G52CPP 2014-2015 Formal Coursework Requirements

Summary: This coursework involves implementing the classes and functions that are necessary to produce an animated interactive program.

Functional requirements:

You need to meet all of the functional requirements and then your mark will depend upon how well you did in the various marking criteria. Each functional requirement which you miss will reduce your mark by 10%.

1: Change the base class to use a container class for the array of displayable object pointers.

The current BaseEngine class has a member called m_ppDisplayableObjects which is an array of pointers to displayable objects. This is not ideal since it is a static array. Modify the code in BaseEngine.h and BaseEngine.cpp to use an appropriate STL container class to store these pointers instead.

You must keep the external interface (i.e. function declarations) the same so that sub-class code does not need to be modified, it will just work. i.e. in many cases pointers will still be passed in and this must not change. Similarly, sub-class code will set the last array element to NULL.

Think carefully about the following and include a justification of your implementation choice for each of these decisions in your documentation.

a. Choose where to declare the container class object (e.g. as a global, member, etc).
b. Choose whether to declare a pointer (and create it with new) or the object itself (as a global object or a data member of that class type). E.g. “Array<…>* p; or Array<…> arr….; if you use an array.
c. Ensure that you accept the NULL terminator which sub-classes will attempt to set. Make a decision about whether to store this in the container or to calculate the size in some other way.
d. If someone stores 4 entries and sets the second to NULL, make a decision about whether to assume that the array now has one entry or three (in positions 0,2,3). Justify your decision and ensure that your code handles this consistently.
e. Make a decision about whether you will allow sub-classes to re-size this array after it has been created.

Expected time to implement this: ~2 hours to give you enough time to investigate what need to be changed, think about the best way to do it, make the changes and debug the code.

Note: I have made this the first requirement because doing this will make you have to understand how the object array works and see what you can do with it (e.g. notify objects). It also means that you could enhance the features if you wish. Thus, you may want to do it first.

2. Create your own BaseEngine sub-class and draw a background.

Create your own BaseEngine subclass, with a classname based upon your capitalised cs username with Main added to the end. e.g. if your username is abc001 then your classname should be Abc001Main.

Implement the code within your BaseEngine subclass to draw at least one background for your program. Note: you may want to draw multiple backgrounds for different states (see the requirement below), but as long as you draw at least one you can meet this requirement. Also see also the requirement for the tile manager below. Your background must be different from the demos (i.e. do not just copy the demo code without modification) and must not just be a solid colour. Particularly complex backgrounds which are hard (code-wise) to achieve (e.g. scrolling backgrounds) could add to your complexity mark and you should highlight this in your documentation, but you do not need to do a really complex background.

Expected time to complete: No more than an hour maximum, and probably only a few minutes to do a basic implementation.

3. Create and use the tile manager sub-class.

Create at least one tile based displayable region, as a subclass of TileManager and use it in your program. Name it with your capitalised username followed by the text TileManager. e.g. if your username was ghi003 then your class would be called Ghi003TileManager.

You must display your tiles on the screen at some point. If your idea of a program does not need a tile manager then you can use it just to draw a background somewhere, just to show that you understand the class and are able to use it. Hint: look at the demos you have available.

Expected time to complete: You may take an hour or so to fully understand this and implement something simple.

4. Create at least two displayable object classes for object which can move around the screen, without destroying the background.

Create at least two moving object classes as different subclasses of DisplayableObject, and use them in your program. You should name each with your capitalised username followed by a descriptive name. e.g. if your username is dfe002 and it was the player object then your classname would be Def002Player. (One could be a sub-class of the other if appropriate or you could have a common intermediate base class for them, or they could be separate.)

Meeting this requirement involves creating the object, making it appear on the screen and ensuring that when it moves around the screen the background is not obliterated/damaged. i.e. it is also a test of whether you implement the redraw and drawing within the regions for the displayable objects correctly.

Expected time to complete: For a simple implementation this will take very little time, but you will need to take care as you alter the objects to ensure that you do not break the redraw code.

5. Have a user-controlled object which can move around the screen under a user's control

You should have a facility such that at least one of the displayable objects is controlled by a player in some way, using the mouse, keyboard or both. There needs to be some interaction

between the user and at least one of your DisplayableObjects. e.g. the player could move something using keyboard or mouse, or click on something to change its appearance, etc.

Expected time to complete: For a simple implementation this will take less than an hour since you have examples of doing this.

6. Have an automated displayable object OR have some automated decision making somewhere else in the program

To meet this requirement you need to EITHER have at least one displayable object which is controlled automatically, e.g. an object which chases the player, which bounces around the screen or which moves randomly,

OR you can instead have some automated decision making elsewhere, e.g. to detect whether a player enters a specific area, to automate/animate the background in some way or some other facility.

Expected time to complete: This may take you a couple of hours to get something relatively complex, but something similar to the samples would take a very short time.

7. Load and save some information from/to a file.

You need to load some information from a file (e.g. score, level layout, text, game state) and save some information to a file (e.g. high score, game state). You can use either the C-style functions or the C++ classes (which will probably be simpler but you will have less experience with it). This could be as simple as a high score or could involve loading a screen layout (e.g. a game level). You must both save and load some data to meet this criterion.

