INFO3220 Object Oriented Design

Assignment 2 - Stage 1 Code Review

Documentation:

There was no separate documentation so I shall just speak about in-code comments. The in-code comments were limited to method comments, however, were fairly extensive and prescribed to the C++ commenting guidelines.

The inline comments were few and far between, however, were included were helpful. Documentation and commenting were specifically missing from the IOFile and BattleSphere implementation files where a lot of the code was hard coded and yet without explanation.

Whilst not strictly documentation the separating of the inclusion headers for Qt specific inclusions, std inclusions and project file inclusions served well for clarity when first reading through the project code.

Extensibility:

The code was very inextensible with great amounts of it being hardcoded and as such a lot of it had to either be changed or ignored for the purposes of keeping it for the assignment. Specifically, the IOFile was hard coded and as such didn't allow for extension. Also, the bullet and defender objects did not really leave room for an extension as they were both concrete rather than abstract which was a shame since alien bullets and alien objects need to be treated differently.

One well thought out abstract class was the game element object of which simply has the information for the X and Y coordinates of any on-screen objects. The objects almost forced the next developer to choose different design patterns as abstract objects hadn't been set up so design pattern such as the adapter design pattern.

The code also didn't have sound implemented completely and as such had to be finished so as to add the functionality.

There did seem to be much consideration for the extension though as there were getters and setters and default constructors for most objects and their member variables even though they weren't used in the first stage. These, however, did not come in handy as the objects themselves were not abstract and as such needed to be extended to become useful themselves.

Also, the inclusion of the config file by absolute path rather than adding it as a resource seemed like a poor choice for an assignment that would be passed out to different students with different directories.

Design:

The design pattern chose was Builder and was used to its minimum capability of which was enough for stage one of the assessment. It allowed for extension upon the defender despite there not needing to be for stage two. The choice for a design pattern was a good one if only stage one was to be considered however a design pattern such as abstract factory would have served better for extension allowing for aliens and defenders to be made from an abstract warrior class for example. The choose for the builder, however, was simpler for programmers to follow on afterwards as it is generally easier to understand.

The choose of making the IOFile a singleton was very good. This is appropriate as the config file throughout the coming stages only needs to be read in one time and as such shouldn't be able to have multiple instances of.

Implementation:

The code implementation had strengths and weaknesses. The builder design pattern was implemented well, however, there were a few objects where the meter variables were public and shouldn't have been such as the abstract game element interface which goes completely against the thought pattern for having getters and setters of which the object set out to have. The IOFile was implemented with no room for error whatsoever even accounting for specific spaces needing to be in the config.ini file and also commands having to come at specific line numbers rather than after each other. This could be fixed with a simple counter for the current line.

The code also creates most of the objects on stack memory and even when bullets are off the screen they are still in memory and are even still moved which adds to time and space complexity. The defender, bullets and stars are all painted on the display by simply referencing a variable storing the image with a name to indicate its the image for that object rather than actually being stored within the objects themselves. If the images were created the same way the code could at least store a pointer to them and act upon them by calling a getter from the object itself which would be more appropriate especially when considering object-oriented design principles, better yet a flyweight design pattern could be implemented so as to ensure one and only one instance of each media need to be created.

The implementation of the command centre also hinders extension as for the option of making levels the config file may need to be read in loops and as such an iterator rather than a list with commands being popped off would be more appropriate.

Some objects such as bullet also didn't have destructors and didn't seem to have any interest in being made dynamic.

The config file formatting also could have been further considered with options such as comma separating like in csv files or object formatting such as in XML or JSON, with a consideration that we were advised not to use son specifically. However, a structure that allowed for the further section to be added such as swarms of aliens, their trajectory and alien bosses without having to write additional hard-coded solutions.

The code also seemed to use mainly std C++ functionality rather than looking into the extensive library of Qt objects and classes. The Qt library is also optimised for their software and allows a greater transition for the next coder.

Another single example of the programmer thinking about extensibility was the inclusion of an unimplemented updateX method for the bullet class that if implemented properly would allow for the bullets to be moved diagonally when being fired, however, there are a lot better ways to do this.

Style:

The code was clearly beginning from the tutorial of which had unrelated class names of which were not changed and make for unclear meaning such as battle sphere and defender. The naming conventions did improve drastically after which the code was original implementation with each class having a succinct name of which was not too long nor too short and explained the purpose quickly. The layout of the code was consistent, with consistent whitespace, commenting and layout. The one exception would be that there was a mixture of newline opening brackets and ending line opening brackets for the exact same purposes such as methods, loops or conditional statements.

The code also lacked the use of const in places where it could provide stability and reduce errors or other programmers changing the functionality of objects in the next stage that they shouldn't be changing. An example of where const would be useful would be the pictures used for the defender, bullets and stars. As the stars are used to create a background they could especially be made const. An exception is the getter methods in defender specify they will not change the code and as such are const of which getters should be most of the time.