUILabel(to_string(top_five[i])));
                mPrintScore->SetHorizontalAlignment(GUIComponent::GUI_HALIGN_CENTER);
                shared_ptr<GUIComponent> print_score_component =
static_pointer_cast<GUIComponent>(mPrintScore);
                mGameDisplay->GetContainer()->AddComponent(print_score_component,
GLVector2f(0.7, yPos));

                //displays the player name by using map to find the name by score
                mPrintName = shared_ptr<GUILabel>(new
GUILabel(playerRecords[top_five[i]]));
                mPrintName->SetHorizontalAlignment(GUIComponent::GUI_HALIGN_CENTER);
                shared_ptr<GUIComponent> print_name_component =
static_pointer_cast<GUIComponent>(mPrintName);
                mGameDisplay->GetContainer()->AddComponent(print_name_component,
GLVector2f(0.3, yPos));

                //decreases the yPos by 0.1f
                yPos = yPos - 0.1f;
        }


}
```

At last, I added another Timer variable in Asteroids.h which will read and write score table.

```
const static uint SHOW_HIGH_SCORE_TABLE = 3;
```

Then I added another if statement in the OnTimer method in Asteroids.cpp to write and read. And also make the enter name gui disappear.

```
if (value == SHOW_HIGH_SCORE_TABLE) {
    WriteScore();
    ReadScore();
    mEnterName->SetVisible(false);
}
```

Then this is called when the player presses 1 after typing their name.

```
//stop player from typing their name
case '1':
    enterName = false;
    SetTimer(500, SHOW_HIGH_SCORE_TABLE);
    break;
```

That’s it the High score table should be working!

One last thing I added was displaying the name when typing. I did that by displaying string name where the name is stored after every key is pressed.

add this underlined code to every letter case in the asteroids.cpp.

```
case 'a':
    if (enterName) {

        name += "a";
        if (name.size() > 1) {
            mTShowName->SetVisible(false);
        }
        CreateNameGUI();
    }
    break;
```

Then add the code for create the pointers gui label to display the name.

```
shared_ptr<GUILabel> mTShowName;
shared_ptr<GUILabel> mName;
```

Then added the code to display mName where the player will be typing their name: this should be in the CreateGUI method in the Asteroids.cpp

```
// Create a new GUILabel and wrap it up in a shared_ptr
mName = shared_ptr<GUILabel>(new GUILabel("Name: "));
// Set the horizontal alignment of the label to GUI_HALIGN_CENTER
mName->SetHorizontalAlignment(GUIComponent::GUI_HALIGN_CENTER);
// Set the vertical alignment of the label to GUI_VALIGN_MIDDLE
mName->SetVerticalAlignment(GUIComponent::GUI_VALIGN_BOTTOM);
// Set the visibility of the label to false (hidden)
mName->SetVisible(false);
// Add the GUILabel to the GUIContainer
shared_ptr<GUIComponent> name_component
    = static_pointer_cast<GUIComponent>(mName);
mGameDisplay->GetContainer()->AddComponent(name_component, GLVector2f(0.3f, 0.5f));
```

And set it to true when the game ends . underlined code. In asteroids.cpp in the OnTimer method.

```
if (value == SHOW_GAME_OVER)
{

    spaceshipAlive = false;
    mGameOverLabel->SetVisible(true);
    mScoreLabel->SetVisible(false);
    mLivesLabel->SetVisible(false);
    mEnterName->SetVisible(true);
    mName->SetVisible(true);
    enterName = true;

