



El Space Aventura!

Name :: Masana Ikeshima
Matric :: Number :: 200601037
Course :: Games Software Development Year 4
Module :: Games Programming 3
Group :: 2

I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award.

Signature _____

Date _____

Contents

Requirements	4
Design	5
Menu Screen	6
Story Screen	7
Controls Screen	8
Default Camera / HUD.....	9
Secondary Camera	10
Player Movement	11
Enemy Movement	12
Ambient Model Movement.....	13
Game Creation	14
Setup	15
Game1 Constructor.....	19
Initialize	20
Setup Camera.....	21
Load Content.....	23
Setup Effect Transformation Defaults	24
Draw Model.....	25
Setup Models	26
Menu	29
Game Loop	33
Player Move	38
Write Text.....	40
Draw Method	41
Unload Content.....	46
Get Input	47
Class :: Ambient Model	49
Class :: Cameras	51
Class :: Player	53
Class :: Enemy.....	55

Class :: Laser	59
Class :: Shield.....	62
Development Issues.....	64
References.....	65
Code	67
File :: Game1.cs	67
File:: ambientModel.cs.....	95
File:: Camera.cs	96
File:: Enemy.cs.....	98
File:: laserClass.cs	102
File:: Player.cs.....	103
File:: Shield.cs.....	105

Requirements Specification

Requirements

The client's specification states that the following key aspects must be included within the project for it to be classified a success:

- A minimum of six models of which one must be controlled by the user
- Each model must be skinned using appropriate textures
- Sound, which is applicable to scene
- A minimum of two cameras of which one must be third/first person
- Collision detection must be included
- Keyboard input controlling the sound and switch of the cameras

Solution

In order to meet all the above criteria within the specified timeframe, it was decided that the most efficient way would be to base the project on an already made game. This would result in a shorter design phase, maximising the implementation process to ensure the end product was as high as possible.

The developer decided to base the game on the classic retro game – Space Invaders. Space Invaders is a simple top-down shooting game, where the player has to avoid enemy alien fire by counter firing or using the shields for cover.

The game was fairly basic in terms of design and visual components due to the restrictions of the hardware capability. Visually, the only items on screen were the aliens, player, shield and the score. Similarly, in terms of audio the sound effects were very basic and featured a very simple background song.

With recent games such as Bionic Commando: Rearmed and Super Mario Bros. Wii, revamping the classic 2D game play with three dimensional backdrops and game play elements, the developer has decided to target this project using the retro game of space invaders with the latest technology.

The developer has decided that a simple recreation of Space Invaders would not suffice, thus it is intended that the game will include new features and elements to provide a more intense and exhilarating game play than before. For this reason, new vibrant-next generation sounds will be included, alongside new visual components – 3D models and two camera perspectives to highlight these new features.

Design

Overview

It was important to design various storyboard to allow the developer to have a focused and clear point of reference when developing the game. The following list states all the storyboard that were created:

- Menu screen
- Instructions screen
- Controls screen
- Game over / Win screen
- Default camera
- Secondary (first person) camera
- Heads Up Display design
- Player movement
- Enemy movement
- Ambient model movement

Key

The following key should be used when viewing the in-game storyboards:

Item	Description
Blue Square	Enemy position
Red Rectangle	Shield position
Green Rectangle	Player position
Brown Rectangle	Ambient model [satellite] position
Grey Circle	Ambient model [moon] position
Yellow Rectangle	Ambient model [blade] position

Design :: Menu Screen



Figure 1.1 Menu Screen

Figure 1.1 clearly shows that the menu screen the user is first presented with upon starting the program has three options:

- Start game (Figure 2.1)
 - Begins the game loop, allowing the player to experience the full game.
- Instructions (Figure 1.2, Figure 1.3)
 - The instructions option serves to display two primary sets of information.
 - The first is the story, why the player is fighting alien forces and why s/he is in space to begin with.

It was decided that a story would be included in the game, despite not being included in the original Space Invaders, as the designer firmly believes that a game with a story has a greater sense of purpose and a heightened experience.
 - From the story the player is then shown the controls of the game. This will be in the form of an annotated Xbox 360 controller, stating each buttons purpose and it's keyboard counterpart.
- Exit
 - This will exit the game.

It is intended that depending on the selection, the option highlighted will be in red text to stand out from the rest.

Design :: Story Screen

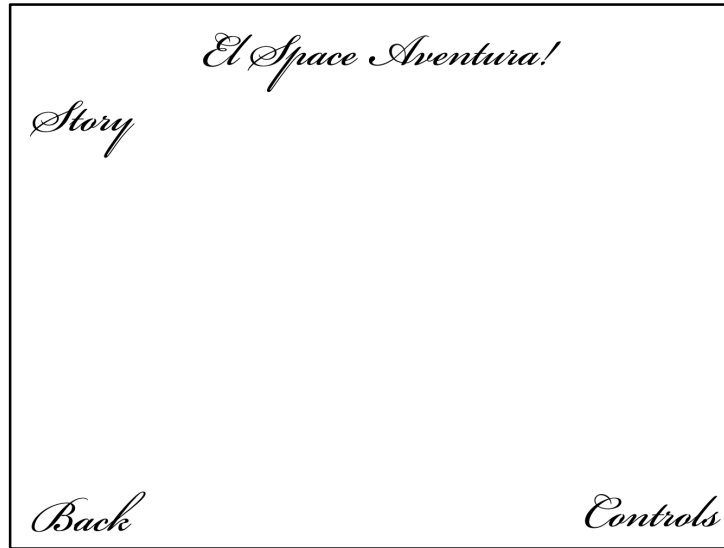


Figure 1.2 Story Screen

The story screen contains two options “Back” and “Controls” this is to allow the player to either move back to the main menu screen (Figure 1.1) or to move to the controls screen (Figure 1.3). During the design phase the story had not been finalised hence there is no text, however it would be located below the heading “Story”.

Design :: Controls Screen

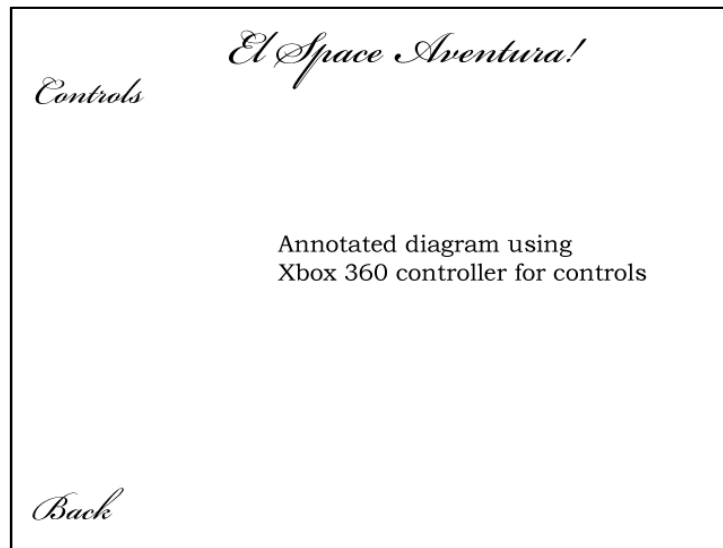


Figure 1.3 Controls Screen

The controls screen's purpose is to tell the player how to play the game, using either the keyboard or the Xbox 360 controller. The designer believed that the most effective way to notify the player would be using an annotated diagram of the Xbox 360 controller showing, what each button is mapped to. Below each function, it is intended to state the keyboard alternative button, this way the player can easily switch between the two.

There is only one option in this screen "Back" to take the user back to instructions screen (Figure 1.2).

Design :: Default Camera

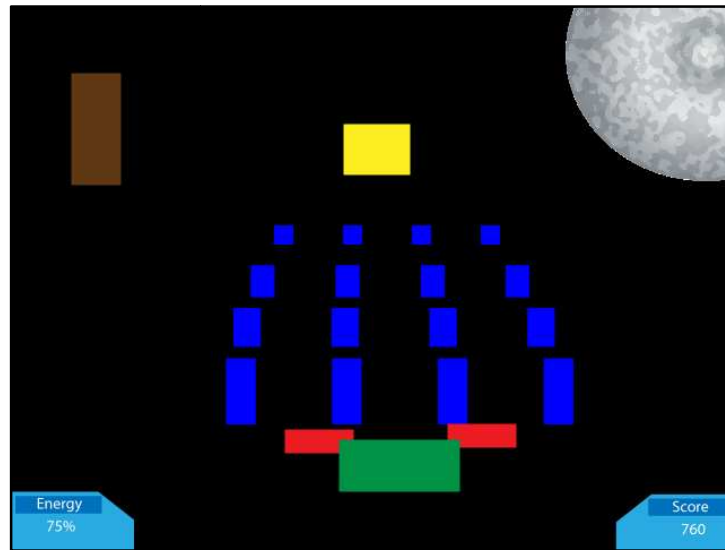


Figure 2.1 Default Camera

Figure 2.1 shows the view of the default camera in the game. The camera will be setup so that it is above and slightly behind the player spaceship.

The reducing size of the blue rectangles is to represent the depth of the screen and how the enemies will be coming towards the player from the distance. Likewise, the two red rectangles represent the shields that the player has to protect himself with.

To create a sense of realism in the game, three ambient models were included – a blade, a moon and a satellite that will orbit it. By including these three models the game space feels less empty and heightens the player's sense of flow.

The models intended to be selected will be a mix between cartoony and realistic graphics. The original space invaders had very little colour and the designs were simplistic, this should be reflected in the models.

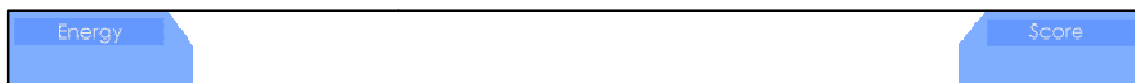


Figure 2.2 Heads up display

As evident in Figure 2.2 the heads up display (HUD) will notify the player of two key statistics: the energy they have left and their current score.

Under the energy label they will be shown a percentage of their life remaining, once it reaches zero, its game over. Similarly, the score label will be show their score they have obtained through destroying the enemy spaceships.

Design :: Secondary Camera

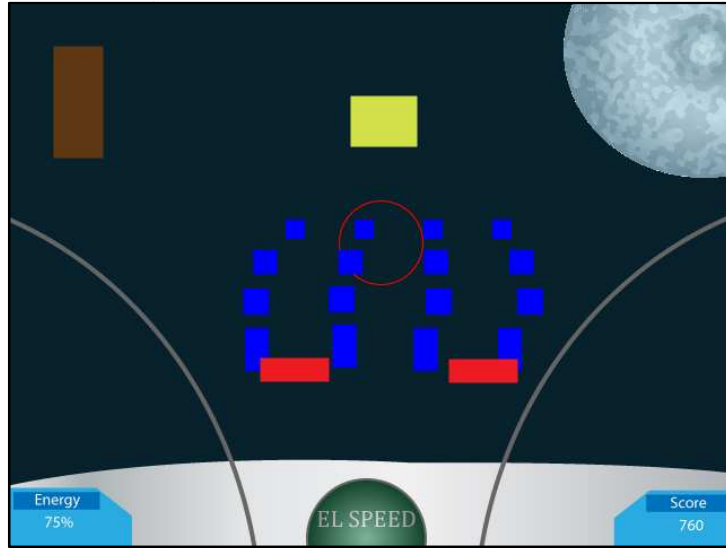


Figure 3.1 Secondary Camera

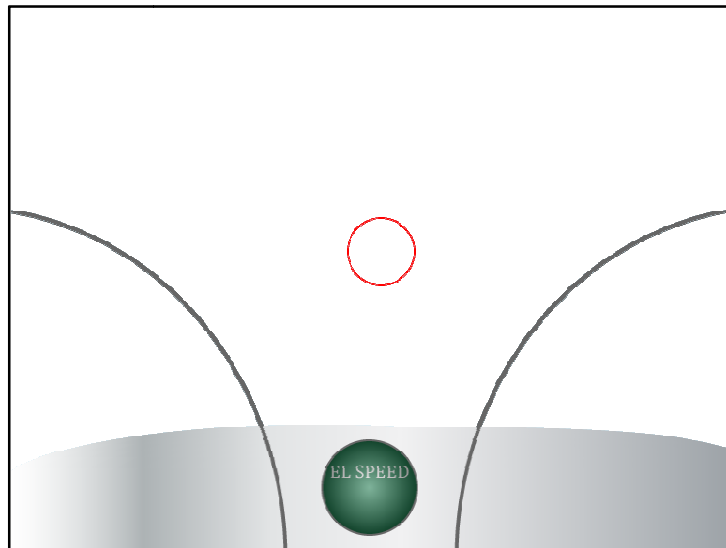


Figure 3.2 First Person Overlay

The secondary camera will be a first person perspective of the game. This will be executed by obtaining the player's position and setting it to the camera's position. To make the player feel as though they are inside the ship, an overlay (Figure 3.2) will be added to display the target.

Design :: Player Movement

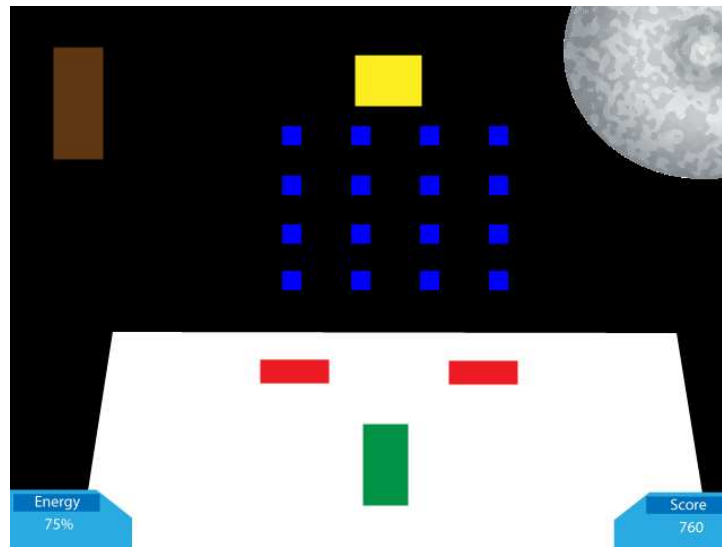


Figure 4.1 Player Movement

Like the original Space Invaders, in “El Space Aventura!”, players have the capability to move side by side to evade enemy fire. However, also included is the ability to move both forward and backwards. The reason for the inclusion is to provide the user more freedom, and to provide intense action towards the end of the level. Technically it will also be used to provide the user to view the high quality three dimensional models used in the game.

From the default camera (Figure 2.1), view down the middle is a parallelogram, which is why the white boundary is too a parallelogram.

As well as moving within the parallelogram, the player is capable of firing lasers. They will move in a constant speed in a straight line, if they hit an enemy they will disappear or if they reach an end point that is behind the last enemy in the array it will disappear.

Design :: Enemy Movement

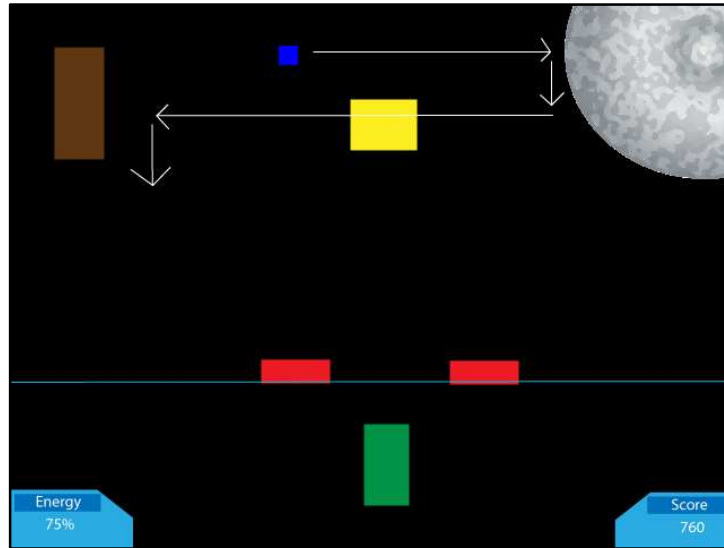


Figure 5.1 Enemy Movement

Figure 5.1 shows how the enemies in “El Space Aventura!” will move. The default position of the enemies will be in the middle of screen , following commencement of the game, they will move to the maximum position on the right and then move forwards. Once they have moved forwards they will then move to the opposite side, and again move forwards until they have reached the sky blue line, at which point it will be game over.

The skyblue line will not be displayed on the screen and merely for visual purposes only in the storyboard. The reason for this was to maintain tension in the game, as the player will not know how long they have left.

Like the lasers fired by the player, if they hit either the shield or the player their respective health statistic will be reduced. If nothing is hit the laser passes beyond the camera then it will disappear.

