COMP2006, 2020/21, Coursework Requirements: GUI Framework

Overview

This coursework has two parts – an initial 'easier' part to get you working on this earlier, and the final part which is more complex and lets you innovate more. Your part 2 coursework can be completely different and separate to your part 1 – you do not need to build on the part 1 submission to make part 2, although doing so may save some work.

Part 1 aims to introduce you to a C++ framework for building GUI programs. **Part 2** 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. It may seem large to you but in fact is very small compared with the sizes and complexities of class libraries you would usually be working with in C++. 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 initially, so that the coursework can be released earlier in the semester.

There are two demo tutorials (labelled Demo A and Demo B) which you should work through to help you to understand the basics. Please complete these to understand how to do part 1. There are also some examples in the coursework code provides, to illustrate how you can do some things. Your submissions must be different enough from the demo tutorials and supplied examples that it is clear to the markers that you actually understand the concepts rather than just copying them.

Part 1 will be marked via a demo in lab times (it is designed to be fast to mark this way) and you will know your mark immediately. (You should be able to check the marking criteria in advance and know what mark to expect anyway – hopefully full marks for each of you.) 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. As long as you submit by the deadline you may demo your work first before submission if that is easier for you, but you do also need to submit the files. Failing to demo your project will also result in a mark of 0. Except when extenuating circumstances apply, no late submissions are permitted, however early marking and submissions are encouraged.

Part 2 is more advanced and is designed to be harder to complete. Part 2 is designed to be hard in some parts! You may not be able to do all parts. Please set yourself a time limit of 30 hours after completing coursework part 3 (for a 30% coursework), do what you can and then stop (making sure it compiles and runs etc). DO NOT SPEND TOO LONG ON CW part 2!

Part 2 requires a submission to moodle by a deadline and is then marked in specified demo times (currently expected to be via MS teams screen-sharing). Submission of documentation and code (and any extra demo videos) will be via moodle. This allows more time for marking while ensuring that everyone has the same amount of time to complete the coursework (this part may need more thought and/or time).

Getting started:

1. **Download the zip file of the coursework framework. Unzip it and open it in Visual Studio** – on your own computer or the windows virtual desktop. 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.

   (Mac,Linux) Optionally: If you prefer then you can follow the instructions provided on moodle to build on Mac or Linux. We are assuming that, if you wish to do this, it is because you are the expert on Mac or Linux so will be able to resolve any issues. This should ensure that those who are more comfortable on a different operating system are able to use it, but please be aware that we may not be able to support issues on these platforms.

   Whatever you run this on, you will need to be able to screen share on teams to demonstrate your work for marking as well as to submit your source code files to moodle. (It should be possible for you to demo on any platform you wish and demo on the windows virtual desktop if necessary by copying files back into the visual studio project.)

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. **Start on the requirements for part 1**, using what you learned from demo tutorials A and B. Coursework part 1 should not be too bad using the walkthroughs for demo A and demo B and the samples. Try changing which object type is created in the mainfunction.cpp file (change which line is commented out to change which demo to run) and look at the provided demos.
   You may want to consider the 'SimpleDemo' demo, for things like string displaying, and potentially either the BouncingBall or Maze demos if you can't work out things like the TileManager. Work on only the simple demos initially! You will not need the advanced demos until you get to part 2.

5. **Once you have had part 1 marked, look at the part 2 requirements and decide on how you will develop a program to be able to do all of these, considering what you learned from part 1.** You should think at the start rather than trying to consider this later because some programs will make it more complex than others. Note: in some cases you may find it impractical to integrate some requirements into your main program. Don't forget that you could use a fancy 'intro' or 'ending' screen (state) to demo things that may not fit into your main program (e.g. TileManager in past years has not always fitted for some students). Think outside the box and remember that your aim is to get the best mark, not be exact in matching some existing program idea.
   Consider as a part of this what the advanced demos actually do. Most of them are really small but they demonstrate specific features and how to do specific things which you should not need for part 1.

6. **Read the Frequently Asked Questions document**. This has developed over the years to address questions which I often get asked. It will also give you clues to things you may not know of may not think of.

