**General Notes Summary/My Thoughts**

* TODO: Add more comments in general, especially for math/physics
* Encapsulate more functionality - rendering is particularly a pain.
* Standardize code formatting and documentation; continue with cleaning the code.
* Clarify Cmake, libraries, and build processes. Readme download and setup should be clear and hassle-free to any contributors. Cmake build procedure partially done 2024-08-14.

**Handling Objects**

* Global vs Local coordinates - Is there any way to prevent confusion? This is unavoidable. Global means the whole scene, and local means a particular object such as a beyblade. If you want to spin the the beyblade, you rotate it around its own axis. If you want to view the entire game from a different side you can rotate the world or you can move the camera. Note also that it is customary to scale, rotate, and then translate (in that order). Chanaging the order produces vastly different and confusing results.
* Does GameObject encapsulate everything needed for all objects that might be needed in the future? Unanswerable!
* Currently, a RigidBody/ImmovableRigidBody class is being used as physical objects updated in PhysicsWorld->update. However, as Beyblades and Stadiums need specific interactions, I think it is better to have dedicated BeybladeBody and StadiumBody classes. No problem.

**Beyblade**

baseCenter

baseHeight

axisOfRotation

layerHeight

layerRadius

driverRadius

* Method of identifying position/orientation: **baseCenter** and **axisOfRotation**
  + This is assuming that a top’s rotation always matches its orientation.
  + Currently thinking of using quaternions for the entire codebase.
  + You may need to know the most extreme edge, too, in the event that you want to detect the upper part of beyblade colliding with another blade. Of course you can figure that out using the radius and angle of rotation. Quaternions are pretty cool in simplifying oddball movement. In my TenGun app, I use quaterions so you can click the ends of two different pipes, and then join the two ends together, even if the pipes are initially parallel.
* Minimum necessary variables
  + Object Information (rendering)
  + PhysicsState (linear/rotational mass, velocity, acceleration)
  + Physical measurements - layerRadius, layerHeight, baseHeight, driverRadius
  + LayerRecoil - class that samples a distribution (such as normal distribution)
  + DriverFriction - coefficient of friction.
* Future considerations
  + One popular feature of Beyblades is that parts are interchangeable. It perhaps might be helpful to allow Beyblade instantiation with classes Layer (top), Weight, (middle), Driver (bottom), which combine to provide the overall variables.
  + When instantiating with loadModel, there should be an option to manually input necessary variables. The ability to scale the Beyblade’s size to allow for consistency would also be greatly helpful.

**BoundingBox**

* How necessary are bounding boxes given an existing method to detect collisions? I could see them used for debugging and efficient broad detection, but my current use case does not have many items. Just comment out any bounding box code with #if 0. Down the road you can just delete it if no need reemerges.

**Buffers**

* TODO: More comments, and clean up unused functions.

**Callbacks**

* Could organize better into event handlers/listeners, especially with key presses.
* More key maps for debugging and new features

**GameControl.h**

* Is it standard to pass in all variables like this for available access? The GameControl class is just a handy place to store a bunch of not necessarily related variables. There is no “standard” way of doing this. If we were making an API that would be used by millions of people, it would be worth removing unnecessary items, but that is not the case here.

**Initialize**

* Nice consideration for monitors.

**Main.cpp**

* Should I be trying to minimize the number of global variables? A “global variable” is one that is accessible to the entire code base, and various functions access it by declaring it as “extern”. Use of global variables in object-oriented code is generally frowned upon, but sometimes unavoidable. Either you the one-and-only instance of some important variable in a global, or you have a local variable (such as in main) that you then pass to other code as a parameter. You don’t have any global variables.
* Definitely keep efficiency in mind; I’m not very performance-focused (mainly desire a simple working version at first), but inspection with depth test changes and ImGui rendering as you have mentioned as example would be greatly beneficial for this project’s future. My rule: Make it work (is it viable?), then make it fast and beautiful. Fujitsu rule: Make it beautiful in the advertsing, and if anybody wants it then create the product.
* I was trying to work with text rendering before finding out about ImGui, so I don’t have an immediate need for freetype text for now. However, having custom font animations for special attacks could be nice in the future. You need some kind of font support. Stick with freetype, otherwise you will have to make Windows-specific code. TODO: Build freetype static library for VS2022. This eliminates the missing dll problem.
* Better management/encapsulation of memory cleanup at program end? As a responsible software developer, always clean up the mess you made. Not deleting that things you allocated can lead to “memory leaks”, where you lose track of allocated memory, and eventually the system runs of of free memory. But good news: (1) your app isn’t doing anything like that, and (2) the operating system (or at least Windows) deletes all dynamically allocated memory when a process finishes.

