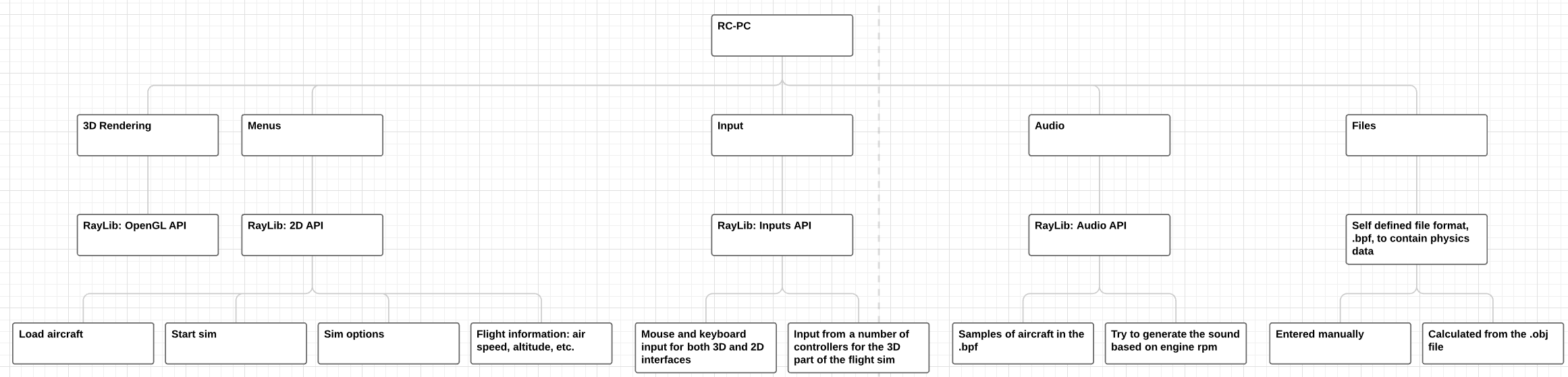
## ## Design



| ### Menus |
| --- |
| ### Objects |
| #### Objects |
| |Variable|Datatype|Reason| |:—|:—|:—| |position|3D Vector|for object location in 3D space| |velocity|3D Vector|to integrate to location in 3D space| |acceleration|3D Vector|to integrate to velocuty; simplifies gravity| |rotation|Quaternion|avoids gimble lock present in Euler implementations| |angular velocity|3D Vector|to integrate to rotation| |scale|3D Vector|for the x, y and z scale of each object| |model|raylib::Model|the model data: textures, meshes, etc.| |
| - Functions - Initialiser - Take a string from which to load the object data into the model attribute - Draw - Use the attributes to draw the object to the world - Update - Use numerical integration on the movement attributes |
| #### Vectors and Matrices |
| Use a 2D array to allow dynamic sizes of vectors and matrices Will need to consider how to invert / find the conjunction of square matrices; probably using an if statement and accounting for specific cases would be best |
| - Attributes - 1D element array - integer columns - integer rows - Functions - Indexing - Basic arithmetic - Powers - Inverse - Normalise |
| #### Other |
| To make full use of the axis-angle rotation, quaternions will be required This will allow full use of angular velocity and accelerations |
| While raylib already implements many of these, it may be beneficial to write my own vector and matrix lib to allow full use of all required overloads |

### Algorithms

#### Vector and Matrix Maths Lib

* Simulating a 2D array using a 1D array

double[] array = new array[width \* height]  
  
double get(x, y):  
 return array[x \* width + y]  
  
void set(x, y, val):  
 array[x \* width + y] = val

This will allow me to make use of a dynamic 2D array in C++ as there is no built in structures to allow this

matrix add(other):  
 for i in range(columns):  
 for j in range(rows):  
 self(i, j) += other(i, j)  
  
matrix sub(other):  
 for i in range(columns):  
 for j in range(rows):  
 self(i, j) -= other(i, j)  
  
matrix mult(other):  
 for i in range(columns):  
 for j in range(rows):  
 sum = 0  
 for k in range(columns):  
 sum += self(k, j) \* other(i, k)  
 self(i, j) = sum