General requirements (apply to both parts):

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 submit 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 1, but part 2 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 must 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 is encouraged, as you don't need to change the existing files to do that.) I tried to make the classes flexible enough to allow behaviour to be modifiable by subclasses relatively easily.
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 files.** See comments above for point 4.
   - **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 1**, worth 1% of the module mark each, for a total of 10% of the module mark for this part of the coursework.
   When you have completed the coursework part 1, demo your work to a lab helper in a lab (ask for part 1 marking) and have it marked. (The marker will upload your mark to our system and tell you which marks you got and didn't get.) Also submit a zip of your source code to moodle:
   - **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 1 coursework submission on Moodle.
   - **Your submission should include all of the supplied files** (including demo files) as well as your new files.
   - **Also include your completed documentation sheet(s) in your submission – as a separate file outside of the zip.**

It should be possible when you have done this for someone to download the zip file, unzip it, do a 'rebuild all' to rebuild it, and run it, with no other changes.
It should also be possible for us to check your documentation file to see whether the mark matched what you expected or where any issues were.
Note: if you worked on Mac or Linux please include your make files etc that you used, ideally in a format which we can use to build and test the code ourselves.

7. **There are 30 marks available for part 2**, worth 1% of the module mark each, for a total of 30% of the module mark for that part of the coursework. These are usually a lot harder than the part 1 requirements. Start with the easy ones! Limit your time on part 2 to a maximum of 30 hours!
Read the requirements carefully to see which are optional and which are compulsory.
When you have completed part 4 upload the project file to the CW2 submission on moodle, as you did for CW1 (i.e. clean it, delete the .vs directory and zip the folder). A demo will be arranged where everyone will give a short live demo showing their program running and answering questions.
You will need to upload a documentation file before the demo explaining which features you achieved.
For some marks you will also need to provide a justification/explanation in the documentation file, and potentially even submit a video of the running program, so that moderation can determine whether this really is one of the best courseworks in the year (in terms of complexity of task or of impact).

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 part 1 (CW1) and part 2 (CW2):

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.)
- Your program crashes on exit or has a clear memory leak. (Lose 10% of your mark.)

The mark sheet that you take to the marking for parts 1 and 2 has entries for the above and your marker will annotate it according to their experience of your demo of your software.

Coursework Part 1 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 (both the bold and unbolded text) to determine whether a requirement was met or not. Please read the criteria carefully to ensure that you don't miss anything. If I need to clarify any of these criteria I will do so 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. **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.
4. **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.)
5. **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.

6. **Ensure that both keyboard and mouse input are handled in some way and do something.** *At least* one should influence one or more moving objects. 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 4). Or if your object is keyboard controlled, you need to handle mouse movement or button pressing. Both mouse and keyboard could influence the moving object if you prefer, or one could change something else.
7. **Provide an automated moving object which is a sub-class of DisplayableObject and different from the one in requirement 5.** 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, without user input (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 also have a different appearance to the object in requirement 5, and look different to the demos/samples as well.
8. **Include collision detection for at least 2 moving objects, so that they interact with each other.** This means that at least two of your objects should react to each other when they collide. Hint: look at UtilCollisionDetection.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). Rectangle-rectangle or circle-circle interaction is fine for meeting this requirement.
9. **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.
10. **Have at least one moving object interact correctly with the tile manager, changing a tile:** have one object which changes specific tiles when it passes over them. Consider the bouncing ball demo if you can't work out how to do this from the information in demos A and B, but you need to have at least some difference in behaviour from that. Assessment will check that you correctly detected the specific tile that the moving object was over and handled it appropriately.

Coursework part 2 functional requirements

The following requirements apply, and have a variety of marks available:

1. **Add states to your program (max 3 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 **NOT** 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, which differ in some way in both the appearance and behaviour.
   2 marks: As for one mark, but provide at least five states including at least one win or lose state as well as those mentioned above. (Note: if you are not doing a game, then have a state which allows a reset, e.g. 'load new document' in a text editor.)
   There must be significant differences between the states in behaviour and/or appearance.
   The program must be able to correctly go back from the win/lost (or reset) state to the starting state to start a new game/document, correctly initialising any data.
   3 marks (advanced): As for the 2 marks but you must implement the state model (design pattern) 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 your implementation is wrong (and there will be other issues since you may have more than one window as well).

2. **Save and load some non-trivial data (max 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 AND loading of at least one value to/from 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 data 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): 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 a game – saving **all** relevant variables for **all** objects) and it should be possible to reload this state later, with everything acting correctly and 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 and debugging to make it work properly.

