

|  |  |
| --- | --- |
| **Name:** | Michele Tammaro |
| **Matriculation No.:** | S0908639 |
| **Course:** | Computer Games Software Development |
| **Module:** | Games Programming 3 |
| **Module** **Code:** | MHG605291-13-A |
| **Module** **Leader:** | Bobby Law |
| **Date:** | 16/12/2013 |

“Yellow Submarines vs. The Blue Meanies” – Games Programming 3 Coursework. A game developed using Visual Studio and the XNA 4.0 Framework

*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 fulfillment of this or any other award.*

Signed: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date:\_\_\_\_\_\_\_\_\_\_\_

# Code Explanation

*The following explanation provides a high level overview of the functions that XNA runs by default. It will reference other user defined methods and classes that perform a wide range of actions. For more information on what these methods and classes do see “Explanation of User Defined Methods, Classes and Structs”.*

When the user begins to play the game, the first task that the code performs is to invoke the main class’ constructor (a special method that initializes the class). In this constructor the variables are initialized, and then the method “Initialize” is called. This method creates the game Window and calls 2 methods; InitializeTransform and ResetDFG. These methods create the cameras and the enemies that the player will battle.

When the constructor has been invoked, the values of the variables are then declared. Once this step has been completed the LoadContent method is called by XNA. This method is responsible for loading and assigning models, fonts, sound effects etc. to the appropriate variables. For each model the method SetupEffectTransformDefaults method is called so that transformation matrices and BasicEffcts can be assigned to the models and appropriate cameras.

When content has been loaded and assigned the next method that XNA calls by default is Update. This method is called once every frame. The Update method performs different tasks depending on what GameState is set in the game. Regardless of the State, the update method will always initialize and assign a value to keyboard states (for input), values for positive and negative speed (for movement of the enemies).

The update function calls different methods and performs different tasks depending on the state that the game is in. Some GameStates share common functionality such as player movement, spawning (resetDFG) and updating enemies (updateModels) and collision checks between on-screen entities. (collisionCheck).

When the game begins, it defaults to the “Menu” GameState at run-time. During this state the Update Function looks for player input to change the state to “Play”

When "Play" is active it runs a check for the number of enemies that have been defeated. When this count reaches a predetermined number (e.g. 50) then the GameState is changed to Boss. When Boss is active the collision Sphere is created within collisionCheck and the moveBoss method is called.

If the players health reaches 0 then the GameState changes from Play or Boss to "Lose" and text is drawn on the screen using the method writeText informing the player that the game is over. It prompts to user to restart (gameReset) or exit the game. If the bosses health reaches 0 then the GameState changes to "Win” using the method writeText informing the player that the game is over. It prompts to user to restart (gameReset) or exit the game. In both these states the players score is recorded – using a StreamWriter – onto a .txt file.

The final method that XNA calls is the "Draw" function; the method responsible for drawing models/sprites/fonts in the game Window. Like the Update function this method is called every frame, and will draw different components depending on what State the game is in. By default, the game will always draw the Player model and the terrain.

Following this the game will then draw different components in different states. The game begins from the “Menu” GameState, and 2D Texture is drawn and displayed to the player, prompting for input. When the game starts and is in “Play”, the player healthbar, bolts and enemies are drawn and updated on the screen. When the GameState changes to “Boss” the game additionally draws the enemy boss, the bolts that the boss shoots and its healthbar.

When the game enters “Win” or “Lose” text is drawn on the screen in this method. The text and position of each state changes, however the same method is used. The text prompts the user to reset or exit the game.

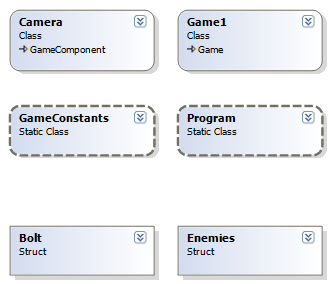
## Explanation of User Defined Methods, Classes and Structs

There are a number of methods that the user has defined in order to develop this game. This section will discuss in-depth the function of these methods.

