INFO3220 Assignment Stage 2 Report

Justin Ting, jtin2945, 430203826

April 2015

Documentation

BaseCodeVersionD did not supply any documentation in the form of a readme file, nor within the source code itself. As such, there were hurdles in setting up the code to run on a machine other than the original coder's. Information that would have been useful to include in the readme would have been:

- *Original build platform* - this would alert a programmer who would later use the code that perhaps some files (specifically, the config.ini where it made an actual difference) would have platform-specific elements, such as Windows line endings in text files. While the original coder was not expected to identify specifically this issue, mentioning Windows as the used platform would have proven helpful.
- *Use of config.ini* - this would be equally useful for a programmer as it is an end-user - by briefly describing what options are available, how they can be modified, and what behaviour would arise from basic error cases with a corrupted/invalid config.ini file, an end-user would know how much control they have over the program's environment, while a programmer, would know before even looking at code roughly how the config.ini is handled in the source code, and will more quickly interpret its contents once they do delve into it.
- *Miscellaneous* - while some of the following points don't/necessarily apply to this specific assignment, other information that generally would only be beneficial in a readme include the author, how to report bugs, feature requests, etc., other contact information as necessary, legal notices, etc.

Extensibility

This section will be used to discuss extensibility of elements not directly pertaining to design patterns, and will refer to extensibility in the general sense - not the adding of actual extensions in the context of the assignment.

The way the character object was implemented presents difficulties in extending its functionality, particularly with regards to interacting with other objecs in the game world. This is in part due to the fact that it is handled considerably differently from other objects which are drawn - one being the background which is rendered through QDialog's *paintEvent* method. The player, however, is a .gif movie which is simply placed on top of the background (via a QMovie set in a QLabel). Because all functionality of the player beyond stage 1 requires movement, a render method needs to be written to accommodate this.

Furthermore, because the first stage didn't require defining boundaries of the player object, the gif movie did not intrinsically contain any (an arbitray width was hard-coded), so the width of the player for each of the sizes had to be measured manually through trial and error.

Design

The Builder Pattern was claimed to have been used through naming of the relevant source files, a gppd choice of pattern given the task at hand, if we are to have foresight of the second and third stages. This is because while the first stage only contains the "game world" objects of the player and the background, the second stage adds

obstacles, while the third adds a user score, as well as powerups. The *Game* object becomes increasingly complex, and depending on how backwards compatability is handled, it may be the case that only a subset of these features should be included at compile-time.

The use of RAII was also claimed (via comments) within this code base. This is a similarly appropriate, and even necessary pattern - every line that is read from the configuration file has the potential to be an erroneous one, and seeing as this code base handles it by printing to console and exiting, it is important that all resources created and files opened are released/closed respectively in the destructor to ensure everything is handled safely upon exiting.

See the **Implementation** section on commentary regarding the quality with which these design patterns were implemented.

Implementation

The implementation of the designs that were claimed to have been used were somewhat lackluster overall.

In regards to the Builder, while the general structure of one was adhered to in relation to the Director, Builder, Concrete Builder, and Product, its actual implementation defeated the purpose of a builder almost entirely. A builder's intention is to separate its parts and how they are assembled, but in this instance rather than being separated they have been coupled through the fixed number of arguments the *Game* object requires to be created. While the builder was intended to allow the simple creation of additional objects should the program be extended by simply adding a *.buildExtraPart()*, now the modification of the Game constructor is required to allow this - demoting it to the behaviour of a normal class initialised through constructors. This was apparent in Stage 2 as a new part of the game map needed to be bulit - the obstacles. Furthermore, to allow backwards compatability, it can't actually be included in the constructor, leaving a programmer to resort to using getters and setters, further straying from how a Builder should function.

RAII was not as crucial here as it may have been in a larger program - but as there is still I/O to/from files on disk, it is still a relevant and useful pattern in this context. RAII would dictate that a file is always opened to in the constructor and closed in the destructor of a class - and if multiple opens/closes are needed, then multiple classes are needed to accommodate this. However, the same file was both opened and closed within the constructor, and again in the destructor in this code base, violating the principles of RAII.

There are a few other points worth nothing regarding the configuration file as well. Firstly, while an option was provided to change the height of the game window, the code did not acutally account for this properly - any space in the window beyond the dimensions of the original fixed image show as white space, rather than stretching or tiling the image vertically. Erroneous values also provide similarly erroneous behaviour - incorrect image paths don't provide default values (leaving the screen blank), and invalid player properties such as size results in no player being rendered at all. The majority of the remaining configuration items are also not handled optimally, though with fewer consequences - the player's initial position is allowed to be negative (leaving it off-screen), and a negative velocity results in the background moving backwards but revealing whitespace.

Style

Style was consistent overall, with minor issues like occassional bracket inconsistency (newline vs same line). Naming conventions were also consistent throughout the code, if sometimes mildly uninformative - e.g. for the player, getSize() gave the height, rather than its full dimensions, so getHeight() and getWidth() would be created in its place instead. Constants (such as keys from the configuration file) were also scattered throughout the code, which would create extra work if they were to be changed - it would have been more appropriate to include a constants header to centralise the use of constant strings.