}
```

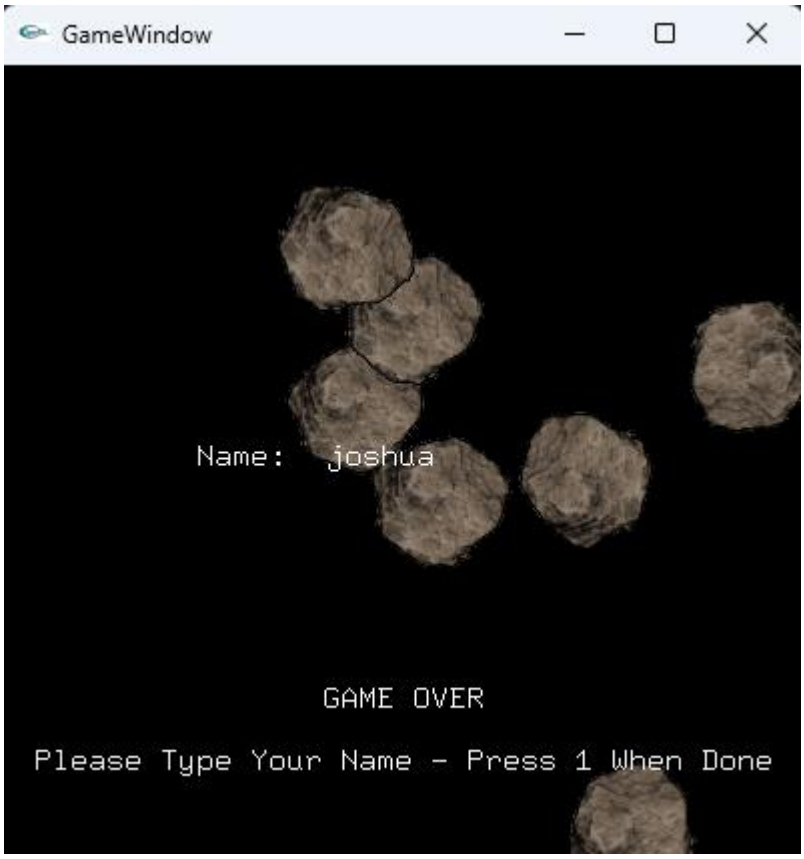
The implementation for the CreateNameGUI will be:

```
void Asteroids::CreateNameGUI() {
    // Create a new GUILabel and wrap it up in a shared_ptr
    mTShowName = make_shared<GUILabel>(name);
    // Set the vertical alignment of the label to GUI_VALIGN_BOTTOM
    mTShowName->SetVerticalAlignment(GUIComponent::GUI_VALIGN_BOTTOM);

    mTShowName->SetVisible(true);
    // Add the GUILabel to the GUIComponent
    shared_ptr<GUIComponent> T_Show_Name_component = static_pointer_cast<GUIComponent>(mTShowName);
    mGameDisplay->GetContainer()->AddComponent(T_Show_Name_component, GLVector2f(0.4f, 0.5f));
    count == 1;
}
```

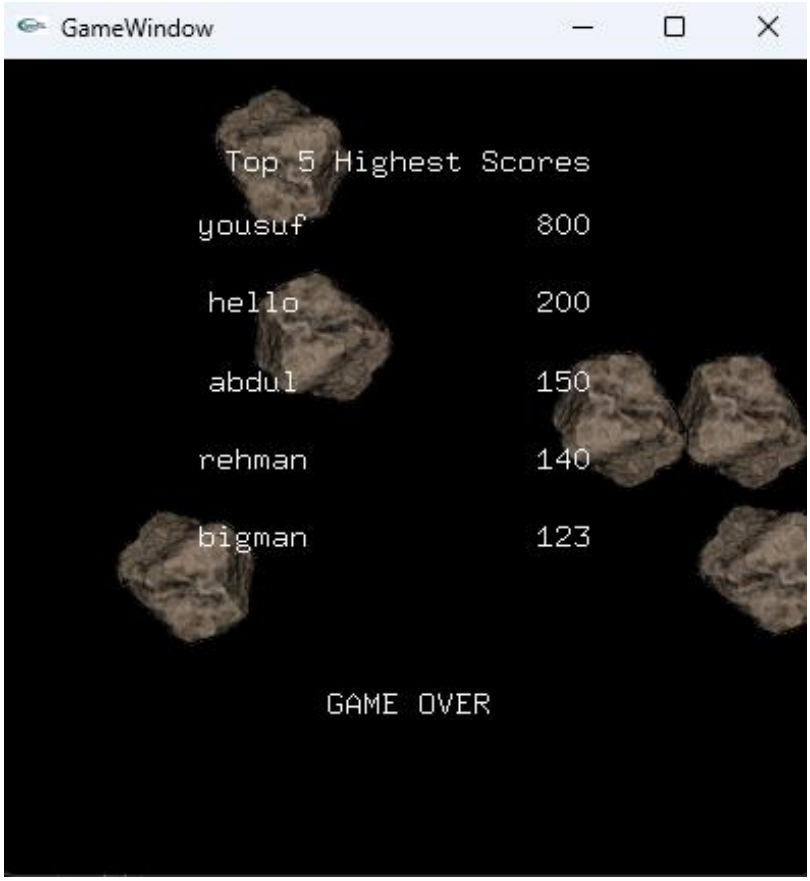
Also make sure to add the method in the Asteroids.h

