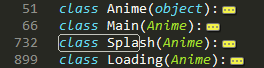
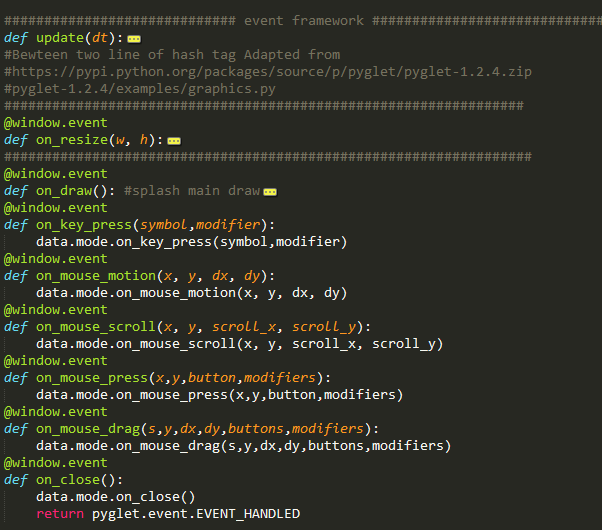
Program Design and Explanation

## Program Design

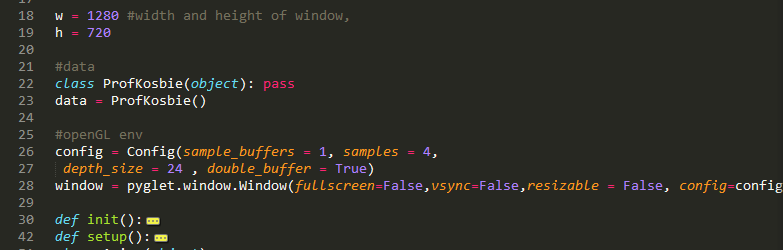
The program I implemented is a design of a 3D air combat shooting game. It is a mix between first player shooting, third player shooting, action game and flying simulating game. The biggest distinctive feature of this game is that players are able to pitch, roll, yaw in any way they wish, just like in real life in a 3D world. With the 3D flying implementation, players can now fight with computer controlled characters by emitting shells and missiles in 3D world

## Technical Implementation

1. The whole program is implement in object oriented programming. The program have Main screen, Splash screen and Loading screen, these three kinds of rendering mode. Thus each mode is implemented as an object and each have their own function of handling window events. Thus the main program only need to store the current mode object and the dispatcher will always send events to current object medthods.

 Anime class is the parent class with empty dispatcher so that children class can override handles as they wish.

Event framework dispatches events.