Design :: Ambient Model Movement

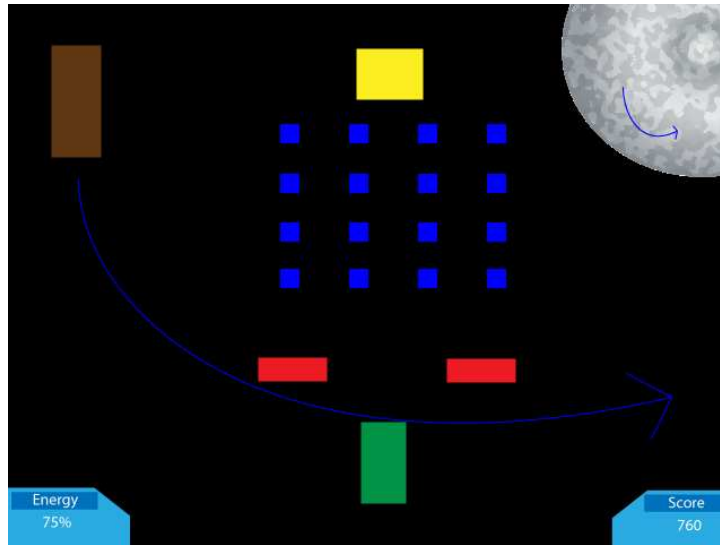


Figure 6.1 Ambient Model Movement

The satellite will be on a path to orbit the moon denoted by the blue arrow. It will do so by spinning around and below the game space, this is to further enhance the experience. Similarly the moon will spin, whilst maintain the same position. The blade, however will remain stationary.

Game Creation

The following section's purpose is to describe the code that is used to create the game. Each function and class has been split up into sections to help the reader understand the code.

The *Overview* section is to provide the reader with an insight has to the overall purpose of the function or class.

The *Variable Description* section breaks down each variable used and what it's purpose is within the function or the class.

The *Functional Description* breaks down each function into manageable sections, followed by an in-depth explanation to how the achieved result is obtained.

It has been intended to structure the following section in order of run-time.

Game Variables :: Setup

Overview

The following variables are used throughout the game; below each variable is an explanation of its purpose and description.

Sound Related Variable Descriptions

themeMusicInstance

```
private SoundEffectInstance themeMusicInstance;
```

An instance of the background music.

themeMusic

```
private SoundEffect themeMusic;
```

Stores the background music.

playerFire

```
private SoundEffect playerFire;
```

Stores the sound played when the player fires.

enemyFire

```
private SoundEffect enemyFire;
```

Stores the sound played when the enemy fires

explosion

```
private SoundEffect explosion;
```

Stores the sound played when either the player or an enemy dies.

shieldDestroy

```
private SoundEffect shieldDestroy;
```

This sound will be used when the shield has been destroyed.

soundsOn

```
private bool soundIsOn = true;
```

This boolean is used to check if the sound is on or off, if it's off then all sound effects won't be played until it is turned to true.

Player Related Variable Descriptions

Player instance

```
private Player player = new Player();
```

This line of code, creates an instance of the Class Player using the variable name, player.

health

```
public int health;
```

Health is used to determine how much health is left on the player.

score

```
public int score;
```

Score is used to count up how many points the player has managed to rack up over the game.

scoreWin

```
private int scoreWin;
```

This variable is used when the player wins the game, by destroying all the enemies. If all the enemies are destroyed, then their score is multiplied by their health they have left and is assigned to this variable.

Enemy Related Variable Descriptions

Enemy array

```
private Enemy[] enemy = new Enemy[20];
```

An array of 20 instances of class Enemy are created using this code.

setup

```
private bool setup = false;
```

The variable setup is used to determine whether or not setup has been executed.

xChange

```
private Vector3 xChange;
```

The variable xChange is used in the setting up of the enemy array. Each time a new enemy is added, this value is reduced, to create a row of enemies.

zChange

```
private float zChange;
```

zChange is used in conjunction with xChange to create the rows of enemies.

Camera Related Variable Descriptions

Create camera instances

```
private Camera camera1 = new Camera();  
private Camera camera2 = new Camera();
```

These two lines of code, create two cameras (camera1 and camera2) both of the class Camera.

cameraActive

```
private int cameraActive = 1;
```

This variable is used to determine, which of the two cameras is active. By default camera 1 is always active.

Other Model Related Variable Descriptions

shieldArray

```
private Shield[] shieldArray = new Shield[2];
```

This creates an array of two instances of the class Shield.

Create ambient models

```
private ambientModel sat = new ambientModel();  
private ambientModel moon = new ambientModel();  
private ambientModel blade = new ambientModel();
```

This creates the ambient models (moon, satellite and blade) all of the class ambientModel.

enemyLaserArray

```
private Laser[] enemyLaserArray = new Laser[2];
```

This creates an array of two instances of the class Laser, used for the enemies.

laserArray

```
private Laser[] LaserArray = new Laser[10];
```

This creates ten instances of the class Laser, used for the player.

Miscellaneous Variable Descriptions

hud

```
Texture2D hud;
```

Hud is used to draw the 2D texture over the camera's to show the player's score and remaining health.

lastInput

```
private int lastInput;
```

The variable lastInput is used to prevent the same input being recognized as being pressed rapidly as it waits for the next update to allow for new input.

retrying

```
private bool retrying = false;
```

The variable `retrying` is used when the game is being retried and requires all object instances to be set to the original setting.

playOnce

```
private bool playOnce = false;
```

`playOnce` is a variable used for the background music. If the game has just begun, `playOnce` will be `false` to start the song, however once they are in the game, the functions – `pause` and `resume` are only relevant.

Game Function :: Game1 Constructor

Overview

The game1 constructor's purpose is called by the file programs when creating a new instance of the game.

Functional Description

The constructor

```
public Game1(){  
    graphics = new GraphicsDeviceManager(this);  
    Content.RootDirectory = "Content"; }
```

The variable graphics is defined as a new instance of GraphicsDeviceManager and the location of the content to be loaded is specified.

Game Function:: Initialize

Overview

The purpose of the initialize function is to set the game up using the default variables and settings. It will only be called when the application has begun.

Functional Description

Initialize cameras and models

```
protected override void Initialize(){
    IsMouseVisible = true;
    setupCamera();
    setupModels();
    base.Initialize();}
```

This function sets the mouse visible whilst the game is being played. As a default, XNA sets this to false, meaning the player cannot view the mouse, which can be an inconvenience for the user.

After setting up the mouse, two functions, setupCamera and setupModels are called, to instantiate both the camera and the models used in the game.

Finally the line, base.Initialize(); enumerates through all the game components to be added.

Game Function :: Setup Camera

Overview

This function is called to setup the cameras that will be used in the game. Each camera used below, are an instance of the class, Camera.

Variable Description

AspectRatio

This variable will be used to pass through the value of the aspect ratio. It is calculated by dividing the width by the height of the screen.

CameraPosition and cameraPosition2

cameraPosition and cameraPosition2 , holds the position of Camera 1 and Camera2, respectively.

Target1 and target2

target1 and target2 hold the look at target for their respective cameras.

Functional Description

Setting up aspect ratio

```
private void setupCamera(){
    float aspectRatio =
        (float)graphics.GraphicsDevice.Viewport.Width /
        (float)graphics.GraphicsDevice.Viewport.Height;
    camera1.setProjectionMatrix(aspectRatio);
    camera2.setProjectionMatrix(aspectRatio);
}
```

Once the function has been called, the variable aspect ratio will be assigned the value of the viewport's width divided by the height. This will then be assigned for both cameras.

Setting the cameras position and target

```
Vector3 cameraPosition = new Vector3(0.0f, 5.0f, 10.0f);
Vector3 cameraPosition2 = new Vector3(0,0,0);
Vector3 target1 = new Vector3(0, 0, 0);
Vector3 target2 = new Vector3(0, 0, -5);
camera1.setViewMatrix(cameraPosition,target1);
camera2.setViewMatrix(cameraPosition2, target2);}
```

Each camera is assigned their respective position and target to look at by passing through the vector 3 values.

Game Function :: Load Content

Overview

The purpose of this function is to load the content of the game and assign them to the relative variable.

Functional Description

Load 2D textures

```
protected override void LoadContent(){
    spriteBatch = new SpriteBatch(GraphicsDevice);
    bgMenu = Content.Load<Texture2D>("Menu\\background");
    startGame = Content.Load<Texture2D>("Menu\\startGameAI");
    exit = Content.Load<Texture2D>("Menu\\Exit");
    spacer = Content.Load<Texture2D>("Menu\\spacer");
    startGameOVER = Content.Load<Texture2D>("Menu\\startGameAI2");
    instructionsOVER =
    Content.Load<Texture2D>("Menu\\Instructions2");
    exitOVER = Content.Load<Texture2D>("Menu\\Exit2");
    instructionsBG =
    Content.Load<Texture2D>("Menu\\instructions_BG");
    gameOver = Content.Load<Texture2D>("Menu\\gameOverScreen");
    win = Content.Load<Texture2D>("Menu\\winScreen");
    firstPerson = Content.Load<Texture2D>("Menu\\firstPerson");
    instructions = Content.Load<Texture2D>("Menu\\Instructions");
    instructionsBG2=
    Content.Load<Texture2D>("Menu\\instructions_BG2");
    instructionsBG3 =
    Content.Load<Texture2D>("Menu\\instructions_BG3");
    hud = Content.Load<Texture2D>("HUD\\hud");
```

All the two dimensional textures were created using Adobe Illustrator, and are now imported through the game and assigned to the relative variable. Everything except hud and firstperson is used in the menu, with hud used over the screen to display the score and health, whilst first person is used to portray a view from inside the ship.

Load 3D models and setup transformation

```
player.loadModel(Content.Load<Model>("Models\\jet"));
player.modelTrans(SetupEffectTransformDefaults(player.getModel()));

for (int k = 0; k < 2; k++){
    shieldArray[k].loadModel(Content.Load<Model>("Models\\shield_ne
w"));
    shieldArray[k].modelTrans(SetupEffectTransformDefaults(shieldAr
ray[k].getModel()));}
for (int i = 0; i < 20; i++){
    enemy[i].loadModel(Content.Load<Model>("Models\\smallViper"));
    enemy[i].modelTrans(SetupEffectTransformDefaults(enemy[i].getMo
del()));}
for (int i = 0; i < 10; i++){
    LaserArray[i].loadModel(Content.Load<Model>("Models\\plaser"));
    LaserArray[i].modelTrans(SetupEffectTransformDefaults(LaserArra
y[i].getModel()));}
```

```

for (int i = 0; i < 2; i++){
    enemyLaserArray[i].loadModel(Content.Load<Model>("Models\\Enemy
    laser"));
    enemyLaserArray[i].modelTrans(SetupEffectTransformDefaults(enem
    yLaserArray[i].getModel()));}
sat.loadModel(Content.Load<Model>("Models\\cassini"));
sat.modelTrans(SetupEffectTransformDefaults(sat.getModel()));
moon.loadModel(Content.Load<Model>("Models\\Moon"));
moon.modelTrans(SetupEffectTransformDefaults(moon.getModel()));
blade.loadModel(Content.Load<Model>("Models\\prop"));
blade.modelTrans(SetupEffectTransformDefaults(blade.getModel()));

```

Similar to the code for 2D textures, this section of code assigns a variable to load its respective model. However, unlike the 2D texture code, it also asks for the model to then be transformed using the `setupEffectTransformDefaults` method.

Load audio and font

```

fontToUse = Content.Load<SpriteFont>("Fonts\\Army");
themeMusic = Content.Load<SoundEffect>("Audio\\Waves\\powerplant");
playerFire = Content.Load<SoundEffect>("Audio\\Waves\\playerfire");
enemyFire = Content.Load<SoundEffect>("Audio\\Waves\\enemyfire");
explosion = Content.Load<SoundEffect>("Audio\\Waves\\explosion");
shieldDestroy =
Content.Load<SoundEffect>("Audio\\Waves\\shield_destroy");}

```

The purpose of this code is assign and load the font that will be used throughout the game, and to load the audio that will be used.

Game Function :: Setup Effect Transformation Defaults

Overview

The purpose of this function is to obtain the data that makes up the model that has been passed in. Initially, the default camera will be camera 1 (third person), thus the effects of the third person will be used.

Functional Description

Obtain model bones and apply effects

```
private Matrix[] SetupEffectTransformDefaults(Model myModel){
    Matrix[] absoluteTransforms = new Matrix[myModel.Bones.Count];
    myModel.CopyAbsoluteBoneTransformsTo(absoluteTransforms);
    foreach (ModelMesh mesh in myModel.Meshes){
        foreach (BasicEffect effect in mesh.Effects){
            effect.EnableDefaultLighting();
            effect.View = camera1.getViewMatrix();
            effect.Projection = camera1.getProjectionMatrix();}
    return absoluteTransforms;}
```

This function takes in a 3D model and outputs a matrix of the transformation using the third person camera's view matrix and projection matrix, whilst applying a basic default lighting.

Game Function :: Draw Model

Overview

The function of the draw model method is to draw the model by applying the effects relative to each camera that is currently active.

Functional Description

Draw model method

```
public void DrawModel(Model model, Matrix modelTransform, Matrix[] absoluteBoneTransforms){
    foreach (ModelMesh mesh in model.Meshes){
        foreach (BasicEffect effect in mesh.Effects){
            effect.World =
                absoluteBoneTransforms[mesh.ParentBone.Index] *
                modelTransform;
            if (cameraActive == 1){
                effect.View = camera1.getViewMatrix();
                effect.Projection = camera1.getProjectionMatrix();}
            if (cameraActive == 2){
                effect.View = camera2.getViewMatrix();
                effect.Projection = camera2.getProjectionMatrix();}}
        mesh.Draw();}}
```

The draw model method takes in three parameters, the model, its transformation and its bone transformation. For each of the meshes it is then applied an effect relative to the active camera.

Game Function :: Setup Models

Overview

The purpose of this function is to setup the models. It will always be called at the start of the game, but can also be called when the game is over or has been won to retry the game.

Variable Description

Health

The variable health is used to store the player's health status. As this function will only ever be ran at the start of a game, it will need to be set the 100.

Score

This variable will be used to store the player's score, like the variable above, it will be set to zero as there is no points achieved.

Functional Description

Retrying?

```
public void setupModels(){
    health = 100;
    score = 0;
    if (retrying == true){
        //Retrying Code
    }
    if (retrying == false){
        //Starting code
    }
    sat.setPosition(new Vector3(-14.0f, -4.0f, -30.0f));
    moon.setPosition(new Vector3(7.0f, 5.0f, -8.0f));
    blade.setPosition(new Vector3(0.0f, 0.0f, -3000.0f));
}
```

Depending on whether the player is restrat the game or not, different code will be executed. Regardless of the choice, the player's health and score will always be reset, hence outside either if statement. Likewise, the satellite, moon and blade's position will be reset.

Retrying the game - Shields

```
for (int i = 0; i < 2; i++){
    shieldArray[i].isActive = true;
    shieldArray[i].shieldHealth = 150;}
}
```

Set the shield as active, and restore their health.

Retrying the game - Enemies

```
for (int j = 0; j < 20; j++){
    enemy[j].isAliveBool = true;
    enemy[j].edgeCollision = false;}
}
```

For each of the enemies, set them as active, and set edgeCollision to false.

```
for (int m = 0; m < 4; m++){
    xChange = new Vector3(4.5f, 0.0f, -10.0f);
    zChange = m * -5.0f;
}
```

```

xChange.Z += zChange;
for (int b = m * 5; b < (m * 5) + 5; b++){
    enemy[b].setPosition(xChange);
    enemy[b].setOriginalPos(xChange.X);
    xChange.X += -2.2f;}}

```

This function is to reset each of the enemies position back to the default. By setting a value for xChange (the position for the first enemy) each enemy after will have a reduce X position, until the fifth enemy. At the fifth enemy the Z position will be reduced, and then continue the reducing the X position. The purpose of this is to create a neat 5 by 4 row of enemies.

Retrying the game – Player lasers

```

for (int k = 0; k < 10; k++){
    LaserArray[k].setActive(false);}

```

For each of the player's ten lasers, ensure they are inactive.

Retrying the game – Enemy lasers

```

for (int p = 0; p < 2; p++){
    enemyLaserArray[p].setActive(false);}

```

For each of the two enemy laser's ensure they are inactive.

Starting the game - Shields

```

for (int j = 0; j < 2; j++){
    shieldArray[j] = new Shield();}
shieldArray[0].setPosition(new Vector3(-1.7f, 0.0f, 1.5f));
shieldArray[1].setPosition(new Vector3(1.7f, 0.0f, 1.5f));

```

For each of the shields, create a new instance of class Shield, then specify the position for both shields.

