G52CPP, 2019/20, Coursework Parts 3 and 4, GUI Framework

Overview

Coursework 3 aims to introduce you to a C++ framework for building GUI programs. **Coursework 4** aims to give you the freedom to do more with the framework and potentially create something impressive.

The coursework framework is a simplified cut-down framework covering just the essentials and aims to simplify the process of software development for you. You will need to understand some C++ and OO concepts to be able to complete the coursework, but will (deliberately) not need to understand many complex features, so that the coursework can be released earlier in the semester.

There are two demo tutorials which you should work through to help you to understand the basics.

There are then a variety of coursework examples, which are deliberately divided into different levels of difficulty, so that if you are not so confident with C++ you can ignore the complex ones but still get a good mark on the coursework and still get the C++ programming experience you will need, whereas if you are finding things easy you could create something truly impressive.

Please see the Issue Sheet and moodle page for deadlines.

Coursework 3 will be marked in labs (it is designed to be fast to mark) and you will know your mark immediately. A zip file of your project (including all source, project and resource files) MUST be uploaded to moodle in order for your mark to remain valid – failing to upload your project will count as a failed submission and result in a mark of 0. Except when extenuating circumstances apply, no late submissions are permitted, however early marking and submissions are encouraged.

Coursework 4 requires a submission to moodle by a deadline and is then marked in labs or demo times. This allows more time for marking while ensuring that everyone has the same amount of time to complete the coursework (it is the component that may need more thought and/or time).

IMPORTANT: Coursework part 4 is designed to be hard! You may not be able to do all parts. Please set yourself a time limit of 20 hours after completing coursework part 3 (for a 20% coursework), do what you can and then stop (making sure it compiles and runs etc). DO NOT SPEND TOO LONG ON CW part 4!

Getting started:

1. Download the zip file of the coursework framework. Unzip it and open it in Visual Studio. (See tutorial on getting started.) Compile and run the initial framework to ensure that it compiles and runs with no problems on your computer. Read the getting started document if you are stuck.
2. **Do the framework exercise A** and ensure that you understand about the basic features of the BaseEngine class and drawing the background, including the tile manager.
3. **Do the framework exercise B** and ensure that you understand about the basic features of simple displayable objects.
4. Experiment with the framework. Try changing which object type is created in the mainfunction.cpp file (change which line is commented out) and look at the various demos.
5. Start on the requirements below, using what you learned from demo tutorials A and B. Coursework part 3 should not be too bad using the walkthroughs for demo A and demo B and the samples.

Please read the issue sheets for CW3 and CW4 – they contain important information.

General requirements:

You MUST meet the following general requirements:

1. **All of the code that you submit must be either your own work or copied from the supplied demos. You may not use work from anyone else. You may NOT work with other people to develop your coursework. You may NOT have someone else write any or all of the code in your submission. You will be required to sign a simple documentation sheet making clear that you understand this.**
2. **Create a program which runs without crashing or errors and meets the functional requirements** listed below and on the following page. Look at the Hall of Fame page to see some examples of previous work and decide on a basic idea for your program. You don't need to implement that much for part 3, but part 4 will involve finishing it.
3. **Your program features which are assessed must be different from all of the demos** and from exercises A and B. i.e. do not copy demo code or the lab exercises code too closely. You will not get marks for copying code, but would get some marks for your own modifications which show your understanding of C++ if you did *similar* things to the demos.
4. **You should not alter existing files in the framework** without prior agreement with from the module convenor (and a good reason) – the aim of the coursework is to work with an existing framework not to improve it. (Note: subclassing existing classes and changing behaviour in that way is permitted, and encouraged.)
5. The aim of this coursework is the investigate your ability to understand existing C++ code and to extend it using sub-classing, rather than to use some alternative libraries:
   - **Your program MUST use the supplied framework in the way in which it is intended**. i.e. the way in which demo tutorials A and B use it to draw the background and moving objects is the way that you should use it.
   - **You must not alter the supplied framework classes.** It is a good exercise to use existing classes and potentially sub-class them to extend or change behaviour rather than changing the original class. E.g. with many commercial classes you may not have the ability to change their code. I tried to make it possible for you to change the behaviour you need to by using sub-classing.
   - **You should not use any external libraries other than standard C++ class libraries**, and libraries which are already used by the framework code, unless a requirement specifically says to do so (e.g. sound). There any many useful libraries which you could use to improve your programs, but the point of this exercise is to show that you can understand and use the library I supply.
6. **There are 10 functional requirements in part 3**, worth 1% of the module mark each, for a total of 10% of the module mark for this part of the coursework.
7. When you have completed the coursework part 3, demo your work to a lab helper and have it marked. Then:
   - **Clean your program** (use the Clear Solution option under the build menu – to delete the temporary files.
   - **Delete the hidden .vs folder in the root of your project directory**. This can become huge, is not needed, and will make your zip file unnecessarily bit.
   - **Zip up the rest of your project directory** and submit it to part 3 coursework submission on Moodle.
   - **Your submission should include all of the supplied files** (including demos) as well as your new files.

