Formal Coursework, G52CPP, 2013: Create an animated, interactive program, which uses files and has some ‘intelligence’ somewhere

Overview

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

Please ensure that you read all of the sections of this document. This document will explain to you how to submit your coursework, what class and function libraries you may use and warn you about plagiarism. All of these are important to know.

Documentation: You are advised to consider the documentation file (`cppcw.docx`) which you will need to fill in (and submit) right from the beginning of your work on this coursework. It will be much easier to fill this in if you think about it as you work.

Allow yourself plenty of time to do this coursework. Although the requirements are fairly open and straight-forward, your will be marked on how good your program is, how well it performs and so on, rather than upon just whether you meet the requirement.

The specific requirements for this coursework can be found in the following section and are deliberately open so that you can decide how to implement them. You may decide to do an extremely basic implementation, or you may decide to go for a complex implementation. Given the same level of code reliability and quality, a more complex implementation will result in a higher mark.

Important: Please read the Assessment section for details of marking. You cannot assume that you will get a good mark for this coursework by providing a simple implementation of each of the specific requirements. The requirements are there to guide you into what functionality to provide to meet the marking criteria. In particular, please note that marking is on the basis of: Functionality, Complexity, Compilation, Reliability, Clarity and Code Structure. If you want a good mark then you need to show in your code that you do well on each of these, and you should explain to us in your readme/documentation that you have done so.

Submission Date

You must submit your coursework by 17:00 (5pm) on Wednesday 24th April 2012 (in the first week back after the holidays). You are strongly recommended to submit before the holidays, so that you can enjoy a break before exam revision. If you follow the timescale on the labs page then you should have plenty of time to complete this before the holidays.

Plagiarism and copying

Your program must be all your own work, except for the code provided in the supplied framework files. You can, obviously, include in your program the framework code which you have been provided with, and can freely re-use code from any sample which I have provided to illustrate how to do things.

In particular, there must be no sharing of code and you must not get anybody to write the code for you or with you. On the rare occasions in the past where this has been found, it has been treated seriously and it can, ultimately, become a University disciplinary issue. I have deliberately made the coursework extremely open again this year, so that it will be obvious if people have worked together. If your program looks or works similarly to somebody else’s then please keep this in mind.

You must NOT write the code with someone else, give your code to somebody else or take somebody else’s code and use it. Similarly, you must not use source code given to you by somebody else. You MAY sit together to work out how to do something but when it comes to actually writing the code you should do this separately.

1 General requirements

The following general requirements apply to this coursework.

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. You should use the files MyProjectMain.h and MyProjectMain.cpp as the start of your program, and may modify them as much as you wish. You should create any other files which you need and include them into the project.
- Unless absolutely necessary, you should not change the base classes. It is possible that in some cases that I have not thought of that you will need to alter these files, although I believe at the moment that anything you wish to change could be changed in a sub-class. If you do need to alter these files then you should explain in the documentation file which files you altered, in what way, and why. Note: If you were using a class library, you would normally not alter that library to use it. I am trying to simulate that for this coursework.
- You MUST submit a documentation file as a part of this coursework. **Please ensure that you read the documentation requirements section BEFORE you implement the program requirements!** The documentation will be much easier to write if you at least make notes as you go along. This document is important because it is basically your opportunity to tell us what you think you deserve marks for. Please think of it that way and use it to point out where you think you did well.
- 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. If you are not sure what you can and cannot use, then you can find a list of the components of the Standard C++ here: `http://www.cplusplus.com/reference/`.

  You should not use any functions or classes which are not in the standard C++ libraries, other than SDL functions. i.e. you should not need to add any additional libraries other than those in the supplied project files.

  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.
- Your program must not make any direct operating system calls (via `system()` or otherwise) since this would make it platform dependent. i.e. any operations that you perform must be performed using C/C++ code, not operating system commands or shell script.
- All of the source code must be your own work. Please see the section on plagiarism.
- The structure of your source code WILL contribute to your mark. Please consider the comments on code structure that you have heard in lectures. e.g. you should ensure that your source code is easy to read (using comments where appropriate), is well decomposed into functions and/or classes and uses appropriate data structures and algorithms.

2 Specific program requirements

Before panicking about how you will do this coursework, please note that I will show you examples of how to do most of these in the Tuesday lectures. You then just have to do similar things yourself and fit everything together to make a full program. You should already know how to do the file load/save from the informal coursework, as well as having the experience with pointers that you will probably need. Please take the opportunity to demo your program in the labs and get useful feedback.