3. **Use appropriate sub-classing with automated objects (max 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 class which add more behaviour.

   Example added to clarify this after a student question:
   e.g. a complex example can be found in ExampleDragableObjects.h:
   DragableObject and DragableImageObject are both DragableBaseClassObjects, which is a DisplayableObject, so DragableBaseClassObjects is an intermediate class which adds some behaviour.

4. **Creating new displayable objects during operation (max 1 mark):** meeting this requirement means that you can dynamically add one or more displayable objects to the program 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.

   You must not just re-create the object array contents to do this, although you could add to it. This requirement 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, adding new objects. 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.

5. **Correctly destroy displayable objects during operation (max 1 mark):** meeting this requirement means that during the running of one state you correctly remove a displayable object from the objects array and destroy/delete the object. You could do requirements 4 and 5 together, with a system which creates objects as they are needed, adds them to the array, then removes them later and deletes the object correctly. Requirement 4 has been deliberately written to be more flexible though, so it is easier to do (e.g. by showing and hiding objects). Warning: If you change the object array you need to tell anything using it that you did so, hence why this is in as a separate requirement – it can be tricky to get right. Please see the `drawableObjectsChanged()` method and investigate how it works to get this right.

6. **Complex intelligence on an automated moving object (max 4 marks):** As a minimum this criterion would involve something more than moving randomly or homing in on a player.

   1 mark: 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 rather than a higher mark. Note that this has to be more than a single constant behaviour – e.g. the moving randomly or homing in behaviour, so that the behaviour changes somehow according to the situation.
If your logic can be formulated as "if <condition> then do behaviour type 1, else do behaviour type 2" where behaviour types 1 and 2 are things which themselves involve a decision (e.g. go towards player or move randomly) then it fits this criterion.

2 marks: this means a good implementation of the intelligence of a moving object.
There should be some level of calculation or multiple-decision-making element involved. This needs to be more complex than the one mark criteria, and could involve some more complex calculations or a series of decisions.
E.g. calculating how to cut off a player, or intercept them, assuming that they maintain the same speed (predict where they will be and plan a path to get there), or showing some apparent intelligence which is not obvious to the markers how to implement, and not just random.
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.

3 marks (advanced): this means a good implementation of some complex algorithm for 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 tracking a player's decisions and predicting a player's path (in some non-trivial way) 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.
**** For 3 marks you need to include in the documentation a clear description of what you did and how you did it, including any screen shots or diagrams.***

4 marks (advanced): this means an **exceptional** implementation where the marker **is impressed with the complexity of the problem** you are solving, and the elegance of the C++ code that you are using to implement it. (Note: complexity of problem, rather than an overly-complex algorithm.)
This is not easy to get and is designed to allow the most capable students to excel. You are recommended to consider whether there is some link between some of the other C++ features (e.g. templates, operator overloading, smart pointers) if you go for this mark, and please don't be upset if you don't get it.
One key criterion to consider for the 4 mark criteria is: "does this show you to be one of the best C++ programmers in the year?"
**** For 4 marks you need to include in the documentation a clear description of what you did and how you did it, including any screen shots or diagrams, and also upload a video of up to three minutes (preferably shorter) demonstrating it working. It should be possible to work out from the video, documentation, and marker's comments whether you deserve 3 marks, 4 marks or neither.***

7. **Non-trivial pixel-perfect collision detection (max 2 marks):** 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. Using the supplied collision detection class is not sufficient to get either mark. Collision detection for rectangles and/or ovals is not sufficient.
1 mark: improved collision detection which will work on more complex shapes than the supplied collision detection methods, and which works well in your program.
2 marks (advanced): pixel-perfect collision detection on a complex irregular shape. Note that this is hard to get, so if your implementation is not solving a really complex task and/or was trivial to code then you probably will not get this mark. Your method should work with concavities in object

outline and your demo should show this. Examples would be automatic triangularisation of shapes and collision detection on these, or pixel-perfect checking (if a coloured pixel in one image is in the same place as a coloured pixel in another image). In particular, for 2 marks, your approach should work even if the shapes of the objects colliding are modified (potentially with some minor changes to account for shape types or colours).