It should be possible when you have done this for someone to download the zip file, unzip it, run rebuild all to rebuild it, and run it, with no other changes.

8. **There are a number of requirements in part 4 which are worth 1 or more marks each. These are usually a lot harder than the part 3 requirements. Start with the easy ones! Limit your time on part 4 to a maximum of 20 hours!** Read the requirements carefully to see which are optional and which are compulsory.
9. When you have completed part 4 upload the project file to the CW4 submission on moodle, as you did for CW3 (i.e. clean it, delete the .vs directory and zip the folder). A demo will be arranged after submission. For some marks you will also need to submit a video of the running program. Finally, you will need to complete and bring along the documentation file, since it tells the markers what you want marking (i.e. which requirements you did). You will leave this sheet with the marker.

Some clarifications (based on questions in previous years):

The key priority is that you understand the code. The assessment aim is to **test your ability to understand C++ code and to write it** – not to copy paste. Here are some examples of things which are and are not allowed as far as marking is concerned:

- If you take a demo and just add some changes then you are assessed only on your changes. i.e. if maze demo already did something (e.g. background drawing, pause facility, etc) then it doesn't count for a mark for you because you did not do it. Similarly, it does not count as you creating a new subclass because you didn't. Please start from a new class and add your own code to it.
- If instead you write your own code, and take from the demos the understanding of how they do it – possibly even copying some parts of the code, *but showing your own understanding of it when you do*, then that is OK. (This is partly why we ask you to demo it – so you can answer questions if necessary.) E.g. if you copy the way that pause works and implement it correctly then you would understand how it works and be able to explain it fully when asked about it.
- As another example, if you copy whole functions, such as a draw function, from a demo then that is a case of something being the same as maze demo. If you do so and make a slight change to make it a different colour then it's basically the same apart from that slight change, so **it is not different enough. Don't do it.**
- However, you CAN copy small parts into your own classes, showing your understanding of them. In which case you should be able to explain fully how they work if asked during the marking – that is an important reason that we ask you to demo your work, so that we can test your understanding.

General marking criteria for both CW3 and CW4:

You will lose marks if any of the following apply. In each case we will apply a percentage mark penalty to your mark.

- Your program crashes on exit or has a memory leak. (Lose 10% of your mark.)
- Your program crashes at least once during its operation. (Lose 20% of your mark.)
- Your program crashes multiple times. (Lose 30% of your mark.)
- Your program crashes frequently. (Lose 40% of your mark.)
- Your program has some odd/unexpected behaviour/errors. (Lose 10% of your mark.)
- Your program has a lot of unexpected behaviour/errors. (Lose 20% of your mark.)

The mark sheet that you take to the marking for parts 3 and 4 has entries for the above and your marker will annotate it according to their experience of your demo of your software.

Coursework Part 3 Functional Requirements

Functional requirements: (1 mark each, marker will tick off all that you have done on their mark sheet).