1. Top part of the code initialize openGL environment and create initial mode object.
2. To realize real 3D freedom flying in the world, the world coordinate need to be rotated about the player. This is rotated by matrix arithmetic’s, openGL has built in functions of rotation and transformations but sometime c-types arrays, as openGL matrix, still need to be calculated by the main program and push it to openGL. The matrix.py modulus include most of the matrix and vector arithmetic functions. Among Ru function is the matrix transformation function that allow rotation about any axis in a certain angle. This one is used to rotate the player’s coordinate as after the initial rotation, player’s local coordinate system is no longer the homogeneous coordinate system x,y,z.
3. While player can fly around and rotate the world in any manner, their ship need to be always at the center of the world and they will need to be able to look around based on their own coordinate system. Player is rendered at the center of the screen by rendering them and discarding all the previous transformation matrix in openGL stack. Thus player will always be the ‘center of the world’. Player’s camera movement is realized by openGL function gluLookAt. I need to feed this function my look start direction, look vector and my normal vector. It is indeed a combination of glTranslatef and glRotatef, which move and rotate the world. Then I applied vector math and trigonometry to work out my coordinates. There is a shift in my preference of math method form trig math to vect math as I learned, a lot, about trig math alone the way. For example, sin(x)cos(x) do the same job as a dot product.
4. Next technical importance is loading models from files, which is the functionality of modulus loadObj1.py. Models I use is save in obj, mtl and texture map format. In the obj file, it records the homogenous coordinate of every single vertex, also it record the texture coordinates, face connection relationship and normal direction between every three or four vertex. This file tells openGL what to draw, as normally I feed openGL with vertex coordinates, texture coordinates, etc to let it render a primitive. Model is just a collection of 10k and more primitives. The mtl file records the material information including texture map(picture file name), lighting parameters, etc. It make a blank white primitive colored and reflecting light. This part is more like a word processing process, yet each group of vertex with different textures need to be drawn in a specific order so that the texture paste on them (openGL load a texture file and paste to any vertex drawn afterward). This is achieved by BindTexture class, I reference the method from pylget documentation. All asteroids, player, shell, missiles are models.
5. Weapons launched from the player need to follow the direction the player is looking at, also their model need to be rotated to the specific direction. For shell models, because their direction is fixed, I transformed their model to player look coordinate at the time they are launched by inverse matrix. For missiles, which are rotating all the time, I calculate the inverse matrix, change it into c-type matrix and feed it to openGL.
6. Shells travel in parabolic arcs so it’s easier to apply gravity in homogenous coordinate thus I override the moveable class method. Player, target and missile use the method which always move the player toward their local cordinates’ negative Z axis. (this is very confusing and actually caused tons of bugs and exceptions. Because x is to the right, y is up, as conventional and easier to understand. But this means in a right hand coordinate system, Z axis will be facing us, aka, we are facing Z negative axis.) With this method, I only need to change player, target and missile coordinate direction to make them move reasonably.
7. During player movement, their collision detect with the asteroid is not very exact as asteroids are irregular in shape but a spherical collision detect is already costing enough in terms of performance.
8. Computer controlled characters are programmed to move between a distance range from the player. I simple change their forward vector proportional to the distance from the distance boundary so that they will get closer or go further away as need while continuing their circular motion.
9. Computer controlled character ‘cheated’ in attacking player. They fetch the speed of the player and calculate the amount of lead they needed to hit the player. Thus constant velocity motion will be punished by NPCs really hard. In the meantime, a random spread is applied so that they are not too OP and also allow then to hit when player rotating.
10. As mentioned earlier, shells and missiles are launched from the player. I added the functionality of following the shell with camera. Namely, player can trace the shell in third person view by having their viewing camera moving together with the shell and look at the direction they are flying to, this is cool is effect. This is achieved by vector math and gluLookAt.
11. Launching missiles need lock on, for this feature, I required the player to look at an NPC for a while. Then I pass the target to the missile and allow them to launch. Missiles will trace their target by aligning their forward direction to position of their target per update.
12. I implemented explosion effect by building BillBoard (note – not the BillBoard class I used, that one is just for displaying textured block).  The Explosion class is for rendering an explosion and ExpQuads class is the billboard that always turn toward the player. When I draw an explosion texture sequence on a quad and let the quad always face me, it will looks like an explosion.(at least there is no distortion). A better way to to have multiple quads with different textures and size, but it’s too costly to program directly. Also openGL transparency blending issue is rather confusing.
13. The universe around me, and the earth, is a box. Skybox is a box drawn around me and moving together with me with texture painted on it. If you look closely, you can see the lines on the fringe of the world and tell it’s a box. But it’s pretty realistic.