Starting the game - Enemies

```

for (int k = 0; k < 4; k++){
    xChange = new Vector3(4.5f, 0.0f, -10.0f);
    zChange = k * -5.0f;
    xChange.Z += zChange;
    for (int i = k * 5; i < (k * 5) + 5; i++){
        enemy[i] = new Enemy();
        enemy[i].isAliveBool = true;
        enemy[i].edgeCollision = false;
        enemy[i].setPosition(xChange);
        enemy[i].setOriginalPos(xChange.X);
        xChange.X += -2.2f;}}

```

This code is almost identical to the code used when retrying the game, however with the difference that each enemy is created as a new instance of class Enemy.

Starting the game – Player lasers

```

for (int i = 0; i < 10; i++){
    LaserArray[i] = new Laser();
    LaserArray[i].setActive(false);}

```

For each of the player's lasers, create a new instance of class Laser and then set them to inactive.

Starting the game – Enemy lasers

```
for (int i = 0; i < 2; i++){  
    enemyLaserArray[i] = new Laser();  
    enemyLaserArray[i].setActive(false);}}
```

For each of the enemy's laser, create a new instance of class Laser and like the player laser, set them to inactive.

Game Creation :: Menu

Overview

Before any of the cameras or models is used, the player is presented with a menu, which consists of three options – Start game, Instructions and Exit.

The menu was created using Adobe Illustrator to produce PNG image files that could be drawn to the screen as a two dimensional texture. Depending on the input from the user, the images would change accordingly to navigate through the different textures.

Initialization

```
private int gameState = 1;
private int menuSelection = 1;
private int showInstructions = 0;

Texture2D bgMenu;
Texture2D startGame;
Texture2D instructions;
Texture2D exit;
Texture2D startGameOVER;
Texture2D instructionsOVER;
Texture2D exitOVER;
Texture2D spacer;
Texture2D instructionsBG;
```

Variable Description

gameState

Variable gameState is used to refer to what is being processed on the screen. There are a total of four gameStates :

1. Display menu screen
2. Display game
3. Display game over screen
4. Display winning screen

The first gameState will always be one – this is to allow the player to view the instructions and/or quit the game, prior to the game.

menuSelection

The integer menuSelection is used to refer to what option is being selected on the menu screen. There are three options:

1. Start Game
2. Instructions
3. Exit

Start Game will always be the first selection; as a result menuSelection will equal one at the start of the program.

showInstructions

showInstructions in an integer that is used to refer to which instruction image is being shown. There are total of three images:

1. Story with “back” highlighted
2. Story with “controls” highlighted
3. Controls with “back” highlighted

Texture2D

As all the menu screens are an image, they are effectively a two dimensional texture that will be drawn to the screen. To hold the information, they must be a Texture2D.

Functional Description

Exit Game

```
if (getInput() == 6){this.Exit();}
```

This code will be accessible throughout the entire game. If the player hits either the back button the Xbox 360 controller pad, or the escape button, the game will exit.

Switching game State

```
switch (gameState){
```

The purpose of the gameState is to switch between displaying the menu screen, the actual game, game over screen and the winning screen.

gameState = 1

```
case 1:
    if (showInstructions == 0){
        if ((getInput() == 1) && (getInput() != lastInput)){
            if (menuSelection + 1 > 3){
                menuSelection = 3;
            }else{
                menuSelection++;
            }
        }
        if ((getInput() == -1) && (getInput() != lastInput)){
            if (menuSelection - 1 < 1){
                menuSelection = 1;
            }else{
                menuSelection--;
            }
        }
    }
}
```

The above code checks to see if the instructions are shown, if not it will then check the input from the user. If the user hits right then the value of menuSelection is increased, however as there are only three options, anything greater than 3 is set to 3. Likewise, anything less than 1 is set to 1.

Selection within instructions

```
if (showInstructions != 0){
    if ((getInput() == -1) && (getInput() != lastInput) &&
        (showInstructions != 3)){
        showInstructions = 1;}
    if ((getInput() == 1) && (getInput() != lastInput) &&
        (showInstructions != 3)){
        showInstructions = 2;}}
```

If any of the instructions are shown and if the player presses right, the second instructions image with “controls” highlighted is displayed as there can only be two possible displays. Similarly, if the player presses left, the “back” image is pressed.

Swap instructions screen

```
if ((getInput() == 3) && (getInput() != lastInput)){
    if (showInstructions != 0){
        switch (showInstructions){
```

If the player hits the enter button or “A” on the 360 controller, and they are currently shown the instructions image, then switch the showInstructions variable.

```
    case 1:
        showInstructions = 0;
        break;
    case 2:
        showInstructions = 3;
        break;
    case 3:
        showInstructions = 1;
        break;}
    break;}
```

- If the showInstructions equals 1 – “back” highlighted – then set it to 0, as no instructions are shown.
- If the showInstructions equals 2 – “Controls” highlighted – then show the third image – controls.
- Finally, if the controls image is displayed with “back” highlighted, display number 1.

Switching menuSelection

```
if (showInstructions == 0){
    switch (menuSelection){
        case 1:
            gameState = 2;
            break;

        case 2:
            showInstructions = 1;
            break;

        case 3:
            this.Exit(); break; }}} break;
```

If there are no instruction screens being displayed and enter has been pressed:

- If gameState is equal to 1 – start the game.
- If gameState is equal to 2 – show instructions.
- If gameState is equal to 3 – exit the game.

Remaining game states

```
case 2:
    //Start Game Code
    break;
case 3:
    if (getInput() == 3){
        gameState = 1;
        menuSelection = 1;
        showInstructions = 0;
        retrying = true;
    }
    break;

case 4:
    if (getInput() == 3){
        gameState = 1;
        menuSelection = 1;
        showInstructions = 0;
        retrying = true;
    }
    break;

lastInput = getInput();
base.Update(gameTime);}
```

If the menuSelection is equal to 2 then start the game – code omitted, available below.

If the menuSelection is equal to 3 or 4, then restart the game, resetting the variables.

Game Function :: Game Loop

Overview

This loop runs constantly if the gameState is 2. The purpose of this loop is to initially begin setting up the models and cameras of the scene. Once everything has been loaded the loop will make the enemies move, fire lasers and if the player moves or fires a laser, collisions will be calculated calling the respective methods to handle each event.

Functional Description

Play audio once

```
if (playOnce == false){
    themeMusicInstance = themeMusic.Play(1f, 0.0f, 0.0f, true);
    playOnce = true;}
```

If the game has just started, playOnce will be false and the music will play.

Retrying Game

```
if (retrying == true){
    if (soundIsOn == true){
        themeMusicInstance.Resume();}
    setupModels();
    retrying = false;
    setup = false;}
```

If the player is retrying the game then reset all the positions of the model by calling setupModels, and setting the two booleans retrying and setup to false. If the sound is already on, then restart the music.

Move Player

```
playerMove();
```

Call function playerMove.

Enemy Movement

```
for (int i = 0; i < 20; i++){
    if (enemy[i].isAliveBool == true){
        if (enemy[i].checkCollision() == true){
            enemy[i].position.Z += 1.0f;
            enemy[i].velocity.X *= -1.0f;
            enemy[i].setCollision(false);}
        enemy[i].position.X += enemy[i].velocity.X;
        if (enemy[i].position.Z > 0.0f){
            gameState = 3;
            themeMusicInstance.Pause();}}}
```

For each of the enemies in the game, check if they are alive. Then it will do one of three things depending on the position.

- If the enemy has reached the edge of the screen then move the enemy forwards and change direction.
- If they have reached the edge and their Z position is greater than 0, then it's game over.
- If neither are true, then increase the X position of the enemy

Check if game is won

```
int count = 0;
for (int i = 0; i < 20; i++){
if (enemy[i].isAliveBool == false){
    count++;
    if (count == 19){
        gameState = 4;
        themeMusicInstance.Pause();
    }
}
```

Set the counter to zero to begin with. For each enemy in the array, if the enemy isn't alive increase the counter. If the counter reaches 19, then all enemies are dead, resulting in the game winning screen being shown and the music paused.

Setup enemy lasers to fire

```
int previousLaser = 21;
if (setup == false){
for (int i = 0; i < 2; i++){
    for (int k = 0; k < 20; k++){
        if ((enemy[k].isAliveBool == true) && (previousLaser != k)){
            enemyLaserArray[i].setPosition(enemy[k].getPosition());
            enemyLaserArray[i].setActive(true);
            if (soundIsOn == true){
                enemyFire.Play(0.4f, 0f, 0f, false);
            }
            previousLaser = k;
            break;
        }
    }
    setup = true;
}
```

The first phase of firing the enemy lasers is to set the previous laser to a number greater than 20. The purpose of this is to ensure that two lasers are set in the loop to follow.

If setup (variable to check if both lasers haven't been setup) is false, begin the loop for both enemy lasers. Search through all the enemies and for the first two enemies, set the lasers to active with their positions. If the sound is on, play the enemyFire sound and set the variable previousLaser to k, to ensure that two different enemies fire.

Move enemy lasers

```
for (int i = 0; i < 2; i++){
    if (enemyLaserArray[i].isActive == true){
        if (enemyLaserArray[i].checkOutOfScreen(2) == true){
            enemyLaserArray[i].setActive(false);
            setup = false;
            enemyLaserArray[i].Update(timeDelta, 2);
        }
    }
}
```

For each of the enemy lasers, check to see if their active. If they are then call the function checkOutOfScreen(2) to see if they have reached past the camera position. If they have then set them to inactive and set setup to false. If they haven't reached the end then update the position.

Move player lasers

```
for (int i = 0; i < 10; i++){
    if (LaserArray[i].isActive == true){
        if (LaserArray[i].checkOutOfScreen(1) == true) {
            LaserArray[i].setActive(false);
        }
        LaserArray[i].Update(timeDelta, 1);
    }
}
```

Similar to move enemy lasers function, this checks to see all of the player lasers fired. If they are active, then check them to see if they are beyond the furthest enemy, if they are then set them to inactive. Otherwise, update the position of the laser.

Move satellite

```
Vector3 satPos = sat.getPosition();
satPos.X += 0.015f;
satPos.Y -= 0.01f;
sat.setPosition(satPos);
```

Get the satellite position and set it to vector 3 satPos. Increase the new variables X and reduce Y variables. Set this updated variable as the position of the satellite - as it's being updated it will move across the screen.

Move moon

```
Vector3 spin = moon.getPosition();
spin.X -= 0.005f;
spin.Y += 0.002f;
spin.Z += 0.003f;
moon.setPosition(spin);
```

The purpose of this method is purely to move the moon, very similar to the satellite movement. The variable spin is passed the moon's position and alters the X, Y and Z positions and applies it to the moon.

Collision Detection – Player laser

```
for (int i = 0; i < 20; i++){
    if (enemy[i].isAliveBool == true){
        BoundingBox enemyBounding = new
        BoundingBox(enemy[i].getPosition(),
        enemy[i].enemyModel.Meshes[0].BoundingBox.Radius * 0.09f);
    }
}
```

For each of the enemies that are active, create a new bounding sphere around them. This function creates the bounding sphere by getting the position of the enemy and creates a bounding sphere using the meshes of the model.

```

for (int k = 0; k < 10; k++){
    if (LaserArray[k].isActive == true){
        BoundingBox laserSphere = new
        BoundingBox(LaserArray[k].getPosition(),
        LaserArray[k].pLaserModel.Meshes[0].BoundingBox.Radius *
        0.006f);
    }
}

```

For each of the player lasers active, create a similar bounding sphere to the enemy's.

```

if (enemyBounding.Intersects(laserSphere)){
    if (soundIsOn == true){
        explosion.Play(0.4f, 0f, 0f, false);
        enemy[i].isActive = false;
    }
    LaserArray[k].isActive = false;
    score += 20;
    break;
}
}
}
}

```

If the enemy's bounding sphere intersects with the laser's bounding sphere, then play the explosion clip if the sound is on. Set the enemy to inactive as well as the laser and increase the score, then break from the loop.

Setup player bounding spheres

```

BoundingBox playerBounding = new
BoundingBox(player.getPosition(),
player.playerModel.Meshes[0].BoundingBox.Radius * 1.2f);

```

Similar to the bounding of the enemy, the playerBounding takes the position and creates a bounding sphere.

Collision Detection – Enemy laser and Player / Shield

```

for (int i = 0; i < 2; i++){
    if (enemyLaserArray[i].isActive == true){
        BoundingBox enemyLaserBounding = new
        BoundingBox(enemyLaserArray[i].getPosition(),
        enemyLaserArray[i].pLaserModel.Meshes[0].BoundingBox.Radius
        * 0.006f);
    }
}

```

Create new bounding sphere for each of the enemies lasers.

```

for (int k = 0; k < 2; k++){
    if (shieldArray[k].isActive == true){
        BoundingBox shieldBounding = new
        BoundingBox(shieldArray[k].getPosition(),
        shieldArray[k].shieldModel.Meshes[0].BoundingBox.Radius *
        0.02f);
    }
}

```

For each of the shields create a new bounding sphere.

```

if (shieldBounding.Intersects(enemyLaserBounding)){
    if (soundIsOn == true){
        shieldDestroy.Play(0.4f, 0f, 0f, false);
        enemyLaserArray[i].isActive = false;
        shieldArray[k].hit();
        shieldArray[k].checkShieldStatus();
    }
    if ((enemyLaserArray[0].isActive == false) &&
    (enemyLaserArray[1].isActive == false)){
        setup = false; break; }
}
}
}
}

```

If the shield intersects the enemy laser bounding sphere, then play the shieldDestroy sound if the sound is on. Set the enemy laser to inactive, then call the two shield functions – hit and check shield status, to see if the shield is still active.

If the both enemy lasers are false, then set setup to false.

```
if (playerBounding.Intersects(enemyLaserBounding)){
    if (soundIsOn == true){
        shieldDestroy.Play(0.4f, 0f, 0f, false);}
        enemyLaserArray[i].setActive(false);
        health -= 25;
        if ((enemyLaserArray[0].isActive == false) &&
            (enemyLaserArray[1].isActive == false)){
            setup = false;}
        break;}}}}}
```

If the player intersects with the enemy laser, then play the shieldDestroy sound clip, if the sound is on. Set the laser to inactive, and reduce the player health by 25. If both the lasers are inactive, set setup as false.

Check player's health

```
if (health < 1){
    gameState = 3;}
    themeMusicInstance.Pause();
}
break;
```

If the player's health is less than 1, then set the gameState to 3 (game over screen) and pause the music.

Game Function :: Player Move

Overview

This function checks to see the player input, if it's any of the listed below it will either move the object, play or mute the audio, change the camera or fire a laser.

Functional Description

Move left

```
private void playerMove(){
    if (getInput() == -1){
        player.moveLeft();
        camera2.setCameraPosition(player.getPosition());
    }
}
```

If the input equals -1 (if the left keyboard arrow, or the left thumbstick has been pressed left) then move the player left. Update the first person camera to the current position.

Move right

```
if (getInput() == 1){
    player.moveRight();
    camera2.setCameraPosition(player.getPosition());
}
```

If the input equals 1 (if the right keyboard arrow, or the left thumbstick has been pressed right) then move the player right. Update the first person camera to the current position.

Move forwards

```
if (getInput() == 8){
    player.moveForwards();
    camera2.setCameraPosition(player.getPosition());
}
```

If the input equals 8 (if the up keyboard arrow, or the left thumbstick has been pressed up) then move the player forwards. Update the first person camera to the current position.

Move backwards

```
if (getInput() == 2){
    player.moveBackwards();
    camera2.setCameraPosition(player.getPosition());
}
```

If the input equals 2 (if the down keyboard arrow, or the left thumbstick has been pressed down) then move the player backwards. Update the first person camera to the current position.

Change to camera 1

```
if ((getInput() == 20) && (getInput() != lastInput)){
    cameraActive = 1;
}
```

If the button "1" or the left trigger is pressed, then set camera 1 as active.

Change to camera 2

```
if ((getInput() == 12) && (getInput() != lastInput)){
    cameraActive = 2;}

```

If the button “2” or the right trigger is pressed, then set camera 2 as active.

Toggle sound

```
if ((getInput() == 12) && (getInput() != lastInput)){
    if (soundIsOn == true){
        themeMusicInstance.Pause();
        soundIsOn = false;
    }else{
        themeMusicInstance.Resume();
        soundIsOn = true;}}

```

If the “X” button on the joypad, or “S” on the keyboard is pressed, toggle the sound on or off, if the sound is already off or on, respectively.

Shoot

```
if ((getInput() == 5) && (getInput() != lastInput)){
    for (int i = 0; i < 10; i++){
        if (LaserArray[i].isActive == false){
            LaserArray[i].setPosition(player.getPosition());
            LaserArray[i].setActive(true);
            if (soundIsOn == true){
                playerFire.Play(0.4f, 0f, 0f, false);}
            break; }}}
    lastInput = getInput(); }

```

If the fire button has been pressed (“RB” or right shoulder button on the joypad, or the spacebar on the keyboard), then check all the lasers for an inactive laser. If there is an inactive laser, set its position to the player position then set it to active, then play the fire sound if the sound is on.

The code “lastInput = getInput()” is to prevent the player pressing the button and constantly firing until a different key is pressed.

