

**Games Programming 3**

(MHG605291)

**Coursework Documentation**

**Laurie Green**

S0907727

**BSc Computer Games (Software Development)**

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

# An explanation of the code used to generate the game

The code used to create this project came from varied sources. While some tasks were straight forward and used techniques that I was familiar with, others were more complex and required more work. This section will highlight areas which are worth discussing in more detail, the attached code is heavily commented which will explain the code on a line by line basis.

## Camera Class

The camera class was required so I could manipulate the camera to make it move around the scene. The camera class required a few things such as its own position, yaw and pitch values, view, projection and world matrices and a point to look at. I also gave my camera class several methods which would be called each frame to respond to certain things such as user input or a change in its position or lookat. In the final version I ended up not using the yaw and pitch values since I was not rotating the camera in any situation.

### View, Projection and World Matrices

These matrices are required to represent a digital 3D world, each ones plays a role in building the world.   
**View Matrix**The view matrix is used to define the position and orientation of the camera, what is visible from our current position and angle?  
**World Matrix**The world matrix is used to determine where the mesh will be drawn in the 3D space, where are we drawing according to the world’s coordinate system?  
**Projection Matrix**The projection matrix is used to translate the coordinates from 3D space into screen coordinates, also used to set whether orthographic projection will be used or perspective projection.

I attached these matrices to the camera so keep all the similar matrices attached to the same object.

### Camera behaviour method

I tackled the task of creating different camera’s by implementing a simple Boolean based system where, depending on the states of a set of three Booleans, different camera behaviour would be passed back into the main class. The camera’s I decided to use where a Third Person, First Person, and a Top Down camera.   
Depending on the users input (1, 2 or 3) the camera would be given different values for its position and lookat.

### Lack of camera rotation

It was intention from the beginning to have a proper third person camera which rotated around the player as the player turned. However after spending a significant amount of time trying to work out the maths required for a rotation around a point I decided that my time was better spent developing a terrain, which I felt would make the project seem more complete. Given more time I would definitely include a third person camera, I got fairly close on several occasions but often ended up in a mess and having to start again.

## Terrain Class

The terrain class was required to spawn a landscape in the 3D world. It used a simple grey-scale heightmap to take the height values and then, using these values it would create a mesh and apply a texture. I used an online tutorial to help me integrate the system into my project but I then adapted it by, using alternate heightmaps and textures to create more varied landscapes.

### Height Map

A heightmap is simply an image which uses a simple system of colour to feed information into the project. It supplies the data by using pixels from black to white (and all the grey in-between) to demonstrate the height of a particular point; typically the whiter a pixel, the higher the point in space. Figure 1 is an example of a typical heightmap.



Figure 1

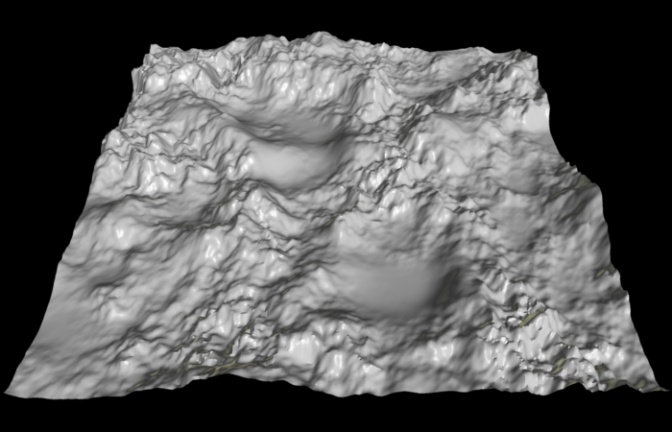


Figure 2

Each pixel in the image then become a point the in the 3D world and using this information a mesh of indices and vertices is constructed using the heightmap as a guide. Figure 2 is the same heightmap converted into terrain.

### Collision Detection

It was my intention to add collision detection between the terrain and the other objects in the game world. However, given the time constraints, and as this was not a priority since I had already implemented collision detection between the player, the enemies and the lasers, I felt my time would be better spent on other areas of the project.

## Menu class

To draw the menu’s I opted to use a simple solution which I found in a tutorial online. The technique was to simply use states to track what the user wished to do (browse menus or play the game) depending on the state being used the draw and update functions would behave differently. For example, if the user was in the menu state then instead of drawing all the models and terrain, the project would simply draw an image with text written across it. I felt this basic menu would suffice for the needs of the project; it allowed the user to view the instructions and controls and gave them the option to quit or play the game. The system also allowed for the user to be presented with a “game over” screen when he/she ran out of lives.

Using the XNA spritebatch method, this solution was quick to setup and easy to manipulate. However given more time I would have implemented the XNA game-state-manager system which is a much “fuller” solution providing a diverse selection of options for customization and offers efficient and clean content management.

## Skybox

To create the skybox, I followed a simple tutorial online which I then adapted to fit my individual needs. I created an fx file which would be accessed via the main class and applied to a skybox texture. The fx file would map the texture to certain areas of a simple cube model which would then be placed around my entire world to give the impression of a landscape in the distance.

## Player, Daleks and Lasers

These three classes were all created since they all have different attributes they couldn’t all just be instances of a simple gameobject class. Player requires some sound effects, an update method and a input checker. In comparison, the Dalek and Laser classes are much simpler as they only need positions, directions and a simple update method to track their movement and remove inactive instances. Since these classes are heavily commented and fairly simple, not much more discussion is needed to explain techniques.