Note: the markers will use the literal text below to determine whether a requirement was met or not. If I need to clarify any of these criteria I will do so in the FAQ document and let you know that it was updated.

In general the requirements below get progressively harder (in that they need more understanding of the framework and C++ to do the later ones) but please do check later ones even if you decide some earlier one is too hard for you to do.

1. **Create an appropriate sub-class of BaseEngine with an appropriate background which is different from the demos**. Create an appropriate new sub-class of the base engine. Name your class using your capitalised username followed by the text Engine. e.g. if your username was psxabc then your class would be called PsxabcEngine. You MUST create a new class – you must not just change/rename one of the existing demo classes.
2. **Show your ability to use the drawing functions:** draw your background ensuring that you use at least one of the shape drawing functions to draw on the background and that you draw at least one image to the background, which is different from the demos and shows your understanding. Be prepared to explain what you have done to the marker if asked. A blank background will not get this mark – even if you change the colour – you MUST use one of the shape drawing functions (i.e. a drawBackground…() function other than drawBackgroundPixel()) and at least one image, to show your understanding.
3. **Provide a user controlled moving object which is a sub-class of DisplayableObject and different to the demos:** Have a moving object that the user can move around, using either the keyboard OR the mouse (or both).Note: you could have an indirect subclass of DisplayableObject if you wish (i.e. a subclass of a subclass of DisplayableObject). The aim of this requirement is for you to show that you understand how to use EITHER the keyboard or mouse input functions (or both) as well as to show that you can create a moving object. Be prepared to explain how this works to the marker if asked.
4. **Ensure that both keyboard and mouse input are handled in some way and do something.** The starting point is probably to look at whether you used mouse or keyboard for the moving object above and just use the other one here. E.g. if you user-controlled object is mouse controlled then make it so that something happens when a key is pressed (e.g. a counter is incremented, which appears on the screen – see requirement 10). Or if your object is keyboard controlled, you need to handle mouse movement or button pressing.
5. **Provide an automated moving object which is a sub-class of DisplayableObject and different from the one in requirement 3:** Have another moving object (separate to the user-controlled one, with a different class) whose movement is not directly controlled by the player, and which acts differently to the objects in the samples/demos/code that I provided. i.e. show that you understand something about how to make an object move on its own (changing its x and y coordinates and redrawing it appropriately). Be prepared to explain how this works to the marker if asked. Your object must have some behaviour that is different to the demos (i.e. a copy-paste of the demo code is not sufficient) and you must be able to explain how it works and justify that it is different from the demos in some way.
   Your object must have a different appearance to the object in requirement 3, and look different to the demos/samples as well.

