# Programming for 3D - report

# Game: Space Mission

Candidate: 52665

**Motivation**

The main purpose of this project was to demonstrate the knowledge of Microsoft XNA 4.0, C# programming language and basic game development techniques. Another important factor that motivated the development of the game was the ability to create my own game which I can modify as needed and send to my friends and family. It was exciting to find out that 3D game development in XNA is not hard after breaking through the basic understanding of vector and matrix manipulation, game’s looping structure and content pipeline. Released game version is my fifth project attempt which was evolving throughout the dozen of smaller testing XNA projects.

**Goal of the game**

The game is called Space Mission and its goal is to collect astronauts by spaceship on time and avoiding as many obstacles as possible while carrying out the mission. Player also has to make sure to complete the mission in shortest time possible to obtain highest score possible. Astronauts can be picked up by running spaceship into them. Spaceship can be navigated by keys (Left, Right, Up, Down, A, Z, Q, W) and case camera can be switched to orbit camera by pressing (Space) and (J, L, I, K) to navigate it. Orbit camera can assist by better viewing angles when avoiding obstacles.

**Class and method description**

***Camera*** – One of the most important classes in the game. It took many hours to get this right. Although the code is not written from scratch but implemented by following the tutorial (ref. #1 - camera), I had to get familiar with concepts like (Yaw, Pitch, Roll) and matrix rotation and translation in XNA to acquire the flexibility in development my game.

ResetCamera() – this method sets the initial position of the camera with the offset from the cased model, speed with which it will react to model movements as well as sets the view and projection matrix.

public void Update(Matrix chasedObjectsWorld) – Camera’s update method which takes object to be chased and passing it to the update helper method.

private void UpdateViewMatrix(Matrix chasedObjectsWorld) – Update’s helper method which handles two camera views (case and orbit). This method takes object to chase and pull all important matrix data from it to set the future position and rotation of the camera as object changes position. Chase camera’s transition steps from current to future position are smoothed to ensure more realistic effect. Orbit camera allows for viewing the model from all directions.

private void HandleInput() – Assist’s mainly to orbit camera navigation

public void SwitchCameraMode() – Chase / orbit camera switch

***Game1*** – Main XNA class, first finds the resolution of the screen used and sets the game to full screen in that resolution. Then loads all contents (models, sprites, audio), updates and draws sprites and models on every game loop iteration. Game has a simple state management of four game states (Loading, Running, Won, Lost) and these are active depending on where the player is in the game. This technique was learned and implemented by following the tutorial (ref. #2 – game state management).

protected override void Update(GameTime gameTime) – This method first gets current keyboard state for use in below code, then it checks for game states and executes the code respectively. As this is the main Update method, there are calls made to other helper method from here which are responsible for updating individual models. This is to ensure that modularity and code readability is in place.

private void CheckForCollision() – checks for all collision detection in the game except ground-spaceship (which is handled in the SpaceShip class). When spaceship hits the moving asteroid it is removed from the scene, explosion is played, hit variable is incremented by one, DecreaseLife() method is called to act respectively and score is decreased by the amount specified in each asteroid (obstacle) type. This technique of detecting a collision was learned and implemented by following a tutorial (ref. #3 – collision detection). See more details in SpaceShip class paragraph below. When spaceship intersects with an astronaut it removes astronaut from the scene, reward sound is played, astronautCollected variable is incremented by one and score is incremented by 333.

protected override void Draw(GameTime gameTime)- Main draw method calls other helper draw methods depending on the state game is in.

private void CreateObstacles1, 2, 3, 4, 5, 6() – All these methods create obstacles (asteroids) and place them in positions specified by a vector3.

private void CreateAstronauts() – Method creates astronauts and places them in positions specified by a vector3.

private void CreateStars() – Method creates the universe by randomly placing little sphere models (stars) in X, Y, Z coordinates.

private void moveObstacles1, 2, 3, 4, 5, 6() – Method moves obstacles (asteroids) according to vector3 and bound limits. Counters are used as bounding limits to ensure correct up, down, forward, backward movements within game’s loop.

public void PlayCue(string cue) – Method plays sound from the track cue. Each sound is specified by a string parameter which this method takes. The concept of audio engine and a track cue was learned and implemented by following a tutorial (ref. #4 - audio).

private void DrawSplashScreen() – Helper method draws the loading/splash screen with an image and instructions on how to play the game. It centres the string used in the middle of the screen by finding a string width/height and size of the screen.

private void DrawGameplayScreen() – Helper method draws the running game screen with all 3D objects (asteroids, spaceship, astronauts) and 2D sprites (life bar, UHD). It sets the GraphicDevice so that it renders properly 2D sprites in 3D environment.

private void DrawStats() – Helper method to draw game stats (time, astronauts collected). This method is also responsible for decreasing score as time progress.

private void DrawWinOrLossScreen(string gameResult) - Helper method draws the Win/Loose screen by taking appropriate string argument depending on the outcome of the game. It also displays the final score.