# Appendix A – Fully Commented 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;

namespace CourseworkLGreen

{

/// <summary>

/// Laurie Green 11/12

/// Main Class for Games Programming 3 Coursework

/// List of all tutorials and sources used:

/// http://msdn.microsoft.com/en-us/library/bb197293(v=xnagamestudio.31).aspx

/// http://rbwhitaker.wikidot.com/skyboxes-1

/// http://soundbible.com/1949-Pew-Pew.html

/// http://soundbible.com/1461-Big-Bomb.html

/// http://en.wikibooks.org/wiki/Game\_Creation\_with\_XNA/3D\_Development/Landscape\_Modelling

/// </summary>

public class Game1 : Microsoft.Xna.Framework.Game

{

public static GameStates gamestate;

// Instance of menu

private Menu menu;

// Instance of player

private Player player;

// Instance of skybox

private Skybox skybox;

// Instance of cam

private Camera cam;

// Instance of terrain

private Terrain landscape2;

// Keyboard states for input

KeyboardState keyboardState;

KeyboardState lastState;

// Graphics Device

GraphicsDeviceManager graphics;

// Spritebatch for drawing

SpriteBatch spriteBatch;

// Font

SpriteFont fontToUse;

// Creates an array of enemies

private Model mdlDalek;

private Matrix[] mdDalekTransforms;

private Daleks[] dalekList = new Daleks[GameConstants.NumDaleks];

public int Daleknum = 0;

// Creates an array of enemies

private Model mdlLaser;

private Matrix[] mdlLaserTransforms;

private Laser[] laserList = new Laser[GameConstants.NumLasers];

// Random number

private Random random = new Random();

// Shooting sound

// http://soundbible.com/1461-Big-Bomb.html

// Accessed: 01/12/12

private SoundEffect explosionFX;

// Menu sound

// http://www.soundjay.com/button/sounds/button-27.mp3

// Accessed: 01/12/12

private SoundEffect menuFX;

private Song music;

// Texture for menu

Texture2D backgroundTex;

// Width and Height ints

int screenWidth;

int screenHeight;

// The aspect ratio determines how to scale 3d to 2d projection.

float aspectRatio;

// Initialises basic effect for drawing model

BasicEffect basicEffect;

public Game1()

{

graphics = new GraphicsDeviceManager(this);

Content.RootDirectory = "Content";

}

// Game States for menus

public enum GameStates

{

Menu,// Main Menu

Instructions,// Instructions

Running,// Game mode

End// Game over screen

}

// Method to setup scene

protected override void Initialize()

{

// Gives window a title

Window.Title = "Space Conquerors";

// Gets aspectRatio using the Graphics device

aspectRatio = GraphicsDevice.Viewport.AspectRatio;

// Creates instances of classes

menu = new Menu();

gamestate = GameStates.Menu;

cam = new Camera();

// Sets default values for the camera

cam.InitializeCamera(aspectRatio);

// Sets dafault values for effect

InitializeEffect();

// Creates instance of terrain

landscape2 = new Terrain(GraphicsDevice);

// Spawns baddies

ResetDaleks();

base.Initialize();

}

/// <summary>

/// LoadContent will be called once per game and is the place to load

/// all of your content.

/// </summary>

protected override void LoadContent()

{

// Create a new SpriteBatch to draw textures

spriteBatch = new SpriteBatch(GraphicsDevice);

screenWidth = GraphicsDevice.PresentationParameters.BackBufferWidth;

screenHeight = GraphicsDevice.PresentationParameters.BackBufferHeight;

// Loads font

fontToUse = Content.Load<SpriteFont>(".\\Fonts\\DrWho");

// Creates instance of player class, passing thruogh file paths as strings

player = new Player("Models/p1\_wedge", "Audio/pew", Content);

// Loads image for menu

backgroundTex = Content.Load<Texture2D>("Graphics/newsplash copy");

// Loads music, explosion and menu sound

music = Content.Load<Song>(".\\Audio\\gamesound");

explosionFX = Content.Load<SoundEffect>("Audio\\boom");

menuFX = Content.Load<SoundEffect>("Audio\\menu");

// Loads all assets for daleks and lasers

mdlDalek = Content.Load<Model>(".\\Models\\dalek");

mdDalekTransforms = SetupEffectTransformDefaults(mdlDalek);

mdlLaser = Content.Load<Model>(".\\Models\\laser");

mdlLaserTransforms = SetupEffectTransformDefaults(mdlLaser);

// Starts music

MediaPlayer.Play(music);

// Creates terrain instance, loading a height map and a texture

landscape2.SetHeightMapData(Content.Load<Texture2D>("Graphics/heightMap2"), Content.Load<Texture2D>("Graphics/newlandtex"));

// Creates skybox instance, passing file path as a string

skybox = new Skybox("Graphics/SkyBox", Content);

aspectRatio = graphics.GraphicsDevice.Viewport.AspectRatio;

}

// Sets up effect defaults by passing in the mesh

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();

// Takes matrices from the camera

effect.Projection = cam.projectionMatrix;

effect.View = cam.viewMatrix;

}

}

return absoluteTransforms;

}

// Method to be used when drawing models from an array, passes in model and transform details

public void DrawModel(Model model, Matrix modelTransform, Matrix[] absoluteBoneTransforms)

{

//Draw the model, a model can have multiple meshes, so loop

foreach (ModelMesh mesh in model.Meshes)