Game Function :: Write Text

Overview

The writeText function is used to print text to the screen.

Functional Description

Write text to screen

```
private void writeText(string msg, Vector2 msgPos, Color msgColour){
    spriteBatch.Begin();
    string output = msg;
    Vector2 FontOrigin = fontToUse.MeasureString(output) / 2;
    Vector2 FontPos = msgPos;
    spriteBatch.DrawString(fontToUse, output, FontPos, msgColour);
    spriteBatch.End();
}
```

The function write text obtains three variables; the message to be displayed; its position on screen and the colour of the text. Upon entering the function, the spriteBatch is initiated using the Begin function, then the variable output is assigned the message needed to be displayed. FontOrigin calculates the centre of the string and is then all put together using the DrawString method.

Game Function :: Draw Method

Overview

The purpose of the draw method is to draw the items on the screen when requested by the main update loop. There are four respective different draw calls depending on the gameState: the menu, game, game over and win screen.

Functional Description

Setting up draw

```
protected override void Draw(GameTime gameTime){
    Window.Title = "El Space Aventura!";
```

Draw the screen, setting the title to “El Space Aventura!” the title of the game.

The Menu

Drawing the menu background

```
if (gameState == 1){
spriteBatch.Begin(SpriteBlendMode.AlphaBlend, SpriteSortMode.Deferred,
SaveStateMode.SaveState);
    if (showInstructions == 0){
        spriteBatch.Draw(bgMenu, new Rectangle(0,
            Window.ClientBounds.Height - bgMenu.Height,
            bgMenu.Width, bgMenu.Height), Color.White);
```

If there are no instructions being shown, then draw the default menu background.

Drawing “Start game”

```
if (menuSelection == 1){
spriteBatch.Draw(startGameOVER, new Rectangle(201, 300,
startGameOVER.Width, startGameOVER.Height), Color.White);
}else{
spriteBatch.Draw(startGame, new Rectangle(235, 295, startGame.Width,
startGame.Height), Color.White);}
```

If the menuSelection is 1 (the first menu selection is highlighted) then draw “Start Game” as red, otherwise draw it as white (two different images).

Drawing the spacer

```
spriteBatch.Draw(spacer, new Rectangle(295, 322, spacer.Width,
spacer.Height), Color.White);
```

The space is always drawn in the menu and does not need to change.

Drawing “Instructions”

```
if (menuSelection == 2){
    spriteBatch.Draw(instructionsOVER, new Rectangle(390, 300,
        instructionsOVER.Width, instructionsOVER.Height), Color.White);
}else{
    spriteBatch.Draw(instructions, new Rectangle(390, 300,
        instructions.Width, instructions.Height), Color.White);}
```

Similar to the code for “Start game”, if the item is selected swap images with the red text, otherwise display the white.

Drawing the spacer

```
spriteBatch.Draw(spacer, new Rectangle(455, 321, spacer.Width,
spacer.Height), Color.White);
```

Drawing “Exit”

```
if (menuSelection == 3){
    spriteBatch.Draw(exitOVER, new Rectangle(545, 300,
    exitOVER.Width, exitOVER.Height), Color.White);
}else{
    spriteBatch.Draw(exit, new Rectangle(545, 300, exit.Width,
    exit.Height), Color.White); }
```

Again, similar to “Start game”, if the player has selected “Exit” then swap it for the red image, otherwise display the white.

Drawing Instructions – “Back”

```
if (showInstructions == 1){
    spriteBatch.Draw(instructionsBG, new Rectangle(0,
    Window.ClientBounds.Height - instructionsBG.Height,
    instructionsBG.Width, instructionsBG.Height), Color.White);}
```

If showInstructions is equal to 1, then display the instructions, which shows the background story and “Back” highlighted.

Drawing Instructions – “Controls”

```
if (showInstructions == 2){
    spriteBatch.Draw(instructionsBG2, new Rectangle(0,
    Window.ClientBounds.Height - instructionsBG2.Height,
    instructionsBG2.Width, instructionsBG2.Height), Color.White);}
```

If showInstructions is equal to 2, then display the instructions, which shows the same story, but with “Controls” highlighted.

Drawing Instructions – “The Controls”

```
if (showInstructions == 3){
    spriteBatch.Draw(instructionsBG3, new Rectangle(0,
    Window.ClientBounds.Height - instructionsBG3.Height,
    instructionsBG3.Width, instructionsBG3.Height), Color.White);}
spriteBatch.End();}
```

If showInstructions equals 3, then display the instructions, which shows the controls – “Back” will always be highlighted.

The Game

Drawing the game and rendering enemy laser

```
if (gameState == 2){
    GraphicsDevice.Clear(Color.Black);
    for (int i = 0; i < 2; i++){
        if (enemyLaserArray[i].isActive == true){
            Matrix ELaserTransform =
            Matrix.CreateTranslation(enemyLaserArray[i].getPosition()
            );
            DrawModel(enemyLaserArray[i].getModel(), ELaserTransform,
            enemyLaserArray[i].getTrans()); }}
```

If the game has been set to begin, set the background to black, and draw the enemy lasers that are active. The model is drawn by obtaining the position, the model and transformation and then rendered.

Rendering player

```
Matrix modelTransform =
Matrix.CreateTranslation(player.getPosition());
DrawModel(player.getModel(), modelTransform, player.getTrans());
```

Similar to the enemy laser, obtain the player's position, model and transformation to create the model to be rendered.

Rendering shield

```
for (int k = 0; k < 2; k++){
    if (shieldArray[k].isActive == true){
        Matrix shieldTrans =
        Matrix.CreateTranslation(shieldArray[k].getPosition());
        DrawModel(shieldArray[k].getModel(), shieldTrans,
        shieldArray[k].getTrans()); }}
```

Check if the either of the shields are active, if they are then render using the same procedure as the player and enemy laser.

Rendering the enemy

```
for (int i = 0; i < 20; i++){
    if (enemy[i].isAliveBool == true){
        Matrix enemyTransform =
        Matrix.CreateRotationY(enemy[i].getRotation()) *
        Matrix.CreateTranslation(enemy[i].getPosition());
        DrawModel(enemy[i].getModel(), enemyTransform,
        enemy[i].getTrans());}}
```

Very similar to rendering the shield, check if the enemy is alive, if it is use the same procedure as the player and enemy laser.

Rendering player laser

```
for (int i = 0; i < 10; i++){
    if (LaserArray[i].isActive == true){
        Matrix LaserTransform =
        Matrix.CreateTranslation(LaserArray[i].getPosition());
        DrawModel(LaserArray[i].getModel(), LaserTransform,
        LaserArray[i].getTrans());}}
```

If the player laser is active, then draw it as per above.

Rendering the satellite

```
Matrix satTransform = Matrix.CreateTranslation(sat.getPosition());
DrawModel(sat.getModel(), satTransform, sat.getTrans());
```

Render the satellite using the same procedure as above.

Rendering the moon

```
Matrix moonTransform = Matrix.CreateTranslation(moon.getPosition()) *
Matrix.CreateScale(0.5f);
DrawModel(moon.getModel(), moonTransform, moon.getTrans());
```

Render the moon using the same procedure.

Rendering the blade

```
Matrix bladeTransform = Matrix.CreateTranslation(blade.getPosition())
* Matrix.CreateScale(0.0125f);
DrawModel(blade.getModel(), bladeTransform, blade.getTrans());
```

Render the blade using the same procedure.

The Overlays

Begin rendering of 2D textures

```
spriteBatch.Begin(SpriteBlendMode.AlphaBlend, SpriteSortMode.Deferred,
SaveStateMode.SaveState);
```

Initiate rendering of 2D textures.

Rendering first person perspective

```
if (cameraActive == 2){
    spriteBatch.Draw(firstPerson, new Rectangle(0,
Window.ClientBounds.Height - firstPerson.Height,
firstPerson.Width, firstPerson.Height), Color.White);}
```

If the cameraActive is 2 (first person view) draw the first person perspective overlay, by creating a new rectangle and applying the texture over it.

Rendering the heads up display

```
spriteBatch.Draw(hud, new Rectangle(0, Window.ClientBounds.Height -
hud.Height, hud.Width, hud.Height), Color.White);
spriteBatch.End();
```

The heads up display (HUD) is always rendered when the game has started. It is rendered using the same process as above. When its finished, end the spriteBatch as it's there are no more 2D textures.

Drawing text to screen

```
writeText(health + "%", new Vector2(35, 573), Color.White);  
writeText(score.ToString(), new Vector2(730, 573), Color.White);
```

The first is to draw the player's remaining health and a percentage sign after it. The second `writeText`, takes the variable and converts it to a text (otherwise it can't print) and displays it below the respective heading in the HUD.

Game over or Winner

Displaying either game over or winner

```
if ((gameState == 3) || (gameState == 4)){
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend,
        SpriteSortMode.Deferred, SaveStateMode.SaveState);
```

Both game over and winner require the spriteBatch to be reinitiated.

Drawing game over

```
if (gameState == 3){
    spriteBatch.Draw(gameOver, new Rectangle(0,
        Window.ClientBounds.Height - gameOver.Height, gameOver.Width,
        gameOver.Height), Color.White);
    scoreWin = score;}
```

If the gameState is gameOver, draw the game over screen and print the score.

Drawing winner

```
if (gameState == 4){
    spriteBatch.Draw(win, new Rectangle(0,
        Window.ClientBounds.Height - win.Height, win.Width, win.Height),
        Color.White);
    scoreWin = score * health;}
```

If the gameState is win, then draw the game won screen. The score to be printed will be multiplied by the health they have left.

Displaying score – for winner

```
string scoreText = scoreWin.ToString();
Vector2 FontOrigin = fontToUse.MeasureString(scoreText) / 2;
Vector2 FontPosition = new Vector2(450, 305);
spriteBatch.DrawString(fontToUse, scoreText, FontPosition,
    Color.White);
spriteBatch.End();}
base.Draw(gameTime);}
```

In order to display the score for the winner, it must first calculate its length and reposition it then redraw the text.

Game Function :: Unload Content

Overview

The purpose of this function is to unload all the content of the game.

Functional Description

Unload content

```
protected override void UnloadContent() { }
```

There is no code other than the function name, however this code will still remove all the content upon exiting the application.

Game Function :: Get Input

Overview

This method, which can be called at anytime during the runtime, takes the value of the keyboard or the Xbox 360 controller. After receiving the input, it interprets the code by returning an integer value respective to the button pressed.

Functional Description

Get input - Right

```
public int getInput(){  
    if (GamePad.GetState(PlayerIndex.One).ThumbSticks.Left.X > 0.7f){  
        return 1;}  
    if (keyboardState.IsKeyDown(Keys.Right)){  
        return 1;}  
}
```

If the keyboard right button is pressed or the left thumb stick is pressed to the right, return 1.

Get input - Left

```
if (GamePad.GetState(PlayerIndex.One).ThumbSticks.Left.X < -0.7f){  
    return -1;}  
if (keyboardState.IsKeyDown(Keys.Left)){  
    return -1;}  
}
```

If the keyboard left button is pressed or the left thumb stick is pressed to the left, return -1.

Get input - Up

```
if (GamePad.GetState(PlayerIndex.One).ThumbSticks.Left.Y > 0.7f){  
    return 8;}  
if (keyboardState.IsKeyDown(Keys.Up)){  
    return 8;}  
}
```

If the keyboard up button is pressed, or the left thumb stick is pressed up, return 8.

Get input - Down

```
if (GamePad.GetState(PlayerIndex.One).ThumbSticks.Left.Y < -0.7f){  
    return 2;}  
if (keyboardState.IsKeyDown(Keys.Down)){  
    return 2;}  
}
```

If the keyboard down button is pressed, or the left thumb stick is pressed down, return 2.

Get input - Enter

```
if (GamePad.GetState(PlayerIndex.One).Buttons.A ==  
    ButtonState.Pressed){  
    return 3;}  
if (keyboardState.IsKeyDown(Keys.Enter)){  
    return 3;}  
}
```

If the keyboard enter button is pressed, or the "A" button is pressed, return 3.

Get input - Exit

```
if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==  
ButtonState.Pressed){  
    return 6;}  
if (keyboardState.IsKeyDown(Keys.Escape)){  
    return 6;}
```

If the keyboard escape button is pressed, or the “Back” button is pressed, return 6.

Get input - Fire

```
if (GamePad.GetState(PlayerIndex.One).Buttons.RightShoulder ==  
ButtonState.Pressed){  
    return 5;}  
if (keyboardState.IsKeyDown(Keys.Space)){  
    return 5;}
```

If the keyboard spacebar button is pressed, or the right shoulder “RB” button is pressed, return 5.

Get input – Default camera

```
if (GamePad.GetState(PlayerIndex.One).Triggers.Left > 0.7f){  
    return 20;}  
if (keyboardState.IsKeyDown(Keys.D1)){  
    return 20;}
```

If the keyboard “1” button is pressed, or the left trigger “LT” is pressed, return 20.

Get input – First person camera

```
if (GamePad.GetState(PlayerIndex.One).Triggers.Right > 0.7f){  
    return 30;}  
if (keyboardState.IsKeyDown(Keys.D2)){  
    return 30;}
```

If the keyboard “2” button is pressed, or the right trigger “RT” is pressed, return 30.

Get input – Audio on/off

```
if (GamePad.GetState(PlayerIndex.One).Buttons.X ==  
ButtonState.Pressed){  
    return 12;}  
if (keyboardState.IsKeyDown(Keys.S)){  
    return 12;}
```

If the keyboard “S” button is pressed, or the “X” button is pressed, return 12.

Get input - Otherwise

```
return 0;}
```

Otherwise, if there is no input value to the list above, return 0.

Game Class :: Ambient Model

Overview

The purpose of the ambient model is to provide the player with aesthetically pleasing visuals to enhance the overall game play and believability.

In the game, there are three ambient models – a blade, a moon and a satellite – the moon and satellite will merely move about the screen, whilst the blade will be stationary. Neither of them will be shoot-able, however it should provide the player a greater sense of being in space.

Class File

```
ambientModel.cs  
  
public Model ambientModelM;  
public Matrix[] ambientTrans;  
public Vector3 position;  
  
public ambientModel(){}
```

Variable Description

ambientModelM

This variable will hold the model that is used to for the ambient model.

ambientTrans

This variable will hold the transformation applied to the model .

position

The position of the ambient model will be stored .

Functional Description

Load model

```
public void loadModel(Model mdl){  
    ambientModelM = mdl;}  
}
```

This function takes in a model and sets the variable ambientModelM to the passed in model.

Set model transformation

```
public void modelTrans(Matrix[] mdlTrans){  
    ambientTrans = mdlTrans;}  
}
```

This function takes in the model transformation and sets it to the variable ambientTrans.

Get model transformation

```
public Matrix[] getTrans(){  
    return ambientTrans;}  
}
```

Returns the matrix array of the ambient model's transformation.

Get model

```
public Model getModel(){  
    return ambientModelM;}  
}
```

Returns the model held in the variable ambientModelM.

Set position

```
public void setPosition(Vector3 sP){  
    position = sP;}  
}
```

This function takes in the value of a vector 3 and assigns this to the position of the model.

Get position

```
public Vector3 getPosition(){  
    return position;}  
}
```

This function returns the value of the position.

Game Class :: Cameras

Overview

There are two main cameras that were created, the first was the third person perspective camera and the second was a first person perspective camera.

The third person camera was statically positioned slightly behind and above the player, whilst the first person camera is made to update its position depending on the player's character position.

Class File

Camera.cs

```
private Matrix viewMatrix;  
private Matrix projectionMatrix;  
private Matrix worldMatrix;  
private Vector3 camLookTarget;
```

Variable Description

World matrix

Variable worldMatrix transforms 3D data from model space to world space.

View matrix

Variable viewMatrix transforms from world space to view space.

Projection matrix

Variable projectionMatrix transforms from view space to screen space.

Camera look at target

The camLookTarget is the position where the camera should face.

Functional Description

Set View Matrix

```
public void setViewMatrix(Vector3 position, Vector3 target){  
    camLookTarget = target;  
    viewMatrix = Matrix.CreateLookAt(position, target, Vector3.Up);}
```

Pass in variables – position and target, this will be used to create the view matrix, setting its position and look at target, respectively.

Set Projection Matrix

```
public void setProjectionMatrix(float aR){  
    projectionMatrix =  
    Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(45),  
    aR, 1.0f, 1000.0f);  
    worldMatrix = Matrix.Identity;}
```

Create the projection matrix by declaring its field of view, aspect ratio and the near and far clipping planes. Finally, create the world matrix, setting it to an identity matrix.

Set Camera Position

```
public void setCameraPosition(Vector3 playerPosition){
    camLookTarget.X = playerPosition.X;
    camLookTarget.Y = 0f;
    viewMatrix = Matrix.CreateLookAt(playerPosition, camLookTarget,
    Vector3.Up);}
```

This function passes in the value of the player position from the main function. The (first person) camera is then set to the player position, and its look at target is the same x value of the player position. This is to ensure the camera looks in a straight line.