All four above methods have been reused and modified from (ref. #2 – game state management).

private void DecreaseLife() – This method check the hit variable and swaps the sprite image appropriately to show life decrease.

void graphics\_PreparingDeviceSettings(object sender, PreparingDeviceSettingsEventArgs e) – Method which checks the resolution of current display. This concept was used from (ref. #5 – screen resolution detection).

***GameConstants –*** Contains most of the constants and strings used throughout the game for easy access.

***Ground –*** Class representing the ground (mars surface) of the game. It has a load method which takes a content pipeline variable and name of the mode in string to be loaded. Its draw method takes the camera as an argument and uses camera’s view and projection matrices to correctly draw the model.

***Obstacle –*** Class representing asteroids, astronauts and stars and has same methods as the Ground method with exception of a constructor which takes a float parameters to set position of each object individually.

***SpaceShip –*** Class representing a space ship which can freely change position in X, Y, Z direction. It has a tilting effect when changing a position Left/Right for more realistic effect. Spaceship rotation has boundaries to prevent object from turning upside-down when a “0” value is reached in pitch rotation. Spaceship also returns to its original rotation after Left/Right keys are up to ensure 0% roll and maintain the spaceship straight.

public void ResetPosition() – Sets the spaceship to its initial position

public void Update(KeyboardState keyBoardState, Camera camera, Ground ground) – Taking the keyboardstate, camera and ground variable and uses them to correctly move/rotate the spaceship. At the same time GroundHit() method is called to check for collision with the ground and camera is also updated on each iteration of a game loop. Couple of if statements ensure all the rotation and translation is handled. The spaceship uses 4 different matrices to do this right. The main ShipWorld matrix is responsible for spaceship position. ShipRotationGlobal is responsible for correct yaw rotation where ShipRotationLocal handles the pitch rotation. The reason for having two matrices for handling rotation is that as spaceship changes position, its Right and Up vector also change. Finally there is the ShipTranslation matrix wich is modified and multiplied to ShipWorld matrix to move the spaceship in appropriate direction.

public void Draw(Camera camera) – Method draws the spaceship by iterating through all model’s meshes and applying the effects, transformation, view and projection to each mesh.

public bool CollidesWith(Model otherModel, Matrix otherWorld)

As each model has a bounding sphere computed on creation of the model there was no need for any extra code as shown in Fuel Cell game example. All it is need is to compare spaceship model to model which intersects with other model at any given game loop iteration. More specifically method iterates through each model meshes bounding spheres and checks if they intersect.

public void GroundHit() – Method for collision detection checking between spaceship and ground. It check the current spaceship position on Y coordinate and if it’s less then specified value it pulls the spaceship back on its Backward vector.

Some self-explanatory method description was left out as well as those methods already described.

**Advanced functionality**

* Space ship has built in maneuvering enhancements such as tilting (roll) and constraints on pitch rotation as well as automatic steering to 0% roll (when Left/Right keys are up) to ensure more realistic experience.
* My own model of asteroid produced in 3DS max.
* Asteroids controlled by computer
* Different scoring system for colliding with different asteroids
* Game state management
* Upper head display
* 2 types of camera view (chase, orbit)

**Conclusion**

Although the game could seem very simple and wouldn’t probably by most selling game of the 2011, there has been lot of effort put into it. I have underestimated the power of XNA and time it takes to produce a simple 3D game as well as inevitable knowledge of vector and matrix manipulation. On the other hand I am very proud I managed to put together my very first 3D game using cutting edge technology and C# programming language.

**References**

ref. #1 - camera  
<http://www.sgtconker.com/2010/09/article-simple-3d-camera-in-xna/>

ref. #2 – game state management  
<http://msdn.microsoft.com/en-us/library/dd254740.aspx>

ref. #3 – collision detection  
<http://oreilly.com/catalog/0636920013709>  
Chapters (9 - 12)

ref. #4 - audio  
<http://oreilly.com/catalog/0636920013709>  
Chapter (12)

ref. #5 – screen resolution detection  
<http://social.msdn.microsoft.com/Forums/en-US/xnaframework/thread/7dda014e-8d9e-47dc-8bb0-278a81fed461>

Additional references:

* Model of astronaut  
  <http://artist-3d.com/>
* Model of spaceship  
  Microsoft Direct X10
* Texture of ground  
  <http://phrogram.com/forums/t/1787.aspx>
* Audio  
  <http://oreilly.com/catalog/0636920013709>  
  (additional files)