Expected time to complete: This could take very little time for a high score, or take a couple of hours for a complex facility to save and load a game state.

8. Support different states

You need to support different states. You can do this using polymorphism or switch statements, it doesn't matter which. You must have states which are transitioned between under specific circumstances. E.g. a startup state which displays a message, a game playing state where things happen, a pause state and/or a 'win' or 'lose' state. You will need to clearly state in your documentation the conditions for state changes to occur (e.g. press Space to continue) and the background should not be identical in each state, so that they can be clearly identified.

Expected time to complete: Adding a simple switch to the important functions is very quick. Adding the state transition triggers (e.g. press SPACE) and making sure that everything works properly will be longer, but it should still take less than an hour to implement this requirement. Hint: look at demo 4.

9. Something must react and change

To meet this requirement you need to have something which identifies some state and change accordingly. Examples could be: parts of the background which change when something happens, e.g. an object moves over them or a timer goes off.

Expected time to complete: Again a simple implementation to detect that time has expired and change the background would take much less than an hour. A more complex implementation could do a collision detection between objects, or change the background (or tile manager shown on the background) in reaction to an object moving over it. This may take a little longer to implement and debug.

10: Complete the documentation

If you do not fill in the documentation file then you are likely to lose marks because the documentation file is your way to tell us what you have done so that we can give you marks for it. Making this a requirement means that we can explicitly penalise marks for people who do not submit a documentation file, or give partial marks for people who submit an unusable documentation file which does not give the necessary information.

Allow yourself enough time to check this at the end, but I suggest filling in much of this as you go along.

Marking Criteria

The quality criteria will be used for the marking, with each contributing a percentage of the mark. Once you have the total quality mark for these criteria it will be reduced by 10% (of the mark) for each of the functional requirements which you did not complete. You should be able to complete all requirements though, since you only need to do a simple implementation to get the functionality mark.

e.g. if you got a total quality mark of 80% but only did 3 of the 10 functional requirements then you will get a mark of 80 * 3/10 = 24%.

i.e. it is really important to meet all of the functionality requirements in at least a simple way.

Quality criteria

Code style and readability: 15%.

This factor considers elements such as:

- You used a consistent naming style for variables/classes/functions/etc (and followed the rules I gave earlier for the main classes, <username>Main and <username><objectname>.
- Source code files are named appropriately.
- Appropriate indenting is used for code.
- You used comments appropriately.
- Unusual elements of the code are all labelled with explanatory comments. E.g. switch statements which deliberately do something then drop through to the next statement rather than having a break statement should be labelled to make it clear that this was not an error (e.g. add a comment line ‘deliberate drop through’ or ‘intentionally missing break’). i.e. If your code does something unexpected then make it clear with a short comment.
- You used const appropriately. E.g. functions which should be const really are.
- You used functions and loops to avoid copy-pasting code.

This factor also, importantly, covers how easy it is to understand what your code does: although algorithms could be complex, any difficulties in immediately understanding them should be due to algorithm complexity rather than badly structured code, and comments should help with this.

Example marks would be 15: no problems. 10: minor issues or inconsistencies. 5: We had problems understanding parts of your code or it was really messy. 0: Your code is really hard to understand compared with how clear it could have been to do the same thing.

Efficiency: 15%.

In general it is easier to identify inefficient code, so here are some examples of inefficiencies which would lose you marks in this category:

- Constantly creating and destroying objects (on the heap or stack) which could have appropriately been kept and reused.
- Unnecessary looping.
- Redrawing static things to the foreground which could have been easily drawn to the background.
- Creating unnecessarily large arrays or data structures.
- Repeating things many times unnecessarily.

It should be relatively easy for you to avoid introducing inefficiencies I hope, so you should be able to score well on this one fairly easily.

Example marks would be: 15: no problems found. 10: minor problems found. 5: multiple minor problems or a significant efficiency problem found. 0: It's really inefficient in one or more areas.

Robustness, Compilation, Correctness and Reliability: 30%

Basically this means whether your program works and whether it avoids crashing.

Compilation warnings will also reduce this mark, with the amount depending upon the severity of the problem that the warning is describing.

If your program crashes then expect a really low mark (probably 0 but maybe higher if it's under specific unimportant circumstances). The reduction for crashes will depend upon how often it happens and possibly why (e.g. how much of an understanding of C++ problem does the crash show).

Finally, you can lose marks in this category for display errors, redraw problems, logic errors, incorrect functions, etc, which show problems with the program.

Program complexity: 30%

This important factor measures how much complexity you added to the project. You do NOT need complexity in all of the functionality, but you do need it somewhere and the more complex this is and the more you have shown of your understanding of C++, the higher mark you should expect. I would think that this and the impact are the criteria which will most differentiate the marks between someone who does a good coursework and someone who does a very simple one. It should be easy to make an efficient and reliable program, but getting it to do something complex is a lot harder.