```
void CreateNameGUI();
```

Here is what the game should look like when the game is over. You should be able to see the name you are typing.



Then after you type your name and press the key 1. The high score table to appear on the screen.


GameWindow
Top 5 Highest Scores
yousuf 800
hello 200
abdul 150
rehman 140
bigman 123
GAME OVER

Part III

A demo mode.

In demo mode I wanted to have another type of spaceship that will play the game in the background once the game ends. So, I decided to choose the enemy_fs from the filmstrip folder as my demo spaceship.

Then I added the enemy-fs from the filmstrip folder to the assets directory in the project and renamed it demospaceship_fs.

The I created the classes for the demospaceship which are similar to spaceship classes. I added 2 files “DemoSpaceship.cpp” and “DemoSpaceship.h” in the Src directory of the project.

Code for DemoSpaceship.h:

```
#ifndef __DEMOSPACESHIP_H__
#define __DEMOSPACESHIP_H__

#include "GameUtil.h"
#include "GameObject.h"
#include "Shape.h"

class DemoSpaceship : public GameObject
{
public:
	DemoSpaceship();
	DemoSpaceship(GLVector3f p, GLVector3f v, GLVector3f a, GLfloat h, GLfloat
r);
	DemoSpaceship(const DemoSpaceship& s);
	virtual ~DemoSpaceship(void);

	virtual void Update(int t);
	virtual void Render(void);

	virtual void Thrust(float t);
	virtual void Rotate(float r);
	virtual void Shoot(void);

	void SetSpaceshipShape(shared_ptr<Shape> demo_spaceship_shape) {
mDSpaceshipShape = demo_spaceship_shape; }
	void SetThrusterShape(shared_ptr<Shape> thruster_shape) { mThrusterShape =
thruster_shape; }
	void SetBulletShape(shared_ptr<Shape> bullet_shape) { mBulletShape =
bullet_shape; }

	bool CollisionTest(shared_ptr<GameObject> o);
	void OnCollision(const GameObjectList& objects);

private:
	float mThrust;

	shared_ptr<Shape> mDSpaceshipShape;
```

```
	shared_ptr<Shape> mThrusterShape;
	shared_ptr<Shape> mBulletShape;
};

#endif#pragma once
```

Code for the DemoSpaceship.cpp:

```
#include "GameUtil.h"
#include "GameWorld.h"
#include "Bullet.h"
#include "DemoSpaceship.h"
#include "BoundingSphere.h"

using namespace std;

// PUBLIC INSTANCE CONSTRUCTORS ////////////////////////////////////////////

/**  Default constructor. */
DemoSpaceship::DemoSpaceship()
	: GameObject("DemoSpaceship"), mThrust(0)
{
}

/** Construct a spaceship with given position, velocity, acceleration, angle, and
rotation. */
DemoSpaceship::DemoSpaceship(GLVector3f p, GLVector3f v, GLVector3f a, GLfloat h,
GLfloat r)
	: GameObject("DemoSpaceship", p, v, a, h, r), mThrust(0)
{
}

/** Copy constructor. */
DemoSpaceship::DemoSpaceship(const DemoSpaceship& s)
	: GameObject(s), mThrust(0)
{
}

/** Destructor. */
DemoSpaceship::~DemoSpaceship(void)
{
}

// PUBLIC INSTANCE METHODS /////////////////////////////////////////////////

/** Update this spaceship. */
void DemoSpaceship::Update(int t)
{
	// Call parent update function
	GameObject::Update(t);
}

/** Render this spaceship. */
void DemoSpaceship::Render(void)
```

```
{
	if (mDSpaceshipShape.get() != NULL) mDSpaceshipShape->Render();

	// If ship is thrusting
	if ((mThrust > 0) && (mThrusterShape.get() != NULL)) {
		mThrusterShape->Render();
	}

	GameObject::Render();
}

/** Fire the rockets. */
void DemoSpaceship::Thrust(float t)
{
	mThrust = t;
	// Increase acceleration in the direction of ship
	mAcceleration.x = mThrust * cos(DEG2RAD * mAngle);
	mAcceleration.y = mThrust * sin(DEG2RAD * mAngle);
}