6. **Draw some text on the background.** Draw some text onto the background (not foreground) and ensure that the text is not obliterated when moving objects move over it. Ensure that moving objects can/do move **over** at least part of this text, so that the object appears in front of the text, and demonstrate that it is redrawn after the object has moved. i.e. show that you understand how the background and foreground are drawn to and when to draw text. Be prepared to explain how this works to the marker if asked.
7. **Have some changing text, refreshing/redrawing appropriately which is drawn to the foreground (not background), in front of moving objects.** This text may change over time (e.g. showing the current time, or a counter) or could show a score which changes, for example. It could also be drawn to the foreground as a part of an object (e.g. a moving label) if you wish, but does not need to move around with objects if you don't want it to. When the text changes, the old text should be appropriately removed from the screen. Be prepared to explain how this works to the marker if asked. This shows your understanding of drawing text to the foreground.
   The text has to be drawn such that moving objects would move under it rather than on top of it though. i.e. not to the background, and basically it means it'll be drawn after at least some of the objects. For marking we will check the code where it is drawn if there is any doubt. E.g. is the function which draws it called before or after drawing objects. (Look at the different functions to see which to use – the point of this criterion is to see whether you realised the difference between these.)
8. **Create your own subclass of TileManager**. Create a subclass of the tile manager which has different behaviour (at least a little) to the demos. Name your class using your username followed by the text TileManager. e.g. if your username was psxabc then your class would be called PsxabcTileManager. Be prepared to explain the difference(s) from the demo versions to the marker. Hint: Look at the existing demos, including the bouncing ball demo. Display the tile manager on the background, so that the tiles are visible. It must be different from the demos but can still be simple. You are just showing that you understand how to use the tile manager class. Be prepared to explain how this works to the marker if asked.
   Use a different tile size and number of rows and columns to the demos (e.g. 13x13 tiles and 6 rows and 9 columns if you can't think of some numbers yourself).
   Your TileManager must not be a copy of an existing one, or just an implementation of the demo tutorial example without change. i.e. you must show some understanding of how to do this, not just blindly repeat the steps of demo tutorial A.
9. **Have at least one moving object interact correctly with the tile manager, changing a tile:** have one object which when it moves over specific tiles the tile changes. Consider the bouncing ball demo if you can't work out how to do this from the information in demos A and B. The key part which will be assessed is that you appropriately detected the tile that the moving object was over and handled it appropriately. See BouncingBall demo for an example of this, but you need to have at least some difference in your behaviour from this. You may NOT just copy-paste the code from BouncingBall, but you CAN use the information of how to do this from BouncingBall and produce a similar example.
10. **Have at least two moving objects interact with each other:** this means that at least two of your objects should react to each other. Hint: look at CollisionDetection.h if you don't want to do the maths for rectangle or circle intersection yourself – and see MazeDemoObject.cpp for an example of using these functions. Assessment of whether you achieved this or not will be on the basis that the intersection of two objects is correctly assessed and something happens in reaction to this (e.g. objects move, change direction, something else changes (e.g. a score) etc).

Coursework part 4 functional requirements

Up to 8 marks for compulsory requirements, and up to 12 more for optional requirements. Total 20%.

Part 3 ensures that you know the basics of using the framework. Part 4 gives you the opportunity to innovate to make something really effective.

For purposes of moderating marks, some of these requirements mean that you need to also upload a short video of your running program, so that it can be seen running. You do not need to upload the video if you do not try to get these marks.

Mandatory/compulsory requirements: you MUST be assessed on these and can get up to 8 marks:

1. **Add states to your program (0 to 2 marks).** This means that your program correctly changes its behaviour depending upon what state it is in. Each stage should have a correctly drawn background which is different to the demos, and are is trivial (e.g. a blank background). This could use an image, a tile manager or be drawn using the fundamental drawing functions on the base engine. You can get a variety of marks for this:
   1 mark: You provide at least a start-up state, a pause state and a running state. Both the appearance and behaviour must both change in some way between states.
   2 marks: (advanced mark) You must provide at least 4 states, including a pause state, implementing the state model using subtype polymorphism rather than if/switch statements. Look up the 'State Pattern' to see what this means and think about it. It's an advanced mark so we will not explain how to do this beyond the following: there will be a basic state base class and a subclass for each of the different states. Your BaseEngine sub-class will need to know which state object is currently valid and the different methods will call virtual methods on this object. The different behaviour will therefore occur due to having a different object being used for each state rather than having a switch in each of the methods. If you have if or switch statements specifying what to do in different states then you won't have done in properly so you won't get this mark. You should NOT have more than the one sub-class of BaseEngine if you do it correctly. If you had to create multiple sub-classes of BaseEngine then the implementation is wrong (and there will be other issues since you will have more than one window as well).