Get View Matrix

```
public Matrix getViewMatrix(){
    return viewMatrix;}
```

This function returns the view matrix value.

Get Projection Matrix

```
public Matrix getProjectionMatrix(){
    return projectionMatrix;}
```

This function returns the projection matrix.

Get World Matrix

```
public Matrix getWorldMatrix(){
    return worldMatrix;}
```

This function returns the world matrix.

Game Class :: Player

Overview

The player class will be used to create an instance of the player.

Class File

```
Player.cs  
  
public Model playerModel;  
public Matrix[] playerTrans;  
public Vector3 playerPosition;
```

Variable Description

Player model

Variable playerModel holds the player model information.

Player transformation

Variable playerTrans holds the player models transformation.

Player position

Variable playerPosition is used to store the position of the player.

Functional Description

Constructor

```
public Player(){playerPosition = new Vector3(0.0f, 0.0f, 0.0f);}
```

When an instance of player is created, set it's position to the origin.

Load model

```
public void loadModel(Model mdl){playerModel = mdl;}
```

Pass in a model and set variable playerModel to the loaded model.

Load Model Transformation

```
public void modelTrans(Matrix[] mdlTrans){playerTrans = mdlTrans;}
```

Load the model transformation and set variable playerTrans to the loaded transformation.

Get Model Transformation

```
public Matrix[] getTrans(){return playerTrans;}
```

Return the player model transformation.

Get Model

```
public Model getModel(){return playerModel;}
```

Return the player model.

Set player position

```
public void setPosition(Vector3 sP){playerPosition = sP;}
```

Set the position of the player.

Get player position

```
public Vector3 getPosition(){return playerPosition; }
```

Return the player position.

Move player left

```
public void moveLeft(){  
    if(playerPosition.X < -3.4f){ playerPosition.X = -4.0f;}  
    else{playerPosition.X -= 0.5f;}  
}
```

If the player's position is less than -3.4 units, then set it to -4, otherwise, reduce it by 0.5.

Move the player right

```
public void moveRight(){  
    if (playerPosition.X > 3.4f){playerPosition.X = 4.0f;}  
    else{playerPosition.X += 0.5f;}  
}
```

If the player's position is greater than 3.4 units, set it to 4 units, otherwise, increase by 0.5.

Move the player forwards

```
public void moveForwards(){  
    if (playerPosition.Z < -1.9f){playerPosition.Z = -2.0f;}  
    else{playerPosition.Z += -0.5f;}  
}
```

If the players position in the z axis is less than -1.9 units, then set it to -2 units. Otherwise reduce the position by half a unit.

Move the player backwards

```
public void moveBackwards(){  
    if (playerPosition.Z > 3.9f){playerPosition.Z = 4.0f;}  
    else{playerPosition.Z += 0.5f;}  
}
```

If the player position in the z axis is greater than 3.9 units, then set it to 4; otherwise increase the position by half a unit.

Game Class :: Enemy

Overview

The enemy class will be used to create multiple enemies used in the game. Each enemy created will require a position, rotation, a Boolean to check if it's alive as well as variables for used for its movement.

Class File

```
Enemy.cs  
  
public Model enemyModel;  
public Matrix[] enemyTrans;  
public Vector3 position;  
public float rotation;  
public bool isAliveBool;  
private float originalPos;  
private float nextPosR;  
private float nextPosL;  
public Vector3 velocity;  
public bool edgeCollision;
```

Variable Description

enemyModel

This variable will hold the model data for the enemy.

enemyTrans

The enemy transformation will be held in this variable.

Position

The enemy position will be stored.

Rotation

The rotation of the model will be stored.

isAliveBool

This boolean variable will be used to check if the enemy is alive or not.

originalPos

This variable will be used to store the original position of the enemy, which will later be used when moving the enemy.

nextPosR

The variable nextPosR is used in the checkCollision function, to see if the enemy has collided with the right edge of the screen.

nextPosL

The variable nextPosL is used in the checkCollision function, to see if the enemy has collided with the left edge of the screen.

Velocity

This variable will hold the speed of the enemy in the form of a vector 3, for all three directions – x,y and z.

edgeCollision

The variable edgeCollision will be used to check if a collision has occurred with the screen.

Functional Description

Constructor

```
public Enemy(){
    rotation = 1.55f;
    velocity.X = 0.0625f;
    velocity.Z = 0.125f;
    velocity.Y = 0.0f;}
```

The constructor of the class enemy is used to set the rotation of the model and to set the default velocity for each enemy created.

checkCollision

```
public bool checkCollision(){
    nextPosR = originalPos + 5.0f;
    nextPosL = originalPos - 5.0f;
    if (position.X > nextPosR){
        edgeCollision = true;
        return edgeCollision;}
    if (position.X < nextPosL){
        edgeCollision = true;
        return edgeCollision;}
    return false;}
```

This function is used to check if the enemy has hit the edge of the screen. It sets the values of “nextPosR” and “nextPosL” to five units plus and minus the original position respectively.

If the current position of the enemy is greater or less than “nextPosR” or “nextPosL”, respectively, then it has reached the edge, making the boolean edgeCollision true and hence changing direction.

setCollision

```
public void setCollision(bool input){
    edgeCollision = input;}
```

The purpose of this function is to set whether a collision should occur, this is mainly used when restarting the game.

setOriginalPos

```
public void setOriginalPos(float input){
    originalPos = input;}
```

This function is used to set the value of the original position by calling the function and passing in a float value.

getOriginalPosition

```
public float getOriginalPos(){  
    return originalPos;}  
}
```

This function returns the original position of the enemy.

isAlive

```
public Boolean isAlive(){  
    return isAliveBool;}  
}
```

This function returns whether or not the enemy is alive.

getSpeed

```
public Vector3 getSpeed(){  
    return velocity;}  
}
```

This function returns the velocity of the enemy.

loadModel

```
public void loadModel(Model mdl){  
    enemyModel = mdl;}  
}
```

This function loads in a model and assigns it to the variable enemyModel.

modelTrans

```
public void modelTrans(Matrix[] mdlTrans){  
    enemyTrans = mdlTrans;}  
}
```

This function's purpose is to assign the passed in matrix array of model transformation to enemyTrans.

getTrans

```
public Matrix[] getTrans(){  
    return enemyTrans;}  
}
```

This function returns the model transformation.

getModel

```
public Model getModel(){  
    return enemyModel;}  
}
```

Return the enemy's model.

setPosition

```
public void setPosition(Vector3 sP){  
    position = sP;}  
}
```

Set the enemy's position to the vector 3 passed in.

getPosition

```
public Vector3 getPosition(){  
    return position;}  
}
```

Return the value of the enemy's position.

moveLeft

```
public void moveLeft(Vector3 move){  
    position += move;}
```

Increase the position value by the passed in vector 3 value.

setRotation

```
public void setRotation(float sR){  
    rotation = sR;}
```

This function sets the rotation value by using the passed in float value.

getRotation

```
public float getRotation(){  
    return rotation;}
```

This function returns the value of the rotation.

Game Class :: Laser

Overview

The laser class is used for both enemy and player lasers, as both lasers are only indifferent through the direction they travel and the colour of the texture applied to the model.

Class File

```
LaserClass.cs
```

```
public Model pLaserModel;  
public Matrix[] pLaserTrans;  
public Vector3 pLaserPos;  
public float speed;  
public bool isActive;
```

Variable Description

```
pLaserModel
```

This variable will be storing the model of the laser.

```
pLaserTrans
```

This variable will be holding the transformations of the laser model.

```
pLaserPos
```

The position of the model will be stored in this variable.

```
Speed
```

The speed of the laser will be stored in this variable.

```
isActive
```

This Boolean variable will be used to check if the laser is active.

Functional Description

```
Constructor
```

```
public Laser(){speed = 6.5f;}
```

This constructor will specify a speed of 6.5 units for each laser created.

```
Load model
```

```
public void loadModel(Model mdl){  
    pLaserModel = mdl;} 
```

This function passes in a model and then sets the variable pLaserModel to it.

```
Load model transformation
```

```
public void modelTrans(Matrix[] mdlTrans){  
    pLaserTrans = mdlTrans;} 
```

This function loads in the model transformations and assigns it to the variable pLaserTrans.

Get laser transformation

```
public Matrix[] getTrans(){  
    return pLaserTrans;}  
}
```

This function returns the matrix array of the laser transformation held in pLaserTrans.

Get model

```
public Model getModel(){  
    return pLaserModel;}  
}
```

This function returns the model of the laser held in pLaserModel.

Set position

```
public void setPosition(Vector3 sP){  
    pLaserPos = sP;}  
}
```

This function sets the variable of pLaserPos (position) by using the passed in variable.

Get position

```
public Vector3 getPosition(){  
    return pLaserPos;}  
}
```

This function returns the value of pLaserPos, which is the position.

Check if active

```
public bool checkActive(){  
    return isActive;}  
}
```

This function will return whether or not the laser is active.

Set active

```
public void setActive(bool input){  
    isActive = input;}  
}
```

This function will set if the laser to active or inactive.

Update laser position

```
public void Update(float delta, int laserType){  
    if (laserType == 1){  
        pLaserPos.Z -= speed * delta;}  
    if (laserType == 2){  
        pLaserPos.Z += speed * delta;}}  
}
```

This function takes in two parameters! the time difference and the laser type. If the laser type is 1 then it belongs to the player, otherwise it's the enemies. The importance of this is that the laser will move in opposite directions depending on the laser owner.

The laser will move by changing the Z value by adding or subtracting the speed multiplied by the time difference.

Check if the laser is outside the screen

```
public bool checkOutOfScreen(int laserType){
    if (laserType == 1){
        if (pLaserPos.Z < -28.0f){
            return true;}
        return false;}
    if (laserType == 2){
        if (pLaserPos.Z > 3.0f){
            return true;}
        return false;}
    return false;}}
```

This function's purpose is to check whether the laser has gone out of the screen. Like the update function before this, this function takes in the value of type int to distinguish whether or not this is the enemies' or player's laser.

If it's the player's laser and its beyond -28 units, or the enemies' is beyond 3 units, then return true, otherwise return false.

Game Class :: Shield

Overview

The shield class is used to create shields in the game that can be destroyed. Like the shields in the original Space Invaders, these shields are also stationary.

Class File

Shield.cs

```
public Model shieldModel;  
public Matrix[] shieldTrans;  
public Vector3 position;  
public float rotation;  
public int shieldHealth;  
public bool isActive;
```

Variable Description

shieldModel

This variable will be used to hold the model data for the shield.

shieldTrans

This variable will be used to hold the model transformation.

Position

This variable will hold the position of the shield.

Rotation

This variable will hold the rotation of the shield.

shieldHealth

ShieldHealth will hold how much health the shield has remaining.

isActive

The isActive boolean is used to see if the shield is still active.

Functional Description

Constructor

```
public Shield(){  
    shieldHealth = 150;  
    isActive = true;}  
}
```

The constructor sets the the shield to active and sets it's health to 150.

Hit

```
public void hit(){  
    shieldHealth = shieldHealth - 50;}  
}
```

This function is called when the shield has been hit. Once called the shield's health is reduce by 50.

Load model

```
public void loadModel(Model mdl){
    shieldModel = mdl;}

```

This function sets the passed in model to shieldModel.

Load model transformation

```
public void modelTrans(Matrix[] mdlTrans){
    shieldTrans = mdlTrans;}

```

This function is used to set the passed in model transformation to the variable shieldTrans.

Get Transformation

```
public Matrix[] getTrans(){
    return shieldTrans;}

```

This function returns the value of the shield model's transformation.

Get model

```
public Model getModel(){
    return shieldModel;}

```

This function returns the model of the shield.

Set position

```
public void setPosition(Vector3 sP){
    position = sP;}

```

This function sets the position of the shield using the passed in variable.

Get position

```
public Vector3 getPosition(){
    return position;}

```

This function returns the position of the shield.

Set rotation

```
public void setRotation(float sR){
    rotation = sR;}

```

This function's purpose is to set the rotation of the model using the passed in value.

Get rotation

```
public float getRotation(){
    return rotation;}

```

This function returns the value of the rotation.

Check shield status

```
public void checkShieldStatus(){
    if (shieldHealth < 1){
        isActive = false;}}

```

This function's purpose is to set the shield to inactive, if it's health is less than 1.

Development Issues

Throughout the development process of “El Space Aventura!” a number of problems were encountered, the following is a reasoning of how each problem was corrected or why it is still left in the program.

Enemy Firing

It was intended to have the enemy fire their lasers in a similar fashion to that of Space Invaders by having a random alien fire a laser in the front row, and if that random alien was deceased, then it would select the alien in the row behind it. However, despite many coding attempts it was not possible to attain the same functionality.

In order to attain a convincing form of artificial intelligence, the developer opted for a more simplistic approach, whereby the code would check from the start and select the first two alive enemies to fire.

Whilst this does provide the gamer with some form of emergency to shoot the correct alien, it is too predictable and makes the game feel quite basic.

Enemy Re-spawn

It was the developer’s intention to create a game, where the player could retry the game, should s/he fail or win the game. However, despite numerous attempts to correct the code the enemies re-appear out of sync of each other upon retrying the game.

It was with regret that this could not be fixed given the time constraint of the project.

Shield Collision Detection

The original model used for the shield was a massive model, that had to be scaled by a float value of 0.005, despite the bounding sphere scaled by the same factor, the collision would not register.

In order to correct this, the developer altered the dimensions using 3DS Max and re-imported it into the game. This corrected the problem allowing for reasonable collision detection.

References

Models

3Ds Max:

<http://usa.autodesk.com/adsk/servlet/pc/index?siteID=123112&id=13567410>

Player spaceship (Accessed 20th November 2009):

<http://www.xnafusion.com/?s=ship>

Enemy spaceship (Accessed 21st November 2009):

<http://www.theforce.net/cifi3d>

Player and enemy laser (Accessed 20th November 2009):

Note: Despite being the same model, this model was subsequently edited using 3DS Max and was applied a different texture to provide a different colour for each of the laser types.

<http://www.archibase.net/download/6568.html>

Ambient model – Moon (Accessed 20th November 2009):

<http://www.sharecg.com/v/5157/3d-model/golfball>

Ambient model – Satellite (Accessed 20th November 2009):

http://www.nasa.gov/multimedia/3d_resources/assets/Cassini_Assy.html

Ambient model – Blade (Accessed 8th December 2009):

<http://www.turbosquid.com/FullPreview/Index.cfm/ID/444060>

Audio

Theme Song

“Power Plant” composed by Simon Viklund

Published by Sumthing Else Music Works 2008

Player Laser (Accessed 25th November 2009):

<http://www.freesound.org/samplesViewSingle.php?id=28917>

Enemy Laser (Accessed 27th November 2009):

<http://www.freesound.org/samplesViewSingle.php?id=39459>

Explosion (Accessed 26th November 2009):

<http://www.freesound.org/samplesViewSingle.php?id=35643>

Shield Explosion (Accessed 25th November 2009):

<http://www.freesound.org/samplesViewSingle.php?id=35463>

Code

Code assistance when key was pressed down, the lasers would rapid fire:

http://www.gamedev.net/community/forums/topic.asp?topic_id=430108&whichpage=1�

Enemy movement code – despite not using the assistants code, it was useful to see how other alternatives could be done:

<http://forums.xna.com/forums/t/42694.aspx>

Explanation of matrices

http://www.toymaker.info/Games/XNA/html/xna_matrix.html

Code :: Game1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace Lab6CollisionDetectionV3
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        SpriteFont fontToUse;

        //Menu
        private int gameState = 1;
        private int menuSelection = 1;
        private int showInstructions = 0;
        Texture2D bgMenu;
        Texture2D startGame;
        Texture2D instructions;
        Texture2D exit;
        Texture2D startGameOVER;
        Texture2D instructionsOVER;
        Texture2D exitOVER;
        Texture2D spacer;
        Texture2D instructionsBG;
        Texture2D instructionsBG2;
        Texture2D instructionsBG3;

        //First Person HUD
        Texture2D firstPerson;

        //Game States - Game Over Screen / Win Screen
        Texture2D gameOver;
        Texture2D win;

        //Sounds
        private SoundEffectInstance themeMusicInstance;
        private SoundEffect themeMusic;
        private SoundEffect playerFire;
        private SoundEffect enemyFire;
        private SoundEffect explosion;
        private SoundEffect shieldDestroy;
        private bool soundIsOn = true;

        //Player
        private Player player = new Player();
        public int health;
        public int score;
        private int scoreWin;
    }
}
```

```

//Shield
private Shield[] shieldArray = new Shield[2];

//Array of Enemies
private Enemy[] enemy = new Enemy[20];
private bool setup = false;
private Vector3 xChange;
private float zChange;

//Ambient Model
private ambientModel sat = new ambientModel();
private ambientModel moon = new ambientModel();
private ambientModel blade = new ambientModel();