Note that this measures the complexity of the task you solved, not necessarily the complexity of the code (although clever code can add to the mark). If you solve a simple problem in an extremely complex manner then don't expect complexity marks for it unless you can show that it was a reasonable thing to do. i.e. if you write unnecessarily complex code to do a simple thing then you won't get extra complexity marks (and may lose efficiency or readability marks).

This is to encourage you to show how good your C++ coding skills are, getting your program to do more than just the basic things.

Example marks:

0 means that you did a basic implementation of everything. (Yes, this really does mean 0 for this criterion if you do a basic implementation, but you will pick up the marks elsewhere.)

10 means that you made an effort to do something non-trivial and it worked.

20 means that there's a reasonable level of complexity to your project, some of which would not have been obvious to us how to do.

30 means "Wow!" and that this impressed us with how complex a task you achieved.

Important: This measures the complexity of what you had to achieve, now how complex a way you did it. If you do a really (unnecessarily) complex implementation of something which should have been easy to do in a slightly different way then you won't get the complexity marks. So ensure that it really is a complex task, don't just try to make an easy task look hard.

Impact: 10%

How impressed were we with your overall program? Expect to get marks around the range of 10 for ‘wow, it’s beautiful and works really well’ to 0 for ‘it looks ugly and doesn’t seem to do much’. This assesses how good the appearance is, how well it works, and basically our general impression of the quality as far as being a finished, polished C++ program.

Submission

Keep the name of your documentation file as CPP_CW2015.docx and ensure that it is in the top level of your project directory (the directory called CPP_CW2015 with the CourseworkBase.sln file in it).

Run clean on your project to delete intermediate/temporary files.

Zip up your CPP_CW2015 directory (including all sub-directories and files, and your documentation file).

Submit your zip file following the instructions which will appear in the coursework section of the the moodle page when submissions are opened. Note: your coursework will be submitted through the same system which you have used to submit automatically marked coursework in the past, but it will be marked manually, not using the coursework code marking system. Unfortunately the moodle submission system hides some information which is needed for me to send you the feedback via email, so using this system removes these unnecessary barriers. At the time of writing this, the system is undergoing a UI overhaul to make it more user friendly, so I will provide details when this has been completed.

Note: The submission system will verify that the CPP_CW2015 directory exists in the zip file and that the documentation file exists (with the correct name) and at least the src and SDL directories exist within that directory.

The standard University late mark system applies: your mark is reduced by 5 percentile points per working day (or part of a day) late that you submit. For example, if you submit 3 days late, a mark of 50% would be reduced to 35%. Your mark cannot go negative, so 0% is the lowest mark in this case. ***Submissions will not normally be accepted more than 5 days late.***

If you submit late then you accept that you may not get the feedback within the 21 working days from submission and may not get it before your exam.

Testing your submission

You are advised to test your submission in the same way that we will mark it:

Unzip your zip file to a different directory than the one it was in before. (This is to check that you have not included any absolute rather than relative paths in your project.)

Recompile the whole project (use the 'rebuild' option). Check for (and remove) any warnings or errors. Note any minor issues in your documentation file if you can't fix them.

Test that it works. Note any issues in your documentation file if you cannot fix them.

Check that your documentation file is OK. We will use this to test and mark your work.

We will test your coursework on a machine equivalent to the A32 lab PCs, so please test on these yourself. (This may mean that we use the A32 lab machines, or that we use another machine and only use the A32 machine if we find problems to verify that it is not a problem on our system.)

General notes and answers to questions:

The following (.h and .cpp) files count as Framework Base Classes: BaseEngine.h, BaseEngine.cpp, DisplayableObject.h, DisplayableObject.cpp, FontManager.h, FontManager.cpp, JPGImage.h, JPGImage.cpp, TileManager.h, TileManager.cpp, MovementPosition.h, Templates.h

You should utilise the supplied framework base classes, using sub-classing as necessary and overriding the methods which you need to change.

Unless absolutely necessary, you should not change the base classes except for the change to use STL container classes. This is to show your ability to work with an existing class library. If you change it then you need to justify why with a very good reason and any changes should be generic (i.e. the changes must work with any sub-class we have for testing, not just your own program). If you believe that changes are absolutely necessary then please discuss any changes with Jason first and have them approved before submission.

You should leave the demo files in your project and they should compile and run correctly when we test your submission. I suggest that you test these again after making your changes.

You must not use classes/functions other than those in the framework/examples (including the supplied SDL dlls), the standard class libraries, and your own code. In particulat, you should not make use of any third-party libraries other than the standard C++ library and the SDL libraries that are already included within the supplied framework.

Note: The standard C library is part of the standard C++ library, so you may use any function that we saw in the earlier lectures, or any of the C++ classes which you research and find in the standard C++ library.

The Standard Template Library (STL) is also a part of the standard C++ library, so you may use it if you wish, but we will not have covered it in lectures by the time you will be doing the coursework.

It will make no difference to your mark whether you use the C++ classes or the C library functions. You may find that the C++ classes do more of the work for you, but you would need to learn how to use those which you have not seen them in lectures.

All of the source code must be your own work, however you have permission to freely copy and modify code from the samples and demos that I gave you. You must not copy code from any other source, since the purpose of this exercise is for you to practice programming and for us to assess your ability to do so.