The finished program must compile (under **Visual Studio 2012**) and run (as a windows executable) on the Computer Science PCs in the A32 lab. **You should test your own zip file, using the assessment process discussed later, to ensure that this process will correctly build your program.**

You should implement a program, based upon the supplied framework classes, which supports the following functionality. This will involve sub-classing various base classes (when needed) and implementing various virtual and non-virtual methods.

- Draw an appropriate background.

  You need to ensure that you draw an appropriate background for the program. In particular, you will probably wish to alter the background according to the current state of the program (see the requirement to support different states). You will also need to change the background due to player interaction (see below).

- Have moving objects.

  You should have one or more objects which move or change, refreshing only part of the screen.

- Have interaction between the moving objects and the environment.

  For example, as an object is moving have it interact with the background to change the background as it passes it, to bounce off the background (potentially changing it) or to affect the behaviour of the object as it passes (e.g. slow it down, change its direction or speed it up).

- Provide player interaction.

  Provide a facility for the player to interact with the program. For example, allowing the player to move a player controlled object around the map.

- Provide AI-controlled objects.

  There should be some kind of intelligence or non-basic algorithm in the program, to demonstrate your ability in this area. e.g. some things which move on their own, or some enemy which chases the player.

- Save and load information to and from files.

  You must provide the facility to load and save information from a file. You don't need to load and save the same information, if you wish.

  Examples of loading information: If you are writing a game then you could load the level layout and the positions/speeds of any moving items, for either a puzzle or an action game. Alternatively, a program to replay some kind of movement could load positional information from a file and replay the movement. A drawing program could load existing objects to draw from a file.

  Examples of saving information: You could save some progress information. Or maybe provide a save and load facility to save the state of the program and reload it later.

  Alternatively, for a relatively simple implementation, you could include a high score loading/saving facility, or a configuration loading/saving facility.

- Display status information on the screen.

  You need to display some kind of status information on the screen. For a game this may be in the form of a score and/or number of lives left. For an editor it could be information such as the current mode, selected item, etc.

- Support different states.

  You should provide the facility for the program to be in a number of different states where the behaviour changes. For example, a game may have a startup state where the screen says 'press ENTER to continue', a state where a game is running, a state for winning a level and loading a new level or a state where a player loses a life.

3 Documentation requirements

You MUST provide a documentation document to tell us how to mark your coursework. There are a number of sections which you should provide. To help you, I have provided a template for the documentation file in the file `cppcw.docx`, so that you just need to fill in all of the gaps. The template document provides instructions about what you should put into each of the sections. You should submit a single documentation file. This should be named `cppcw.pdf`, `cppcw.doc` or `cppcw.docx` as appropriate to the file type.

Please note that copying someone else's documentation file is plagiarism.

The documentation file is compulsory. If you do not either fill it in or submit it, then please expect to get fewer marks, since your documentation should tell us what we should give you marks for. If you do not mention something in the documentation then you should not expect marks for it. **In summary: if you want marks for something then please make sure that you mention it in your documentation! Otherwise you risk us missing something.**

4 Evaluation Method

Evaluation of your submission will be performed as follows:

1. Login to one of the Computer Science PCs in the A32 lab.
2. Obtain your submitted project as a zip file.
3. Unzip your project to a new directory, maintaining the directory structure that is in the zip file.
4. Double-click on the .sln file to open the project in **Visual Studio 2012**.
5. Select 'Rebuild all' from the menu to rebuild the entire project.

   If your program will not compile, then you should expect to get very few (if any) marks. You cannot afford to be in that situation!
6. Look for any warnings.

   The presence of warnings are an indication of potential errors and will be investigated. Warnings will probably reduce your mark! The extent of the loss will depend upon what the warning is.

   Exception: The aim of this coursework is to test your C/C++ knowledge and your ability to work within existing framework code. Warnings that are due solely to **Visual Studio 2010** configuration options or similar will not lose you marks.
7. Read the documentation file to see what the program should do and how to use it. This is important since it will tell us what to try to give marks for.
8. Run the executable and observe the behaviour and features.
9. Each requirement will be evaluated and marked in turn. i.e. we will consider whether you implemented each requirement, and how well you implemented it and will use your documentation file to assess your program.
10. Your source code will be considered and a mark will be awarded for the clarity and structure. We are not looking for any specific coding style, but for how clear the code is. One measure of this will be how long it takes us to understand your code compared with how complex the features are which have been implemented.
11. Your documentation will be used to determine whether or not you implemented requirements and how you implemented them. You should ensure that your documentation is adequate to explain how each requirement was implemented so that you get all of the marks that your implementation deserves. If we miss something that you do not mention in your documentation then you can expect to not get the marks for it.
12. We will assess the robustness of your program. Runtime errors and crashes will lose you marks. The extent of the loss will be related to both the cause of and effects of the error. If you mention any problems in your documentation then the penalty will be lower, since you obviously tested it better to know about the problem.
13. We will consider the difficulty of creating the program which you have created. Actually meeting the requirements is relatively simple, and they could be met by either a very simple program or a very complex program. You should expect to get more marks for the more complex and feature-filled programs than for simple implementations.