2. **Save and load some non-trivial data (0 to 3 marks).** You can use either C++ classes or the C functions to do this – your choice will not affect your mark. You can get a variety of marks for this:
   1 mark: saving/loading of at least one value to a file, e.g. a high score (e.g. a single number to a file)
   2 marks: completed the saving/loading above, and also load some kind of more complex data, e.g. map for multiple levels, or formatted text documents where the formatting will be interpreted. The main criteria are that it must be multiple read/write operations and something must change depending upon what is loaded (e.g. set tiles according to data read and/or set positions of moving objects according to the data).
   3 marks: (advanced mark) completed the above but also save/load the non-trivial state of the program to a file. A user should be able to save where they are (e.g. the current document that they are working on or the state of the game – saving ALL relevant variables for all objects) and it should be possible to reload this state later, with everything acting correctly, continuing from where it was. Note: this means saving/reloading the positions/states of all moving objects as well as anything like changeable tiles, etc. You will need to provide some way to reload from the state as

well – e.g. when the program starts or in response to some command from the user (e.g. pressing S for save and L for load). This is meant to be non-trivial and may need some thought to make it work properly. **IMPORTANT: to get 3 marks for this you MUST also upload a short video demonstrating your program to the moodle video submission.**

3. **Interesting and impressive automated objects (0-1 mark):** To get this mark you must be using multiple displayable objects from at least three displayable object classes, with different appearances and behaviour and you should have an intermediate class of your own between the framework class and your end class, which adds some non-trivial behaviour. i.e. you are showing that you can create a subclass which adds some behaviour, and some other subclasses of that which add more behaviour. At least one class should use an image and you should load the image only once and keep it in the DisplayableObject which uses it, NOT keep reloading it each frame if you want this mark. **IMPORTANT: to get this mark you MUST upload a short video demonstrating your program to the moodle video submission.**
   1 mark: you meet the wording above

4. **Impact/impression (0 to 2 marks)** These two marks are awarded based upon the overall impact/impression of the program. Many people will just do the minimum and will not get these marks. These marks will take some effort to attain and are designed to ensure that the people who are most capable with C++ get higher marks. Be careful not to spend too long on this coursework if you are struggling!
   1 mark: (advanced mark) This is non-trivial. You made an effort to ensure that it is beyond the minimum to just tick boxes. You had at least three moving objects, at least one is user controlled and you accept both mouse and keyboard input in some way. Your program should work smoothly and looks good. E.g. you use images and drawing primitives appropriately, have appropriate states, everything looks good, etc. As long as you completed part 3 of the coursework, this basically just means that you made an effort to make it look good and work well. If the marker looks at it and thinks 'looks nice' and when it is used it works properly and well, with no problems, you will get this mark. As a minimum this means: you made some effort with the graphical appearance, the background is relevant and not plain or the same as any of the demos, including at least some use of relevant shapes (e.g. separating off a score by putting it in a box and labelling it) and/or images, and moving objects are not just plain circles or squares.
   2 marks: (advanced mark) Meets all criteria for 1 mark and is also very impressive. It works very well doing some complex tasks and you obviously made a significant effort on this coursework. If it's a game it's fun, interesting to play or a complex program performing a useful task (e.g. a drawing package or word processor?) and has at least 2 different levels (to demonstrate your ability to do this). If it's a useful program then it achieves its purpose well. It should be impressing the markers to get this mark. **IMPORTANT: to get 2 marks for this you MUST upload a short video demonstrating your program to the moodle video submission.**

(See next pages for the optional requirements)

CW4 optional requirements: You can get a maximum of 12 marks from these and can decide which to do, so do not do more than 12% worth of these. Note that there are more than 12 possible marks (16 at the moment) for this section so you don't need to do all of them and should not attempt more than 12 marks! Some are deliberately hard because we have been told that we must include hard aspects which not all students could achieve on courseworks.

A: Correctly implement scrolling and zooming using the framework's FilterPoints class (1 mark). To get this mark you need to demonstrate your understanding of how the FilterPoints class is used in the demos to implement scrolling and zooming. Be prepared to answer questions about this. The aim is for you to work it out from looking at the code, so we will NOT explain to you how the code works.

1 mark: Correctly met the wording above.