/** Set the rotation. */
void DemoSpaceship::Rotate(float r)
{
	mRotation = r;
}

/** Shoot a bullet. */
void DemoSpaceship::Shoot(void)
{
	// Check the world exists
	if (!mWorld) return;
	// Construct a unit length vector in the direction the spaceship is headed
	GLVector3f spaceship_heading(cos(DEG2RAD * mAngle), sin(DEG2RAD * mAngle),
0);
	spaceship_heading.normalize();
	// Calculate the point at the node of the spaceship from position and
heading
	GLVector3f bullet_position = mPosition + (spaceship_heading * 4);
	// Calculate how fast the bullet should travel
	float bullet_speed = 30;
	// Construct a vector for the bullet's velocity
	GLVector3f bullet_velocity = mVelocity + spaceship_heading * bullet_speed;
	// Construct a new bullet
	shared_ptr<GameObject> bullet
	(new Bullet(bullet_position, bullet_velocity, mAcceleration, mAngle, 0,
2000));
	bullet->SetBoundingShape(make_shared<BoundingSphere>(bullet->GetThisPtr(),
2.0f));
	bullet->SetShape(mBulletShape);
	// Add the new bullet to the game world
	mWorld->AddObject(bullet);

}

bool DemoSpaceship::CollisionTest(shared_ptr<GameObject> o)
{
	if (o->GetType() != GameObjectType("Asteroid")) return false;
	if (mBoundingShape.get() == NULL) return false;
	if (o->GetBoundingShape().get() == NULL) return false;
	return mBoundingShape->CollisionTest(o->GetBoundingShape());
}
```

```
void DemoSpaceship::OnCollision(const GameObjectList& objects)
{
	mWorld->FlagForRemoval(GetThisPtr());
}
```

After adding this code which is basically the same code as the spaceship class because it will perform the same actions but, in this instance, it will act like an ai and perform action automatically.

To make it act like an ai we will have to compare the positions of the demospaceship and asteroids on the map. If an asteroid is in range of x the demospaceship will shoot.

To get the asteroids positions I created a vector to hold shared_ptr<GameObject> in Asteroids.h

```
vector<shared_ptr<GameObject>> AsteroidPosTraker;
```

Then I cleared the vector at the start of the createasteroids method and pushback the asteroid pointer to the vector. Createdasteroids is in the asteroids.cpp.

```
void Asteroids::CreateAsteroids(const uint num_asteroids)
{
    //clears the vector which holds the asteroids object pointer so the
    //old ones are replaced by the new ones everytime asteroids are created
    AsteroidPosTraker.clear();
    mAsteroidCount = num_asteroids;
    for (uint i = 0; i < num_asteroids; i++)
    {
        Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("asteroid1");
        shared_ptr<Sprite> asteroid_sprite
            = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
        asteroid_sprite->SetLoopAnimation(true);
        shared_ptr<GameObject> asteroid = make_shared<Asteroid>();
        asteroid->SetBoundingShape(make_shared<BoundingSphere>(asteroid->GetThisPtr(), 10.0f));
        asteroid->SetSprite(asteroid_sprite);
        asteroid->SetScale(0.2f);
        mGameWorld->AddObject(asteroid);

    // adds the pointer to the asterid object to the vector to get the object pos later
        AsteroidPosTraker.push_back(asteroid);
    }
}
```

Then I created a method for creating the demospaceship.

Code for Asteorids.h:

```
shared_ptr<DemoSpaceship> mDemoSpaceship;
```

```
shared_ptr<GameObject> CreateDemoSpaceship();
```

Code for Asteorids.cpp;

```
shared_ptr<GameObject> Asteroids::CreateDemoSpaceship()
{
    // Create a raw pointer to a spaceship that can be converted to
    // shared_ptrs of different types because GameWorld implements IRefCount
    Animation* anim_ptr = AnimationManager::GetInstance().GetAnimationByName("demospaceship");
    shared_ptr<Sprite> demospaceship_sprite = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
    demospaceship_sprite->SetLoopAnimation(true);
    mDemoSpaceship = make_shared<DemoSpaceship>();
    mDemoSpaceship->SetBoundingShape(make_shared<BoundingSphere>(mDemoSpaceship->GetThisPtr(), 4.0f));
    shared_ptr<Shape> bullet_shape = make_shared<Shape>("bullet.shape");
    mDemoSpaceship->SetBulletShape(bullet_shape);

    mDemoSpaceship->SetSprite(demospaceship_sprite);
    mDemoSpaceship->SetScale(0.1f);
    // Reset spaceship back to centre of the world
    mDemoSpaceship->Reset();
    // Return the spaceship so it can be added to the world
    return mDemoSpaceship;