* InitializeTransform()
  + This method initializes the games aspect ratio to allow the game to determine how to scale the 3D to 2D projection. It then initializes two cameras; the main and the secondary. For each camera position and rotation values are passed, as well as the speed. It ends by attach the camera to a Game Component.
* gameControls()
  + This method is responsible for the majority of the control input for the game, and allows the game to be controlled using both the mouse/keyboard and Official Xbox 360 GamePad. This method moves the player model, fires projectiles and handles toggles for sound and camera.
* ResetDFG()
  + This method spawns and/or resets the games enemies. It creates a roation matrix to be applied to each enemy before assigning different attributes to the enemies in a for-loop. This foor loop assigns position, direction, speed, rotation and active properties to each enemy.
* SetupEffectTransformDefaults(Model myModel)
  + This method returns a transformation matrix for the model that has been passed into it. Before it can return a matrix it needs to create a Matrix using the models bone count, before copying this transform back to the matrix. It then applies effects to all of the meshes within the model. These effects include lighting and View/Projection Matrices for the cameras.
* DrawModel(Model model, Matrix modelTransform Matrix[] absoluteBoneTransforms)
  + Before this method can be called the user must pass in a number of parameters. This method needs a model, an associated transform and the returned value of the previous method. As models have multiple meshes the user needs to loop through each mesh. Within the loop for the mesh there is another loop to apply effects to the meshes, such as lighting and fog.
* writeText(string msg, Vector2 msgPos, Color msgColour)
  + This method allows the user to draw text on the screen within any method. It needs a string, a position and a colour to be passed into it. The method begins a sprite batch before defining variable values depending on what has been passed into it. With these variables it utilizes the “DrawString” method of spriteBatch and draws the message on the screen
* updateModels(float timeDelta)
  + This method is responsible for updating non-player models on the screen such as the bolts and the enemies. This method takes in a float called timeDelta. This float is passed in from the Update function, and is required so that it is updated at the same time as frame changes occur. This ensures that movement of enemies appears smooth. The game loops for each enemy and bolt within individual for loops, and utilizes the “update” function of each array, passing in timeDelta to these methods.
* collisionCheck()
  + This method is responsible for creating invisible geometry around models so that it can check for collisions. The method creates a “Boundsphere” around the player, enemies and bolts, and performs checks and actions when these spheres intersect. BoundingSpheres are created depending on model position, radius and then scaled depending on the user preference.  
    Collisions are checked for in a loop, depending on whether or not associated arrays are active. If the enemies are active BoundingSpheres are created for each enemy. Within this loop another loop is instantiated for the bolts if they are active. When the Boundspheres have been created checks are performed for if any of them intersect. Intersection between an enemy and a bolt renders both inactive. Intersection between an enemy and the player reduces the players health and destroys that enemy in the loop. Intersection between the player and a boss bolt dramatically reduces the players health and deactivates the bolt.
* updateBoss(float delta)
  + Like previous methods, this method requires changes in gameTime to be passed in as a float. This method moves the enemy boss between waypoints that have been instantiated at run-time. It creates a small bounding sphere for the boss, and bounding spheres for each waypoint. When the spheres do not intersect the boss moves to the next waypoint in the loop. The bosses’ movement changes depending on his position relative to the next waypoint (i.e. if the bosses y position is lower than the waypoints then he moves in a positive direction on the y-axis). This method will also loop through an array of bolts that the boss will shoot. The loop – like gameControls() – defines attributes for the bolt such as speed and direction. Firing is continuous while the boss is active.
* playSongs()
  + The games main theme changes depending on what game state the game is in. This method handles the changes in theme by reloading the content when the state changes. This allows the game to use one variable for multiple themes.
* toggleSongs()
  + This method is responsible for toggling sound effects on and off. Input from gameControls changes the value of a Boolean. This method performs different actions depending on the value of this Boolean.
* gameReset()
  + This method resets all values back to their default state. Resetting the game occurs at 3 different points in the game (During Play, after a Win and after a loss) after user input so a single method is required to handle this.
* Bolt.cs
  + A **struct**ure that creates position, direction, speed and active status of each object created. This struct has an update method that changes the bolts position
* Camera.cs
  + This class is used to create and update the cameras movement and position. This allows multiple cameras to be created in a game with different and unique attributes. This class handles keyboard, mouse and GamePad input into it
* Enemies.cs
  + This struct defines the variables and methods for the enemies. In a dedicated update function the enemies positions are updated, and direction changed depending on their location in the world.
* GameConstants.cs
  + This static class contains a collection of Constant variables that are used throughout the project in different classes and structs. This class is static, and the values are constants to prevent any changes from occurring at RunTime.

**Class Diagram**

*The following class diagram was copy directly from the Visual Studio 2010 project.*



**Storyboards**

# References

Dowhilefor 2012, *Special effects in XNA 3D* [online]. Available at: <http://stackoverflow.com/questions/13661137/special-effects-in-xna-3d> [Accessed 12/09 2013].

Fibericon 2012, *Switching songs – MediaPlayer lags the game* [online]. Available at: <http://gamedev.stackexchange.com/questions/25919/switching-songs-mediaplayer-lags-the-game> [Accessed 12/08 2013].

Gamerdad81 2013, *Programming a 2D space shooter tutorial #27 – XNA  - game states (menu)*[online]. Available at: <http://www.youtube.com/watch?v=Z2OzDTYeCy0> [Accessed 12/06 2013].

George Clingerman 2008, *The state of things* [online]. Available at: <http://www.xnadevelopment.com/tutorials/thestateofthings/thestateofthings.shtml> [Accessed 12/06 2013].

<http://www.shockwave-sound.com/sound-effects/scream_sounds.html> 2013, *Sound effects* [online]. Available at: <http://www.shockwave-sound.com/sound-effects/scream_sounds.html> [Accessed 12/09 2013].

Microsoft Developer Network 2011, *XNA game studio 4.0* [online]. Available at: <http://msdn.microsoft.com/en-us/library/bb200104(v=xnagamestudio.40).aspx> [Accessed 12/10 2013].

muffinman8641 2011, *Playing song (background music) with XNA 4.0* [online]. Available at: <http://www.dreamincode.net/forums/topic/241125-playing-song-background-music-with-xna-40/> [Accessed 12/06 2013].

Oyyou 2011, *XNA tutorial 13 – enum* [online]. Available at: <http://www.youtube.com/watch?v=KvwCr95FsgU> [Accessed 12/06 2013].

Sharky 2007, *XNA collision detection for 3D models – part 2* [online]. Available at: <http://sharky.bluecog.co.nz/?p=119> [Accessed 12/04 2013].

Sokolowski, M. 2012, *c# XNA is key toggled?*[online]. Available at: <http://gamedev.stackexchange.com/questions/34133/c-xna-is-key-toggled> [Accessed 12/08 2013].

Sonix2001 2006, *Vibrate controller for 1 second* [online]. Available at: <http://social.msdn.microsoft.com/Forums/en-US/baad9676-b525-46bd-9a02-f96fd6ed25fb/vibrate-the-controller-for-1-second> [Accessed 12/08 2013].

Tigran 2010, *2D particles in XNA* [online]. Available at: <http://blog.tigrangasparian.com/2010/10/08/2d-particles-in-xna-part-1-of-3/> [Accessed 12/09 2013].

# Appendix 1