{

//This is where the mesh orientation is set

foreach (BasicEffect effect in mesh.Effects)

{

effect.EnableDefaultLighting();

effect.View = cam.viewMatrix;

effect.World = absoluteBoneTransforms[mesh.ParentBone.Index] \* modelTransform;

}

//Draw the mesh, will use the effects set above.

mesh.Draw();

}

}

protected override void UnloadContent()

{

}

// Method to spawn daleks, called at beginning of game

private void ResetDaleks()

{

float xStart;

float zStart;

// Loops through dalek list

for (int i = 0; i < GameConstants.NumDaleks; i++)

{

// Starting position using random numbers

xStart = -((float)-GameConstants.PlayfieldSizeX + 0);

zStart = -(float)random.NextDouble() \* GameConstants.PlayfieldSizeZ;

// Gives daleks position using randomised xStart and zStart

dalekList[i].position = new Vector3(xStart, 2.0f, zStart);

// Creates a random angle

double angle = random.NextDouble() \* 2 \* Math.PI;

dalekList[i].direction.X = -(float)Math.Sin(angle);

dalekList[i].direction.Z = (float)Math.Cos(angle);

// Gives speed using random factor

dalekList[i].speed = GameConstants.DalekMinSpeed +

(float)random.NextDouble() \* GameConstants.DalekMaxSpeed;

// Makes dalek active by default

dalekList[i].isActive = true;

// Console write for testing

Console.WriteLine("dalekpop");

Daleknum++;

}

}

// Method for checking input concerning the game

public void CheckGameInput()

{

lastState = keyboardState;

player.ShipControls(laserList);

keyboardState = Keyboard.GetState();

if ((GamePad.GetState(PlayerIndex.One).Buttons.Back ==

ButtonState.Pressed) || (keyboardState.IsKeyDown(Keys.Escape)))

this.Exit();

}

// Method for checking input concerning the menu

public void CheckMenuInput()

{

lastState = keyboardState;

keyboardState = Keyboard.GetState();

if (gamestate == GameStates.Menu)

{

if (keyboardState.IsKeyDown(Keys.Down) && (lastState.IsKeyUp(Keys.Down)))

{

// Move selection down

menu.Iterator++;

menuFX.Play();

}

else if (keyboardState.IsKeyDown(Keys.Up) && (lastState.IsKeyUp(Keys.Up)))

{

// Move selection up

menu.Iterator--;

menuFX.Play();

}

if (keyboardState.IsKeyDown(Keys.Enter))

{

if (menu.Iterator == 0)

{

gamestate = GameStates.Running;

}

else if (menu.Iterator == 1)

{

gamestate = GameStates.Instructions;

}

else if (menu.Iterator == 2)

{

this.Exit();

}

menu.Iterator = 0;

}

}

else if (gamestate == GameStates.Instructions)

{

if (keyboardState.IsKeyDown(Keys.Escape))

{

gamestate = GameStates.Menu;

}

}

else if (gamestate == GameStates.End)

{

this.Exit();

}

}

// Update method, updates all active instances, removes "dead" lasers and daleks, checks for input

protected override void Update(GameTime gameTime)

{

// Do only if game is in "running state"

if (gamestate == GameStates.Running)

{

CheckGameInput();

CheckCollisions();

cam.camUpdate(player.position);

float timeDelta = (float)gameTime.ElapsedGameTime.TotalSeconds;

// Update dalek list

for (int i = 0; i < GameConstants.NumDaleks; i++)

{

dalekList[i].Update(timeDelta);

}

// Update laser list

for (int i = 0; i < GameConstants.NumLasers; i++)

{

if (laserList[i].isActive)

{

laserList[i].Update(timeDelta);

}

}

}

// Do only if in a menu

else if ((gamestate == GameStates.Menu) || (gamestate == GameStates.Instructions) || (gamestate == GameStates.End))

{

CheckMenuInput();

}

// Music controls

if (keyboardState.IsKeyDown(Keys.C))

MediaPlayer.Pause();

else if (keyboardState.IsKeyDown(Keys.V))

MediaPlayer.Resume();

base.Update(gameTime);

}

// Check collisions between the different models

private void CheckCollisions()

{

// Give bounding body to player

BoundingSphere PlayerSphere =

new BoundingSphere(player.position,

player.myModel.Meshes[0].BoundingSphere.Radius \*

GameConstants.ShipBoundingSphereScale);

// Loop through dalek list

for (int i = 0; i < dalekList.Length; i++)

{

if (dalekList[i].isActive)

{

// Give bounding body to dalek

BoundingSphere dalekSphereA =

new BoundingSphere(dalekList[i].position, mdlDalek.Meshes[0].BoundingSphere.Radius \*

GameConstants.DalekBoundingSphereScale);

// Dalek vs Laser

// Loop through lasers

for (int k = 0; k < laserList.Length; k++)

{

if (laserList[k].isActive)

{

// Give bounding body to laser

BoundingSphere laserSphere = new BoundingSphere(

laserList[k].position, mdlLaser.Meshes[0].BoundingSphere.Radius \*

GameConstants.LaserBoundingSphereScale);

// If dalek and laser intersect

if (dalekSphereA.Intersects(laserSphere))

{

// play explosion FX

explosionFX.Play();

// Set them to false if they collide so they are removed

dalekList[i].isActive = false;

laserList[k].isActive = false;

// Increase score

player.score++;

// Decrease enemy number

Daleknum--;

break;

}

}

}

// Dalek vs player

// If dalek and player intersect

if (dalekSphereA.Intersects(PlayerSphere))