**PhysicsWorld**

* Instead of checking bounding boxes, we can store BeybladeBody and StadiumBody objects and perform collision checks based on position/orientation only. Sounds good to me.

**RigidBody**

* As mentioned before, I intend to split into BeybladeBody (collision info) and StadiumBody (immovable) classes.

**ShaderPath**

* Is putting all defined paths into a single file standard? Change to variables instead of macros, or convert to configuration file, definitely some better way of organizing. A config file is always nicer – but get it working first.

**ShaderProgram**

* In general, need to clarify functionality and remove any extraneous variables to make code more readable. Yes!

**UI**

* Encapsulate/Modularize ImGui functions, and maybe split into multiple classes as size grows.
* TODO: Clarify screen flow. Is there any better way to handle logic than booleans or status variables in GameControl?
* In the future, we need to add functionality to customize and info screens.

**Object Shaders**

* Implement objectColor in object.vs. This one confused me quite a bit! The tricky part in doing this is I think there are multiple colors either in the stadium or in the beyblade (I forgot). If that can be resolved, all of the colors in the vertex data can be eliminated.

**Changelog**

* Seen and noted. Will we have a more formal changelog and/or use git/github? I would like to store progress on github periodically. That is certainly possible. For the time being, I will leave the github stuff to you – it is a pain when multiple people are working on exactly the same source code and making massive changes because everybody has to merge everybody else’s changes all the time. Once the code has more or less stable, this merging is less annoying.

**Coding Style**

* I have taken a look at some of the example rules, and I don’t think I have enough experience to judge most things, so I will follow whichever method you use/suggest.
* The only preferences I have:
  + Use type\* name and type& name. Yes!
  + Only camelCase or PascalCase. Yes!
  + If {
  + } Else { I would align this under the “if”, but it is your code.
  + }

**TODO.xlsx**

* I think having on-screen buttons to move UDLR and dragging the screen to rotate would be sufficient for mobile.
* However, the priority is for PC. The two perspectives I have envisioned during battle are free-move and attached above a top.
* **Ideal gameflow:** User starts at home menu.
  + They can either import their own beyblade model (PC only?) or customize a basic model from parameters or existing parts.
  + These beys will show up in a collection menu, which contains all of their beys.
* After clicking play, they are presented with options for what bey they use, what stadium to play on, how many points to go to, who their opponent is, etc.
* They are taken to the main 3D environment, where they can select a position, power, and direction of their launch. (Launch = initial spin of beyblade)
  + Once the launch happens, they can spectate the bey from a normal perspective or from their bey itself
  + The focus of the game is on accurate physics, but they can also have the opportunity to perform special moves or something.
  + Once a win condition is detected, points are awarded, and another launch happens. If the round ends, display a summary and return to the home screen.
* Online play would be very cool but that’s a whole other can of worms. Synchronization is not too much of a worry if physics is the only driving factor after launch.

**Next Steps**

* I am able to work on the program periodically as school starts again, though your advice and supervision of changes (at least for an initial period) would be greatly appreciated. My hope is that as this project goes on longer, I will be more knowledgeable and able to make more progress independently. I don’t have the budget for you to finish most of the game 🙃 (As stated before, budget is ~$5000 for a basic working version, but could be more if we reach a point where game release is viable)
* If you have any ideas of which improvements to make first, we could set those as tasks and discuss which components you would need to do.
  + Otherwise, my next steps in mind are the TODOs you note and the general notes listed at the top of this document. How about if I do the following:
    - A bit more study of GLFW / OpenGL (I haven’t used OpenGL recently), and see if we can turn the data into something more hierachical.
    - Figure out how to make objectColor work.