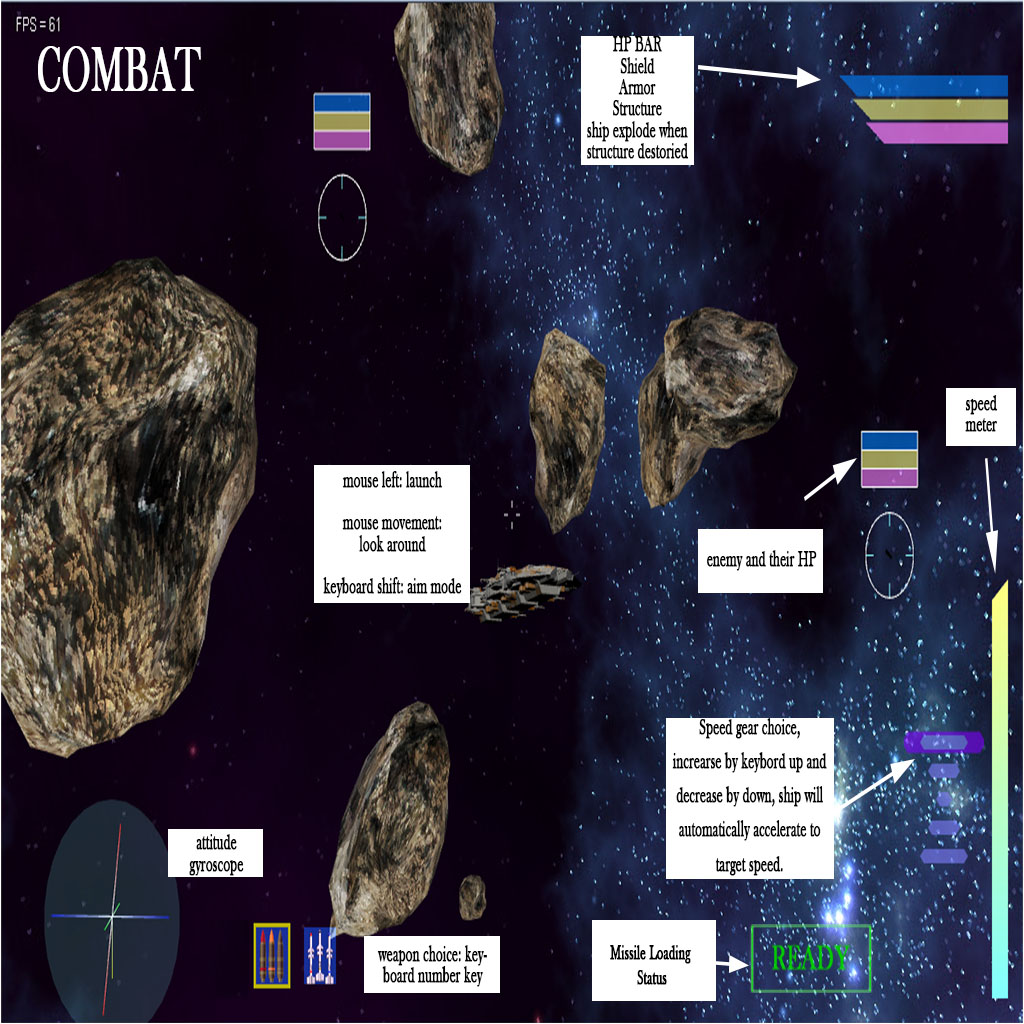
In conclusion, the most important technical implementation in this program is using vector math, matrix rotation and transformations to turn objects around and render the world in the way they should look like in real life.

## UI Design and Implemented Features

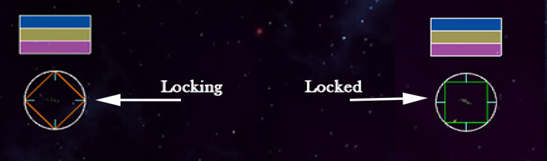


This is the splash screen UI. As it’s 3D space battle game, I want concept of 3D being stressed. The welcome UI is a space where asteroids coming toward the user. This design shall be able to cause a visual shock to the user. Then for the bars and bottoms I’ve chosen blue and h the theme color as blue is my favorite color and it fit exactly with space background. Blue, I believe, give people a feeling of ever expanding space, unlimited, unbounded.