8. **Have advanced animation for background and moving objects (max 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 advanced demos which show you how you could do some elements of this, but you should not just switch between two images (as one of the demos does), it should look smooth (i.e. extending from 2 images to enough to be smooth, or looking at how to make it smoothly animate in another way). 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.
   This cannot just be a copy-paste of the demo code – even for one mark you need to show enough awareness of what you are doing to justify that you 'showed understanding'.
   1 mark: showed understanding of how to animate background so it changes over time AND how to animate moving objects, but maybe just as a proof of concept rather than being smooth. Or smoothly animates one of background and some objects but not both.
   2 marks (advanced): animation of both objects and background is *smooth* **and** *visually impressive*. Please do note the 'impressive'. So please don't just rotate objects using facilities from a demo as that would not get you the marks.

9. **Interesting and impressive tile manager usage (max 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.
   Note: If you use multiple images then you could also potentially meet the animated appearance of automated objects criterion, and/or the animated background if you do it appropriately.
   1 mark: You meet the wording above in all requirements.
   2 marks (advanced): The marker is impressed by what you have done, it looks great, works well and has some interesting behaviour. Also, you particularly need a situation where moving onto some tiles would have effects on tiles elsewhere, e.g. standing on a key tile visibly unlocks a door tile. This requires *visibly* changing one tile from a position where the displayable object which does so is NOT over that tile (i.e. you update a different part of the screen.)

10. **Allow user to enter text which appears on the graphical display (max 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.

    1 mark: Do the above, including meeting the following key requirements:

    - 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

11. **Image rotation/manipulation using the ImagePixelMapping object (max 1 mark):** for this mark show that you understand how to create a new ImagePixelMapping class which acts differently to the existing ones and does something at least slightly differently to the existing examples, and that you use it appropriately.
    1 mark: show your understanding by creating and using your own ImagePixelMapping class and object to draw an image. Your class must do something different enough to the provided ones to allow the marker to see that you understand how to use it fully.

12. **Show your understanding of templates (max 1 mark).** To get this mark you should create your own template functions or template classes which are non-trivial and you should use them in some appropriate way.
    1 mark: created and used a template class or template function, showing your understanding of both how to use than and what they are used for.

13. **Show your understanding of operator overloading (max 1 mark).** To get this mark you should implement operator overloading in one of your classes for some operator other than = and == and use it appropriately, in a relevant manner. Hint: You may want to consider whether you can do this as a part of your loading/saving, since you can use global functions to overload << and >>.
    1 mark: created and used overloaded operators (other than = and ==) and used them appropriately, showing that you understand what operator overloading is for and how to use it.

14. **Use your own or standard smart pointers appropriately (max 1 mark).** This means using smart pointers appropriately in a way which illustrates your understanding of how to use these (i.e. using any that I already use will not show this). This means not just using SimpleImage, but creating your own classes and/or objects and using them appropriately somehow.
    1 mark: Example which illustrates that you understand how to use smart pointers in an appropriate manner which is actually useful.

15. **Correctly implement scrolling, allowing the user to scroll around using keys and/or mouse (max 1 mark).** Hint: look at the FilterPoints class to see how this can be done relatively easily. Also consider the advanced demos where this is done in some cases. The aim is for you to work it out for yourself or to work out from looking at the code which uses FilterPoints, so we will NOT explain to you how the code works.
    1 mark: you wrote C++ code which works to allow a user to scroll the screen using keys or mouse. You must be able to move the apparent view on the screen up, down, left AND right, rather than only in one direction. Something which scrolled the background appropriately (as in one of the demos) and moved all objects appropriately would meet this criterion (unlike for Criterion 16), as would something which integrated with the FilterPoints system.

16. **Correctly implement zooming, allowing the user to zoom in and out using keyboard and/or mouse (max 1 mark).** Utilise your own sub-class of the FilterPoints class to implement zooming. This is an example of how to use one class (a FilterPoints subclass) to modify the behaviour of another (the BaseEngine subclass), so demonstrates an important area of understanding. Consider

the advanced demos where this is done in some cases. The aim is for you to work it out for yourself, by examining example code, how to do this, so we will NOT explain to you how the code works.

1 mark: you wrote a FilterPoints subclass which works to allow a user to zoom in and out using keys or mouse. Your code should show your understanding of how to integrate your zooming with the framework provided – using the FilterPoints class that you create.

17. **Impact/impression/WOW factor! (max 3 marks)** These marks are awarded based upon the overall impact/impression of the program and whether people would probably pay money to own it. 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!

    0 marks: your program works well but really just meets the requirements rather than having a great impact or impression. These are meant to be advanced marks, so don't be disappointed with this.