B. Have advanced animation for background and moving objects (1-2 marks): To get this mark you need to show your ability to create smooth animations for both some part of the background and at least some of the moving objects. Note: there are various demos which show you how to do elements of this, but you should not just switch between two images, it should look smooth. One person in a previous year had flickering torches animating the background and animated characters moving around, but there are many ways you could do this. **IMPORTANT: to get this mark you MUST upload a short video demonstrating your program to the moodle video submission.**

1 mark: showed understanding of how to animate background so it changes over time AND how to animate moving objects.

2 marks: animation of both objects and background is smooth and visually impressive.

C. Interesting and impressive tile manager usage (1-2 marks): To get this mark you must be using a tile manager, and must have multiple displayable objects. Your tile manager must draw a number of appropriate and different pictures (either using images or the drawing primitives) for the different tile types, which are not just different colours. You should have at least 5 different tile types (not just different sizes ovals/rectangles). At least one tile must be drawn using an image. You should load the image only once and keep it in a relevant object, NOT keep reloading it each frame if you want these marks. If you use multiple images then you could also meet the animated appearance of automated objects criterion, and/or the animated background if you do it appropriately. **IMPORTANT: to get either of these marks you MUST upload a short video demonstrating your program to the moodle video submission.**

1 mark: You meet the wording above.

2 marks: The marker is impressed by what you have done, it looks great, works well and has some interesting behaviour. Moving onto some tiles would have effects elsewhere, e.g. standing on a key tile visibly unlocks a door tile?

D. Creating new displayable objects during the game (1 mark): meeting this requirement means that you can dynamically add one or more displayable objects to the game temporarily after it has started and that this works correctly. These could disappear again after a while. For example, add an object which appears and moves for a while for the player to chase if a certain event happens. Adding a bomb that can be dropped, or a bullet that can be fired also would meet this requirement if you implement these as displayable objects. Something like pressing a key to drop a bomb which then blows up later, while displaying a countdown on the bomb, and then changes the tiles in a tile manager would meet a number of requirements in one feature.

Hint: This does NOT mean that when you change states the objects are created and when you change back they are destroyed. That would not count. The idea behind this requirement is that moving objects appear and disappear while using the program within the state.

You could do this by setting objects visible or not, or by changing which values are in the displayable object array. In theory you could destroy and recreate the array, as long as you copy the other objects over

properly. (It may take you some thought to work out how to do that.) Usually the key is to make the array plenty big enough to start with so you don't need to resize it.

Remember: if you make objects invisible you may also need to ensure that their own and other DoUpdates ignore the objects. e.g. just because you don't draw an enemy may not stop the player dying if they move over the enemy.

E. Allow user to enter text which appears on the graphical display (1 mark): For example, when entering a name for a high score table, capture the letter key presses and pressing of delete key, and show the current string on the screen (implementing delete at well may be important). This needs thought but is useful to demonstrate your understanding of strings. I added this optional requirement because some people did it anyway last year for entering high score tables and I wanted the marking scheme to reflect that it was done by giving a mark for it. Entering text into the console window does not count for this.

The key requirements here are:

- Capture the key presses for letters/characters.
- Store the key presses somewhere
- Capture delete key press and handle it appropriately
- Display the text on the screen

F. Complex intelligence on an automated moving object (1 or 2 marks): As a minimum this criterion would involve something more than moving randomly or homing in on a player.

1 mark criteria this could involve something like 'if player is on the same column then home in, otherwise move randomly', or 'keep the same direction until I bump into something then change direction towards player and repeat'. Simple equations and decisions would get this mark not the 2% mark.

2 mark criteria mean a really good implementation of the intelligence of a moving object, e.g. to use a shortest path algorithm to find the shortest way through a maze to get to a player, or predicting a player's path and moving towards that rather than the player itself, or showing some apparent intelligence. Note that the important thing for marking here is the skill you show in your C++ ability by implementing this – so something trivial will not count. The 2% mark requires some non-trivial algorithms to determine what to do, showing your C++ coding skill.

For this mark you need to outline in the documentation sheet what your intelligence does that is so clever.