//Array of Enemy Lasers
private Laser[] enemyLaserArray = new Laser[2];

//Array of Player Lasers
private Laser[] LaserArray = new Laser[10];

//HUD
Texture2D hud;

//Cameras
private Camera camera1 = new Camera();
private Camera camera2 = new Camera();
private int cameraActive = 1;

private int lastInput;

//Retry
private bool retrying = false;
private bool playOnce = false;

//Constructor of the game
public Game1()
{
    //Create a new GraphicsDeviceManager to handle the
configuration
    //and management of the graphics specified for the game
    graphics = new GraphicsDeviceManager(this);

    //Set default content location
    Content.RootDirectory = "Content";
}

/**
 * Method: Initialize
 * Initialize the basis of the game
 */

protected override void Initialize()
{
    //Set the mouse visible inside the game window
    IsMouseVisible = true;

    //Setup the cameras
    setupCamera();

    //Setup the models
    setupModels();
}

```

```

        //Automatically enumerate through all game components added
        base.Initialize();
    }

    /**
     * Method: SetupEffectTransformDefaults
     * Obtain the data that makes the model
     * Apply basic effects onto the model - lighting, view and
projection matrix
     * Return the end product
     */

    private Matrix[] SetupEffectTransformDefaults(Model myModel)
    {
        //Obtain bones used in the model
        Matrix[] absoluteTransforms = new Matrix[myModel.Bones.Count];

        //Flatten bone hierarchy and put the bones into an array
        myModel.CopyAbsoluteBoneTransformsTo(absoluteTransforms);

        //For each mesh in the model
        foreach (ModelMesh mesh in myModel.Meshes)
        {
            //Apply the basic effect
            foreach (BasicEffect effect in mesh.Effects)
            {
                //As the default camera is cameral, use the following
variables
                effect.EnableDefaultLighting();
                effect.View = camera1.getViewMatrix();
                effect.Projection = camera1.getProjectionMatrix();
            }
        }
        return absoluteTransforms;
    }

    /**
     * Method: Write Text
     * Obtain centre of the string and print
     */

    private void writeText(string msg, Vector2 msgPos, Color msgColour)
    {
        //Begin rendering of text
        spriteBatch.Begin();

        //Declare the string output the string passed in
        string output = msg;

        //Obtain the centre position of the string
        Vector2 FontOrigin = fontToUse.MeasureString(output) / 2;

        //Decalre the position of the font to the value passed in
        Vector2 FontPos = msgPos;

        //Draw the string in, using the values just declared
        spriteBatch.DrawString(fontToUse, output, FontPos, msgColour);

        //End of text rendering
        spriteBatch.End();
    }

```

```

    }

    /**
     * Method: Draw Model
     * Method for drawing the model
     */

    public void DrawModel(Model model, Matrix modelTransform, Matrix[]
absoluteBoneTransforms)
    {
        //For each mesh in the model
        foreach (ModelMesh mesh in model.Meshes)
        {
            //For each effect in the mesh
            foreach (BasicEffect effect in mesh.Effects)
            {
                //Set the world matrix to the mesh multiplied by the
transform created in the method: SetupEffectTransformDefaults
                effect.World =
absoluteBoneTransforms[mesh.ParentBone.Index] * modelTransform;

                //If camera 1 is active
                if (cameraActive == 1)
                {
                    //Set the effect view to camera 1's view matrix
                    effect.View = camera1.getViewMatrix();

                    //Set the effect projection to camera 1's
projection matrix
                    effect.Projection = camera1.getProjectionMatrix();
                }

                //If camera 2 is active
                if (cameraActive == 2)
                {
                    //Set the effect view to camera 2's view matrix
                    effect.View = camera2.getViewMatrix();

                    //Set the effect projection to camera 2's
projection matrix
                    effect.Projection = camera2.getProjectionMatrix();
                }
            }
            //Draw the mesh
            mesh.Draw();
        }
    }

    /**
     * Method: Setup Camera
     * Set the two cameras default position and the target
     * Define variable aspect ratio and pass the for Projection Matrix
     */

    private void setupCamera()
    {
        //Set aspectRatio to the width of the window divided by the
height

```

```

        float aspectRatio =
(float)graphics.GraphicsDevice.Viewport.Width /
(float)graphics.GraphicsDevice.Viewport.Height;

        //Set camera 1's position
Vector3 cameraPosition = new Vector3(0.0f, 5.0f, 10.0f);

        //Set camera 2's position
Vector3 cameraPosition2 = new Vector3(0, 0, 0);

        //Set the look at target for camera 1
Vector3 target1 = new Vector3(0, 0, 0);

        //Set the look at target for the camera 2
Vector3 target2 = new Vector3(0, 0, -5);

        //Pass the values of position and look at target, for camera
1's view matrix
        camera1.setViewMatrix(cameraPosition, target1);

        //Pass in the value of aspect ratio, for camera 1's projection
matrix
        camera1.setProjectionMatrix(aspectRatio);

        //Pass the values of position and look at target, for camera
2's view matrix
        camera2.setViewMatrix(cameraPosition2, target2);

        //Pass in the value of aspect ratio, for camera 2's projection
matrix
        camera2.setProjectionMatrix(aspectRatio);
    }

    /**
     * Method: Setup Mddels
     * Reset player stats - Health and Score
     * If the player is retrying the game, reset object status
     * If player is playing for the first time, create instances of
objects
    */

    public void setupModels()
    {
        //Set player health to 100
        health = 100;

        //Set the player's score to 0
        score = 0;

        //If retrying is true
        if (retrying == true)
        {
            //Reset Shields
            for (int i = 0; i < 2; i++)
            {
                //Set the shield to acitve
                shieldArray[i].isActive = true;

                //Reset the shield's health to 150
                shieldArray[i].shieldHealth = 150;
            }
        }
    }

```

```

//Reset Enemies
for (int j = 0; j < 20; j++)
{
    //Set all enemies to alive
    enemy[j].isAliveBool = true;

    //Set the enemy to not reached the edge
    enemy[j].edgeCollision = false;
}

//Reset Enemy Positions
for (int m = 0; m < 4; m++)
{
    //Ensure all rows begin with same positions
    xChange = new Vector3(4.5f, 0.0f, -10.0f);

    //Change z value dependant on column
    zChange = m * -5.0f;

    //Update z value of change
    xChange.Z += zChange;

    for (int b = m * 5; b < (m * 5) + 5; b++)
    {
        //Define enemy position using values
        enemy[b].setPosition(xChange);

        //Define original position;
        enemy[b].setOriginalPos(xChange.X);

        //Make next enemy to the right of created one
        xChange.X += -2.2f;
    }
}

//Reset Player Lasers
for (int k = 0; k < 10; k++)
{
    //Set the Player Laser to not active
    LaserArray[k].setActive(false);
}

//Reset Enemy Lasers
for (int p = 0; p < 2; p++)
{
    //Set Enemy Laser to not active
    enemyLaserArray[p].setActive(false);
}
}

//If the player is not retrying - i.e. starting for first time
if (retrying == false)
{
    //Shield
    for (int j = 0; j < 2; j++)
    {
        //Create new instance of the shield
        shieldArray[j] = new Shield();
    }

    //Set the shield 1's position
    shieldArray[0].setPosition(new Vector3(-1.7f, 0.0f, 1.5f));
}

```

```

//Set the shield 2's position
shieldArray[1].setPosition(new Vector3(1.7f, 0.0f, 1.5f));

//Loop for column
for (int k = 0; k < 4; k++)
{
    //Ensure all rows begin with same positions
    xChange = new Vector3(4.5f, 0.0f, -10.0f);

    //Change z value dependant on column
    zChange = k * -5.0f;

    //Update z value of change
    xChange.Z += zChange;

    //Loop for row
    for (int i = k * 5; i < (k * 5) + 5; i++)
    {
        //Create new instance of enemy
        enemy[i] = new Enemy();

        //Set the enemy to alive
        enemy[i].isAliveBool = true;

        //Set enemy to not reached edge
        enemy[i].edgeCollision = false;

        //Define enemy position using values
        enemy[i].setPosition(xChange);
        //Define original position;

        enemy[i].setOriginalPos(xChange.X);

        //Make next enemy to the right of created one
        xChange.X += -2.2f;
    }
}

//Setup Laser Array
for (int i = 0; i < 10; i++)
{
    //Create new instance of Player Laser
    LaserArray[i] = new Laser();

    //Set as not active
    LaserArray[i].setActive(false);
}

//Setup Enemy Laser Array
for (int i = 0; i < 2; i++)
{
    //Create new instance of Enemy Laser
    enemyLaserArray[i] = new Laser();

    //Set as not active
    enemyLaserArray[i].setActive(false);
}
}

//Satellite position
sat.setPosition(new Vector3(-14.0f, -4.0f, -30.0f));

```

```

        //Moon position
        moon.setPosition(new Vector3(7.0f, 5.0f, -8.0f));
        blade.setPosition(new Vector3(0.0f, 0.0f, -3000.0f));
    }

    /**
     * Method: Load Content
     * For each menu screen, model and sound file that needs to be
loaded
     * For the 2D textures, load the respective 2D texture using the
directory within the brackets
     * For the 3D models, once loading the model, send the model
through to obtain the transformation
     * For the audio, load the respective audio file using the
directory within the brackets
     */

    protected override void LoadContent()
    {
        // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);

        //Menu
        bgMenu = Content.Load<Texture2D>("Menu\\background");
        startGame = Content.Load<Texture2D>("Menu\\startGameAI");
        exit = Content.Load<Texture2D>("Menu\\Exit");
        spacer = Content.Load<Texture2D>("Menu\\spacer");
        startGameOVER = Content.Load<Texture2D>("Menu\\startGameAI2");
        instructionsOVER =
Content.Load<Texture2D>("Menu\\Instructions2");
        exitOVER = Content.Load<Texture2D>("Menu\\Exit2");
        instructionsBG =
Content.Load<Texture2D>("Menu\\instructions_BG");

        //Game Over
        gameOver = Content.Load<Texture2D>("Menu\\gameOverScreen");

        //Win
        win = Content.Load<Texture2D>("Menu\\winScreen");

        //First person overlay
        firstPerson = Content.Load<Texture2D>("Menu\\firstPerson");

        //Instructions
        instructions = Content.Load<Texture2D>("Menu\\Instructions");
        instructionsBG2 =
Content.Load<Texture2D>("Menu\\instructions_BG2");
        instructionsBG3 =
Content.Load<Texture2D>("Menu\\instructions_BG3");

        //HUD
        hud = Content.Load<Texture2D>("HUD\\hud");

        //Player
        player.loadModel(Content.Load<Model>("Models\\jet"));

        player.modelTrans(SetupEffectTransformDefaults(player.getModel()));

        //Shield
        for (int k = 0; k < 2; k++)

```

```

    {
shieldArray[k].loadModel(Content.Load<Model>("Models\\shield_new"));

shieldArray[k].modelTrans(SetupEffectTransformDefaults(shieldArray[k].getModel()));
    }

    //Enemy
    for (int i = 0; i < 20; i++)
    {

enemy[i].loadModel(Content.Load<Model>("Models\\smallViper"));

enemy[i].modelTrans(SetupEffectTransformDefaults(enemy[i].getModel()));
    }

    //Laser
    for (int i = 0; i < 10; i++)
    {

LaserArray[i].loadModel(Content.Load<Model>("Models\\plaser"));

LaserArray[i].modelTrans(SetupEffectTransformDefaults(LaserArray[i].getModel()));
    }

    //Enemy Laser
    for (int i = 0; i < 2; i++)
    {

enemyLaserArray[i].loadModel(Content.Load<Model>("Models\\EnemyLaser"));

enemyLaserArray[i].modelTrans(SetupEffectTransformDefaults(enemyLaserArray[i].getModel()));
    }

    //Satellite
    sat.loadModel(Content.Load<Model>("Models\\cassini"));
    sat.modelTrans(SetupEffectTransformDefaults(sat.getModel()));

    //Moon
    moon.loadModel(Content.Load<Model>("Models\\Moon"));
    moon.modelTrans(SetupEffectTransformDefaults(moon.getModel()));

    //Blade
    blade.loadModel(Content.Load<Model>("Models\\prop"));

blade.modelTrans(SetupEffectTransformDefaults(blade.getModel()));

    //Font
    fontToUse = Content.Load<SpriteFont>("Fonts\\Army");

    //Audio
    themeMusic =
Content.Load<SoundEffect>("Audio\\Waves\\powerplant");
    playerFire =
Content.Load<SoundEffect>("Audio\\Waves\\playerfire");
    enemyFire =
Content.Load<SoundEffect>("Audio\\Waves\\enemyfire");

```

```

        explosion =
Content.Load<SoundEffect>("Audio\\Waves\\explosion");
        shieldDestroy =
Content.Load<SoundEffect>("Audio\\Waves\\shield_destroy");
    }

    /**
     * Method: Player Move
     * Depending on the input found, move the player to the correct
position
     * If camera2 is active, as its first person, update its position
to the player position
     */

private void playerMove()
{
    //Left
    if (getInput() == -1)
    {
        player.moveLeft();
        camera2.setCameraPosition(player.getPosition());
    }

    //Right
    if (getInput() == 1)
    {
        player.moveRight();
        camera2.setCameraPosition(player.getPosition());
    }

    //Forwards
    if (getInput() == 8)
    {
        player.moveForwards();
        camera2.setCameraPosition(player.getPosition());
    }

    //Backwards
    if (getInput() == 2)
    {
        player.moveBackwards();
        camera2.setCameraPosition(player.getPosition());
    }

    //Change Camera
    if ((getInput() == 20) && (getInput() != lastInput))
    {
        cameraActive = 1;
    }
    if ((getInput() == 30) && (getInput() != lastInput))
    {
        cameraActive = 2;
    }

    //Toggle Sound
    if ((getInput() == 12) && (getInput() != lastInput))
    {
        //If the sound is already on
        if (soundIsOn == true)
        {
            //Pause the theme music

```

```

        themeMusicInstance.Pause();

        //Set soundIsOn to false
        soundIsOn = false;
    }
    else
    {
        //Resume the theme music
        themeMusicInstance.Resume();

        //Set soundIsOn to true
        soundIsOn = true;
    }
}

//Shoot
if ((getInput() == 5) && (getInput() != lastInput))
{
    //For each of the Player Lasers
    for (int i = 0; i < 10; i++)
    {
        //If the Player Laser is not active
        if (LaserArray[i].isActive == false)
        {
            //Set the position of the laser to the current
            player position
            LaserArray[i].setPosition(player.getPosition());

            //Set the laser to active
            LaserArray[i].setActive(true);

            //If the sound is on
            if (soundIsOn == true)
            {
                //Play the playerFire sound
                playerFire.Play(0.4f, 0f, 0f, false);
            }
            break;
        }
    }
}
//Set the last input as the current input
lastInput = getInput();
}

/**
 * Method: Unload Content
 * Unload all the content
 */

protected override void UnloadContent() { }

/**
 * Method: Update
 * Allow the player to exit at any time
 * Switch the game state - Instructions, Menu and Exit
 *
 * -- Instructions
 * Within Instructions, swap the textures to display the relative
instruction
 *

```

```

* -- Game
* If starting the game for the first play play the music
* If retrying the game resume the music, and reset the values
* Await player movement
* Update enemy movement
* Check enemy status
* Fire enemy lasers
* Move enemy laser
* Move player laser
* Update ambient model position
* Check collision detection [Player Laser v Enemy Ship]
* Check collision detection [Enemy Laser v Shield]
* Check collision detection [Enemy Laser v Player]
* If the player is dead, show Game Over Screen
*
* -- Exit
* Exit - Exit the game
*/

```

```

protected override void Update(GameTime gameTime)
{
    //Set timeDelta to the total time
    float timeDelta = (float)gameTime.ElapsedGameTime.TotalSeconds;

    //Exit game at any time
    if (getInput() == 6)
    {
        this.Exit();
    }

    //Switch the gameState
    switch (gameState)
    {
        //Main Menu
        case 1:

            //If the instructions aren't shown
            if (showInstructions == 0)
            {
                //If Right has been pressed
                if ((getInput() == 1) && (getInput() != lastInput))
                {
                    //If the current menu selection plus one is
                    more than 3
                    if (menuSelection + 1 > 3)
                    {
                        //Set it to 3
                        menuSelection = 3;
                    }
                    //Otherwise
                    else
                    {
                        //Increase the menu selection
                        menuSelection++;
                    }
                }
                //If Left has been pressed
                if ((getInput() == -1) && (getInput() != lastInput))
                {
                    //If the menu selection minus one is less than

```

1

```

        if (menuSelection - 1 < 1)
        {
            //Set the menu selection to 1
            menuSelection = 1;
        }
        //Otherwise
        else
        {
            //Reduce the menu selection by 1
            menuSelection--;
        }
    }
}
//If either show instructions 1, 2 or 3 is displayed
if (showInstructions != 0)
{
    //If Left is pressed [There are only two options
possible]
    if ((getInput() == -1) && (getInput() != lastInput)
&& (showInstructions != 3))
    {
        //Display show instructions 1
        showInstructions = 1;
    }