{

// play explosion FX

explosionFX.Play();

// Put player back to starting position

ResetPlayer();

break;

}

}

}

}

// Method to reset the player when it is killed

public void ResetPlayer()

{

// Decrease lives

player.lives--;

// Reset player position to playerspawn position

player.position = player.playerspawn;

// Write line for testing

Console.WriteLine(player.lives);

// If player is out of lives then trigger game over screen

if (player.lives == 0)

{

gamestate = GameStates.End;

}

}

// Method to setup effect

private void InitializeEffect()

{

// Creates new effect, passes in graphics device

basicEffect = new BasicEffect(graphics.GraphicsDevice);

// Takes view, projection and world matrices from camera

basicEffect.View = cam.viewMatrix;

basicEffect.Projection = cam.projectionMatrix;

basicEffect.World = cam.worldMatrix;

// Enable texture on effect

basicEffect.TextureEnabled = true;

}

// Draw method of terrain

public void DrawLand2()

{

landscape2.basicEffect.CurrentTechnique.Passes[0].Apply();

SetEffects(landscape2.basicEffect);

GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;

foreach (EffectPass pass in landscape2.basicEffect.CurrentTechnique.Passes)

{

// Call draw method from terrain class

landscape2.Draw();

}

}

public void SetEffects(BasicEffect basicEffect)

{

basicEffect.View = Matrix.CreateLookAt(cam.camPosition, cam.camLookat, Vector3.Up);

basicEffect.Projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(45.0f), aspectRatio, 1.0f, 50000.0f);

basicEffect.World = cam.worldMatrix;

}

// Method to draw text on screen

private void writeText(string msg, Vector2 msgPos, Color msgColour)

{

spriteBatch.Begin();

string output = msg;

// Find the center of the string

Vector2 FontOrigin = fontToUse.MeasureString(output) / 2;

Vector2 FontPos = msgPos;

// Draw the string

spriteBatch.DrawString(fontToUse, output, FontPos, msgColour);

spriteBatch.End();

}

// Main draw method for game, calls draw method on all instances of objects

protected override void Draw(GameTime gameTime)

{

graphics.GraphicsDevice.Clear(Color.CornflowerBlue);

// All drawing code for game if it's running

if (gamestate == GameStates.Running)

{

Matrix[] transforms = new Matrix[player.myModel.Bones.Count];

// Draw player

player.Draw(cam, aspectRatio, transforms);

// Loop list of daleks

for (int i = 0; i < GameConstants.NumDaleks; i++)

{

// if instance is active, draw it

if (dalekList[i].isActive)

{

Matrix dalekTransform = Matrix.CreateScale(GameConstants.DalekScalar) \* Matrix.CreateTranslation(dalekList[i].position);

DrawModel(mdlDalek, dalekTransform, mdDalekTransforms);

}

}

// Loop list of lasers

for (int i = 0; i < GameConstants.NumLasers; i++)

{

// if instance is active, draw it

if (laserList[i].isActive)

{

Matrix laserTransform = Matrix.CreateScale(GameConstants.LaserScalar) \* Matrix.CreateTranslation(laserList[i].position);

DrawModel(mdlLaser, laserTransform, mdlLaserTransforms);

}

}

// Draw terrain

DrawLand2();

graphics.GraphicsDevice.RasterizerState = RasterizerState.CullClockwise;

// Draw skybox

skybox.Draw(cam);

graphics.GraphicsDevice.RasterizerState = RasterizerState.CullCounterClockwise;

// Draw HUD

writeText("Score: "+player.score,new Vector2 (10,10),Color.Red);

writeText("Lives: " + player.lives, new Vector2(10, 25), Color.Red);

writeText("Camera: " + cam.camtext, new Vector2(10, 40), Color.Red);

writeText("Baddies left: " + Daleknum, new Vector2(10, screenHeight - 25), Color.Red);

// Reset drawing states after spritebatch from writetext changes them

GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;

GraphicsDevice.BlendState = BlendState.Opaque;

GraphicsDevice.DepthStencilState = DepthStencilState.Default;

}

// Drawing code for Main Menu

else if (gamestate == GameStates.Menu)

{

spriteBatch.Begin();

Rectangle screenRectangle = new Rectangle(0, 0, screenWidth, screenHeight);

spriteBatch.Draw(backgroundTex, screenRectangle, Color.White);

menu.DrawMenu(spriteBatch, screenWidth, fontToUse);

spriteBatch.End();

GraphicsDevice.BlendState = BlendState.Opaque;

GraphicsDevice.DepthStencilState = DepthStencilState.Default;

}

// Drawing code for Game Over Screen

else if (gamestate == GameStates.End)

{

spriteBatch.Begin();

Rectangle screenRectangle = new Rectangle(0, 0, screenWidth, screenHeight);

spriteBatch.Draw(backgroundTex, screenRectangle, Color.White);

menu.DrawEnd(spriteBatch, screenWidth, fontToUse,player.score);

spriteBatch.End();

GraphicsDevice.BlendState = BlendState.Opaque;

GraphicsDevice.DepthStencilState = DepthStencilState.Default;

}

// Drawing code for Instructions sc

else if (gamestate == GameStates.Instructions)

{

spriteBatch.Begin();

Rectangle screenRectangle = new Rectangle(0, 0, screenWidth, screenHeight);

spriteBatch.Draw(backgroundTex, screenRectangle, Color.White);

menu.DrawInstructions(spriteBatch, screenWidth, fontToUse);

spriteBatch.End();