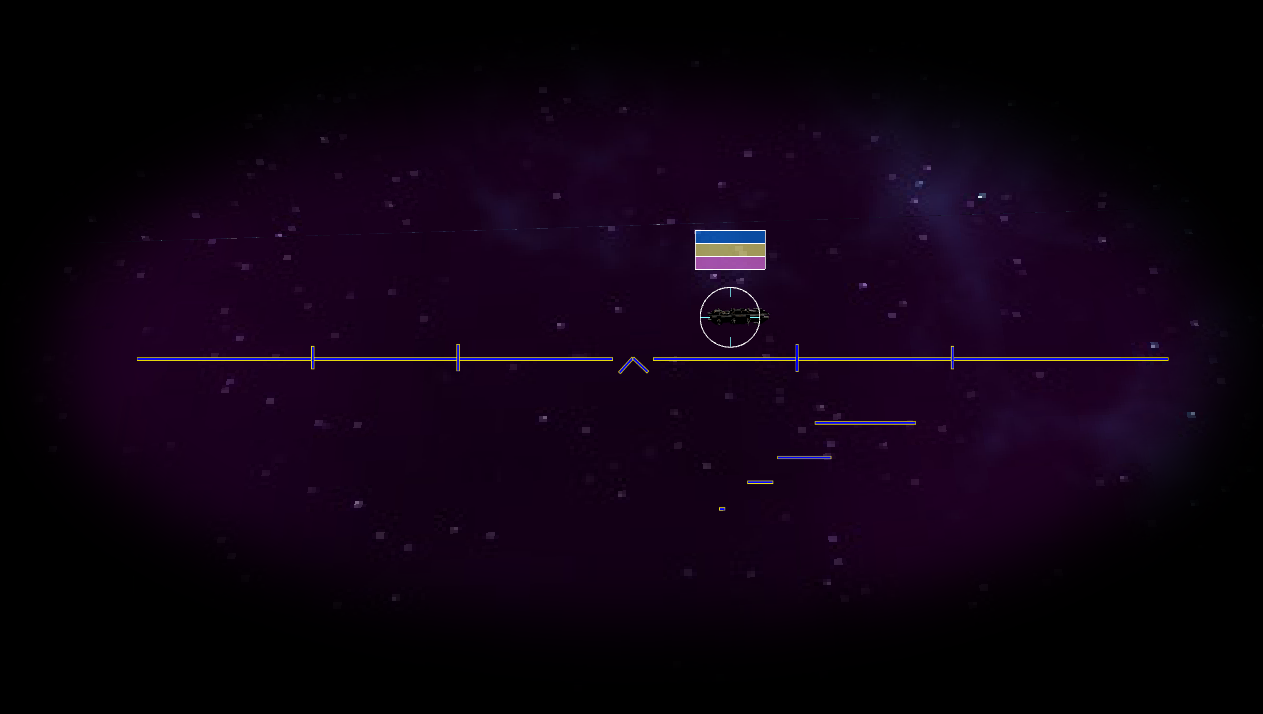
The main battle UI experience is like the following pictures from my game tutorial.

The attitude gyroscope in the left bottom corner is a components necessary for air combat game.

The weapon choice bar looks like the one in World of Warships, this is actually a feature from that game where shells of different types have different fetures.

The greenish READY and the lock on icons originated from other air combat games and air combat movie. I guess this is what people expect for LOCK-ON . But missiles, as in other games, should require lock on and have maximun flying time, making them dogable.

The three level HP bar is a feature I inherited from EVE, the universe based MMO. Ships should have multiple layers of armor and allowing for more tactical plays. Here in my game, I let the missile do damgages to shiled and armors first and a then penetrate ( with punishiment) to structure, while shells does damage directly on structure. But shells are really hard to use.

I always render the player at the center of the screen is a method implement in all three games: WarThunder, World of Warships and EVE. This is make for sense for a shooting game as everyuthiung lauching from the player come from that origin. Then I implemted the aiming mode in this manner , this is actually a feature I inherited from World of Warship.

World of Warship:

http://i.imgur.com/CkSplb6.png

In my opinion, zooming in and getting rid of everthing else can help player concentrate and shot more accurate so I removed everything from the screen

Also the shell tracing feature I implemented also come from World of Warship



<https://www.themittani.com/sites/default/files/Shelltime.jpg>

My game:

Although, without particle effect and salvo shots this looks not as awesome.

But I have missile trace, which is special for my air battle only.

In game World of Warships, players can move their cameras around their ship in a circular orbit. I extend that into a sphere so that players can look around and shoot in every direction.

In conclusion, for my UI experience I applied lots of features from existent games such as World of Warships, WarThunder and EVE and thus some aspect of my UI design resemble their games in multiples ways. However, as my game is set in universe air combat, I fit the UI experience well into my theme.