    //If Right is pressed
    if ((getInput() == 1) && (getInput() != lastInput)
&& (showInstructions != 3))
    {
        //Display show instructions 2
        showInstructions = 2;
    }
}

//If Enter is pressed
if ((getInput() == 3) && (getInput() != lastInput))
{
    //If instructions are shown
    if (showInstructions != 0)
    {
        //Switch showInstructions
        switch (showInstructions)
        {
            //If the first show instructions is shown
[Back highlighted] then set instructions to not shown
            case 1:
                showInstructions = 0;
                break;

            //If the second instructions is shown
[Controls highlighted] display the controls
            case 2:
                showInstructions = 3;
                break;

            //If the controls instructions are shown,
show the instructions [Back highlighted]
            case 3:
                showInstructions = 1;
                break;
        }
    }
}

```

```

    }
    break;
}

//If the instructions aren't shown
if (showInstructions == 0)
{
    //Switch menuSelection
    switch (menuSelection)
    {
        //If "Start Game" is highlighted, begin
game
        case 1:
            gameState = 2;
            break;

        //If "Instructions" is highlighted, display
instructions [Back highlighted]
        case 2:
            showInstructions = 1;
            break;

        //If "Exit" is highlighted, exit the game
        case 3:
            this.Exit();
            break;
    }
}
break;

//Start Game
case 2:
    //If playone is false
    if (playOnce == false)
    {
        //Start the theme music
        themeMusicInstance = themeMusic.Play(1f, 0.0f, 0.0f,
true);

        //Set playOnce to true
        playOnce = true;
    }

    //If the player is retrying
    if (retrying == true)
    {
        //If the sound is on
        if (soundIsOn == true)
        {
            //Resume the music
            themeMusicInstance.Resume();
        }

        //Setup the models
        setupModels();

        //Set retrying to false
        retrying = false;

        //Set setup to false

```

```

        setup = false;
    }

    //Player Movement
    playerMove();

    //Enemy Movement
    for (int i = 0; i < 20; i++)
    {
        //If the enemy is alive
        if (enemy[i].isAliveBool == true)
        {
            //Check if its reached the edge of screen
            if (enemy[i].checkCollision() == true)
            {
                //Move forwards
                enemy[i].position.Z += 1.0f;

                //Change direction
                enemy[i].velocity.X *= -1.0f;

                //Set edge of collision to false
                enemy[i].setCollision(false);
            }
            //Update position
            enemy[i].position.X += enemy[i].velocity.X;

            //If the enemy has reached the end point
            if (enemy[i].position.Z > 0.0f)
            {
                //Set the game to game over
                gameState = 3;

                //Pause the theme music
                themeMusicInstance.Pause();
            }
        }
    }

    //Check if game has been won

    //Set count to 0
    int count = 0;

    //For all the enemies
    for (int i = 0; i < 20; i++)
    {
        //If the enemy is dead
        if (enemy[i].isAliveBool == false)
        {
            //Increase the count
            count++;

            //If the count is 19 [starts at 0]
            if (count == 19)
            {
                //Then player has won
                gameState = 4;

                //Pause the music
                themeMusicInstance.Pause();
            }
        }
    }
}

```

```

    }
}

//Set the previous laser to 21 - an unattainable number
int previousLaser = 21;

//Fire enemy lasers

//If setup is false
if (setup == false)
{
    //For each of the enemy laser
    for (int i = 0; i < 2; i++)
    {
        //For all of the enemies
        for (int k = 0; k < 20; k++)
        {
            //If the enemy is alive and doesnt equal
            the previous enemy
            if ((enemy[k].isAliveBool == true) &&
            (previousLaser != k))
            {
                //Set the enemy laser position to the
                enemy position
                enemyLaserArray[i].setPosition(enemy[k].getPosition());

                //Set the laser to active
                enemyLaserArray[i].setActive(true);

                //If the sound is on
                if (soundIsOn == true)
                {
                    //Play the enemy fire sound
                    enemyFire.Play(0.4f, 0f, 0f, false);
                }

                //Set the previous laser to the current
                enemy
                previousLaser = k;
                break;
            }
        }
    }

    //Set setup to true
    setup = true;
}

//Move Enemy Laser
for (int i = 0; i < 2; i++)
{
    //If the enemy laser is active
    if (enemyLaserArray[i].isActive == true)
    {

        //If the enemy laser has gone out of the screen
        if (enemyLaserArray[i].checkOutOfScreen(2) ==
true)
        {

```

```

        //Set the enemy laser to not active
        enemyLaserArray[i].setActive(false);

        //Set setup to false
        setup = false;
    }

    //Otherwise update the enemy laser position
    enemyLaserArray[i].Update(timeDelta, 2);
}
}

//Move Laser
for (int i = 0; i < 10; i++)
{
    //If the player laser is active
    if (LaserArray[i].isActive == true)
    {
        //If the player laser is out of the screen
        if (LaserArray[i].checkOutOfScreen(1) == true)
        {
            //Set the laser to not active
            LaserArray[i].setActive(false);
        }

        //Otherwise, update the laser position
        LaserArray[i].Update(timeDelta, 1);
    }
}

//Move Satellite
Vector3 satPos = sat.getPosition();

//Set the increment on the X Position
satPos.X += 0.015f;

//Set the increment on the Y position
satPos.Y -= 0.01f;

//Update the position
sat.setPosition(satPos);

//Move moon
Vector3 spin = moon.getPosition();

//Set the increment on the X Position
spin.X -= 0.005f;

//Set the increment on the Y Position
spin.Y += 0.002f;

//Set the increment on the Z Position
spin.Z += 0.003f;

//Update the moon position
moon.setPosition(spin);

//Collision Detection
//Compares each enemy with all lasers before moving to
next enemy
for (int i = 0; i < 20; i++)

```

```

    {
        //If the enemy is alive
        if (enemy[i].isAliveBool == true)
        {
            //Set the bounding sphere for the enemy
            BoundingBox enemyBounding = new
BoundingBox(enemy[i].getPosition(),
enemy[i].enemyModel.Meshes[0].BoundingBox.Radius * 0.09f);

            //For each of the player's laser
            for (int k = 0; k < 10; k++)
            {
                //If the laser is active
                if (LaserArray[k].isActive == true)
                {
                    //Set the bounding sphere for the laser
                    BoundingBox laserSphere = new
BoundingBox(LaserArray[k].getPosition(),
LaserArray[k].pLaserModel.Meshes[0].BoundingBox.Radius * 0.006f);

                    //If Player Laser hits Enemy
                    if
(enemyBounding.Intersects(laserSphere))
                    {
                        //If the sound is on
                        if (soundIsOn == true)
                        {
                            //Play the explosion sound
                            explosion.Play(0.4f, 0f, 0f,
false);
                        }

                        //Set the enemy to not active
                        enemy[i].isAliveBool = false;

                        //Set the player laser to not
                        active
                        LaserArray[k].isActive = false;

                        //Increase the score by 20
                        score += 20;
                        break;
                    }
                }
            }
        }
    }

    //Player and Shield Collision Bounding Sphere
    BoundingBox playerBounding = new
BoundingBox(player.getPosition(),
player.playerModel.Meshes[0].BoundingBox.Radius * 1.2f);

    //For each enemy laser active
    for (int i = 0; i < 2; i++)
    {
        //If the laser is active
        if (enemyLaserArray[i].isActive == true)
        {
            //Set the laser's bounding sphere

```

```

        BoundingBox enemyLaserBounding = new
BoundingBox(enemyLaserArray[i].getPosition(),
enemyLaserArray[i].pLaserModel.Meshes[0].BoundingBox.Radius * 0.006f);

        //For each shield
for (int k = 0; k < 2; k++)
{
    //If the shield is active
    if (shieldArray[k].isActive == true)
    {
        //Set the shield's bounding sphere
        BoundingBox shieldBounding = new
BoundingBox(shieldArray[k].getPosition(),
shieldArray[k].shieldModel.Meshes[0].BoundingBox.Radius * 0.02f);

        //If the shield intersects the laser
        if
(shieldBounding.Intersects(enemyLaserBounding))
        {
            //If the sound is on
            if (soundIsOn == true)
            {
                //Play the explosion sound
                shieldDestroy.Play(0.4f, 0f, 0f,
false);
            }
            //Set the laser as inactive
            enemyLaserArray[i].isActive = false;

            //Reduce the lasers health by half
            shieldArray[k].hit();

            //Check if the shield has been
            destroyed
            shieldArray[k].checkShieldStatus();

            //If both lasers are inactive,
            setup is true, setting up new lasers
            if ((enemyLaserArray[0].isActive ==
false) && (enemyLaserArray[1].isActive == false))
            {
                setup = false;
            }
            break;
        }
        //If the player intersects with the
        enemy laser
        if
(playerBounding.Intersects(enemyLaserBounding))
        {
            //If the sound is on
            if (soundIsOn == true)
            {
                //Play the shieldDestroy sound
                shieldDestroy.Play(0.4f, 0f, 0f,
false);
            }
            //Set the enemy laser to not active
            enemyLaserArray[i].setActive(false);

            //Reduce the players health

```

```

        health -= 25;
        //If both the enemy lasers are not
active
        if ((enemyLaserArray[0].isActive ==
false) && (enemyLaserArray[1].isActive == false))
        {
            //Set setup to false
            setup = false;
        }
        break;
    }
}
}
}
}
}
//If the player's health is less than 1
if (health < 1)
{
    //Game over
    gameState = 3;

    //Pause the theme music
    themeMusicInstance.Pause();
}
break;
//Game Over
case 3:
    //If enter is pressed
    if (getInput() == 3)
    {
        //Set the game state to menu
        gameState = 1;
        //Set the menu selection to "Start Game"
        menuSelection = 1;
        //Do not show instructions
        showInstructions = 0;
        //Set retrying to true
        retrying = true;
    }
    break;
//Win
case 4:
    //If enter is pressed
    if (getInput() == 3)
    {
        //Set the game state to menu
        gameState = 1;
        //Set the menu selection to "Start Game"
        menuSelection = 1;
        //Do not show instructions
        showInstructions = 0;

        //Set retrying to true
        retrying = true;
    }
    break;
}
//Set the last input to the current input
lastInput = getInput();
base.Update(gameTime);
}

```

```

/**
 * Method: Draw
 * Switch the game state - Menu Screen, Game, Game Over and Win
Screen
 *
 * -- Menu Screen
 * Show default background with three options - Start Game,
Instructions and Exit
 * If the menu item is selected, show the relevant image [Selected
text in red]
 *
 * -- Instructions
 * Three different images for showing instructions
 * [First page::Back highlighted, First page::Controls highlighted,
Second page::Back highlighted]
 * Switch between them depending on input
 *
 * -- Game
 * Set the background to Black
 * Render Enemy Lasers that are active
 * Render the Player
 * Render the Shields that are active
 * Render the Enemies that are active
 * Render the Player Lasers that are active
 * Render Satellite
 * Render Moon
 * Render HUD
 * Print Health and Score text to screen
 *
 * -- Game Over
 * Show 2D Texture that shows game over
 * Print score text to screen
 *
 * -- Win
 * Show 2D Texture that shows game won state
 * Update score variable - multiply health remaining by score
 * Print score on text
 */
protected override void Draw(GameTime gameTime)
{
    //Set window title
    Window.Title = "El Space Aventura!";
    //Display menu
    if (gameState == 1)
    {
        spriteBatch.Begin(SpriteBlendMode.AlphaBlend,
SpriteSortMode.Deferred, SaveStateMode.SaveState);

        //If instructions aren't shown
        if (showInstructions == 0)
        {
            //Draw menu background
            spriteBatch.Draw(bgMenu, new Rectangle(0,
Window.ClientBounds.Height - bgMenu.Height, bgMenu.Width, bgMenu.Height),
Color.White);

            //If first item is highlighted
            if (menuSelection == 1)
            {
                //Draw "Start Game" in red

```

```

        spriteBatch.Draw(startGameOVER, new Rectangle(201,
300, startGameOVER.Width, startGameOVER.Height), Color.White);
    }
    else
    {
        //Draw "Start Game" in white
        spriteBatch.Draw(startGame, new Rectangle(235, 295,
startGame.Width, startGame.Height), Color.White);
    }

    //Draw spacer graphic
    spriteBatch.Draw(spacer, new Rectangle(295, 322,
spacer.Width, spacer.Height), Color.White);

    //If second item is highlighted
    if (menuSelection == 2)
    {
        //Draw "Instructions" in red
        spriteBatch.Draw(instructionsOVER, new
Rectangle(390, 300, instructionsOVER.Width, instructionsOVER.Height),
Color.White);
    }
    else
    {
        //Draw "Instructions" in white
        spriteBatch.Draw(instructions, new Rectangle(390,
300, instructions.Width, instructions.Height), Color.White);
    }
    //Draw spacer graphic
    spriteBatch.Draw(spacer, new Rectangle(455, 321,
spacer.Width, spacer.Height), Color.White);

    //If third item is highlighted
    if (menuSelection == 3)
    {
        //Draw "Exit" in red
        spriteBatch.Draw(exitOVER, new Rectangle(545, 300,
exitOVER.Width, exitOVER.Height), Color.White);
    }
    else
    {
        //Draw "Exit" in white
        spriteBatch.Draw(exit, new Rectangle(545, 300,
exit.Width, exit.Height), Color.White);
    }
}

//If instructions are shown [Back highlighted]
if (showInstructions == 1)
{
    //Draw instructions
    spriteBatch.Draw(instructionsBG, new Rectangle(0,
Window.ClientBounds.Height - instructionsBG.Height, instructionsBG.Width,
instructionsBG.Height), Color.White);
}
//If instructions are shown [Controls highlighted]
if (showInstructions == 2)
{
    //Draw instructions 2

```

```

        spriteBatch.Draw(instructionsBG2, new Rectangle(0,
Window.ClientBounds.Height - instructionsBG2.Height, instructionsBG2.Width,
instructionsBG2.Height), Color.White);
    }

    //If controls are shown
    if (showInstructions == 3)
    {
        //Draw controls
        spriteBatch.Draw(instructionsBG3, new Rectangle(0,
Window.ClientBounds.Height - instructionsBG3.Height, instructionsBG3.Width,
instructionsBG3.Height), Color.White);
    }
    spriteBatch.End();
}
//Display game
if (gameState == 2)
{
    //Set background to black
    GraphicsDevice.Clear(Color.Black);
    //Render Enemy Laser Array
    for (int i = 0; i < 2; i++)
    {
        //If the enemyLaser is active
        if (enemyLaserArray[i].isActive == true)
        {
            //Set transform and draw model
            Matrix ELaserTransform =
Matrix.CreateTranslation(enemyLaserArray[i].getPosition());
            DrawModel(enemyLaserArray[i].getModel(),
ELaserTransform, enemyLaserArray[i].getTrans());
        }
    }
    //Render Player
    Matrix modelTransform =
Matrix.CreateTranslation(player.getPosition());
    DrawModel(player.getModel(), modelTransform,
player.getTrans());
    //Render Shield
    for (int k = 0; k < 2; k++)
    {
        //If the shield is active
        if (shieldArray[k].isActive == true)
        {
            //Set shield transform and draw model
            Matrix shieldTrans =
Matrix.CreateTranslation(shieldArray[k].getPosition());
            DrawModel(shieldArray[k].getModel(), shieldTrans,
shieldArray[k].getTrans());
        }
    }
    //Render Enemy
    for (int i = 0; i < 20; i++)
    {
        //If the enemy is active
        if (enemy[i].isAliveBool == true)
        {
            //Set enemy transform and draw model
            Matrix enemyTransform =
Matrix.CreateRotationY(enemy[i].getRotation()) *
Matrix.CreateTranslation(enemy[i].getPosition());

```