GraphicsDevice.BlendState = BlendState.Opaque;

GraphicsDevice.DepthStencilState = DepthStencilState.Default;

}

base.Draw(gameTime);

}

}

}

**Camera.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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;

namespace CourseworkLGreen

{

/// <summary>

/// Adapted from:

/// http://blackboard.gcal.ac.uk/bbcswebdav/pid-1163578-dt-content-rid-776603\_2/courses/MHG605291-12-A/GP3%20Lab%204%20Exercises%20%26%20Code%20Snippet.pdf

/// Tutorial for creating camera

/// Accessed: 05/11/12

/// </summary>

public class Camera

{

// Rotation Matrix for camera to reflect movement around Y & X axis

public Matrix camRotationMatrixY;

public Matrix camRotationMatrixX;

// Position of camera

public Vector3 camPosition;

// Direction that the camera is currently looking or pointing at

public Vector3 camLookat;

// Used for repositioning the camera after it has been rotated

public Vector3 camTransformX;

public Vector3 camTransformY;

// Amount of rotation

public float camRotationSpeed;

// Rotation on Y axis

public float camYaw;

// Rotation on Z axis

public float camPitch;

// Text to display which cam is active, default third person

public string camtext = "Third Person";

// Booleans for checking which camera is active

public bool firstperson = false;

public bool thirdperson = true;

public bool freecam=false;

// Matrices for projection, world and view

public Matrix projectionMatrix;

public Matrix worldMatrix;

public Matrix viewMatrix;

// Method to set default values for a camera instance

public void InitializeCamera(float aspectRatio)

{

camPosition = new Vector3(0.0f, 5.0f, 10.0f);

camRotationSpeed = 100.0f;

viewMatrix = Matrix.CreateLookAt(camPosition, Vector3.Zero, Vector3.Up);

projectionMatrix = Matrix.CreatePerspectiveFieldOfView(

MathHelper.ToRadians(45),

aspectRatio,

1.0f, 10000.0f);

worldMatrix = Matrix.Identity;

}

// Keeps camera updated with changes in the world

public void camUpdate(Vector3 Position)

{

camRotationMatrixY = Matrix.CreateRotationY(camYaw);

camRotationMatrixX = Matrix.CreateRotationX(camPitch);

camTransformX = Vector3.Transform(Vector3.Forward, camRotationMatrixX);

camTransformY = Vector3.Transform(Vector3.Forward, camRotationMatrixY);

// Updates where camera is looking

camLookat = camPosition + camTransformX + camTransformY;

// Detects if a change in camera has been requested

ChangeCamera();

// Passes out different values depending on cam type detected

CamBehaviour(Position);

// Updates viewMatrix using new values

viewMatrix = Matrix.CreateLookAt(camPosition, camLookat, Vector3.Up);

}

// Method to switch camera depending on user input

public void ChangeCamera()

{

KeyboardState keyboardState = Keyboard.GetState();

if (keyboardState.IsKeyDown(Keys.D1))

{

// First person

firstperson = true;

camtext = "First Person";

thirdperson = false;

freecam = false;

}

if (keyboardState.IsKeyDown(Keys.D2))

{

// Third person

thirdperson = true;

camtext = "Third Person";

freecam = false;

firstperson = false;

}

if (keyboardState.IsKeyDown(Keys.D3))

{

// Top down

freecam = true;

camtext = "Top Down";

thirdperson = false;

firstperson = false;

}

}

// Method to make the camera behave differently depending on type selected

public void CamBehaviour(Vector3 Position)

{

if (thirdperson == true)

{

// Position set to behind and above player

camPosition = new Vector3(Position.X, Position.Y + 3, Position.Z + 10);

// Looking at player

camLookat = Position;

}

else if (freecam == true)

{

// Position set to directly above player

camPosition = new Vector3(Position.X, Position.Y + 60, Position.Z + 1);

// Looking slightly infront of player

camLookat = new Vector3(Position.X, Position.Y, Position.Z - 20);

}

else if (firstperson == true)

{

// Position set to players position

camPosition = new Vector3(Position.X, Position.Y, Position.Z-1);

// Looking slightly infront of player

camLookat = new Vector3(Position.X, Position.Y, Position.Z - 3);

}

}

}

}

**Terrain.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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;

namespace CourseworkLGreen

{

/// <summary>

/// Adapted from:

/// http://en.wikibooks.org/wiki/Game\_Creation\_with\_XNA/3D\_Development/Landscape\_Modelling

/// Tutorial for creating landscape using a heightmap

/// Accessed: 23/11/12

/// </summary>

class Terrain

{

// Class for creating terrain from heightMap

GraphicsDevice graphicsDevice;

// HeightMap

Texture2D heightMap;

// Texture for ground

Texture2D heightMapTexture;

// Terrain vertices

VertexPositionTexture[] vertices;

// Width and height for area, taken from heightMap

int width;

int height;

// Effect

public BasicEffect basicEffect;

// Terrain vertices

int[] indices;

// Array to read heightMap data

float[,] heightMapData;

public Terrain(GraphicsDevice graphicsDevice)

{

this.graphicsDevice = graphicsDevice;

}

// Sets values to prepare, then builds terrain

public void SetHeightMapData(Texture2D heightMap, Texture2D heightMapTexture)

{

this.heightMap = heightMap;

this.heightMapTexture = heightMapTexture;

width = heightMap.Width;

height = heightMap.Height;