5 Marking Criteria

Your coursework will be assessed using the following criteria:

- Functionality. You will get marks for implementing the different requirements, and how well you implement them. You should ensure that your documentation helps us to give you marks for this by making it easier to see how wonderful your implementation of each requirement is.
- Complexity. Since you have a lot of freedom to choose to do a simple or a complex program, this criterion measures how complex the task was that you set yourself. More complex programs (in terms of the number and complexity of the features provided, not the messiness of the code) will gain better marks.
- Compilation. You will lose marks for warnings when we compile, unless these warnings are for Microsoft-specific things, as mentioned previously.
- Reliability. You will lose marks if your program crashes under testing. The amount will depend upon how badly it crashes and how frequently.
- Clarity: How simple is the source code to understand? Is it commented correctly? You will get a better mark if your code is easy to understand than if we find it very difficult to understand.
- Code structure. Keep your code structure tidy and logical. We will consider the appropriateness of your use of data structures (e.g. arrays, linked lists, structs and classes), and algorithm structures. We will also consider whether you decomposed your code into appropriate classes and functions.
- Efficiency. You will be awarded fewer marks if your code is unnecessarily inefficient. e.g. loops inside loops, inside loops, etc, when there was an easier way.
- Appearance. We will consider how good it looks, and how technically difficult it was to achieve that appearance - so that making something look really good will give a reward when it is appropriate.
- Documentation. Please be prepared to lose marks if you do not submit a documentation file.

When marking, we will consider the following grades for each of the above categories. These are similar to those for the group or individual projects but we understand that this is a MUCH smaller piece of work and that we are assessing only your ability with C/C++, not your ability to write a dissertation:

- Exceptional (90-100%)

  Better than both Outstanding and Excellent.

  Highest quality program, bug-free, efficient, and user friendly.

  Perfectly structured source code that is very easy to understand. Sub-classing and virtual functions would be used where appropriate and data access would be restricted appropriately in all cases (i.e. public vs protected vs private).
- Outstanding (80-89%)

  Not quite as good as exceptional, but better than excellent.

  There may be a minor problem with some element, but the program is user-friendly and virtually bug free and the source code would be almost perfect.

  Note: This is quite hard and time consuming to achieve. I do not suggest that you should be spending the time necessary to get this kind of mark on a single coursework, even if it is worth 40% of the module. Examples of courseworks that fit into this category are in the hall of fame from last year's G52CFJ module.
- Excellent (70-79%)

  The work should display a complete and thorough understanding of the elements of C++ which are used.

  The source code should be easy to understand and excellently structured.

  All requirements will have been met to an overall excellent standard.

  The documentation should be excellent and tell us exactly how to test the program to demonstrate its features (in bullet-points).

  It should be obvious that the program was well tested and it should be very hard to make it go wrong.

- Good (60-69%)

  The program should show a good understanding of C++ and should be a reasonably complex task.

  Code structure should be good and the code should be easy to understand, appropriately commented and clear.

  The quality of the implementation of the requirements should be good although there may be very minor flaws.

  The program should work correctly the vast majority of the time, and be usable, although there may be some minor flaws.

- Average (50-59%)

  The program should show a reasonable understanding of the key components of C++ and be a reasonably complex application.

  There may be weaknesses in some points, but there should be some evidence of awareness of these (e.g. mentioning them in the documentation) and the program should be reasonably reliable.

  Documentation would be readable and reasonably complete, allowing testing of the program.

- Adequate (40-49%)

  The program and documentation should show an adequate understanding of C++ but there may be a number of problems. Source code may not be as clear as we would like, and not as well structured.

  The program may occasionally crash but should be reasonably reliable.

- Poor (below 40%)

  The work would display a poor understanding of C++ and have one or more problems.

  Code structure would be very weak and potentially hard to understand.

  The program may be very simple, perhaps just a little more than the demonstration programs.

  Documentation may be very badly written or missing.