```

        DrawModel(enemy[i].getModel(), enemyTransform,
enemy[i].getTrans());
    }
}
//Render LaserArray
for (int i = 0; i < 10; i++)
{
    //If the Player Laser is active
    if (LaserArray[i].isActive == true)
    {
        //Set Player Laser transform and draw model
        Matrix LaserTransform =
Matrix.CreateTranslation(LaserArray[i].getPosition());
        DrawModel(LaserArray[i].getModel(), LaserTransform,
LaserArray[i].getTrans());
    }
}
//Render Satellite
Matrix satTransform =
Matrix.CreateTranslation(sat.getPosition());
DrawModel(sat.getModel(), satTransform, sat.getTrans());
//Render Moon
Matrix moonTransform =
Matrix.CreateTranslation(moon.getPosition()) * Matrix.CreateScale(0.5f);
DrawModel(moon.getModel(), moonTransform, moon.getTrans());
//Render blade
Matrix bladeTransform =
Matrix.CreateTranslation(blade.getPosition()) * Matrix.CreateScale(0.0125f);
DrawModel(blade.getModel(), bladeTransform,
blade.getTrans());
//Begin rendering of 2D Textures
spriteBatch.Begin(SpriteBlendMode.AlphaBlend,
SpriteSortMode.Deferred, SaveStateMode.SaveState);
//If the first person camera is active
if (cameraActive == 2)
{
    //Draw the first person HUD
    spriteBatch.Draw(firstPerson, new Rectangle(0,
Window.ClientBounds.Height - firstPerson.Height, firstPerson.Width,
firstPerson.Height), Color.White);
}
//Draw the HUD
spriteBatch.Draw(hud, new Rectangle(0,
Window.ClientBounds.Height - hud.Height, hud.Width, hud.Height),
Color.White);
//End of 2D Texturing
spriteBatch.End();
//Print Text
writeText(health + "%", new Vector2(35, 573), Color.White);
writeText(score.ToString(), new Vector2(730, 573),
Color.White);
}
//Display Game Over
if ((gameState == 3) || (gameState == 4))
{
    //Begin rendering of 2D Textures
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend,
SpriteSortMode.Deferred, SaveStateMode.SaveState);

    //If gameover
    if (gameState == 3)

```

```

        {
            //Draw game over background
            spriteBatch.Draw(gameOver, new Rectangle(0,
Window.ClientBounds.Height - gameOver.Height, gameOver.Width,
gameOver.Height), Color.White);
            //Set scoreWin to score value
            scoreWin = score;
        }
        //If win
        if (gameState == 4)
        {
            //Draw win background
            spriteBatch.Draw(win, new Rectangle(0,
Window.ClientBounds.Height - win.Height, win.Width, win.Height),
Color.White);

            //Score is score multiplied by health
            scoreWin = score * health;
        }
        //Set the score to a string
        string scoreText = scoreWin.ToString();
        //Find the centre of the string
        Vector2 FontOrigin = fontToUse.MeasureString(scoreText) / 2;
        //Set string position
        Vector2 FontPosition = new Vector2(450, 305);

        //Print the score, using the decalred font, string,
position and colour of the text
        spriteBatch.DrawString(fontToUse, scoreText, FontPosition,
Color.White);

        //End of 2D Texturing
        spriteBatch.End();
    }
    base.Draw(gameTime);
}
/**
 * Method: Get Input
 * Set variable keyboardState to get the keyboard input
 *
 * [Return Value] :: Function - Input Keyboard // 360 Input
 *
 * [1] :: Right - Right // Left Stick pressed right
 * [-1] :: Left - Left // Left Stick pressed left
 * [8] :: Forwards - Up // Left Stick pressed up
 * [2] :: Backwards - Down // Left Stick pressed down
 * [3] :: Confirm - Enter // Button A
 * [6] :: Exit - Escape // Button Back
 * [5] :: Fire - Space // Right Shoulder
 * [20] :: Camera 1 - No. 1 // Left Trigger
 * [30] :: Camera 2 - No. 2 // Right Trigger
 * [12] :: Sound - S // Button X
 *
 */
public int getInput()
{
    //Obtain the keyboards input
    KeyboardState keyboardState = Keyboard.GetState();
    //If the left stick is pressed to the right
    if (GamePad.GetState(PlayerIndex.One).ThumbSticks.Left.X > 0.7f)
    {
        return 1;
    }
}

```

```

    }
    //If the right arrow key is pressed
    if (keyboardState.IsKeyDown(Keys.Right))
    {
        return 1;
    }
    //If the left stick is pressed to the left
    if (GamePad.GetState(PlayerIndex.One).ThumbSticks.Left.X < -
0.7f)
    {
        return -1;
    }
    //If the left arrow key is pressed
    if (keyboardState.IsKeyDown(Keys.Left))
    {
        return -1;
    }

    //If the left stick is pressed up
    if (GamePad.GetState(PlayerIndex.One).ThumbSticks.Left.Y > 0.7f)
    {
        return 8;
    }
    //If the up arrow key is pressed
    if (keyboardState.IsKeyDown(Keys.Up))
    {
        return 8;
    }
    //If the left stick is pressed down
    if (GamePad.GetState(PlayerIndex.One).ThumbSticks.Left.Y < -
0.7f)
    {
        return 2;
    }
    //If the down arrow key is pressed
    if (keyboardState.IsKeyDown(Keys.Down))
    {
        return 2;
    }
    //If the button "A" is pressed
    if (GamePad.GetState(PlayerIndex.One).Buttons.A ==
ButtonState.Pressed)
    {
        return 3;
    }
    //If the enter key is pressed
    if (keyboardState.IsKeyDown(Keys.Enter))
    {
        return 3;
    }
    //If the button "Back" is pressed
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed)
    {
        return 6;
    }
    //If the escape key is pressed
    if (keyboardState.IsKeyDown(Keys.Escape))
    {
        return 6;
    }
}

```

```

        //If the Right Shoulder button is pressed
        if (GamePad.GetState(PlayerIndex.One).Buttons.RightShoulder ==
ButtonState.Pressed)
        {
            return 5;
        }
        //If the spacebar is pressed
        if (keyboardState.IsKeyDown(Keys.Space))
        {
            return 5;
        }
        //If the Left Trigger is depressed
        if (GamePad.GetState(PlayerIndex.One).Triggers.Left > 0.7f)
        {
            return 20;
        }
        //If the number 1 key is pressed
        if (keyboardState.IsKeyDown(Keys.D1))
        {
            return 20;
        }
        //If the Right Trigger is depressed
        if (GamePad.GetState(PlayerIndex.One).Triggers.Right > 0.7f)
        {
            return 30;
        }
        //If the number 2 key is pressed
        if (keyboardState.IsKeyDown(Keys.D2))
        {
            return 30;
        }
        //If the button "X" is pressed
        if (GamePad.GetState(PlayerIndex.One).Buttons.X ==
ButtonState.Pressed)
        {
            return 12;
        }
        //If the "S" key is pressed
        if (keyboardState.IsKeyDown(Keys.S))
        {
            return 12;
        }
        //Otherwise
        return 0;
    }
}
}

```

Code :: ambientModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace Lab6CollisionDetectionV3
{
    class ambientModel
    {
        public Model ambientModelM;
        public Matrix[] ambientTrans;
        public Vector3 position;
        //Constructor for ambient model
        public ambientModel()
        {
        }
        //Load the ambient model
        public void loadModel(Model mdl)
        {
            ambientModelM = mdl;
        }
        //Set ambient model transformation
        public void modelTrans(Matrix[] mdlTrans)
        {
            ambientTrans = mdlTrans;
        }
        //Get enemy transformation
        public Matrix[] getTrans()
        {
            return ambientTrans;
        }
        //Get the ambient model model
        public Model getModel()
        {
            return ambientModelM;
        }
        //Set ambient model position
        public void setPosition(Vector3 sP)
        {
            position = sP;
        }
        //Get ambient model position
        public Vector3 getPosition()
        {
            return position;
        }
    }
}
```

Code :: Camera.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace Lab6CollisionDetectionV3
{
    public class Camera
    {
        private Matrix viewMatrix;
        private Matrix projectionMatrix;
        private Matrix worldMatrix;
        private Vector3 camLookTarget;
        //Set the view matrix
        public void setViewMatrix(Vector3 position, Vector3 target)
        {
            //Obtain camera look at target
            camLookTarget = target;
            //Create View Matrix - Position / Camera Target / Vector 3 UP
            viewMatrix = Matrix.CreateLookAt(position, target, Vector3.Up);
        }
        //Set projection matrix
        public void setProjectionMatrix(float aR)
        {
            //Field of view / Aspect Ratio / Near Clipping Plane / Far
            Clipping Plane
            projectionMatrix =
            Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(45), aR, 1.0f,
            1000.0f);
            //World matrix
            worldMatrix = Matrix.Identity;
        }
        //Set camera position (for first person view)
        public void setCameraPosition(Vector3 playerPosition)
        {
            //Set the look at target to the players position, always down
            the same line
            camLookTarget.X = playerPosition.X;
            //Set the camera's y position to zero
            camLookTarget.Y = 0f;
            //Recreate the view matrix
            viewMatrix = Matrix.CreateLookAt(playerPosition, camLookTarget,
            Vector3.Up);
        }
        //Get the view matrix
        public Matrix getViewMatrix()
        {
            return viewMatrix;
        }
        //Get the projection matrix
        public Matrix getProjectionMatrix()
        {

```

```
        return projectionMatrix;
    }
    //Get the world matrix
    public Matrix getWorldMatrix()
    {
        return worldMatrix;
    }
}
```

Code :: Enemy.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace Lab6CollisionDetectionV3
{
    class Enemy
    {
        public Model enemyModel;
        public Matrix[] enemyTrans;
        public Vector3 position;
        public float rotation;
        public bool isAliveBool;
        private float originalPos;
        private float nextPosR;
        private float nextPosL;
        public Vector3 velocity;
        public bool edgeCollision;
        //Constructor of enemy class
        public Enemy()
        {
            //Rotate the model to face the correct way
            rotation = 1.55f;

            //Declare velocity values
            velocity.X = 0.0625f;
            velocity.Z = 0.125f;
            velocity.Y = 0.0f;
        }
        //Check if the enemy has hit the side of the screen
        public bool checkCollision()
        {
            //Calculate the next position
            nextPosR = originalPos + 5.0f;
            nextPosL = originalPos - 5.0f;
            //If the next position is greater than 5 units plus the
original position
            if (position.X > nextPosR)
            {
                //Collision occurred
                edgeCollision = true;
                return edgeCollision;
            }
            //If the next position is less than 5 units minus the original
position
            if (position.X < nextPosL)
            {
                //Collision occurred
                edgeCollision = true;
                return edgeCollision;
            }
        }
    }
}
```

```

        return false;
    }
    //Pass in value to set value of edge collision
    public void setCollision(bool input)
    {
        edgeCollision = input;
    }
    //Pass in value of original position
    public void setOriginalPos(float input)
    {
        originalPos = input;
    }
    //Get original position
    public float getOriginalPos()
    {
        return originalPos;
    }
    //Check to see if the enemy is alive
    public Boolean isAlive()
    {
        return isAliveBool;
    }
    //Get the enemy speed
    public Vector3 getSpeed()
    {
        return velocity;
    }
    //Load enemy model
    public void loadModel(Model mdl)
    {
        enemyModel = mdl;
    }
    //Transform enemy model
    public void modelTrans(Matrix[] mdlTrans)
    {
        enemyTrans = mdlTrans;
    }
    //Get enemy transformation
    public Matrix[] getTrans()
    {
        return enemyTrans;
    }
    //Get enemy model
    public Model getModel()
    {
        return enemyModel;
    }

    //Set enemy position
    public void setPosition(Vector3 sP)
    {
        position = sP;
    }
    //Get enemy position
    public Vector3 getPosition()
    {
        return position;
    }
    //Move enemy left
    public void moveLeft(Vector3 move)
    {

```

```
        position += move;
    }
    //Set enemy rotation
    public void setRotation(float sR)
    {
        rotation = sR;
    }
    //Get enemy rotation
    public float getRotation()
    {
        return rotation;
    }
}
}
```

Code :: laserClass.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace Lab6CollisionDetectionV3
{
    class Laser
    {
        public Model pLaserModel;
        public Matrix[] pLaserTrans;
        public Vector3 pLaserPos;
        public float speed;
        public bool isActive;
        //Laser constructor - set the speed value
        public Laser(){
            speed = 6.5f;
        }
        //Load laser model
        public void loadModel(Model mdl)
        {
            pLaserModel = mdl;
        }
        //Set laser transformation
        public void modelTrans(Matrix[] mdlTrans)
        {
            pLaserTrans = mdlTrans;
        }
        //Get laser transformation
        public Matrix[] getTrans()
        {
            return pLaserTrans;
        }
        //Get the laser model
        public Model getModel()
        {
            return pLaserModel;
        }
        //Set laser position
        public void setPosition(Vector3 sP)
        {
            pLaserPos = sP;
        }

        //Get laser position
        public Vector3 getPosition()
        {
            return pLaserPos;
        }
        //Check if the laser is active
        public bool checkActive()
        {

```

```

        return isActive;
    }
    //Set the laser to active / inactive
    public void setActive(bool input)
    {
        isActive = input;
    }
    //Update the laser position - depending on the enemy/player laser
[different z directions]
    public void Update(float delta, int laserType)
    {
        //If the laser belongs to player
        if (laserType == 1)
        {
            pLaserPos.Z -= speed * delta;
        }

        //If the laser belongs to the enemy
        if (laserType == 2)
        {
            pLaserPos.Z += speed * delta;
        }
    }
    //Check if the laser has gone out of the screen
    public bool checkOutOfScreen(int laserType)
    {
        //If the laser is the player's
        if (laserType == 1)
        {
            //If it has gone past the -28 units
            if (pLaserPos.Z < -28.0f)
            {
                //Return true
                return true;
            }
            //Otherwise false
            return false;
        }
        //If the laser is the enemies
        if (laserType == 2)
        {
            //If it is greater than 3 units
            if (pLaserPos.Z > 3.0f)
            {
                //Return true
                return true;
            }
            //Otherwise return false
            return false;
        }

        //Return false - necessary for type bool function
        return false;
    }
}
}
}

```

Code :: Player.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace Lab6CollisionDetectionV3
{
    public class Player
    {
        public Model playerModel;
        public Matrix[] playerTrans;
        public Vector3 playerPosition;
        //Constructor - set player default position
        public Player()
        {
            playerPosition = new Vector3(0.0f, 0.0f, 0.0f);
        }
        //Load the player model
        public void loadModel(Model mdl)
        {
            playerModel = mdl;
        }
        //Set the player transformation
        public void modelTrans(Matrix[] mdlTrans)
        {
            playerTrans = mdlTrans;
        }
        //Get the player transformation
        public Matrix[] getTrans()
        {
            return playerTrans;
        }
        //Get the player model
        public Model getModel()
        {
            return playerModel;
        }
        //Set the player position
        public void setPosition(Vector3 sP)
        {
            playerPosition = sP;
        }
        //Get the player position
        public Vector3 getPosition()
        {
            return playerPosition;
        }
        //Move the player left
        public void moveLeft()
        {
            //If the player position is less than -3.4 units
            if(playerPosition.X < -3.4f)
```

```

    {
        //Set position to -4 units
        playerPosition.X = -4.0f;
    }
    //Otherwise
    else
    {
        //Reduce the position by half a unit
        playerPosition.X -= 0.5f;
    }
}
//Move the player right
public void moveRight()
{
    //If the player position is greater than 3.4 units
    if (playerPosition.X > 3.4f)
    {
        //Set the position to 4 units
        playerPosition.X = 4.0f;
    }
    //Otherwise
    else
    {
        //Increase the position by half a unit
        playerPosition.X += 0.5f;
    }
}
//Move the player forwards
public void moveForwards()
{
    //If the player position is less than -3.4 units
    if (playerPosition.Z < -1.9f)
    {
        //Set the position to -2 units
        playerPosition.Z = -2.0f;
    }
    //Otherwise
    else
    {
        //Increase the position by half a unit
        playerPosition.Z += -0.5f;
    }
}
//Move the player backwards
public void moveBackwards()
{
    //If the player position is greater than 3.9 units
    if (playerPosition.Z > 3.9f)
    {
        //Set the player position to 4
        playerPosition.Z = 4.0f;
    }
    //Otherwise
    else
    {
        //Increase the position by half a unit
        playerPosition.Z += 0.5f;
    }
}
}
}
}

```

Code :: Shield.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace Lab6CollisionDetectionV3
{
    class Shield
    {
        public Model shieldModel;
        public Matrix[] shieldTrans;
        public Vector3 position;
        public float rotation;
        public int shieldHealth;
        public bool isActive;
        //Shield constructor
        public Shield()
        {
            //Set shield to 150
            shieldHealth = 150;
            //Set it to active
            isActive = true;
        }
        //If the shield has been hit
        public void hit()
        {
            //Reduce by 50
            shieldHealth = shieldHealth - 50;
        }
        //Load shield model
        public void loadModel(Model mdl)
        {
            shieldModel = mdl;
        }
        //Transform shield model
        public void modelTrans(Matrix[] mdlTrans)
        {
            shieldTrans = mdlTrans;
        }
        //Get transformation
        public Matrix[] getTrans()
        {
            return shieldTrans;
        }
        //Get model
        public Model getModel()
        {
            return shieldModel;
        }
        //Set shield position
    }
}
```

```

public void setPosition(Vector3 sP)
{
    position = sP;
}
//Get shield position
public Vector3 getPosition()
{
    return position;
}
//Set shield rotation
public void setRotation(float sR)
{
    rotation = sR;
}
//Get shield rotation
public float getRotation()
{
    return rotation;
}
//Check if the shield is still alive
public void checkShieldStatus()
{
    //If the shield health is less than 1
    if (shieldHealth < 1)
    {
        //Set active to false
        isActive = false;
    }
}
}
}

```