// Values now ready, build terrain

SetHeights();

SetVertices();

SetIndices();

SetEffects();

}

// Takes height from heightMap depending on pixel brightness

public void SetHeights()

{

Color[] greyValues = new Color[width \* height];

heightMap.GetData(greyValues);

heightMapData = new float[width, height];

for (int x = 0; x < width; x++)

{

for (int y = 0; y < height; y++)

{

heightMapData[x, y] = greyValues[x + y \* width].G / 3.1f;

}

}

}

// Creates coordinates for building terrain

public void SetIndices()

{

// Amount of triangles

indices = new int[6 \* (width - 1) \* (height - 1)];

int number = 0;

// Collect data for corners

for (int y = 0; y < height - 1; y++)

for (int x = 0; x < width - 1; x++)

{

// Create double triangles

// Two triangles make a square

indices[number] = x + (y + 1) \* width; // up left

indices[number + 1] = x + y \* width + 1; // down right

indices[number + 2] = x + y \* width; // down left

indices[number + 3] = x + (y + 1) \* width; // up left

indices[number + 4] = x + (y + 1) \* width + 1; // up right

indices[number + 5] = x + y \* width + 1; // down right

number += 6;

}

}

// Creates shapes by linking points

public void SetVertices()

{

vertices = new VertexPositionTexture[width \* height];

Vector2 texturePosition;

for (int x = 0; x < width; x++)

{

for (int y = 0; y < height; y++)

{

texturePosition = new Vector2((float)x / 25.5f, (float)y / 25.5f);

vertices[x + y \* width] = new VertexPositionTexture(new Vector3(x, heightMapData[x, y], -y), texturePosition);

}

}

}

// Applies the texture to the shapes

public void SetEffects()

{

// Takes in Graphics Device from main class

basicEffect = new BasicEffect(graphicsDevice);

// Applies texture from main class

basicEffect.Texture = heightMapTexture;

basicEffect.TextureEnabled = true;

}

// Draw method to draw terrain

public void Draw()

{

graphicsDevice.DrawUserIndexedPrimitives<VertexPositionTexture>(PrimitiveType.TriangleList, vertices, 0, vertices.Length, indices, 0, indices.Length / 3);

}

}

}

**Menu.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

namespace CourseworkLGreen

{

/// <summary>

/// Adapted from:

/// http://ross-warren.co.uk/pong-clone-in-xna-4-0-for-windows/4/

/// Tutorial for creating menu

/// Accessed: 15/11/12

/// </summary>

class Menu

{

// Class for creating a menu

// A list for the menu items

private List<string> MenuItems;

// Int to keep track of selected item

private int iterator;

// Strings for title of menu and info text

public string InfoText { get; set; }

public string Title { get; set; }

// get and set for the iterator

public int Iterator

{

get

{

return iterator;

}

set

{

iterator = value;

if (iterator > MenuItems.Count - 1) iterator = MenuItems.Count - 1;

if (iterator < 0) iterator = 0;

}

}

// Constructor for menu

public Menu()

{

Title = "";

MenuItems = new List<string>();

MenuItems.Add("Play");

MenuItems.Add("Instructions");

MenuItems.Add("Exit");

Iterator = 0;

InfoText = string.Empty;

}

public int GetNumberOfOptions()

{

return MenuItems.Count;

}

public string GetItem(int index)

{

return MenuItems[index];

}

// Draw method for Main menu screen

public void DrawMenu(SpriteBatch batch, int screenWidth, SpriteFont arial)

{

batch.DrawString(arial, Title, new Vector2(screenWidth / 2 - arial.MeasureString(Title).X / 2, 20), Color.White);

int yPos = 120;

for (int i = 0; i < GetNumberOfOptions(); i++)

{

Color colour = Color.White;

if (i == Iterator)

{

colour = Color.Cyan;

}

batch.DrawString(arial, GetItem(i), new Vector2(screenWidth / 2 - arial.MeasureString(GetItem(i)).X / 2, yPos), colour);

yPos += 30;

}

}

// Draw method for the Gameover screen

public void DrawEnd(SpriteBatch batch, int screenWidth, SpriteFont arial, int Score)

{

int yPos = 120;

int xpos = 30;

batch.DrawString(arial, GameConstants.endtext+Score, new Vector2(xpos , yPos), Color.Red);

string prompt = "Press Escape to End";

batch.DrawString(arial, prompt, new Vector2(screenWidth / 2 - arial.MeasureString(prompt).X / 2, 400), Color.White);

}

// Draw method for the Instructions screen

public void DrawInstructions(SpriteBatch batch, int screenWidth, SpriteFont arial)

{

batch.DrawString(arial, Title, new Vector2(screenWidth / 2 - arial.MeasureString(Title).X / 2, 20), Color.White);

int yPos = 120;

int xpos = 30;

Color colour = Color.White;

batch.DrawString(arial, "Instructions", new Vector2(xpos, (yPos + 20)), Color.Red);

batch.DrawString(arial, GameConstants.text, new Vector2(xpos, yPos + 50), colour);

batch.DrawString(arial, "Controls", new Vector2(xpos, (yPos + 90)), Color.Red);

batch.DrawString(arial, GameConstants.controls, new Vector2(xpos, yPos+120), colour);

yPos += 30;

}

}

}

**Player.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

namespace CourseworkLGreen

{

/// <summary>

/// Adapted from:

/// http://ross-warren.co.uk/pong-clone-in-xna-4-0-for-windows/4/

/// Tutorial for creating menu

/// Accessed: 15/11/12

/// </summary>

class Menu

{

// Class for creating a menu

// A list for the menu items

private List<string> MenuItems;

// Int to keep track of selected item

private int iterator;

// Strings for title of menu and info text

public string InfoText { get; set; }

public string Title { get; set; }

// get and set for the iterator

public int Iterator

{

get

{

return iterator;

}

set

{

iterator = value;

if (iterator > MenuItems.Count - 1) iterator = MenuItems.Count - 1;

if (iterator < 0) iterator = 0;

}

}

// Constructor for menu

public Menu()

{

Title = "";

MenuItems = new List<string>();

MenuItems.Add("Play");

MenuItems.Add("Instructions");

MenuItems.Add("Exit");

Iterator = 0;

InfoText = string.Empty;

}

public int GetNumberOfOptions()

{

return MenuItems.Count;

}

public string GetItem(int index)

{

return MenuItems[index];

}

// Draw method for Main menu screen

public void DrawMenu(SpriteBatch batch, int screenWidth, SpriteFont arial)

{

batch.DrawString(arial, Title, new Vector2(screenWidth / 2 - arial.MeasureString(Title).X / 2, 20), Color.White);

int yPos = 120;

for (int i = 0; i < GetNumberOfOptions(); i++)

{

Color colour = Color.White;

if (i == Iterator)

{

colour = Color.Cyan;

}

batch.DrawString(arial, GetItem(i), new Vector2(screenWidth / 2 - arial.MeasureString(GetItem(i)).X / 2, yPos), colour);

yPos += 30;

}

}

// Draw method for the Gameover screen

public void DrawEnd(SpriteBatch batch, int screenWidth, SpriteFont arial, int Score)

{

int yPos = 120;

int xpos = 30;

batch.DrawString(arial, GameConstants.endtext+Score, new Vector2(xpos , yPos), Color.Red);

string prompt = "Press Escape to End";

batch.DrawString(arial, prompt, new Vector2(screenWidth / 2 - arial.MeasureString(prompt).X / 2, 400), Color.White);

}

// Draw method for the Instructions screen

public void DrawInstructions(SpriteBatch batch, int screenWidth, SpriteFont arial)

{

batch.DrawString(arial, Title, new Vector2(screenWidth / 2 - arial.MeasureString(Title).X / 2, 20), Color.White);

int yPos = 120;

int xpos = 30;

Color colour = Color.White;

batch.DrawString(arial, "Instructions", new Vector2(xpos, (yPos + 20)), Color.Red);

batch.DrawString(arial, GameConstants.text, new Vector2(xpos, yPos + 50), colour);

batch.DrawString(arial, "Controls", new Vector2(xpos, (yPos + 90)), Color.Red);

batch.DrawString(arial, GameConstants.controls, new Vector2(xpos, yPos+120), colour);

yPos += 30;

}

}

}

**Daleks.cs**

using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.Xna.Framework;

namespace CourseworkLGreen

{

/// <summary>

/// Taken from Robert Law's lab 5

/// Simple class to creat instance of dalek

/// </summary>

struct Daleks

{

// Enemies position

public Vector3 position;

// Enemies direction

public Vector3 direction;

// Enemies speed

public float speed;

// Bool to hold whether instance is active or not

public bool isActive;

public void Update(float delta)

{

// Move enemy using direction, speed and randomly generated number

position += direction \* speed \*

GameConstants.DalekSpeedAdjustment \* delta;

// If player leaves play field invert their direction so they stay within the area

if (position.X > GameConstants.PlayfieldSizeX + 80)

direction = -direction;

if (position.X < -GameConstants.PlayfieldSizeX + 90)

direction = -direction;

if (position.Z > GameConstants.PlayfieldSizeZ)

direction = -direction;

if (position.Z < -GameConstants.PlayfieldSizeZ)

direction = -direction;

}

}

}

**Laser.cs**

using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.Xna.Framework;

namespace CourseworkLGreen

{

/// <summary>

/// Taken from Robert Law's lab 5

/// Simple class to creat instance of laser

/// </summary>

struct Laser

{

// Laser position

public Vector3 position;

// Laser direction

public Vector3 direction;

// Laser speed

public float speed;

// Bool to hold whether instance is active or not

public bool isActive;

public void Update(float delta)

{

// Move laser using direction, speed and randomly generated number

position += direction \* speed \*

GameConstants.LaserSpeedAdjustment \* delta;

// Deactivate laser if it goes out of play field

if (position.X > GameConstants.PlayfieldSizeX +80||

position.X < -GameConstants.PlayfieldSizeX + 90 ||

position.Z > GameConstants.PlayfieldSizeZ ||

position.Z < -GameConstants.PlayfieldSizeZ)

isActive = false;

}

}

}

**Skybox.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Content;

namespace CourseworkLGreen

{

/// <summary>

/// Adapted from:

/// http://rbwhitaker.wikidot.com/skyboxes-1

/// Tutorial for creating skybox

/// Accessed: 19/11/12

/// </summary>

public class Skybox

{

// The skybox model, which is a cube

private Model skyBox;

// The skybox texture

private TextureCube skyBoxTexture;

// The effect file that the skybox will use to render

private Effect skyBoxEffect;

// The size of the cube, used so that we can resize the box for different sized environments.

private float size = 50f;

// Creates a new skybox instance using the texture provided

public Skybox(string skyboxTexture, ContentManager Content)