}
```

Then I added the createdemospaceship to the SHOW_GAME_OVER if statement in the OnTimer method of the asteroids.cpp:

```
if (value == SHOW_GAME_OVER)
{

    gameStatus = false;
    mGameOverLabel->SetVisible(true);
    mScoreLabel->SetVisible(false);
    mLivesLabel->SetVisible(false);
    mEnterName->SetVisible(true);
    mName->SetVisible(true);
    enterName = true;

    mGameWorld->AddObject(CreateDemoSpaceship());
```

Then I created to OnTimer if statements to reset the demospaceship when destroyed and another if statement for starting the demo mode and implement the ai behaviour.

First I added two uint to store the values for the ontimer method:

```
const static uint START_DEMO_MODE = 4;

const static uint RESET_DEMO_SPACESHIP = 5;
```

Code for the Start_Demo_Mode:

```
if (value == START_DEMO_MODE) {
    mDemoSpaceship->Thrust(rand()%15 +(2));
    mDemoSpaceship->Rotate(rand() % 110 + (45));
    // Get the position of the demo spaceship
    GLfloat demospaceshipPos[3];
    demospaceshipPos[0] = mDemoSpaceship->GetPosition().x;
    demospaceshipPos[1] = mDemoSpaceship->GetPosition().y;
    demospaceshipPos[2] = mDemoSpaceship->GetPosition().z;

    GLfloat shootingDistance = 15.0f;

    for (int i = 0; i < AsteroidPosTraker.size(); ++i) {
        // Get the position of the current asteroid
        GLfloat asteroidPos[3];
        asteroidPos[0] = AsteroidPosTraker[i]->GetPosition().x;
        asteroidPos[1] = AsteroidPosTraker[i]->GetPosition().y;
        asteroidPos[2] = AsteroidPosTraker[i]->GetPosition().z;

        // Calculate the distance between the spaceship and the asteroid
        GLfloat dx = demospaceshipPos[0] - asteroidPos[0];
        GLfloat dy = demospaceshipPos[1] - asteroidPos[1];
        GLfloat dz = demospaceshipPos[2] - asteroidPos[2];
        GLfloat distance = sqrt(dx * dx + dy * dy + dz * dz);

        if (distance <= shootingDistance) {
            mDemoSpaceship->Shoot();
        }
    }

    SetTimer(800, START_DEMO_MODE);
}
```

Explanation: First I set the thrust and the rotation of the demospaceship and the store the position of the demo spacespace in the GLfloat array where I am store the x, y and z pos of the demo spaceship. Then I set the distance in which the demospaceship will shoot if the asteoird enters that parameter.

Then I loop through the vector which contains all the pointers to the asteoid gameobject. Then create glfloat array and store the x,y and z. then calculated the distance between the ship and the asteroid by subtracting the x,y and z pos of both object and store it in the dx,dy and dz.

Then I perform a sqrt(dx*dx + dy*dy + dz*dz) to calculate the distance.

Then finally checking if the distance is less than or equal to the range the demospaceship will shoot.

Then I call this if statement on a recursive call every 8 millisecond.

Code for Reset_Demo_Spaceship:

```
if (value == RESET_DEMO_SPACESHIP) {
    mDemoSpaceship->Reset();
    mGameWorld->AddObject(mDemoSpaceship);
}
```

Explanation: This just resets the demo spaceship and adds to the game world.

One last thing I added was to call the RESET_DEMO_SPACESHIP once its destroyed. I did this by added the following code in the OnObjectRemoved method of the asteroids.cpp:

```
if (object->GetType() == GameObjectType("DemoSpaceship")) {
    SetTimer(500, RESET_DEMO_SPACESHIP);
}
```

Additional Features

Back Thrust.

Code for OnSpecialKeyPressed method in asteroids.cpp:

```
case GLUT_KEY_DOWN:

    if (gameStatus) {
        mSpaceship->Thrust(-10);
    }

    break;
```

Code for OnSpecialKeyReleased method of the asteroids.cpp:

```
case GLUT_KEY_DOWN:

    if (gameStatus) {
        mSpaceship->Thrust(0);
    }

    break;
```