I will also need to be able to calculate a matrix inverse for rotating quaternions and normal vectors Doing this will require: - A function to produce the minor matrix from a matrix and element location within the matrix - A function to produce the full minor matrix - A recursive function to calculate the determinant of a matrix - A function to calculate the cofactors - A function to calculate the final inverse/transpose of the matrix

matrix minor(x, y):  
 minor = matrix(this.m - 1, this.n - 1)  
 a = 0  
 b = 0  
 a\_inc = False  
  
 for i in range(this.m):  
 for j in range(this.n):  
 if not (x == i or y == j):  
 minor[a, b] = this[i, j]  
 b += 1  
 a\_inc = True  
 if a\_inc:  
 a += 1  
 a\_inc = False  
 b = 0  
 return minor  
  
double det():  
 if this.m == this.n:  
 if this.m == 2:  
 return this[0,0] \* this[1,1] - this[0,1] \* this[1,0]  
 else:  
 det = 0.0f  
 for i in range(this.m):  
 if i % 2 == 0:  
 det += this[i, 0] \* this.minor(i, 0).det()  
 else:  
 det -= this[i, 0] \* this.minor(i, 0).det()  
 return det  
  
matrix matOfMinors():  
 rtn = matrix(this.m, this.n)  
 if this.m == this.n:  
 for i in range(this.m):  
 for j in range(this.n):  
 rtn[i, j] = this.minor(i, j).det()  
 return rtn  
  
matrix cofactors():  
 rtn = matrix(this.m, this.n)  
 if this.m == this.n:  
 minors = this.matOfMinors();  
 for i in range(this.m):  
 for j in range(this.n):  
 if (i \* this.n + j) % 2 == 1:  
 rtn(i, j) = -minors[j, i]  
 else:  
 rtn[i, j] = minors[j, i]  
 return rtn  
  
matrix inverse():  
 return this.cofactors() \* (1/this.det())

|  |
| --- |
| ### Input Validation |
| - Combinations of keys: e.g. Ctrl, Shift, Caps Lock, etc. |

### Unit testing

* Testing specific functions with specific test cases
* Set up test functions to be run at program start while in debug mode to test the basic functions of classes and functions

##### Testing the matrix class

| Function to test | Given input | Expected output |
| --- | --- | --- |
| Vector get | [[matADef.png]] | [[vecGetTest.png]] |
| Vector set | [[VecSetInp.png]] | [[VecSetOut.png]] |
| Vector add | [[VecArithIn.png]] | [[VecAddOut.png]] |
| Vector subtract | [[VecArithIn.png]] | [[VecSubOut.png]] |
| Quaternion get | [[QuatDef.png]] | [[QuatGet.png]] |
| Quaternion set | [[QuatSetIn.png]] | [[QuatSetOut.png]] |
| Quaternion add | [[QuatAddIn.png]] | [[QuatAddOut.png]] |
| Quaternion multiply | [[QuatMultIn.png]] | [[QuatMultOut.png]] |
| Quaternion multiply with a negative | [[QuatMultNegIn.png]] | [[QuatMultNegOut.png]] |
| Quaternion rotate | [[QuatRotIn.png]] | [[QuatRotOut.png]] |
| Quaternion rotate continuity | [[QuatRotContIn.png]] | [[QuatRotContOut.png]] |
| Matrix multiply | [[MatMultIn.png]] | [[matMultOut.png]] |
| Matrix scale | [[MatScaleIn.png]] | [[MatScaleOut.png]] |
| Identity matrix inverse | [[IMatInvIn.png]] | [[IMatInvOut.png]] |
| Arbitrary matrix inverse | [[ArbMatInvIn.png]] | [[ArbMatInvOut.png]] |

This testing is run at startup of the program [[Startup test.png]] This function outputs the debug results to the terminal if the boolean argument is set to true [[MatClassDebug.png]] This ensures that the class is functioning correctly before entering the main loop