6 Submission

Your submission should consist of a documentation file and a zip of your project directory, including both of the project files which contain the settings required to compile your program (the `.sln` and `.vcproj`) and the source code files. You should clean your solution prior to zipping it up. (This will delete any temporary files or compiled files, and can be achieved by using the 'Clean ¡ProjectName¿' option from the 'Build' menu.)

The zip file containing your files should be named `"cppcw.zip"`.

The single document file should be named `cppcw.doc` or `cppcw.pdf` according to its type.

You should submit your two files using the coursework submission activity on the Computer Science Moodle page: http://teaching.cs.nott.ac.uk/cw/.

I have put a test submission on the same moodle page as well so that you can try out the submission process safely using that to see how it works. Please ensure that you use the coursework one to submit your final coursework, not the test one.

IMPORTANT: please test your zip file prior to submission. (i.e. download it from the moodle site you submitted it to (before sending in the submission), unzip it, try a build and try to run the built exe.) Every year a number of people submit old versions of their project instead of the latest versions or even submit corrupt zip files. This just means throwing marks away.

Although you should find that the source code will compile and run under both `OS X` and `Linux`, providing you install the correct packages/libraries, for those who prefer these environments to the Windows platform provided in the PC labs, your final code MUST compile and execute correctly on the PCs in A32, since that is where it will be evaluated.

Note: to build and run under a different operating system, you will probably need to get the correct version of SDL for that platform and install the various libraries. You could try here: `http://www.libsdl.org/` and/or search for the dll names to seek the development versions of each of the libraries for your operating system. I am afraid that I currently have no access to either a desktop linux PC or a Mac to test or execute the code on.

You are strongly advised to test your submission on the PCs in A32 to ensure that it compiles and executes correctly even if you actually develop your program on a different computer or platform.

7 Where to start

- Start early, to give yourself plenty of time.
- Attend the lectures! I will go through a lot of the coursework files and give examples in the later demo lectures each week.
- Ensure that you understand how functionality is supported in the samples. i.e. in general you just need to adapt the things that you learn from the samples, lab exercises or lectures and fit them into your chosen program.
- Attend the laboratory sessions. The demonstrators are there to help. Although they will not (and should not) answer questions directly about how to produce your coursework in terms of writing code for you, they can help you to find problems and to understand how to do things. Start attending as soon as possible, otherwise there may be insufficient time to have all of your questions answered. Try to present generic questions, or questions about the way in which the samples work and use their knowledge.
- Read the entire of this requirements document through at least twice, rather than just reading one requirement at a time. Be sure that you think about all of the requirements and how they will fit into your program.
- Plan out your program in advance. Consider each requirement and decide what you intend to do to meet the requirement.
- Identify which class(es) are associated with the functionality for each of the requirements, and in what way. If possible, identify any existing functions which are related to the functionality.

  As long as you understand the following functions, you have a good start on knowing where to make your changes:

  - Drawing a moving object is the responsibility of the `Draw()` method of a `DisplayableObject` sub-class.
  - Moving/controlling a moving object is the responsibility of the `DoUpdate()` method of a `Displayable-Object` sub-class.
  - The `KeyDown()` and `KeyUp()` methods of the `BaseEngine` subclass are used for handling key presses and releases, although there is also an `IsKeyPressed()` method which can be used to detect whether a key is currently pressed down.
  - The `MouseMoved()`, `MouseDown()` and `MouseUp()` methods of the `BaseEngine` subclass are used for handling mouse button presses and mouse movement.
  - The `SetupBackgroundBuffer()` method of the `BaseEngine` subclass is used for drawing the background for the window. e.g. the tiles.
  - The `InitialiseObjects()` method of the `BaseEngine` subclass is used for creating the moving objects (i.e. player and opponents).
  - The `GameInit()` method of the `BaseEngine` subclass is the function which calls the `SetupBack-groundBuffer()` and `InitialiseObjects()` methods, so is the place to alter the behaviour if you need to do so.
  - The `GameAction()` method of the `BaseEngine` subclass is the place where you will do most of the work of controlling the program. This is the function which is constantly executed to do things, and is responsible for telling the `DisplayableObject` sub-class objects to update themselves, by calling the `UpdateAllObjects()` objects.
  - The `GameRender()` method is the main display method. This will call the `DrawScreen()` or the `DrawChanges()` methods, depending upon whether the whole window needs to be refreshed or just the moving objects.
- Make regular backups. One of the major problems with a larger program is that it is easy to break something and spend a long time trying to get back to a working version. At least every time you achieve a new piece of functionality you should take a complete backup of your source files. You will then have something to compare with later, and potentially something to revert to if really bad things happen.
- Become familiar with the **Visual Studio** integrated debugger. The project has been set up with both debug and release versions. The debug version will run slightly slower but contains the information necessary to find errors.