G. Non-trivial pixel-perfect collision detection (1 mark): If you implement some really complex collision detection, such as complex outline interactions (e.g. someone did bitmap-bitmap interaction in the past, checking for coloured pixels interacting, and someone else split complex shapes into triangles which could be collision detected and checked every triangle interaction) then you get this mark. Note that this is hard to get, so if your implementation is not solving a really complex task then you probably will not get this mark. Using the supplied collision detection class is not sufficient to get this mark. Collision detection for rectangles and/or ovals is not sufficient. **IMPORTANT: to get this mark you MUST upload a short video demonstrating your program to the moodle video submission.**

1 mark: pixel-perfect collision detection on a complex shape.

H. Image rotation/manipulation using the CoordinateMapping object (1 mark): for this mark show that you understand how to create a new CoordinateMapping object which acts differently to the existing ones and does something at least slightly differently to the existing examples.

1 mark: show your understanding by creating and using your own CoordinateMapping object to draw an image. Your object must do something different enough to the provided ones to allow the marker to see that you understand how to use it fully.

I. Integrate sound using SDL (1 mark). This has been added because various people ask me about sound each year. If you want to download additional SDL sound libraries you may, but it should work without I think. This isn't trivial and means you have to show some understanding of using structures and pointers, so you can have one mark for doing so. You should use only the SDL libraries and it should appropriately interact with the existing framework.

1 mark: correctly got sound working while using only SDL functions.

J. Show your understanding of templates and/or operator overloading (1 mark). To get this mark you should use template functions of template classes in some non-trivial, appropriate way, or use operator overloading for some operator other than = and == appropriately, creating the overload on an object and using it in an appropriate manner.

1 mark: created and used a template class or template function appropriately OR overloaded operators (other than = and ==) and used them appropriately.

K. Use your own smart pointers appropriately (1 mark), in a manner which illustrates your understanding of how to use these. This means not just using SimpleImage, but creating your own and using them appropriately somehow.

1 mark: Example which illustrates that you understand how to use smart pointers in an appropriate manner.

L. Sellable quality (1 mark): This is supposed to be hard to get and basically means that your 'impact' of your program was even more than the 2 marks value of the impact/impression mark. It basically means that the marker and Jason both went 'wow' when they saw this and thought that people would easily pay money to buy this program. **Probably not one to aim for** since any program meeting this will already get the advanced marks elsewhere I would expect, but it gives us the opportunity to comment 'wow!' about your program. In some cases it is possible that a program could be sellable while not implementing too many of the other features, since some app store programs are relatively simple, for example. **IMPORTANT: to get this mark you MUST upload a short video demonstrating your program to the moodle video submission.**

M. An advanced feature I didn't think of but you had pre-approved (1 mark): I hope that this is rare and that most features fit the earlier criteria, but this gives you the opportunity to do something else really complex that I didn't think of and still get a mark. <u>You should discuss any advanced feature with Jason in advance and get approval</u> for it to be able to gain a bonus mark if appropriately implemented. Without approval in advance you don't get the mark! Any feature which is not pre-approved will not be eligible for this mark. Note that to be eligible the feature needs to illustrate C++ knowledge and/or ability to work within the supplied framework and should not be simple/trivial to do. E.g. an extraordinarily complex (in terms of C++ code) feature to do something I didn't think about. *I repeat: to get this mark you MUST have agreed it with Jason in advance and he will not always agree things that you suggest if they do not show the required skills/abilities. In general these features should be AT LEAST as complex to do as the other requirements – i.e. these are for flexibility, not easy marks.* **IMPORTANT: to get this mark you MUST upload a short video demonstrating your program to the moodle video submission.**

I have left two extra spaces on the marking sheet beyond mark M in case you think of more than one advanced feature. You must have each pre-approved in writing so that Jason can inform the markers before the demos. This involves you explaining what you will do (or have done) and get approval.

Note: If there are common things which could apply to many students then I may update the requirements, describing them here, to make it clear that anyone could do this for a mark.