{

// Loads cube model

skyBox = Content.Load<Model>("Models/cube");

// Loads texture

skyBoxTexture = Content.Load<TextureCube>(skyboxTexture);

// Loads fx file

skyBoxEffect = Content.Load<Effect>("Effects/Skybox");

}

// Draws Skybox, no world matrix since it won't move

// Passes in camera instance so to use current view and projection matrices

public void Draw(Camera cam)

{

// Go through each pass in the effect, but we know there is only one...

foreach (EffectPass pass in skyBoxEffect.CurrentTechnique.Passes)

{

// Draw all of the components of the mesh, even though the cube only has one mesh

foreach (ModelMesh mesh in skyBox.Meshes)

{

// Assign the appropriate values to each of the parameters

foreach (ModelMeshPart part in mesh.MeshParts)

{

part.Effect = skyBoxEffect;

part.Effect.Parameters["World"].SetValue(

Matrix.CreateScale(size) \* Matrix.CreateTranslation(cam.camPosition));

part.Effect.Parameters["View"].SetValue(cam.viewMatrix);

part.Effect.Parameters["Projection"].SetValue(cam.projectionMatrix);

part.Effect.Parameters["SkyBoxTexture"].SetValue(skyBoxTexture);

part.Effect.Parameters["CameraPosition"].SetValue(cam.camPosition);

}

// Draw the mesh with the skybox effect

mesh.Draw();

}

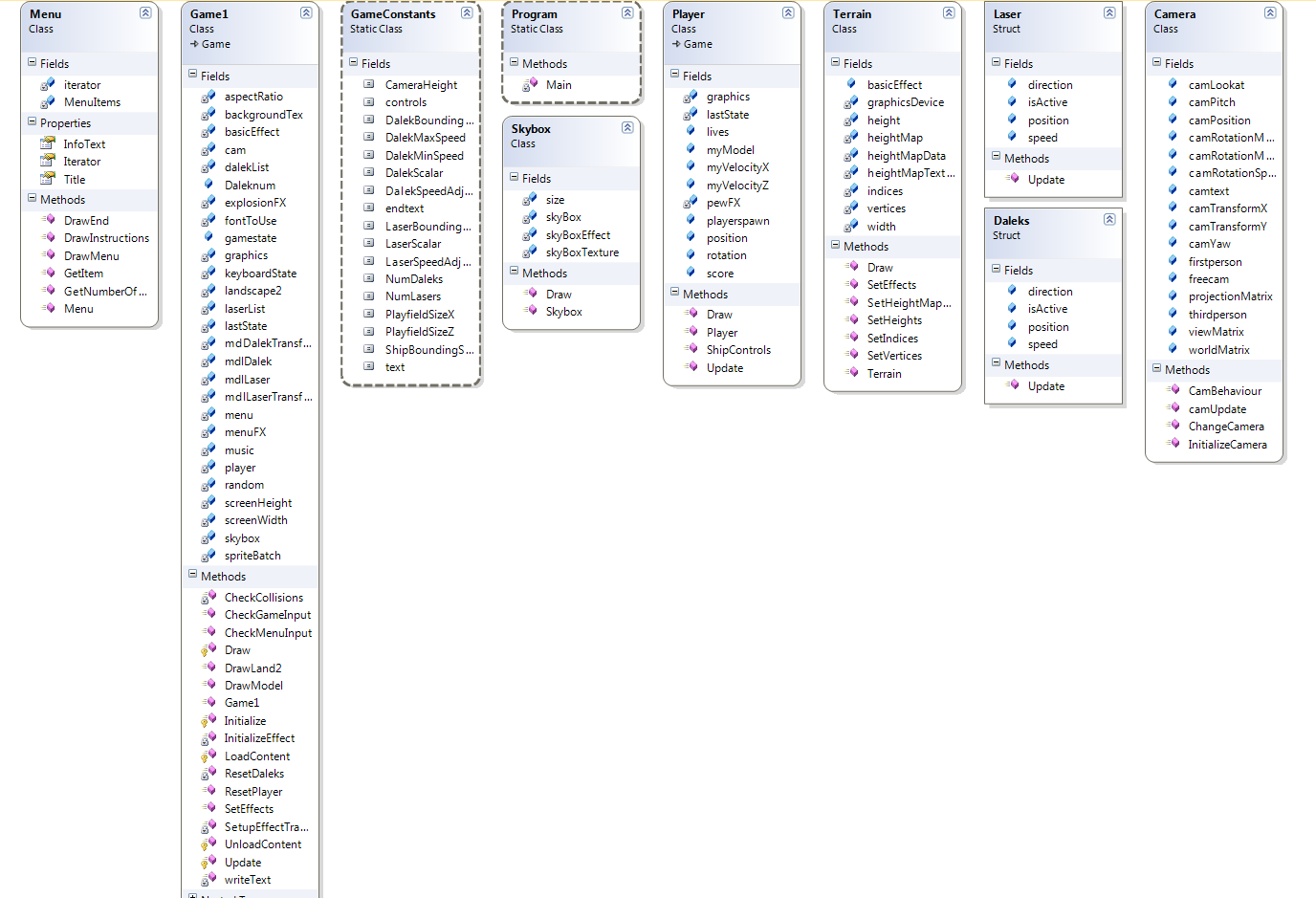
}

}

}

}

# Appendix B – Class Diagram



# Appendix C – StoryboardsC:\Users\Laurie Green\Pictures\CONTENT.jpg