**Alisdair Muircroft**

**S1019695**

**Computer Games (Software Development)**

**Games Programming 3 Coursework Documentation**

*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:*

**Explanation of code used to generate the game**

This game uses eight classes to generate the game. Each class will be covered individually in this explanation. First the GameConstants.cs class, this class holds the constant variables that are used throughout the program, these include the size of the game play area, the number, scale and bounding sphere size of all object that will be used as well as their speed values.

Laser.cs contains the update behaviour of the lasers this method will move the laser in the correct position as long as it is within the game play area. When the laser moves outside this area it deactivates it.

Asteroid.cs and eShip.cs contain the same behaviours but with minor differences. They contain an update and a reset method. The reset method moves the model to a random x,y,z coordinate at their spawn point. The update method moves the model towards the player and if it goes behind the camera positions it back at their spawn point with a random x,y,z coordinate and increases their speed by one up to a cap which is set, the eShips cap is lower than the asteroids. Start.cs contains the same update bewhaviour however does not have the reset method. Program.cs is the main entry pint for the application it has one method which runs the game.

Camera.cs contains the code for the cameras. It holds all variables linked to the camera and its constructor takes in a position variables to set its initial position. The InitilizeTransform() method creates the view,world and projection matrixes that the camera uses. Its camUpdate should rotate the camera around the player ship. Finally the RotateCamera Method should update the position and direction of the camera

Game1.cs is where the main functionality of the application is held it has eleven methods as well as holding the majority of variables that will be used. This class will be described as the code is written from the top down thus not necessarily in the order the code is run. At the beginning this class defines and sets up most of the variables that will be sued within this class these include the model objects, arrays to hold them, cameras positions of objects controller states and the current game state. The first method in the class is Game1() this method creates the graphics device manager that will be used and finds the directory that holds the content of the game.

MoveModel() is the next method it holds the controls that the player will use through the game separated by game state. It begins by getting the current states of the input devices and positions of the mouse and model. There are two controls that are always active no matter what state the game is in these are the mute toggle which checks wither the M key has been pressed and toggles the sound accordingly, the other is the quit key which exits the application when the escape key is pressed. The method then splits the controls into game and control states. When the game is in the pause state the O key un-pauses the game, when un the main menu the 1,2 and 3 keys choose the control type and start the game. When in the game state how the player controls the model is defined by what key they pressed in the menu however the pause key is always P which changes the game state to pause and the R key will reset the game. If they chose the 1 key the player uses the mouse to control the model. This is worked out by binding the mouse to the screen viewport finding where the mouse position is in relation to the viewport then normalizing it against a plane to give responsive movement and moving the model to its position, it also checks to see when the left mouse button has been clicked and fires a laser if it has.

The next type of movement is the keyboard. It uses the arrow keys to directly move the model about the screen and moves it back if it would leave the game play area. The space bar is used to fire the laser in this control scheme. The last control scheme is the Xbox 360 gamepad this uses the d-pad to directly move the model in the same way as the keyboard and uses the A button to shoot, additionally the start button can be used to reset the game and the back button can be used to quit the game.

The following method is position models this method initializes the positions of the models that are not the player and creates the direction and the speed they will be moving in. The method SetupEffectTransformDefaults is used to get the points of each model and apply the camera effects to them this makes sure each part of the model is shown onscreen. DrawModel() sets up the model to be draw by setting up where the mesh orientation of each mesh in the model will be and applies the world effect to them before passing them on to be drawn. The WriteText() method creates the sprite batch, vectors and functionality to draw text onto the screen. After this is the Initilize() method this sets the positions of the cameras, calls the positionModels(), Initilizes the window and sets the camera. LoadContent() finds and loads in all the external resources that are required and handles them appropriately. UnloadContent however is not used as nothing is unloaded.

Update() contains everything that will be running during the applications updates. This is also split into gamestates. MoveModel() is called every update as it handles the users input. When the game is paused the camUpdate() method of the main camera is called which should orbit the camera around the player ship. When in the game state it checks to see if the players health is below 0 if so changes state to the game-over screen, it then updates the game time and calls the update method of all the models in the scene. After this it creates the bounding sphere of the player and starts to check for collisions, it does this by creating a bounding sphere on every model and checking if they intersect. If a asteroid intersects with another asteroid it moves one forward slightly. If the asteroid hit the player it reduces the player health then calls the asteroid reset() method, if a laser is hit the laser gets deactivated. For the enemy ships they reduce the players health even more if they hit the player before resetting their position, however if a laser hits them score is added before the reset() method and the laser deactivated. For all collisions if the sound is on a noise is played finally the game Time is once again updates.

Finally the Draw() method, this method draws everything onto the screen. This is split up into game states. If the mainMenu state is on the menu text is drawn onto the screen. If in the end screen state game over text and the final score is shown. When the game is running it draws all models onto the screen with the current health and score. Lastly when in the paused state it draws the game scene without the health and score, with instructions on how to un-pause.

**References:**

http://gamedev.stackexchange.com/questions/24894/making-mouse-pos-realtive-to-a-moving-object

http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx

http://msdn.microsoft.com/en-us/library/bb200104.aspx