    1 mark (advanced): This is non-trivial to get. You made an effort to ensure that it is beyond the minimum to just tick boxes. As long as you completed part 1 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,
    - the background includes 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,
    - moving objects are not just plain circles or squares.
    - have at least three moving objects
    - at least one is user controlled
    - you accept both mouse and keyboard input in some way
    - you use both images and drawing primitives appropriately
    - you have appropriate states (at least 2)
    - have something beyond the minimum required for the other marks (*please say what*)
    - your program should work smoothly
    - everything looks good with no problems

    You need to include one or more screen shots in your documentation, along with an explanation of what you did that was beyond the minimum in order to get this mark.

    2 marks (advanced): Meets all criteria for 1 mark and it is also *very impressive for the markers*. It needs to work very well, doing some complex tasks and you obviously made a significant effort on this coursework. If it's a game then it's fun and interesting to play, and has at least 2 different levels (to demonstrate your ability to do this). As a non-game (e.g. a drawing package or word processor?) it should be a complex program with different states (e.g. different page view layouts) performing a

useful task and it should achieve its purpose well.
It should be impressing the markers to get this mark.
This mark is subject to moderation across markers, to avoid subjectivity.
You need to provide information in your documentation about why this is so impressive, and include some screen shots, making an argument for getting this mark.
Note: If you think that you MAY have a sellable quality game which is one of the best in the year, then please also submit the video mentioned in the 3 mark criteria so that this can be assessed.

3 marks - sellable quality: This is supposed to be hard to get and basically means that your 'impact' of your program was even more than the 2 mark value of the impact/impression mark, which will not be easy. This basically means that both the marker and Jason went 'wow' when they saw this and thought that people would easily pay money to buy this program.
This mark is there so that we can differentiate the courseworks of the top students. In the same way that one would not always expect to be able to get 100% in an exam, not all students would be able to achieve this mark in a reasonable time, so should not be trying to do so.
Note: this mark is to assess the C++ ability, not the length of time in level design or data entry. As long as we can see where this is going, you should not spend a long time designing lots of extra levels or entering a lot of data for use in the program. You shouldn't need to be designing more than three levels as a proof of concept, if the levels you design illustrate your program well.
Your program should be working really well, smoothly, look good and make a good demonstration and should be comparable in quality to the sort of programs available on an app store.
I also note that 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 but still sellable if presented well.
This mark is subject to moderation.
To get this mark you need to provide the same level of documentation as for the 2 marks, but also a short demo video (maximum 3 minutes, but shorter is better) of it running and the sort of sales pitch that one would see in an app store, to illustrate why it is so great and why people should buy it.

18. **Additional complexity, pre-agreed in advance with Jason (max 1 mark, advanced).** Talking to students it seems like a number of students have ideas that are complex or interesting but don't fit into these mark schemes. This mark is available for covering features where you show a good knowledge of C++, use some complexity, or create something that is exceptionally impressive. The mark aims to reward only the students who are the very best at C++.
Talk to Jason in advance if you think that you are doing something extremely impressive in some way which is not covered by the marking scheme as written.
Importantly this mark has to show exceptional ability in C++, and must cover some feature which is not covered by another criterion.
It would be possible to get this mark for a topic covered by one of the other marking criteria if your implementation was truly impressive, well beyond the scope of the existing mark scheme (like an 'extra advanced' mark for any of the other criteria – agreed in advance with Jason), but it is also achievable for something which demonstrates a useful extension to the framework which shows your significant understanding of C++.
You need to get this mark agreed in advance, specifying clearly what you plan to do, why it is beyond the current criteria and how it demonstrates your considerable skill with C++!

Edit for v1.1: It was pointed out to me that there are only 29 marks available rather than 30, because I labelled saving/loading as out of 4 instead of 3. I have added requirement 18 to give you flexibility into how to get the final mark, by doing something impressive that you agree with me in advance, rather than specifying a specific thing. Hopefully this benefits you are rather than restricting it to saving/loading.

Edit for v1.2: Corrected class name from CoordinateMapping to ImagePixelMapping in requirement 11, since I changed the class name in the code.
I also labelled a few other marks with 'advanced' because it is probably good to point out to students which marks I expect to be the more complex ones. This does not change the requirements, but may help students trying to pick up the maximum marks with least effort.