8 Examples which would meet these requirements

The most straight-forward approach to developing something which meets the above requirements is to write a simple action game. However, I did not want to force you to have to do so, so you could also meet them by developing other programs. Your mark will depend upon your ability with C/C++, not upon whether you choose to develop a game, editor, replay tool, etc. However, it will be influenced by the complexity of the task you undertake.

8.1 Action games

There are many arcade games which would meet these requirements. Tile-based games are especially appropriate, as are platform games or puzzles. e.g. Pacman, Boulderdash (`http://www.onlinegames.com/boulderdash/`). You can find many examples at these places: `http://www.onlinegames.com/`, `http://www.lalena.com/Games/`, `http://www.jgames.com/`, or at many other places you can find using a search engine, especially those which run within a browser, since they tend to be simpler.

For something much simpler, you could consider `Pong` (`http://en.wikipedia.org/wiki/Pong`) or similar games. But please be aware that simpler programs are worth fewer marks.

8.2 Other options

Maybe you could consider an animated interactive demonstation program, with some kind of saving and loading of the layout information or of what to display. Perhaps a program to allow data to be loaded and displayed as a chart/graph, and edited by the user, as long as there is some kind non-basic algorithm involved. Or a program to draw a layout specified in a file and to allow a user to interact with the program by selecting and moving items around the screen, creating new items, saving and loading the items and so on. You would need some kind of animation in order to ensure that it met all of the requirements and some kind of algorithm to do something non-basic, such as moving other objects out of the way as you drag one object around? (This is probably not simple.)

9 Features from Pacman and 3d-cube game from previous years

In previous years I set the task to create a pacman and 3d-cube game. The requirements for these could be summarised as:

1. Support and draw different types of tile.
2. Have a tile which must be collected/changed to finish a level.
3. Have a player controlled object which can move around collecting the tiles.
4. Provide enemy objects which kill the player. Give them some level of intelligence to make it difficult for the player.
5. Provide tiles which give extra abilities to the player, e.g. bonus score, extra score for a while, immunity from dying for a short time, ability to kill the enemies rather than vice versa. I also suggested that pacman should slow down when eating, which made for an interesting coding task.
6. Increase the level as appropriate when all collectables have been collected.
7. Maintain a score for the player and display it on the screen.
8. Store a high score in a file and update when a player loses.
9. Maintain a number of lives for the player.
10. Have level files in ASCII text files.

You can probably see that such a game would meet the requirements listed earlier:

- Draw an appropriate background.

  The pacman game draws the maze in the background, as a series of tiles. Each tile is either a wall, passage, pellet to eat or power-up pellet. When pellets are eaten, the tile is changed from a pellet tile to an empty passage tile, and tile manager is told to update the tile on the screen.

- Have moving objects.

  The pacman game has one moving pacman and a number of enemies to chase the pacman.

- Have interaction between the moving objects and the environment.

  The changing of the tiles when the pacman eats the pellets is an example of this.

- Provide player interaction.

  The player can control pacman, moving him/her up/down/left and right. Whenever the current movement has ended, the system checks whether each key is pessed and sets off a movement in the relevant direction, to go to the centre of the next tile.

- Provide AI-controlled objects.

  The chasers (traditionally ghosts) are examples of these. The enemies move semi-randomly, with a small bias towards moving towards the pacman.

- Save and load information.

  Levels are stored in files, which specify the maze layout and the initial positions of the enemies.

  The high score is both loaded and saved.

- Display status information on the screen.

  The score, high score, number of lives and remaining power-up duration are all shown on the screen.

- Support different states.

  The pacman game has the following states:

  – Loaded: display first level, wait for player to press a key
  – Runing: playing the game
  – Player lost a life: wait for a key press
  – Player lost all of their lives: update high score, give them an appropriate message, wait for a key press
  – Player finished level: give appropriate message, load new level, display level and wait for player to press a key to start

  I used an enum for the state number. Each was implemented by putting a switch on the state value into the important functions. e.g. `SetupBackgroundBuffer()`, `GameAction()`, `KeyDown()` to change their behaviour according to what the state is. An even better way may be to use virtual functions to switch the behaviour based upon the current state sub-class.

10 Version history

Version 1.0: Initial version for 2013.